Algorithmes Parallèles Asynchrones : implémentation et simulation d'applications modélisées par des équations pseudo-linéaires.

Thierry Garcia

HDR

En vue de l'obtention de

L'HABILITATION A DIRIGER DES RECHERCHES DE L'UNIVERSITÉ DE TOULOUSE

Délivré par : *l'Institut National Polytechnique de Toulouse (INP Toulouse)*

Présentée et soutenue le *09/01/2024* **par :**

Thierry GARCIA

Algorithmes Parallèles Asynchrones : implémentation et simulation d'applications modélisées par des équations pseudo-linéaires.

JURY

Pr. André-Luc Beylot	INP-IRIT-ENSEEIHT	Président
Pr. Martin J. Gander	Université de Genève	Rapporteur
Pr. Christophe Guyeux	Université de Franche-Comté	Rapporteur
Pr. Frédéric Magoules	CentraleSupélec	Rapporteur
Pr. Raphaël Couturier	Université de Franche-Comté	Examinateur
Pr. Nahid Emad	Université de Versailles	Examinatrice
Pr. Jean-Marc Pierson	Université Paul Sabatier IRIT	Examinateur
Pr. Pierre Spitéri	INP-IRIT-ENSEEIHT	Examinateur, Correspondant

---

Unité de Recherche :

Institut de Recherche en informatique (UMR 5505)

Résumé

Le travail, présenté dans ce mémoire, concerne des travaux en algorithmique parallèle effectués depuis une quinzaine d'années. Plus particulièrement, il présente en détails l'implémentation d'algorithmes itératifs parallèles synchrones et asynchrones dans le cadre de calculs scientifiques effectuée depuis 2009. Ces algorithmes permettent la résolution de grands systèmes algébriques linéaires ou pseudo-linéaires, éventuellement contraints.

Sur le plan théorique, les itérations asynchrones sont associées à des itérations point fixe, effectuées en parallèle prenant en compte des techniques de décomposition en grand blocs ; dans le cas de la résolution numérique d'équations aux dérivées partielles cela correspond à des méthodes de décomposition de domaine. Les différents processeurs impliqués dans une résolution parallèle asynchrone n'attendent pas la réception complète des messages correspondant aux valeurs d'interface entre les sous-domaines et on évite ainsi les temps d'inactivité liés aux éventuelles synchronisations. L'intérêt de tel algorithme se révèle intéressant compte tenu de l'évolution des plateformes de calcul parallèle comme les grilles, les architectures pair-à-pair, le cloud et les serveurs pour calcul haute performance (HPC). Sur le plan expérimental, nous nous sommes intéressés à des plateformes de calcul parallèles ayant des nœuds géographiquement éloignés et hétérogènes.

Le champ d'application de ce type d'algorithme est important et plusieurs problèmes linéaires et non-linéaires ont été résolus de façon comparative en parallélisme synchrone et asynchrone dans le cadre de nos travaux. Plusieurs expérimentations ont été effectuées sur diverses plate-formes afin de montrer la validité des algorithmes parallèles asynchrones pour traiter diverses applications couvrants différents domaines, tels que les mathématiques financières, la mécanique, l'interaction fluide-structure, la solidification de l'acier, le problème de séparation de protéine par électrophorèse, ... Cette étude a pour but de présenter les moyens mis en œuvre pour mettre en place et réaliser les différentes simulations afin de résoudre ces différents problèmes en utilisant les primitives non bloquantes de base du standard MPI afin de mettre en œuvre des itérations asynchrones.

Mots-Clés

Itérations Asynchrones, Calcul Haute performance, Simulation Numérique, Équations aux Dérivées Partielles, Calcul Parallèle

Abstract

The work presented in this dissertation concerns parallel algorithmic work carried out over the last fifteen years. More specifically, it presents in detail the implementation of synchronous and asynchronous parallel iterative algorithms in the context of scientific calculations carried out since 2009. These algorithms can be used to solve large linear or pseudo-linear algebraic systems, which may be constrained.

Theoretically, asynchronous iterations are associated with fixed-point iterations, carried out in parallel using large-block decomposition techniques ; in the case of the numerical solution of partial differential equations, this corresponds to domain decomposition methods. The various processors involved in an asynchronous parallel solution do not have to wait for the messages corresponding to the interface values between the sub-domains to be received in full, thus avoiding the inactivity time associated with any synchronisations. Such an algorithm is of interest given the evolution of parallel computing platforms such as grids, peer-to-peer architectures, the cloud and high-performance computing (HPC) servers. From an experimental point of view, we were interested in parallel computing platforms with geographically distant and heterogeneous nodes.

The field of application of this type of algorithm is wide and several linear and non-linear problems have been solved comparatively in synchronous and asynchronous parallelism as part of our work. Several experiments have been carried out on various platforms in order to demonstrate the validity of asynchronous parallel algorithms for various applications covering different fields, such as financial mathematics, mechanics, fluid-structure interaction, steel solidification, the problem of protein separation by electrophoresis, etc. The aim of this study is to present the means used to set up and carry out the various simulations to solve these different problems using the basic non-blocking primitives of the MPI standard in order to implement asynchronous iterations.

Keywords

Asynchronous Iterations, High performance Computing, Numerical Simulation, Partial Derivative Equations, Parallel Computing

Remerciements

Ce mémoire prend en compte mes travaux de depuis plus de 20 ans effectués au sein du laboratoire IRIT-ENSEEIHT-INP de Toulouse. Je tiens à remercier les membres du jury pour avoir accepté d'évaluer mon travail dans le cadre de la préparation d'une habilitation à diriger des recherches.

Ainsi je tiens à remercier chaleureusement le Professeur Pierre Spitéri grâce à qui j'ai pu découvrir et progresser dans un domaine de recherche passionnant. Mes travaux n'aurait pas pu aboutir sans ses conseils avisés sur les aspects scientifiques.

Je remercie aussi le Professeur André-Luc Beylot, qui m'a toujours soutenu, qui m'a apporté de précieux conseils et qui a accepté d'être le Président du Jury de mon HDR.

Je remercie également les Professeurs Martin J. Gander, Christophe Guyeux, Frédéric Magoules qui ont accepté d'être rapporteur ainsi que les Professeurs Raphaël Couturier, Nahid Emad et Jean-Marc Pierson qui ont accepté d'être examinateur.

Je tiens aussi à remercier Ming Chau, une personne avec qui j'ai pu être en phase sur bien des points scientifiques, qui m'a apporté beaucoup par ses conseils et nos échanges sur notre passion du développement informatique.

Je remercie également Raphaël Couturier, avec qui j'ai eu l'occasion de collaborer de façon très productive et qui m'a conseillé sur l'orientation de ma carrière.

Je remercie aussi Clovis Tauber et Vincent Partimbene avec qui j'ai pu collaborer scientifiquement.

Je tiens enfin à remercier toutes les personnes de l'ENSEEIHT, l'IRIT, du LAAS-CNRS, de l'Université de Franche-Comté et de l'Université d'Annaba avec qui j'ai pu collaborer.

Enfin, je tiens à remercier mon épouse Nathalie qui m'a toujours soutenu ainsi que ma famille.

"On ne peut éviter que nos proches disparaissent mais on peut les garder à jamais dans notre cœur."

Table des matières

Liste des tableaux

Liste des figures

Introduction générale

L'informatique d'antan avec la construction de l'$ENIAC$ (Electronic Numerical Integrator and Computer) en 1946, composé de 19000 tubes à vide, pesant 30 tonnes, ayant une vitesse de calcul de $0,005$ MIPS[1], n'a plus rien à voir avec l'informatique actuelle où les processeurs (CPU) des ordinateurs atteignent une vitesse de calcul supérieure à 400 000 MIPS.

Les premiers ordinateurs étaient essentiellement constitués d'une mémoire chargée de stocker les programmes informatiques ainsi que les données nécessaires à ces derniers, une unité arithmétique et logique chargée d'effectuer des opérations spécifiques, et les unités d'échanges vers les périphériques (disques, imprimantes, etc.).

Sur ce type d'architecture, un code de calcul était exécuté en mode séquentiel sur cette seule ressource ; cela correspond à l'exécution des instructions l'une après l'autre, donc une seule à la fois, même si les opérations à effectuer sont indépendantes. Ce mode d'exécution a très rapidement atteint des limites de performances lorsque le calcul ne permet pas d'obtenir des résultats dans des délais raisonnables et nécessite un stockage mémoire important. Ainsi, pour des calculs rendus nécessaires pour traiter de grosses applications, comme par exemple en météorologie, ce type de programmation séquentielle n'est plus adapté et les performances de calcul ne sont pas satisfaisantes pour obtenir des résultats de simulation dans des temps relativement rapides.

L'intuition qu'un travail constitué de tâches quasi-indépendantes peut être réalisé en beaucoup moins de temps s'il est réparti efficacement entre plusieurs personnes ou de nombreuses machines a permis d'introduire le parallélisme. Ce dernier a été appliquée avec réussite dans plusieurs activités de la vie courante comme les récoltes, la distribution du courrier, ou encore les chaînes de montage en usine. L'augmentation du nombre de travailleurs permet de terminer la tâche plus rapidement. Une limite peut, bien sûr être atteinte, de sorte qu'augmenter encore le nombre de travailleurs, n'apporte plus de gain de temps. En fait, certaines tâches sont purement séquentielles et ne peuvent être exécutées que par une seule personne à la fois. Par exemple, deux marathoniens ne peuvent se partager la distance à parcourir et réclamer la médaille d'or ([2]).

1. millions d'instructions par seconde

Le calcul parallèle est donc apparu comme une solution permettant d'améliorer les performances des ordinateurs et consiste à exploiter au mieux les ressources de calcul et de mémorisation.

Le parallélisme combiné à une algorithmique performante permet donc de diminuer le temps de traitement afin de répondre au mieux à d'importants besoins. Il rompt avec l'approche classique qui consiste à gagner du temps en effectuant plus rapidement chaque opération grâce à l'augmentation de la vitesse des processeurs, approche limitée par les lois de la physique. Actuellement, l'emploi du parallélisme est d'abord justifié par des besoins de performance pour satisfaire d'une part, des contraintes de délai qui sont dues à la nature temps réel des problèmes tels que la prévision météorologique, l'interprétation de photographies par satellites de régions critiques, etc... et d'autre part la taille des problèmes. Citons également la nature intrinsèquement parallèle de certains problèmes qui gagneront en clarté à être programmés selon un modèle parallèle offrant les constructions adéquates. Le parallélisme a très largement multiplié les moyens : nombre de processeurs, agencement des machines, réseau de communication, mémoire partagée ou répartie, contrôle centralisé ou distribué, etc... et a donc grandement contribué à la multiplication de modélisation d'algorithmes parallèles.

La solution retenue revient à paralléliser les codes, c'est-à-dire à effectuer plusieurs calculs simultanément sur différentes ressources constituées pour chacune d'elle par une unité arithmétique et logique, appelée aussi processeur ; ainsi un plus grand nombre d'opérations sont effectuées en un temps minimal.

Pour paralléliser un code de calcul, l'utilisateur est amené à décomposer le problème global à traiter en plusieurs sous-problèmes couplés chacun de taille plus petite. Ainsi plusieurs tâches de calcul pourront être traitées simultanément sur ces processeurs, le couplage entre les sous-problèmes étant effectué par des échanges de résultats entre les processus coopérants en parallèle. Par ce biais, à condition de bien optimiser les codes de calcul parallélisés, les temps d'exécution pourront nettement diminuer ce qui permettra de résoudre des problèmes de plus grande taille et de traiter des volumes de données plus importants.

Ce nouveau mode de calcul nécessite d'une part de remettre en cause l'architecture des premiers ordinateurs, d'autre part d'adapter le programme de calcul à un mode d'exécution parallèle et enfin d'apprendre de nouvelles méthodes et outils de programmation.

Les architectures des superordinateurs ont donc évoluées d'abord par la construction de machines à mémoire commune où tous les processeurs ont accès à cette dernière. Cependant ce type de machine est d'une part onéreux à construire et d'autre part il ne permet pas un parallélisme massif souhaitable pour traiter de grosses applications. Les constructeurs ont donc conçu des machines à mémoire distribuée où chaque processeur possède sa propre mémoire locale, les liaisons entre ces proces-

seurs s'effectuant par un réseau d'interconnexion ; ce dernier permet des échanges de messages entre les processeurs ce qui réalise une mise en œuvre du parallélisme. Il existe également des architectures mixtes combinant les deux cas précédents.

Par ailleurs sur le plan de la programmation de ce type de machine, il convient de repenser le codage des algorithmes afin de prendre en compte les possibilités offertes d'effectuer simultanément plusieurs instructions en même temps. La solution idéale est de découper le problème à résoudre en tâches indépendantes. Ce mode de calcul est rarement possible et en général le programmeur considère un découpage du problème en tâches couplées entre elles ; dans ces conditions les processeurs échangent les résultats de leurs calculs avec les autres unités avec lesquelles ils doivent communiquer. On distingue essentiellement le mode de communication synchrone et le mode de communication asynchrone.

La simulation numérique est nécessaire dans de nombreux domaines scientifiques et industriels. Les moyens informatiques actuels d'un point de vue puissance de calcul et capacité de stockage permettent donc désormais de réaliser des simulations numériques à grande échelle. Par exemple, la simulation numérique peut intervenir en avionique, en génie chimique, en mathématiques financières, en mécanique des fluides, dans le nucléaire, en physique des plasmas, en météorologie, en océanographie ou encore en biologie moléculaire. Ces nombreuses applications nécessitent une démarche qui consiste à simuler par l'informatique des phénomènes modélisés par des modèles mathématiques en particulier des équations aux dérivées partielles. Simuler sur les équations modélisant ces différents domaines est source de difficulté en raison des temps de calcul très importants induits par le nombre d'inconnues à résoudre et des exigences sur la précision des résultats à obtenir. Compte tenu du caractères creux des matrices de discrétisation des EDP, on utilise des méthodes itératives.

Pour fixer les idées nous allons utiliser un mode de calcul itératif bien adapté à la résolution de problèmes de grande dimension ; dans ces conditions :

— en mode de communication synchrone chaque processeur peut commencer une nouvelle itération lorsqu'il a reçu les données de tous les processeurs avec lesquels il doit communiquer ce qui implique des périodes d'inactivité entre les itérations ; ce mode de communication est donc bloquant ;

— en mode de communication asynchrone, chaque processeur exécute ses propres itérations sans tenir compte de la progression des autres processeurs avec lesquels il doit communiquer ; ainsi les processeurs effectuent leurs calculs à leur propre rythme sans attendre les données émises par les autres processeurs en utilisant les dernières données disponibles délivrées par ces derniers. Dans ce mode de communication, il n'y a donc pas de période d'inactivité ; ce mode de communication est donc non bloquant.

L'utilisation des architectures parallèles actuelles et des algorithmes parallèles associés permet de palier le manque de ressources suffisantes à ces simulations. La résolution parallèle de systèmes algébriques fait intervenir des algorithmes néces-

sitant des échanges de données entre différentes machines à travers un réseau de communication. Ces échanges sont réalisés grâce à des primitives de communications point-à-point ou collectives et des synchronisations entre les machines sont nécessaires afin de conserver la cohérence des calculs effectués.

Malgré le potentiel des machines et des réseaux actuels, ces algorithmes parallèles synchronisés subissent des temps de latence induits par les accès synchrones aux données distantes. En effet, plusieurs situations peuvent induire l'attente de la disponibilité des données distantes pour poursuivre les calculs et peuvent provoquer une chute des performances d'un algorithme parallèle. Par exemple, cette dégradation peut résulter des éventuelles différences de vitesse de calcul des machines, de l'inactivité de certains processeurs durant ces attentes en mode synchrone, de la charge de chaque processeurs par exemple, lors de traitements numériques différents ou de taille disproportionnées, des délais lorsque les machines utilisées sont éloignées géographiquement ou de la surcharge du réseau d'interconnexion.

Afin de surseoir à ce type de problème, les algorithmes parallèles asynchrones ont été développés dès 1969 par D. Chazan et W. Miranker [39]. Ils permettent de tirer parti au maximum de la puissance de calcul en supprimant les temps d'inactivité dues à d'éventuelles attentes bloquantes. Ce type d'algorithme a été expérimenté dès 1967 par J.L. Rosenfeld et leur convergence a été étudiée en 1969, dans le cadre de la résolution de systèmes linéaires, par D. Chazan et W. Miranker. Par la suite pour résoudre des problèmes non-linéaires, des études et travaux ont été menés par G. Baudet, J. Bahi, Z.Z. Bai, D.P. Bertsekas, R.Couturier, D. El Baz, M.N. El Tarazi, A. Frommer, F. Magoulès, J.C. Miellou, P. Spiteri, D. Szyld et J. Tsitsiklis sachant que cette liste n'est pas exhaustive.

Dans le travail présenté dans ce mémoire, plusieurs résultats de convergence pour résoudre des problèmes non linéaires, avec des modèles d'asynchronisme de plus en plus généraux, prenant en compte des situations très diverses ont été élaborés et plusieurs expérimentations ont été effectuées sur diverses plate-formes afin de montrer la validité des algorithmes parallèles asynchrones pour traiter diverses applications couvrants différents domaines, tels que les mathématiques financières, la mécanique, l'interaction fluide-structure, la solidification de l'acier, le problème de séparation de protéine par électrophorèse, ... Nous nous sommes principalement intéressés à l'implémentation et à la simulation d'algorithmes parallèles asynchrone dans le cadre de la résolution de problèmes modélisées par des équations aux dérivées partielles linéaires ou pseudo-linéaires discrétisées définis dans des domaines tridimensionnels. Cette étude a pour but de présenter les moyens mis en œuvre pour mettre en place et réaliser les différentes simulations afin de résoudre ces différents problèmes.

Structure du mémoire

Ce mémoire est articulé de la façon suivante.

Le chapitre 1 se focalise très brièvement sur la modélisation des algorithmes parallèles asynchrones et leur analyse. Il présente la modélisation des algorithmes itératifs parallèle asynchrones, le modèle de point fixe associé, l'analyse de la convergence, les méthodes de sous-domaines avec et sans recouvrement, les méthodes de multi-décomposition et la terminaison des algorithmes asynchrones.

Le chapitre 2 permet de préciser le cadre informatique de ces travaux et de l'implémentation des algorithmes synchrones et asynchrones. Ce chapitre reprendra brièvement les bases du parallélisme, d'architectures et plateformes de calculs parallèles ainsi que les modes de communication associées afin d'appréhender l'implémentation des algorithmes étudiés et exécutés au cours de nos travaux. Il présentera l'implémentation de ces algorithmes itératifs asynchrones comme la mise en œuvre d'échanges de messages non bloquants, la détection de la terminaison de ces algorithmes et la mesure des performances des algorithmes.

Le chapitre 3 présente une synthèse de quelques applications traitées par des algorithmes itératifs asynchrones. Le traitement numérique est décrit brièvement et les résultats des simulations seront donnés. Par souci de concision, on renvoie au chapitre 5 pour un complément d'information sur d'autres applications.

Le chapitre 4 apporte une conclusion à cette étude et permet de prévoir des axes ultérieurs de recherche.

Ce mémoire se termine par un chapitre 5 comportant une sélection de travaux publiés.

Chapitre 1

Modélisation des algorithmes parallèles asynchrones

La simulation numérique est un atout essentiel pour l'étude de phénomènes multidisciplinaires modélisées mathématiquement par des lois les régissant. Ces modèles mathématiques peuvent être très complexe à résoudre du fait des structures de données volumineuses qu'ils peuvent générer, des temps de calculs extrêmement importants et des demandes de prise en compte de leur évolution dans le temps ; de plus les méthodes séquentielles ne permettent plus d'avoir des résultats dans des temps raisonnables et atteignent leur limite lorsque la taille des données et le temps de calcul augmentent.

Les architectures et les algorithmes parallèles permettent la résolution de systèmes de très grande dimension. Ces architectures sont composées d'unités de calculs permettant de multiplier et d'accélérer les possibilités de traitements simultanés. L'enjeu du parallélisme prend ici tout son sens car il va permettre de diminuer le temps de restitution des calculs et d'utiliser au mieux les ressources des machines en tenant compte de leurs architectures en fonction du problème à résoudre.

Contrairement aux applications parallèles classiques, résoudre des grands systèmes algébriques fait intervenir des algorithmes nécessitant de nombreux échanges de données entre les unités de calculs. Il est donc nécessaire de mettre en œuvre des techniques permettant de limiter les attentes entre celles-ci. En effet, le fait de devoir attendre des données distantes ou pas pour poursuivre un calcul peut dégrader la performance d'un algorithme parallèle. De plus, lors de communications synchrones entre les unités de calculs, l'efficacité peut décroître rapidement même si le nombre d'unités de calcul augmente.

Dès les années 1970, les algorithmes parallèles itératifs asynchrones permettant des actualisations des composantes du vecteur itéré sans synchronisation ni ordre précis ont été introduites par Chazan et Miranker [39] dans le cas de grands problèmes

linéaires.

Une des caractéristiques majeures des algorithmes itératifs asynchrones est leur aptitude à fonctionner correctement avec tout type de réseau de communication. Ces algorithmes pour lesquels les communications sont recouvertes par du calcul peuvent présenter une très bonne efficacité lorsqu'il y a beaucoup de synchronisations à effectuer entre les processeurs. Ce type de méthode permet de considérer des algorithmes distribués au cours desquels les processeurs effectuent leurs calculs à leur propre rythme en fonction de leurs caractéristiques intrinsèques et de leur charge propre.

De nombreux travaux scientifiques ont été réalisés dans ce domaine de recherche. Les principales références bibliographiques sont annexées dans ce mémoire et dans les différents articles présentés dans le chapitre 5.

Dans ce document, nous nous intéressons plus particulièrement à la résolution numérique d'équations aux dérivées partielles (EDP) elliptiques (modélisées par un laplacien ou par l'équation de convection-diffusion), paraboliques (pour l'équation de la chaleur) ou hyperboliques du second ordre (pour l'équation des ondes). La discrétisation de ces équations stationnaires conduit à la résolution de systèmes linéaires ou non linéaires de grande dimension suivant la nature du problème dont la matrice est creuse et dans le meilleur des cas a une structure bande. De même la discrétisation d'EDP d'évolution par des schémas implicites ou semi-implicite conduit également à la résolution de systèmes linéaires ou non linéaires ayant les mêmes caractéristiques.

Notons que dans le cas de la résolution numérique d'une équation aux dérivées partielle non linéaire, il est nécessaire de résoudre un système algébrique non linéaire en utilisant par exemple la méthode itérative de Newton ce qui nécessitera, à chaque itération, une linéarisation de l'application considérée autour du point courant et la résolution d'un système linéaire de grande dimension. Dans le cas moins classique de problèmes avec contraintes sur la solution, il existe d'autres techniques de linéarisation conduisant à la résolution de problèmes complémentaires où la méthode de la linéarisation est différente [34].

L'étude de la convergence de ce type de méthode est effectuée soit par des méthodes de contraction, soit par des techniques d'ordre partiel.

1.1 Problèmes modèles

On considère la résolution du problème pseudo-linéaire suivant :

$$AU^* + \Phi\left(U^*\right) = G \tag{1.1.1}$$

où A est une matrice de dimension $M \in \mathbb{N}$, ensemble des entiers naturels, G et U sont des vecteurs de dimension M et

$$U \to \Phi(U) \text{ est une application diagonale continue croissante.} \tag{1.1.2}$$

Un cas plus général que l'hypothèse (1.1.2) est réalisé quand l'opérateur Φ est monotone $< \Phi(U) - \Phi(V), U - V > \ \geqslant 0, \ \forall U, V$

L'application $U \to \Phi(U)$ est supposée diagonale ce qui signifie que la composante numéro i de Φ est donnée par $\Phi_i(U) \equiv \Phi_i(U_i)$. On suppose pour simplifier que l'espace $\mathcal{R}^M$ est normé par la norme euclidienne $||U||_2 = (< U, U >)^{1/2}$.

Lorsque la solution U est soumise à des contraintes du type inégalité suivante

$$U_{min} \leqslant U^* \text{ ou bien } U_{min} \leqslant U^* \leqslant U_{max} \text{ ou bien } U^* \leqslant U_{max} \tag{1.1.3}$$

on est amené classiquement à résoudre le problème suivant

$$AU^* + \partial\Psi\,(U^*) - G \ni 0 \tag{1.1.4}$$

où $U \to \Psi(U)$ est la fonction indicatrice de l'ensemble convexe K définissant les contraintes et $\partial\Psi(U)$ est le sous-différentiel de cette fonction indicatrice ce qui correspond en gros à la dérivée faible car Ψ n'est pas partout dérivable ;

On sait alors que l'application

$$\partial\Psi(U) \text{ est diagonale monotone.} \tag{1.1.5}$$

Dans les deux cas précédents, le problème sera résolu par une méthode spécifique. Pour le problème univoque (1.1.1), une linéarisation locale correspondant à l'implémentation par exemple de la méthode itérative de Newton sera effectuée. Pour le problème multivoque (1.1.4), en se limitant à la situation de problèmes où la solution est soumise aux contraintes de type inégalité (1.1.3), on combine judicieusement de façon alternée des opérations élémentaires d'algèbre linéaire et de projections sur l'ensemble convexe K des contraintes.

Dans chaque cas, la détermination de la solution consiste à résoudre itérativement un système algébrique de grande taille. Par un procédé adapté, on associe alors à chacun de ces systèmes non linéaires précédent (1.1.1) et (1.1.4), une équation de point fixe définie formellement par

$$U^* = F(U^*), \tag{1.1.6}$$

Cette équation est ensuite résolue par un algorithme parallèle asynchrone ou plus particulièrement synchrone. Par la suite on expliquera comment est définie chacune des applications de point fixe associée aux problèmes (1.1.1) et (1.1.4) et on analysera le comportement des algorithmes parallèles itératifs utilisés.

1.2 Modélisation des algorithmes parallèles asynchrones

Considérons maintenant la solution exacte U^* du système pseudo-linéaire (1.1.1) ou (1.1.4) où A est issue de la discrétisation d'une EDP.

Soit $\mathcal{E} = \mathbb{R}^{\mathbb{M}}$ un espace vectoriel normé par la norme euclidienne, pour simplifier la présentation. Soit α un entier naturel positif, correspondant au nombre de blocs associé à la décomposition du problème et tel que l'espace $\mathcal{E} = \prod_{l=1}^{\alpha} \mathcal{E}_l$ soit décomposé en un produit fini de α sous-espaces vectoriel normés notés $\mathcal{E}_l = \mathbb{R}^{m_l}$, avec $\sum_{l=1}^{\alpha} m_l = M$ où m_l représente la dimension du l-ème sous-espace $\mathcal{E}_l$.

Notons par par $|.|_l$, pour $l \in 1, ..., \alpha$ la norme de $\mathcal{E}_l$

On considère le problème général de point fixe associé à (1.1.6) définit par

$$\text{déterminer } U^* \in \mathcal{E} \text{ tel que } U^* = F(U^*) \tag{1.2.1}$$

avec $F : \mathcal{D}(F) \subset \mathcal{E} \longmapsto \mathcal{D}(F), V \longmapsto F(V)$ est une application de point fixe associée à la décomposition du problème.

En tenant compte de la décomposition de $\mathcal{E}$, on considère la décomposition correspondante en α blocs de $V \in \mathcal{E}$ et de $F(V) \in \mathcal{E}$

$$V = (V_1, \ldots, V_\alpha), F(V) = (F_1(V), \ldots, F_\alpha(V)) \tag{1.2.2}$$

où on note V_l ou F_l le bloc composante de V ou F pour $l \in 1, \ldots, \alpha$.

Pour résoudre ce problème du point fixe (1.2.1), on considère les itérations parallèles asynchrones définies de la façon suivante :

soit $U^0 \in \mathcal{E}$ donné alors, pour tout $r \in \mathbb{N}$, U^{r+1} est défini récursivement par

$$U_l^{r+1} = \begin{cases} F_l(U_1^{\kappa_1(r)}, \ldots, U_k^{\kappa_k(r)}, \ldots, U_\alpha^{\kappa_\alpha(r)}) \text{ si } l \in s(r) \\ U_l^r \text{ si } l \notin s(r) \end{cases} \tag{1.2.3}$$

où

$$\begin{cases} \forall r \in \mathbb{N}, s(r) \subset \{1, \ldots, \alpha\} \text{ et } s(r) \neq \emptyset, \\ \forall l \in \{1, \ldots, \alpha\}, \{r | l \in s(r)\} \text{ est dénombrable,} \end{cases} \tag{1.2.4}$$

et $\forall k \in \{1, \ldots, \alpha\}$

$$\begin{cases} \forall r \in \mathbb{N}, \kappa_k(r) \in \mathbb{N}, 0 \leqslant \kappa_k(r) \leqslant r, \\ \kappa_k(r) = r \text{ si } k \in s(r) \text{ et } \lim_{r \to \infty} \kappa_k(r) = \infty. \end{cases} \tag{1.2.5}$$

Les algorithmes parallèles itératifs asynchrones décrivent une méthode de calcul où le découpage du vecteur en α blocs prend en compte l'aspect parallèle du traitement numérique. Ainsi chaque bloc l est associé à un des α processeurs de calcul. Le parallélisme entre les processeurs est décrit par l'ensemble $s(r)$ qui contient le numéro des composantes relaxées par chaque processeur de manière parallèle.

L'utilisation dans (1.2.3) de composantes retardées $U_k^{\kappa_k(r)}$ provenant de relaxations antérieures dont les résultats sont disponibles permet de modéliser un comportement non déterministe des mises à jour et n'implique pas d'inefficacité du schéma de calcul distribué considéré. Cela modélise l'aspect asynchrone du calcul précisé par l'exposant avec $\kappa_k(r)$ défini par $\kappa_k(r) = r - d_k(r)$ où $d_k(r)$ est le retard de mise à jour de la composante k.

Remarquons qu'un cas particulier des itérations parallèles asynchrones est constitué par les itérations parallèles synchrones. En effet, il suffit de considérer l'absence de retards $d_k(r)$ c'est à dire $\forall k \in \{1, \ldots, \alpha\}$, $\forall k \in \mathbb{N}$ on a $k_k(r) = r$. De plus dans ce cas synchrone, on peut retrouver les algorithmes classiques suivants :

— si $\forall r \in \mathbb{N}$, $s(r) = \{1, \ldots, \alpha\}$, on obtient la méthode de Jacobi par blocs,
— si $\forall r \in \mathbb{N}$, $s(r) = 1 + (r \bmod \alpha)$, on obtient la méthode de Gauss-Seidel par blocs.

En résumé, les itérations asynchrones définies récursivement par (1.2.3) sont des méthodes itératives générales dans lesquelles les itérations sont effectuées en parallèle par un maximum de α processeurs sans ordre ni synchronisation. Dans le processus de calcul, l'asynchronisme est modélisé par l'introduction de composantes retardées calculées par d'autres processeurs pour tenir compte du couplage nécessaire entre les différents processus. La principale caractéristique de cette classe de méthodes itératives est de permettre des communications plus souples entre les processeurs et de minimiser le poids des synchronisations entre les processus parallèles. Dans ce type de méthode de calcul, afin de tirer le meilleur parti de la puissance de calcul en éliminant les temps importants dus aux attentes, on peut s'affranchir des synchronisations. Ceci évite d'attendre la communication des valeurs calculées par les autres processeurs ; ainsi, chaque processus effectue ses propres calculs en utilisant les données disponibles calculées par d'autres processeurs. Ainsi, chaque processeur exécute ses propres calculs à son propre rythme, les communications ne se faisant pas dans un ordre préétabli. Le choix des composants relaxés est effectué automatiquement et implicitement par le système informatique en sélectionnant plusieurs composants de la stratégie de $s(r)$ à chaque étape r du calcul et ce sans que le programmeur n'ait à faire quoique ce soit. Cette stratégie $s(r)$ est en fait un sous-ensemble non vide de l'ensemble $1, 2, ..., \alpha$ qui modélise bien le parallélisme entre les processus, puisque chaque élément de la stratégie n'est pas limité à un seul élément. En outre, chaque composante de bloc vecteur itéré est continuellement mise à jour ; dans la pratique, la méthode itérative parallèle est stoppée par un critère d'arrêt, ce qui, dans le contexte asynchrone, est très difficile à mettre en œuvre.

En pratique, la mise à jour de la l-ième composante s'effectue avec les dernières valeurs disponibles $U_k(r)$ de chaque composante. Il convient également de noter que dans le modèle d'itération parallèle asynchrone, les informations échangées entre les processeurs ne sont plus limitées dans le temps et peuvent avoir des délais de communication illimités, ce qui permet de prendre en compte d'éventuelles défaillances temporaires des machines multiprocesseurs.

Le schéma général d'un programme d'algorithme itératif parallèle asynchrone peut être modélisé de la façon suivante :

Algorithme 1 : Algorithme itératif parallèle asynchrone

$U^0 \leftarrow (U_1^0, \ldots, U_\alpha^0)$
$r \leftarrow 0$
tantque *norme* $>$ *tolerance*
  $r \leftarrow r + 1$
  pour $l \leftarrow 1$ **à** α **faire**
    si $l \in s(r)$ **alors**
      $U_l^{r+1} \leftarrow F_l(U_1^{\kappa_1(r)}, \ldots, U_k^{\kappa_k(r)}, \ldots, U_\alpha^{\kappa_\alpha(r)}$
    fin
    sinon
      $U_l^{r+1} \leftarrow U_l^r$
    fin
  fin
  Calcul de la norme
fin

1.2.1 Comportement des itérations asynchrones

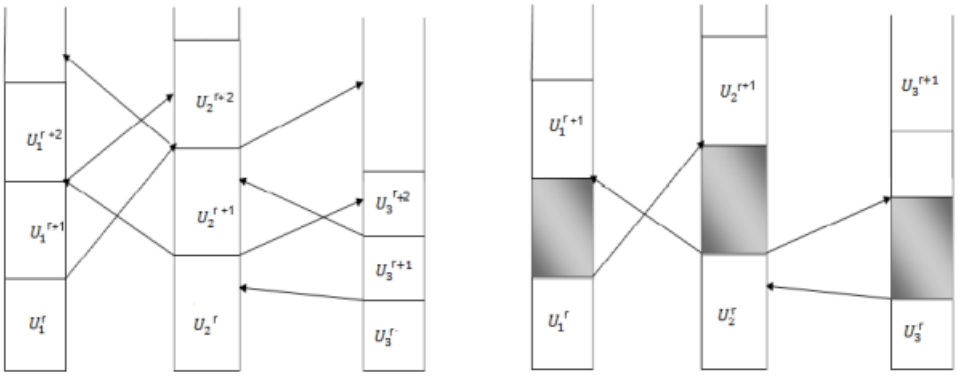


FIGURE 1.1 – Comportement des itérations parallèles synchrones et asynchrones

La figure 1.1 permet de visualiser le comportement des itérations asynchrones (à gauche) et synchrones (à droite) sur 3 processeurs. Lors de l'exécution en mode synchrone, les zones grisées correspondent à des phases d'inactivité des processeurs, phases dues aux attentes des processeurs de données produites par les autres processeurs ; en conséquence cette inactivité des processeurs engendre des synchronisations entre ces derniers.

Dans la version asynchrone, les zones grisées ont disparu et les processeurs sont constamment actifs. Intuitivement, si les temps d'inactivité lors de l'exécution de l'algorithme synchrone sont importants à cause d'un nombre important de synchronisations, on peut prédire un temps de restitution de la méthode en mode synchrone plus important qu'en mode asynchrone.

Cependant, cette constatation est à nuancer en tenant compte d'une part de la vitesse de convergence de la méthode itérative qui aura une influence sur le temps d'exécution de l'algorithme parallèle et d'autre part de l'architecture de la machine multiprocesseurs, en particulier du réseau d'interconnexion entre les processeurs ainsi que de la vitesse de communication entre ces derniers.

Classification des méthodes paralléles

Dans cette partie, nous allons montrer comment on classifie les méthodes parallèles synchrones et asynchrones.

SISC - *Synchronous Iterations, Synchronous Communications*

Les méthodes itératives les plus courantes sont celles utilisant des itérations synchrones et des communications synchrones (SISC). Le synchronisme des itérations est dû au fait que chaque processeur ne peut commencer à calculer la nouvelle itération que s'il a reçu les données calculées à l'itération précédente par tous ses voisins. Par conséquent, tous les processeurs commencent les calculs de la même itération en même temps et échangent les données à la fin de chaque itération par le biais de communications globales synchrones.

La figure 1.2 illustre l'exécution d'un algorithme parallèle SISC. Le synchronisme des itérations d'un algorithme SISC implique un nombre d'itérations identique à celui de l'algorithme séquentiel correspondant. Donc, les conditions de convergence des algorithmes parallèles SISC sont les mêmes que celles d'un algorithme séquentiel. Cependant, la synchronisation des communications peut souvent être la cause de périodes d'inactivité des processeurs car ils doivent attendre des mises à jour avant de communiquer leurs données ou encore car la vitesse du réseaux engendre des communications trop lentes.

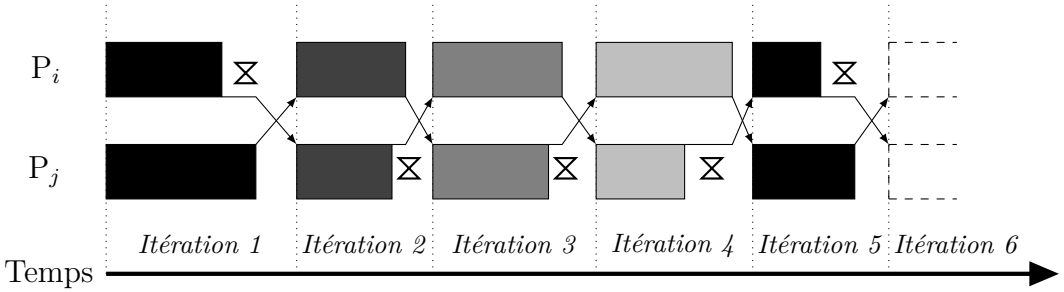


FIGURE 1.2 – Exemple d'un schéma SISC sur deux processeurs P_i et P_j

SIAC - *Synchronous Iterations, Asynchronous Communications*

Les solveurs itératifs utilisant des itérations synchrones mais des communications asynchrones (SIAC) ont été développés avec pour objectif d'améliorer les performances sur des réseaux avec des inter-connexions lentes et/ou hétérogènes. De manière identique aux algorithmes SISC, un processeur attend toujours la réception des données partagées calculées par tous ses voisins à l'itération précédente. Cependant, les communications synchrones globales sont remplacées par des envois asynchrones et des réceptions bloquantes.

La figure 1.3 illustre l'exécution d'un algorithme parallèle SIAC. Les conditions de convergence sont les mêmes que pour un algorithme SISC ou un algorithme séquentiel. Cependant, les communications asynchrones permettent une superposition entre calculs et transferts de données. Un processeur peut envoyer des données partagées à ses voisins dès qu'elles sont prêtes à être utilisées dans la nouvelle itération, ce qui a pour conséquence de réduire les temps d'attente nécessaires pour que les données soient reçues entre deux itérations successives.

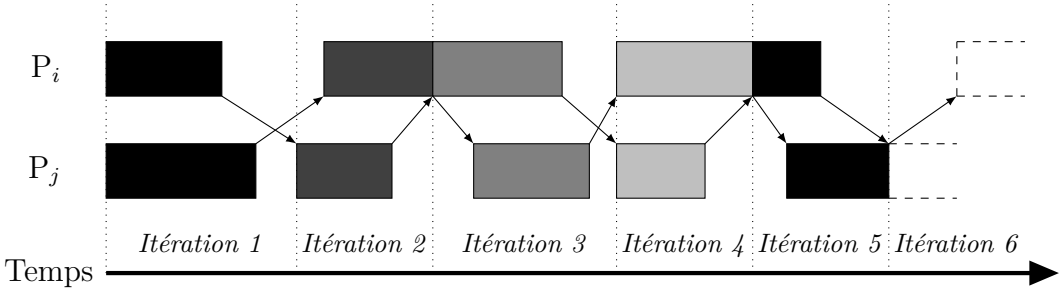


FIGURE 1.3 – Exemple d'un schéma SIAC sur deux processeurs P_i et P_j

AIAC - *Asynchronous Iterations, Asynchronous Communications*

Les solveurs itératifs asynchrones avec communications asynchrones (AIAC) ont été développés pour améliorer les performances sur des plateformes de calcul constituées de ressources hétérogènes et géographiquement distantes. En effet, les méthodes itératives parallèles SISC et SIAC sont relativement faciles à mettre en œuvre mais pénalisent les performances sur ce type de plateforme à cause du synchronisme de leurs itérations. Sur ce type d'architecture, les réseaux de communication sont souvent lents et/ou l'hétérogénéité des ressources engendre des temps de calcul pouvant être très différents suivant les processeurs.

Dans les méthodes parallèles AIAC, chaque processeur exécute ses propres itérations sans se préoccuper de la progression des autres processeurs ; il passe d'une itération à l'autre sans attendre la réception de résultats calculés par les autres processeurs voisins à l'itération précédente et utilise des données locales ainsi que les versions les plus récentes des résultats calculés par les autres processeurs disponibles au début de chaque itération. Les solveurs AIAC utilisent des communications asynchrones de type envois et réceptions non bloquants.

La figure 1.4 illustre l'exécution d'un algorithme parallèle AIAC. Les itérations asynchrones permettent d'éliminer les temps d'inactivité des processeurs, prennent en compte des itérations avec des numéros différents qui peuvent alors être exécutées à un même moment ; certains processeurs peuvent donc être plus rapides et effectuer plus d'itérations que d'autres.

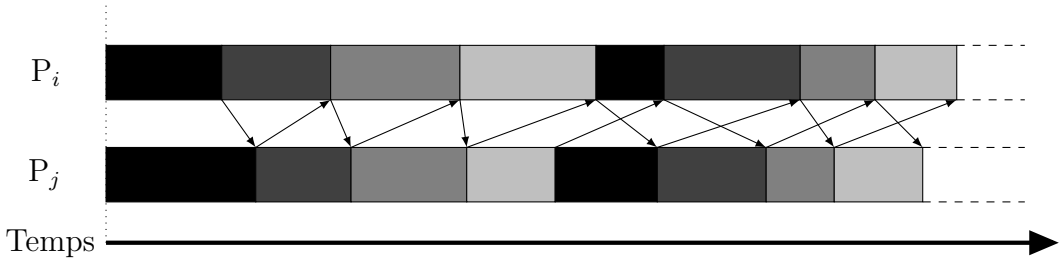


FIGURE 1.4 – Exemple d'un schéma AIAC sur deux processeurs P_i et P_j

Ces méthodes nécessitent des communications de type relaxation asynchrones.

1.2.2 Analyse de la convergence des méthodes parallèles asynchrones

Considérons le problème pseudo-linéaire (1.1.1) et (1.1.4) issu de la discrétisation de problèmes aux limites.

L'hypothèse (1.1.2) étant satisfaite pour le problème (1.1.1) ou encore l'hypothèse

(1.1.6) étant valide pour le problème (1.1.4), on suppose que l'hypothèse suivante est vérifiée.

$$A \text{ est une M-matrice} \tag{1.2.6}$$

Rappelons qu'une M-matrice est une matrice à coefficients hors diagonaux négatifs ou nuls et dont l'inverse est non-négative.

On montre classiquement que les problèmes (1.1.1) et (1.1.4) admettent une solution unique (voir par exemple [17], [77]).

Comme indiqué précédemment, l'analyse de la convergence des méthodes parallèles asynchrones peut s'effectuer essentiellement soit par des techniques de contraction, ou soit par des techniques d'ordre partiel liée à l'utilisation du principe du maximum discret, ou soit encore en utilisant les ensembles emboîtés ce qui permet d'unifier les deux approches précédentes mais ne fournit pas de critère pratique de caractérisation de la convergence. Nous rappelons les résultats suivants :

Theorem 1 (Convergence en norme vectorielle)**.**
Soit F une application de point fixe de $D(F) \subset \mathcal{E}$ à valeurs dans $D(F)$. Supposons que F vérifie l'inégalité suivante

$$q(F(V) - F(U^*)) \leqslant J.q(V - U^*), \forall V \subset \mathcal{E}$$

où

- *J est une matrice non négative de dimension α de rayon spectral $\rho(J) < 1$*
- *$q(.)$ est une norme vectorielle définie sur $\mathcal{E}$ par*

$$q(V) = (|V_1|_1, \ \ldots \ , |V_\alpha|_\alpha)$$

alors F admet un point fixe unique U^*. De plus la suite $(U^r)_{r\in\mathbb{N}}$ construite à l'aide de l'algorithme parallèle asynchrone (1.2.3) converge vers la solution U^* des problèmes (1.1.1) et (1.1.4).

Ce résultat a été a démontré par J.C. Miellou dans [113] dans le cas de retards bornés mais il reste encore valable dans le cas de retards non bornés ([18]).

A partir de ce résultat, on peut établir un théorème de convergence basé sur l'utilisation de la norme uniforme avec poids définie par :

$$\forall U \in \mathcal{E}, ||U||_\Gamma = \max_{1\leqslant i\leqslant\alpha} \frac{|x_i|_i}{\gamma_i} \tag{1.2.7}$$

où le vecteur Γ de composante $\gamma(i)$ strictement positive et de dimension α vérifie $J\Gamma \leqslant \nu\Gamma$ ([150]).

On peut donc énoncer le résultat suivant

Theorem 2 (Convergence en norme scalaire)**.**
Soit F une application de $D(F) \subset \mathcal{E}$ à valeurs dans $D(F)$. Alors sous les hypothèses précédentes, F admet un point fixe unique $U^* \in \mathcal{E}$ Si de plus F est contractante en U^* pour la norme uniforme avec poids précédente $||.||_\Gamma$, soit ceci :

$$\exists \nu \in [\rho(J)\,,\,1[, telque\ ,\ \|F(V) - F(U^*)\|_\Gamma \leqslant \nu \|V - U^*\|_\Gamma, \forall V \in \mathcal{E} \tag{1.2.8}$$

alors la suite $(U^r)_{r\in\mathbb{N}}$ construite à l'aide de l'algorithme parallèle asynchrone (1.2.3) converge vers le point fixe U^*, pour la norme $||.||_\Gamma$.

Nous renvoyons à [113] et [150] pour les démonstrations.

L'intérêt de l'analyse par des techniques de contraction, permet d'estimer la vitesse de convergence grâce à la valeur de la constante de contraction ν ; si ν est petit, la convergence sera rapide, par contre si ν est proche de 1, la convergence sera lente.

Si l'hypothèse (1.2.6) est vérifiée, ainsi que les hypothèses (1.1.2) ou (1.1.5) alors les méthodes parallèles asynchrones pour résoudre les problèmes (1.1.1) ou (1.1.4) sont convergentes.

On peut aussi analyser la convergence des itérations asynchrones soit par des technique d'ordre partiel soit encore par des propriétés d'ensembles emboîtés qui seront précisés ci-dessous.

Pour l'analyse en ordre partiel des itérations parallèles asynchrones, on suppose que l'hypothèse (1.2.6) est vérifiée et que Φ est une fonction croissante. On considère une donnée initiale $U^0 \in \mathcal{E}$ (respectivement $V^0 \in \mathcal{E}$) qui est une $\mathcal{A}$-sur-solution (respectivement une $\mathcal{A}$-sous-solution) vérifiant $\mathcal{A}(U^0) = A(U^0) + \Phi(U^0) - G \geqslant 0$ (respectivement $\mathcal{A}(V^0) = A(V^0) + \Phi(V^0) - G \leqslant 0$). Alors, avec ces initialisations, on a

$$U^r \downarrow U^*, r \longrightarrow \infty, \tag{1.2.9}$$

et respectivement

$$V^r \uparrow U^*, r \longrightarrow \infty, \tag{1.2.10}$$

où 1.2.9 signifie que $\lim\limits_{r\to\infty} U^r = U^*$ et $U^* \leqslant \ldots \leqslant U^{r+1} \leqslant U^r \leqslant \ldots \leqslant U^0$, (respectivement 1.2.10 signifie que $\lim\limits_{r\to\infty} V^r = U^*$ et $U^* \geqslant \ldots \geqslant V^{r+1} \geqslant V^r \geqslant \ldots \geqslant V^0$).

Donc si on considère le problème (1.1.1) uniquement, à condition que l'approximation U^0 soit une $\mathcal{A}$-sur-solution (où V^0 soit une $\mathcal{A}$-sous-solution) les itérations

parallèles asynchrones appliquées à la résolution de de problème convergent de façon monotone.

Une autre approche d'analyse de la convergence des itérations parallèles asynchrones proposée par D. Bertsekas [25] (voir aussi D. Bertsekas et J. Tsitsiklis [26]) est obtenue lorsque que les itérations successives sont localisées dans des ensembles emboîtés admettant pour centre la solution U^* . Par conséquent comme ces ensembles deviennent de plus en plus petits, les méthodes itératives parallèles asynchrones convergent.

1.2.3 Découpage en sous-domaines

En calcul parallèle afin de résoudre des équations aux dérivées partielles, pratiquement, il est utile d'utiliser les méthodes de décompositions de domaine qui consistent à découper un domaine en plusieurs sous-domaines. Ainsi, afin de permettre de résoudre des problèmes de taille de plus en plus importante et de plus en plus rapidement, on résout donc en parallèle des sous-problèmes de plus petites tailles définies sur chaque sous-domaine couplés entre eux ce qui implique des communications synchrones ou asynchrones entre les processus. Par contre il est nécessaire que la décomposition ait une granularité suffisamment élevée car des tailles de sous-domaines trop petites auront un effet négatif sur l'efficacité des méthodes utilisées ainsi qu'une augmentation du coût de communications et de synchronisations entre les processus de calcul. L'avantage de ce type de découpage est l'excellente compatibilité avec un haut niveau de parallélisme (voir Figure 1.5).

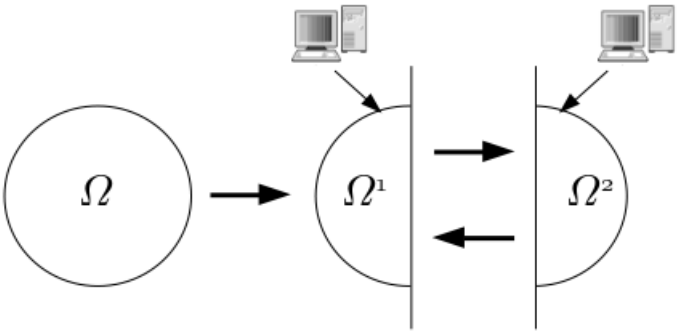


FIGURE 1.5 – Exemple d'un découpage

Soit Ω un domaine décomposé en β sous-domaines tels que $\Omega = U_{k=1}^{\beta}\Omega_k$, sachant que ces β sous-domaines Ω_k peuvent se recouvrir ou non en fonction de la méthode utilisée. Sur chaque sous-domaine Ω_k, la résolution de problèmes locaux est de même nature ou pas que le problème global. En fait, le couplage des problèmes locaux se fait alors au niveau des interfaces ou des recouvrements entre les sous-domaines.

La parallélisation de ces méthodes est alors naturelle : les sous-domaines et les calculs locaux correspondants sont répartis sur différents processeurs.

Le but est de limiter les surcouts de calculs locaux par rapport au calcul global et de minimiser les couplages entre sous-domaines.

Notons que certaines méthodes de décomposition de domaine ont un équivalent algébrique : les matrices du système linéaire associé au problème global sont décomposées en blocs correspondant au découpage en sous-domaines.

Il existe deux méthodes possibles de définition des méthodes de sous-domaine :
— celles avec recouvrement ;
— celles sans recouvrement.

Méthode avec recouvrement

En ce qui concerne le premier type de méthode, à la fin du 19^e siècle, le mathématicien suisse H. A. Schwarz introduisit une méthode itérative dans le but de résoudre analytiquement des problèmes de diffusion ou de convection-diffusion. L'idée fondamentale de Schwarz est de décomposer le problème global en un ensemble de sous-problèmes de telle sorte que le domaine spatial global soit partitionné avec recouvrement (voir figure 1.6).

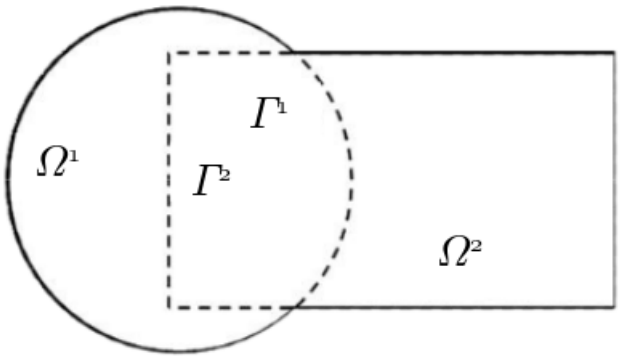


FIGURE 1.6 – Problème proposé par Schwarz

Soit Ω de frontière $\partial\Omega$ le domaine global sur lequel on cherche à résoudre le problème :

$$\begin{cases} \mathcal{L}u = f \quad \text{dans } \Omega \\ u = g \quad \text{sur } \partial\Omega \end{cases} \tag{1.2.11}$$

où u est l'inconnue du problème et $\mathcal{L}$ un opérateur différentiel. Le problème (1.2.11)

est partitionnée en β sous-domaines avec recouvrement tels que :

$$\forall i \in \{1, \ldots, \beta\},\ \Omega_i \subset \Omega\ \Omega = \bigcup_{k=1}^{\beta} \Omega_k \text{ et } \Omega_i \cap \Omega_j \neq \emptyset \text{ et } \forall i, j \text{ avec } i \neq j \qquad (1.2.12)$$

Ce processus itératif est aujourd'hui connu sous le nom de méthode alternée de Schwarz. La démonstration de la convergence a été établie par Schwarz via le principe du maximum, puis par P.L. Lions [91] à l'aide de considérations variationnelles.

Notons adj(i) l'ensemble des sous-domaines adjacents à Ω_i, défini comme suit :

$$\text{adj}(i) = \{j \mid j \neq i \text{ et } \Omega_i \cap \Omega_j \neq \emptyset\}.$$

Soient $(\Gamma^i)_{1 \leqslant i \leqslant \beta}$ les restrictions des frontières de Ω à celles des sous-domaines Ω_i tels que $\Gamma^i = \partial\Omega_i \cap \partial\Omega$ frontières entre Ω^i et ses sous-domaines adjacents, elles sont notées : $\gamma_i^j = \partial\Omega^i \cap \Omega^j$, pour tout $j \in \text{adj}(i)$.

La méthode de Schwarz étant un algorithme itératif, cela revient à résoudre à chaque itération β sous-problèmes parallèles définis sur les sous-domaines $(\Omega_i)_{1 \leqslant i \leqslant \beta}$.

Cette méthode est bien adaptée au parallélisme [85]. Chaque sous-problème associé à une itération de l'algorithme est résolu indépendamment mais simultanément des autres (voir [114, 55, 58, 63, 65] pour des présentations détaillées de cette méthode).

La méthode de Schwarz avec recouvrement présentée ci-dessus est assez facile à mettre en œuvre, excepté pour la gestion du recouvrement car il faut gérer la transmission des données d'une frontière à l'autre. L'analyse de la convergence de cette méthode s'effectue sur le même principe que celui présenté au paragraphe (1.2.2).

Méthodes sans recouvrement

Les méthodes sans recouvrement consistent à regrouper plusieurs composantes adjacentes du vecteur à calculer pour obtenir un bloc de plus grande taille. En fait,ces méthodes sans recouvrement correspondant à des méthodes de relaxation par grands blocs.

Il est à signaler que l'analyse du comportement du processus itératif de plus forte granularité ainsi obtenu se déduit aisément des résultats obtenus dans le cadre de l'analyse de la convergence par les techniques de contraction présentées dans la sous-section 1.2.2 ainsi que des techniques d'ordre partiel présentées précédemment. En effet, si la matrice de discrétisation est une M-matrice, une situation à laquelle on peut souvent se ramener en choisissant convenablement les schémas de discrétisation, les méthodes parallèles de sous-domaine asynchrones analysées par des techniques de contraction sont convergentes quelle que soit la décomposition plus grossière.

En ce qui concerne l'analyse de la convergence par des techniques d'ordre partiel des algorithmes parallèles asynchrones appliqués à la résolution de problèmes pseudo-linéaires uniquement de type (1.1.1), grâce aux propriétés des M-matrices, des résultats de convergence similaires peuvent être obtenus ; en effet, les sous-problèmes obtenus en regroupant les blocs diagonaux de la matrice de discrétisation sont également des M-matrices ; les résultats établis dans le cadre de la contraction s'appliquent donc également à ce cas.

De ces faits dans les deux cas précédents, pour paralléliser les calculs, on découpe le domaine Ω comme suit

— dans le cas bi-dimensionnel, le domaine Ω est un rectangle décomposé en rectangles se recouvrant ou non,
— dans le cas tri-dimensionnel, le domaine Ω est un parallélépipède décomposé en parallélépipèdes se recouvrant ou non.

Dans le cas où Ω est de forme quelconque, on adapte au mieux le découpage.

FIGURE 1.7 – Exemple d'un domaine sans recouvrement

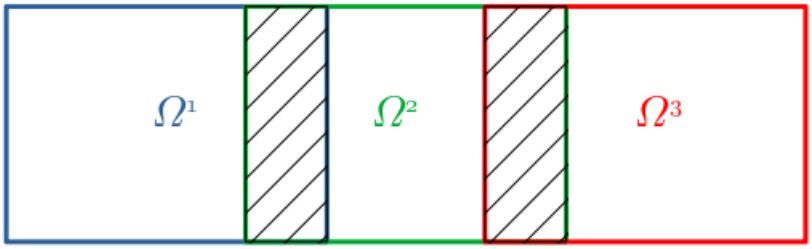


FIGURE 1.8 – Exemple d'un domaine avec recouvrement

On peut noter que la décomposition du problème global en sous-problèmes peut augmenter la précision des calculs effectués, particulièrement si le problème est mal conditionné, situation où l'accumulation des erreurs de chute peut dénaturer les résultats obtenus. En effet, pour des tailles de domaines qui diminuent le nombre de conditionnement diminue ce qui améliore la qualité numérique de la résolution.

1.2.4 Méthodes de multi - décomposition asynchrones

Le principe des méthodes de multi-décomposition (ou méthodes de multi-splitting) permet de donner une présentation unifiée de la plupart des méthodes de sous-domaines et par conséquent, permet d'analyser le comportement des toutes ces méthodes d'une façon unique. L'introduction de ces méthodes permet également de s'affranchir d'une décomposition en sous-domaines. Ces méthodes ont été beaucoup étudiées (voir [5, 12, 14, 16, 53, 54, 118, 143, 148, 153, 155, 119, 139, 69, 70]).

Considérons la solution du problème discrétisé (1.1.1) ou (1.1.4) respectivement lorsque les hypothèses (1.2.6) et (1.1.2) ou (1.1.5) respectivement sont vérifiées. Soit les splitting réguliers suivants de la matrice A

$$A = M^l - N^l, \quad l = 1, ..., m, m \in \mathbb{N} \tag{1.2.13}$$

où $(M^l)^{-1} \geqslant 0$ et $N^l \geqslant 0$.

Soit $F^l : R^n \to R^n$, $l = 1, ..., m$, applications de point fixe associés au problème (1.1.1) ou (1.1.4), une multi-décomposition formelle associée au deux problèmes précédents est défini par l'ensemble des problèmes de points fixes (voir [12])

$$U^* = F^l(U^*), \ l = 1, ..., m. \tag{1.2.14}$$

Soit $E = \mathbb{R}^M$ et considérons la décomposition par blocs de E

$$E = \prod_{l=1}^{m} E_l,$$

où $E_l = \mathbb{R}^M$.

Par la suite, on analyse ces méthodes aussi bien par des techniques de contraction ([12]) que, excepté pour le problème (1.1.4), par des techniques d'ordre partiel.

Soit $\tilde{U} \in \mathcal{E}$ défini par

$$\tilde{U} = \{U^1, \ldots, U^l, \ldots, U^m\} \in \prod_{l=1}^{m} E_l.$$

Définition 1.2.1. L'application de point fixe étendue $\tilde{T} : \mathcal{E} \to \mathcal{E}$ associée est définie comme suit

$$\tilde{T}(\tilde{V}) = \tilde{U}, \text{ tel que } U^l = F^l(W^l) \text{ avec } W^l = \sum_{k=1}^{m} D_{lk} V^k, l = 1, ..., m,$$

où D_{lk} sont des matrices de pondération diagonale non négatives satisfaisant pour tous $l \in \{1, \ldots, m\}$

$$\sum_{k=1}^{m} D_{lk} = I_l,$$

I_l étant la matrice d'identité dans $L(E_l)$.

Puisque $F^l(\mathcal{D}(F^l)) \subset \mathcal{D}(F^l)$, alors $\tilde{T}(\mathcal{D}(\tilde{T})) \subset \mathcal{D}(\tilde{T})$ où $\mathcal{D}(\tilde{T}) = \prod_{l=1}^{m} \mathcal{D}(F^l)$.

Il convient de noter qu'une économie considérable de calcul peut être possible en utilisant une telle méthode, puisqu'une composante de V^k n'est pas nécessaire à calculer si la matrice de pondération est nulle ; dans ce cas, en calcul parallèle, le rôle de ces matrices de pondération peut être considéré comme déterminant la répartition du travail de calcul des différents processeurs.

Il est à noter que pour un choix particulier des matrices de pondération D_{lk}, on peut obtenir différentes méthodes itératives et notamment, d'une part, une méthode de sous-domaine sans chevauchement et, d'autre part, la méthode classique de Schwarz alternée (voir [12]). En effet, par exemple, selon [12], la méthode de Jacobi par bloc correspond au choix suivant de M^l

$$M^l = diag(I_1, \ldots, I_{l-1}, A_{l,l}, I_{l+1}, \ldots, I_n) \ , \tag{1.2.15}$$

et le choix de $D_{lk} \equiv \bar{D}_l$ est donné par

$$\bar{D}_l = diag(0, \ldots, 0, I_l, 0, \ldots, 0), \tag{1.2.16}$$

ce qui, en d'autres termes, signifie que les coefficients des matrices de pondération sont égales à un ou à zéro.

Pour la méthode de Schwarz alternée, plusieurs processeurs calculent les valeurs actualisées d'une même composante, et les matrices $\bar{D}_l$ ont des coefficients positifs inférieures à un. On renvoie à [118] pour d'autres choix de pondération des matrices diagonales et pour la définition d'autres méthodes de multi-décomposition.

1.2.5 Terminaison des algorithmes asynchrones

Le critère d'arrêt des itérations parallèles asynchrones est un problème particulièrement difficile du fait du comportement non déterministe des méthodes asynchrones. En effet, la terminaison des itérations parallèles asynchrones est délicate à coder car il relève à la fois des mathématiques appliquées et mais aussi de l'informatique.

Nous examinerons au paragraphe 2.2.5 la terminaison de ces algorithmes d'un point de vue informatique.

Sur un plan mathématique, la terminaison doit se réaliser lorsque le vecteur itéré est suffisamment proche de la solution du problème. Les paragraphes suivants présentent des méthodes de terminaison des algorithmes asynchrones d'un point de vue numérique.

Nous présentons ci-dessous, trois approches distinctes de détection de terminaison d'algorithmes parallèles asynchrones basées sur les propriétés qui ont permis d'analyser le comportement de ces algorithmes itératifs en particulier leur convergence.

Utilisation de l'algorithme secondaire du contrôle des erreurs

Dans [113], J.C. Miellou a proposé une méthode de détection de convergence basée sur l'utilisation d'un algorithme secondaire de contrôle des erreurs dérivé de l'algorithme de F. Robert - G. Schroeder [125]. Dans le cadre de l'étude de la convergence par des techniques de contraction, cet algorithme secondaire permet de résoudre le système $z = Jz$ de dimension α, où J est la matrice de contraction non négative de rayon spectral plus petit que l'unité.

Quelque soit la donnée initiale z^0 , l'algorithme secondaire convergera évidement vers zéro, solution du problème $z = Jz$.

Donc à chaque pas r, en choisissant pour l'algorithme principal parallèle asynchrone et l'algorithme secondaire la même stratégie de choix des composantes à relaxer et les mêmes valeurs retardées, J.C. Miellou a établi qu'à condition d'initialiser l'algorithme secondaire par un vecteur z^0 vérifiant $q(U^0 - U^*) \leqslant z^0$, où $U^0 - U^*$ est l'erreur initiale, à chaque pas r de l'algorithme l'inégalité suivante est vérifiée $q(U^r - U^*) \leqslant z^r$.

On pourra donc stopper l'algorithme parallèle asynchrone principal, lorsque toutes les composantes du vecteur z^r sont de module inférieure à une tolérance donnée ϵ.

L'algorithme secondaire nécessite nettement moins de calculs que l'algorithme principal et peut très bien être implémenté à moindre coût en fournissant une méthode de terminaison fiable et économique des méthodes parallèles asynchrones.

Utilisation de la convergence monotone

Dans la sous-section 1.2.2, concernant l'analyse en ordre partiel des itérations parallèles asynchrones, on peut aussi obtenir un critère d'arrêt fiable, lorsqu'on considère deux suites de vecteurs itérés associées à une A-sur-solution de A et à une A-sous-solution de A respectivement.

Si on considère les itérations de point fixe parallèles asynchrones générées par ces données initiales, on a

$$V^0 \leqslant V^1 \leqslant \ldots \leqslant V^r \leqslant \ldots \leqslant U^* \leqslant \ldots \leqslant U^q \leqslant \ldots \leqslant U^1 \leqslant U^0, \tag{1.2.17}$$

et, si ϵ est la tolérance considérée pour arrêter le processus itératif, alors on peut considérer le critère d'arrêt fiable mais cependant coûteux suivant :

$$U^q - V^r \leqslant \epsilon, \forall q, r \in \mathbb{N}. \tag{1.2.18}$$

Utilisation des ensembles emboîtés

En prenant en compte la perturbation par les erreurs d'arrondi [147] et [107], une autre approche pour stopper de manière dynamique les algorithmes itératifs parallèles asynchrones a été développée par J.C. Miellou, P. Spiteri et all dans [112], [111] et [137] dans un contexte topologique général.

Cette approche utilise essentiellement la notion d'ensembles emboîtés étudiée par D. Bertsekas et al (voir [25] et [26]) pour analyser la convergence de ces méthodes.

Le principe de cette approche est le même que le problème soit linéaire ou non linéaire. Cependant l'effet des erreurs d'arrondi est pris en compte en utilisant une notion de contraction approchée introduite par J. Wilkinson (voir [158]) et étendue dans [107] dans le contexte d'itérations parallèles asynchrones.

Afin de donner une estimation de la distance entre la solution exacte et la valeur de l'itéré courant, et donc de proposer plusieurs critères d'arrêt originaux en ligne pour stopper les itérations parallèles asynchrones, donc utilisables pour les méthodes générales de point fixe définies dans un espace produit, J.C. Miellou, P. Spiteri et all ont développé une étude complète permettant d'obtenir, lorsque les tests d'arrêt envisagés sont satisfaits, des majorations d'erreur ainsi que des estimations précises et très utiles des normes du résidu.

Ces méthodes de terminaison sont définies grâce à une estimation du diamètre des ensembles emboîtés, centrés sur la solution cherchée, et qui contiennent un certain nombre de vecteurs itérés sélectionnés ; la construction de tels ensembles est possible lorsque la convergence des méthodes itératives parallèles asynchrones sont analysées par des techniques de contraction ou de contraction approchée (voir [115]) dans ce dernier cas défini par $||F(U)-F(V)|| \leqslant \nu||V-U||+\delta$ où δ prend en compte les erreurs de chute. Cela correspond au fait que l'application de point fixe est contractante par rapport à une norme vectorielle (voir [112], [111] et [137]). En d'autres termes, les critères d'arrêt considérés sont formulés en prenant en compte la distance entre les extrémités du diamètre de l'ensemble emboîté courant contenant un ou plusieurs vecteurs itérés. Ainsi, si ce diamètre est inférieur ou égal à une tolérance donnée, le processus itératif sera stoppé.

Les critères d'arrêt proposés correspondent aux trois cas suivants
- un critère d'arrêt par rapport à l'erreur absolue,
- un critère d'arrêt par rapport à l'erreur relative,
- un critère d'arrêt mixte par rapport à la norme vectorielle de l'erreur, c'est-à-dire la combinaison de l'erreur absolue et de l'erreur relative.

Les trois critères d'arrêt précédents sont décrits par rapport aux distances générales définies sur l'espace à M dimensions. On peut également noter que le critère d'arrêt défini par rapport à l'erreur absolue et le critère d'arrêt défini par rapport

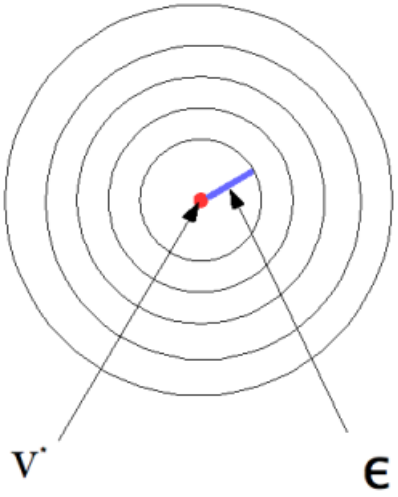


FIGURE 1.9 – Technique des ensembles emboîtés.

à l'erreur relative peuvent être formulés en utilisant des normes générales définies dans l'espace à M dimensions ; par contre le critère d'arrêt mixte est défini dans un contexte plus particulier.

Dans tous les cas, il a été démontré que chaque test de terminaison est cohérent et que les bornes de la norme du résidu et de l'erreur sont estimées, aussi bien pour le test d'arrêt par rapport à la valeur absolue, que pour le test d'arrêt par rapport à l'erreur relative ainsi que pour le critère d'arrêt mixte. Enfin, la détection dynamique de la terminaison basée sur les erreurs absolues et les erreurs en norme vectorielle sont particulièrement bien adaptées à des contextes d'utilisation de machines à mémoire distribuée, de grille de calcul, de calcul pair à pair ("peer to peer") et de cloud computing. A noter que les preuves de validité des différents tests d'arrêt peuvent être consultés sur [112], [111] et [137].

1.3 Conclusion

Ce chapitre a permis de mettre en place très brièvement les notions essentielles décrivant les algorithmes parallèles asynchrones. Nous avons décrit les méthodes parallèles de point fixe, analysé la convergence des méthodes parallèles asynchrones. Nous avons aussi présenté la décomposition en sous-domaines, la multi-décomposition asynchrone correspondant à une généralisation des méthodes de sous-domaine et les modes de terminaison numériques des algorithmes asynchrones.

Le chapitre suivant va nous permettre de décrire comment implémenter d'une manière informatique les concepts définis ci-dessus.

Chapitre 2

Cadre informatique

2.1 Introduction

Ce chapitre va permettre de présenter le cadre informatique utilisé pour le développement et la simulation des algorithmes itératifs parallèles asynchrones. Dans un premier temps, nous allons rappeler les notions d'architectures parallèles, de parallélisme et des modes de communication. Il va permettre aussi de décrire les routines essentielles de la bibliothèque MPI qui permettent d'échanger des messages bloquants et non-bloquants. Une méthode originale de mesure de performance des algorithmes asynchrones sera présentée et un accent particulier sera donné sur l'implémentation des tests d'arrêt des algorithmes parallèle asynchrones.

2.2 Cadre de l'implémentation informatique

Le calcul haute-performance regroupe la conception d'algorithmes efficaces en temps d'exécution, l'architecture des ordinateurs, la mise en œuvre des programmes et l'analyse des performances. Elle permet la résolution de problèmes de calcul scientifique pluridisciplinaires, de bases de données gourmandes en ressources CPU et en temps de calcul important et permet aussi de passer outre les limites des architectures séquentielles en terme de performance, d'accès à la mémoire et de tolérance aux pannes.

Le principe est d'utiliser des architectures parallèles composées de plusieurs processeurs permettant le traitement d'applications en dépassant les limites de la mémoire locale ainsi que les limites des processeurs et en diminuant notablement la durée des calculs comme par exemple le calcul matriciel, la simulation numérique, ...

2.2.1 Architectures et plateforme de calculs parallèles

L'enjeu du parallélisme est d'utiliser au mieux les ressources des machines en tenant compte de leurs architectures et cela en fonction du problème à résoudre. La diminution par un facteur deux du temps de restitution de calculs est déjà appréciable.

L'utilisation des architectures parallèles permet d'augmenter la performance des calculs : latence, débit, puissance de calcul, augmentation du stockage, ... Ce genre d'architecture permet de s'affranchir de certaines limites d'architectures classiques comme par exemple la taille mémoire ou bien le nombre de processeurs.

Les machines parallèles permettent d'effectuer un calcul distribué, en exploitant la puissance de calcul d'un grand nombre d'ordinateurs ou de processeurs. Différentes architectures existent mais il faut garder à l'esprit que l'hétérogénéité et la distance entre les machines composant cette architecture influent sur la programmation parallèle mais aussi sur la performance des applications simulées.

Il existe principalement deux types de machines parallèles. D'une part, les machines parallèles à mémoire partagée permettant une parallélisation assez facile. Cependant le nombre de processeurs utilisés est limité. D'autre part, les machines parallèles à mémoire distribuée dans lesquelles chaque processeur possède une mémoire dédiée et les communications parallèles se réalisent par échange de messages. Une classification des machines parallèles a été proposée par Flynn ([52]).

On appelle flot (en anglais stream), une séquence d'objets (instructions ou données) exécutés ou traités par un processeur. On peut distinguer plusieurs classes d'architectures parallèles nommées SISD (Single Instruction stream, Single Data stream), SIMD (Single Instruction stream, Multiple Data streams), MISD (Multiple Instruction streams, Single Data stream) et MIMD (Multiple Instruction streams, Multiple Data streams) [72], [84].

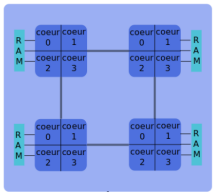


FIGURE 2.1 – Exemple de super-calculateur avec 4 nœuds.

Dans cette dernière classe, un flot multiple d'instructions provenant de plusieurs séquenceurs agit sur un flot multiple de données et cette architecture inclut toutes les configurations multi-processeurs. Dans le principe, chaque processeur traite une donnée distincte et exécute son propre flot d'instructions. Une sous-classe de machines MIMD, nommée SPMD (Single Program Multiple Data) qui représente le cas où les processeurs exécutent en parallèle le même programme sur des données différentes.

Au niveau environnement parallèle, il est possible de citer les super-calculateurs (voir Figure 2.1) qui sont composés de plusieurs milliers de processeurs, d'un grand espace de stockage sur disques et d'une très grande capacité mémoire. Ce sont des machines à mémoire distribuée qui sont composés d'un ensemble de nœuds. A l'intérieur d'un nœud, la mémoire est partagée. Ils sont généralement dédiés aux calculs numériques.

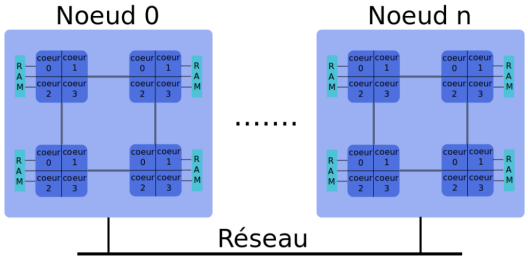


FIGURE 2.2 – Une grappe.

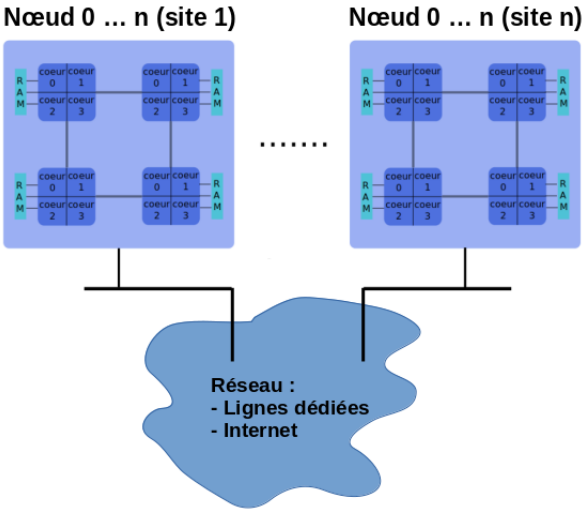


FIGURE 2.3 – Une grille.

Cependant le coût de fabrication de ces machines reste élevé et d'autres solutions ont été mises en place afin de palier cet inconvénient.

Une grappe de machines (cluster en Anglais) (voir Figure 2.2) désigne plusieurs ordinateurs reliés par un réseau d'inter-connexion permettant le partage de ressources informatiques. Cette grappe est généralement composée de machines homogènes en termes d'architecture et de système d'exploitation. Les machines communiquent entre elles en utilisant un réseau de communication.

La grille (voir Figure 2.3) désigne un ensemble important de machines, hétérogènes, réparties sur différents sites géographiquement distants. Ainsi, une grille de

calcul va permettre d'utiliser la puissance de traitement de chaque nœud afin d'offrir une puissance de calcul importante. Un réseau d'interconnexion relie chaque site et fonctionne soit avec des lignes dédiées soit à travers internet.

En France, il existe la grille de calcul Grid'5000 (voir [27] et [79]) composée de grappes distantes et hétérogènes (voir Figure 2.4).

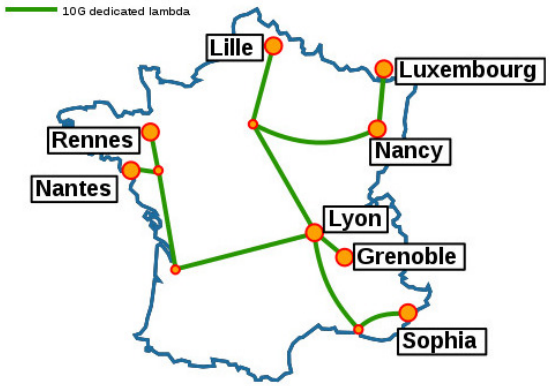


FIGURE 2.4 – La grille de calcul Grid'5000.

Ce type de grille permet d'effectuer des simulations numériques à grande échelle et donne accès à une grande quantité de ressources : 828 nœuds, 12328 cœurs répartis sur 8 sites géographiquement distants le tout connecté à un réseau fédérateur dédié de 10 Gbps. Les sites Grid'5000 sont composés de plusieurs grappes avec des architectures et des performances de calcul différentes. Ce réseau est également doté d'autres technologies comme le GPU.

Le calcul sur grille peut donc fournir une importante capacité de calcul parallèle et permettra à une application pouvant être découpée en plusieurs tâches exécutables sur plusieurs machines de la grille de réduire son temps d'exécution. Le caractère hétérogène des grilles impose des contraintes dans le codage informatique mais aussi dans les communications inter-processus car les ressources sont géographiquement distribuées.

Il est à noter que le concept de nuage ou cloud computing [49] est une évolution de la notion de grille. Ce concept prend en compte plutôt les fournisseurs de ressources qui mettent à disposition la puissance de calcul et de stockage. Le cloud computing est un modèle d'accès à des moyens délocalisés qui permet un accès omniprésent, pratique et à la demande, à un réseau partagé et à un ensemble de ressources informatiques configurables, comme par exemple : des réseaux, des serveurs, du stockage, des applications et des services qui peuvent être provisionnées et libérées avec un minimum d'administration [28] et [4].

Ce modèle est composé de cinq caractéristiques essentielles et de trois modèles de services. Les cinq caractéristiques essentielles sont :

— ressources en libre-service partout dans le monde ;
— élasticité : la puissance et la capacité de stockage peuvent être adaptées automatiquement en fonction des besoins des utilisateurs ;
— ouverture : les services sont accessibles depuis un ordinateur, un téléphone portable ou une tablette ;
— mutualisation : correspond à un partage de ressources hétérogènes (matériel, logiciel, trafic réseau) pour servir plusieurs utilisateurs à qui les ressources sont automatiquement attribuées ;
— paiement à l'usage : la quantité de service est mesurée à des fins de contrôle, d'adaptation des moyens techniques et de facturation.

Les trois principaux modèles de services sont :

— IaaS (Infrastructure as a Service) : c'est le service de plus bas niveau ; il consiste à offrir un accès à un parc informatique virtuel à l'utilisateur qui peut y installer un système d'exploitation et des applications ;
— PaaS (Platform as a Service) : c'est le service de moyen niveau ; le système d'exploitation et les outils d'infrastructure sont sous la responsabilité du fournisseur mais l'utilisateur garde le contrôle des applications et peut ajouter ses propres outils ;
— SaaS (Software as a Service) : dans ce type de service, des applications sont mises à la disposition de l'utilisateur qui n'a pas à se soucier du maintien de ces dernières ; les services de courrier électronique constituent un exemple illustrant de ce type de modèle.

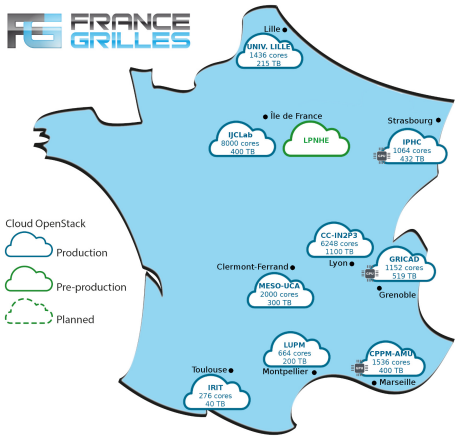


FIGURE 2.5 – Ressources Cloud de FG Cloud

En France, des simulations de calcul peuvent être réalisées sur France Grilles-Cloud (voir Figure 2.5 et [51]) qui offre aux utilisateurs des services sur le cloud permettant

le calcul, le stockage et le réseau à la demande ; l'instanciation de clusters, de services web dédiés à la communauté scientifique ainsi que des tests d'algorithmes.

Par rapport à une grille de calcul qui est intéressante dans le cadre des algorithmes itératifs parallèles asynchrone , le cloud computing dispose de sur-couches logicielles qui permettent de faire abstraction des problématiques matérielles au niveau utilisateur. Compte tenu de ces couches logicielles supplémentaires, l'utilisation des algorithmes itératifs parallèles asynchrones peut présenter un intérêt dans ce type d'environnement par rapport à leurs équivalents synchrones.

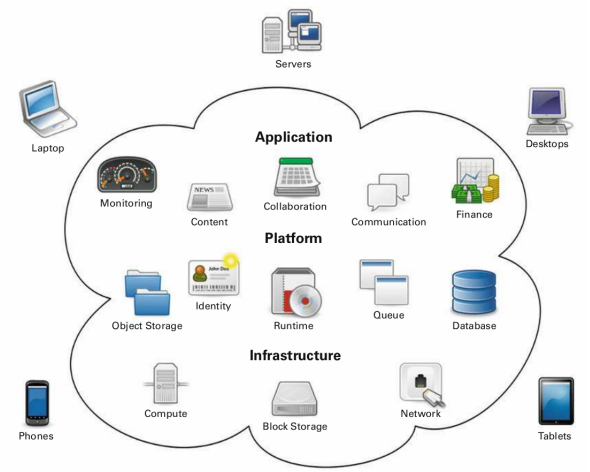


FIGURE 2.6 – Illustration du cloud computing (d'après Wikipedia).

Notons, qu'il existe deux types de machines parallèles : les machines parallèles à mémoire partagée (voir Figure 2.7) et les machines parallèles à mémoire distribuée (voir Figure 2.8).

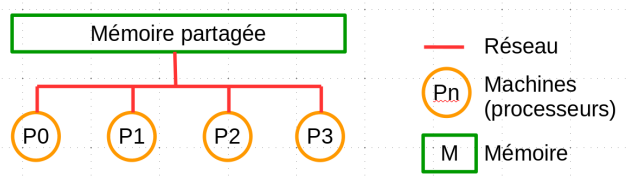


FIGURE 2.7 – Machines parallèles à mémoire partagée.

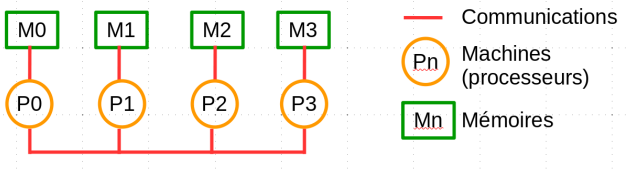


FIGURE 2.8 – Machines parallèles à mémoire distribuée.

Dans nos travaux, nous nous sommes surtout intéressé au machines parallèles à

mémoire distribuée. Nous allons donc utiliser des machines indépendantes et interconnectées possédant une mémoire locale à chaque processeur qui exécute des instructions identiques sur des données différentes (SPMD) et fonctionne avec un parallélisme par échange de messages. Le paragraphe suivant va nous permettre d'introduire la notion d'échange de message disponible dans la bibliothèque (MPI).

2.2.2 Introduction et utilisation de la bibliothèque MPI

La librairie MPI [157] prend comme modèle la classification SPMD. Ce n'est pas un langage de programmation mais un ensemble de primitives permettant des échanges de messages entre les processus comme par exemple l'envoi et la réception des données ou la synchronisation des processus. Grâce à ces primitives, il est possible de développer des applications parallèles, sans tenir compte de la technologie réseau utilisée (Ethernet Gigabit, infiniband, ...).

Nous n'allons pas développer la totalité de cette bibliothèque mais simplement mettre l'accent sur certaines primitives qui ont été nécessaires pour l'implémentation des différentes applications que nous avons étudiés dans nos travaux. Pour plus d'informations, il est possible de consulter la formation en ligne [120].

Il est possible de développer du code parallèle dans divers langages de programmation comme le langage Fortran, C, C++ ou Python ; comme compilateur on utilise gcc (voir [156]).

Listing 2.1 – exemple de compilation

```
mpicc bonjour.c -o bonjour (mpic++)
mpif90 fbonjour.f90 -o fbonjour (mpifort)
```

L'exécution d'un programme parallèle s'effectue grâce à une primitive de la bibliothèque MPI permettant de signaler que le programme qui va s'exécuter est une application parallèle ainsi que d'initialiser et de stopper les processus parallèles.

Listing 2.2 – exemple mpirun

```
mpirun -np <nb processus> -machinefile <machines> para_prog
```

La parallélisation MPI utilise plusieurs instances qui s'exécutent en même temps et chaque instance est un processus auquel est assigné un numéro unique qui représente son rang ; l'instance peut obtenir son rang lorsqu'elle est lancée.

Un programme MPI doit comprendre un fichier d'en-tête approprié (par exemple mpi.h).

Les instances doivent se coordonner grâce à l'appel d'une primitive d'initialisation :

Listing 2.3 – initialisation

```
int MPI_Init(int *argc, char **argv[]);
```

et se terminer par une primitive de nettoyage avant la fin du programme.

Listing 2.4 – finalisation

```
int MPI_Finalize(void);
```

Deux autres primitives permettent de connaître la valeur du rang d'un processus ainsi que le nombre total de processus. Dans ces primitives, l'argument comm est appelé communicateur. C'est un ensemble de processus permettant de s'envoyer entre eux des messages. Le mot-clef MPI_COMM_WORLD est un communicateur prédéfini par MPI et représente l'ensemble des processus en cours d'exécution.

Listing 2.5 – informations

```
int MPI_Comm_size(MPI_Comm comm, int *nproc);
int MPI_Comm_rank(MPI_Comm comm, int *myrank);
```

Pour les différentes implémentations, nous avons utilisé les langages C et Fortran ainsi que la version OpenMPI (voir [152]) de cette bibliothèque MPI.

2.2.3 Mécanismes de communication parallèle

La modélisation de programmation par échange de messages suit le processus suivant :

- le programme est écrit dans un langage de programmation classique ;
- les variables du programme sont privées et résident dans la mémoire locale allouée à chaque processus ;
- chaque processus exécute éventuellement des parties différentes du programme ;
- une donnée est échangée entre deux ou plusieurs processus via un appel à des sous-programmes particuliers.

Les messages échangés sont interprétés et gérés par un environnement qui peut être comparé à la téléphonie, à la télécopie, au courrier postal, à la messagerie électronique, ... Le message est envoyé à une adresse déterminée et le processus récepteur doit pouvoir classer et interpréter les messages qui lui ont été adressés.

Nous avons utilisé deux modes de communication dans nos travaux. D'une part les communications point à point qui précise que la communication a lieu entre deux processeurs (un émetteur et un destinataire) et d'autres part les communications collectives.

Dans le mode de communication point à point des primitives d'envoi ou de réception sont utilisées. Elles sont dites bloquantes si l'espace mémoire servant à la

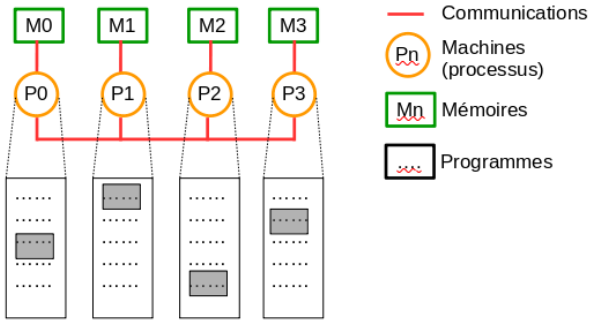


FIGURE 2.9 – Exemple d'échange de message.

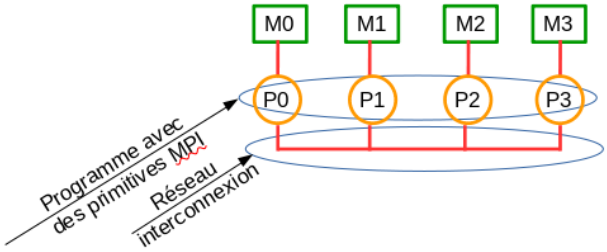


FIGURE 2.10 – Exemple de communication entre les processus.

communication peut être réutilisé immédiatement à la fin de son utilisation ou bien non bloquantes si elle rend la main immédiatement à la fin de son utilisation.

La bibliothèque MPI permet d'utiliser deux primitives afin de réaliser les communications point à point bloquantes qui seront détaillées ci-dessous. Une primitive permet d'envoyer les messages et une seconde permet de les recevoir.

L'émetteur et le récepteur du messages sont identifiés par leur rang dans le communicateur MPI_COMM_WORLD qui définit le groupe de processus utilisé. L'enveloppe d'un message est constitué du rang du processus émetteur, du rang du processus récepteur, d'une étiquette permettant d'envoyer plusieurs messages différents à un processeur destinataire, de données échangées typées (entiers, réels, ... ou types dérivés qui permettent d'utiliser des structures de données plus complexes), de la longueur de ces données et du communicateur utilisé.

Ainsi la primitive MPI_Send (respectivement MPI_ISend) permet d'envoyer un message bloquant (respectivement non bloquant) :

Listing 2.6 – Envoi bloquant

```
int MPI_Send(void* donnees, int longueur, MPI_Datatype type,
int rang_dest, int etiq, MPI_Comm comm)
```

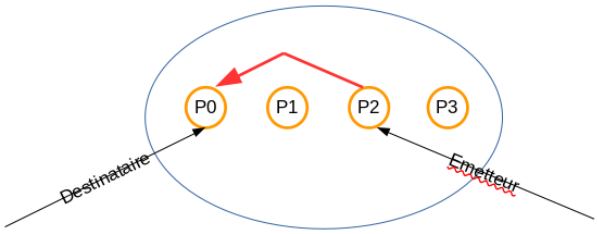


FIGURE 2.11 – Exemple de communication point à point entre les processus.

Cette commande permet à un processus d'un certain rang d'envoyer un message contenu dans *données*, de taille *longueur*, de type *type*, d'étiquette *etiq*, au processus de rang *rang_dest* dans le communicateur *comm*. Cette opération est bloquante c'est à dire l'exécution reste bloquée jusqu'à ce que le contenu du message puisse être réécrit sans risque d'écraser la valeur qui devait être envoyée.

Listing 2.7 – Envoi non bloquant

```
int MPI_Isend(void* donnees, int longueur, MPI_Datatype type,
int rang_dest, int etiq, MPI_Comm comm, MPI_Request *req)
```

Cette commande permet à un processus d'un certain rang d'envoyer un message contenu dans *données*, de taille *longueur*, de type *type*, d'étiquette *etiq*, au processus de rang *rang_dest* dans le communicateur *comm*. Cette opération n'est pas bloquante c'est à dire l'exécution n'est pas bloquée.

La primitive MPI_Recv (respectivement MPI_IRecv) permet de recevoir un message bloquant (respectivement non bloquant) :

Listing 2.8 – Réception bloquante

```
int MPI_Recv(void* donnees, int longueur, MPI_Datatype type,
int rang_emetteur, int etiq, MPI_Comm comm, MPI_Status stat)
```

Elle permet à un processus d'un certain rang de recevoir un message positionné dans *données*, de taille *longueur*, de type *type*, d'étiquette *etiq*, du processus de rang *rang_emetteur* dans le communicateur *comm*. La structure *stat* reçoit des informations sur la communication comme le rang de l'émetteur, l'étiquette, ... L'appel à cette primitive ne pourra fonctionner que si un envoi a bien été réalisé avec les mêmes informations ci-dessus (rang de l'émetteur, rang du destinataire, étiquette, communicateur). Cette opération est bloquante c'est à dire l'exécution reste bloquée jusqu'à ce que le contenu du message corresponde au message reçu.

Listing 2.9 – Réception non bloquante

```
int MPI_Irecv(void* donnees, int longueur, MPI_Datatype type,
int rang_emetteur, int etiq, MPI_Comm comm, MPI_Request *req)
```

Elle permet à un processus d'un certain rang de recevoir un message positionné dans *données*, de taille *longueur*, de type *type*, d'étiquette *etiq*, du processus de rang *rang_emetteur* dans le communicateur *comm*. La structure e*req* permet de savoir si une communication est terminée ou pas, notamment grâce aux routines MPI_Test ou MPI_Wait. Cette opération n'est pas bloquante c'est à dire que l'exécution ne reste pas bloquée.

En résumé, un envoi est dit bloquant si et seulement si au retour de l'envoi il est possible d'écrire dans le buffer d'envoi sans altérer le contenu du message et une réception est bloquante si et seulement si au retour de la réception le buffer de réception contient bien le contenu du message.

De plus, un envoi synchrone bloquant rendra la main quand le message aura été reçu par le destinataire c'est à dire lors d'un envoi synchrone, l'expéditeur renvoie au destinataire une requête d'envoi et attend que le destinataire lui réponde. Quand le destinataire débute sa réception, il attend une requête d'envoi de l'expéditeur et quand le destinataire a reçu la requête d'envoi, il répond à l'expéditeur en lui accordant l'envoi. C'est ainsi que l'expéditeur et que le destinataire sont synchronisés et le transfert de données (c'est à dire le message proprement dit) a lieu puis l'envoi et la réception sont alors terminés. Cet primitives seront utilisés dans le cadre d'algorithmes parallèles synchrones.

Une communication est dite non bloquante si et seulement si au retour de la routine, la bibliothèque de communication ne garantit pas que l'échange de message ait eu lieu. L'accès aux données n'est donc pas garanti après une communication non bloquante.

Les communications non bloquantes laissent le programme poursuivre son exécution, que la communication soit réellement réalisée ou non et présentent donc un avantage indéniable au niveau du temps d'exécution puisqu'un processeur peut travailler tout en envoyant des données ; ainsi on élimine les temps d'inactivité. Les deux primitives MPI_ISend et MPI_IRecv vont donc être primordiale pour l'implémentation du calcul numérique parallèle asynchrone. En effet, comme nous le verrons plus loin, s'affranchir des temps d'attentes entre les processeurs va permettre un gain de temps non négligeable dans les cas où il y a beaucoup de synchronisations.

Les communications collectives permettent de faire en une seule opération plusieurs communications point à point et concernent toujours tous les processus du communicateur considéré. En effet, pour chaque processus, l'appel se terminera lorsque la participation de celui-ci à l'opération collective est terminée c'est à dire quand le buffer peut être modifiée.

Nous pouvons citer la primitive MPI_BARRIER qui permet une synchronisation globale des processus, la primitive MPI_BCAST qui permet la diffusion globale de données vers tous les processeurs, les primitives MPI_SCATTER et MPI_GATHER qui permettent la diffusion et la collecte de données réparties, et enfin la primitive

MPI_ALLREDUCE qui permet en plus de la gestion des communications, d'effectuer des opérations sur les données (somme, produit, maximum, minimum, etc.).

2.2.4 Mesures de performances en asynchrone

Les mesures de performance classique en algorithmique parallèle qui sont l'accélération, l'efficacité et la loi d'Amdhal sont plus difficilement utilisables sur une grille de calcul, sur cloud ou sur machines peer-to-peer. En effet, sur une grappe de processeurs homogènes on peut calculer l'accélération par comparaison au temps séquentiel quand c'est possible et ainsi déduire l'efficacité. Cette homogénéité permet effectivement d'obtenir un temps séquentiel commun aux machines.

Par contre définir l'efficacité sur une grille est plus complexe lorsque des grappes distantes et hétérogènes sont utilisées. Cela nécessite des choix empiriques comme par exemple comparer avec le temps séquentiel obtenu sur le processeur le plus lent, ou bien le plus rapide ou alors par rapport à un temps moyen obtenu sur tous les processeurs de calcul. De plus, lorsqu'on effectue une simulation sur une grille d'un problème de grande taille il est impossible de l'implémenter sur une seule machine et il est difficile de déterminer le temps séquentiel.

Pour comparer l'efficacité d'un algorithme exécuté en mode asynchrone par rapport au même algorithme exécuté en mode synchrone, nous utilisons un nombre τ qui permet de comparer les temps de calcul en mode synchrone et en mode asynchrone. Ce paramètre mesure le rapport du temps écoulé entre les méthodes synchrones et asynchrones. Il permet de mesurer le gain éventuel de l'algorithme asynchrone parallèle étudié.

τ est donc défini par

$$\tau = \frac{\text{temps d'exécution synchrone}}{\text{temps d'exécution asynchrone}} \tag{2.2.1}$$

Dans le cas où le nombre τ est supérieur à 1, l'algorithme parallèle asynchrone est exécuté avec un temps plus petit que son homologue synchrone.

De plus, si le nombre τ est très élevé, l'algorithme parallèle asynchrone a des performances très intéressantes en terme de temps de restitution des calculs.

2.2.5 Terminaison des algorithmes itératifs parallèles asynchrones

Le problème de terminaison des itérations parallèles asynchrones est un problème extrêmement difficile à coder car il relève à la fois des mathématiques appliquées et également de l'informatique. Par ailleurs sur le plan informatique on doit tenir compte

de la spécificité des architectures des machines multiprocesseurs, en particulier sur des systèmes distribués où les communications s'effectuent par passage de messages et un contexte où les processeurs possèdent uniquement des informations locales. Les travaux réalisés dans le contexte informatique ont permis de fournir une preuve formelle de la validité des critères d'arrêt. Nous pouvons distinguer une approche centralisée et une approche décentralisée que nous allons expliciter brièvement ci-dessous.

Approche centralisée

Dans l'approche centralisée de détection de convergence, un processeur reçoit l'état des autres processeurs et détecte la convergence globale lorsque tous les processus ont convergé localement.

Dans [26], les auteurs considèrent que chaque donnée communiquée sur un lien est correctement reçue avec un retard fini mais cependant non spécifié. De ce fait, la méthode de terminaison des algorithmes itératifs asynchrones est basée sur la décomposition du problème en deux parties distinctes : l'algorithme itératif est modifié de telle sorte qu'il se termine en un temps fini et qu'il converge vers un point fixe proche de la solution du problème et une procédure de détection informatique de terminaison est appliquée. Il est à noter que l'algorithme asynchrone est modifié et qu'il est différent de l'algorithme asynchrone classique dans la mesure où la valeur d'une ou plusieurs composantes du vecteur itéré ne changent pas durant un temps fini, contrairement à ce que produit l'algorithme de calcul où les composantes peuvent changer à chaque relaxation.

Les auteurs introduisent une nouvelle application de point fixe définie soit par la valeur réactualisée déduite de l'algorithme de calcul si la valeur de la composante a changé, soit par l'ancienne valeur si la réactualisation n'est pas significative. Ce contexte a conduit les auteurs à modifier l'algorithme itératif de la manière suivante : si la mise à jour d'une composante du vecteur itéré ne change pas significativement sa valeur alors la nouvelle valeur réactualisée n'est pas modifiée et n'est pas communiquée aux autres processeurs. L'arrêt de l'algorithme itératif modifié se produit lorsqu'une mise à jour ne modifie pas la valeur des composantes du vecteur itéré quel que soit le processeur (c'est-à-dire lorsque toutes les conditions de terminaison locales sont satisfaites) et qu'aucun message n'est en transit dans le réseau de communication. Ainsi, en ce qui concerne la non-modification des composantes du vecteur itéré toutes les conditions de terminaison locales sont satisfaites. Plusieurs procédures de détection de terminaison de l'algorithme itératif asynchrone modifié peuvent alors être utilisées comme la procédure de Dijkstra et Scholten [44] qui est basée sur l'accusé de réception de tous les messages et la génération d'activité hiérarchisé. Les processeurs communiquent deux types d'information qui sont la nouvelle valeur des composantes du vecteur itéré et l'accusé de réception des messages contenant une

nouvelle valeur.

Une autre procédure de terminaison, correspondant à l'algorithme de snapshot (voir [29]) basée sur la production d'une image instantanée de l'état global du système obtenue par la production de messages de marquage et de l'enregistrement des états des liens et des états des processeurs lorsque les messages de marquage sont délivrés. Les états enregistrés dans un snapshot ne correspondent pas nécessairement à l'état global réel du système à un instant donné. Cependant les informations contenues dans un snapshot sont suffisantes pour détecter certaines propriétés de l'état global du système, en particulier la terminaison.

Une autre méthode de terminaison particulièrement intéressante a été proposée dans [129] ; elle consiste à définir un modèle décrivant les itérations asynchrones qui est légèrement modifié. Le résultat de chaque nouvelle mise à jour d'une composante du vecteur itéré est pris en compte et communiqué aux autres processeurs s'il est différent de la dernière valeur calculée. En outre des requêtes sont envoyées à tous les processeurs chaque fois qu'une condition de terminaison locale est satisfaite. Un processeur effectue les calculs et transmet les messages et les requêtes aux autres processeurs aussi longtemps que sa condition de terminaison locale n'est pas satisfaite ou qu'il reçoit des requêtes provenant d'autres processeurs. La terminaison intervient lorsque tous les processeurs ont satisfait leur condition locale de terminaison et qu'aucun message relatif à une requête ou à un résultat d'une réactualisation n'est en transit dans le système. Les auteurs ont donné une preuve formelle de validité de cet algorithme de terminaison et le principal avantage de cette méthode est qu'elle peut être appliquée avec succès à une plus large classe d'algorithmes itératifs. Par contre, son principal inconvénient est de nécessiter un très grand nombre de communications et d'imposer des hypothèses restrictives comme, par exemple, des communications ordonnées.

Citons également les travaux des auteurs dans [13] qui ont étudié une approche centralisée de terminaison des méthodes parallèles asynchrones. La difficulté de détection de la convergence est accentuée par celle d'obtenir à tout moment une image correcte de l'état global du processus itératif. En effet les techniques utilisées en informatique distribuée pour récupérer ces informations sont centralisées et synchrones et ce type de de détection n'est pas adaptée aux systèmes distribués de grande taille et/ou distants, ni aux algorithmes itératifs asynchrones.

De plus, la centralisation de ces méthodes de terminaison peut générer le problème classique de goulots d'étranglement. En effet, dans un algorithme centralisé classique, tous les processeurs communiquent directement leurs informations à un processeur maître. Toutefois, ce schéma de communication, impliquant qu'une machine puisse être directement contactée par toutes les autres, n'est pas possible dans tous les systèmes parallèles, en particulier dans les clusteurs distribués où chaque site peut avoir des politiques d'accès restreint pour des raisons de sécurité. Dans la plupart des cas, une seule machine d'un cluster donné n'est accessible de l'extérieur.

Afin de contourner ce problème, un transfert explicite de message peut être effectué à partir de n'importe quel nœud du système vers le nœud central. Une telle méthode présente l'avantage de n'impliquer que des communications entre des nœuds voisins et est bien adaptée aux systèmes de communication hiérarchiques que l'on peut trouver dans les clusters distribuées. Malheureusement, ce dernier schéma implique davantage de communications entre les processeurs, et ce supplément de communications ralentit, indirectement, l'aboutissement du processus itératif lui-même. En outre, il induit également des retards plus importants vers le nœud central.

Approche décentralisée

Pour les méthodes parallèles asynchrones, dans [31] et [36], les auteurs ont utilisé comme critère d'arrêt un algorithme décentralisé où la détection de la convergence asynchrone est réalisée par la circulation d'un jeton. Afin de participer à la détection de la convergence chaque processeur met à jour les composantes du sous-vecteur qu'il doit relaxer et calcule la norme du résidu attachée à ce bloc.

La détection de la convergence est alors effectuée au moyen d'un algorithme de « snapshot » (paragraphe 8.2 de [26], [29]). Il y a convergence lorsqu'un prédicat donné sur un état global est vrai. Un prédicat classique correspond au fait que le vecteur itéré généré par l'algorithme itératif asynchrone est suffisamment proche de la solution du problème (p. 580 de [26]) ; de plus, pour chaque processus, et pour éviter une détection prématurée de convergence, la norme du résidu local doit rester sous un seuil donné après deux mises à jour successives des composantes [26] et [109]. Il est à noter qu'en raison de la détection et de l'arrêt global des processus, la mise en œuvre de chaque variante d'une méthode parallèle asynchrone est alors plus complexe que celle d'une méthode parallèle synchrone ; en effet, dans ce cas asynchrone on doit utiliser des communications point à point entre deux processus parallèles bien identifiés et permettre à n'importe quel processeur de continuer ses propres calculs sans avoir à attendre l'achèvement de toute opération d'envoi ou de réception en cours. On supprime ainsi les temps morts d'inactivité dus aux synchronisations.

Une approche permettant plus de flexibilité est proposée leurs auteurs dans [13]. Elle n'utilise pas de circulation de jeton. De plus cet algorithme de détection de convergence permet une implémentation complètement décentralisée à la fois pour les algorithmes itératifs asynchrones mais également pour les algorithmes itératifs synchrones, avec, dans ce dernier cas, quelques modifications mineures. En ce qui concerne la détection de convergence, la difficulté majeure réside dans l'établissement de preuve que l'algorithme proposé ne détecte pas la convergence prématurément. En effet, dans les méthodes asynchrones, l'introduction de retards inhérents à la modélisation de l'algorithme pourrait conduire à une fausse détection de convergence. Cette situation pourrait se produire généralement dans des contextes d'utilisation de machines hétérogènes, par exemple lorsqu'un processeur effectue des mises à jour

sur le bloc qu'il doit traiter alors qu'un autre processeur plus lent effectue des mises à jour retardées, par exemple dans le cas de l'utilisation de méthodes avec recouvrement. Cette difficulté est encore accrue avec l'utilisation de processeurs distants où le rapport communication/calcul peut être important. En conséquence, le principe de l'algorithme de détection de terminaison décentralisée repose sur deux étapes classiques distinctes. La première étape consiste à détecter la convergence locale sur chaque processeur, tandis que la seconde étape est relative à la détection de la convergence globale.

L'étape de détection de la convergence locale est assez similaire à celle utilisée dans le mode synchrone. Généralement pour situer l'état actuel du processus itératif par rapport au point fixe à déterminer on utilise la norme du résidu pour stopper ce processus itératif local. Ainsi, l'itération locale est terminée lorsque la norme du résidu est inférieure à un seuil donné. Néanmoins, on peut noter que, dans tous les algorithmes itératifs et pas spécialement dans les algorithmes asynchrones, une fausse détection de la convergence globale peut se produire si aucune précaution n'est prise. L'heuristique commune pour la détection d'une convergence locale définitive est alors de supposer que cette convergence locale est atteinte lorsque la norme du résidu est inférieure à un seuil donné pour un nombre donné d'itérations successives. Cette valeur requise du nombre d'itérations successives existe et est finie puisque le processus itératif converge ; le résultat de [109] permet de donner une estimation de la valeur du nombre d'itérations.

La convergence globale se produira lorsque tous les processus locaux auront convergé. Cependant, bien qu'en mode asynchrone la détection de la convergence locale ne présente pas de difficulté majeure, par contre, en raison de la difficulté de représentation d'un état global du processus itératif, la détection de la convergence globale n'est pas évidente. La détection de terminaison d'algorithme décentralisé est alors basée sur un schéma similaire à celui du protocole d'élection. Ce protocole consiste à désigner dynamiquement un processeur pour effectuer une tâche donnée, à savoir, dans notre cas, la détection de la convergence globale. Toutefois, dans le contexte spécifique de la convergence globale, la détection de terminaison nécessite certaines adaptations spécifiques qui impliquent l'utilisation d'un diagramme en arbre.

Un message informe le destinataire que tous les processeurs situés dans la sous-arborescence dépendant de son expéditeur ont atteint la convergence locale. Ainsi, sur chaque processeur, l'algorithme considère le nombre de voisins dans l'arbre desquels aucun message de convergence n'a déjà été reçu. Ainsi, un nœud détectera la convergence globale lorsqu'il a reçu le message de convergence de tous ses voisins et est lui-même en convergence locale. L'exactitude de cette procédure de détection de la convergence globale est prouvée dans le contexte où des techniques de contraction sont utilisées [13] ; dans d'autres cas, le processus de détection de convergence globale est toujours valable, mais une étape de vérification supplémentaire est nécessaire

pour s'assurer que le système était globalement en phase de terminaison au moment de la détection.

Dans [71], les auteurs présentent de nouveaux résultats concernant la détection de convergence pour les itérations parallèles asynchrones. Cette étude s'appuie sur l'estimation consistante du résidu en utilisant des séquences de vecteurs générées par chaque processeur. L'algorithme de « snapshot » constitue un moyen de construire l'état global de n'importe quel système distribué par enregistrement de l'état des processus et des canaux de communication ; il permet dans la suite de proposer deux extensions possibles pour détecter la terminaison des itérations asynchrones dans les environnements de communications en file du type FIFO (First In First Out, soit premier entré, premier sorti). Les auteurs proposent d'étendre les deux protocoles précédemment envisagés et de prendre en compte les messages non-FIFO ; ainsi les deux nouveaux protocoles proposés n'introduisent aucune phase de coordination. Puis, en considérant un modèle de communications où au moins les messages de calcul sont ordonnés en file, les auteurs proposent une autre méthode sans « snapshot », reposant uniquement sur les données de calcul. L'idée principale est d'enregistrer l'état d'un canal lorsque deux messages reçus successivement sont très proches. Les processus ont donc besoin de deux fois plus de mémoire pour les messages reçus. Enfin, ces auteurs introduisent un environnement de communication dans lequel le message non-FIFO délivré peut être caractérisé, en raison des exigences de niveau de performance sur la plateforme de calcul. Ils en déduisent donc deux nouveaux autres protocoles qui permettent de construire approximativement un vecteur d'itération global. Grâce à des hypothèses appropriées, une analyse formelle est effectuée lorsque l'espace est normé par n'importe quelle norme, afin d'obtenir une borne d'erreur de la valeur calculée du résidu. Des expérimentations sur divers super-ordinateurs montrent clairement que le résidu est estimé avec une bonne précision. Par rapport aux précédentes contributions développées par d'autres auteurs, la présente étude montre qu'une seule opération de réduction à chaque itération (au lieu de deux dans les autres contributions) est suffisante pour le calcul du résidu ; ainsi, cette étude minimise à la fois les délais de détection de convergence et les coûts de communication.

Nous détaillerons nos choix de méthodes de terminaison de nos algorithmes dans le paragraphe 3.1.3.

2.2.6 Principe d'implémentation des algorithmes asynchrones

Ce paragraphe est consacré à la présentation du principe d'implémentation des algorithmes parallèles asynchrones. Comme nous le verrons plus tard, l'implémentation de ces algorithmes parallèles a été réalisée sur différentes plateformes. La méthode de parallélisation utilisée correspond à une méthode de décomposition de domaine associée à un partitionnement en grands blocs des matrices, ces blocs étant assignés aux différents processeurs. Le processus de calcul parallèle implique donc la décom-

position du domaine et du maillage. Le principe de mise en œuvre des méthodes itératives parallèles asynchrones et synchrones peut être résumé dans le schéma de l'algorithme 2.

Algorithme 2 : principe d'implémentation d'une méthode de sous-domaines

```
1 tantque non convergence
2 |  pour chaque bloc faire
3 |  |  Effectuer une communication des valeurs frontières des blocs
4 |  |  Effectuer un algorithme de relaxation par blocs utilisant un solveur
  |  |   local
5 |  fin
6 fin
```

Méthode sans recouvrement

La méthode sans recouvrement correspond à une méthode de relaxation par blocs classique où les valeurs des composantes provenant des processus traités par les autres processeurs associés sont communiquées de manière synchrone ou asynchrone selon le cas ; par conséquent, en utilisant les outils de parallélisation, l'implémentation de ces méthodes parallèles sans recouvrement ne présente aucune difficulté majeure. Ainsi, afin de calculer la solution du problème global, on est amené à résoudre des séries de sous-problèmes plus petits sur chaque processeur ; à noter que du fait de la résolution de sous-problèmes de plus petite taille, il y a moins de propagation d'erreurs d'arrondi et on obtient ainsi une meilleure précision. De plus plusieurs blocs adjacents de la décomposition naturelle par blocs sont assignés à chaque processeur afin de mettre en œuvre une stratégie proche d'une stratégie multiplicative ; on obtient alors une décomposition en grands blocs. Afin d'obtenir une vitesse de convergence plus élevée de l'algorithme, chaque processeur traite ses propres blocs, numérotés selon une numérotation rouge-noir ou lexicographique ; il faut noter que la numérotation rouge-noir est plus adaptée à l'utilisation du calcul parallèle et, dans ce contexte de numérotation, on peut adapter simplement les démonstrations pour obtenir des résultats de convergence (voir [34]).

Ainsi, sur chaque grand bloc associé à chaque processus, on peut implémenter une méthode classique et séquentielle de relaxation par blocs - ainsi qu'un autre type d'algorithme numérique - afin de résoudre chaque sous-problème ; pour un problème de diffusion ou de convection- diffusion ce type d'algorithme est mis en œuvre en résolvant chaque bloc tridiagonal, par la méthode d'élimination de Gauss adaptée aux matrices tridiagonales. Afin de minimiser le temps de calcul, la méthode de relaxation des blocs est optimisée par l'élimination des séquences de code redondantes ; la décomposition LU de chaque bloc diagonal n'est effectuée qu'une seule fois dans la phase d'initialisation ce qui permet de diminuer le temps d'exécution. La détec-

tion de convergence des méthodes asynchrones parallèles considérées est effectuée conformément à ce qui a été présentée à la section 2.2.5 (voir aussi [31] et [36]).

Il est également possible d'implémenter diverses méthodes numériques, par exemple la méthode de relaxation ponctuelle ou l'algorithme du gradient conjugué préconditionné. À noter que pour le découpage du problème en sous-problèmes on utilise un système d'indices ou de pointeurs qui définissent le début et la fin de chaque bloc.

L'algorithme de simulation parallèle est basé sur le paradigme maître/esclave qui est une approche couramment utilisée pour les applications parallèles et distribuées. Dans notre cas, un processus maître contrôle la distribution du travail réparti sur un ensemble de processus esclaves. D'où le schéma de l'algorithme 3 décrivant le principe d'implémentation.

Algorithme 3 : Algorithme général

```
si maître alors
    calculer la matrice de discrétisation et le second membre
    envoyer les données d'entrée aux esclaves
    mettre en œuvre l'algorithme parallèle SPMD pour résoudre le système
     algébrique
    recevoir les données de sortie des esclaves
sinon
    recevoir les données d'entrée du maître
    mettre en œuvre l'algorithme parallèle SPMD pour résoudre le système
     algébrique
    envoyer les données de sortie au maître
fin
```

Cet algorithme est utilisé pour résoudre le système algébrique. Le calcul de la matrice et du second membre peut être effectué de manière séquentielle, car cette partie n'est pas très consommatrice de temps de calcul et, sauf cas particulier, est effectuée une seule fois. Ainsi, seuls les calculs intensifs sont parallélisés.

Méthode avec recouvrement

Dans le cas de la parallélisation de la méthode alternée de Schwarz, où les sous-domaines se recouvrent, la mise en œuvre est semblable exceptée que le système d'indices ou de pointeur permettant le découpage en grands blocs doit tenir compte du recouvrement entre les sous-domaines. Le domaine est divisé soit en parallélépipèdes soit en sous-domaines de forme quelconque se chevauchant. Des valeurs minimales de recouvrement sont choisies (environ un ou deux points de maille). On doit aussi définir les conditions de transmission entre deux sous domaines adjacents. La description de la mise en œuvre de la méthode alternée de Schwarz classique, parallèle

synchrone et asynchrone, est donnée par l'algorithme 4.

Algorithme 4 : Méthode alternée de Schwarz

```
1 tantque non convergence globale
2 |  pour chaque sous-domaine assigné à un processeur faire
3 |  |  si convergence locale non atteinte alors
4 |  |  |  recevoir les dernières valeurs aux frontières
5 |  |  |  résoudre les sous - systèmes
6 |  |  |  envoyer les valeurs aux frontières aux processus voisins
7 |  |  fin
8 |  fin
9 fin
```

Les sous-domaines sont affectés à chaque processeur en suivant par exemple une numérotation rouge-noire ou lexicographique. La solution du problème cible est calculée comme précédemment en implémentant soit la méthode de Gauss-Seidel par blocs qui a l'avantage d'avoir un comportement multiplicatif ou une autre méthode numérique. Du point de vue programmation, lors de l'implémentation des méthodes parallèles asynchrones il est nécessaire de permettre un chevauchement complet entre les phases de communications et de calcul durant le traitement parallèle, contrairement à ce qu'il se passe lors de l'exécution d'algorithmes parallèles synchrones où, par exemple, un processeur doit attendre les mises à jour d'un autre processeur qui n'a pas terminé ses propres calculs. Dans le cas où de nombreuses synchronisations sont nécessaires, et si en outre les communications sont lentes, une implémentation de méthodes parallèles asynchrones pourrait comme on le verra au chapitre 3 réduire considérablement le temps de calcul. En mode de communication asynchrone il est préférable de mettre en œuvre des requêtes persistantes MPI [31] et [36], plutôt que d'utiliser des communications collectives ; dans le cas d'utilisation de communications persistantes, les tampons de messages et les processus communicants sont spécifiés une fois pour toutes avant de commencer une itération. De plus il est recommandé d'utiliser également des requêtes de réception non bloquantes entre processeurs. La terminaison du processus itératif est assurée par l'ajout d'une opération de snapshot partiel selon le principe présenté au paragraphe 2.2.5.

2.3 Conclusion

Ce chapitre a permis de poser le cadre informatique qui sera utilisé dans les implémentations d'applications spécifiques aux algorithmes parallèles asynchrones qui seront présentés dans le chapitre suivant.

Chapitre 3

Implémentation des algorithmes parallèles asynchrones

Dans les chapitres précédents, pour résoudre de grands systèmes algébriques par des algorithmes parallèles, nous avons présenté des méthodes de communications bloquantes et non bloquantes. Ces possibilités de communication permettent d'envisager leur utilisation en calcul numérique parallèle. En effet, classiquement le mode de communication synchrone est naturellement privilégié dans la résolution de problèmes d'analyse numérique. Le synchronisme est intéressant pour pouvoir accélérer les calculs en utilisant la puissance des architectures présentées au paragraphe 2.2.1 du chapitre 2. En effet, il permet de synchroniser les machines afin de recevoir des données par exemple tout en gardant un mode déterministe dans les communications. L'objectif de l'asynchronisme est de minimiser les contraintes de synchronisations et les délais de communications. Un nombre élevé de synchronisations induit des attentes pouvant pénaliser les temps de restitution des travaux informatiques et provoquer une sous-utilisation des processeurs obligatoirement mis en attente. Il est donc intéressant de recenser dans quels types de situations le parallélisme asynchrone est le plus pertinent à mettre en œuvre que le parallélisme synchrone. Nos expérimentations ont été majoritairement effectuées sur la plate-forme Grid'5000 (voir figure 2.4) et celle de FG Cloud (voir figure 2.5) en utilisant des grappes distantes afin d'augmenter le nombre de nœuds disponibles et donc d'accélérer les temps de calcul. Nous avons utilisé la classification SPMD car nous exécutons le même code sur les processeurs d'une machine MIMD et les algorithmes asynchrones sont particulièrement bien adaptés aux architectures dont les communications entre les nœuds éloignés géographiquement sont lentes, malgré la présence d'un réseau haut débit entre les nœuds.

Ces méthodes parallèles asynchrones ont permis de résoudre de nombreux problèmes, modélisés par différentes formulations comme les équations aux dérivées partielles fortement non linéaires, parmi lesquels nous pouvons citer les problèmes de

diffusion non linéaires et de convection-diffusion non linéaires, le problème d'obstacle intervenant en mathématique financière et en mécanique mais aussi l'équation d'Hamilton-Jacobi-Bellmann intervenant en traitement d'image, la résolution de grands systèmes linéaires creux dans le domaine de la modélisation à l'aide de chaînes de Markov et la résolution de grands systèmes linéaires ou non linéaires par les méthodes de multisplitting. Les méthodes asynchrones peuvent aussi s'appliquer à la résolution de problèmes algébro-différentiels, à la programmation dynamique ainsi qu'à bien d'autres types de problème. Pour plus d'informations, nous renvoyons à la liste de référence bibliographique.

Plus particulièrement, dans le cadre de nos travaux, les applications traitées ont par exemple concerné :

- des équations aux dérivées partielles fortement non-linéaires par la méthode de multi-décomposition - voir [70] et [69],
- des problèmes couplés en biologie concernant la séparation de protéines par électrophorèse modélisé par l'équation de Navier-Stokes couplée à une équation de convection-diffusion et à une équation de diffusion - voir [35],
- des problèmes formulés sous forme complémentaire appliqué à l'équation d'Hamilton-Jacobi-Bellman intervenant par exemple en traitement d'images ou aussi des problèmes avec contraintes intervenant en finance et modélisé par l'équation de Black Scholes - voir [67] et [34],
- des problèmes d'interaction fluide-structure modélisé par l'équation de Navier-Stokes couplée à l'équation de Navier - voir [119],
- un problème avec contrainte unilatérale intervenant en mécanique des fluides - voir [36],
- un problème de solidification de l'acier modélisé par une équation de la chaleur prenant en compte des phénomènes de rayonnement à la frontière - voir [68],
- un problème intervenant en mathématiques financières .

Ce chapitre va permettre d'analyser expérimentalement les comportements des algorithmes asynchrones à travers les simulations effectuées. Ces dernières prennent en compte d'une part l'implémentation des méthodes de calculs parallèles synchrones et asynchrones en particulier sur les tests d'arrêt des itérations et d'autre part les enseignements que l'on peut tirer du comportement des algorithmes parallèles considérés en liaison avec l'architecture des machines utilisées et la rapidité du réseau d'inter-connexions.

3.1 Algorithmes et principes de programmation

Cette section va permettre de décrire un modèle d'algorithme général utilisé dans les implémentations ainsi que des principes de programmation.

3.1.1 Modèle d'algorithme parallèle général

Algorithme 5 : Algorithme parallèle général

```
1  début
2      Déclarer les paramètres physiques, les conditions initiales, les
         initialisations
3      Choisir une méthode de sous-domaine avec ou sans recouvrement
4      Lancer une horloge pour mesurer le temps de calcul
5      pour Chaque pas de temps faire
6          Calculer ou initialiser des paramètres
7          pour Chaque sous-domaines faire
8              Calculer le second membre
9              tantque Non convergence
10                 pour Chaque sous-domaine assigné au processeur faire
11                     Choisir un calcul synchrone ou asynchrone
12                     Résoudre tous les systèmes linéaires
13                 fin
14                 Recevoir les valeurs des autres processeurs
15                 et les communiquer aux autres
16                 Envoyer l'indicateur de convergence locale au gestionnaire de
                     terminaison
17                 Recevoir l'indicateur de convergence globale au gestionnaire de
                     terminaison
18                 Barrière de synchronisation si nécessaire
19             fin
20             Le dernier processeur envoie les dernières valeurs
21             du sous-domaine actuel au premier processeur
22             Le premier processeur reçoit les dernières valeurs
23             du sous-domaine précédent du dernier processeur
24             Barrière de synchronisation si nécessaire
25         fin
26         Barrière de synchronisation si nécessaire
27     fin
28     Envoyer un message d'arrêt au processeur gestionnaire de terminaison
29     Arrêter l'horloge et affichage du temps écoulé depuis le début du calcul
         et/ou mesure du temps des communications
30 fin
```

Le principe de l'algorithme général 5 a été utilisé dans plusieurs codes développés pour résoudre divers problèmes. Les parties développées dans cet algorithme sont quasiment identiques à part certaines phases plus spécifiques. Il est à noter que pour les implémentations des algorithmes parallèles asynchrones, il a été important de

coder les méthodes avec et sans recouvrement (voir aussi 2.2.6).

L'algorithme 5 fournit donc un cadre général d'implémentation en considérant des variantes en fonctions des problèmes considérés. Par exemple, dans le cas des algorithme non évolutif en temps, une boucle à chaque pas de temps n'est pas nécessaire.

3.1.2 Cohérences des calculs

Pour communiquer grâce à des messages, MPI utilise des primitives d'envoi et de réception qui sont bloquantes si l'espace mémoire servant à la communication peut être réutilisé immédiatement à la fin de son utilisation. Dans ce cas, l'attente de la terminaison d'une requête est une opération pendant laquelle le processus est inactif et à la fin de la requête, le canal de transmission est réutilisé pour transmettre à nouveau des messages.

De plus MPI peut utiliser des communications non bloquantes qui rendent la main immédiatement à la fin de leur utilisation. Dans ce cas, le test de terminaison de requêtes se substitue à l'attente. Ces communications non bloquantes permettent de réutiliser l'espace mémoire servant à la communication et laissent le programme poursuivre son exécution, que la communication soit réellement réalisée ou non, et présentent un avantage indéniable au niveau du temps d'exécution puisqu'un processeur peut travailler tout en envoyant des données ; elles permettent d'effectuer des communications asynchrones.

Algorithme 6 : Implémentation d'un schéma en temps implicite ou explicite

```
1  début
2      pour chaque pas de temps faire
3          si convergence locale non atteinte alors
4              Calculer le second membre du schéma en temps
5              Résoudre un système algébrique sur chaque cluster
6              Envoyer les valeurs aux frontières aux processus voisins
7          fin
8          Barrière de synchronisation
9      fin
10 fin
```

Dans le cas de la résolution d'un problème d'évolution par un schéma de calcul explicite, implicite ou semi-implicite, il est nécessaire, pour assurer la cohérence du calcul de mettre en place à chaque pas de temps une barrière de synchronisation par la fonction collective MPI_BARRIER qui permet de bloquer les processus jusqu'à ce que le dernier soit arrivé à cette barrière. L'algorithme 6 montre le fonctionnement

de cette méthode.

3.1.3 Critère d'arrêt centralisé et décentralisé

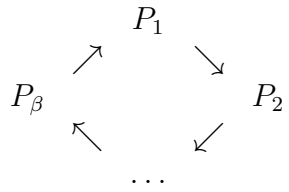
Le test d'arrêt décentralisé

La détection de la convergence des itérations parallèles synchrones et asynchrones est basée sur le principe suivant [26] : un critère de convergence locale est évalué au niveau de chaque processus, de telle sorte que la convergence globale soit atteinte lorsque la convergence locale est réalisée par tous les processus. Les critères de convergence locaux sont par exemple, la norme du résidu pour un bloc-composante ou bien la différence des valeurs entre deux bloc-composantes successives est inférieure à une tolérance donnée.

La détection de la convergence globale des algorithmes synchrones ne pose aucun problème. Il suffit d'utiliser l'opération collective MPI_Allreduce (voir aussi [86]). Cette primitive MPI_Allreduce MPI permet de réaliser des réductions sur des données réparties sur un ensemble de processus. Une réduction est une opération appliquée à un ensemble d'éléments pour en obtenir une seule valeur. Un exemple typique est constitué lorsqu'on veut déterminer la somme des éléments d'un vecteur ou le maximum des valeurs absolues de ses composantes pour calculer la norme uniforme. Dans le cas de cette primitive, comme plusieurs éléments sont concernés par processus, la primitive de réduction est appliquée à chacun d'entre eux (par exemple à tous les éléments d'un vecteur).

Parmi les méthodes de terminaison des itérations parallèles asynchrones disponibles dans la littérature [26, 19, 23], nous avons choisi d'implémenter un algorithme de terminaison utilisant le *snapshot* Chandy et Lamport [29]. Le principe de cet algorithme est donné dans [26].

Le test d'arrêt [30] consiste à faire circuler un *jeton* constitué d'un petit message qui contient les données permettant d'évaluer une condition parmi les processus.



Ce jeton est initialement émis par un des processus choisi arbitrairement et chaque processus doit le retransmettre en utilisant des opérations non-bloquantes.

Le jeton contient les données suivantes :
— un booléen indiquant si tous les processus ont localement convergé ;
— un compteur contenant la différence entre le nombre de messages envoyés et reçus par tous les processus ;

Chaque processus doit compter le nombre de requêtes d'envoi et de réception qui se sont terminées depuis le dernier passage du jeton. La convergence globale est détectée si le jeton indique que tous les processus ont convergé localement et si la réception de tous les messages envoyés est confirmée.

Remarque 1. *Les envois et les réceptions du jeton et du message d'arrêt sont effectués dans le même communicateur (voir 2.2.2) que les messages. Le paramètre* etiq *présenté au chapitre 2, paragraphe 2.2.3 décrit les procédures de communication permet alors de distinguer différents types de message.*

Le test d'arrêt centralisé

Un critère d'arrêt centralisé et efficace peut-être implémenté en utilisant un processeur supplémentaire de gestion de la terminaison dédié à cette tâche. Nous préférons utiliser un autre processeur pour éviter les délais de communication entre les processeurs de calcul. Nous avons utilisé le critère d'arrêt proposé par [13] basé sur l'utilisation d'une méthode centralisée. Le processeur gérant la convergence centralise les convergences locales de tous les processeurs de calcul. Chaque processeur détermine sa convergence locale ; une convergence locale correspond au fait que la norme uniforme de la différence entre deux mises à jour (ou plus) successives est inférieure à un seuil donné fixé. Ainsi, pour un pas courant, lorsqu'une convergence locale est détectée par un processeur de calcul, un message est envoyé au processeur gestionnaire de convergence globale ; néanmoins, cette convergence locale étant atteinte, le processus continue à itérer afin d'éviter une fausse détection de convergence locale. S'il y a fausse détection de convergence, un message d'annulation de convergence est envoyé. Lorsque le processeur gestionnaire de convergence détecte la convergence globale, il envoie un message d'arrêt aux processeurs de calcul.

La comparaison de l'efficacité du critère d'arrêt proposé dans [26] basé sur l'utilisation d'une méthode décentralisée et celle du critère centralisé présenté dans [13] a montré que le nombre de relaxations n'est pas le même et nous avons finalement choisi le critère d'arrêt proposé par [13] car plus de relaxations sont effectuées et par conséquent une meilleure qualité numérique de la solution calculée est obtenue.

A titre d'exemple, le tableau 3.1 est extrait de la publication [67] et présente un exemple de comparaison de différents critères d'arrêt : une méthode centralisée (notée C), une méthode centralisée avec un nombre prédéfini de détections successives (notée C(N), où N est le nombre prédéfini) et une méthode décentralisée (notée D).

	Asynchronous					Synchronous	
Nb. Procs.	Convergence detection	Time/sec	Relaxations			Time/sec	Relaxations
			Min	Max	Average		
2	C	5144	1100	1242	1171	6526	1123
2	C(10)	7353	1133	1909	1521		
2	D	7169	1106	1862	1484		
4	C	2536	1116	1252	1191	3681	1138
4	C(10)	3558	1134	1862	1499		
4	D	3209	1161	1709	1431		
8	C	1200	1103	1253	1151	1659	1138
8	C(10)	1553	1140	1737	1461		
8	D	1464	1096	1720	1400		
16	C	597	1033	1291	1139	626	1138
16	C(10)	587	1124	2609	1594		
16	D	581	1153	2574	1748		
32	C	238	556	1222	1064	361	1145
32	C(10)	313	1132	2517	1999		
32	D	285	1069	2157	1712		
64	C	103	492	1468	1049	318	1165
64	C(10)	108	1291	2077	1700		
64	D	102	495	1469	1049		
128	C	55	357	1408	1020	337	1208
128	C(10)	130	1388	2905	1855		
128	D	109	1284	2685	1704		

FIGURE 3.1 – Comparaison de temps d'exécution et nombre de relaxations moyennes

3.1.4 Synchronisation à chaque pas de temps

Barrière de synchronisation

Certaines implémentations utilisent un processeur supplémentaire $P+1$ qui gère la terminaison des algorithmes asynchrones.

Cependant, la cohérence des calcul (voir 3.1.2) qui implique l'utilisation de la fonction collective MPI_BARRIER ne permet pas de garantir l'indépendance de ce processeur $P+1$ car lui aussi va être bloqué. Il a donc été indispensable d'implémenter une barrière de synchronisation à l'aide d'une fonction de diffusion et c'est le processeur $P+1$ qui s'occupe de la gestion de cette barrière.

Ainsi, pour respecter le schéma en temps, tous les processeurs sont synchronisés grâce à cette barrière et envoient un message au processeur $P+1$. Lorsque le dernier processeur est arrivé, le processeur $P+1$ redémarre chaque processeur de calcul.

L'algorithme 7 décrit l'utilisation du processeur $P+1$:

Algorithme 7 : Algorithme du processeur gestionnaire de convergence

```
1  début
2      tantque la réception d'une commande d'un processeur n'est pas une
           commande de terminaison de l'algorithme P + 1
3          si Barrière alors
4              Attendre tous les processeurs de calcul
5              Envoyer une notification de redémarrage aux processeurs de calcul
6              Envoyer les valeurs aux frontières aux processus voisins
7          fin
8          si convergence alors
9              Réception de la convergence locale
10             Envoyer la convergence globale
11         fin
12     fin
13 fin
```

Barrière de synchronisation Pair-à-Pair

En plus de la barrière de synchronisation vue précédemment dans cette sous-section 3.1.4 et en raison de la conception d'un démonstrateur Pair-à-Pair nommé P2PDC que nous avons utilisé (voir [67], [66]), il a été nécessaire d'implémenter une barrière de synchronisation supplémentaire lorsqu'un nouveau pas de temps est considéré.

En effet, dans ce démonstrateur deux opérations P2P_Send() et P2P_Receive() n'ont pas les mêmes propriétés en fonction de la méthode de communication utilisée et cette caractéristique est spécifique à P2PDC. L'opération P2P_Wait() est utilisée pour synchroniser les processeurs à chaque pas de temps avant le calcul du second membre afin de prendre en compte la valeur de l'itéré précédent en temps. Dans le schéma synchrone (resp. asynchrone), les opérations P2P_Send() et P2P_Receive() sont bloquantes (resp. non bloquantes). Il convient de noter que cette mise en œuvre de la méthode parallèle asynchrone simplifie l'échange des premiers et derniers blocs dans la méthode de relaxation par blocs, mais qu'elle n'est pas adaptée à la partie synchronisée à chaque pas de temps. L'API (Application Programming Interface) de communication P2PDC n'offre pas de véritable barrière de synchronisation mais seulement une fonction qui attend l'arrivée d'un message d'un autre processeur.

Pour satisfaire cette utilisation de barrière à chaque pas de temps, tous les processeurs sont synchronisés grâce à l'opération P2P_Wait inclut et envoient un message à un processeur gestionnaire de barrière dédié parmi tous les processeurs de calcul. Ensuite, ce processeur gestionnaire de barrière redémarre chaque processeur de calcul.

3.1.5 Latence des réseaux

On peut définir la latence des réseaux comme le temps nécessaire pour qu'un paquet de données se déplace d'un point à un autre à travers un réseau en passant d'une machine à une autre via des liaisons et périphériques réseaux qui traitent et acheminent les paquets de données sur différents supports de transmission réseau.

3.1.6 Optimisation par communications persistantes

L'implémentation parallèles synchrones ou asynchrones des problèmes va reposer sur un ensemble de communication qui est répété à l'identique à chaque étape de calcul. L'utilisation de communications bloquantes nécessite l'attente satisfaites de transmissions ou réceptions avant de continuer alors que les communications non bloquantes ne nécessitent pas une transmission ou une réception complète pour poursuivre l'exécution du programme ce qui va permettre d'utiliser le temps dédié aux communications pour effectuer d'autres calculs et ainsi obtenir des gains en performances.

La mise en œuvre de communications persistantes permettent de réduire le traitement des messages qui doivent constamment être envoyés entre deux machines. En effet, ce type de procédé non-bloquant communique continuellement le même message sans avoir à initialiser l'envoi ni la réception à chaque fois. Ces communications persistantes sont de type non-bloquant.

La mise en œuvre de cette optimisation avec MPI peut être effectuée en utilisant les requêtes persistantes proposées par MPI. Pour ce faire, la création du canal de communication consiste à initialiser des requêtes persistantes d'envoi et de réception en faisant appel, pour l'émetteur, à la primitive MPI_SEND_INIT() et pour le récepteur, à la primitive MPI_RECV_INIT().

3.2 Comportement sur diverses architectures parallèles

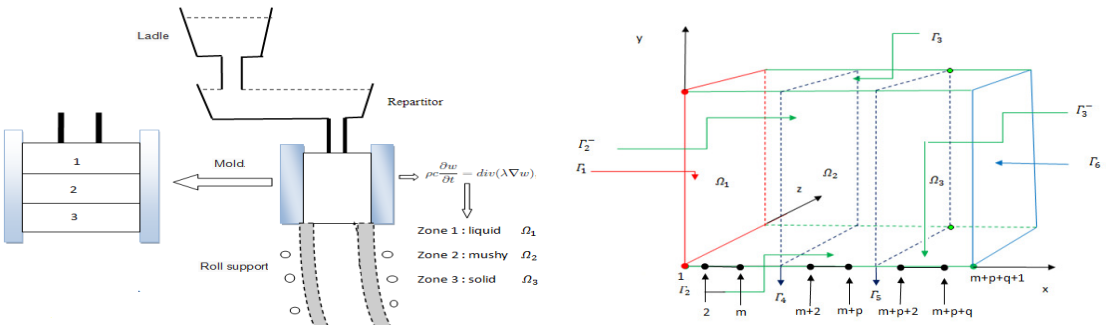
3.2.1 Le problème de solidification de l'acier

Cet article publié sous le titre "Behavior of parallel two-stage method for the simulation of steel solidification in continuous casting" voir [68] et ADES_2019 ainsi que dans la thèse de Ghania Khenniche [90], va nous permettre de décrire le comportement des algorithmes asynchrones pour ce problème particulier considéré, riche en enseignement.

Ce problème modélisé par une équation de la chaleur prenant en compte des phénomènes de rayonnement à la frontière a été très intéressant d'un point de vue comportemental des algorithmes asynchrones. Pour ce type d'application, l'analyse des performances des méthodes parallèles synchrones et asynchrones n'est pas simple à réaliser car l'influence des paramètres décrivant le comportement des méthodes itératives doit être considérée en relation avec les aspects informatiques et surtout algorithmique dans la mesure où ils influent sur la vitesse de convergences des méthodes itératives.

Présentation du problème

Dans l'industrie de l'acier, la coulée continue est un processus intermédiaire entre la fabrication du métal et le laminage. Ce processus, brièvement décrit dans la Figure 3.2a, permet la transformation d'un métal liquide en un métal solide de manière continue.



(a) Schéma général de la coulée continue. (b) Représentation globale des domaines.

FIGURE 3.2 – Principe de la coulée et découpage des domaines.

Pendant le processus, le métal liquide est refroidi par jet d'eau au fur et à mesure dans une machine adaptée. La transformation de la phase liquide en phase solide se

produit dans une zone intermédiaire appelée zone pâteuse. Pour résumer le processus de refroidissement et de solidification, on peut le décrire de la manière suivante : l'acier passe par trois phases dans trois zones désignées par Ω_1 pour la zone liquide, Ω_2 pour la zone pâteuse et Ω_3 pour la zone solide. La température initiale dans Ω_1 est d'environ 1500°C, tandis qu'elle reste égale à 800°C et 600°C respectivement dans Ω_2 et Ω_3. L'acier liquide est versé dans un moule refroidi. La température satisfait l'équation de la chaleur avec des conditions aux limites appropriées décrivant le phénomène physique. Pour décrire cette opération, on suppose que l'échange de chaleur dans le moule est uniforme et que le refroidissement par jet d'eau et le contact thermique avec les rouleaux seront modélisés par la condition de convection et/ou de rayonnement. Il est habituel de supposer que les surfaces en contact avec l'air ambiant sont soumises à des conditions du même type. Les échanges de chaleur qui se produisent entre les sous-domaines liquide-pâteux et pâteux-solide sont modélisés par une condition aux interfaces. Chaque zone de refroidissement est caractérisée par sa température, c'est-à-dire que nous devons successivement trouver le champ de température dans la zone liquide Ω_1, puis dans la zone pâteuse Ω_2, et enfin dans la zone solide Ω_3.

Le modèle mathématique est décrit par l'équation de diffusion de la chaleur dans l'espace tridimensionnel, avec la connaissance de la condition initiale et de six conditions aux limites pour chaque sous-domaine Ω_I, $I = 1, 2, 3$ (voir Figure 3.2a). Ces conditions aux limites sont notamment celles de Dirichlet-Neumann mixtes linéaires et non linéaires non homogènes, ainsi que de conditions linéaires et non linéaires de Fourier, en raison du phénomène de rayonnement, bien décrit par la loi de Stefan sur certaines parties des frontières. De plus, la partie linéaire de l'opérateur continu modélisant la diffusion de chaleur dans chaque sous-domaine Ω_I, $I = 1, 2, 3$, est perturbée par un opérateur continu non linéaire diagonal croissant. Cette dernière propriété découle de la positivité de la température dans chaque zone. En effet, même s'il est raisonnable de penser que physiquement, dans ces trois zones, la température est strictement positive, il n'est pas facile de prouver cette propriété mathématiquement. Par conséquent, il est nécessaire de modifier le modèle mathématique, et nous sommes donc amenés à résoudre trois inéquations variationnelles couplées gouvernant la température définie dans chaque zone. En outre, en prenant en compte à la fois la contrainte de positivité de la température et la non-linéarité découlant du phénomène de rayonnement, nous devons résoudre, dans chaque zone, un problème multivoque, similaire à (1.1.4), car la contrainte de positivité est modélisée par un opérateur multivoque monotone diagonal, qui est en fait le sous-différentiel de la fonction indicatrice des ensembles convexes.

Après la discrétisation, d'une part, de la partie évolutive des problèmes par un schéma implicite en temps et, d'autre part, en utilisant un schéma de différences finies classique pour la discrétisation spatiale avec une grille uniforme, nous devons résoudre numériquement, une séquence de problèmes stationnaires non linéaires multivoques.

Ainsi, à chaque pas de temps, nous devons résoudre trois grands systèmes algébriques couplés, creux et fortement non linéaires.

En raison de la grande taille de ces systèmes, un algorithme itératif est plus approprié pour la résolution des systèmes algébriques. Nous avons donc mis en œuvre des algorithmes de relaxation de type de Newton par point projeté séquentiel, correspondant à un couplage de la méthode de Newton projetée par points, pour les points appartenant à la partie de la frontière où la loi de Stefan se produit, avec la méthode de relaxation projetée, comme la méthode de Gauss-Seidel par points pour les points intérieurs des zones.

De plus, pour obtenir des résultats précis, il est nécessaire de choisir une taille de pas de discrétisation spatiale très petite, ce qui conduit à la résolution de systèmes algébriques multivoques très volumineux. Afin de réduire le temps de calcul et en prenant en compte la propriété bien connue des matrices de discrétisation d'être des M-matrices d'une part, et de la monotonie des opérateurs de discrétisation non linéaires modélisant la loi de Stefan sur certaines parties de la frontière, et également la monotonie du sous-différentiel de la fonction indicatrice des ensembles convexes prenant en compte la positivité de la température sur chaque zone Ω_I, $I = 1, 2, 3$ et d'autre part, nous obtenons des conditions suffisantes pour la convergence des méthodes parallèles synchrones et asynchrones ; compte tenu du mode de résolution considéré des systèmes, nous utilisons donc un algorithme à deux niveaux.

Implémentation parallèle.

La solution numérique a été implémentée en Fortran en utilisant la programmation par passage de messages sur de nombreux ordinateurs indépendants, appelés nœuds ou processeurs. Chaque processeur lance son propre programme et communique avec les autres nœuds en envoyant et en recevant des messages. Les algorithmes parallèles synchrones et asynchrones utilisent le système de passage de messages MPI (Message Passing Interface). Dans notre cas, les données sont divisées en ensembles réguliers et chaque processeur initialise son propre ensemble de données.

La matrice et le second membre ont été implémentées de manière séquentielle puisque cette partie du calcul n'est pas très consommatrice en temps. En d'autres termes, seuls les calculs intensifs ont été parallélisés.

Les solutions parallèles sont implémentées en considérant à chaque pas de temps d'une part une méthode de sous-domaine sans recouvrement et d'autre part une méthode de sous-domaine avec recouvrement (voir paragraphe 1.2.3). Ainsi, chaque zone $\Omega_{\mathcal{I}}, \mathcal{I} = 1, 2, 3$ est divisée en parallélépipède ω_j, chaque parallélépipède étant identifié par un système de pointeurs qui se réfère au nombre de points positionnés sur un des axes de coordonnées et pour la méthode de sous-domaine avec recouvrement on prend en compte le nombre de points en considérant la valeur du recouvrement.

De même, chacune des zones $\Omega_{\mathcal{I}}, \mathcal{I} = 1, 2, 3$ est identifiée par un système d'indices faisant référence soit au premier point soit au dernier point de la zone sur le même axe.

Soit n le nombre de points de discrétisation intérieurs sur chaque axe de coordonnées ; on a alors n^3 points de discrétisation sur le domaine complet Ω qui est divisé en $n/3 \times n \times n$ points sur chaque zone $\Omega_{\mathcal{I}}, \mathcal{I} = 1, 2, 3$.

La méthode décrite au paragraphe 3.1.3 pour la gestion de la terminaison des algorithmes a été utilisée pour l'algorithme synchrone ainsi que pour l'algorithme asynchrone et à chaque pas de temps, celle décrite aux paragraphes 3.1.2 et 3.1.4 va permettre de s'assurer de la cohérence des calculs.

Mise en œuvre parallèle

L'algorithme 5 présenté dans la sous-section 3.1.1 a été implémenté avec des modifications pour s'adapter au problème considéré.

Expérimentations parallèles.

Des expériences de simulation numérique ont été réalisées sur la plateforme Grid'5000 (voir paragraphe 2.2.1). Ces simulations parallèles ont été réalisées sur un seul cluster et sur des clusters couplés distants de la plateforme pour des domaines de tailles différentes. Les expérimentations parallèles ont été réalisées en couplant deux clusters distants de Grid'5000 composés de machines à 8 cœurs avec une vitesse de 2.4 GHz et 128 Go de mémoire vive.

n	Nombre de points dans Ω	Nombre de points dans Ω_1
454	$454 \times 454 \times 454 = 93\ 576\ 664$	$152 \times 454 \times 454 = 31\ 329\ 632$
622	$622 \times 622 \times 622 = 240\ 641\ 848$	$208 \times 622 \times 622 = 80\ 471\ 872$
766	$766 \times 766 \times 766 = 449\ 455\ 096$	$256 \times 766 \times 766 = 150\ 209\ 536$
1051	$1051 \times 1051 \times 1051 = 1\ 160\ 935\ 651$	$350 \times 1051 \times 1051 = 386\ 610\ 350$
n	Nombre de points dans Ω_2	Nombre de points dans Ω_3
454	$151 \times 454 \times 454 = 31\ 123\ 516$	$150 \times 454 \times 454 = 30\ 917\ 400$
622	$207 \times 622 \times 622 = 80\ 084\ 988$	$206 \times 622 \times 622 = 79\ 698\ 104$
766	$255 \times 766 \times 766 = 149\ 622\ 780$	$254 \times 766 \times 766 = 149\ 036\ 024$
1051	$349 \times 1051 \times 1051 = 385\ 505\ 749$	$348 \times 1051 \times 1051 = 384\ 401\ 148$

TABLE 3.1 – Différentes tailles de domaines pour les problèmes.

Nous avons mis en œuvre une méthode de relaxation couplée à la méthode de Newton pour les points de discrétisation aux frontières des sous-domaines $\Omega_{\mathcal{I}}, \mathcal{I} =$

$1, 2, 3$. Pour la résolution des systèmes linéaires, nous utilisons la méthode de Gauss-Seidel par points pour les équations linéaires associées aux points intérieures des sous-domaines $\Omega_{\mathcal{I}}, \mathcal{I} = 1, 2, 3$.

Le domaine Ω est divisé en trois sous-domaines égaux définis comme suit $\Omega_1 = [0, \frac{1}{3}] \times [0, 1]^2$, $\Omega_2 = [\frac{1}{3}, \frac{2}{3}] \times [0, 1]^2$ et $\Omega_3 = [\frac{2}{3}, 1] \times [0, 1]^2$.

Certains paramètres physiques jouent un rôle important sur le comportement des algorithmes et induisent des propriétés de diagonales dominance très forte qui influent fortement sur les vitesse de convergence des algorithmes itératifs.

Pour les simulation numériques, nous avons choisi 100 pas de temps ; la discrétisation d'espace est caractérisée par la valeur de n ce qui conduit à des tailles différentes des zones $\Omega_{\mathcal{I}}, \mathcal{I} = 1, 2, 3$ présentées dans la Table 3.1.

Les résultats pour $n = 454, 622, 766$ et 1051 sont ainsi présentés pour l'utilisation soit d'un cluster, soit d'une grille composée de deux clusters distants lorsque nous utilisons quatre et huit processeurs en mode synchrone et asynchrone.

Pour le critère d'arrêt avec une valeur de seuil $\epsilon = 10^{-3}$ et pour un recouvrement de 3 mailles pour les versions avec recouvrement, les résultats des simulations sur cluster et sur grille pour les modes synchrone et asynchrone sont résumés dans les tableaux présents dans [68] et ADES_2019 sur la page 166 et suivantes.

Analyse des résultats

Dans le cas de l'utilisation de clusters

Le mode parallèle asynchrone ne permet pas d'obtenir de bonnes performances sur cluster car d'une part le taux de convergence est élevé en raison d'une forte diagonale dominance des matrices et d'autre part le réseau d'interconnexion inter-processeurs est extrêmement rapide à 10 Gigabits Ethernet par seconde ; cependant, en mode asynchrone, les temps de communication dus aux échanges entre processeurs sont faibles et les pertes de temps dues aux barrières de synchronisation nécessaires à chaque pas de temps et lors du passage d'un sous-domaine $\Omega_{\mathcal{I}}$ au sous-domaine suivant sont consommatrices de temps.

En mode synchrone, on obtient un effet inverse car les temps de communication dus aux échanges entre processeurs sont importants alors que les barrières de synchronisation sont peu consommatrices de temps.

Néanmoins, par rapport aux temps obtenus en mode séquentiel, on observe que ceux obtenus en mode parallèle synchrone et asynchrone ont diminué de manière significative. Ainsi, la parallélisation de la méthode séquentielle sur cluster dans les deux cas est intéressante.

Dans le cas de l'utilisation de la grille

Dans ce cas, nous utilisons deux et quatre processeurs sur chaque cluster afin d'obtenir une bonne répartition de la charge. Lorsqu'on augmente le nombre de processeurs, le comportement des méthodes asynchrones change complètement ; la vitesse de convergence a moins d'influence sur les performances et les méthodes asynchrones sur une grille sont intéressantes en raison de l'utilisation d'un réseau lent et du peu de synchronisations. Les simulations parallèles sur grille montrent que le mode asynchrone est plus efficace que le mode synchrone.

La méthode asynchrone donne généralement de meilleurs résultats que la méthode synchrone, essentiellement en raison de la latence élevée du réseau qui implique des communications coûteuses. De même, les résultats sont intéressants pour les grands systèmes qui doivent être divisés en sous-systèmes inter-connectés car ils ne peuvent être traités sur un seul ordinateur en raison de la limitation de la taille de la mémoire.

Notons que les valeurs de τ varient entre $1,97$ et $8,92$ lorsque quatre processeurs sont utilisés et entre $3,36$ et $7,06$ sur huit processeurs.

En mode asynchrone, le temps de restitution diminue en raison d'une diminution des communications. Ce gain de performance est dû au poids des synchronisations comme on peut le constater en examinant les temps de communication ; en effet en mode asynchrone les temps d'attente des processeurs sont négligeables par rapport à ceux observés en mode synchrone. C'est le principal avantage des algorithmes asynchrones lors de l'exécution d'un programme sur une architecture comportant de nombreux processeurs hétérogènes éloignés les uns des autres et reliés par un réseau lent classique.

En résumé, plusieurs leçons émergent de ces simulations parallèles. La première remarque concerne l'interaction entre le comportement de l'algorithme numérique itératif utilisé par rapport à l'architecture de la machine. Compte tenu des propriétés de diagonale dominance des matrices, la méthode de calcul converge très rapidement, c'est-à-dire en quelques itérations. Cependant, les performances parallèles ne sont pas les mêmes selon que les calculs sont effectués sur un cluster ou sur une grille de calcul comportant des machines distantes et hétérogènes. Ceci est essentiellement dû au réseau d'interconnexion qui est rapide sur un cluster et lent sur une grille de calcul. Dans ces conditions, et pour cette application cible où la convergence est rapide, le poids des synchronisations joue un rôle important sur les performances de l'algorithme. Il est donc clair que, compte tenu de cette grande vitesse de convergence, sur cluster, les méthodes synchrones sont intéressantes à utiliser dans ce cas étant donné le peu de synchronisations et le réseau rapide utilisé. Par contre, sur une grille de calcul, les méthodes asynchrones montrent tout leur intérêt du fait de l'hétérogénéité des machines distantes utilisées et surtout de la lenteur du réseau.

La deuxième remarque concerne le choix du test d'arrêt. En mode parallèle synchrone, l'implémentation de ce test d'arrêt ne pose pas un énorme problème d'implémentation. Par contre, en mode parallèle asynchrone, notre expérience nous a conduit à implémenter un test centralisé plus efficace.

La troisième remarque concerne, d'une part, le découpage en tâches parallèles, c'est-à-dire la granularité des tâches et, d'autre part, le nombre de processeurs utilisés, compte tenu de la taille du problème et des performances des algorithmes. Dans cette application cible où le poids des calculs est relativement faible compte tenu de la vitesse de convergence, il est préférable d'avoir une granularité élevée et un nombre de processeurs pas très important. Cela permet de réduire considérablement le surcoût des communications à travers le réseau. Expérimentalement, on peut déterminer le nombre de processeurs à utiliser ; en effet, lorsque l'efficacité reste constante lorsque le nombre de processeurs augmente, des processeurs supplémentaires ne sont pas nécessaires.

3.2.2 Le problème de mathématiques financières

Pour cette application, nous renvoyons principalement à "Asynchrone peer-to-peer distributed computing for financial applications" (voir [66] et IEEE) lors d'utilisation d'architecture Peer-to-peer et "Asynchrone Schwarz methods applied to constrained mechanical structures in grid environment" lors d'utilisation d'architecture sur grille (voir [34] et ADES_2014).

En mathématiques financières, on distingue habituellement deux types de modèle économiques constitué d'une part par les options américaines et d'autre part par les options européennes ; elles sont modélisées par l'équation de Black et Scholes. Le problème consiste à résoudre un problème aux limites dépendant du temps défini sur un domaine non borné généralement inclus dans l'espace tridimensionnel.

Les équations sont modélisées par un problème aux limites décrivant l'évolution des options d'achat (call) et de vente (put) [159]. Elle permettent de déterminer un prix pour une action. On évalue l'état initial à partir d'un état final donné.

Un artifice classique consiste à résoudre l'équation de Black et Scholes sur un domaine borné et à augmenter la taille du domaine pour assurer la convergence vers la solution exacte ; on a donc de très grands systèmes algébriques à résoudre. Habituellement le modèle est constitué par des équations de convection-diffusion évolutives ; dans le cas d'options Européennes il n'y a pas de contrainte sur la solution alors que pour les options Américaines la solution est soumises à des contraintes et on doit donc résoudre des inéquations variationnelles. Dans ce dernier cas, on doit donc effectuer une projection sur un sous-ensemble convexe définissant les contraintes.

Présentation du problème

Pour ce type de problèmes, nous avons mis en œuvre les méthodes itératives parallèles asynchrones et synchrones d'une part sur le simulateur peer-to-peer (P2P) et d'autre part sur une architecture grille.

Dans le cas du simulateur P2P, le réel enjeu consistait à développer de nouveaux protocoles et environnements pour le HPC ; dans ce cas l'objectif est la gestion des communications, l'extensibilité, l'hétérogénéité et la volatilité des pairs sur le réseau. Les deux problèmes considérés étant des problèmes d'évolution, il a fallu implémenter une barrière de synchronisation. Par ailleurs, les implémentations sur grille n'ont pas posé de problème particulier.

On indique ci-dessous, d'une part le modèle régissant les options européennes

$$\begin{cases} \frac{\partial v(t,x)}{\partial t} - (r - \frac{\sigma^2}{2})\nabla v - \frac{\sigma^2}{2}\Delta v + rv = 0 \text{ dans } [0,T] \times \Omega \\ v(0,x) = \psi(x) \\ \text{Conditions limites de } v(t,x) \text{ définies sur } \delta\Omega \end{cases} \tag{3.2.1}$$

et d'autre part celui modélisant les options américaines

$$\begin{cases} \frac{\partial v(t,x)}{\partial t} - (r - \frac{\sigma^2}{2})\nabla v - \frac{\sigma^2}{2}\Delta v + rv \geqslant 0,\ v(t,x) \geqslant \psi(x),\ \text{dans } [0,T] \times \Omega \\ (\frac{\partial v(t,x)}{\partial t} - (r - \frac{\sigma^2}{2})\nabla v - \frac{\sigma^2}{2}\Delta v + rv)(v(t,x) - \psi(x)) = 0,\ \text{dans } [0,T] \times \Omega \\ v(0,x) = \psi(x) \\ \text{Conditions limites de } v(t,x) \text{ définies sur } \delta\Omega \end{cases} \tag{3.2.2}$$

où $\psi(x) = \max(x - K, 0)$ dans le cas du *call*, et $\psi = \max(K - x, 0)$ dans le cas du *put*. Les notations utilisées dans 3.2.1 sont les suivantes : r pour le taux d'intérêt, σ pour la volatilité (variabilité), K pour le prix d'exercice, v pour le prix de l'option et T pour le temps final.

Par la suite, après discrétisation, on considère la solution du système linéaire algébrique du type $AU = F$ pour le calcul d'options européennes. La décomposition en blocs du système algébrique linéaire précédent conduit à résoudre :

$$\mathcal{A}_{l,l}U_l = F_l - \sum_{j \neq i} \mathcal{A}_{k,l}U_k), l \in \{1, \ldots, \alpha\}.$$

où α désigne le nombre de blocs. A chaque pas de temps, on associe à l'équation précédente un problème de point fixe.

En ce qui concerne la solution du problème discrétisé de l'option américaine, on considère la projection sur l'ensemble convexe définissant les contraintes de la façon suivante :

$$U_l = Proj(\mathcal{A}_{l,l}^{-1}(F_i - \sum_{j \neq i} \mathcal{A}_{l,k} U_k)), l \in \{1, \ldots, \alpha\}.$$

ce qui définit formellement l'application de point fixe dans la mesure où on inverse jamais les matrice blocs diagonales à cause de la mauvaise représentation des nombres en machine ; pratiquement pour les options américaines on résoud l'équation sans contrainte puis on projette la solution obtenue sur le convexe.

On est ainsi ramené au cadre algorithmique du chapitre 1.

Implémentation parallèle.

Nous avons utilisé en incluant à celui-ci une barrière de synchronisation, un démonstrateur nommé P2PDC développé au LAAS-CNRS. Cet environnement permet des communications entre pairs en utilisant un protocole de communication auto-adaptatif pair à pair nommé P2PSAP qui permet éventuellement de choisir dynamiquement le mode de communication le plus approprié entre les pairs en fonction des schémas de calcul et des éléments du contexte comme la topologie.

Afin de permettre aux programmeurs de développer facilement leur propre application, P2PDC propose un modèle de programmation avec un ensemble réduit d'opérations de communication. Il n'y a que trois opérations classiques : P2P_Send, P2P_Receive et P2P_Wait. L'idée est de faciliter la programmation d'applications P2P à grande échelle et de masquer autant que possible la complexité de la gestion des communications. L'opération P2P_Wait est particulière et est utilisée afin d'attendre l'arrivée d'un message d'un autre pair.

Les messages échangés entre pairs peuvent être décomposés en deux classes : les messages de données et les messages de contrôle. Les messages de données sont utilisés pour échanger les valeurs limites des composants du bloc aux interfaces entre les pairs ; ainsi les conditions aux limites des EDP sont bien prises en compte. Par ailleurs, les messages de contrôle sont utilisés pour échanger l'état de calcul comme les critères de terminaison locaux, la commande de terminaison, etc. (voir [22]).

L'implémentation de l'application financière est réalisée avec l'environnement P2PDC en tenant compte du schéma de calcul d'une part et du critère d'arrêt de l'algorithme itératif d'autre part. L'algorithme parallèle est basé sur le paradigme SPMD, qui est couramment utilisé pour les applications parallèles et distribuées. Dans notre cas, chaque processus initialise son propre ensemble de données.

La méthode décrite dans le paragraphe 3.1.3 pour la gestion de la terminaison des algorithmes a été utilisée pour l'algorithme synchrone ainsi que pour l'algorithme asynchrone et à chaque pas de temps, celle décrite dans le paragraphe 3.1.2 et 3.1.4 va permettre de s'assurer de la cohérence des calculs.

L'algorithme 5 présenté dans la sous-section 3.1.1 a été implémenté avec des modifications pour s'adapter au problème cité précédemment.

Le principe de l'implémentation est décrit par l'algorithme 8 sachant qu'on a utilisé la barrière pair-à-pair.

Algorithme 8 : Algorithme exécuté sur les différents pairs.

```
1  début
2      Donner la condition initiale V_0
3      ...
4      pour Chaque pas de temps faire
5          Calculer le second membre
6          ...
7          tantque Non convergence
8              Avec calcul synchrone ou asynchrone
9              Résoudre avec une méthode de relaxation par block A.U = F
10             Projeter dans le cas des options américaine U_0 = U
11             ...
12         fin
13     fin
14 fin
```

Dans le schéma synchrone (resp. asynchrone), les opérations P2P_Send() et P2P_Receive() sont bloquantes (resp. non bloquantes). Notons également que chaque matrice bloc diagonales est tridiagonal et les sous-systèmes diagonaux sont résolus en utilisant la méthode TDMA qui correspond à la mise en œuvre de la méthode de Gauss pour des matrices tridiagonales. Cette méthode est bien adaptée à la résolution de sous-systèmes tridiagonaux. Plus précisément, dans l'implémentation considérée, plusieurs blocs tridiagonaux adjacents sont rassemblés ; ainsi, ce type de méthode peut être considéré comme une méthode sous-domaine sans recouvrement entre les sous-domaines correspondant donc à une méthode de relaxation par grands blocs.

Expérimentations parallèles.

Les schémas de calcul ont été implémentés en langage C comme un algorithme parallèle utilisant l'environnement P2PDC. La plateforme Grid'5000 a été utilisée comme support pour les expériences de calcul. La taille des systèmes algébriques à résoudre a été de 256^3. Il est à noter que d'autres expérimentations du même type de problèmes hors cadre de l'ANR ont été réalisées sur la plateforme Grid'5000.

Les expérimentations parallèles ont été réalisées en couplant deux clusters distants de Grid'5000 composés de machines hétérogènes avec soit 8 cœurs ou soit 2 cœurs avec une vitesse de 2.83 Ghz ou respectivement 2.4 GHz ainsi que 8 Go ou respectivement

2 Go de mémoire vive.

Les expériences de calcul sont résumées dans les tableaux suivants aussi bien pour le calcul d'options européenne que pour celui d'options américaines.

	Asynchrone		Synchrone		τ
Peers	Temps/s	Relaxation (moy.)	Temps/s	Relaxations	
2	5492	1217	4904	1004	0.89
4	3158	1353	3158	1015	1.00
8	1289	1309	1370	1015	1.06
16	515	1502	589	1030	1.14
32	195	1486	297	1035	1.52
64	90	1478	292	1053	3.24
128	65	1380	303	1091	4.66

TABLE 3.2 – Temps d'exécution, nombre de relaxations moyennes et τ pour les options européennes.

	Asynchrone		Synchrone		τ
Peers	Temps/s	Relaxation (moy.)	Temps/s	Relaxations	
2	7169	1484	6526	1123	0.91
4	3209	1431	3681	1138	1.15
8	1464	1400	1659	1138	1.13
16	581	1748	626	1138	1.08
32	285	1712	361	1145	1.27
64	102	1049	318	1165	3.11
128	109	1704	337	1208	3.09

TABLE 3.3 – Temps d'exécution, nombre de relaxations moyennes et τ pour les options américaines.

Pour d'autres résultats, il est possible de consulter (IEEE) et [66] lors de l'utilisation d'architecture peer-to-peer et ADES_2014 et [34] lors de l'utilisation d'architecture grille.

Analyse des résultats

Pour l'application considérée et l'architecture P2P utilisée, lorsque le nombre de machines est supérieur à 8 pour les options européennes et à 4 pour les options américaines, le schéma de calcul asynchrone est plus performant que le schéma synchrone. Par exemple, avec 128 pairs, le schéma de calcul asynchrone est clairement

cinq fois dans le cas d'options européennes et trois fois dans le cas d'options américaines plus performant que le schéma synchrone. Cette caractéristique montre que les algorithmes asynchrones sont moins sensibles à la latence du réseau. Par conséquent, l'équilibrage de la charge n'est pas nécessaire pour réduire le temps de restitution des travaux.

3.2.3 Autres applications

Dans un souci de concision nous mentionnons ci-dessous d'autres applications traitées par des méthodes parallèles synchrones et asynchrones. Dans ces problèmes, il apparait toujours la résolution de problèmes de diffusion ou de convection-diffusion ce qui implique que les résultats obtenus sont similaires à ceux présentés précédemment.

Les applications traitées concernent :

— Un problème de séparation de protéine par électrophorèse correspondant au couplage de l'équation de Navier-Stockes incompréhensible décrivant le comportement du fluide couplées à autant d'équation de convection-diffusion qu'il y a de protéines à séparer, le tout étant également couplé à une équation de diffusion décrivant le comportement du champ électrique. Pour cette application, nous renvoyons principalement à l'article "Asynchrone grid computing for the simulation of the 3D electrophoresis coupled problem" (voir [35] et ADES_2013) sur architecture grille ainsi qu'aux résultats page 230 et suivantes.

— Un autre problème traité dans le cadre de la préparation de la thèse CIFRE de Vincent Partimbene concerne l'interaction fluide-structure couplant l'équation de Navier-Stockes incompréhensible décrivant le comportement du fluide couplé avec l'équation de Navier décrivant le comportement de la structure. Cette étude a été développée dans le cadre d'une collaboration avec la société SEGULA FRANCE. Pour cette application, nous renvoyons principalement à l'article "Asynchrone multi-splitting method for linear and pseudo-linear problems" (voir [119] et ADES_2019b) sur architecture grille ainsi qu'aux résultats page 146 et suivantes.

— Dans le cadre d'une collaboration avec l'université de Franche-Comté a été réalisé, pour résoudre des problèmes de convection-diffusion non-linéaire, un couplage des méthodes parallèles synchrone et asynchrone avec la méthode de Krylov qui permettait de résoudre les sous-système bloc diagonaux. Dans ce cas, Les méthodes parallèles asynchrones ont été analysées d'une part par des techniques de contraction et d'autre part par des techniques d'ordre partiel. Pour cette application, nous renvoyons principalement à l'article "Coupling parallel Asynchrone multisplitting methods with Krylov methods to solve pseudo-linear evolution 3D problems" (voir [70] et JOCS_2021) sur architecture grille ainsi qu'aux résultats page 107 et suivantes.

— Une autre application permettant de calculer la pression exercée sur un fluide a également été traité. Dans cette étude le modèle considéré est constitue avec par une problème avec contrainte unilatérale au bord. Pour cette application, nous renvoyons principalement à l'article "Grid solution of problem with unilateral constraints" (voir [36] et ALGONUM_2017) sur architecture grille ainsi qu'aux résultats page 200 et suivantes.

Toutes ces études nous ont permis de constater que lorsque les machines sont hétérogène, géographiquement distantes et reliées par un réseau avec un fort taux de latence, les algorithmes parallèles asynchrones permettaient d'obtenir de bonnes performances comparées au méthode synchrone et était très intéressantes à utiliser dans le cas de résolution de problèmes de très grandes tailles.

3.3 Algorithmes parallèles asynchrones sur Cloud

3.3.1 Problèmes linéaires

Compte tenu de l'activité au laboratoire IRIT, nous avons été conduit à utiliser des architectures cloud au lieu d'utiliser des grilles de calcul ; actuellement nous avons des études bien avancées conduisant à la thèse d'Amine Rahalli.

Pour la résolution de problèmes de diffusion et de convection-diffusion, nous renvoyons à l'article "Performances analysis of parallel Asynchrone and Synchrone algorithms sur cloud architecture for PDE" (voir [122] et ADES_2023).

Dans nos travaux antérieurs (voir par exemple [139]), on a observé expérimentalement que les variantes asynchrones donnaient de meilleurs résultats sur les architectures de type grille en raison de la lenteur des communications entre les processeurs parce que la bande passante est partagée par d'autres machines. En effet, les latences pénalisent et affectent les résultats de la simulation. De plus, cet effet est accru par la distance géographique entre les machines et aussi, lorsque les machines sont hétérogènes ce qui est le cas sur des architectures grilles de calcul ou cloud computing.

Dans cette nouvelle étude, on met en œuvre des méthodes de relaxation parallèles pour résoudre des systèmes algébriques, en comparant les versions synchrones et asynchrones. L'objectif est de comparer les performances de ces méthodes sur des architectures cloud computing, en prenant en compte les couches logicielles supplémentaires et leurs effets sur le temps de restitution du travail. L'étude montre que, sur ces architectures, l'implémentation de méthodes itératives asynchrones offrent de meilleurs résultats en termes de temps de résolution par rapport aux versions synchrones.

3.3.2 Implémentation parallèles

Nous décrivons ici, l'implémentation parallèle pour résoudre un problème de diffusion et un problème de convection-diffusion. Conformément à ce qui a été indiqué dans le chapitre 2 et le début du chapitre 3, le principe de mise en œuvre des méthodes itératives parallèles asynchrones et synchrones est décrit par l'algorithme 9, basé sur l'algorithme 5 présenté dans la sous-section 3.1.1.

Algorithme 9 : Algorithme exécuté sur les différents pairs.

```
1  début
2      ...
3      tantque Non convergence
4          pour Chaque sous-domaines faire
5              ...
6              tantque Non convergence
7                  pour Chaque sous-domaine assigné au processeur faire
8                      Choisir un calcul synchrone ou asynchrone
9                      Résoudre tous les systèmes linéaires
10                 fin
11                 Effectuer les communications des valeurs limites du
                    sous-domaine Résoudre avec une méthode de relaxation par
                    block
12                 ...
13             fin
14         fin
15     fin
16 fin
```

Expérimentation parallèles

Expérimentations sur une architecture de type cloud

Comme pour les problèmes précédents, l'implémentation a été réalisée à l'aide de MPI (Interface de Passage de Messages) en utilisant des codes écrits en langages C ou Fortran. La création des matrices et des seconds membres s'est fait de manière parallèle. Pour les algorithmes de sous-domaines parallèles sans recouvrement, une méthode de relaxation par bloc est effectuée sur chaque processus, avec un seul sous-domaine attribué à chaque processeur. Ces sous-domaines sont formés en regroupant des blocs adjacents.

Dans le "short paper" [121], des simulations d'algorithmes parallèles synchrones et asynchrones ont été menées sur une architecture de type cloud computing basée

sur la plateforme Grid'5000.

Les résultats des simulations synchrone et asynchrone effectuées montrent que, pour l'application considérée et l'architecture utilisée, le schéma de calcul asynchrone donne de meilleurs résultats que le schéma synchrone. Sur 64 machines, cet algorithme est nettement plus rapide de $4,6$ et $5,0$ respectivement.

TABLE 3.4 – Résultats des simulations synchrones, asynchrones et la valeur de τ avec une méthode de sous-domaine sans recouvrement pour résoudre un problème de diffusion.

Size	Synchrone		Asynchrone		τ
n^3	Relaxations	Temps exec (s)	Relaxations	Temps exec (s)	
360^3	2 866 016	6 076	3 815 801	1 308	4.6
480^3	6 364 544	38 460	7 895 476	7 728	5.0

Expérimentations sur une architecture cloud

Pour cette nouvelle étude expérimentale, des expériences informatiques ont été menées sur France Grilles-Cloud (voir aussi 2.2.1). Pour chaque problème résolu par la méthode du sous-domaine sans et avec recouvrement, nous avons considéré différentes tailles 240^3, 360^3 et 480^3.

Comme nous l'avons déjà précisé, le cloud computing offre de nombreux avantages mais aussi certaines spécificités comme le caractère hétérogène des ressources proposées, notamment les caractéristiques matérielles, les performances des serveurs et les machines virtuelles (VM) qui composent l'infrastructure cloud. Cette dernière propose donc des ressources incluant des différentes puissances de calcul, de capacité mémoire, de stockage et de systèmes d'exploitation.

L'hétérogénéité des ressources permet donc aux utilisateurs de personnaliser leurs configurations pour répondre à des exigences spécifiques.

Deux autres spécificités sont importantes comme la localisation géographique du fait que les centres de données du fournisseur cloud sont répartis dans le monde entier. Les performances du réseau sont un autre facteur d'hétérogénéité car certaines instances peuvent offrir un accès réseau rapide, tandis que d'autres peuvent avoir des débits plus modestes.

La localisation physique des ressources et les performances réseaux peuvent influencer la latence des réseaux. Certains types d'applications de calcul haute performance nécessitent une faible latence du réseau pour répondre à leurs demandes de calcul. Si celles-ci sont élevées, elles entraînent une dégradation des performances qui peut avoir un impact significatif en particulier dans le cadre de simulations en cloud

computing.

Nous utilisons le mode IaaS et une fois les ressources demandées allouées, nous les utilisons sans connaître leur configuration matérielle spécifiques. De plus, nous ne disposons d'aucune information concernant le partage éventuel de machines physiques entre plusieurs utilisateurs.

Afin de prendre totalement en compte les spécificités du Cloud Computing commercial et leur hétérogénéité, nous avons utilisé ces ressources en prenant en compte la multiplicité des CPU virtuel (vCPU) sur une même machine et les partages de mémoires vive entre les vCPUs. De plus, les machines étant sur le même site du centre de calcul IN2P3 nous avons aussi prévu différentes configurations de débits réseaux pour simuler un comportement géographiquement distant et des accès réseaux différents.

Nous avons aussi utilisé un système d'exploitation uniforme pour plus de facilité. Nous avons également mis en place des scripts pour déployer les codes sur les ressources en nuage, y compris les bibliothèques nécessaires pour l'environnement MPI. Des scripts supplémentaires ont été utilisés pour exécuter les simulations et récupérer les résultats.

Le nombre de machines proposées et utilisées a été fixé à 8 en fonction de la taille mémoire utilisée ($\equiv 8Go$) par nos algorithmes et éviter au maximum le swapping disque.

TABLE 3.5 – Openstack flavors liste.

Nom	RAM	Disk	vCPUs
4CPU_8GB-RAM	8192	64	4
1CPU_8GB-RAM	8192	64	1
2CPU_4GB-RAM	4096	64	2
1CPU_12GB-RAM	12288	64	1
1CPU_2GB-RAM	2048	64	1
8CPU_32GB-RAM	32768	32	8
6CPU_12GB-RAM	12288	64	6

Nous considérons d'abord la solution numérique du problème de diffusion et de convection-diffusion, avec des expériences sans recouvrement et avec un recouvrement de deux mailles.

Les tableaux 3.7, 3.8 et 3.9 montrent clairement les avantages de l'utilisation d'algorithmes parallèles asynchrones sur cloud et on remarque que le paramètre τ augmente avec la dimension de la matrice de discrétisation.

Dans les expériences asynchrones parallèles, on remarque que le nombre de relaxations nécessaires pour atteindre la convergence est plus élevé que celui requis par les schémas synchrones parallèles. C'est un inconvénient bien connu des sché-

TABLE 3.6 – Openstack instance liste.

Flavor	Nom Instance	vCPU	Image (OS/SE)
8CPU_32GB-RAM	myvm1	1	Ubuntu-20.04
8CPU_32GB-RAM	myvm2	1	Ubuntu-20.04
1CPU_12GB-RAM	myvm3	1	Ubuntu-20.04
6CPU_12GB-RAM	myvm4	1	Ubuntu-20.04
1CPU_8GB-RAM	myvm5	1	Ubuntu-20.04
4CPU_8GB-RAM	myvm6	1	Ubuntu-20.04
8CPU_32GB-RAM	myvm7	1	Ubuntu-20.04
8CPU_32GB-RAM	myvm8	1	Ubuntu-20.04

TABLE 3.7 – Synchrone, Asynchrone et τ résultats de simulation avec une méthode de sous-domaine sans recouvrement pour le problème de diffusion.

Size	Synchrone		Asynchrone		τ
n^3	Relaxations	Temps d'exécution (s)	Relaxations	Temps d'exécution (s)	
240^3	6 128	5 141	42 682	762	6.75
360^3	11 904	23 473	43 706	2 191	10.71
480^3	18 920	70 059	44 956	6 319	11.09

TABLE 3.8 – Résultats des simulations synchrones, asynchrones et la valeur de τ avec une méthode de sous-domaine sans recouvrement pour le problème de convection-diffusion.

Size	Synchrone		Asynchrone		τ
n^3	Relaxations	Temps exec (s)	Relaxations	Temps exec (s)	
240^3	23 840	20 772	59 243	1 943	10.70
360^3	133 728	271 516	243 584	11 880	22.86
480^3	183 520	965 295	296 317	42 480	22.72

mas itératifs asynchrones, dû à l'utilisation de composants potentiellement retardés. Cependant, le surcoût de calcul engendré par ces relaxations supplémentaires est inférieur à celui des synchronisations combinées à l'inactivité des processeurs dans les schémas synchrones parallèles.

Bien que la vitesse de convergence des méthodes asynchrones soit généralement plus faible, elle dépend de la fréquence des mises à jour des calculs et de la décomposition initiale du problème en sous-problèmes. Certains types de décomposition peuvent aboutir à un meilleur taux de convergence, mais en général, cela réduit le

TABLE 3.9 – Résultats des simulations synchrones, asynchrones et la valeur de τ avec une méthode de sous-domaine avec recouvrement pour le problème de convection-diffusion.

Size	Synchrone		Asynchrone		τ
n^3	Relaxations	Temps exec (s)	Relaxations	Temps exec (s)	
120^3	11 475	4 578	131 472	441	10.38
240^3	43 889	110 836	245 576	5 720	19.38
360^3	89 601	496 270	290 256	21 887	22.67
480^3	171 671	1 666 259	348 488	61 747	26.99

temps nécessaire pour atteindre la convergence.

Malgré un plus grand nombre de relaxations, le temps écoulé avec le schéma asynchrone est inférieur à celui obtenu avec le schéma synchrone. Ces expériences parallèles montrent que sur une architecture cloud, les méthodes parallèles asynchrones présentent un réel intérêt lorsque de nombreuses synchronisations sont nécessaires. Cela se produit notamment lorsque la convergence est lente, particulièrement lorsque les communications entre les machines sont lentes, en particulier lorsque les connexions ne sont pas directes.

De plus, l'utilisation d'algorithmes asynchrones parallèles s'avère intéressante dans un environnement cloud, surtout en ce qui concerne l'élimination des temps d'inactivité. L'asynchronisme est donc une façon efficace de gérer les surcharges de communication et les déséquilibres de charge. En outre, l'utilisation de méthodes asynchrones a un impact positif sur l'environnement et permet des économies d'énergie électrique, surtout lorsque des granularités faibles sont prises en considération.

3.3.3 Problèmes non linéaires

Par ailleurs nous étudions actuellement les performances sur architecture cloud pour résoudre des problèmes multivoques.

Nous reprenons le problème de la solidification de l'acier par la méthode alternée de Schwartz avec un chevauchement de 3 mailles et 5 pas de temps. Pour ce problème, les résultats des expériences sur une architecture cloud avec 8 machines virtuelles sont présentés dans les Tableaux 3.10 et 3.11. Ainsi, dans ce cas, nous obtenons des valeurs extrêmement intéressantes de τ de l'ordre de 99 (respectivement 114) lorsque $n = 454$ (respectivement $n = 622$).

Dans le cas synchrone, le nombre d'itérations de la méthode de Schwarz pour atteindre la convergence est de 276 lorsque $n = 454$ et de 389 lorsque $n = 622$. Nous avons approximativement mesuré le nombre d'itérations en mode asynchrone

et obtenu respectivement 355 et 574 itérations pour $n = 454$ et $n = 622$; notons que le ratio du nombre de relaxations en mode asynchrone par rapport à celui en mode synchrone permet d'obtenir les mêmes résultats, ce qui permet d'estimer le nombre de macro-itérations en mode asynchrone pour converger.

	Synchrone résultats sur 8 machines				
			communication times		
n	Temps d'exécution	relaxations	compute	barrier	% comms
454	48 648	1 100 453 134	55	20 887	43.05
622	130 814	3 962 917 669	115	55 150	42.25

TABLE 3.10 – Temps d'exécution (sec), relaxations et communications sur cloud avec des algorithmes parallèles synchrones avec recouvrement et seuil $\epsilon = 10^{-3}$ pour le problème de solidification de l'acier.

	Asynchrone résultats sur 8 machines					
			communication times			
n	Temps d'exécution	relaxations	compute	barrier	% comms	τ
454	491	1 418 700 782	34	50	17.11	99.10
622	1 142	5 837 594 448	123	135	22.59	114.55

TABLE 3.11 – Temps d'exécution (sec), relaxations, communications, et τ sur cloud avec algorithme parallèles asynchrones avec recouvrement et seuil $\epsilon = 10^{-3}$ pour le problème de solidification de l'acier.

En reprenant le problème avec contraintes unilatérales sur une architecture cloud, avec utilisation de 8 machines virtuelles et en considérant trois maillages constitués respectivement de 320^3, 400^2 et 480^3 points de grille. Les résultats des expériences parallèles sur une architecture cloud sont présentés dans le Tableau 3.12 et 3.13.

Pour ce problème avec contraintes unilatérales, nous obtenons des valeurs de τ plus petites que celles obtenues pour la résolution du problème de solidification de l'acier. Dans l'étude antérieure (voir [121]), pour la résolution de problèmes de diffusion et de convection-diffusion, nous avons obtenu des valeurs de τ du même ordre, entre vingt et trente, selon la taille du système linéaire discrétisé. Cependant, il convient de noter que l'implémentation sur une architecture cloud améliore très nettement les valeurs de τ.

Enfin, la dernière application concerne un problème financier classique également modélisé par une inégalité variationnelle évolutive. Pour les expériences parallèles sur une architecture cloud, nous utilisons à nouveau les 8 machines virtuelles et nous limitons ces comparaisons d'expériences parallèles à des grilles de 320^3, 400^2 et

	Synchrone résultats sur 8 machines				
			communication times		
n	Temps d'exécution	relaxations	compute	barrier	% comms
320	22 889	13 712	327	1 450	92
400	45 697	17 664	823	1 466	95
480	79 084	21 328	1 809	5 342	91

TABLE 3.12 – Temps d'exécution (sec), relaxations et communications sur cloud avec des algorithmes parallèles synchrones avec recouvrement et seuil $\epsilon = 10^{-6}$ pour le problème avec contrainte unilatérale.

	Asynchrone résultats sur 8 machines					
			communication times			
n	Temps d'exécution	relaxations	compute	barrier	% comms	τ
320	1 037	46 711	802	96	13	22.07
400	1 876	48 300	1 697	94	5	24.36
480	2 557	78 929	2 279	70	8	30.93

TABLE 3.13 – Temps d'exécution (sec), relaxations, communications, et τ sur cloud avec algorithme parallèles asynchrones avec recouvrement et seuil $\epsilon = 10^{-6}$ pour le problème avec contrainte unilatérale.

480^3 points. Les résultats de ces expériences parallèles sur l'architecture cloud sont présentés dans les Tableaux 3.14 et 3.15.

	Synchrone résultats sur 8 machines				
			communication times		
n	Temps d'exécution	relaxations	compute	barrier	% comms
320	56 366	34016	859	11046	79
400	98 863	38448	1778	15179	83
480	155 970	42 384	3 759	16 714	87

TABLE 3.14 – Temps d'exécution (sec), relaxations et communications sur cloud avec des algorithmes parallèles synchrones avec recouvrement et seuil $\epsilon = 10^{-6}$ pour le problème d'options américaines.

Pour ce dernier problème, nous observons un comportement similaire à ceux précédemment étudiés.

En conclusion, les résultats obtenus confirment donc nos attentes : les algorithmes parallèles asynchrones sont plus efficaces que les méthodes synchrones lors

	Asynchrone résultats sur 8 machines					
			communication times			
n	Temps d'exécution	relaxations	compute	barrier	% comms	τ
320	3031	65760	2839	53	5	19
400	3973	74518	3841	47	2	25
480	4 703	88 992	4 006	60	14	33

TABLE 3.15 – Temps d'exécution (sec), relaxations, communications, et τ sur cloud avec algorithme parallèles asynchrones avec recouvrement et seuil $\epsilon = 10^{-6}$ pour le problème d'options américaines.

de l'utilisation de ressources cloud, sans connaissance de leur positionnement, leur utilisation partagée, la latence du réseau et l'architecture matérielle. Les expériences parallèles montrent que sur une architecture cloud, les méthodes parallèles asynchrones présentent un réel intérêt lorsque le traitement parallèle nécessite un grand nombre de synchronisations.

Dans les travaux futurs, nous envisagerons également la comparaison des méthodes itératives parallèles synchrones et asynchrones pour la résolution d'autres types de problèmes aux limites non linéaires.

Chapitre 4

Conclusion et perspectives

4.1 Conclusion

Les travaux effectués concernent la simulation numérique parallèle synchrone et asynchrone appliquée à la résolution de problèmes complexes de très grandes tailles pour des applications pluridisciplinaires et industrielles. Les contributions se situent d'une part vers l'implémentation de méthodes parallèles synchrones et asynchrones en particulier sur les tests d'arrêt des itérations dans un cadre de calculs asynchrones et d'autre part sur les enseignements que l'on peut tirer du comportement des expérimentations en liaison avec l'architecture des machines utilisées et la rapidité du réseau d'interconnexions.

L'utilisation de ces méthodes est intéressante pour la simulation numérique d'applications industrielles et à permis d'une part l'implémentation d'algorithmes parallèles ou distribués, asynchrones ou synchrones et d'autre part de réaliser ainsi des simulations en faisant appel au calcul haute-performance sur des diverses architectures dédiées (HPC, grille, cluster, simulateur peer-to-peer, cloud).

Les analyses des expérimentations ont permis de montrer un gain de temps d'exécution non négligeable pour les méthodes asynchrones par rapport aux algorithmes synchrones dans le cas de machines éloignées géographiquement.

Du fait de la particularité pluridisciplinaire de cette thématique de recherche, mes travaux peuvent s'intégrer dans de nombreux autres domaines pluridisciplinaires et thèmes scientifiques, pour lesquels des compétences, en calcul intensif, parallèle ou en architectures distribuées seraient nécessaires.

4.2 Perspectives

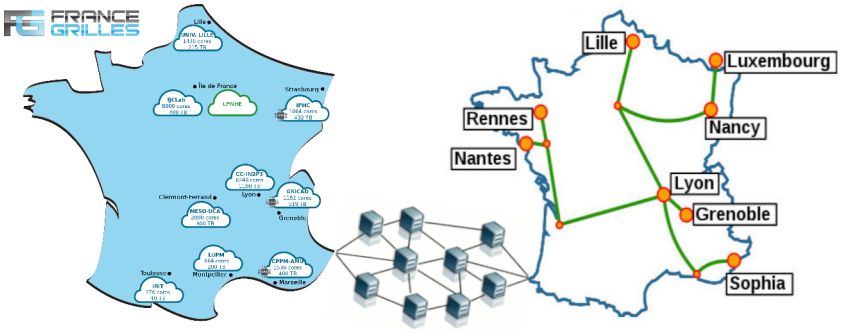
Nous commençons à être bien avancé dans le domaine du Cloud Computing avec de bons résultats qui restent à consolider par plusieurs autres simulations sur de nouveaux problèmes.

Pour les projets à court et moyen terme, nous désirons donc analyser le comportement sur architecture cloud pour de nouveaux problèmes permettant de mesurer les performances pour d'autres type de nonlinéarités (navier stokes, problèmes complémentaires, problème pseudo linéaire univoques).

De plus, toujours dans le cadre des expérimentations sur le cloud, nous proposons d'utiliser les différentes couches PAAS et SAAS du cloud en complément de IAAS afin de voir si cela a une influence sur les résultats synchrones et asynchrones.

Pour nos expérimentations, nous avons toujours considéré que dans le cas des simulations, les machines n'auraient aucune faille. Désormais, nous pensons qu'il est primordial de trouver des moyens de gérer les pannes temporaires afin de pouvoir faire une reprise des calculs et rendre nos algorithmes tolérants aux fautes. Il serait intéressant de mesurer les impacts d'une panne sur une simulation asynchrone puisque dans les principes, cette dernière continue à poursuivre ses calculs.

Enfin, il serait intéressant d'augmenter l'hétérogénéité des plateformes en liant par exemple la plateforme Grid'5000 et FG Cloud.



Liaison FG Cloud et Grid'5000

Pour les perspectives de recherche à plus long terme, nous envisageons de travailler sur l'adaptabilité dynamique des algorithmes parallèles. En effet, la répartition dynamique des charges de travail en fonction de la disponibilité des ressources et la modification en temps réel de la granularité des tâches en ajustant la taille des données traitées en fonction de la charge pourrait être intéressante.

Concernant, l'adaptabilité dynamique pour les algorithmes parallèles asynchrones, la gestion des communications sur les différents nœuds ou processeur avec ajustement des vitesses de communication ou bien la réduction des échanges de données en fonc-

tion des variations de charge pourrait fluidifier les expérimentations.

Un autre point sur lequel on pourrait travailler serait de pouvoir ajuster la fréquence des mises à jour des modèles en fonction de la convergence ou de la stabilité des algorithmes comme le Stochastic Gradient Descent (SGD) dans l'apprentissage machine et ainsi permettre au calcul parallèle de trouver sa place dans d'autres domaines comme l'intelligence artificielle. En effet certains algorithmes de machine learning ou de deep learning pourraient bénéficier d'optimisation dans l'efficacité en temps grâce à des parallélisations à gros grains ou bien à grains fins sachant que la programmation hétérogène peut être une piste prometteuse pour accélérer les calculs.

L'objectif final étant que ces adaptabilités puissent maximiser l'utilisation des ressources tout en minimisant les coûts associés aux communications dans un contexte de verdissement numérique et de réduction de l'empreinte environnementale des technologies.

Chapitre 5

Articles publiés

Pour informations :

- CiteScore et Impact Factor (Sources CiteScore et Journal Searches)
  - Advances in Engineering Software - CiteScore : 7.0 et Impact Factor : 4.255
  - Journal of Computational Science - CiteScore : 5.9 et Impact Factor : 3.817
  - Numerical Algorithms - CiteScore : 4.4 et Impact Factor : 2.37
  - Information Processing Letters - CiteScore : 2.0 et Impact Factor : 0.851

- Top 4 des articles les plus cités (Source Google Scholar et Ad Scientific Index)
  - h-index : 7 - Total citation : 151
  - A coarse-grained multicomputer algorithm for the longest common subsequence problem - 32 citations
  - A work-optimal cgm algorithm for the longest increasing subsequence problem - 27 citations
  - Asynchronous Schwarz methods applied to constrained mechanical structures in grid environment - 21 citations
    Asynchronous peer-to-peer distributed computing for financial applications - 10 citations

- Encadrements
  - Co-encadrement d'une future thèse (ENSEEIHT-IRIT - Mohammed Amine Rahhali) - taux d'encadrement 60 %
  - Co-encadrement d'un post-doctorat (Skikda University | LAMAHIS - Ghania Khenniche) - taux d'encadrement 25 %
  - Co-encadrement d'un doctorat (ENSEEIHT-IRIT en partenariat avec l'Entreprise SEGULA Technologies (CIFRE) - Vincent Partimbene) - taux d'encadrement 20 %
  - Co-encadrement d'un doctorat (Université Badji Mokhtar Annaba - Ghania Khenniche) - taux d'encadrement 20 %

Certains articles ont été réalisés dans le cadre de la préparation du doctorat

— de Ghania Khenniche : "Behavior of parallel two-stage method for the simulation of steel solidification in continuous casting, Advances in Engineering Software, Elsevier, 2019, 131, pp.116-142" - voir ADES_2019 et [68].
— de Vincent Partimbene : "Asynchronous multi-splitting method for linear and pseudo-linear problems, Advances in Engineering Software, Elsevier, 2019, 133, pp.76-95" - voir ADES_2019b et [119].
— et de la préparation de celle de Mohammed Amine Rahhali : "Simulation of parallel synchronous and asynchronous algorithms on cloud computing for PDE. Advances in Engineering Software, Elsevier, 2023" - voir ADES_2023 et [122].

D'autres articles sont consacrés

— au couplage de méthodes parallèles asynchrones de multi-splitting avec des méthodes de Kryvlov pour résoudre des problèmes 3D d'évolution pseudo-linéaire : "Coupling parallel asynchronous multisplitting methods with Krylov methods to solve pseudo-linear evolution 3D problems. Journal of Computational Science, Elsevier, 2021, 51" - voir JOCS_2021 et [70].
— à la résolution de problèmes pseudo-linéaires univalents et multivalués à l'aide de méthodes parallèles asynchrones multisplitting combinées à des méthodes de Krylov : "Solution of univalued and multivalued pseudo-linear problems using parallel asynchronous multisplitting methods combined with Krylov methods. Advances in Engineering Software, Elsevier, 2020, 153" - voir ADES_2020 et [69].
— à la solution d'un problème de contraintes unilatérales sur grille : "Grid solution of problem with unilateral constraints. Numerical Algorithms, Springer Verlag, 2017, 75 (4), pp.879-908" - voir NUMALGO_2017 et [36].
— aux méthodes de Schwarz asynchrones appliquées aux structures mécaniques contraintes dans un environnement de grille : "Asynchronous Schwarz methods applied to constrained mechanical structures in grid environment. Advances in Engineering Software, Elsevier, 2014, 74, pp.1–15" - voir ADES_2014 et [34].
— au calcul asynchrone sur grille pour la simulation du problème couplé de l'électrophorèse en 3D : "Asynchronous grid computing for the simulation of the 3D electrophoresis coupled problem. Advances in Engineering Software, Elsevier, 2013, 60–61, pp.111–121" - voir ADES_2013 et [35].

Lors de ma thèse orientée algorithmique parallèle du texte, on peut citer l'article :

— Thierry Garcia, David Semé. A Coarse-Grained Multicomputer algorithm for the detection of repetitions. Information Processing Letters, Elsevier, 2005, 93 (6), pp.307–313.

Parmi les proceedings (voir les publications dans le CV en section 6 et ci-dessous IEEE et ICCSA), certains ont été réalisés dans le cadre d'une ANR et d'autres dans le cadre de collaborations nationales et internationales avec le LAAS, l'université de Franche-Comté et l'université d'Annaba.

5.1 Article 1

Advances in Engineering Software 186 (2023) 103550

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Performances analysis of parallel asynchronous and synchronous algorithms on cloud architecture for PDE

M.A. Rahhali [a], T. Garcia [b,c,*], P. Spiteri [c]

[a] ENGIE, La Défense Ile de France, France
[b] CY Paris University, CY Tech Campus de Pau 2 boulevard Lucien Favre CS 77563 64075 Pau Cedex, France
[c] University of Toulouse, IRIT-INPT-ENSEEIHT, 2 rue Charles Camichel, B.P. 7122, 31071 Toulouse - Cedex 7, France

ARTICLE INFO

Keywords:
High performance computing
Parallel computing
Cloud computing
Asynchronous iterations
Asynchronous algorithms
Iterative methods

ABSTRACT

In order to solve discretized stationary linear or non linear diffusion and convection–diffusion problems, the present paper deals with a comparison between the performances of parallel iterative synchronous or asynchronous algorithms implemented on a Cloud computing architecture with several sizes of the algebraic systems to be solved. The parallel algorithms are briefly and intuitively presented and analysis of their behavior is recalled. In particular, criteria of convergence are recalled and estimates of convergence speed are given. Implementation on cloud architectures are briefly described. Finally, for each kind of problem, the performance of the target algorithms on cloud architectures are analyzed.

1. Introduction

The discretization of linear or non linear boundary value problems leads to large and generally sparse linear or non linear algebraic systems. In order to solve these algebraic systems, we are currently parallelizing the resolution algorithms in order to significantly reduce the computation time. Among the possible algorithms we focus in the present paper on the parallel relaxation methods applied to the resolution of linear and non linear algebraic systems. A large advantage of the relaxation methods is that synchronous or asynchronous variants can be considered ; in the present work we will compare the performances of parallel iterative asynchronous methods with the synchronous ones ; previous studies have shown that asynchronous versions implemented on clusters or grids allowed to decrease the restitution time when there were many synchronizations between the computing processes. In other words, in this case the asynchronous versions strongly minimize the idle time of the processors. In these previous works (see for example [1]) from a computer sciences point of view we have also experimentally observed that asynchronous variants gave better results on grid-type architectures due to slow communications between processors because the bandwidth is shared by others machines. Indeed latencies penalize and affect the simulation results. Moreover, this effect is increased by the geographical distance between the machines and also, when modes are heterogeneous which is the case for example with computing grids or cloud computing. In the present paper we will then consider an implementation on cloud architecture and we will analyze the behavior of the two target algorithms on such cloud architecture applied to the solution of discretized boundary value problems. In Cloud Computing, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) represent three customer-focused layers of abstraction. This type of representation allows customers to use the infrastructure components without managing them themselves. Given the fact that this last type of architecture includes additional software layers, we have the intuition that the implementation of iterative asynchronous methods will give better results in terms of work restitution time than the synchronous version.

Note that, although the relaxation methods are efficient in the case where the matrix of the algebraic system to be solved is not symmetrical, for the solution of large algebraic systems, we can solve the subproblems by other methods than this last algorithm. For example we refer the reader to Refs. [2,3] in which in order to solve the convection–diffusion problem in the case where the linear part is perturbed by à diagonal monotone operator, we implemented a parallel synchronous algorithm compared to the asynchronous one where the sub-problems are solved by the Krylov method [4]. In such implementation the parallel asynchronous method is more efficient than the synchronous one on a grid architecture.

The main goal of this study is then to compare the performance of asynchronous parallel algorithms implemented on cloud architecture with that of synchronous parallel algorithms, and to see the influence of the additional software layers present in the systems implemented in this type of architecture, which is well suited to solving very large-scale algebraic systems.

* Corresponding author at: University of Toulouse, IRIT-INPT-ENSEEIHT, 2 rue Charles Camichel, B.P. 7122, 31071 Toulouse - Cedex 7, France.
E-mail addresses: marahhali@outlook.com (M.A. Rahhali), thierry.garcia@irit.fr (T. Garcia), pierre.spiteri@enseeiht.fr (P. Spiteri).

https://doi.org/10.1016/j.advengsoft.2023.103550
Received 22 December 2022; Received in revised form 18 October 2023; Accepted 18 October 2023
Available online 3 November 2023

On the other hand, we are currently assisting in the development of new asynchronous domain decomposition methods including successfully coarse space into existing asynchronous Schwarz-type domain decomposition methods. For example in [5,6] are presented recent advances of scalable asynchronous domain decomposition methods ; we refer to these references for further information (see also [7] for an interesting contribution concerning asynchronous Schwarz solvers implemented on GPU's). In [5] the authors consider the use of asynchronous communications based on one-sided MPI (Message Passing Interface) primitive, previously investigated, for instance in [8], in the context of domain decomposition solvers and particularly a scalable additive two-level Schwarz method with comparison of synchronous and asynchronous variants of parallel solvers. In [6] a multiplicative variant of the additive coarse-space method is studied with implementation aspects and parallel numerical simulations showing the very good performances obtained when the asynchronous two-level method is used. Moreover in the Ref. [9] the authors study also the behavior of a scalable non-overlapping asynchronous domain decomposition method constituted by the classical primal Schur complement method where the interface problem is solved by using a multisplitting relaxation algorithm based on a weighted additive Schwarz preconditioner. According to the conclusion inferred in this paper such method outperforms significantly the two-level asynchronous restricted additive Schwarz method. In Ref. [10] the authors define also a scalable nonlinear application of the Restricted Additive Schwarz (RAS) method, which can be algebraically overlapping or not, and the original paper [11] is a clear indication of the RAS method. This paper extends the studies undertaken in the linear case [5] to the non-linear case where each local problem in each subdomain is solved with overlap, but the updates communicated to the other local subproblems are those corresponding to subdomains without overlap for contributing to the next iterate. To the question of knowing if the points of view developed in Refs. [5,6,9,10] are applicable or not to the problems considered or to come in our project of article, it seems that the answer is affirmative in the linear case developed in [5,6,9] and in the nonlinear case considered in [10] nevertheless except perhaps to the case of multivalued problems in the case where the solution is subjected to constraints of type inequality in which case the monotony plays an essential role; in this case the studies presented in [5,6,9,10] must find an adaptation not obvious to propose, but the point of view developed in Ref. [1] and related works allows to solve very simply this kind of nonlinearity by using monotony notions in Banach spaces.

The paper is organized as follows. Sections 2 and 3 deal with the presentation of parallel synchronous and asynchronous algorithms. In this presentation, for clarification, we avoid the use of an abstract formalism. We prefer illustrate simply the difference between synchronous and asynchronous parallel algorithms and indicate what advantage can be taken of the latter algorithm to reduce the restitution time of the computational work. For this parallel computation method, we first indicate how the problem is decomposed into interconnected subproblems, which leads us to present the subdomain methods with and without overlapping between the subdomains. We then describe synchronous and asynchronous variants of the parallel relaxation methods as simply as possible and avoiding a mathematical formalism. As this is an iterative method, starting from a linear problem and also from various pseudo-linear problem, we then indicate convergence criteria which are illustrated in case of each model problem to then lead to more general problems and extend the results presented in a simple case. Finally, the difficult issue of stopping asynchronous iterations is addressed from both a computational and a numerical point of view. For more details the reader is referred to the Ref. [1].

The Section 4 is relative to the implementation of parallel synchronous and asynchronous algorithms on cloud architecture which is a way of providing access to computer resources, characterized by its self-service availability, elasticity, openness, mutualization and pay-per-use; self-service and on-demand resources of storage capacity and computing power are based on the customer's needs. After having performed experiments with virtual machines in a previous study [12], the computational experiments were performed, for this new study, on France Grilles Cloud which offers users cloud services that allow on-demand computation, storage and networking, cluster instantiation, instantiation of web services dedicated to the scientific community and algorithm testing. Results of synchronous and asynchronous parallel simulation on cloud computing for the numerical solution of a linear diffusion problem and a linear convection–diffusion problem are presented and analyzed.

Finally a conclusion and prospects for further study complete this paper.

2. Formulation of parallel synchronous and asynchronous algorithms

As the first supercomputer capable of achieving Exaflop performance (i.e. 10^{18} operations per second) as been announced, researchers and practitioners of high performance computing are confronted with daunting challenges related to the efficient use of such massive computing power in order to accelerate numerical simulations in numerous domains (such as physics, chemistry, economy...) or applications in big data analytics and artificial intelligence. Modern supercomputers draw their power from extreme parallelism (up to millions of cores) and specialized processing units such as GPUs; this technological trends have a major impact on the design and development of numerical algorithms and software or fuel research in high performance computing with the premises of tackling problems which have been, so far, intractable.

Among the numerous numerical algorithms for solving large scale algebraic systems [4], we are especially interested in parallel relaxation methods. These algorithms are defined mathematically by associating a fixed point equation to the algebraic system to be solved. For example, let us consider first the solution of the following linear algebraic system

$$AU = G \tag{1}$$

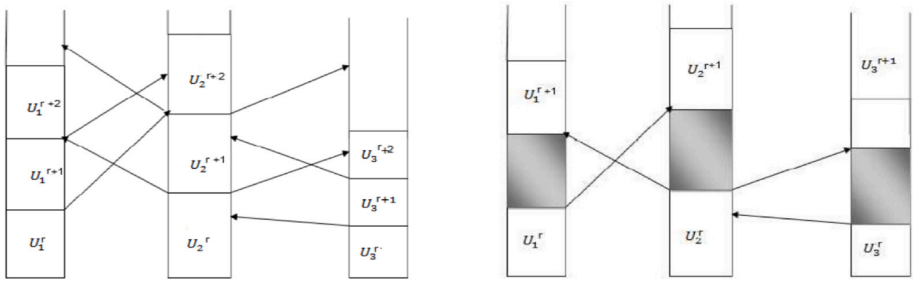
where A is a large sparse matrix generally derived from the discretization of a linear boundary value problem, G is a vector issued from the discretization and U is the unknown vector. Note that this kind of problem occurs when we have to solve elliptic, parabolic or hyperbolic second order boundaries values problems. Let us now associate to problem (1) the following fixed point problem

$$U = F(V) \tag{2}$$

with $F : D(F) \to D(F)$ where F applies from $D(F)$ to $D(F)$, the definition domain of F. For non-linear algebraic systems F can be defined implicitly or explicitly while in the linear case F is defined in an explicit way. For example in the linear case the parallel solution of problem (1) needs the decomposition of the problem in α interconnected linear subsystems associated to the block decomposition of the matrix A in large blocks as follows

$$A_{i,i}U_i = G_i - \sum_{j=1 j\neq i}^{\alpha} A_{i,j}V_j, i = 1, \ldots, \alpha \tag{3}$$

In order to obtain good performances of the parallel algorithms, each sub-problem is of sufficiently large size which allows to have a suitable granularity. This corresponds in fact to sub-domain methods and one can thus consider sub-domain methods without overlap and sub-domain methods with overlap, such as the classical Schwarz alternating method; such classification of sub-domain will be specified later. Let us also note that we can present all these methods in a unified framework constituted by the multisplitting methods (see [2,3]) where we perform a weighted average between various contracting fixed point mappings, each problem being solved in these two previous reference by Krylov methods [4].



(a) Behavior of parallel asynchronous iterations. (b) Behavior of parallel synchronous iterations.

Fig. 1. Behavior of asynchronous and synchronous parallel iterative relaxation algorithms.

In this kind of parallel methods of relaxation we distinguish two types of variants constituted on the one hand by synchronous parallel iterative relaxation methods and on the other hand by asynchronous parallel iterative relaxation methods corresponding respectively to two different modes of communication between the parallel processors.

Synchronous parallel iterative algorithms are computing methods in which the communications between the processors are synchronized at the end of each iteration. The synchronous parallel iterative method corresponds in fact to the most natural formulation of parallel algorithms and can be modeled by a Jacobi's like method. In this kind of method, when the load on each processor is not uniform or when the processors do not have the same performance, at each synchronization point the processing units must wait for the slowest processor. Consequently, given a heterogeneous distributed architecture, the idle times of the processing units will degrade the performance of these synchronous methods and the job restitution times will be penalized when a large lot of synchronizations is necessary to solve the target problem.

Asynchronous parallel iterative algorithms correspond to methods in which the communications between the processors are not synchronized at each iteration. Therefore, when a processing unit has finished its own calculations, it starts the next cycle again using the latest interaction data computed by the other processors and received during the previous cycle, without waiting for the arrival of new results delivered by the other processing units. So no expectation of new results computed by other units are necessary and the parallel computations are performed with available values computed and sended by other units. Thus no idle times will appear in this kind of communications. The asynchronous parallel algorithms are well modeled by introduction of delays in the formulation of the algorithm. The reader is referred to various papers describing formally the formulation of parallel asynchronous relaxation algorithms (see [1] - and for additional informations about our works see also [13] to [14]- and also for other contributions [15–22], etc.) ; note that the formulation of parallel synchronous relaxation algorithms appears like a particular case of the asynchronous when no delays are used in the iterative method.

Fig. 1 illustrate the behavior of synchronous and classical asynchronous parallel iterative relaxation algorithms and in the second case allow to view the disappearance of idle times.

Remark 1. However, there are asynchronous parallel methods defined with more flexible communications between the processors where the interaction data is used without a predetermined order (see for example [13,23]). Thus, the calculations are processed on each processor respecting the own rhythm of each processing unit and using the last available values calculated by the other processors. Depending on the frequency of updates of calculations between processors, the speed of convergence of these asynchronous parallel methods can be slower. But the big advantage lies in the fact that there is generally a reduction in the elapsed time to reach convergence.

The convergence of both synchronous or asynchronous parallel algorithms can be studied by various methods (see [1,24]). When the fixed point operator F associated to the problem to be solved is contracting [25], i.e.

$$\|F(U) - F(V)\| \leq \nu \|U - V\|, \forall\ V \in D(F), \tag{4}$$

where $0 \leq \nu < 1$, one obtains on the one hand convergence whatever be the decomposition of the problem into sub-problems and on the other hand the value of the contraction constant ν gives an estimate of the asymptotic speed of convergence as follows

$$\mathcal{R}_\infty = -\ln(\nu) \tag{5}$$

Note that depending on the sharpness of the decomposition we obtain smaller or larger contraction constants, their values then having an impact on the asymptotic speed of convergence. Note also that in both cases the convergence study of asynchronous relaxation method imply the convergence of synchronous relaxation method but, with a different asymptotic speed of convergence, since the speed of convergence in this kind of method depends on the frequency of updates.

Example 1. For example let us consider the numerical solution of the following elliptic boundary value problem defined in the unit cube $\Omega = [0,1] \times [0,1] \times [0,1]$ and equipped with the homogeneous Dirichlet boundary conditions

$$\begin{cases} -\Delta u(x,y,z) + q.u(x,y,z) = g(x,y,z) \text{ everywhere in } \Omega, q \geq 0 \\ u(x,y,z) = 0 \text{ everywhere in } \partial\Omega, \end{cases} \tag{6}$$

where $\partial\Omega$ is the boundary of Ω; then the discretization of problem (6) by classical finite difference method leads to the solution of the following equations. $1 \leq i,j,k \leq n$

$$\begin{cases} \frac{-u_{i+1,j,k}+2.u_{i,j,k}-u_{i-1,j,k}}{h^2} + \frac{-u_{i,j+1,k}+2.u_{i,j,k}-u_{i,j-1,k}}{h^2} + \frac{-u_{i,j,k+1}+2.u_{i,j,k}-u_{i,j,k-1}}{h^2} \\ \qquad +q.u_{i,j,k} = g_{i,j,k}, 1 \leq i,j,k \leq n \\ u_{0,j,k} = u_{n+1,j,k} = u_{i,0,k} = u_{i,n+1,k} = u_{i,j,0} = u_{i,j,n+1} = 0, 1 \leq i,j,k \leq n \end{cases} \tag{7}$$

where h is the uniform discretization step in space defined by $h = \frac{1.0}{n+1}$ and when $x_i = i.h$ (respectively $y_j = j.h$, $z_k = k.h$), $u_{i,j,k}$ is an approximation of $u(x_i, y_j, z_k)$. Then, when synchronous or asynchronous parallel relaxation subdomain methods without overlapping between the subdomains are used for the numerical solution of problem (6), this kind of method corresponding in fact to classical non-overlapping large-block relaxation methods, which themselves group together several adjacent blocks of the discretization matrix, according to the results set up for example in [1] the smaller contraction constant ν is given by

$$\nu = 1 - \frac{12}{6 + q.h^2}.\sin^2(\frac{\pi}{2}.h) - \frac{q.h^2}{6 + q.h^2} \tag{8}$$

and parallel synchronous and asynchronous iterative relaxation algorithms converge for all decomposition of the problem in α subproblems; in this case, when ϵ is small taking account of $\ln(1-\epsilon) \approx \epsilon$ the asymptotic speed of convergence is given by

$$\mathcal{R}_\infty \approx \frac{3.\pi^2+q}{6+q.h^2}.h^2 \tag{9}$$

since h is in general very small. Moreover note that when $q=0$ then the contraction constant ν is given by

$$\nu = 1 - 2.\sin^2(\frac{\pi}{2}.h) = cos(\pi.h).$$

and (9) reduces to the following expression

$$\mathcal{R}_\infty \approx \frac{\pi^2}{2}.h^2 \tag{10}$$

In the treatment of the previous example, estimates are obtained since many classical results concerning the eigenvalues of the discretization matrix of the Laplacian defined in the unit cube are known. When we are no longer in such an ideal situation we can however obtain estimates of the asymptotic speed of convergence. Indeed according to the results presented in [1] or [26] the convergence of parallel synchronous or asynchronous algorithms is obtained when the spectral radius of the Jacobi matrix $J = Id - D^{-1}A$ associated to A is less than one (see for example [26]), where Id is the identity matrix and D denotes the diagonal of A; then, classically (see [27,28]), in the general case we can also obtain an estimate of the contraction constant ν given by (see [1])

$$\rho(J) \leq \nu \leq \max_{1\leq i\leq M}\Big(\sum_{j=1,j\neq i}^{M}\big(\frac{|a_{i,j}|}{|a_{i,i}|}\big)\Big) \tag{11}$$

where M denotes the size of A. Such overestimation of ν corresponds in fact to a dominance diagonal property. More precisely (see [1,14,25]) for the numerical solution of discretized boundary value problems both parallel relaxation algorithms converge if the diagonal entries $a_{i,i}$ of A are strictly positive, the off-diagonal entries $a_{i,j} \leq 0$ and A is strictly or irreducibly diagonally dominant (i.e. A is an M-matrix - see [27,28]).

Remark 2. Note that the previous estimation applied to the problem (6) leads to

$$\rho(J) \leq \nu \leq \frac{6.0}{6.0+q.h^2} \approx 1.0 - \frac{q.h^2}{6.0};$$

this estimate is thus less sharp than the one obtained in (8)

Remark 3. Note that in estimates (11) the majoration takes into account the influence of rounding errors. In exact arithmetic we would have $\rho(J) = \nu$.

We concentrate here on parallel Schwarz alternating methods, which are based on overlapping subdomains.

Consider now the case where the linear system (1) derived from the discretization of problem (6) is solved by parallel asynchronous Schwarz alternating method ; this last method is also well suited to parallel computing. Recall that in this case problem (6) can be decomposed into α sub-problems as follows : for $i = 1,\dots,\alpha$,

$$\begin{cases} -\Delta u_i + q u_i = g_i, \text{ everywhere in } \Omega_i, \\ u_{i_{/\Gamma_i}} = 0, \\ u_{i_{/\gamma_i^1}} = u_{i-1_{/\gamma_i^1}} \text{ for } 2 \leq i \leq \alpha, \\ u_{i_{/\gamma_i^2}} = u_{i+1_{/\gamma_i^2}} \text{ for } 1 \leq i \leq \alpha - 1, \end{cases} \tag{12}$$

where u_i and g_i, respectively, are the restriction of u and g, respectively, to Ω_i, $\Omega = \bigcup_{i=1}^{\alpha} \Omega_i$, $\Omega_i \bigcap \Omega_{i+1} \neq \emptyset, i \in \{1,\dots,\alpha-1\}$, $\gamma_i^1 = \partial\Omega_i \bigcap \Omega_{i-1}, i \in \{2,\dots,\alpha\}$, $\gamma_i^2 = \partial\Omega_i \bigcap \Omega_{i+1}, i \in \{1,\dots,\alpha-1\}$ and $\Gamma_i = \partial\Omega_i \bigcap \partial\Omega, i \in \{1,\dots,\alpha\}$ (see Fig. 2).

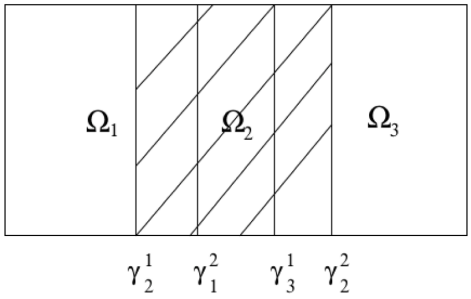


Fig. 2. Example of decomposition of Ω with 3 subdomains.

The decomposition (12) corresponds to a basic overlapping subdomain decomposition describing the principle of the method, whereby u_i is computed using the restriction of its immediate neighbors u_{i-1} and u_{i+1} respectively, on γ_i^1 and γ_i^2 respectively. In the sequential case, the scheme of computation corresponds exactly to a multiplicative Schwarz scheme. In the parallel case, the Schwarz alternating method can be combined with a synchronous or an asynchronous iterative scheme of computation with flexible communication in order to be as close as possible to a multiplicative scheme.

Thanks to a result stated by D.J. Evans and W. Deren [29] the augmentation process due to the Schwarz alternating method transforms the M-matrix A, i.e. a matrix where the diagonal entries $a_{i,i}$ are strictly positive, the off-diagonal entries $a_{i,j} \leq 0$ and in addition A is strictly or irreducibly diagonally dominant [27,28], into an M-matrix $\widetilde{A}$. Thus we are in the same framework than the ones previously considered in the case of subdomain methods without overlap; consequently the convergence criteria of synchronous or asynchronous parallel Schwarz alternating methods are identical to those established in the case of subdomain methods without overlap and if A is an M-matrix these methods are again convergent. Note that due to the overlapping between the subdomains generally the speed of convergence of the Schwarz alternating method is faster than the one corresponding to the subdomain method without overlap.

Example 2. Consider now the numerical solution of the following linear convection–diffusion problem

$$\begin{cases} -\Delta u + a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} + c\frac{\partial u}{\partial z} + qu = g, \text{ in } \Omega, \\ u(x,y,z) = 0 \text{ everywhere in } \partial\Omega, \end{cases} \tag{13}$$

where $q \geq 0$,

For the sake of simplicity, we assume that the discretization grid of the domain Ω is uniform. As previously in the sequel h denotes the discretization step-size. We assume that the columns of the discretization grid are numbered naturally in lexicographical order. The discretization of the operators which occur in problem (13) is made according to the following rules: the Laplacian is discretized via the classical seven points scheme presented in Example 1 and the first derivatives are discretized as follows according to the sign of a, b and c

$$\frac{\partial u}{\partial x} = \begin{cases} \frac{u(x,y,z)-u(x-h,y,z)}{h} + \mathcal{O}(h), \text{ if } a > 0, \\ \frac{u(x+h,y,z)-u(x,y,z)}{h} + \mathcal{O}(h), \text{ if } a < 0. \end{cases} \tag{14}$$

and accordingly for the derivative with respect of y and z.

Let A denote the discretization matrix of the problem (13). If q is strictly positive, then regardless the sign of a, b and c it follows from (14) that off-diagonal entries of matrix A are non-positive and diagonal entries of A are positive. Moreover, the matrix A is strictly diagonally dominant; then we are in the same framework presented in Example 1.

If $q = 0$, then we can show that the matrix A is diagonally dominant. Moreover, by using the characterization of irreducible matrices (see [28]) we can verify that the matrix A is irreducibly diagonally dominant. Thus the framework presented in Example 1 is again satisfied.

Consider now a red-black ordering of the grid points and let $\hat{A}$ be the corresponding discretization matrix derived from A by a permutation which preserves the sign of the entries. We consider the former discretization scheme; if q is strictly positive, then, the matrix $\hat{A}$ is strictly diagonally dominant; if $q = 0$, then we can show analogously that the matrix $\hat{A}$ is irreducibly diagonally dominant. Thus, in both cases $\hat{A}$ satisfy the theoretical framework presented in this section.

Remark 4. Finally, note that under realistic hypotheses, the finite element discretization matrix and the finite volume discretization matrix occurring in some analogous linear partial differential equations satisfy the theoretical framework presented below.

For simplicity, let us consider first the case where $\Omega = [0,1] \times [0,1] \times [0,1]$ and let us compute the contraction constant ν for various values of a, b and c in the case of the subdomain method without overlapping. Recall that the eigenvalues of the discretization matrix, an heptadiagonal matrice with classically a block structure, are given for $i, j, k = 1, \ldots, n$, by

$$\lambda_{i,jk}(A) = \mu - 2.\sqrt{\theta.\beta}.cos(\frac{i.\pi}{n+1}) - 2.\sqrt{\delta.\gamma}.cos(\frac{j.\pi}{n+1}) - 2.\sqrt{\zeta.\eta}.cos(\frac{k.\pi}{n+1}),$$

where μ are the diagonal entries of A, θ and β are the codiagonal entries, δ and γ are the closest entries to the diagonal blocks and ζ and η are the entries of the most extreme blocks.

For example if a, b and c are positive real numbers and for decentered discretization of the first derivative given by (14), the spectral radius is given by

$$\nu = 2.\frac{\sqrt{1+a.h} + \sqrt{1+b.h} + \sqrt{1+c.h}}{6 + q.h^2 + a.h + b.h + c.h}.cos(\pi.h). \tag{15}$$

since $\mu = 6.0 + q.h^2 + a.h + b.h + c.h$ and according to the numbering of the grid points, for example $\theta = \delta = \zeta = -1.0$ and $\beta = -(1.0 + a.h)$, $\gamma = -(1.0 + b.h)$ and $\eta = -(1.0 + c.h)$. Then we can verify that in this context $0 < \nu < 1.0$

Remark 5. Note that estimate of ν given by (15) is very sharp. Il we use the estimate given by (11) we obtain the following result

$$\nu \leq \tilde{\nu} = \frac{6.0 + a.h + b.h + c.h}{6 + q.h^2 + a.h + b.h + c.h}$$

For other combinations of signs of the coefficients a, b and c we obtain also easily that $0 < \nu < 1.0$. This property again comes from the fact that the matrix A has positive diagonal entries and negative or zero off-diagonal entries and that moreover it is either strictly diagonal dominant or irreducibly diagonal dominant. Note that such important property is again verified when Ω is of any form.

In a similar way to our approach during the study of Example 1, we can also solve the convection–diffusion equation by the alternating Schwarz method. If problem (13) is decomposed into α sub-problems then for $i = 1, \ldots, \alpha$, using the same previous notations and the same principle of presentation of the algorithm, this method is defined by

$$\begin{cases} -\Delta u_i + a\frac{\partial u_i}{\partial x} + b\frac{\partial u_i}{\partial y} + c\frac{\partial u_i}{\partial z} + q u_i = g_i, \text{ everywhere in } \Omega_i, \\ u_{i_{/\Gamma_i}} = 0, \\ u_{i_{/\gamma_i^1}} = u_{i-1_{/\gamma_i^1}} \text{ for } 2 \leq i \leq \alpha, \\ u_{i_{/\gamma_i^2}} = u_{i+1_{/\gamma_i^2}} \text{ for } 1 \leq i \leq \alpha - 1, \end{cases} \tag{16}$$

Once again, the decomposition (16) corresponds to an overlapping subdomain decomposition, whereby u_i is computed using the restriction of its immediate neighbors u_{i-1} and u_{i+1} respectively, on γ_i^1 and γ_i^2 respectively, and the same remarks like the ones formulated in the presentation of Example 1 can be formulated.

The effectiveness of domain decomposition methods is well known for boundary value problems. These methods are also well suited to parallel computing (see [30]).

Note that in the general case when Ω is of any form, since once again the augmentation process do not alter the properties of the discretization matrices, using the result stated by D.J. Evans and W. Deren [29], the criteria viewed in the case of the problem of diffusion are still valid in the present case, i.e. when the diagonal entries are strictly positive, the off-diagonal entries are non positive and the discretization matrix is strictly or irreducibly diagonal dominant.

Remark 6. The previous schemes correspond to decentered finite difference scheme for the discretization of the first derivative; such decentered scheme allows to obtain strongly diagonal dominant matrices, which has a strong impact on the convergence speed. Note that it is also possible to consider a centered finite difference scheme defined by the following difference equation corresponding to an approximation of $\frac{\partial u}{\partial x}$

$$\frac{\partial u}{\partial x} \approx \frac{u(x+h, y, z) - u(x-h, y, z)}{2.h};$$

use of such centered scheme cannot provide any guarantee that the discretization matrix satisfies the required properties ensuring the convergence of the iterative process, especially when the convection coefficients, i.e. the coefficients of the first derivative, are dominant. In this case the discretization matrix A is not necessarily diagonal dominant and for large values of the coefficients of convection, off-entries of A are not necessarily negative. Such properties of diagonal dominance and off-diagonal entries negative are satisfied with very restrictive conditions verified by the convection coefficients, i.e.

$$|a| + |b| + |c| \leq \frac{6}{h} + q.h$$

and

$$|a| \leq \frac{2}{h}\ ;\ |b| \leq \frac{2}{h}\ ;\ |c| \leq \frac{2}{h}.$$

In fact the previous restrictive conditions are not necessary. However according to the results presented in [1] the use of such centered schemes can be used and leads to the conclusion that parallel synchronous and asynchronous iterative methods converge. Such convergence property arises from the fact that the discretization matrix A is the sum of a symmetric defined positive matrix arising from the discretization of the Laplacian and an anti-symmetric matrix arising from the discretization of the convection terms. Then, using appropriate norm for analyze the convergence, i.e. in this case the euclidean norm, it can be verified very easily that the contraction constant ν is given by (8) in the case of the resolution of the convection–diffusion problem ; thus in this framework only the diffusive part has an influence on the convergence speed, which is a little bit restrictive. Note that from a numerical point of view the use of decentered schemes such as (14) is recommended to avoid boundary layer problems. Moreover the speed of convergence of the iterative process is increased.

3. Numerical solution of nonlinear convection–diffusion equation

3.1. The general situation

Note that parallel asynchronous and synchronous iterative relaxation algorithms can also be applied for the solution of the following large pseudo-linear single-valued algebraic systems

$$AU + \Phi(U) = G \tag{17}$$

corresponding to a linear system perturbed by a diagonal increasing operator (or more generally monotone operator). Another situation

corresponds to the case where the solution U is subject to the following constraints of inequalities type

$$U_{min} \leq U \text{ or } U_{min} \leq U \leq U_{max} \text{ or } U \leq U_{max}; \tag{18}$$

in this last case the solution U satisfies the following large pseudo-linear multi-valued algebraic systems

$$AU + \partial\Psi(U) - G \ni 0 \tag{19}$$

where Ψ is the characteristic function defining the convex set of constraints and $\partial\Psi(U)$ is the sub-differential of Ψ corresponding in fact to a weak derivative of Ψ; classically the mapping $U \rightarrow \partial\Psi(U)$ is still a diagonal monotone operator. Considering this property of monotony of the operators $\Phi(U)$ and $\partial\Psi(U)$ the convergence criteria of the synchronous and asynchronous parallel algorithms previously stated in the linear case are still valid for the solution of the problems (17) and (19); therefore, once again, if the diagonal entries $a_{i,i}$ of A are strictly positive, the off-diagonal entries $a_{i,j} \leq 0$ and A is strictly or irreducibly diagonally dominant then for the resolution of previous problems (17) and (19) the both parallel relaxation algorithms converge and the estimate (11) of the contraction constant ν is at least still valid for the previous two problem and are still valid when subdomain methods without overlapping are implemented.

More generally for the Schwarz alternating method the criteria of convergence are identical to those presented in Section 2.

Remark 7. In fact the Schwarz alternating method is a particular case of the multi-splitting method and the convergence criteria mentioned above are still valid (see [2,3]).

Remark 8. Note also, in nonlinear framework, that convergence can be analyzed by partial ordering technique related to the use of the discrete maximum principle. In this case provided that the iterative process is initialized by an initial data U^0 (respectively V^0) such that for the problem (17) only we have $AU^0 + \Phi(U^0) - G \geq 0$ (respectively $AV^0 + \Phi(V^0) - G \leq 0$) we generate a sequence U^p decreasing since U^0 (respectively V^p increasing since V^0). The convergence criteria are identical to the previous ones and concern as well as the signs of the coefficients of the matrix A and the diagonal dominance properties. However, this type of analysis does not allow to estimate the speed of convergence of the iterations, contrary to what was possible in the case of an analysis by contraction techniques previously presented. Note that to our knowledge the analysis of the same algorithms by partial order techniques for the problem (19) is an open problem.

The previous methods being iterative in nature must be stopped when the iterated values are stabilized. They are thus completed by a test of stopping the iterations. In the case of synchronous parallel algorithms, this iteration stopping test is identical to the one implemented in the sequential case. For asynchronous parallel algorithms the iteration stopping test is much more delicate to implement. Thus, in this latter case we will distinguish the computer science approach and the numerical analysis approach and to stop the iterative process we have to combine these two approaches.

Concerning the computer science approach we refer to several works concerning such stopping criteria for example [31–36].

In our implementations a stopping test of the iterations is considered from the computer sciences point of view ; thus we have used the circulation of a token corresponding to the simplest implementation of stopping test but coupled as we will see below by a test derived from numerical analysis considerations. According to [37,38] in this case of parallel asynchronous methods we consider a decentralized algorithm where convergence is achieved in this way. Each processor updates the components of the iterate vector associated with each one's own block and computes the residual norm attached to this block in order to participate to the convergence detection. Convergence occurs when a given predicate on a global state is true. A usual predicate corresponds to the fact that the iterate vector generated by the asynchronous iterative algorithm is sufficiently close to the solution of the problem (see [19], page 580); thus, on every process, terminaison detection is obtained when the norm of the local residue remains under a given threshold after two successive updates of the component (see [37,38]). Due to the termination and the detection of the global state of the processes, the implementation of each variant of parallel asynchronous method is then more complex than the synchronous one.

In the case of asynchronous iterations, point to point communications between two processes have been implemented using nonblocking send and receive communications. A test function is used in order to allow any processor to continue the computations without having to wait for the completion of any pending send or receive operations. Idle times due to synchronizations in message passing are suppressed in this way. On the other hand, the parallel synchronous iterative schemes are implemented by using blocking send and receive communications. One has to take care about the deadlock issue when implementing synchronous communications using blocking communications. For more details concerning the implementation and the convergence detection of the considered parallel asynchronous methods, the reader is referred to [38].

Concerning the numerical analysis approach for an efficient terminaison it can be proved that the successive iterates are located in nested sets centered in the solution (see [19,39,40]), which allows to implement efficient numerical stopping tests insofar as one can give an estimate of the diameters of each nested set and stop the iterative process when all diameters are smaller than a given tolerance. Consequently the coupling of the stopping tests derived from computer science and numerical analysis considerations allows to obtain an efficient terminaison of the parallel asynchronous iteration. In previous studies (see [1–3]) we have implemented these methods on grids constituted by heterogeneous and distant machines; we could observe that asynchronous algorithms were very efficient when there was a lot of synchronization between the processors.

3.2. Numerical solution of nonlinear convection–diffusion equation

We turn now to the case where nonlinearities are defined in the domain Ω. As nonlinearities we will consider the following linear problems perturbed by an increasing diagonal operator or more generally monotone. In agreement with the results presented in [1], considering the monotony of the latter operator the convergence criteria will not change and in particular the speed of convergence will be of the same order for linear and nonlinear problems; indeed if we study the convergence by contraction techniques, the contraction constant ν occurring in (4) is the same as in the linear case. So, in both cases, the performances and the behavior of the parallel algorithms must be substantially similar whether the communications between the processors are synchronous or asynchronous, the speed of convergence being frequency dependent of the updates.

The general model can be given as follows

$$\begin{cases} -\Delta u + a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} + c\frac{\partial u}{\partial z} + qu + \varphi(u) = g, \text{ in } \Omega, \\ B.C., \end{cases} \tag{20}$$

where $q \geq 0$, g is a summable square function in Ω, $\varphi : u \rightarrow \varphi(u)$ is a continuous, nondecreasing function and B.C. represents a classical boundary condition, i.e. Dirichlet, Neumann, Robin or mixed.

Concerning the nonlinear part, we consider different types of nonlinear convection–diffusion problems where nonlinearities arise on the boundary or in the domain.

The following nonlinear functions: $\varphi(u) = e^{\alpha u}$, with $\alpha > 0$ or $\varphi(u) = Log(\beta + \delta u)$, with $\delta > 0$ and a suitable sign for β, such that $\beta + \delta u > 0$, can be considered. Then we have to solve an algebraic system like (17), where Φ is a diagonal operator derived from the discretization of φ; according to the properties of φ then Φ is a monotone diagonal increasing mapping. Consequently we are again in the framework

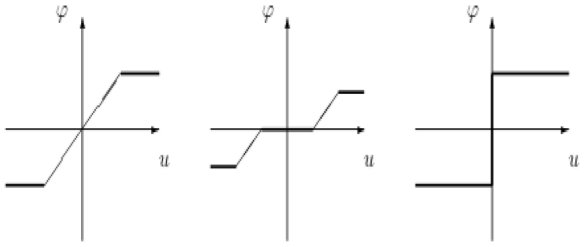


Fig. 3. Different graphs for φ..

presented in Section 2 and the convergence of parallel synchronous or asynchronous relaxation methods is verified with the same speed of convergence since the properties of the mapping $\varphi(u)$, i.e. the property of monotony, do not change the estimate of convergence rate.

Remark 9. The numerical solution of problem (20) where the nonlinearity occurs on the whole domain Ω is done using a Newton method provided that the application $u \to \varphi(u)$ is everywhere derivable on Ω. We refer to [2,3] where tests of synchronous parallel resolution compared to their asynchronous counterparts are presented and implemented on GRID'5000.

We can also consider nonlinear convection–diffusion problems where nonlinearities arise on the boundary of the domain. This kind of problem occurs, for example, in the following boundary temperature control problem

$$\begin{cases} -\Delta u + a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} + c\frac{\partial u}{\partial z} + qu = g \text{ everywhere in } \Omega, \\ \frac{\partial u}{\partial n} + \varphi(u) = 0 \text{ on } \Gamma_d \text{ and } u = 0 \text{ on } \partial\Omega - \Gamma_d, \end{cases} \tag{21}$$

where $\Omega \subset \mathbb{R}^3$, $q \geq 0$, $\Gamma_d \subset \partial\Omega$, g is a summable square function in Ω and $\varphi : u \to \varphi(u)$ where $\varphi : \mathbb{R} \to \mathbb{R}$ is a continuous, nondecreasing, nonlinear function. Fig. 3 displays some examples of graphs for function φ. In particular, the two first graphs model saturation phenomena and the third graph models a multi-valued function corresponding to the boundary condition: $\frac{\partial u}{\partial n} + \varphi(u) \ni 0$ occurring when u is subject to constraints inequality

The discretization techniques presented above can be also used for the interior points of domain Ω. For all points in Γ_d, the discretization of the Neumann condition leads to the solution of the following discrete equations

$$\frac{u_l - u_{l-1}}{h} + \varphi(u_l) = 0. \tag{22}$$

where l is an index of a point which belongs to Γ_d. Thus, if φ is singlevalued we have to solve the problem

$$\mathcal{A}(U) = AU + \Phi(U) - G = 0, \tag{23}$$

where A is the discretization matrix associated with the linear part of the equations, Φ is a diagonal, nondecreasing operator and $(G, U) \in \mathbb{R}^{dim(A)} \times \mathbb{R}^{dim(A)}$.

It follows from (22) that the lth component of φ is equal to $h\varphi(u_l)$ if l is the index of a point which belongs to Γ_d otherwise this component is null. If $q > 0$, then the matrix A is a strictly diagonally dominant matrix with strictly positive diagonal entries and off-diagonal entries non-positive. Then we are in the framework presented in Section 2. In the case where $q = 0$, we can verify by a similar argument that the matrix A is irreducibly diagonally dominant, regardless the sign of a, b and c, and once again we are in the framework presented in Section 2. According to the properties of A and since Φ is a continuous, nondecreasing, diagonal mapping, then parallel synchronous or asynchronous subdomain methods converges.

Remark 10. We consider now the resolution of the nonlinear convection–diffusion problem (21) by limiting ourselves to a nonlinearity corresponding to the two first graphs of Fig. 3 ; indeed the nonlinearity presented in the last graph of this last figure, corresponding to the resolution of a problem with inequality constraints of type (18) will be the subject of a future study. It should be noted that the applications presented in the first two graphs of Fig. 3 are not differentiable everywhere, especially at the junction of the parts where φ is constant with its increasing parts. Therefore the use of Newton's method is not possible in this case. On the other hand one can consider the use of a point relaxation method on each of the sub-domains particularly well adapted to the solution of Eq. (22). To solve this last equation one can implement a method of successive approximations of the type

$$u = \psi(v)$$

with $\psi(v)$ defined by $\psi(v) = h.\varphi(v) + (\tilde{u})_{i-1}$ where $\tilde{u}_{i-1}$ represents the last available value of u_{i-1}. Let β be the maximum absolute value of the slope of the increasing portions of the application $u \to \varphi(u)$; the Lipschitz constant of the application $u \to \psi(u)$ is then equal to $h.\beta$ and for a sufficiently appropriate small value of h, then $h.\beta < 1$ and ψ is contracting and the solution of (22) by the method of successive approximations converges.

Consider now the most general case, i.e. the nonlinear case. We have $\mathcal{A}(U) = AU + \Phi(U) - G$. Assume that A have strictly positive diagonal entries, off-diagonal entries such that $a_{i,j} \leq 0$ and is strictly or irreducibly diagonal dominant and moreover $\Phi(u)$ is a monotone increasing mapping. Then, we are in a context similar to (23). If we solve the nonlinear simultaneous equations $\mathcal{A}(U) = 0$, via the Schwarz alternating method, then the augmentation process of the Schwarz alternating method transforms again the M-matrix A into an M-matrix $\widetilde{A}$ and the monotone increasing mapping Φ into the monotone increasing mapping $\widetilde{\Phi}$ (see [13,29]). Thus, we are in the convergence analysis framework considered above for convergence in [13].

Remark 11. Moreover, to solve the global problem, we consider an overlapping subdomain method corresponding to the alternating Schwarz method which will also be convergent given the results stated in Example 2. From an implementation point of view in the two-dimensional case where the domain Ω is divided into slices, Fig. 2 represents the schematic diagram of this method. However, in Example 2 the domain Ω is three-dimensional and we did not choose a slicing as shown in Fig. 2. The domain Ω has been decomposed into small overlapping three-dimensional elements, for example into small parallelograms. So instead of having two exchanges with its right and left neighbors, this small element will perform data exchanges with six overlapping neighbors, i.e. with neighbors located at the top, bottom, right, left, front and back. So, in this last case of decomposition, there are three times as many communications but the behavior of the algorithm is more multiplicative.

Remark 12. Note that any direct or iterative method can be used on each subdomain. In the case where $\nu = 0$, the discretization matrix is triangular [38]. Thus, a relaxation method converges in only one iteration when the scanning order of the grid matches the triangular discretization matrix. In the case where ν is small, the discretization matrix is nearly triangular. The entries associated with a triangular part of the matrix derived from the decentered discretization scheme have higher order of magnitude than other entries. A relaxation method can then be a quasi-direct method on each subdomain and its computational cost can be very low.

4. Implementation and parallel experiments on cloud architecture

The implementation of the studied problems is briefly detailed since there is not difficult once the resources are allowed to the user; this

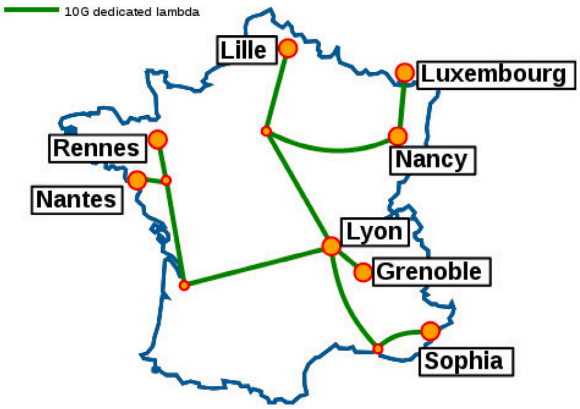


Fig. 4. Platform architecture of Grid'5000.

implementation is carried out with MPI (Message Passing Interface) facilities and the computational codes have been written using C or Fortran language. Matrix and right hand side creation have been implemented by a parallel way. For each simulation, we have considered a cubic domain Ω contained in the 3D space.

In the case of parallel subdomain algorithms without overlapping a block relaxation method is performed on each process and only one subdomain is assigned to each processor. A subdomain is constituted by gathering adjacent blocks. The principle of implementation of parallel asynchronous and synchronous iterative subdomain methods is describe by Algorithm 1.

Algorithm 1: Principle of implementation

```
while not global convergence do
    for each subdomain do
        Perform communications of subdomain boundary values
        Perform block relaxation resolution
    end
end
```

For comparison of synchronous and asynchronous performances, the number τ which is the ratio of elapsed time between synchronous and asynchronous methods given by

$$\tau = \frac{synchronous\ elapsed\ time\ algorithm}{asynchronous\ elapsed\ time\ algorithm}$$

is used; so τ is significant to achieve a comparison between the two communication modes. Indeed, in the case of heterogeneous architecture the notion of speed up and efficiency is not relevant since it is difficult to decide which machine is the reference and also all the more so since for large algebraic systems it is not possible to have the restitution times in sequential mode.

4.1. Parallel experiments on cloud like architecture

In the short paper [12], synchronous and asynchronous parallel algorithms simulation was carried out on a cloud computing like architecture based on a grid platform (called Grid'5000 see [41]). This platform (see Fig. 4), provides access to a large amount of resources (8 sites, 800 compute-nodes), gives an easy access to a wide variety of hardware technologies and is particularly suitable to carry out experiments.

This platform can be used as an infrastructure proposed by a Cloud provider which shares mutualized resources and, a virtual computer corresponds to a quantity of virtual resources. It also allows the execution of virtual computers with hardware virtualization capabilities.

Table 1
Synchronous and asynchronous simulations for solving problem (6) by subdomains method without overlap.

Size	Synchronous		Asynchronous		τ
n^3	Relaxations	Elapsed time (s)	Relaxations	Elapsed time (s)	
360^3	2 866 016	6 076	3 815 801	1 308	4.6
480^3	6 364 544	38 460	7 895 476	7 728	5.0

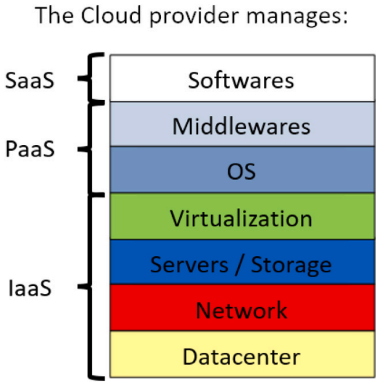


Fig. 5. Cloud services.

In this paper, we simulated problem (6) with several sizes 360^3 and 480^3 on 64 computers resources. In the following Table 1, synchronous and asynchronous results are presented in term of elapsed time and number of relaxations. It can be noticed that for the considered application and the used architecture, the asynchronous computation scheme gives better results than the synchronous scheme; on 64 machines; such algorithm is clearly faster by 4.6 and 5.0 respectively.

4.2. Parallel experiments on cloud architecture

Grid architecture is designed as a distributed system and file manager with the advantage of shared resources but on one hand the disadvantage of a tendency toward complexity for Application Programming Interface (API) which is a way for computers programs to communicate with each other, and on the other hand the deployment of services on this infrastructure.

So the Cloud has emerged from the convergence of several ideas following the maturation of technology for visualization with a simplified API look and feel and also large computing capacity to meet resource requirements.

The Cloud computing (see [42]) is a way of providing access to computer resources, characterized by its self-service availability, elasticity, openness, mutualization and pay-per-use; self-service and on-demand resources of storage capacity and computing power are based on the customer's needs. In the Cloud, the need, automatically detected by the application or at the customer's request, is taken into account and satisfied immediately and we no longer speak of infrastructure components but of services (see Fig. 5).

The different services are :

- IaaS (Infrastructure as a Service) which provide virtualized environments remotely by showing them as physical machines (CPU, disk, memory, network ...). Its advantages are that we can have a customized environment, completely controllable, accessible at any time and usable with a simple API.
- PaaS (Platform as a Service) which provides ready-to-use software and middleware (database, web server, etc.) as, for example, a platform for developing web applications but also an infrastructure for deploying and running applications. Its advantage is that

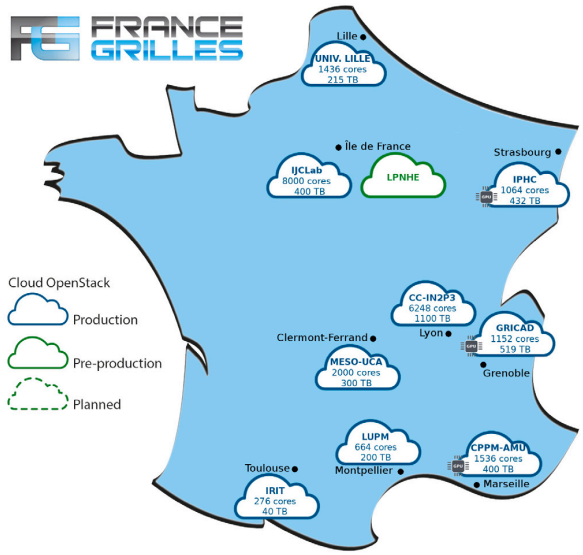


Fig. 6. Cloud resources of FG Cloud.

it provides load balancing and service redundancy features for example. Programmers can develop at a higher level.

- SaaS (Software as a Service) which provides ready-to-use software. It is an application accessible to the web as a hosting service. Its advantages are its simplicity of use with no software deployment, a web interface and its accessibility. The functioning of this service allows to the users to connect and share their applications over internet.

For this new experimental study, computational experiments have been carried out on France Grilles-Cloud (see Fig. 6) which offers to users cloud services that allows computing, storage and network on demand, clusters instanciation, web services instanciation dedicated to the scientific community and tests of algorithms.

Particularly, we use the Computer Center of the National institute of nuclear and particle physics (IN2P3), a french CNRS support and research unit. This institute that pursues and coordinates research on particle physics, nuclear physics and astroparticle physics. Researchers can use servers on which computer calculations are shared through a scheduling system. Calculations may be sequential or parallel. Computing servers, which run a Linux system, mostly feature Intel x86_64 processors and, to a lesser extent, special graphic processors. The resources may be used locally or remotely via a computing infrastructure. The Computing Center have external links of up to 100 Gbit/s and a unique internal network with a 400 Gbit/s backbone.

Our parallel simulations can therefore be run with reserved resources in the platform presented above with different numbers of cores, memory size and CPU. These resources are operating system templates and each instance is it a machine which runs our workloads in the cloud, start as an identical clone of all other instances. Once the client is functional, using a cloud instance is done in 4 steps :

- provide a Secure Shell Protocol (SSH) - which is a network communication protocol that enables two computers to communicate — to access the instances;
- instantiate a server;
- assign a public address IP (Internet Protocol) - which is a series of numbers that identifies the connection to the internet — to the instance;
- connect to the instance.

Table 2
Openstack instance list.

Instance name	Networks	Image (OS)	Flavor
mavm1	Public IP	Ubuntu-20.04	1CPU_8GB-RAM
mavm2	Public IP	Ubuntu-20.04	1CPU_12GB-RAM
mavm3	Public IP	Ubuntu-20.04	2CPU_4GB-RAM
mavm4	Public IP	Ubuntu-20.04	4CPU_8GB-RAM
mavm5	Public IP	Ubuntu-20.04	2CPU_4GB-RAM
mavm6	Public IP	Ubuntu-20.04	4CPU_8GB-RAM
mavm7	Public IP	Ubuntu-20.04	4CPU_8GB-RAM
mavm8	Public IP	Ubuntu-20.04	4CPU_8GB-RAM

In this Computer Center, OpenStack which is a collection of open source software modules and tools that provides a framework to create and manage both public cloud and private cloud infrastructure, is used.

Moreover, OpenStackClient which is a command-line client for OpenStack that brings the command set for the various APIs together in a single shell with a uniform command structure, is used for managing instances.

After creating an SSH key pair, it is possible to establish a SSH session to instances and it is then necessary to provide with Openstack the public key to be injected in the instances for the root account. Then, after consulting the list of available images, we work with the available flavors list. In fact, the flavors represent the available types of virtual machines and determine the virtual machines characteristics. Note that we have virtual characteristics composed of number of CPU (Central Processing Unit) and size of the RAM (Random Access Memory). After creating instances, it is possible to run the instances and realize the experiments (see Table 2). In our project, the network associates a floating IP to our instances in order to have it connected on the network.

For the problems (6) and (13) solved by subdomain method without and with overlap, we have considered the cubic domain Ω with different sizes such as 120^3, 240^3, 360^3 and 480^3.

In the following subsections, we consider two distinct kinds of boundary value problems : on one hand a linear diffusion problem and on the other hand a linear convection–diffusion problem without and with overlapping.

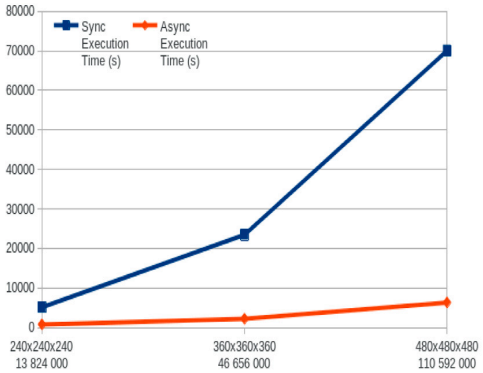
As previously described, we use the Iaas mode. The implementation in the cloud environment is done in a classic way and without real difficulty. Once the resources have been requested, we obtain them without knowing where they come from, their location and their hardware configuration. Moreover, we do not have any information about the possible sharing of physical machines between several users and the latency in the network connections between the 8 resources is not well known at a given time. Moreover, we did not have access to all 22 CPUs mentioned in Table 2, but to a limited number of these CPUs through the use of 8 virtual CPUs.

We started with an installation with a uniform operating system image on the Cloud resources. As mentioned before, our codes were developed in C or Fortran language with the help of the MPI library and scripts were set up to deploy the codes on the cloud resources as well as the libraries necessary for the functioning of the MPI environment. Other scripts were also used to run the simulations and retrieve the results.

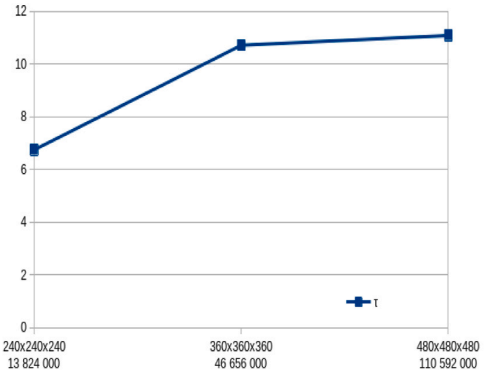
The number of machines used has been limited up to 8 according to the instances reservation; moreover, since the overhead damages the performances of the parallel algorithms, additional machines are not necessary particularly when synchronous scheme are tested.

Numerical solution of the linear diffusion equation implementation.

We consider first the numerical solution of problem (6), for which the results of experiments are summarized in Table 3 for $q = 1000$. We took an high value of q since the system is ill-conditioned knowing that this choice penalizes asynchronous algorithms because the convergence is fast and there is few synchronization. In these table and the



(a) Elapsed time with respect to domain sizes.



(b) Variation of the value τ with respect to the domain sizes.

Fig. 7. Results for problem (6) on 8 cloud machines for $q = 1000$ by subdomains method without overlapping.

Table 3
Synchronous, asynchronous and τ simulation results for solving problem (6) by subdomains method without overlap.

Size	Synchronous		Asynchronous		τ
n^3	Relaxations	Elapsed time (s)	Relaxations	Elapsed time (s)	
240^3	6 128	5 141	42 682	762	6.75
360^3	11 904	23 473	43 706	2 191	10.71
480^3	18 920	70 059	44 956	6 319	11.09

Table 4
Synchronous, asynchronous and τ simulation results for solving problem (13) by subdomains method without overlap.

Size	Synchronous		Asynchronous		τ
n^3	Relaxations	Elapsed time (s)	Relaxations	Elapsed time (s)	
240^3	23 840	20 772	59 243	1 943	10.70
360^3	133 728	271 516	243 584	11 880	22.86
480^3	183 520	965 295	296 317	42 480	22.72

Table 5
Synchronous, asynchronous and τ simulation results for solving problem (13) by subdomains method with overlap of two meshes.

Size	Synchronous		Asynchronous		τ
n^3	Relaxations	Elapsed time (s)	Relaxations	Elapsed time (s)	
120^3	11 475	4 578	131 472	441	10.38
240^3	43 889	110 836	245 576	5 720	19.38
360^3	89 601	496 270	290 256	21 887	22.67
480^3	171 671	1 666 259	348 488	61 747	26.99

following Figs. 7(a) and 7(b), comparison of elapsed time for parallel synchronous and asynchronous methods as well as the parameter τ are presented.

This tables and Fig. 7(a) show clearly the benefits of using parallel asynchronous algorithms in the parallel experiments. Finally, in Fig. 7(b), note that the parameter τ is increasing with the dimension of the discretization matrix.

Numerical solution of the linear convection–diffusion equation implementation without and with overlapping.

We consider first the numerical solution of problem (13), for which the results of experiments without overlapping are summarized in Table 4 and with an overlap of two meshes in Table 5 with both $a = 0.5, b = 1.5$, $c = 1.0$ and $q = 10$. In these tables and the following Figs. 8(a), 8(b), 9(a) and 9(b) comparison of elapsed time for parallel synchronous and asynchronous methods as well as the parameter τ are presented.

In Figs. 8(b) and 9(b), note that the parameter τ can be increasing with the dimension of the discretization matrix. Finally, note again that the use of asynchronous parallel schemes is very interesting in cloud computing with distant and heterogeneous instances for the second problem considered too.

- *Experiments without overlapping* (see Table 4 and Figs. 8(a), 8(b))
- *Experiments with overlapping* (see Table 5 and Figs. 9(a), 9(b))

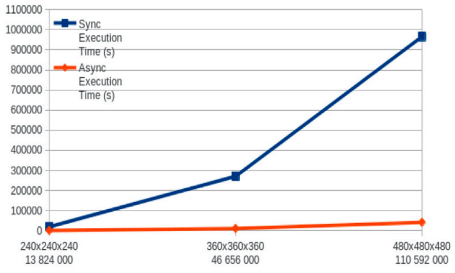
Synthesis of the analysis.

The results obtained confirm our expectations, i.e. that the asynchronous parallel algorithms are more efficient when using cloud resources without knowledge of their positioning, shared usage, network latency and hardware architecture.

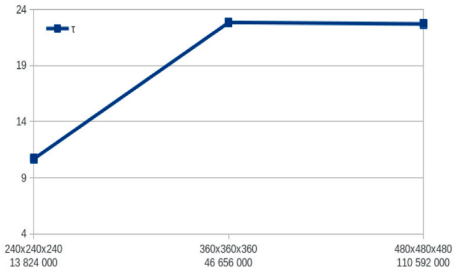
We can noted that, in asynchronous parallel experiments, the number of relaxations necessary to reach convergence is greater than the one necessary when parallel synchronous schemes are used. In fact, this is a well-known drawback of asynchronous iterative schemes, due to the use of possibly delayed components. It turns out that the overhead generated by additional relaxations in the case of asynchronous algorithms is smaller than the synchronization overhead combined with processor idle time of synchronous parallel schemes of computation. In the same way the speed of convergence of parallel asynchronous method is generally slower compared to the synchronous one. Nevertheless the speed of convergence depend of the frequency of updates of calculations and then of the decomposition of the initial problem in subproblems ; so some decomposition may result in a better convergence rate. But generally there is a reduction of the elapsed time to reach convergence. However the increase of the number of relaxations improve the solution's accuracy and the numerical quality of the computations of the numerical solution.

Furthermore, despite higher number of relaxations, elapsed time of asynchronous scheme is less than the one obtained when synchronous scheme is used.

As a consequence the parallel experiments show that on cloud architecture asynchronous parallel methods present real interest when parallel processing requires a large number of synchronizations. This occurs in particular when the convergence is slow, and especially when the communications between the machines are slow particularly when the connections between these machines is not by direct lines. Indeed, communications on a platform are dependent on the latency of the network. This slowdown is accentuated by the distance between machines

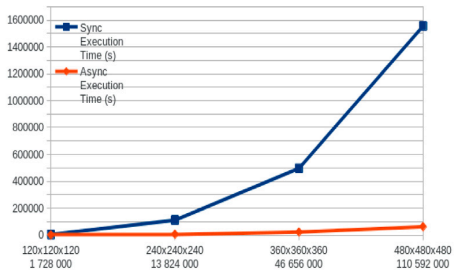


(a) Elapsed time with respect to domain sizes.

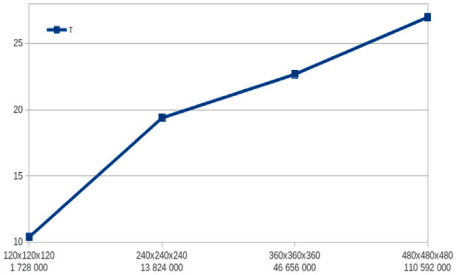


(b) Variation of the value τ with respect to the domain sizes.

Fig. 8. Results for problem (13) on 8 cloud machines for $q = 10$ by subdomains method without overlapping.



(a) Elapsed time with respect to domain sizes.



(b) Variation of the value τ with respect to the domain sizes.

Fig. 9. Results for problem (13) on 8 cloud machines for $q = 10$ with an overlap of 2 meshes.

that are not connected by direct lines. This phenomenon of superiority of the asynchronous parallel methods is thus accentuated when the processors are heterogeneous and distant, which is the case in our simulations on cloud computing. Thus, correlatively, the performance in terms of restitution time of asynchronous parallel methods is much higher than that of synchronous methods. We can thus see here the influence of the architecture of the machines on the performances of the algorithms.

In addition considering the additional software layers, the use of asynchronous parallel algorithms is of interest in a cloud environment compared to the synchronous equivalent insofar as inactivity times are eliminated. Thus asynchronism is an efficient way to deal with communication overhead and load unbalance.

Moreover we obtain also good performances by using parallel asynchronous methods when low granularity is considered (see Table 5 when $n = 120$). Finally using asynchronous methods has a positive impact on environment and allows electrical energy saving.

5. Conclusion

Similar performances could be obtained with other boundary conditions (Neumann conditions, Robin conditions, mixed boundary conditions, etc.). According to our experience and the performances obtained for the model problem (6) and (13) the presented study allows us to hope for the solution of nonlinear problems constituted by :

- other large pseudo-linear single-valued systems with for example application for the simulation of solar oven or situations involved in plasma study,
- large pseudo-linear multi-valued systems with for example application in financial application or in image processing corresponding in this last case to the solution of Hamilton–Jacobi-Bellman equation.

Other possible applications particularly the solution of Navier–Stokes equation corresponding in numerous formulations to the coupling of a diffusion equation to a convection–diffusion equation.

The results obtained during the simulations on the Cloud open up other work perspectives. For example, it would be interesting to use tools that allow us to modify the network speeds and thus decrease or increase the network latency in order to verify its influence on the results of our asynchronous algorithms.

CRediT authorship contribution statement

M.A. Rahhali: Conceptualization, Methodology, Development, Simulation, Writing. **T. Garcia:** Conceptualization, Methodology, Development, Simulation, Writing. **P. Spiteri:** Conceptualization, Methodology, Development, Simulation, Writing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This study has been made possible with the support of France Grilles- cloud thanks to whom we have access to cloud resources LPNHE which is a UMR Sorbonne University/CNRS-IN2P3. In addition, the previous study was made possible by the use of Grid'5000.

References

[1] Spiteri P. Parallel asynchronous algorithms: a survey. Adv Eng Softw 2020;149. http://dx.doi.org/10.1016/j.advengsoft.2020.102896, Elsevier.
[2] Garcia T, Spiteri P, Ziane-Khodja L, Couturier R. Solution of univalued and multivalued pseudo-linear problems using parallel asynchronous multisplitting methods combined with krylov methods. Adv Eng Softw 2020;153. http://dx.doi.org/10.1016/j.advengsoft.2020.102929, Elsevier.
[3] Garcia T, Spiteri P, Ziane-Khodja L, Couturier R. Coupling parallel asynchronous multisplitting methods with Krylov methods to solve pseudo- linear evolution 3D problems. J Comput Sci 2021;51. http://dx.doi.org/10.1016/j.jocs.2021.101303, Elsevier.
[4] Saad Y. Iterative methods for linear systems of equations: a brief historical journey. In: Brenner Susanne C, Shparlinski Igor E, Shu Chi-Wang, Szyld Daniel, editors. 75 years of mathematics of computation. Contemporary mathematics, vol. 754, 2020, p. 197–214.
[5] Glusa C, Boman EG, Chow E, Rajamanickan S, Szyld D. Scalable asynchronous domain decomposition solvers. SIAM J Sci Comput 2020;42(6):C384–409. http://dx.doi.org/10.1137/19M1291303.
[6] Gbikpi-Benissan G, Magoules F. Asynchronous multiplicative course-space correction. SIAM J Sci Comput 2022;44(3):C237–259. http://dx.doi.org/10.1137/21M1432107.
[7] Nayak P, Cojean T, Anzt H. Evaluating asynchronous Schwarz solvers on GPUs. Int J High Perform Comput 2021;35(3):226–36. http://dx.doi.org/10.1177/1094342020946814.
[8] Yamazaki I, Chow E, Bouteiller A, Dongara J. Performance of asynchronous optimized Schwarz with one-sided communication. Parallel Comput 2019;86:66–81.
[9] Gbikpi-Benissan G, Magoules F. Asynchronous multisplitting-based primal Schur method. J Comput Appl Math 2023;425:115060.
[10] Chaouqui F, Chow E, Szyld D. Asynchronous domain decomposition methods for nonlinear PDES. Electron Trans Numer Anal 2023;58:22–42.
[11] Cai X-C, Sarkis M. A restricted additive Schwarz preconditioner for general sparse linear systems. SIAM J Sci Comput 1999;21:792–7.
[12] Rahhali MA, Garcia T, Spiteri P. Behaviour of asynchronous parallel iterative algorithms on cloud computing type architectures. In: Topping BHV, Iványi P, editors. Proceedings of the eleventh international conference on engineering computational technology. Edinburgh, UK: Civil-Comp Press; 2022, http://dx.doi.org/10.4203/ccc.2.8.2, Online volume: CCC 2, Paper 8.2.
[13] Miellou JC, El Baz D, Spiteri P. A new class of asynchronous iterative algorithms with order interval. Math Comp 1998;67–221:237–55.
[14] Giraud L, Spiteri P. Résolution parallèle de problèmes aux limites non linéaires. M2AN 1991;25:579–606.
[15] Chazan D, Miranker W. Chaotic relaxation. Linear Algebra Appl 1969;2:199–222.
[16] Miellou J-C. Algorithmes de relaxation chaotique à retards. RAIRO Anal Numér 1975;1:55–82.
[17] Miellou J-C. Itérations chaotiques à retards, étude de la convergence dans le cas d'espaces partiellement ordonnés. CRAS Paris 1975;280:233–6.
[18] Bertsekas DP, Tsitsiklis JN. Parallel and distributed iterative algorithms: a selective survey. Automatica 1991;25:3–21.
[19] Bertsekas D, Tsitsiklis J. Parallel and distributed computation. Numerical methods, Prentice Hall Englewood Cliffs N.J.; 1989.
[20] Baudet G. Asynchronous iterative methods for multiprocessors. J Assoc Comput Mach 1978;25:226–44.
[21] Frommer A, Szyld D. On asynchronous iterations. J Comput Appl Math 2000;123:201–16.
[22] Bahi JM, Contassot-Vivier S, Couturier R. Parallel iterative algorithms: From sequential to grid computing. Chapman & Hall/CRC; 2007.
[23] El Baz D, Frommer A, Spitéri P. Asynchronous iterations with flexible communications : contracting operators. J Comput Appl Math 2005;176:91–103, Elsevier.
[24] El Tarazi M. Some convergence results for asynchronous algorithms. Numer Math 1984;39:325–40.
[25] Miellou J-C, Spiteri P. Un critère de convergence pour des méthodes générales de point fixe. M2AN 1985;19:645–69.
[26] Partimbene V, Garcia T, Spiteri P, Marthon P, Ratsifrandrihana L. Asynchronous multi-splitting method for linear and pseudo-linear problems. Adv Eng Softw 2019;133:76–95.
[27] Varga RS. Matrix iterative analysis. Prentice - Hall; 1962.
[28] Ortega J, Rheinboldt W. Iterative solution of nonlinear equations in several variables. New York: Academic Press; 1970.
[29] Evans DJ, Deren W. An asynchronous parallel algorithm for solving a class of nonlinear simultaneous equations. Parallel Comput 1991;17:165–80.
[30] Hoffmann KH, Zou J. Parallel efficiency of domain decomposition methods. Parallel Comput 1993;19:1375–91.
[31] Magoules F, Gbikpi-Benissan G. Distributed convergence detection based on global residual error under asynchronous iterations. IEEE Trans Parallel Distrib Syst 2018;29(4):819–29.
[32] Magoules F, Gbikpi-Benissan G. JACK2: An MPI-based communication library with non-blocking synchronization for asynchronous iterations. Adv Eng Softw 2018;11:116–33.
[33] Gbikpi-Benissan G, Magoules F. Protocol-free asynchronous iterations termination. Adv Eng Softw 2020;146:102827.
[34] Chow E, Frommer A, Szyld DB. Asynchronous Richardson iterations: theory and practice. Numer Algorithms 2021;87:1635–51.
[35] Dijkstra EW, Scholten CS. Termination detection for diffusing computation. Process Lett 1980;11:1–4.
[36] Savari SA, Bertsekas DP. Finite termination of asynchronous iterative algorithms. Parallel Comput 1996;22(1):39–56.
[37] Chau M, Laouar A, Garcia T, Spitéri P. Grid solution of problem with unilateral constraints. Numer Algorithms 2017;75-4:879–908, Springer-Verlag.
[38] Chau M, El Baz D, Guivarch R, Spitéri P. MPI implementation of parallel subdomain methods for linear and nonlinear convection–diffusion problems. J Parallel Distrib Comput 2007;67(5):581–91.
[39] Miellou JC, Spitéri P, El Baz D. A new stopping criterion for linear perturbed asynchronous iterations. J Comput Appl Math 2008;219(2):471–83, Elsevier.
[40] Spitéri P. Finite precision computation for linear fixed point methods of parallel asynchronous iterations. In: Topping BHV, Ivanyi P, editors. Techniques for parallel, distributed and cloud computing in engineering. Saxe-Coburg Publications; 2015, p. 163–96.
[41] Bolze R, Cappello F, Caron E, Daydé M, Desprez F, Jeannot E, et al. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. Int J High Perform Comput Appl 2006;20(4):481–94.
[42] Magoules F, Pan J, Teng F. Cloud computing data-intenbsive computing and scheduling. Chapman & Hall/CRC Numer Anal Sci Comput 2013.

5.2 Article 2

ARTICLE IN PRESS

Appl. Numer. Math. xxx (xxxx) xxx

Contents lists available at ScienceDirect

Applied Numerical Mathematics

journal homepage: www.elsevier.com/locate/apnum

Research Paper

Parallel cloud solution of large algebraic multivalued systems

M.A. Rahhali [a], T. Garcia [a,b,*], P. Spiteri [a]

[a] University of Toulouse, IRIT-INPT-ENSEEIHT, 2 rue Charles Camichel, B.P. 7122, 31071 Toulouse - Cedex 7, France
[b] CY Paris University, CY Tech Campus de Pau, 2 boulevard Lucien Favre, CS 77563, 64075 Pau Cedex, France

ARTICLE INFO

Keywords:
High performance computing
Parallel computing
Cloud computing
Asynchronous iterations
Asynchronous algorithms
Iterative methods
Multivalued problems

ABSTRACT

The present paper deals with the resolution on cloud architecture of synchronous and asynchronous iterative parallel algorithms of stationary or evolution variational inequations formulated by a multivalued model. The performances of synchronous and asynchronous iterative parallel methods are compared with previous ones obtained on cluster or when grid architecture is used. Thanks to the properties of the algebraic systems resulting from problem discretization we are able to analyze the behavior of the iterative algorithm in particular the convergence and the speed of convergence. The implementation of the studied methods on cloud architecture is described. Then we present various applications in particular the solidification of steel in continuous casting, the cavity pressure calculation described by a problem subject to unilateral constraints and finally a financial problem modeled by American option pricing.

1. Introduction

In [1], we used cloud computing architecture resources for the numerical solution of linear diffusion and convection - diffusion equation by the way of parallel iterative methods; in this recent work the communication between the machines were performed both synchronously and asynchronously. Due to the additional software layers necessary for managing the system in cloud architecture, the performances of parallel asynchronous iterative algorithms are significantly higher than the ones obtained with synchronous communications between the machines. Consequently for the resolution of large scale algebraic systems on cloud architecture, these numerical experiments have encouraged us to continue our work on implementation on cloud architecture for solving linear and nonlinear boundary problems; besides on the other hand, on this type of architecture, comparison of performances of parallel iterative asynchronous methods with synchronous ones is also pursued.

In the present study, we consider the solution of highly nonlinear boundary problems corresponding to situations where the solution is subject to inequality-type constraints, which leads to the solution of stationary or evolutionary variational inequations. There are several formulations of such variational inequalities. We consider here a multivalued formulation of these problems constituted by a linear or pseudo-linear problem perturbed by a multivalued operator constituted by the subdifferential of the convex indicator function modeling the constraints on the solution (see [2] to [4]). Taking into account the properties of the matrices resulting from the discretization of the partial-derivatives obtained by finite-difference, finite-element or finite-volume, and also of the monotony of the subdifferential of the indicator function, we are able to analyze the behavior of synchronous and asynchronous iterative parallel algorithms, in particular to obtain convergence results and an estimate of the speed of convergence; the termination of these methods is also implemented by well-adapted algorithms [5].

* Corresponding author at: University of Toulouse, IRIT-INPT-ENSEEIHT, 2 rue Charles Camichel, B.P. 7122, 31071 Toulouse - Cedex 7, France.
E-mail addresses: mohammedamine.rahhali@toulouse-inp.fr (M.A. Rahhali), thierry.garcia@irit.fr (T. Garcia), pierre.spiteri@enseeiht.fr (P. Spiteri).

https://doi.org/10.1016/j.apnum.2024.03.012
Received 29 November 2023; Received in revised form 29 February 2024; Accepted 12 March 2024

Available online 16 March 2024

The main goal of this study is then to compare the performances of asynchronous parallel algorithms implemented on cloud architecture with that of synchronous parallel algorithms, and to see the influence of the additional software layers present in the systems implemented in this type of architecture; in other words, the main aim of the study is to find out which type of communication is well suited to solving very large algebraic systems.

The main originality of the study lies in its parallel implementation on this type of machine, encompassing both types of communications considered. Furthermore, we also compare the performances of both parallel methods obtained on cloud architecture with the previous performances of the same algorithms implemented on clusters and grid architecture. In contrast to the information available when using clusters or grid computing, where the characteristics of the machines are well known, comparing performances on cloud architecture is not easy, since we do not know on which machines in the cloud will be running the computing jobs and consequently the characteristics of cloud machines are unknown.

To compensate for this lack of information, we define and use an indicator denoted τ, which is defined as the ratio of calculation restitution times in synchronous and asynchronous communication modes. It is therefore an additional means of measuring the performances of parallel algorithms using various types of communications, and enables us to analyze the influence of additional software layers on the two modes of communication under consideration. The results presented in the experimental section show that, due to operating constraints, asynchronous parallel algorithms perform well even at relatively modest algebraic system sizes.

In particular, in situations where the speed of convergence is very high, this comparison of synchronous and asynchronous communication modes shows excellent results on cloud architecture, while the opposite effect was obtained when both parallel methods are implemented on clusters using high-performance networks. This state of affairs is essentially due to the influence of the communications network, the heterogeneity of the machines and their remoteness. Similarly, in view of the results to be presented, an examination of the value of the τ parameter shows that performance on cloud architecture is better than on grid computing.

To avoid numerous cross-references to bibliographical references, the writing of this manuscript is complete and relatively self-contained. This paper is organized as follows. In paragraph 2 we present synchronous and asynchronous parallel algorithms in an unified framework, and give results that allow these methods to be analyzed using contraction techniques. In the next section, we present elements for implementing synchronous and asynchronous parallel algorithms on cloud architecture. In connection with paragraph 2 the rest of the paper presents several applications formulated in a multivalued way. In paragraph 4, we present simulation results on cloud architecture for an industrial problem involving the solidification of steel in continuous casting; this problem is very interesting in view of the strong diagonal dominance properties of the discretization matrices, and shows that the studied algorithms perform differently on cluster, grid and cloud architecture, illustrating the fact that the behavior of parallel iterative algorithm is architecture-dependent. Paragraph 5 is devoted to cloud simulations for the determination of cavity pressure modeled by problem subject to unilateral constraints. Finally, paragraph 6 presents cloud simulations to solve a financial problem modeled by American option pricing. Note that for the three considered applications we will mainly compare the performances obtained by using FG Cloud IN2P3 resources with the ones previously obtained by using cluster and Grid'5000 facilities. Paragraph 7 provides a synthesis of the parallel experiments on target architectures. Finally, the paper discusses prospects for addressing other highly nonlinear boundary problems in the future.

2. Numerical background

2.1. Discretized multivalued model problems

In the sequel, the set of natural integers will be denoted by $\mathbb{N}$. Let $M \in \mathbb{N}$ and let us also denote by $\mathcal{E} = \mathbb{R}^M$ a normed vector space. Let us consider large linear problems $AU = G$, where $A \in \mathcal{L}(\mathbb{R}^M)$ is a sparse matrix issued from the discretization of various boundary value problems, $U \in \mathbb{R}^M$ and $G \in \mathbb{R}^M$. Consider the case where the solution U is submitted to some inequality constraints such that

$$U_{min} \leq U \text{ or } U \leq U_{Max} \text{ or } U_{min} \leq U \leq U_{Max}. \tag{1}$$

In this case, classically, the problem can be expressed by a multivalued formulation (see for example [2], [3]) as follows

$$AU + \partial\Psi(U) - G \ni 0, \tag{2}$$

where $\partial\Psi(U)$ results from the discretization of $\partial\psi(u)$, ψ being the indicator mapping of the set K defining the convex set of constraints and $\partial\psi(u)$ denotes the subdifferential mapping of ψ (see [2]) corresponding in fact to the weak derivative of ψ; classically $\partial\Psi(U)$ satisfies the following property

$$\partial\Psi(U) \text{ is a diagonal increasing or more generally monotone maximal operator.} \tag{3}$$

Thus U is the solution of the pseudo - linear multivalued problem (2). In order to solve efficiently system (2) we consider in the sequel parallel implementation of fixed point algorithms. Such parallel fixed point methods need the decomposition of the whole problem in $\alpha \in \mathbb{N}, \alpha > 0$, interconnected subsystems associated to a large block decomposition of the algebraic system as follows

$$\sum_{k=1}^{\alpha} A_{lk}.U_k + \partial\Psi_l(U_l) - G_l \ni 0, \forall l \in \{1, \ldots, \alpha\}; \tag{4}$$

Then for $\alpha \in \mathbb{N}$, let us consider also that the space $\mathcal{E} = \prod_{l=1}^{\alpha} \mathcal{E}_l$ is a finite product of α subspaces denoted $\mathcal{E}_l = \mathbb{R}^{m_l}$, where $\sum_{l=1}^{\alpha} m_l = M$; note that $\mathcal{E}_l$ is also a normed vector space and let us denote by $|.|_l$ the associated norm, for all $l \in \{1, \dots, \alpha\}$; consequently in (4) $U_l \in \mathcal{E}_l$, $A = (A_{lk})$, and $\partial\Psi_l$ result from the associated block decomposition. The subdifferential mapping being multivalued, the fixed point mapping associated to the decomposition (4) is implicitly defined by

$$A_{ll}.U_l + w_l(U_l) = G_l - \sum_{k \neq l}^{\alpha} A_{lk} V_k, w_l(U_l) \in \partial\Psi_l(U_l), \forall l \in \{1, \dots, \alpha\}. \tag{5}$$

With similar assumptions a more general problem can be also considered by adding a diagonal operator to problem (2); so we consider the following multivalued problem

$$AU + \Phi(U) + \partial\Psi(U) - G \ni 0, \tag{6}$$

where Φ results from the discretization of a diagonal operator such that the $l-$th component of Φ is equal to $\Phi_l(U_l)$ and satisfies in addition

$$\Phi \text{ is a diagonal increasing or more generally monotone maximal operator.} \tag{7}$$

So accordingly to (4) and (5) the associated fixed point mapping is defined by

$$A_{ll}.U_l + w_l(U_l) + \Phi_l(U_l) = G_l - \sum_{k \neq l}^{\alpha} A_{lk} V_k, w_l(U_l) \in \partial\Psi_l(U_l), \forall l \in \{1, \dots, \alpha\}. \tag{8}$$

2.2. Parallel fixed point synchronous and asynchronous algorithms

Let us consider now the following fixed point problem

$$\begin{cases} \text{Find } U^* \in \mathcal{E} \text{ such that} \\ U^* = F(U^*) \end{cases} \tag{9}$$

where $V \mapsto F(V)$ applies from $D(F) \subset \mathcal{E}$ to $D(F)$. Let us assume that $U^\star$ is the exact solution of (9). According to the decomposition of $\mathcal{E}$, let us consider the corresponding block decomposition of F and V

$$F(V) = (F_1(V), \dots, F_\alpha(V)) \text{ and } V = (V_1, \dots, V_\alpha)$$

The general parallel asynchronous fixed point iterations $\{U^p\}_{p \in \mathbb{N}}$, are formally defined as follows (see [6] to [13]):

Definition 1. $U^0 \in \mathcal{E}$ being given, for all $p \in \mathbb{N}$ and $l \in \{1, \dots, \alpha\}$:

$$U_l^{p+1} = \begin{cases} F_l(\tilde{U}^p) \text{ if } l \in s(p), \\ U_l^p \text{ if } l \notin s(p), \end{cases} \tag{10}$$

with

$$\tilde{U}^p = \{U_1^{\rho_1(p)}, \dots, U_l^{\rho_l(p)}, \dots, U_\alpha^{\rho_\alpha(p)}\} \in \mathcal{E}, \text{ if } p \geq 1 \tag{11}$$

where $s(p)$ and $\rho(p)$ satisfy the following assumptions
- $\{s(p)\}, p \in \mathbb{N}$, is a sequence of non void sets such that for all p, $s(p) \subset \{1, \dots, \alpha\}$, satisfying in addition

$$\{p \in \mathbb{N} \mid l \in s(p)\} \text{ is infinite.}$$

- Moreover a sequence of delayed relaxation numbers $\{\rho(p)\}$

$$\rho(p) = (\rho_1(p), \dots, \rho_l(p), \dots, \rho_\alpha(p)) \in \mathbb{N}^\alpha$$

is such that for all $p \in \mathbb{N}$ and $k \in \{1, \dots, \alpha\}$ we have $0 \leq \rho_k(p) \leq p$ if $k \neq l$ and $\rho_l(p) = p$ if $l \in s(p)$. Moreover the delayed iteration numbers satisfy

$$\lim_{p \to \infty} \rho_l(p) = +\infty, \forall l \in \{1, \dots, \alpha\}.$$

Remark 1. Asynchronous parallel iterations defined recursively by (10)-(11) are carried out by up to α processors without any order nor synchronization. The non void strategy $s(p)$ models the parallelism between the processes since at each step p several block components of U are selected and relaxed. The sequence of delayed relaxation numbers models the asynchronism between processors and allows the elimination of idle times due to blocking synchronizations. Consequently each process performs its own calculations using available data calculated by other processors. Note that theoretically, each block component is continuously updated; but in

practice the parallel iterative method is ended by a stopping criterion. The previous computational model is general since parallel synchronous methods are modeled when $\rho_k(p) = p$ for all $p \in \mathbb{N}$ and for all $k \in \{1, \dots, \alpha\}$; moreover, in this case, if for all $p \in \mathbb{N}$, $s(p) = \{1, \dots, \alpha\}$ (respectively $s(p) = (1 + p \mod \alpha)$) then algorithm (10)-(11) model the sequential block-Jacobi (respectively block-Gauss-Seidel) method. For more details, the reader is referred to [6] to [13].

2.3. Convergence of asynchronous iterations

For the convergence analysis of parallel asynchronous algorithms applied to the resolution of multivalued problems such that (2) and (6) contraction techniques are usually used; indeed such analysis method provides useful information on the behavior of parallel iterative methods. Sophistical mathematical tools are needed to carry out such studies and in the sequel we simply point out a few useful results and refer the reader to [5] for more information. In fact the analysis of convergence for this particular problem type is based on monotony notions in Banach spaces (see [2]) requiring the use of the duality notion well defined in these normed vector spaces. On the one hand this notion generally used in applied analysis provides interesting results in linear algebra and on the other hand allows to analyze the behavior of asynchronous parallel methods for solving nonlinear problems, especially those where the solution is subject to constraints of type (1) for example, which in each case requires defining the convex set of constraints denoted in the sequel K. So for more details we refer to [5] and in what follows we limit ourselves to indicate some useful convergence criteria.

Then, in brief, assumptions (3) and / or (7) being always satisfied, consider now the block decomposition of problems to solve. For $l = 1, \dots, \alpha$, let us denote by $< ., . >_l$ the pairing[1] between the vector space $\mathcal{E}_l$ and $\mathcal{E}'_l$ its topological dual space; $| \ . \ |_l$ being always the associated norm in $\mathcal{E}_l$, then $G_l(U_l)$ the duality mapping, is classically defined by

$$G_l(U_l) = \{g_l(U_l) \in \mathcal{E}'_l \mid \ < U_l, g_l >_l = |U_l|_l^2 \text{ and } |g_l|_l^* = |U_l|_l\},$$

where $|g_l|_l^*$ denotes the norm of $g_l \in \mathcal{E}'_l$. Note that $G_l(U_l)$ is the subdifferential mapping of $\frac{1}{2}| \ . \ |_l^2$; moreover when $\mathcal{E}_l$ is an Hilbert space then the pairing is identical to the scalar product of $\mathcal{E}_l$ and in this case $G_l(U_l)$ is the derivative of $\frac{1}{2}| \ . \ |_l^2$ (for more details we refer to [2]).

Consider now the following assumption: there exist α numbers $\eta_{l,l} > 0$ such that

$$< A_{ll}U_l, g_l >_l \geq \eta_{l,l}|U_l|_l^2, \forall U_l \in \mathcal{E}_l, \ l = 1, \dots, \alpha, \tag{12}$$

and in addition for all $l, k \in \{1, \dots, \alpha\}, l \neq k, \forall U_l \in \mathcal{E}_l, \forall U_k \in \mathcal{E}_k$ there exist $\alpha.(\alpha - 1)$ numbers denoted $\eta_{l,k}$ verifying

$$\eta_{l,k} \geq 0 \text{ such that } | < A_{lk}U_k, g_l >_l | \leq \eta_{l,k}|U_k|_k.|U_l|_l, \tag{13}$$

where for $l \neq k, \eta_{l,k}$ denotes in fact the matrix norm of A_{lk}.

Then practically

- on the one hand, concerning inequality (12) (see [14]), if for $l = 1, \dots, \alpha$, the diagonal blocks A_{ll} of matrix A are either strongly defined positive in the case where the subspaces $\mathcal{E}_l$ are normed by the euclidean norm and so verifying

$$< A_{ll}U_l, U_l > \geq \eta_{l,l}.|U_l|_2^2 \tag{14}$$

where $\eta_{l,l}$ is the positive smallest eigenvalue of A_{ll} or else if A_{ll} are strongly diagonal dominant by row then satisfying

$$a_{l,l} - \sum_{k \neq l}^{dim(A_{l,l})} |a_{l,k}| \geq \eta_{l,l}, \forall l \in \{1, \dots, dim(A_{l,l})\}, \tag{15}$$

when the space $\mathcal{E}_l$ is normed by the uniform-norm,

- and on the other hand the matrix norms $\eta_{l,k} \ k \neq l$ of the off-blocks A_{lk} are of relatively low modulus,

then, when these properties are satisfied, we can define the $\alpha \times \alpha$ matrix $N = (\eta_{l,k})$ by

$$N = \begin{cases} \eta_{l,l}, \text{ if } l = k \\ -\eta_{l,k}, \text{ if } l \neq k \end{cases}.$$

Remark 2. Note that when a linear boundary value problem is discretized by classical schemes such that finite difference method or finite volume method and appropriate finite element method especially if the angle condition is satisfied, then assumption (14) or (15) is satisfied (for more details see [5]). If in addition parabolic or second order hyperbolic evolution problems are considered, these problems being discretized by implicit or semi-implicit time marching scheme, then, for sufficiently small time step, once again (14) or (15) is satisfied.

Consider now problems (2) or (6); let us subtract from the system satisfied by the exact solution the system where we inject the components of the current iteration and multiply the result by $g_l(U_l)$; taking account of properties (3) or (7), due to the monotonicity of the operator disturbing the affine system, we obtain finally the following vectorial Lipschitz condition

[1] i.e. a bilinear form, from a normed vector space $\mathcal{E}_l \times \mathcal{E}'_l$ onto $\mathfrak{R}$.

$$|U_l - U_l^\star|_l \leq \sum_{k\neq l}^{\alpha} \frac{\eta_{l,k}}{\eta_{l,l}} |U_k - U_k^\star|_k \ , \forall l \in \{1,\dots,\alpha\} \tag{16}$$

Let us define the matrix J by

$$J_{l,k} = \begin{cases} 0, \text{ if } l = k \\ \frac{\eta_{l,k}}{\eta_{l,l}}, \text{ if } l \neq k \end{cases} .$$

Note that for the $\alpha-$block decomposition J is the Jacobi matrix of N and is a non-negative matrix. If in addition the spectral radius of J is less than one, then J is a contraction matrix for the $\alpha-$ subspace decomposition of $\mathcal{E}$, each subspace $\mathcal{E}_l$ being normed by a classical l_r-norm, with $r = 2$ in an Hilbertian background or $r = \infty$ otherwise in a non-Hilbertian context. Thus, for the $\alpha-$block decomposition, let us consider the vectorial norm $q(.)$ defined on $\mathcal{E}$ by

$$q(U) = \{|U_1|_1, \dots, |U_\alpha|_\alpha\}. \tag{17}$$

Then, using this notation, (16) is written as follows

$$q(F(U) - F(U^\star)) \leq J.q(U - U^\star). \tag{18}$$

Consider now the parallel solution of problems (2) or (6), when N is an M-matrix (see [15] and [16]); according to a result stated by J.C. Miellou [7], we obtain easily a result which ensures the convergence of parallel asynchronous iterative methods.

Proposition 1. *Consider the problems (2) or (6) decomposed with the $\alpha-$block decomposition. Assume that the matrix N associated to the matrix A is an M-matrix with respect to the l_r-norms defined in $\mathcal{E}_l, l = 1,\dots,\alpha$, with $r = 2$ or $r = \infty$ and that the affine operator $AU - G$ is perturbed by a multivalued monotone maximal operator. Then, we can associate to these previous problems defined in the space $\mathcal{E}$ a contracting fixed point mapping F, associated to the block - decomposition, and there exists one and only one fixed point $U^\star$ also unique solution of the discretized problems (2) or (6). In addition, the parallel asynchronous iterative methods used to solve problems (2) or (6) converge toward $U^\star$.*

Remark 3. Note also that, instead of the $\alpha-$block decomposition, we can consider also a point to point decomposition; then if A is an M-matrix, by using the same approach, we can easily obtain the same result of convergence of parallel asynchronous method with respect to the "by-point"-decomposition. In addition, according to a result stated in [17] the parallel asynchronous algorithms converge for every coarser decomposition, i.e. a $\beta-$decomposition, where $\beta >> \alpha$.

Remark 4. According to the result stated in Proposition 1, note that, for the study of the behavior of the parallel asynchronous algorithms, it is not necessary that A was an M-Matrix. Indeed it is sufficient that the numbers $\eta_{l,l}$ and $\eta_{l,k}$ exist such that the matrix N is an M-matrix.

Remark 5. When J is irreducible note also that the spectral radius of J denoted $\rho(J)$ allows to estimate the speed of convergence of the iterative process. According to [18] when J is reducible the number λ satisfying $\lambda \in [\rho(J), 1[$ allows also to estimate the speed of convergence (see [5] and [18]). Indeed by using a result based on the Perron-Frobenius theory (see [15] to [18]) we have

$$J.\Gamma \leq \lambda.\Gamma, \lambda \in [\rho(J), 1[: \tag{19}$$

According to [18] $\lambda = \rho(J)$ if J is irreducible.

Then, for the block decomposition, (19) yields to the following result

$$\|F(U^\star) - F(V)\|_{\Gamma,\infty} \leq \lambda \|U^\star - V\|_{\Gamma,\infty}, \forall V \in \mathcal{E},$$

where $\|.\|_{\Gamma,\infty}$ is the uniform weighted norm associated to the $\alpha-$block decomposition and defined by

$$\|V\|_{\Gamma,\infty} = \max_{1\leq j\leq\alpha} \left(\frac{|V_j|}{\Gamma_j}\right); \tag{20}$$

so, by applying Theorem's El Tarazi [19], we conclude once again to the convergence of parallel asynchronous iterative methods.

2.4. Schwarz alternating method

In terms of efficiency of the implemented parallel algorithms, it is necessary that they have a sufficiently high granularity. Indeed, too small calculation process sizes will have a negative effect on the efficiency of the implemented methods, an increase in communication cost and synchronizations between the computation processes, which will degrade the performance of the parallel methods. Therefore developers gather computational tasks together in order to avoid this type of algorithmic drawback.

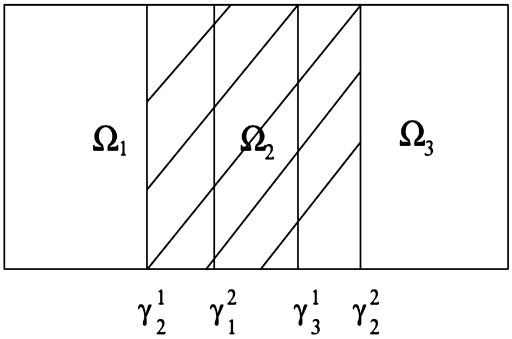


Fig. 1. General scheme of overlapping in Schwarz's alternating method in the 2D case.

For example in the case of the numerical solution of partial differential equations, this type of grouping of computational tasks leads to the development of subdomain methods. There are various kinds of subdomain methods; on the one hand, subdomain methods without overlapping corresponding in fact to a large block relaxation algorithm and on the other hand, subdomain methods with overlapping such as the Schwarz alternating algorithm where the subdomains overlap each other as shown in Fig. 1.

Subdomain methods without overlap consist in grouping several adjacent blocks of the vector to be calculated in order to obtain a block of larger size. The analysis of the behavior of the iterative process with high granularity thus obtained follows from the results obtained previously and mentioned in paragraph 2.3. Indeed, if the discretization matrix is an M-matrix, a situation to which we can often reduce ourselves by choosing appropriate discretization schemes, we have seen that in this case the asynchronous parallel methods analyzed by contraction techniques were convergent whatever be the coarser decomposition (see [5–17]). As mentioned above this result can also be extended to situations where the discretization matrix is not an M-matrix, but has appropriate properties.

Furthermore, subdomain methods with overlapping can also be used, such as Schwarz's alternating method; when the overlap between blocks is sufficiently large, this type of method accelerates the convergence and is well suited to the parallel solution of boundary values problems (see [20]). Parallel asynchronous Schwarz algorithms for two processors consist in solving at each iteration

$$\begin{cases} \Lambda_1.u_1^{p+1} = f_1 \text{ on } \Omega_1 \\ B_1.u_1^{p+1} = g_1 \text{ on } \Gamma \cap \Omega_1 \\ \quad u_1^{p+1} = \tilde{u}_2^p \text{ on } \partial\Omega_1 \cap \Omega_2 \end{cases} \quad \text{and} \quad \begin{cases} \Lambda_2.u_2^{p+1} = f_2 \text{ on } \Omega_2 \\ B_2.u_2^{p+1} = g_2 \text{ on } \Gamma \cap \Omega_2 \\ \quad u_2^{p+1} = \tilde{u}_1^p \text{ on } \partial\Omega_2 \cap \Omega_1 \end{cases} \tag{21}$$

where $\tilde{u}_1^p$ and $\tilde{u}_2^p$ denote the available values of the components of the iterate vector (u_1, u_2) at the current iteration. In the synchronous algorithm, we have $\tilde{u}_1^p = u_1^p$ and $\tilde{u}_2^p = u_2^p$. Besides, in the classical asynchronous algorithm (see [6] and [7]), these components may be delayed as follows $\tilde{u}_1^p = u_1^{\rho_1(p)}$ and $\tilde{u}_2^p = u_2^{\rho_2(p)}$, with $\rho_i(p) \geq 0, i = 1, 2$. Finally, in the case of asynchronous algorithm with flexible communications (see [21]), $\tilde{u}_i^p$ are not necessarily associated to components that are labeled by an outer iteration number as communication may occur at any time. Then, in this class of method, partial updates, i.e. the current value of any component of the iterate vector, can be used at any time in the computation. Thus, flexible data exchanges between processors are allowed; as a consequence, the coupling between communication and computation can be improved.

In this case, to analyze the behavior of these asynchronous methods, a result of D.J. Evans and J. Van Deeren [22] is used. These last authors established that, if A is an M-matrix, the matrix $\bar{A}$ obtained by the Schwarz augmentation process is also an M-matrix. Therefore, for solving linear problems, the behavior of the asynchronous Schwarz method can be immediately analyzed both by contraction techniques.

Due to the monotony properties of increasing diagonal operators, both when using non-overlapping subdomain methods and subdomain methods with overlapping, the convergence of these methods applied to solving multivalued pseudo-linear problems is trivially obtained (see [5]), this latter situation corresponding to the case of classical partial differential equations (P.D.E.) whose solution is subjected to constraints.

Finally, this analysis of subdomain methods with or without overlapping is still valid for the block numberings usually used, whether it is the lexicographic numbering of the blocks or the red-black numbering of these latter [23].

2.5. Termination of parallel asynchronous algorithms

The previous methods being iterative in nature must be stopped when the iterated values are stabilized. They are thus completed by a test of stopping the iterations. In the case of synchronous parallel algorithms, this iteration stopping test is identical to the one implemented in the sequential case. For asynchronous parallel algorithms the iteration stopping test is much more delicate to implement. Thus, in this latter case we will distinguish the computer science approach and the numerical analysis approach and to stop the iterative process we have to combine these two approaches.

Concerning the computer science approach several methods was developed by various authors and we refer the reader to [24–30].

In our implementations a stopping test of the iterations is considered first from the computer science point of view; thus we have used the circulation of a token corresponding to the simplest implementation of stopping test but coupled as we will see below by a test derived from numerical analysis considerations. According to [31] - [32] in the case of parallel asynchronous methods we consider a decentralized algorithm where convergence is achieved in this way. Each processor updates the components of the iterate vector associated with each one's own block and computes the residual norm attached to this block in order to participate to the convergence detection. Convergence occurs when a given predicate on a global state is true. A usual predicate corresponds to the fact that the iterate vector generated by the asynchronous iterative algorithm is sufficiently close to the solution of the problem (see [8], page 580); thus, on every process, termination detection is obtained when the norm of the local residue remains under a given threshold after two successive updates of the component. The Algorithm 1 below illustrates the termination management principle of asynchronous parallel algorithms. For more details concerning the implementation and the convergence detection of the considered parallel asynchronous methods, the reader is referred to [32].

Algorithm 1: Convergence manager processor algorithm.

```
begin
    while no reception of termination command for algorithm of p + 1
        if Barrier then
            Wait for all computing processors
            Send notification for restarting computing processors
            Send boundary values to neighboring processes
        end
        if convergence then
            Receive local convergence
            Send global convergence
        end
    end
end
```

Concerning the numerical analysis approach it can be proved that the successive iterates are located in nested sets centered in the solution (see [8], [33] and [34]), which allows to implement efficient numerical stopping tests insofar as one can give an estimate of the diameters of each nested set and stop the iterative process when a diameter is smaller than a given tolerance. Consequently the coupling of the stopping tests derived from computer science and numerical analysis considerations allows to obtain an efficient termination of the parallel asynchronous iteration. In previous studies (see [5] and [35]) we have implemented these methods on grids constituted by heterogeneous and distant machines; we could observe that asynchronous algorithms are very efficient when there was a lot of synchronization between the processors.

3. Cloud simulation

The implementation of the studied problems is briefly detailed in the present section. Matrix and right hand side creation have been implemented by a parallel way. For each simulation, we have considered a cubic domain Ω contained in the 3D space.

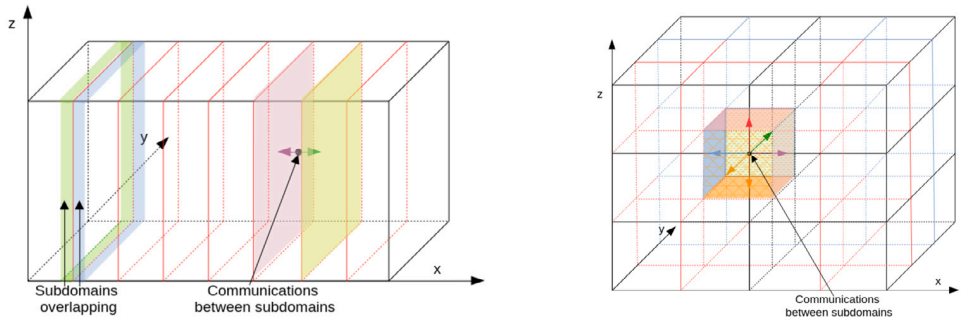
In the case of parallel subdomain algorithms without overlapping a block relaxation method is performed on each process and only one subdomain is assigned to each processor. A subdomain is constituted by gathering adjacent blocks.

Moreover, to solve the global problem, we also consider an overlapping subdomain method corresponding to the alternating Schwarz method. From an implementation point of view in the two-dimensional case where the domain Ω is divided into slices, Fig. 1 represents the schematic diagram of this method. However, in the present study the domain Ω is three-dimensional. Thus we can also consider an analogous slicing, shown in Fig. 2a, made up in this case of overlapping parallelepipeds; so, as in the two-dimensional case, each subdomain communicates with its left and right neighbors, thus minimizing communication costs. But, in the three-dimensional case, we can also consider another partition where the domain Ω is decomposed into small overlapping three-dimensional overlapping elements, for example into small parallelograms around each subdomain as shown in Fig. 2b. So instead of having two exchanges with its right and left neighbors as in the previous case, this small subdomain will perform data exchanges with six overlapping neighbors, i.e. with neighbors located at the top, bottom, right, left, front and back. So, in this last case of decomposition, there are three times as many communications but the behavior of the algorithm can be more multiplicative.

The principle of implementation of parallel asynchronous and synchronous iterative subdomain methods is described by Algorithm 2.

Algorithm 2: Principle of implementation.

```
while not global convergence do
    for each subdomain do
        Perform communications of subdomain boundary values
        Perform block relaxation resolution
    end
end
```



(a) Decomposition of Ω in slices. (b) Decomposition of Ω in cubes.

Fig. 2. Various decomposition of Ω in vertical slices and cubes.

In accordance with the model describing asynchronous parallel iterations mathematically and recursively, modeled by (10)-(11), the Algorithm 3 below gives the general scheme.

Algorithm 3: Parallel asynchronous iterative algorithm.

$U^0 \leftarrow (U_1^0, \ldots, U_\alpha^0)$
$r \leftarrow 0$
while $norm > tolerance$
  $r \leftarrow r+1$
  for $l \leftarrow 1$ **to** α **do**
    if $l \in s(r)$ **then**
      $U_l^{r+1} \leftarrow F_l(U_1^{\rho_1(r)}, \ldots, U_k^{\rho_k(r)}, \ldots, U_\alpha^{\rho_\alpha(r)})$
    end
    else
      $U_l^{r+1} \leftarrow U_l^r$
    end
  end
  Computing norm
end

Furthermore, the following Algorithm 4 presents the principle of execution of the parallel method by the p-th processor.

Algorithm 4: Algorithm executed by the p-th processor.

begin
  // p processors and $rank$ current processor
  Test reception of token from process $(rank-1) \bmod p$
  if *token is received* **then**
    Update token
    Submit token reception for the next round
    Wait for the end of the previous token send
    Submit token send to process $(rank+1) \bmod p$
  end
end

Note that in the case of time evolution, it is essential to set a synchronization barrier after each time step.

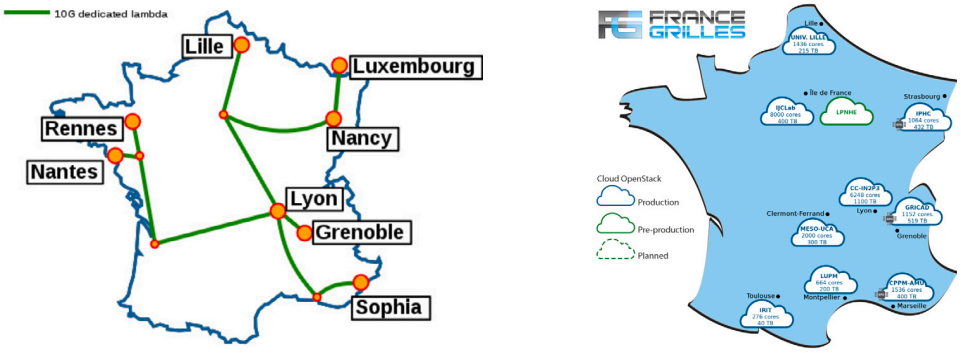
For comparison of synchronous and asynchronous performances, the number τ which is the ratio of elapsed time between synchronous and asynchronous methods given by

$$\tau = \frac{synchronous\ elapsed\ time\ algorithm}{asynchronous\ elapsed\ time\ algorithm}$$

is used; so τ is significant to achieve a comparison between the two communication modes. Indeed, in the case of heterogeneous architecture the notion of speed up and efficiency is not relevant since it is difficult to decide which machine is the reference and also all the more so since for large algebraic systems it is not possible to have the restitution times in sequential mode.

3.1. Parallel experiments on cloud architecture

Grid architecture is designed as a distributed system and file manager with the advantage of shared resources but on one hand the disadvantage of a tendency towards complexity for Application Programming Interface (API) which is a way for computers programs to communicate with each other, and on the other hand the deployment of services on this infrastructure.



(a) Platform architecture of Grid'5000. (b) Cloud resources of FG Cloud.

Fig. 3. Grid'5000 and France Grid resources.

For example Grid'5000 is a platform (see Fig. 3a), which provides access to a large amount of resources (8 sites, 800 compute-nodes), and gives an easy access to a wide variety of hardware technologies; this platform is particularly suitable to carry out experiments.

On the other hand the Cloud has emerged from the convergence of several ideas following the maturation of technology for visualization with a simplified API look and feel and also large computing capacity to meet resource requirements.

The Cloud computing, as described in [36], is a technology that enables the storage, management, and access of data and computing resources over the Internet, rather than storing them locally on individual computers or servers. There are several service models in the cloud, including public cloud, private cloud, and hybrid cloud, each with its own advantages and disadvantages. It is based on the virtualization of computing resources, such as storage, computing power, memory, and applications. The layers of cloud computing are generally divided into three main categories: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS).

The different services include:

- Infrastructure as a Service (IaaS) that provides basic computing resources such as virtual machines, storage, and networks. Users have more direct control over the operating system and applications. Its advantages include a customizable environment, full control, accessibility at any time, and usability through a simple API.
- Platform as a Service (PaaS) that offers a development environment for developers, providing tools and services for application development, deployment, and management. It allows hiding the underlying infrastructure management. Programmers can thus work at a higher level.
- Software as a Service (SaaS) intended for end users, providing ready-to-use software applications via the Internet. Users do not need to worry about managing the underlying infrastructure. Its advantages include ease of use with no software deployment, a web interface, and high accessibility. This service enables users to connect and share their applications over the Internet.

For this experimental study, we conducted computational experiments on France Grilles-Cloud (see Fig. 3b). This platform provides cloud services to users, offering on-demand access to computing, storage, network resources, cluster instantiation, web service instantiation tailored to the scientific community, and algorithm testing.

Specifically, we utilized the Computer Center of the National Institute of Nuclear and Particle Physics (IN2P3), a research unit supported by the French CNRS. IN2P3 focuses on coordinating research in the fields of particle physics, nuclear physics, and astroparticle physics. Researchers at this institute have access to servers shared through a scheduling system for computational tasks. These tasks may be either sequential or parallel in nature. The computing servers, which operate on a Linux system, predominantly feature Intel x86_64 processors, complemented by some special graphic processors. These resources can be used locally or accessed remotely via a computing infrastructure. The Computing Center is equipped with external links offering speeds of up to 100 Gbit/s and an internal network with a 400 Gbit/s backbone.

Our parallel simulations were conducted using allocated resources on the aforementioned platform, with each instance serving as a machine that runs our workloads in the cloud, starting as an identical clone of all other instances.

At the IN2P3 Computer Center, OpenStack, a collection of open-source software modulus and tools for creating and managing both public and private cloud infrastructures, is employed. Furthermore, OpenStackClient, a command-line client for OpenStack, consolidates the command set for various APIs into a single shell with a uniform command structure for instance management.

Once an SSH key pair is generated, a Secure Shell Protocol (SSH) session can be established with the instances. It is essential to provide Openstack with the public key to inject into the instances for the root account. After that, we work with the available image and flavor lists. Flavors represent different types of virtual machines and define their characteristics, including the number of CPUs (Central Processing Units) and RAM size. After creating instances and associating a floating IP address with them to ensure network connectivity, we were able to start them and conduct our experiments.

Table 1
Openstack flavors list.

Name	RAM	Disk	vCPUs
4CPU_8GB-RAM	8192	64	4
1CPU_8GB-RAM	8192	64	1
2CPU_4GB-RAM	4096	64	2
1CPU_12GB-RAM	12288	64	1
1CPU_2GB-RAM	2048	64	1
8CPU_32GB-RAM	32768	32	8
6CPU_12GB-RAM	12288	64	6

Table 2
Openstack instance list.

Flavor	Instance Name	vCPU	Image (OS)
8CPU_32GB-RAM	myvm1	1	Ubuntu-20.04
8CPU_32GB-RAM	myvm2	1	Ubuntu-20.04
1CPU_12GB-RAM	myvm3	1	Ubuntu-20.04
6CPU_12GB-RAM	myvm4	1	Ubuntu-20.04
1CPU_8GB-RAM	myvm5	1	Ubuntu-20.04
4CPU_8GB-RAM	myvm6	1	Ubuntu-20.04
8CPU_32GB-RAM	myvm7	1	Ubuntu-20.04
8CPU_32GB-RAM	myvm8	1	Ubuntu-20.04

Remark 6. Cloud computing offers numerous advantages but also comes with certain specificities, such as the heterogeneous nature of the provided resources, including hardware specifications, server performance, and virtual machines (VMs) that constitute the cloud infrastructure. This diversity in resources allows users to customize their configurations to meet specific requirements.

Two other important specificities are geographic location, as cloud service provider data centers are distributed worldwide. Network performance is another factor of heterogeneity, as some instances may offer fast network access, while others may have more modest bandwidth.

The physical location of resources and network performance can influence network latency, which represents the time required for a data packet to move from one point to another across the network, passing through various network links and devices that process and transmit data packets over different network transmission mediums.

Certain types of high-performance computing applications require low network latency to fulfill their computational demands. High latency can lead to performance degradation, which can have a significant impact, especially in the context of cloud computing simulations.

For our experiments, we had the opportunity to use various resources (flavors) on the IN2P3 computing center architecture (see Table 1). We used the IaaS mode, so the implementation in the cloud environment is straightforward and presents no significant challenges.

Once the resources were requested, we acquired them without knowledge of their specific hardware configuration. Additionally, we had no information regarding the possible sharing of physical machines among multiple users.

However, to fully consider the specificities of commercial Cloud Computing and their heterogeneity, as mentioned in Remark 6, we used these resources while taking into account the multiplicity of virtual CPUs (vCPUs) on the same machine and shared memory allocations among vCPUs. Additionally, since the machines were on the same site in the IN2P3 computing center, we also set up different network configurations to simulate geographically distant behavior and different network access rates.

For simplicity we also used a uniform operating system. We developed our codes using the C or Fortran languages, along with the MPI library. We also set up scripts to deploy the codes on the cloud resources, including the libraries necessary for the MPI environment. Additional scripts were used to run the simulations and retrieve the results.

In the case of asynchronous iterations, point to point communications between two processes have been implemented using non-blocking send and receive communications. A test function is used in order to allow any processor to continue the computations without having to wait for the completion of any pending send or receive operations. Idle times due to synchronizations in message passing are suppressed in this way. On the other hand, the parallel synchronous iterative schemes are implemented by using blocking send and receive communications. One has to take care about the deadlock issue when implementing synchronous communications using blocking communications.

The number of machines used was set at 8 based on the memory size used ($\equiv 8$ Go) by our algorithms and to minimize disk swapping (see Table 2 and Fig. 4).

However, as we will see in the analysis of the results, additional machines are unnecessary and can have a negative impact on the performance of synchronous parallel algorithms.

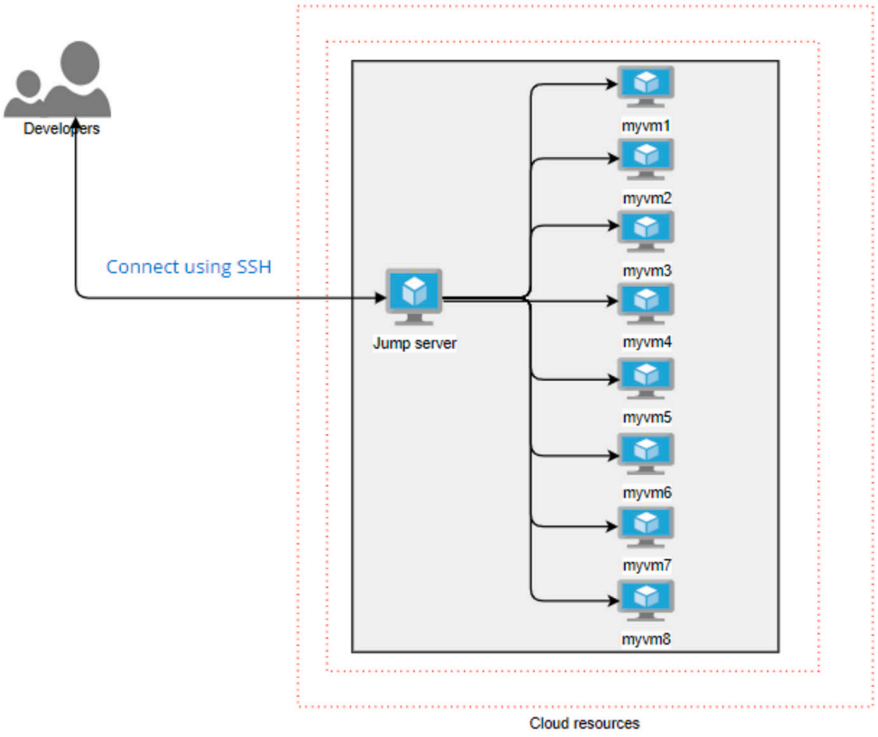
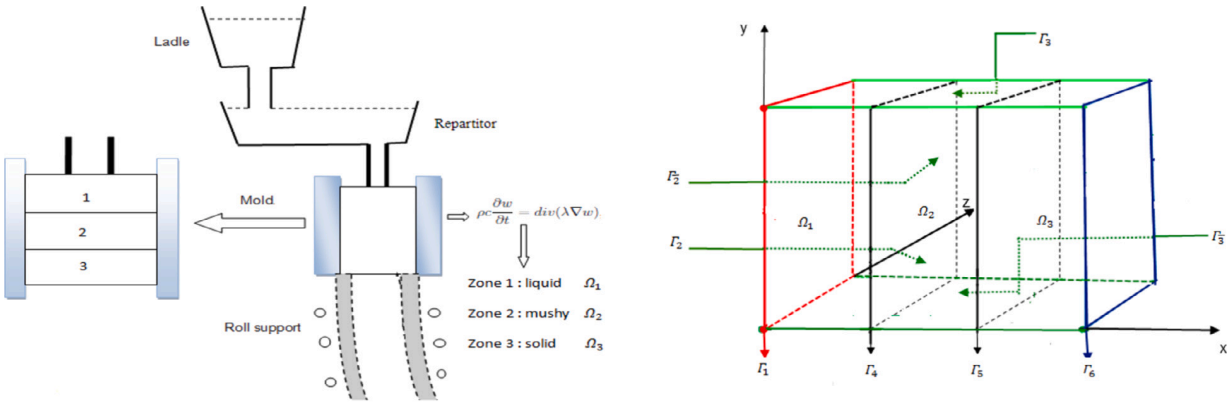


Fig. 4. Cloud resources.



(a) General scheme of continuous casting. (b) The domain Ω divided in three areas.

Fig. 5. Global representation of the domain Ω.

4. Cloud simulation computation of steel solidification in continuous casting

4.1. Process description: physical formulation

In steel industry, the continuous casting is a process between the metal making and rolling. This process, shortly described in Fig. 5a, allows the transformation of a liquid metal into a solid metal in a continuous way.

During the process the liquid metal is water cooled during the progress in an adapted machine. The transformation of the liquid phase into the solid phase takes place in an intermediary zone the so called mushy zone. To summarize the cooling solidification process, we can describe it in the following manner: the steel goes through three phases in three areas denoted by Ω_1 for the liquid area, Ω_2 for the mushy area and Ω_3 for the solid area. The initial temperature in Ω_1 is about 1500 °C, while it remains equal to 800 °C and 600 °C, respectively in Ω_2 and Ω_3. The liquid steel is poured into a mould cooled by water. To describe this operation, it is assumed that the exchange of heat in the mould is uniform and the water jet cooling and thermal contact with the rollers constituted by cylinders will be modeled by the condition of convection and / or radiation. It is customary to assume that the contact

surfaces with the ambient air are subject to conditions of the same type. The heat exchange that occurs between the subdomains liquid-mushy and mushy-solid is modeled by a Dirichlet boundary condition at the interfaces. Every cooling zone is characterized by its temperature i.e. we have to find successively the temperature field in the liquid zone Ω_1, then in the mushy zone Ω_2 and finally in the solid zone Ω_3.

The mathematical model is described by the heat equation in three dimensional space with the knowledge of initial condition and six appropriate boundary conditions describing the physical phenomenon for each subdomain $\Omega_{\mathcal{I}}, \mathcal{I} = 1,2,3$ (see Fig. 5); these boundary conditions are constituted by non-homogeneous linear and nonlinear mixed Dirichlet-Neumann and Fourier linear and nonlinear boundary conditions due to radiation phenomenon, well described by the Stefan law on some parts of the boundaries. Moreover in order to take into account the positivity of the temperature in each area, the mathematical model describing the steel solidification is modified by perturbing the affine part of the continuous operator modeling the heat diffusion in each subdomain $\Omega_{\mathcal{I}}, \mathcal{I} = 1,2,3$ by a diagonal multivalued monotone continuous operator which is in fact the subdifferential of the indicator function of the convex sets. Indeed even if physically it is reasonable to think that in these three zones the temperature is strictly positive, it is not easy to prove mathematically this property; consequently, it is necessary to change the mathematical model and we are therefore drawn into solving three coupled variational inequalities governing the temperature defined in each zone. Thus taking into account both of the constraint of temperature positivity and of the nonlinearity derived from the radiation phenomenon we have to solve on each area a multivalued formulation of the problem, similar than (6).

After discretization on one hand of the evolution part of the problems by an implicit time marching scheme and on the other hand by using classical finite difference scheme for the spatial discretization with an uniform mesh, we have to solve, by a numerical way, a sequence of large coupled stationary multivalued strongly nonlinear problems.

In order to obtain accurate results, it is necessary to choose very small spatial discretization step-size which leads to solve very large multivalued algebraic systems. Owing to the great size of such system an iterative algorithm is more appropriate for the solution of the algebraic systems to solve at each time step. Thus we have implemented sequential projected point Newton - relaxation algorithms, corresponding to a coupling of the point projection Newton method for points belonging to the part of the boundary where the Stefan law occurs, with the projected relaxation method, like the point Gauss-Seidel method, for the interior points of the zones. But such numerical solution is then time consuming. Thus, in order to reduce the elapsed time and taking into account the well-known property of M-matrix of the spatial discretization matrices on one hand and also the property of monotony for the nonlinear discretized operators modeling the Stefan law on some parts of the boundary and also the monotony of the subdifferential of the indicator function of the convex sets, we obtain sufficient conditions for the convergence of the considered parallel synchronous and asynchronous two-stage methods for the solution of multivalued nonlinear algebraic systems.

4.2. Multivalued mathematical model

Let $\Omega \subset \mathfrak{R}^d, d = 3$, be a bounded open domain with boundary denoted by $\partial\Omega$. We consider that $\Omega = \bigcup_{\mathcal{I}=1}^{3} \Omega_{\mathcal{I}}$ such that $\Omega_{\mathcal{I}} \cap \Omega_{\mathcal{J}} = \emptyset,\ \mathcal{I} \neq \mathcal{J}$. For the target industrial application note that it is sufficient to consider that Ω has a parallelepipedic shape (see Fig. 5b); so let $\mathbf{x} \equiv (x,y,z)$ and for $\mathcal{I} = 1,2,3$, let $\gamma_{\mathcal{I}} \equiv \Gamma_{\mathcal{I},2} \cup \Gamma_{\mathcal{I},3} \cup \Gamma_{\mathcal{I},\bar{2}} \cup \Gamma_{\mathcal{I},\bar{3}}$ with

$$\Gamma_1 = \{0 \leq y,\ z \leq 1,\ x = 0\},$$

and

$$\Gamma_{\mathcal{I},2} = \{z = 0, 0 \leq y \leq 1,\ \frac{\mathcal{I}-1}{3} < x < \frac{\mathcal{I}}{3}\}; \Gamma_{\mathcal{I},3} = \{y = 1, 0 \leq z \leq 1,\ \frac{\mathcal{I}-1}{3} < x < \frac{\mathcal{I}}{3}\},$$
$$\Gamma_{\mathcal{I},\bar{2}} = \{z = 1, 0 \leq y \leq 1,\ \frac{\mathcal{I}-1}{3} < x < \frac{\mathcal{I}}{3}\}; \Gamma_{\mathcal{I},\bar{3}} = \{y = 0, 0 \leq z \leq 1, \frac{\mathcal{I}-1}{3} < x < \frac{\mathcal{I}}{3}\}.$$

Clearly $\Gamma_{\mathcal{I},2}$ (respectively $\Gamma_{\mathcal{I},\bar{2}}$) shows the front (respectively backward) boundary of $\Omega_{\mathcal{I}}$; similarly $\Gamma_{\mathcal{I},3}$ (respectively $\Gamma_{\mathcal{I},\bar{3}}$) represents the upper side (respectively under side) boundary of $\Omega_{\mathcal{I}}$. The symbols $\bar{2}$ and $\bar{3}$, are a convenient notation to simply define boundaries $\gamma_{\mathcal{I}}, \mathcal{I} = 2,3$, and allows a unified notation of these.

Let also Γ_4 and Γ_5 be the interfaces between Ω_1 and Ω_2 on one hand, and between Ω_2 and Ω_3 on the other hand; thus Γ_4 and Γ_5 are defined by

$$\Gamma_4 = \{0 \leq y,\ z \leq 1,\ x = \frac{1}{3}\}, \Gamma_5 = \{0 \leq y,\ z \leq 1,\ x = \frac{2}{3}\}.$$

The last frontier delimiting the domain Ω is defined by

$$\Gamma_6 = \{0 \leq y,\ z \leq 1,\ x = 1\}.$$

Then we have to find successively the temperature in each subdomain $\Omega_{\mathcal{I}},\ \mathcal{I} = 1,2,3$ satisfying three coupled parabolic boundary value problems (for more details see [37]).

Then we consider the classical Euler temporal discretization of such problems by an implicit time marching scheme. Let us denote by $w_{\mathcal{I}}(\mathbf{x})$, for $\mathcal{I} = 1,2,3$ the solution of the stationary obstacle problems where according to positivity assumption we have

$$w_{\mathcal{I}}(\mathbf{x},t) \geq 0, \text{ for } \mathcal{I} = 1,2,3. \tag{22}$$

So, at each time step, we have to solve three stationary multivalued problems on each subdomain $\Omega_{\mathcal{I}}$, $\mathcal{I}=1,2,3$. Thus the three problems to solve are formulated by three multivalued problems respectively defined in each subdomain $\Omega_{\mathcal{I}}$, $\mathcal{I}=1,2,3$, as follows, where $\mathbf{E}_{\mathcal{I}}, \mathcal{I}=1,2,3$ denote appropriate Sobolev vector spaces:
in the liquid zone Ω_1, we have to find $w_1 \in \mathbf{E}_1$

$$\begin{cases} -div(\lambda_1 \nabla w_1) + \frac{\rho_1 c_1}{\Delta t} w_1 - \bar{g}_1 + \partial \Psi_{\mathcal{K}_1}(w_1) \ni 0 & \\ -\lambda_1 \frac{\partial w_1}{\partial n} = \Phi_{imp}, & on\ \ \Gamma_1, \\ -\lambda_1 \frac{\partial w_1}{\partial n} = \chi_1(w_1), & on\ \ \gamma_1, \\ -\lambda_1 \frac{\partial w_1}{\partial n} = \Upsilon_1(w_1), & on\ \ \Gamma_4 \end{cases} \tag{23}$$

in the mushy zone Ω_2, we have to find $w_2 \in \mathbf{E}_2$

$$\begin{cases} -div(\lambda_2 \nabla w_2) + \frac{\rho_2 c_2}{\Delta t} w_2 - \bar{g}_2 + \partial \Psi_{\mathcal{K}_2}(w_2) \ni 0 & \\ w_2 = w_{1|\Gamma_4}, & on\ \ \Gamma_4 \\ -\lambda_2 \frac{\partial w_2}{\partial n} = \chi_2(w_2), & on\ \ \gamma_2, \\ -\lambda_2 \frac{\partial w_2}{\partial n} = \Upsilon_2(w_2), & on\ \ \Gamma_5 \end{cases} \tag{24}$$

in the solid zone Ω_3, we have to find $w_3 \in \mathbf{E}_3$

$$\begin{cases} -div(\lambda_3 \nabla w_3) + \frac{\rho_3 c_3}{\Delta t} w_3 - \bar{g}_3 + \partial \Psi_{\mathcal{K}_3}(w_3) \ni 0 & \\ w_3 = w_{2|\Gamma_5}, & on\ \ \Gamma_5 \\ -\lambda_3 \frac{\partial w_3}{\partial n} = \chi_3(w_3), & on\ \ \gamma_3, \\ w_3 = w_{imp}, & on\ \ \Gamma_6 \end{cases} \tag{25}$$

where

$$\chi_{\mathcal{I}}(u_{\mathcal{I}}) = h_{cv,\mathcal{I}}(u_{\mathcal{I}} - u_{ext}) + \xi\delta(u_{\mathcal{I}}^4 - u_{ext}^4), \quad \mathcal{I}=1,2,3, \tag{26}$$

describe the radiation phenomenon, modeled by the Stefan law while for $\mathcal{I}=1,2$,

$$\Upsilon_{\mathcal{I}}(u_{\mathcal{I}}) = \frac{1}{R}(u_{\mathcal{I}} - u_{cst}),$$

where $g_{\mathcal{I}} = \frac{\rho_{\mathcal{I}} c_{\mathcal{I}}}{\Delta t} w_{\mathcal{I}}^{prec}$, $w_{\mathcal{I}}^{prec}$, $\mathcal{I}=1,2,3$ are given thanks to the result obtained at the previous time step and Δt is the time step.

In (23)-(25), the affine operator is perturbed by a diagonal monotone maximal operator constituted by the sum of two such operators, i.e. $\chi_{\mathcal{I}}$ and the multivalued increasing diagonal continuous operator $\partial\Psi_{\mathcal{K}_{\mathcal{I}}}$ given by for $\mathcal{I}=1,2,3$

$$(\partial \Psi_{\mathcal{K}_{\mathcal{I}}}(w_i))_i = \begin{cases} \emptyset, & \text{if } w_i < 0, \\ ]-\infty, 0], & \text{if } w_i = 0, \\ 0, & \text{if } w_i \in \mathcal{K}_{\mathcal{I}}. \end{cases}$$

Finally for $\mathcal{I}=1,2,3$, $c_{\mathcal{I}}$ is the specific heat capacity, $\rho_{\mathcal{I}}$ is the metal density, $h_{cv,\mathcal{I}}$ is the coefficient of exchange by convection, $u_{\mathcal{I}}$ is the temperature, Φ_{imp} is the flux, u_{imp} is the temperature imposed, u_{ext} is the temperature of environment, u_{cst} is the constant temperature defined at the interface Γ_4 and Γ_5 between two consecutive subdomains where the thermal exchange is performed, R is the thermal resistance, ξ is the emissivity, δ is the constant of Stefan-Boltzman and $\vec{n}$ is the outward normal vector. In the sequel, such physical coefficients are assumed to be constant.

4.3. Numerical solution

4.3.1. Discretization

Thanks to the approximations on one hand by an implicit time marching scheme and on the other hand by spatial classical finite-difference method, the discretization matrices are M-matrices and we are therefore in the theoretical framework presented in section 2 for the analysis of the behavior of the iterative algorithms used for the resolution of the problem like (6).

4.3.2. Two-stage methods for pseudo-linear problem

For pseudo-linear problem such (6), we give below a particular class of multisplitting method which unifies the presentation of subdomain methods; in the literature this method is usually called two-stage method; such two-stage method corresponds to the fact that each discretized subsystem is solved by an algorithm with two iteration levels the first one being the iterative method applied to the overall problem and the second consisting in the use of another iterative method to solve each subproblem resulting from the large block decomposition. Let us assume that for each problem defined in $\Omega_{\mathcal{I}}$ we have m regular splitting $\mathcal{A} = \mathcal{M}^l - \mathcal{N}^l$, $l=1,...,m$, of the matrix $\mathcal{A}$ corresponding of the discretization matrix in each area $\Omega_{\mathcal{I}}$ [16]; in such splittings for $l=1,...,m$, $\mathcal{M}^l$ refers to

a $\alpha-$block diagonal matrix. Then in each Ω_I we have α subproblems related to the decomposition of $\mathcal{M}^l$ in α blocks. When the subsystems are solved iteratively, we consider in addition the splitting of $\mathcal{M}^l$ defined by

$$\mathcal{M}^l = \mathcal{P}^l - \mathcal{Q}^l,\ l = 1, ..., m,$$

where $\mathcal{P}^l$ are diagonal matrices and in addition for $l = 1, ..., m$, $\mathcal{M}^l = \mathcal{P}^l - \mathcal{Q}^l$ is a weak regular splitting [16]. From an algorithmic point of view we can perform a fixed number of inner iterations or alternatively, under the previous appropriate assumptions, perform iterations until convergence. Using such new decomposition, for $l = 1, ..., m$, we can define an asynchronous two-stage method as follows

$$\mathcal{P}^l U^{l,r+1} + \Phi(U^{l,r+1}) + \omega(U^{l,r+1}) = \mathcal{Q}^l U^{l,r} + \mathcal{N}^l \tilde{U}^l + G,\ \omega(U^{l,r+1}) \in \partial(\Psi_{\mathcal{K}})(U^{l,r+1})$$

where $\tilde{U}^l = (U_1^{l,\rho_1(r)}, \ldots, U_\alpha^{l,\rho_\alpha(r)})$.

Then, for $l = 1, \ldots, m$, and for $i = 1, \ldots, \alpha$, such asynchronous two-stage multisplitting method is given by

$$\begin{cases} \mathcal{P}_i^l U_i^{l,r+1} + \Phi_i(U_i^{l,r+1}) + \omega_i(U_i^{l,r+1}) = (\mathcal{Q}^l U^{l,r})_i + (\mathcal{N}^l(\sum_{k=1}^{m} W_{lk} V^{k,\rho_k(r)}) + G)_i, \text{ if } i \in s(r) \\ U_i^{l,r+1} = U_i^{l,r} \text{ if } i \notin s(r) \end{cases}$$

where $\omega_i(U_i^{l,r+1}) \in \partial(\Psi_{\mathcal{K}})_i(U_i^{l,r+1})$.

Then for each subdomain Ω_I we have to solve successively an algebraic system. Moreover each subdomain Ω_I is divided into smallest overlapping subdomains, constituted by a vertical slicing represented in Fig. 2a, in such a way to solve the overall system defined in Ω_I by a Schwarz alternating method; in addition we solve each algebraic system in each small subdomain by a point Gauss - Seidel method. Then we obtain a parallel two stage method with synchronous or asynchronous communications between the processes by combining the Schwarz alternating method with the point relaxation method. As previously indicated we use on one hand a Newton method for solving the equation defined on the discretized points where occurs the radiation phenomenon and on the other hand we have to project each computed component in order to respect the positivity constraint.

So, due to the fact that each discretization matrices $\mathcal{A}$ are M-matrices and that assumptions (3) and (7) are satisfied, we can now recall a result (see [5] for the proof) ensuring that the fixed point mapping associated to the two-stage method is contractive, and conclude on the convergence of the sequential and parallel synchronous and asynchronous two-stage methods applied to the solution of the problems (23) to (25).

Proposition 2. *Consider the solution of the three constrained discretized algebraic systems (23) to (25); then since in each area Ω_I, $\mathcal{A}$ is an M-matrix and assumptions (7) and (3) due to the positivity constraint are well verified, the sequential and parallel synchronous and asynchronous two-stage method starting from every initial guess U_I^0 converge to the solution U_I^*, $\mathcal{I} = 1, 2, 3$, of these three problems.*

In the sequel the physical parameters are given by Table 4, mainly for $\mathcal{I} = 1, 2, 3$, the coefficients ρ_I, c_I, $h_{cv,\mathcal{I}}$ and λ_I the units being expressed respectively as $(kg.m^{-3})$ for the metal density, $(J.kg^{-1}.K^{-1})$ for the specific heat, and as $(W.m^{-1}.K^{-1})$ for the thermal conductivity; the parameters $u_{ext} = 40.(C)$, $u_{imp} = 100.(C)$, $\Phi_{imp} = 50.$, $R = 15.$, $\sigma = 5.67e-08\ W.m^{-2}.K^{-4}$, $\xi = .18$, u_{cst} being given in each area by: $u_{in,1} = 1500.(C)$, $u_{in,2} = 900.(C)$ and $u_{in,3} = 600.(C)$.

4.4. Cloud simulation

In a previous work [37] we have solved the problem of steel solidification in continuous casting on one hand on a cluster and on the other by using Grid'5000 facilities. In these previous simulation on grid, we have chosen 100 times steps and we limit to 5 times steps the parallel experiments on the cloud to save time as tests take a long time, particularly for synchronous testing. Across all experiments we have solved the problem of steel solidification by the Schwarz alternating method with overlapping of 3 meshes.

The aim of this paragraph is therefore to compare, as far as possible, the performance of the algorithms on these different architectures.

In [37] numerical sequential experiments have been also carried out on the machine parasilo located in Rennes for different size of problem, i.e. $n = 454$ and 622. Thus for these values of n, the subdomain partitioning is defined in Table 3. The performance of the sequential algorithms is summarized in Table 5. It should be noted that for all the experiments carried out in both sequential and parallel mode, Newton's method used for the nonlinear equations describing the radiation phenomenon converges extremely quickly in about 15 iterations.

Due to very strongly properties of diagonal dominance of the discretization matrices resulting from high values of the specific heat capacity c_I and also of the metal density ρ_I for $\mathcal{I} = 1, 2, 3$, the convergence of the iterative sequential method is very fast. Moreover on the cluster parasilo, using 8 machines, with the synchronous algorithms we obtained better results (presented in Tables 6 and 7) than those obtained with the asynchronous method; indeed in this case of fast convergence there is little synchronizations between processors, resulting in low latency, which clearly explains the high performance of synchronous methods compared to the asynchronous ones and presented in [37].

Table 3
Different sizes of domains for the problem to solve.

n	Number of points in Ω	Number of points in Ω_1
454	$454 \times 454 \times 454 = 93\,576\,664$	$152 \times 454 \times 454 = 31\,329\,632$
622	$622 \times 622 \times 622 = 240\,641\,848$	$208 \times 622 \times 622 = 80\,471\,872$

n	Number of points in Ω_2	Number of points in Ω_3
454	$151 \times 454 \times 454 = 31\,123\,516$	$150 \times 454 \times 454 = 30\,917\,400$
622	$207 \times 622 \times 622 = 80\,084\,988$	$206 \times 622 \times 622 = 79\,698\,104$

Table 4
Main physical parameters in each subdomain.

subdomain	Ω_1	Ω_2	Ω_3	unit
parameters	$c_1 = 688.$	$c_2 = 745.$	$c_3 = 485.$	$J.kg^{-1}.K^{-1}$
	$\rho_1 = 7340.$	$\rho_2 = 7659.$	$\rho_3 = 7834.$	$kg.m^{-3}$
	$h_{cv,1} = 25.$	$h_{cv,2} = 15.$	$h_{cv,1} = 5.$	$W.m^{-2}.K^{-1}$
λ_I constant	$\lambda_1 = 32.$	$\lambda_2 = 35.6$	$\lambda_3 = 51.$	$W.m^{-1}.K^{-1}$

Table 5
Elapsed time (sec) and relaxations with sequential algorithm and threshold $\epsilon = 10^{-3}$.

	Sequential results on 1 machine	
n	elapsed time	relaxations
454	843.23	51 044 387 879
622	2 342.18	152 385 850 898

Table 6
Elapsed time (sec), relaxations and communications on cluster with synchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-3}$.

	Synchronous results on 8 machines				
			communication times		
n	elapsed time	relaxations	compute	barrier	% comms
454	220.02	7 312 241 083	41.58	21.17	28.52
622	592.82	21 848 928 125	96.19	57.18	25.87

Table 7
Elapsed time (sec), relaxations and communications on cluster with asynchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-3}$.

	Asynchronous results on 8 machines				
			communication times		
n	elapsed time	relaxations	compute	barrier	% comms
454	569.00	25 861 444 388	1.28	40.81	7.40
622	1 517.71	53 424 870 465	1.99	100.97	6.78

Remark 7. For clarity let us specify how the compute time value is determined; in fact the compute time is equal to elapsed time multiplied by the percentage of communications minus the time consumed by the synchronization barriers at each time step.

On the other hand, as shown in [37], for parallel experiments on Grid'5000 grisou and parasilo machines respectively located in Nancy and Rennes, were used with always 8 machines. Thus, on Grid'5000 architecture, due to the heterogeneity of the machines and the great distance between them, as well as the latency of the network, taking into account the size of the problems and therefore the fineness of the discretization, asynchronous methods require about 4 times less computing time (see Tables 8 and 9).

These previously obtained performances will now be compared with those obtained on cloud architecture with the 8 virtual machines considered in Table 2; the results are presented in Tables 10 and 11. Thus in this case we obtain extremely interesting values of τ around 99 (respectively 114) when $n = 454$ (respectively $n = 622$). Note that these interesting values of τ derive from the mode of decomposition of Ω into small vertical overlapping subdomains, i.e. in slices represented in Fig. 2a, where each small subdomain performs communications with its left and right neighbors. In the synchronous case the number of iteration of the Schwarz

Table 8
Elapsed time (sec), relaxations and communications on grid with synchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-3}$.

	Synchronous results on 8 machines				
			communication times		
n	elapsed time	relaxations	compute	barrier	% comms
454	6 662.29	7 312 241 083	4 767.37	1 740.10	97.67
622	13 391.76	21 848 928 125	9 592.78	3 363.38	96.74

Table 9
Elapsed time (sec), relaxations, communications, and τ on grid with asynchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-3}$.

	Asynchronous results on 8 machines					
			communication times			
n	elapsed time	relaxations	compute	barrier	% comms	τ
454	1 713.20	35 711 264 519	149.15	751.52	52.57	3.88
622	3 985.08	88 959 003 614	207.54	1 048.40	31.51	3.36

Table 10
Elapsed time (sec), relaxations and communications on cloud with synchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-3}$.

	Synchronous results on 8 machines				
			communication times		
n	elapsed time	relaxations	compute	barrier	% comms
454	48 648	1 100 453 134	55	20 887	43.05
622	130 814	3 962 917 669	115	55 150	42.25

Table 11
Elapsed time (sec), relaxations, communications, and τ on cloud with asynchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-3}$.

	Asynchronous results on 8 machines					
			communication times			
n	elapsed time	relaxations	compute	barrier	% comms	τ
454	491	1 418 700 782	34	50	17.11	99.10
622	1 142	5 837 594 448	123	135	22.59	114.55

Table 12
Values of uniform norm of residue for parallel synchronous and asynchronous algorithms with overlapping and threshold $\epsilon = 10^{-3}$.

n	454	454	622	622
Parallel mode	Synchronous	Asynchronous	Synchronous	Asynchronous
Ω_1	$4.93\ 10^{-4}$	$2.33\ 10^{-4}$	$6.64\ 10^{-4}$	$4.06\ 10^{-4}$
Ω_2	$3.75\ 10^{-4}$	$7.93\ 10^{-8}$	$9.60\ 10^{-4}$	$1.44\ 10^{-4}$
Ω_3	$4.58\ 10^{-5}$	$3.09\ 10^{-4}$	$6.88\ 10^{-4}$	$6.03\ 10^{-4}$

method to reach convergence is equal to 276 when $n = 454$ and 389 when $n = 622$. We coarsely measured the number of iterations in asynchronous mode and obtained 355 and 574 iterations respectively for $n = 454$ and $n = 622$; note that the ratio of the number of relaxations in asynchronous mode to that in synchronous mode gives the same results, which makes it possible to estimate the number of macro-iterations in asynchronous mode to converge.

The values of the uniform norm of the residue used to stop the iterative process for both synchronous and asynchronous methods are shown below in Table 12. For $n = 454$ (respectively $n = 622$) this norm is of order of $5.0\ 10^{-4}$ (respectively $9.6\ 10^{-4}$) in synchronous mode and $3.0\ 10^{-4}$ (respectively $6.0\ 10^{-4}$) in asynchronous mode and may even be of order $8.0\ 10^{-08}$ in Ω_2.

Finally, since in cloud architecture we do not know really what resources have been allocated, their location and their hardware configuration, so it is a priori difficult to compare the performances obtained in such architecture with those previously obtained on cluster or on Grid'5000. Nevertheless the use of 8 machines in each case, and the comparison of τ values, allows us to observe the

improvement in restitution time of asynchronous parallel algorithms compared with those obtained in synchronous mode. Indeed when Grid'5000 architecture is used, τ is of order of four and reaches values on order of 100 on cloud architecture, that is 25 times more, which corresponds to an excellent improvement; the same observation can be made if we compare the τ values obtained on cluster.

5. Cloud simulation computation of problem with unilateral constraints

In what follows we will consider a distinct physical problem previously studied in [31] which, for an easy analysis of the behavior of parallel synchronous or asynchronous iterative algorithms, can be formulated like a multivalued problem.

5.1. The model problem

Consider a flow in a bounded domain $\Omega \subset \mathbb{R}^3$. Let us denote by $\partial\Omega$ the boundary of Ω and assume that $\partial\Omega = \Gamma_0 \bigcup \Gamma_1$; in the physical application Γ_0 is assumed to be permeable and Γ_1 semipermeable, in the following way: the fluid (assumed to be less compressible) is free to enter in Ω through Γ_1. If $u = u(\mathbf{x}, t)$ denotes the pressure of the fluid at the point $\mathbf{x} = (x_1, x_2, x_3)$ and at time t, then u is the solution of the following parabolic boundary value problem

$$\frac{\partial u(\mathbf{x},t)}{\partial t} - \Delta u(\mathbf{x},t) = f(\mathbf{x},t), \text{ everywhere in } \Omega \text{ and for } 0 < t \leq T,$$

where $f \in \mathcal{L}^2(\Omega \times [0,T]), T$ is the final time and Δ is the Laplacian operator defined in the three-dimensional space. The boundary condition associated to the previous parabolic equation can be formulated as follows [4]: if a known pressure $\varphi(\mathbf{x})$ is applied on the boundary $\partial\Omega = \Gamma_0 \bigcup \Gamma_1$, then we have:

$$u(\mathbf{x},t) = \varphi(\mathbf{x}), \forall \mathbf{x} \in \Gamma_0 \text{ and } \forall t > 0;$$

on Γ_1 we have two distinct cases to consider:

- first case: if

  $$\varphi(\mathbf{x}) < u(\mathbf{x},t), \forall \mathbf{x} \in \Gamma_1 \text{ and } \forall t > 0,$$

  the fluid is inclined to come out of Ω, which is impossible; so the output is zero and then

  $$\frac{\partial u(\mathbf{x},t)}{\partial n} = 0, \forall \mathbf{x} \in \Gamma_1 \text{ and } \forall t > 0,$$

  where $\frac{\partial}{\partial n}$ is the outward normal derivative with respect to Γ_1; thus the normal derivative is equal to zero if the constraint is not active;
- second case: if

  $$\varphi(\mathbf{x}) \geq u(\mathbf{x},t), \forall \mathbf{x} \in \Gamma_1 \text{ and } \forall t > 0,$$

  the fluid is inclined to come in Ω, which is possible since it is free to enter in Ω; then we have a continuous flow, and consequently there is no jump of pressure. Thus, we have

  $$\varphi(\mathbf{x}) = u(\mathbf{x},t) \text{ and } \frac{\partial u(\mathbf{x},t)}{\partial n} \geq 0, \forall \mathbf{x} \in \Gamma_1 \text{ and } \forall t > 0.$$

Let us denote by $u(\mathbf{x},0) = u_0(\mathbf{x})$ the initial pressure. Then, we can summarize the problem of hydrodynamic in semipermeable environment as follows

$$\begin{cases} \frac{\partial u(\mathbf{x},t)}{\partial t} - \Delta u(\mathbf{x},t) = f(\mathbf{x},t), \text{ everywhere in } \Omega \text{ and for } 0 < t \leq T, \\ u(\mathbf{x},t) \geq \varphi(\mathbf{x}), \frac{\partial u(\mathbf{x},t)}{\partial n} \geq 0, (u(\mathbf{x},t) - \varphi(\mathbf{x})) \frac{\partial u(\mathbf{x},t)}{\partial n} = 0, \forall \mathbf{x} \in \Gamma_1, \ \forall t > 0, \\ u(\mathbf{x},t) = \varphi(\mathbf{x}), \forall \mathbf{x} \in \Gamma_0, \ \forall t > 0, \\ u(\mathbf{x},0) = u_0(\mathbf{x}), \forall \mathbf{x} \in \Omega. \end{cases} \tag{27}$$

Problem (27) is currently solved by an implicit time marching scheme. Consider now the associated stationary problem and let us denote by $v = v(\mathbf{x})$ the solution of this problem; then we have to solve the following problem

$$\begin{cases} -\Delta v(\mathbf{x}) + \sigma v(\mathbf{x}) = g(\mathbf{x}), \text{ everywhere in } \Omega, \sigma > 0, \\ v(\mathbf{x}) \geq \varphi(\mathbf{x}), \frac{\partial v(\mathbf{x})}{\partial n} \geq 0, (v(\mathbf{x}) - \varphi(\mathbf{x})) \frac{\partial v(\mathbf{x})}{\partial n} = 0, \forall \mathbf{x} \in \Gamma_1, \\ v(\mathbf{x}) = \varphi(\mathbf{x}), \forall \mathbf{x} \in \Gamma_0, \end{cases} \tag{28}$$

where $g \in \mathcal{L}^2(\Omega)$ is derived from the implicit time marching scheme and from the discretization of the right-hand side $f(\mathbf{x},t)$ of (27) and σ is a positive parameter, in fact the inverse of the time step.

Let us define the following convex set

$$K = \{v | v \in \mathcal{H}^1(\Omega), \text{ such that } v(\mathbf{x}) \geq \varphi(\mathbf{x}) \text{ on } \Gamma_1,\ v(\mathbf{x}) = \varphi(\mathbf{x}) \text{ on } \Gamma_0\},$$

where $\mathcal{H}^1(\Omega)$ is the classical Hilbert Sobolev space and let

$$a(u,v) = \int_\Omega (\nabla v \nabla u + \sigma\, uv) d\mathbf{x} \text{ and } L(v) = < g, v > = \int_\Omega g v d\mathbf{x},$$

where $< .,. >$ denotes the inner product defined in $\mathcal{L}^2(\Omega)$. Then, according to [4] v is the solution of the following stationary variational inequality.

$$\begin{cases} \text{Find } u \in K \text{ such that} \\ a(u, v-u) \geq L(v-u), \forall v \in K. \end{cases} \tag{29}$$

Note that since K is a closed convex set of the Hilbert space $\mathcal{H}^1(\Omega)$, $a(u,v)$ is obviously a symmetric bilinear continuous and elliptic form and $L(v)$ is a linear continuous form, then by applying the Stampacchia's theorem [3], the variational inequality (29) has a unique solution.

5.2. Multivalued formulation of the problem

In the sequel, we have also to consider the indicator function ψ of the convex subset K and the subdifferential mapping of ψ. Since $a(u,v)$ is a bilinear continuous form, and $L(v)$ is a linear continuous form, then we can apply the Riesz representative theorem. So, there exists one and only one element denoted $\bar{A}u \in E'$ (E' being identified with the Hilbert space $E = \mathcal{H}^1(\Omega)$), where $\bar{A}$ is a continuous linear mapping from $E \to E'$, such that

$$a(u,v) = < \bar{A}u, v >_{E \times E'}, \forall v \in E = \mathcal{H}^1(\Omega);$$

similarly, there exists also one and only one element, denoted $\bar{g} \in E'$ such that

$$L(v) = < \bar{g}, v >_{E \times E'}, \forall v \in E = \mathcal{H}^1(\Omega);$$

then the stationary variational inequality (29) can be written as follows

$$\begin{cases} \text{Find } u \in K_{ad} \text{ such that} \\ < \bar{A}u - \bar{g}, v - u >_{E \times E'} \ \geq 0, \forall v \in K. \end{cases} \tag{30}$$

In [4] the multivalued formulation of the problem with unilateral constraints is formulated and clarified; it is proved that (30) is equivalent to the multivalued problem

$$\begin{cases} \text{Find } u \in K \text{ such that} \\ \bar{A}u - \bar{g} + \partial\psi(u) \ni 0, \end{cases} \tag{31}$$

then, the model problem (28) can be written as follows

$$\begin{cases} -\Delta u(\mathbf{x}) + \sigma u(\mathbf{x}) = \bar{g}(\mathbf{x}), \text{everywhere in } \Omega, \\ u(\mathbf{x}) = \varphi(\mathbf{x}), \forall \mathbf{x} \in \Gamma_0 \\ \frac{\partial u(\mathbf{x})}{\partial n} + \partial\psi(u(\mathbf{x})) \ni 0, \forall \mathbf{x} \in \Gamma_1. \end{cases} \tag{32}$$

So, in the sequel we will mainly use this multivalued formulation (32) of the model problem in order to analyze the behavior of the parallel iterative synchronous and asynchronous algorithms used for the solution of this problem. Note that this formulation includes both the inhomogeneous Dirichlet's condition in a part of the boundary and the inequality constraint on the other part.

5.3. Discretization

In the sequel, we will consider that $\Omega = [0,1]^3 \subset \mathbb{R}^3$ and that Γ_1 is the square $[0,1]^2$ constituting by the lower face of the cube, the rest defining Γ_0. We consider also that Ω is discretized with an uniform mesh size h. So the complete discretization of the stationary boundary value problem (28) leads, at each time step, to the solution of a large multivalued nonlinear algebraic system of the following form

$$(A + \sigma I)U - G + \partial\Psi(U) \ni 0, \tag{33}$$

where G is derived from the discretization of the right-hand side of (28), A is the spatial block discretization matrix obtained by finite difference method by using the classical seven points scheme, I is the identity matrix, $\partial\Psi(U)$ results from the discretization of the subdifferential of the indicator function $\psi(U)$. Thus, classically A is an M-matrix (see [16]) and consequently the matrix $(A + \sigma I)$ is also an M-matrix. Thus we are again in the general framework presented in subsection 2.1 to 2.4 and we can conclude on the convergence of parallel synchronous or asynchronous Schwarz methods. Note that the block decomposition of the problem is

Table 13
Elapsed time and average number of relaxations for parallel Schwarz alternating method with 400^3 on 2 sites of Grid'5000 and threshold $\epsilon = 10^{-6}$.

	Asynchronous				Synchronous		
Proc	Time/s	Rel. Min	Rel. Max	Average	Time/s	Relaxations	τ
4	3075	1422	1785	1587	4603	1112	1.50
8	1413	1320	1694	1506	2235	1112	1.58
16	740	1325	1806	1549	1084	1112	1.46
32	405	1385	2006	1613	511	1112	1.26
64	185	916	1837	1441	522	1112	2.82

Table 14
Elapsed time and average number of relaxations for parallel Schwarz alternating method with 320^3 on 3 sites of Grid'5000 and threshold $\epsilon = 10^{-6}$.

	Asynchronous				Synchronous		
Proc	Time/s	Rel. Min	Rel. Max	Average	Time/s	Relaxations	τ
4	1432	1234	1321	1355	2449	1015	1.71
8	648	1144	1476	1267	1257	1015	1.94
16	339	1165	1580	1354	616	1015	1.57
32	194	1144	1878	1458	548	1015	2.82
64	87	841	1661	1256	456	1015	5.24
128	37	517	2037	1251	248	1015	6.70

derived from a decomposition of the domain Ω into small parallelograms around each subdomain as shown in Fig. 2b; so the iterate vector is decomposed into α blocks. In parallel computation, such decomposition requires communication with the six neighboring subdomains.

5.4. Cloud simulation

In [31] we have also previously solved problem (33) on one hand sequentially and on other hand by using Grid'5000 facilities. The overlap between the subdomains is two meshes and the tolerance for the stopping test is fixed at $\epsilon = 10^{-6}$.

For a mesh with $400^3 = 64\,000\,000$ (respectively $320^3 = 32\,768\,000$) grid points, the sequential Schwarz alternating method is executed with an elapsed time of order 9324 (respectively 4475) seconds in Rennes and 7104 (respectively 3416) seconds on the cluster suno located in Nice. The number of relaxations is equal to 1112 (respectively 1015) for the problem with 400^3 (respectively 320^3) grid points.

In [31] for the parallel experiments on Grid'5000 platform two (respectively three) distant sites have been used for our parallel simulations; for further details we refer to the Tables 13 and 14. When two sites are used the size of the algebraic systems was about equal to 400^3 while with three sites the domain Ω is discretized with 320^3 grid points. For each benchmark between 4 and 64 processors have been used (and in addition on three distant sites 128 processors). In each, case the elapsed time obtained when respectively synchronous or asynchronous communications between the processors are taken into account is respectively noted in Tables 13 and 14. For a comparison as valid as possible with the performances obtained with 8 machines on cloud architecture we focus on the results obtained with the same number of machines on Grid'5000. Thus with eight processors on two sites (respectively three) the elapsed time is equal to 1413 (respectively 648) seconds for asynchronous communications mode and when synchronous communications are implemented the elapsed time is equal to 2235 (respectively 1257) seconds; thus the factor τ is of order 1.58 (respectively of order 1.94). Note that the number of relaxations is respectively estimated by 1506 (respectively 1267) in asynchronous mode and is equal to 1112 (respectively 1015) in synchronous mode.

Nevertheless, once again a comparison with elapsed time obtained on cloud architecture is tricky to perform since grid and cloud architecture do not use the same machines and have not similar architecture. This is why, in order to situate performances on each architecture using 8 machines only, we compare the values of τ obtained on Grid'5000 with the values of τ obtained on cloud architecture. Thus on Grid'5000, on two (respectively three) sites the values of τ vary between 1.50 (respectively 1.71) and 2.82 (respectively 6.70). Consequently, we can notice that in these parallel experiments on Grid'5000 we obtain in each case modest τ values. However on three sites, large τ values were obtained when 64 or 128 processors were used, corresponding to significant resources deployed.

For parallel experiments on cloud architecture, we use the 8 virtual machines (see Table 2) and we consider three meshes constituted by 320^3, 400^3 and in addition $480^3 = 110\,592\,000$ grid points. The results of parallel experiments on cloud architecture are given in Tables 15 and 16.

For this application, the Table 17 shows the values of the uniform norm of residue for parallel synchronous and asynchronous Schwarz's algorithms with a threshold equal to $\epsilon = 10^{-6}$.

Table 15
Elapsed time (sec), relaxations and communications on cloud with synchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-6}$.

	Synchronous results on 8 machines				
			communication times		
n	elapsed time	relaxations	compute	barrier	% comms
320	22 889	13 712	327	1 450	92
400	45 697	17 664	823	1 466	95
480	79 084	21 328	1 809	5 342	91

Table 16
Elapsed time (sec), relaxations, communications, and τ on cloud with asynchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-6}$.

	Asynchronous results on 8 machines					
			communication times			
n	elapsed time	relaxations	compute	barrier	% comms	τ
320	1 037	46 711	802	96	13	22.07
400	1 876	48 300	1 697	94	5	24.36
480	2 557	78 929	2 279	70	8	30.93

Table 17
Values of uniform norm of residue for parallel synchronous and asynchronous algorithms with overlapping and threshold $\epsilon = 10^{-6}$.

n	Synchronous	Asynchronous
320	$1.02\ 10^{-6}$	$9.95\ 10^{-6}$
400	$1.01\ 10^{-6}$	$9.97\ 10^{-7}$
480	$8.61\ 10^{-7}$	$9.98\ 10^{-7}$

In the synchronous case the number of iterations of the Schwarz method to reach convergence is equal to 857 when $n = 320$, 1 104 when $n = 400$ and 1 333 when $n = 480$. A coarse estimate of the number of iterations in asynchronous mode gives 1 719, 2 734 and 3 340 iterations respectively for $n = 320$, $n = 400$ and $n = 480$.

For this problem with unilateral constraints we obtain values of τ smaller than those obtained for the resolution of the steel solidification problem. This decrease in number values of τ is due to the increase in the number of communications as a result of cutting the domain Ω into small parallelograms around each subdomain as shown in Fig. 2b. In a previous study (see [1]), for the resolution of diffusion and convection - diffusion problems, we have obtained values of τ of the same order between twenty and thirty depending on the size of the discretized linear system. However, it should be noted that compared to the values of τ obtained on Grid'5000, the implementation on cloud architecture clearly improves again strongly these values of τ.

Remark 8. In agreement with the results given in Table 16 we can notice that the values of τ increase with the number of grid points. For example when we consider 480^3 grid points the elapsed time with synchronous communications is equal to 79 084 seconds while in asynchronous mode it is equal to 2 557 seconds. The percentage of communications is equal to 91 in synchronous mode (corresponding to an elapsed time of 79 084 seconds and 21 328 relaxations) while with asynchronous communications it is equal to 8 (corresponding to an elapsed time of 2 257 seconds and 78 929 relaxations); thus in this case $\tau = 30.93$.

Remark 9. With synchronous communications only when 8 processors are used the comparison between the elapsed times obtained with Grid'5000 are better than the ones obtained by using the cloud architecture; indeed, for the two first values of n, in the first case 2 235 and 1 257 seconds are necessary to reach convergence while in the second case the corresponding elapsed time are equal to 22 899 and 45 697. On the contrary this fact is disappearing when the communications are asynchronous; indeed for Grid'5000 use the elapsed time is respectively equal to 1 413 and 648 while it is equal respectively to 1 037 and 1 876 on cloud architecture; thus, due to the fact that the duration of the communications in this last case is minimal, on the two computational systems, comparable elapsed times are obtained. This is another advantage of asynchronous communications in high performance computing.

6. Cloud simulation computation for American options pricing

The last application concerns a classical financial problem modeled also by an evolution variational inequality.

6.1. The model obstacle problem

We consider here the case of American options modeled by the Black-Scholes equations [38]. This classical Black-Scholes equation is a boundary value problem describing the evolution of call or put options in the field of mathematics of financial contracts which may be exercised at any time prior to expiry, i.e. when the current time θ takes any value between 0 and T, where T denotes the expiry date. Classically, such option is modeled by the following retrograde time dependent nonlinear diffusion equation:

$$\begin{cases} \frac{\partial u}{\partial \theta} + \frac{\sigma^2}{2}\Delta u - ru \geq 0,\ u \geq \phi, e.w.\ in\ [0,T]\times\mathbb{R}^d \\ (\frac{\partial u}{\partial \theta} + \frac{\sigma^2}{2}\Delta u - ru)(u-\phi) = 0, e.w.\ in\ [0,T]\times\mathbb{R}^d \\ u(T,S) = \phi \end{cases} \tag{34}$$

where, e.w. means every where, $\phi = \phi(S) = \max(S-H,0)$ in the case of call option or $\phi = \max(H-S,0)$ in the case of put option; in the previous equations u denotes the value of the considered option, i.e. a call or a put option; $u = u(\theta, S)$ is a function of the current value of the underlying asset S and of the time θ. Note also that the considered option also depends on the following parameters:
- r the interest rate,
- σ the volatility of the underlying asset, σ being in fact the instantaneous standard deviation of the price with respect to the exercise price H, classically called strike and fixed beforehand; σ characterizes the uncertainty of the option's behavior.

Note that the previous boundary value problem is not defined on a bounded domain, but is defined on the unbounded domain $\mathbb{R}^d, d \geq 1$; in the sequel we consider that $d = 3$. This difficulty is solved by considering the problem defined on a bounded domain $\Omega \subset \mathbb{R}^d$, and it can be proven that the solution of the retrograde time dependent diffusion equation defined on the bounded domain Ω converges to the solution of problem (34) when the measure of Ω tends to infinity (see [39]).

Another particularity of the problem to solve, is that the value of the option is not known at the initial time $\theta = 0$; only the value $u(T,S)$ is known. In fact the problem consists in computing $u(0,S)$.

These previous two issues can be resolved, firstly by considering problem (34) defined in a bounded large domain Ω and secondly by a change of variables concerning the time, which consists in replacing the variable θ by a variable $t = T - \theta$. Thus, problem (34) is replaced by a classical time dependent diffusion problem modeled as follows:

$$\begin{cases} \frac{\partial u}{\partial t} - \frac{\sigma^2}{2}\Delta u + ru \geq 0,\ u \geq \phi, e.w.\ in\ [0,T]\times\Omega \\ (\frac{\partial u}{\partial t} - \frac{\sigma^2}{2}\Delta u + ru)(u-\phi) = 0, e.w.\ in\ [0,T]\times\Omega \\ u(0,S) = \phi \\ B.C.\ on\ u(t,S)\ defined\ on\ \partial\Omega \end{cases} \tag{35}$$

where B.C. describes the boundary conditions on the boundary $\partial\Omega$ of the domain Ω. Practically, the Dirichlet condition or the Neumann condition is classically considered.

The previous time dependent problem is solved numerically by considering an implicit or semi-implicit time marching scheme, where at each time step the following stationary nonlinear problem is solved.

$$\begin{cases} -\frac{\sigma^2}{2}\Delta u + (r+\delta)u - g \geq 0,\ u \geq \phi, e.w.\ in\ \Omega \\ (-\frac{\sigma^2}{2}\Delta u + (r+\delta)u - g)(u-\phi) = 0, e.w.\ in\ \Omega \\ u(0,S) = \phi \\ B.C.\ on\ u(t,S)\ defined\ on\ \partial\Omega \end{cases}$$

where δ is the inverse of the time step and $g = \delta u^{prec}$, where u^{prec} is the solution obtained at the previous time step.

Consequently at each time step we have to solve a variational inequation which, as in subsection 5.2, can be expressed as a multivalued stationary problem. Thus when the discretization scheme of the diffusion term is performed by the classical seven points finite difference scheme and due to the monotony of the subdifferential mapping the parallel synchronous or asynchronous iterative Schwarz alternating method converges to the problem solution. To implement Schwarz's alternating method, we consider a decomposition of the domain Ω into small parallelograms around each subdomain as shown in Fig. 2b.

6.2. Cloud simulation

For this financial problem we refer to [23] for the results obtained on sequential and parallel mode on Grid'5000 using two or three sites when the problem is solved by the Schwarz alternating method. For $320^3 = 32\ 768\ 000$ grid points, the main results on sequential mode lead to elapsed times of order 7 494 (respectively 8 748 and 11 780) seconds in Sophia (respectively Nancy and cluster pastel in Toulouse); in each site the number of relaxations is equal to 2105. For $400^3 = 64\ 000\ 000$ grid points, the sequential elapsed time is of order 16 099 (respectively 21 214) seconds in Sophia (respectively 21 214 in Rennes), while in each case the number of relaxations is equal to 2380.

Concerning the performances obtained on Grid'5000 the results are summarized in Tables 18 and 19 respectively when Ω is discretized with 64 000 000 and 32 768 000 grid points and the threshold is equal to $\epsilon = 10^{-6}$.

Table 18
Elapsed time (seconds), number of relaxations (average) and communication time with Schwarz alternating method for the multivalued problem on cluster on two sites of Grid'5000 (Sophia and Rennes) for 64 000 000 mesh points.

P	Asynchronous	Relax	Comms	Synchronous	Relax	Comms	τ
4	8409	4014	521 (6.2%)	11300	2380	5255 (46%)	1.34
8	3977	3729	249 (6.3%)	6033	2380	2411 (40%)	1.51
16	1709	3269	68 (3.9%)	2625	2380	830 (32%)	1.53
32	934	3355	60 (6.4%)	1599	2380	397 (25%)	1.67
64	428	2876	37 (8.6%)	1120	2380	261 (13%)	2.61

Table 19
Elapsed time (seconds), number of relaxations (average) and communication time with Schwarz alternating method for the multivalued problem on three sites of Grid'5000 (Sophia, Nancy and Toulouse) for 32 768 000 mesh points.

P	Asynchronous	Relax	Comms	Synchronous	Relax	Comms	τ
4	3198	2885	46 (1.4%)	5257	2105	2284 (43%)	1.64
8	1822	2971	78 (4.0%)	2494	2105	873 (34%)	1.37
16	773	2176	25 (3.2%)	1156	2105	278 (26%)	1.50
32	442	2059	33 (7.4%)	958	2105	239 (24%)	2.17
64	202	1139	22 (11.0%)	815	2105	189 (23%)	4.03
128	57	683	8 (14.0%)	549	2105	133 (24%)	9.63

Table 20
Elapsed time (sec), relaxations and communications on cloud with synchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-6}$.

	Synchronous results on 8 machines				
			communication times		
n	elapsed time	relaxations	compute	barrier	% comms
320	56 366	34 016	859	11 046	79
400	98 863	38 448	1 778	15 179	83
480	155 970	42 384	3 759	16 714	87

Table 21
Elapsed time (sec), relaxations, communications, and τ on cloud with asynchronous parallel algorithms with overlapping and threshold $\epsilon = 10^{-6}$.

	Asynchronous results on 8 machines					
			communication times			
n	elapsed time	relaxations	compute	barrier	% comms	τ
320	3 031	65 760	2 839	53	5	19
400	3 973	74 518	3 841	47	2	25
480	4 703	88 992	4 006	60	14	33

On two sites with 64 000 000 grid points 4, 8, 16, 32 and 64 processors are used while on three sites with 32 768 000 grid points 4, 8, 16, 32, 64 processors are again taken into account and in addition of this previous case 128 processors are also used. Similarly to what was presented in section 5.2, for both number of grid points, due to the importance of communications in synchronous mode, elapsed time obtained in asynchronous communication mode is less than the ones obtained in synchronous communication mode. Once again, the number of relaxation to reach convergence in asynchronous mode is greater than the one obtained in synchronous mode; but in the first case, the numerical quality of the solution is better. Thus as previously considered, for comparison with the performances on cloud architecture the value of the factor τ is the most significant to consider and on Grid'5000 this value varies between 1.34 and 2.61 (respectively 1.64 and 9.63) on two (respectively three) sites.

In the sequel, for a comparison with performances obtained on cloud architecture, we focus once again on the case where 8 machines are used. Indeed, for parallel experiments on cloud architecture, 8 virtual machines are used (see Table 2). We limit these parallel experiment comparisons to 320^3, 400^3 and 480^3 grid points. The results of parallel experiments on cloud architecture are given in Tables 20 and 21.

For this financial problem, we obtain a behavior similar to the previous one (see section 5) and it should be pointed out that, if we compare the results of the τ values obtained from the experiments with the Grid'5000 environment, implementation on cloud architecture considerably improves these τ values. Moreover, in the synchronous case the number of iterations to reach convergence

Table 22
Values of uniform norm of residue for parallel synchronous and asynchronous algorithms with overlapping and threshold $\epsilon = 10^{-6}$.

n	Synchronous	Asynchronous
320	$8.48\ 10^{-7}$	$1.03\ 10^{-6}$
400	$8.38\ 10^{-7}$	$1.01\ 10^{-6}$
480	$8.61\ 10^{-7}$	$9.97\ 10^{-7}$

is equal to 4 250 when $n = 320$, 4 804 when $n = 400$ and 5 296 when $n = 480$. A coarse estimate of the number of iterations in asynchronous mode gives 6 598, 6 746 and 6 812 iterations respectively for $n = 320$, $n = 400$ and $n = 480$.

Similarly, as with the previous problem, we can notice that the values of τ increase with the number of grid points and for 480^3 we have an interesting value of $\tau = 33$.

For this last application, the Table 22 indicates the values of the uniform norm of the residue for parallel synchronous and asynchronous algorithms with overlapping when the threshold equals $\epsilon = 10^{-6}$.

7. Synthesis of parallel cloud experiments

Despite relatively modest algebraic system sizes, chosen due to operating constraints on IN2P3 center machines, the results obtained confirm our expectations, i.e. that the asynchronous parallel algorithms are more efficient compared to the synchronous ones when using cloud resources without knowledge of their positioning, shared usage, network latency and hardware architecture. Moreover, the improvement in results is very considerable compared to previous tests carried out on clusters and grids.

We can note that, in asynchronous parallel experiments, the number of relaxations necessary to reach convergence is greater than the one necessary when parallel synchronous schemes are used. In fact, this is a well-known drawback of asynchronous iterative schemes, due to the use of possibly delayed components. It turns out that the overhead generated by additional relaxations in the case of asynchronous algorithms is smaller than the synchronization overhead combined with processor idle time of synchronous parallel schemes of computation. In the same way the speed of convergence of parallel asynchronous method is generally slower compared to the synchronous one. Nevertheless, the speed of convergence depends on the frequency of updates of calculations and then on the decomposition of the initial problem in subproblems. But generally there is a reduction of the elapsed time to reach convergence. However the increase of the number of relaxations improves the accuracy of the computed solution and the numerical quality of the final obtained solution.

Furthermore, despite the higher number of relaxations, the elapsed time of the asynchronous scheme is less than the one obtained when synchronous scheme is used.

As a consequence, for solving very large scale algebraic systems, the parallel experiments show that on cloud architecture asynchronous parallel methods present real interest when parallel processing requires a large number of synchronizations. This occurs in particular when the convergence is slow, and especially when the communications between the machines are slow particularly when the connections between these machines are not by direct lines. Indeed, communications on a platform are dependent on the latency of the network. This slowdown is accentuated by the distance between machines that are not connected by direct lines. This phenomenon of superiority of the asynchronous parallel methods is thus accentuated when the processors are heterogeneous and distant. Thus, correlatively, the performance in terms of restitution time of asynchronous parallel methods is much higher than that of synchronous methods. We can thus see here the influence of the architecture of the machines on the performances of the algorithms.

Furthermore, while distance is a primary factor, it is not the sole element contributing to this latency that can occur with a server configuration in a data center environment. For instance, each of the software layers presented earlier (see subsection 3.1) offers different levels of abstraction and control to users, but they are all subject to the issue of this network latency, especially those involving frequent data exchanges between the cloud layers. The use of asynchronous parallel algorithms, therefore, has an advantage in a cloud environment compared to the synchronous equivalent, as idle times are eliminated. Thus, asynchrony is an efficient means of managing communication overhead and load imbalances.

In addition considering the additional software layers, the use of asynchronous parallel algorithms is of interest in a cloud environment compared to the synchronous equivalent insofar as inactivity times are eliminated. Thus asynchronism is an efficient way to deal with communication overhead and load unbalance.

Finally, for the implementation of the Schwartz alternating method, note that the decomposition of the domain Ω by slices with two neighbors compared with the decomposition into small parallelograms around each subdomain with six neighbors, allows to highlight the performances of parallel asynchronous algorithms compared to their synchronous counterparts.

8. Conclusion

For the numerical resolution of very large scale algebraic systems the study presented in the present paper shows clearly that parallel asynchronous iterative algorithms are very efficient compared to their synchronous counterparts when the convergence is slow.

In addition, in the same context, the use of asynchronous methods could have a positive impact on the environment and also lead to savings in electrical energy.

In future works we will consider also comparison of synchronous and asynchronous iterative parallel methods for the resolution of other kinds of nonlinear boundary value problems. Moreover the study presented in this paper was carried out with a limited number of 8 machines, which enabled us to solve discretized problems with relatively modest sizes. In future work, we hope to be able to access a larger number of heterogeneous machines far removed from each other.

Acknowledgement

This study has been made possible with the support of France Grilles cloud thanks to whom we have had cloud resources LPNHE which is a UMR Sorbonne University / CNRS-IN2P3. In addition, the previous study was made possible by the use of Grid'5000 facilities.

References

[1] M.A. Rahhali, T. Garcia, P. Spiteri, Performances analysis of parallel asynchronous and synchronous algorithms on cloud architecture for PDE, Adv. Eng. Softw. 186 (2023), https://doi.org/10.1016/j.advengsoft.2023.103550.
[2] V. Barbu, Nonlinear Semigroups and Differential Equations in Banach Spaces, Noordhoff International Publishing, Leyden, the Netherlands, 1976.
[3] R. Glowinski, J.L. Lions, R. Tremolières, Analyse numérique des inéquations variationnelles, Dunod, Paris, 1976, tome 1 and 2.
[4] G. Duvaut, J.L. Lions, Les inéquations en mécanique et physique, Dunod, Paris, 1972.
[5] P. Spiteri, Parallel asynchronous algorithms: a survey, Adv. Eng. Softw. 149 (2020), https://doi.org/10.1016/j.advengsoft.2020.102896, Elsevier.
[6] D. Chazan, W. Miranker, Chaotic relaxation, Linear Algebra Appl. 2 (1969) 199–222.
[7] J.-C. Miellou, Algorithmes de relaxation chaotique à retards, RAIRO. Anal. Numér. 1 (1975) 55–82.
[8] D. Bertsekas, J. Tsitsiklis, Parallel and Distributed Computation: Numerical Methods, Prentice Hall, Englewood Cliffs, New Jersey, 1989.
[9] G. Baudet, Asynchronous iterative methods for multiprocessors, J. ACM 25 (1978) 226–244.
[10] A. Frommer, D. Szyld, On asynchronous iterations, J. Comput. Appl. Math. 123 (2000) 201–216.
[11] Y. Saad, Iterative methods for linear systems of equations: a brief historical journey, in: Susanne C. Brenner, Igor E. Shparlinski, Chi-Wang Shu, Daniel Szyld (Eds.), 75 Years of Mathematics of Computation, in: Contemporary Mathematics, vol. 754, 2020, pp. 197–214.
[12] C. Brezinski, G. Meurant, M. Redivo-Zaglia, A Journey Through the History of Numerical Linear Algebra, SIAM, Philadelphia, 2022.
[13] J.M. Bahi, S. Contassot-Vivier, R. Couturier, Parallel Iterative Algorithms: from Sequential to Grid Computing, Chapman & Hall/CRC, 2007.
[14] L. Giraud, P. Spiteri, Résolution parallèle de problèmes aux limites non linéaires, ESAIM: M2AN 25 (1991) 579–606.
[15] R.S. Varga, Matrix Iterative Analysis, Prentice - Hall, l, Englewood Cliffs, New Jersey, 1962.
[16] J. Ortega, W. Rheinboldt, Iterative Solution of Nonlinear Equations in Several Variables, Academic Press, New York, 1970.
[17] V. Partimbene, T. Garcia, P. Spiteri, P. Marthon, L. Ratsifandrihana, Asynchronous multisplitting method for linear and pseudolinear problems, Adv. Eng. Softw. 133 (2019) 76–95.
[18] P. Spiteri, D. El Baz, J.C. Miellou, Perturbation of parallel asynchronous linear iterations by floating point errors, Electron. Trans. Numer. Anal. 13 (2002) 38–55.
[19] M. El Tarazi, Some convergence results for asynchronous algorithms, Numer. Math. 39 (1984) 325–340.
[20] K.H. Hoffmann, J. Zou, Parallel efficiency of domain decomposition methods, Parallel Comput. 19 (1993) 1375–1391.
[21] J.C. Miellou, D. El Baz, P. Spiteri, A new class of asynchronous iterative algorithms with order interval, Math. Comput. 67 (221) (1998) 237–255.
[22] D.J. Evans, W. Deren, An asynchronous parallel algorithm for solving a class of nonlinear simultaneous equations, Parallel Comput. 17 (1991) 165–180.
[23] M. Chau, T. Garcia, P. Spiteri, Asynchronous Schwarz methods applied to constrained mechanical structures in grid environment, Adv. Eng. Softw. 74 (2014) 1–15.
[24] F. Magoules, G. Gbikpi-Benissan, Distributed convergence detection based on global residual error under asynchronous iterations, IEEE Trans. Parallel Distrib. Syst. 29 (4) (2018) 819–829.
[25] F. Magoules, G. Gbikpi-Benissan, JACK2: an MPI-based communication library with non-blocking synchronization for asynchronous iterations, Adv. Eng. Softw. 11 (2018) 116–133.
[26] G. Gbikpi-Benissan, F. Magoules, Protocol-free asynchronous iterations termination, Adv. Eng. Softw. 146 (2020), https://doi.org/10.1016/j.advengsoft.2020.102827.
[27] E. Chow, A. Frommer, D.B. Szyld, Asynchronous Richardson iterations: theory and practice, Numer. Algorithms 87 (2021) 1635–1651.
[28] K.M. Chandy, L. Lamport, Distributed snapshots: determining global states of distributed systems, ACM Trans. Comput. Syst. 3 (1) (1985) 63–75.
[29] E.W. Dijkstra, C.S. Scholten, Termination detection for diffusing computation, Inf. Process. Lett. 11 (1980) 1–4.
[30] S.A. Savari, D.P. Bertsekas, Finite termination of asynchronous iterative algorithms, Parallel Comput. 22 (1) (1996) 39–56.
[31] M. Chau, A. Laouar, T. Garcia, P. Spitéri, Grid solution of problem with unilateral constraints, Numer. Algorithms 75 (4) (2017) 879–908.
[32] M. Chau, D. El Baz, R. Guivarch, P. Spitéri, MPI implementation of parallel subdomain methods for linear and nonlinear convection-diffusion problems, J. Parallel Distrib. Comput. 67 (5) (2007) 581–591.
[33] J.C. Miellou, P. Spitéri, D. El Baz, A new stopping criterion for linear perturbed asynchronous iterations, J. Comput. Appl. Math. 219 (2) (2008) 471–483.
[34] P. Spitéri, Finite precision computation for linear fixed point methods of parallel asynchronous iterations, in: B.H.V. Topping, P. Ivanyi (Eds.), Techniques for Parallel, Distributed and Cloud Computing in Engineering, Saxe-Coburg Publications, 2015, pp. 163–196.
[35] T. Garcia, P. Spiteri, L. Ziane-Khodja, R. Couturier, Solution of univalued and multivalued pseudo-linear problems using parallel asynchronous multisplitting methods combined with Krylov methods, Adv. Eng. Softw. 153 (2020), https://doi.org/10.1016/j.advengsoft.2020.102929.
[36] F. Magoules, J. Pan, F. Teng, Cloud Computing Data-Intensive Computing and Scheduling, Numerical Analysis and Scientific Computing, Chapman and Hall/CRC, New York, 2013.
[37] T. Garcia, P. Spitéri, G. Khenniche, Behavior of parallel two-stage method for the simulation of steel solidification in continuous casting, Adv. Eng. Softw. (131) (2008) 116–142, https://doi.org/10.1016/j.advengsoft.2018.11.12.
[38] P. Wilmott, J. Dewyne, S. Howison, Option Pricing-Mathematical Models and Computation, Oxford Financial Press, Oxford, 1993.
[39] P. Jaillet, D. Lamberton, B. Lapeyre, Variational inequalities and the pricing of American option, Acta Appl. Math. 21 (3) (1990) 263–289.

Advances in Engineering Software 186 (2023) 103550

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Performances analysis of parallel asynchronous and synchronous algorithms on cloud architecture for PDE

M.A. Rahhali [a], T. Garcia [b,c,*], P. Spiteri [c]

[a] ENGIE, La Défense Ile de France, France
[b] CY Paris University, CY Tech Campus de Pau 2 boulevard Lucien Favre CS 77563 64075 Pau Cedex, France
[c] University of Toulouse, IRIT-INPT-ENSEEIHT, 2 rue Charles Camichel, B.P. 7122, 31071 Toulouse - Cedex 7, France

ARTICLE INFO

Keywords:
High performance computing
Parallel computing
Cloud computing
Asynchronous iterations
Asynchronous algorithms
Iterative methods

ABSTRACT

In order to solve discretized stationary linear or non linear diffusion and convection–diffusion problems, the present paper deals with a comparison between the performances of parallel iterative synchronous or asynchronous algorithms implemented on a Cloud computing architecture with several sizes of the algebraic systems to be solved. The parallel algorithms are briefly and intuitively presented and analysis of their behavior is recalled. In particular, criteria of convergence are recalled and estimates of convergence speed are given. Implementation on cloud architectures are briefly described. Finally, for each kind of problem, the performance of the target algorithms on cloud architectures are analyzed.

1. Introduction

The discretization of linear or non linear boundary value problems leads to large and generally sparse linear or non linear algebraic systems. In order to solve these algebraic systems, we are currently parallelizing the resolution algorithms in order to significantly reduce the computation time. Among the possible algorithms we focus in the present paper on the parallel relaxation methods applied to the resolution of linear and non linear algebraic systems. A large advantage of the relaxation methods is that synchronous or asynchronous variants can be considered ; in the present work we will compare the performances of parallel iterative asynchronous methods with the synchronous ones ; previous studies have shown that asynchronous versions implemented on clusters or grids allowed to decrease the restitution time when there were many synchronizations between the computing processes. In other words, in this case the asynchronous versions strongly minimize the idle time of the processors. In these previous works (see for example [1]) from a computer sciences point of view we have also experimentally observed that asynchronous variants gave better results on grid-type architectures due to slow communications between processors because the bandwidth is shared by others machines. Indeed latencies penalize and affect the simulation results. Moreover, this effect is increased by the geographical distance between the machines and also, when modes are heterogeneous which is the case for example with computing grids or cloud computing. In the present paper we will then consider an implementation on cloud architecture and we will analyze the behavior of the two target algorithms on such cloud architecture applied to the solution of discretized boundary value problems. In Cloud Computing, Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) represent three customer-focused layers of abstraction. This type of representation allows customers to use the infrastructure components without managing them themselves. Given the fact that this last type of architecture includes additional software layers, we have the intuition that the implementation of iterative asynchronous methods will give better results in terms of work restitution time than the synchronous version.

Note that, although the relaxation methods are efficient in the case where the matrix of the algebraic system to be solved is not symmetrical, for the solution of large algebraic systems, we can solve the subproblems by other methods than this last algorithm. For example we refer the reader to Refs. [2,3] in which in order to solve the convection–diffusion problem in the case where the linear part is perturbed by à diagonal monotone operator, we implemented a parallel synchronous algorithm compared to the asynchronous one where the sub-problems are solved by the Krylov method [4]. In such implementation the parallel asynchronous method is more efficient than the synchronous one on a grid architecture.

The main goal of this study is then to compare the performance of asynchronous parallel algorithms implemented on cloud architecture with that of synchronous parallel algorithms, and to see the influence of the additional software layers present in the systems implemented in this type of architecture, which is well suited to solving very large-scale algebraic systems.

* Corresponding author at: University of Toulouse, IRIT-INPT-ENSEEIHT, 2 rue Charles Camichel, B.P. 7122, 31071 Toulouse - Cedex 7, France.
E-mail addresses: marahhali@outlook.com (M.A. Rahhali), thierry.garcia@irit.fr (T. Garcia), pierre.spiteri@enseeiht.fr (P. Spiteri).

https://doi.org/10.1016/j.advengsoft.2023.103550
Received 22 December 2022; Received in revised form 18 October 2023; Accepted 18 October 2023
Available online 3 November 2023

On the other hand, we are currently assisting in the development of new asynchronous domain decomposition methods including successfully coarse space into existing asynchronous Schwarz-type domain decomposition methods. For example in [5,6] are presented recent advances of scalable asynchronous domain decomposition methods ; we refer to these references for further information (see also [7] for an interesting contribution concerning asynchronous Schwarz solvers implemented on GPU's). In [5] the authors consider the use of asynchronous communications based on one-sided MPI (Message Passing Interface) primitive, previously investigated, for instance in [8], in the context of domain decomposition solvers and particularly a scalable additive two-level Schwarz method with comparison of synchronous and asynchronous variants of parallel solvers. In [6] a multiplicative variant of the additive coarse-space method is studied with implementation aspects and parallel numerical simulations showing the very good performances obtained when the asynchronous two-level method is used. Moreover in the Ref. [9] the authors study also the behavior of a scalable non-overlapping asynchronous domain decomposition method constituted by the classical primal Schur complement method where the interface problem is solved by using a multisplitting relaxation algorithm based on a weighted additive Schwarz preconditioner. According to the conclusion inferred in this paper such method outperforms significantly the two-level asynchronous restricted additive Schwarz method. In Ref. [10] the authors define also a scalable nonlinear application of the Restricted Additive Schwarz (RAS) method, which can be algebraically overlapping or not, and the original paper [11] is a clear indication of the RAS method. This paper extends the studies undertaken in the linear case [5] to the non-linear case where each local problem in each subdomain is solved with overlap, but the updates communicated to the other local subproblems are those corresponding to subdomains without overlap for contributing to the next iterate. To the question of knowing if the points of view developed in Refs. [5,6,9,10] are applicable or not to the problems considered or to come in our project of article, it seems that the answer is affirmative in the linear case developed in [5,6,9] and in the nonlinear case considered in [10] nevertheless except perhaps to the case of multivalued problems in the case where the solution is subjected to constraints of type inequality in which case the monotony plays an essential role; in this case the studies presented in [5,6,9,10] must find an adaptation not obvious to propose, but the point of view developed in Ref. [1] and related works allows to solve very simply this kind of nonlinearity by using monotony notions in Banach spaces.

The paper is organized as follows. Sections 2 and 3 deal with the presentation of parallel synchronous and asynchronous algorithms. In this presentation, for clarification, we avoid the use of an abstract formalism. We prefer illustrate simply the difference between synchronous and asynchronous parallel algorithms and indicate what advantage can be taken of the latter algorithm to reduce the restitution time of the computational work. For this parallel computation method, we first indicate how the problem is decomposed into interconnected subproblems, which leads us to present the subdomain methods with and without overlapping between the subdomains. We then describe synchronous and asynchronous variants of the parallel relaxation methods as simply as possible and avoiding a mathematical formalism. As this is an iterative method, starting from a linear problem and also from various pseudo-linear problem, we then indicate convergence criteria which are illustrated in case of each model problem to then lead to more general problems and extend the results presented in a simple case. Finally, the difficult issue of stopping asynchronous iterations is addressed from both a computational and a numerical point of view. For more details the reader is referred to the Ref. [1].

The Section 4 is relative to the implementation of parallel synchronous and asynchronous algorithms on cloud architecture which is a way of providing access to computer resources, characterized by its self-service availability, elasticity, openness, mutualization and pay-per-use; self-service and on-demand resources of storage capacity and computing power are based on the customer's needs. After having performed experiments with virtual machines in a previous study [12], the computational experiments were performed, for this new study, on France Grilles Cloud which offers users cloud services that allow on-demand computation, storage and networking, cluster instantiation, instantiation of web services dedicated to the scientific community and algorithm testing. Results of synchronous and asynchronous parallel simulation on cloud computing for the numerical solution of a linear diffusion problem and a linear convection–diffusion problem are presented and analyzed.

Finally a conclusion and prospects for further study complete this paper.

2. Formulation of parallel synchronous and asynchronous algorithms

As the first supercomputer capable of achieving Exaflop performance (i.e. 10^{18} operations per second) as been announced, researchers and practitioners of high performance computing are confronted with daunting challenges related to the efficient use of such massive computing power in order to accelerate numerical simulations in numerous domains (such as physics, chemistry, economy...) or applications in big data analytics and artificial intelligence. Modern supercomputers draw their power from extreme parallelism (up to millions of cores) and specialized processing units such as GPUs; this technological trends have a major impact on the design and development of numerical algorithms and software or fuel research in high performance computing with the premises of tackling problems which have been, so far, intractable.

Among the numerous numerical algorithms for solving large scale algebraic systems [4], we are especially interested in parallel relaxation methods. These algorithms are defined mathematically by associating a fixed point equation to the algebraic system to be solved. For example, let us consider first the solution of the following linear algebraic system

$$AU = G \tag{1}$$

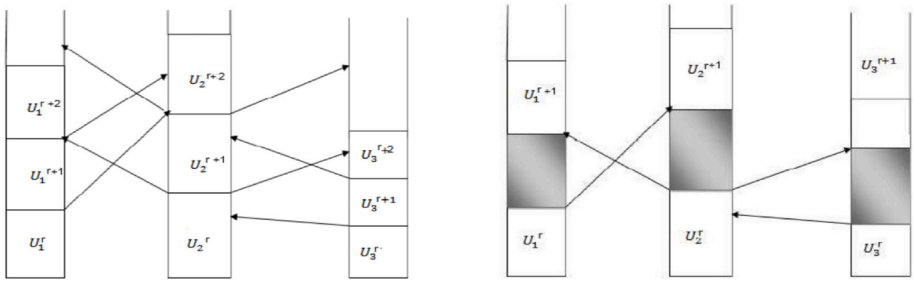
where A is a large sparse matrix generally derived from the discretization of a linear boundary value problem, G is a vector issued from the discretization and U is the unknown vector. Note that this kind of problem occurs when we have to solve elliptic, parabolic or hyperbolic second order boundaries values problems. Let us now associate to problem (1) the following fixed point problem

$$U = F(V) \tag{2}$$

with $F : D(F) \to D(F)$ where F applies from $D(F)$ to $D(F)$, the definition domain of F. For non-linear algebraic systems F can be defined implicitly or explicitly while in the linear case F is defined in an explicit way. For example in the linear case the parallel solution of problem (1) needs the decomposition of the problem in α interconnected linear subsystems associated to the block decomposition of the matrix A in large blocks as follows

$$A_{i,i}U_i = G_i - \sum_{j=1 j\neq i}^{\alpha} A_{i,j}V_j, i = 1, \ldots, \alpha \tag{3}$$

In order to obtain good performances of the parallel algorithms, each sub-problem is of sufficiently large size which allows to have a suitable granularity. This corresponds in fact to sub-domain methods and one can thus consider sub-domain methods without overlap and sub-domain methods with overlap, such as the classical Schwarz alternating method; such classification of sub-domain will be specified later. Let us also note that we can present all these methods in a unified framework constituted by the multisplitting methods (see [2,3]) where we perform a weighted average between various contracting fixed point mappings, each problem being solved in these two previous reference by Krylov methods [4].



(a) Behavior of parallel asynchronous iterations. (b) Behavior of parallel synchronous iterations.

Fig. 1. Behavior of asynchronous and synchronous parallel iterative relaxation algorithms.

In this kind of parallel methods of relaxation we distinguish two types of variants constituted on the one hand by synchronous parallel iterative relaxation methods and on the other hand by asynchronous parallel iterative relaxation methods corresponding respectively to two different modes of communication between the parallel processors.

Synchronous parallel iterative algorithms are computing methods in which the communications between the processors are synchronized at the end of each iteration. The synchronous parallel iterative method corresponds in fact to the most natural formulation of parallel algorithms and can be modeled by a Jacobi's like method. In this kind of method, when the load on each processor is not uniform or when the processors do not have the same performance, at each synchronization point the processing units must wait for the slowest processor. Consequently, given a heterogeneous distributed architecture, the idle times of the processing units will degrade the performance of these synchronous methods and the job restitution times will be penalized when a large lot of synchronizations is necessary to solve the target problem.

Asynchronous parallel iterative algorithms correspond to methods in which the communications between the processors are not synchronized at each iteration. Therefore, when a processing unit has finished its own calculations, it starts the next cycle again using the latest interaction data computed by the other processors and received during the previous cycle, without waiting for the arrival of new results delivered by the other processing units. So no expectation of new results computed by other units are necessary and the parallel computations are performed with available values computed and sended by other units. Thus no idle times will appear in this kind of communications. The asynchronous parallel algorithms are well modeled by introduction of delays in the formulation of the algorithm. The reader is referred to various papers describing formally the formulation of parallel asynchronous relaxation algorithms (see [1] - and for additional informations about our works see also [13] to [14]- and also for other contributions [15–22], etc.) ; note that the formulation of parallel synchronous relaxation algorithms appears like a particular case of the asynchronous when no delays are used in the iterative method.

Fig. 1 illustrate the behavior of synchronous and classical asynchronous parallel iterative relaxation algorithms and in the second case allow to view the disappearance of idle times.

Remark 1. However, there are asynchronous parallel methods defined with more flexible communications between the processors where the interaction data is used without a predetermined order (see for example [13,23]). Thus, the calculations are processed on each processor respecting the own rhythm of each processing unit and using the last available values calculated by the other processors. Depending on the frequency of updates of calculations between processors, the speed of convergence of these asynchronous parallel methods can be slower. But the big advantage lies in the fact that there is generally a reduction in the elapsed time to reach convergence.

The convergence of both synchronous or asynchronous parallel algorithms can be studied by various methods (see [1,24]). When the fixed point operator F associated to the problem to be solved is contracting [25], i.e.

$$\|F(U) - F(V)\| \leq \nu \|U - V\|, \forall\ V \in D(F), \tag{4}$$

where $0 \leq \nu < 1$, one obtains on the one hand convergence whatever be the decomposition of the problem into sub-problems and on the other hand the value of the contraction constant ν gives an estimate of the asymptotic speed of convergence as follows

$$\mathcal{R}_\infty = -\ln(\nu) \tag{5}$$

Note that depending on the sharpness of the decomposition we obtain smaller or larger contraction constants, their values then having an impact on the asymptotic speed of convergence. Note also that in both cases the convergence study of asynchronous relaxation method imply the convergence of synchronous relaxation method but, with a different asymptotic speed of convergence, since the speed of convergence in this kind of method depends on the frequency of updates.

Example 1. For example let us consider the numerical solution of the following elliptic boundary value problem defined in the unit cube $\Omega = [0,1] \times [0,1] \times [0,1]$ and equipped with the homogeneous Dirichlet boundary conditions

$$\begin{cases} -\Delta u(x,y,z) + q.u(x,y,z) = g(x,y,z) \text{ everywhere in } \Omega, q \geq 0 \\ u(x,y,z) = 0 \text{ everywhere in } \partial\Omega, \end{cases} \tag{6}$$

where $\partial\Omega$ is the boundary of Ω; then the discretization of problem (6) by classical finite difference method leads to the solution of the following equations. $1 \leq i,j,k \leq n$

$$\begin{cases} \frac{-u_{i+1,j,k}+2.u_{i,j,k}-u_{i-1,j,k}}{h^2} + \frac{-u_{i,j+1,k}+2.u_{i,j,k}-u_{i,j-1,k}}{h^2} + \frac{-u_{i,j,k+1}+2.u_{i,j,k}-u_{i,j,k-1}}{h^2} \\ \qquad +q.u_{i,j,k} = g_{i,j,k}, 1 \leq i,j,k \leq n \\ u_{0,j,k} = u_{n+1,j,k} = u_{i,0,k} = u_{i,n+1,k} = u_{i,j,0} = u_{i,j,n+1} = 0, 1 \leq i,j,k \leq n \end{cases} \tag{7}$$

where h is the uniform discretization step in space defined by $h = \frac{1.0}{n+1}$ and when $x_i = i.h$ (respectively $y_j = j.h$, $z_k = k.h$), $u_{i,j,k}$ is an approximation of $u(x_i, y_j, z_k)$. Then, when synchronous or asynchronous parallel relaxation subdomain methods without overlapping between the subdomains are used for the numerical solution of problem (6), this kind of method corresponding in fact to classical non-overlapping large-block relaxation methods, which themselves group together several adjacent blocks of the discretization matrix, according to the results set up for example in [1] the smaller contraction constant ν is given by

$$\nu = 1 - \frac{12}{6 + q.h^2} . \sin^2(\frac{\pi}{2}.h) - \frac{q.h^2}{6 + q.h^2} \tag{8}$$

and parallel synchronous and asynchronous iterative relaxation algorithms converge for all decomposition of the problem in α subproblems; in this case, when ϵ is small taking account of $\ln(1-\epsilon) \approx \epsilon$ the asymptotic speed of convergence is given by

$$\mathcal{R}_\infty \approx \frac{3.\pi^2+q}{6+q.h^2}.h^2 \tag{9}$$

since h is in general very small. Moreover note that when $q=0$ then the contraction constant ν is given by

$$\nu = 1 - 2.\sin^2(\frac{\pi}{2}.h) = cos(\pi.h).$$

and (9) reduces to the following expression

$$\mathcal{R}_\infty \approx \frac{\pi^2}{2}.h^2 \tag{10}$$

In the treatment of the previous example, estimates are obtained since many classical results concerning the eigenvalues of the discretization matrix of the Laplacian defined in the unit cube are known. When we are no longer in such an ideal situation we can however obtain estimates of the asymptotic speed of convergence. Indeed according to the results presented in [1] or [26] the convergence of parallel synchronous or asynchronous algorithms is obtained when the spectral radius of the Jacobi matrix $J = Id - D^{-1}A$ associated to A is less than one (see for example [26]), where Id is the identity matrix and D denotes the diagonal of A; then, classically (see [27,28]), in the general case we can also obtain an estimate of the contraction constant ν given by (see [1])

$$\rho(J) \leq \nu \leq \max_{1\leq i\leq M}\Big(\sum_{j=1,j\neq i}^{M}\big(\frac{|a_{i,j}|}{|a_{i,i}|}\big)\Big) \tag{11}$$

where M denotes the size of A. Such overestimation of ν corresponds in fact to a dominance diagonal property. More precisely (see [1,14,25]) for the numerical solution of discretized boundary value problems both parallel relaxation algorithms converge if the diagonal entries $a_{i,i}$ of A are strictly positive, the off-diagonal entries $a_{i,j} \leq 0$ and A is strictly or irreducibly diagonally dominant (i.e. A is an M-matrix - see [27,28]).

Remark 2. Note that the previous estimation applied to the problem (6) leads to

$$\rho(J) \leq \nu \leq \frac{6.0}{6.0+q.h^2} \approx 1.0 - \frac{q.h^2}{6.0};$$

this estimate is thus less sharp than the one obtained in (8)

Remark 3. Note that in estimates (11) the majoration takes into account the influence of rounding errors. In exact arithmetic we would have $\rho(J) = \nu$.

We concentrate here on parallel Schwarz alternating methods, which are based on overlapping subdomains.

Consider now the case where the linear system (1) derived from the discretization of problem (6) is solved by parallel asynchronous Schwarz alternating method ; this last method is also well suited to parallel computing. Recall that in this case problem (6) can be decomposed into α sub-problems as follows : for $i = 1,\ldots,\alpha$,

$$\begin{cases} -\Delta u_i + q u_i = g_i, \text{ everywhere in } \Omega_i, \\ u_{i_{/\Gamma_i}} = 0, \\ u_{i_{/\gamma_i^1}} = u_{i-1_{/\gamma_i^1}} \text{ for } 2 \leq i \leq \alpha, \\ u_{i_{/\gamma_i^2}} = u_{i+1_{/\gamma_i^2}} \text{ for } 1 \leq i \leq \alpha - 1, \end{cases} \tag{12}$$

where u_i and g_i, respectively, are the restriction of u and g, respectively, to Ω_i, $\Omega = \bigcup_{i=1}^{\alpha} \Omega_i$, $\Omega_i \bigcap \Omega_{i+1} \neq \emptyset, i \in \{1,\ldots,\alpha-1\}$, $\gamma_i^1 = \partial\Omega_i \bigcap \Omega_{i-1}, i \in \{2,\ldots,\alpha\}$, $\gamma_i^2 = \partial\Omega_i \bigcap \Omega_{i+1}, i \in \{1,\ldots,\alpha-1\}$ and $\Gamma_i = \partial\Omega_i \bigcap \partial\Omega, i \in \{1,\ldots,\alpha\}$ (see Fig. 2).

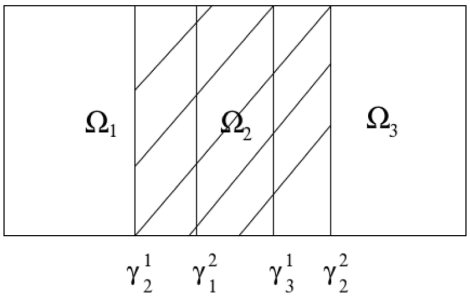


Fig. 2. Example of decomposition of Ω with 3 subdomains.

The decomposition (12) corresponds to a basic overlapping subdomain decomposition describing the principle of the method, whereby u_i is computed using the restriction of its immediate neighbors u_{i-1} and u_{i+1} respectively, on γ_i^1 and γ_i^2 respectively. In the sequential case, the scheme of computation corresponds exactly to a multiplicative Schwarz scheme. In the parallel case, the Schwarz alternating method can be combined with a synchronous or an asynchronous iterative scheme of computation with flexible communication in order to be as close as possible to a multiplicative scheme.

Thanks to a result stated by D.J. Evans and W. Deren [29] the augmentation process due to the Schwarz alternating method transforms the M-matrix A, i.e. a matrix where the diagonal entries $a_{i,i}$ are strictly positive, the off-diagonal entries $a_{i,j} \leq 0$ and in addition A is strictly or irreducibly diagonally dominant [27,28], into an M-matrix $\widetilde{A}$. Thus we are in the same framework than the ones previously considered in the case of subdomain methods without overlap; consequently the convergence criteria of synchronous or asynchronous parallel Schwarz alternating methods are identical to those established in the case of subdomain methods without overlap and if A is an M-matrix these methods are again convergent. Note that due to the overlapping between the subdomains generally the speed of convergence of the Schwarz alternating method is faster than the one corresponding to the subdomain method without overlap.

Example 2. Consider now the numerical solution of the following linear convection–diffusion problem

$$\begin{cases} -\Delta u + a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} + c\frac{\partial u}{\partial z} + qu = g, \text{ in } \Omega, \\ u(x,y,z) = 0 \text{ everywhere in } \partial\Omega, \end{cases} \tag{13}$$

where $q \geq 0$,

For the sake of simplicity, we assume that the discretization grid of the domain Ω is uniform. As previously in the sequel h denotes the discretization step-size. We assume that the columns of the discretization grid are numbered naturally in lexicographical order. The discretization of the operators which occur in problem (13) is made according to the following rules: the Laplacian is discretized via the classical seven points scheme presented in Example 1 and the first derivatives are discretized as follows according to the sign of a, b and c

$$\frac{\partial u}{\partial x} = \begin{cases} \frac{u(x,y,z)-u(x-h,y,z)}{h} + \mathcal{O}(h), \text{ if } a > 0, \\ \frac{u(x+h,y,z)-u(x,y,z)}{h} + \mathcal{O}(h), \text{ if } a < 0. \end{cases} \tag{14}$$

and accordingly for the derivative with respect of y and z.

Let A denote the discretization matrix of the problem (13). If q is strictly positive, then regardless the sign of a, b and c it follows from (14) that off-diagonal entries of matrix A are non-positive and diagonal entries of A are positive. Moreover, the matrix A is strictly diagonally dominant; then we are in the same framework presented in Example 1.

If $q = 0$, then we can show that the matrix A is diagonally dominant. Moreover, by using the characterization of irreducible matrices (see [28]) we can verify that the matrix A is irreducibly diagonally dominant. Thus the framework presented in Example 1 is again satisfied.

Consider now a red-black ordering of the grid points and let $\hat{A}$ be the corresponding discretization matrix derived from A by a permutation which preserves the sign of the entries. We consider the former discretization scheme; if q is strictly positive, then, the matrix $\hat{A}$ is strictly diagonally dominant; if $q = 0$, then we can show analogously that the matrix $\hat{A}$ is irreducibly diagonally dominant. Thus, in both cases $\hat{A}$ satisfy the theoretical framework presented in this section.

Remark 4. Finally, note that under realistic hypotheses, the finite element discretization matrix and the finite volume discretization matrix occurring in some analogous linear partial differential equations satisfy the theoretical framework presented below.

For simplicity, let us consider first the case where $\Omega = [0,1] \times [0,1] \times [0,1]$ and let us compute the contraction constant ν for various values of a, b and c in the case of the subdomain method without overlapping. Recall that the eigenvalues of the discretization matrix, an heptadiagonal matrice with classically a block structure, are given for $i, j, k = 1, \ldots, n$, by

$$\lambda_{i,jk}(A) = \mu - 2.\sqrt{\theta.\beta}.cos(\frac{i.\pi}{n+1}) - 2.\sqrt{\delta.\gamma}.cos(\frac{j.\pi}{n+1}) - 2.\sqrt{\zeta.\eta}.cos(\frac{k.\pi}{n+1}),$$

where μ are the diagonal entries of A, θ and β are the codiagonal entries, δ and γ are the closest entries to the diagonal blocks and ζ and η are the entries of the most extreme blocks.

For example if a, b and c are positive real numbers and for decentered discretization of the first derivative given by (14), the spectral radius is given by

$$\nu = 2.\frac{\sqrt{1+a.h} + \sqrt{1+b.h} + \sqrt{1+c.h}}{6 + q.h^2 + a.h + b.h + c.h}.cos(\pi.h). \tag{15}$$

since $\mu = 6.0 + q.h^2 + a.h + b.h + c.h$ and according to the numbering of the grid points, for example $\theta = \delta = \zeta = -1.0$ and $\beta = -(1.0 + a.h)$, $\gamma = -(1.0 + b.h)$ and $\eta = -(1.0 + c.h)$. Then we can verify that in this context $0 < \nu < 1.0$

Remark 5. Note that estimate of ν given by (15) is very sharp. Il we use the estimate given by (11) we obtain the following result

$$\nu \leq \tilde{\nu} = \frac{6.0 + a.h + b.h + c.h}{6 + q.h^2 + a.h + b.h + c.h}$$

For other combinations of signs of the coefficients a, b and c we obtain also easily that $0 < \nu < 1.0$. This property again comes from the fact that the matrix A has positive diagonal entries and negative or zero off-diagonal entries and that moreover it is either strictly diagonal dominant or irreducibly diagonal dominant. Note that such important property is again verified when Ω is of any form.

In a similar way to our approach during the study of Example 1, we can also solve the convection–diffusion equation by the alternating Schwarz method. If problem (13) is decomposed into α sub-problems then for $i = 1, \ldots, \alpha$, using the same previous notations and the same principle of presentation of the algorithm, this method is defined by

$$\begin{cases} -\Delta u_i + a\frac{\partial u_i}{\partial x} + b\frac{\partial u_i}{\partial y} + c\frac{\partial u_i}{\partial z} + q u_i = g_i, \text{ everywhere in } \Omega_i, \\ u_{i_{/\Gamma_i}} = 0, \\ u_{i_{/\gamma_i^1}} = u_{i-1_{/\gamma_i^1}} \text{ for } 2 \leq i \leq \alpha, \\ u_{i_{/\gamma_i^2}} = u_{i+1_{/\gamma_i^2}} \text{ for } 1 \leq i \leq \alpha - 1, \end{cases} \tag{16}$$

Once again, the decomposition (16) corresponds to an overlapping subdomain decomposition, whereby u_i is computed using the restriction of its immediate neighbors u_{i-1} and u_{i+1} respectively, on γ_i^1 and γ_i^2 respectively, and the same remarks like the ones formulated in the presentation of Example 1 can be formulated.

The effectiveness of domain decomposition methods is well known for boundary value problems. These methods are also well suited to parallel computing (see [30]).

Note that in the general case when Ω is of any form, since once again the augmentation process do not alter the properties of the discretization matrices, using the result stated by D.J. Evans and W. Deren [29], the criteria viewed in the case of the problem of diffusion are still valid in the present case, i.e. when the diagonal entries are strictly positive, the off-diagonal entries are non positive and the discretization matrix is strictly or irreducibly diagonal dominant.

Remark 6. The previous schemes correspond to decentered finite difference scheme for the discretization of the first derivative; such decentered scheme allows to obtain strongly diagonal dominant matrices, which has a strong impact on the convergence speed. Note that it is also possible to consider a centered finite difference scheme defined by the following difference equation corresponding to an approximation of $\frac{\partial u}{\partial x}$

$$\frac{\partial u}{\partial x} \approx \frac{u(x+h, y, z) - u(x-h, y, z)}{2.h};$$

use of such centered scheme cannot provide any guarantee that the discretization matrix satisfies the required properties ensuring the convergence of the iterative process, especially when the convection coefficients, i.e. the coefficients of the first derivative, are dominant. In this case the discretization matrix A is not necessarily diagonal dominant and for large values of the coefficients of convection, off-entries of A are not necessarily negative. Such properties of diagonal dominance and off-diagonal entries negative are satisfied with very restrictive conditions verified by the convection coefficients, i.e.

$$|a| + |b| + |c| \leq \frac{6}{h} + q.h$$

and

$$|a| \leq \frac{2}{h}\ ;\ |b| \leq \frac{2}{h}\ ;\ |c| \leq \frac{2}{h}.$$

In fact the previous restrictive conditions are not necessary. However according to the results presented in [1] the use of such centered schemes can be used and leads to the conclusion that parallel synchronous and asynchronous iterative methods converge. Such convergence property arises from the fact that the discretization matrix A is the sum of a symmetric defined positive matrix arising from the discretization of the Laplacian and an anti-symmetric matrix arising from the discretization of the convection terms. Then, using appropriate norm for analyze the convergence, i.e. in this case the euclidean norm, it can be verified very easily that the contraction constant ν is given by (8) in the case of the resolution of the convection–diffusion problem ; thus in this framework only the diffusive part has an influence on the convergence speed, which is a little bit restrictive. Note that from a numerical point of view the use of decentered schemes such as (14) is recommended to avoid boundary layer problems. Moreover the speed of convergence of the iterative process is increased.

3. Numerical solution of nonlinear convection–diffusion equation

3.1. The general situation

Note that parallel asynchronous and synchronous iterative relaxation algorithms can also be applied for the solution of the following large pseudo-linear single-valued algebraic systems

$$AU + \Phi(U) = G \tag{17}$$

corresponding to a linear system perturbed by a diagonal increasing operator (or more generally monotone operator). Another situation

corresponds to the case where the solution U is subject to the following constraints of inequalities type

$$U_{min} \leq U \text{ or } U_{min} \leq U \leq U_{max} \text{ or } U \leq U_{max}; \tag{18}$$

in this last case the solution U satisfies the following large pseudo-linear multi-valued algebraic systems

$$AU + \partial\Psi(U) - G \ni 0 \tag{19}$$

where Ψ is the characteristic function defining the convex set of constraints and $\partial\Psi(U)$ is the sub-differential of Ψ corresponding in fact to a weak derivative of Ψ; classically the mapping $U \rightarrow \partial\Psi(U)$ is still a diagonal monotone operator. Considering this property of monotony of the operators $\Phi(U)$ and $\partial\Psi(U)$ the convergence criteria of the synchronous and asynchronous parallel algorithms previously stated in the linear case are still valid for the solution of the problems (17) and (19); therefore, once again, if the diagonal entries $a_{i,i}$ of A are strictly positive, the off-diagonal entries $a_{i,j} \leq 0$ and A is strictly or irreducibly diagonally dominant then for the resolution of previous problems (17) and (19) the both parallel relaxation algorithms converge and the estimate (11) of the contraction constant ν is at least still valid for the previous two problem and are still valid when subdomain methods without overlapping are implemented.

More generally for the Schwarz alternating method the criteria of convergence are identical to those presented in Section 2.

Remark 7. In fact the Schwarz alternating method is a particular case of the multi-splitting method and the convergence criteria mentioned above are still valid (see [2,3]).

Remark 8. Note also, in nonlinear framework, that convergence can be analyzed by partial ordering technique related to the use of the discrete maximum principle. In this case provided that the iterative process is initialized by an initial data U^0 (respectively V^0) such that for the problem (17) only we have $AU^0 + \Phi(U^0) - G \geq 0$ (respectively $AV^0 + \Phi(V^0) - G \leq 0$) we generate a sequence U^p decreasing since U^0 (respectively V^p increasing since V^0). The convergence criteria are identical to the previous ones and concern as well as the signs of the coefficients of the matrix A and the diagonal dominance properties. However, this type of analysis does not allow to estimate the speed of convergence of the iterations, contrary to what was possible in the case of an analysis by contraction techniques previously presented. Note that to our knowledge the analysis of the same algorithms by partial order techniques for the problem (19) is an open problem.

The previous methods being iterative in nature must be stopped when the iterated values are stabilized. They are thus completed by a test of stopping the iterations. In the case of synchronous parallel algorithms, this iteration stopping test is identical to the one implemented in the sequential case. For asynchronous parallel algorithms the iteration stopping test is much more delicate to implement. Thus, in this latter case we will distinguish the computer science approach and the numerical analysis approach and to stop the iterative process we have to combine these two approaches.

Concerning the computer science approach we refer to several works concerning such stopping criteria for example [31–36].

In our implementations a stopping test of the iterations is considered from the computer sciences point of view ; thus we have used the circulation of a token corresponding to the simplest implementation of stopping test but coupled as we will see below by a test derived from numerical analysis considerations. According to [37,38] in this case of parallel asynchronous methods we consider a decentralized algorithm where convergence is achieved in this way. Each processor updates the components of the iterate vector associated with each one's own block and computes the residual norm attached to this block in order to participate to the convergence detection. Convergence occurs when a given predicate on a global state is true. A usual predicate corresponds to the fact that the iterate vector generated by the asynchronous iterative algorithm is sufficiently close to the solution of the problem (see [19], page 580); thus, on every process, terminaison detection is obtained when the norm of the local residue remains under a given threshold after two successive updates of the component (see [37,38]). Due to the termination and the detection of the global state of the processes, the implementation of each variant of parallel asynchronous method is then more complex than the synchronous one.

In the case of asynchronous iterations, point to point communications between two processes have been implemented using nonblocking send and receive communications. A test function is used in order to allow any processor to continue the computations without having to wait for the completion of any pending send or receive operations. Idle times due to synchronizations in message passing are suppressed in this way. On the other hand, the parallel synchronous iterative schemes are implemented by using blocking send and receive communications. One has to take care about the deadlock issue when implementing synchronous communications using blocking communications. For more details concerning the implementation and the convergence detection of the considered parallel asynchronous methods, the reader is referred to [38].

Concerning the numerical analysis approach for an efficient terminaison it can be proved that the successive iterates are located in nested sets centered in the solution (see [19,39,40]), which allows to implement efficient numerical stopping tests insofar as one can give an estimate of the diameters of each nested set and stop the iterative process when all diameters are smaller than a given tolerance. Consequently the coupling of the stopping tests derived from computer science and numerical analysis considerations allows to obtain an efficient terminaison of the parallel asynchronous iteration. In previous studies (see [1–3]) we have implemented these methods on grids constituted by heterogeneous and distant machines; we could observe that asynchronous algorithms were very efficient when there was a lot of synchronization between the processors.

3.2. Numerical solution of nonlinear convection–diffusion equation

We turn now to the case where nonlinearities are defined in the domain Ω. As nonlinearities we will consider the following linear problems perturbed by an increasing diagonal operator or more generally monotone. In agreement with the results presented in [1], considering the monotony of the latter operator the convergence criteria will not change and in particular the speed of convergence will be of the same order for linear and nonlinear problems; indeed if we study the convergence by contraction techniques, the contraction constant ν occurring in (4) is the same as in the linear case. So, in both cases, the performances and the behavior of the parallel algorithms must be substantially similar whether the communications between the processors are synchronous or asynchronous, the speed of convergence being frequency dependent of the updates.

The general model can be given as follows

$$\begin{cases} -\Delta u + a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} + c\frac{\partial u}{\partial z} + qu + \varphi(u) = g, \text{ in } \Omega, \\ B.C., \end{cases} \tag{20}$$

where $q \geq 0$, g is a summable square function in Ω, $\varphi : u \rightarrow \varphi(u)$ is a continuous, nondecreasing function and B.C. represents a classical boundary condition, i.e. Dirichlet, Neumann, Robin or mixed.

Concerning the nonlinear part, we consider different types of nonlinear convection–diffusion problems where nonlinearities arise on the boundary or in the domain.

The following nonlinear functions: $\varphi(u) = e^{\alpha u}$, with $\alpha > 0$ or $\varphi(u) = Log(\beta + \delta u)$, with $\delta > 0$ and a suitable sign for β, such that $\beta + \delta u > 0$, can be considered. Then we have to solve an algebraic system like (17), where Φ is a diagonal operator derived from the discretization of φ; according to the properties of φ then Φ is a monotone diagonal increasing mapping. Consequently we are again in the framework

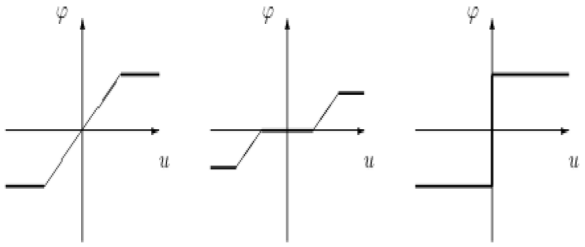


Fig. 3. Different graphs for φ..

presented in Section 2 and the convergence of parallel synchronous or asynchronous relaxation methods is verified with the same speed of convergence since the properties of the mapping $\varphi(u)$, i.e. the property of monotony, do not change the estimate of convergence rate.

Remark 9. The numerical solution of problem (20) where the nonlinearity occurs on the whole domain Ω is done using a Newton method provided that the application $u \to \varphi(u)$ is everywhere derivable on Ω. We refer to [2,3] where tests of synchronous parallel resolution compared to their asynchronous counterparts are presented and implemented on GRID'5000.

We can also consider nonlinear convection–diffusion problems where nonlinearities arise on the boundary of the domain. This kind of problem occurs, for example, in the following boundary temperature control problem

$$\begin{cases} -\Delta u + a\frac{\partial u}{\partial x} + b\frac{\partial u}{\partial y} + c\frac{\partial u}{\partial z} + qu = g \text{ everywhere in } \Omega, \\ \frac{\partial u}{\partial n} + \varphi(u) = 0 \text{ on } \Gamma_d \text{ and } u = 0 \text{ on } \partial\Omega - \Gamma_d, \end{cases} \tag{21}$$

where $\Omega \subset \mathbb{R}^3$, $q \geq 0$, $\Gamma_d \subset \partial\Omega$, g is a summable square function in Ω and $\varphi : u \to \varphi(u)$ where $\varphi : \mathbb{R} \to \mathbb{R}$ is a continuous, nondecreasing, nonlinear function. Fig. 3 displays some examples of graphs for function φ. In particular, the two first graphs model saturation phenomena and the third graph models a multi-valued function corresponding to the boundary condition: $\frac{\partial u}{\partial n} + \varphi(u) \ni 0$ occurring when u is subject to constraints inequality

The discretization techniques presented above can be also used for the interior points of domain Ω. For all points in Γ_d, the discretization of the Neumann condition leads to the solution of the following discrete equations

$$\frac{u_l - u_{l-1}}{h} + \varphi(u_l) = 0. \tag{22}$$

where l is an index of a point which belongs to Γ_d. Thus, if φ is singlevalued we have to solve the problem

$$\mathcal{A}(U) = AU + \Phi(U) - G = 0, \tag{23}$$

where A is the discretization matrix associated with the linear part of the equations, Φ is a diagonal, nondecreasing operator and $(G, U) \in \mathbb{R}^{dim(A)} \times \mathbb{R}^{dim(A)}$.

It follows from (22) that the lth component of φ is equal to $h\varphi(u_l)$ if l is the index of a point which belongs to Γ_d otherwise this component is null. If $q > 0$, then the matrix A is a strictly diagonally dominant matrix with strictly positive diagonal entries and off-diagonal entries non-positive. Then we are in the framework presented in Section 2. In the case where $q = 0$, we can verify by a similar argument that the matrix A is irreducibly diagonally dominant, regardless the sign of a, b and c, and once again we are in the framework presented in Section 2. According to the properties of A and since Φ is a continuous, nondecreasing, diagonal mapping, then parallel synchronous or asynchronous subdomain methods converges.

Remark 10. We consider now the resolution of the nonlinear convection–diffusion problem (21) by limiting ourselves to a nonlinearity corresponding to the two first graphs of Fig. 3 ; indeed the nonlinearity presented in the last graph of this last figure, corresponding to the resolution of a problem with inequality constraints of type (18) will be the subject of a future study. It should be noted that the applications presented in the first two graphs of Fig. 3 are not differentiable everywhere, especially at the junction of the parts where φ is constant with its increasing parts. Therefore the use of Newton's method is not possible in this case. On the other hand one can consider the use of a point relaxation method on each of the sub-domains particularly well adapted to the solution of Eq. (22). To solve this last equation one can implement a method of successive approximations of the type

$$u = \psi(v)$$

with $\psi(v)$ defined by $\psi(v) = h.\varphi(v) + (\tilde{u})_{i-1}$ where $\tilde{u}_{i-1}$ represents the last available value of u_{i-1}. Let β be the maximum absolute value of the slope of the increasing portions of the application $u \to \varphi(u)$; the Lipschitz constant of the application $u \to \psi(u)$ is then equal to $h.\beta$ and for a sufficiently appropriate small value of h, then $h.\beta < 1$ and ψ is contracting and the solution of (22) by the method of successive approximations converges.

Consider now the most general case, i.e. the nonlinear case. We have $\mathcal{A}(U) = AU + \Phi(U) - G$. Assume that A have strictly positive diagonal entries, off-diagonal entries such that $a_{i,j} \leq 0$ and is strictly or irreducibly diagonal dominant and moreover $\Phi(u)$ is a monotone increasing mapping. Then, we are in a context similar to (23). If we solve the nonlinear simultaneous equations $\mathcal{A}(U) = 0$, via the Schwarz alternating method, then the augmentation process of the Schwarz alternating method transforms again the M-matrix A into an M-matrix $\widetilde{A}$ and the monotone increasing mapping Φ into the monotone increasing mapping $\widetilde{\Phi}$ (see [13,29]). Thus, we are in the convergence analysis framework considered above for convergence in [13].

Remark 11. Moreover, to solve the global problem, we consider an overlapping subdomain method corresponding to the alternating Schwarz method which will also be convergent given the results stated in Example 2. From an implementation point of view in the two-dimensional case where the domain Ω is divided into slices, Fig. 2 represents the schematic diagram of this method. However, in Example 2 the domain Ω is three-dimensional and we did not choose a slicing as shown in Fig. 2. The domain Ω has been decomposed into small overlapping three-dimensional elements, for example into small parallelograms. So instead of having two exchanges with its right and left neighbors, this small element will perform data exchanges with six overlapping neighbors, i.e. with neighbors located at the top, bottom, right, left, front and back. So, in this last case of decomposition, there are three times as many communications but the behavior of the algorithm is more multiplicative.

Remark 12. Note that any direct or iterative method can be used on each subdomain. In the case where $\nu = 0$, the discretization matrix is triangular [38]. Thus, a relaxation method converges in only one iteration when the scanning order of the grid matches the triangular discretization matrix. In the case where ν is small, the discretization matrix is nearly triangular. The entries associated with a triangular part of the matrix derived from the decentered discretization scheme have higher order of magnitude than other entries. A relaxation method can then be a quasi-direct method on each subdomain and its computational cost can be very low.

4. Implementation and parallel experiments on cloud architecture

The implementation of the studied problems is briefly detailed since there is not difficult once the resources are allowed to the user; this

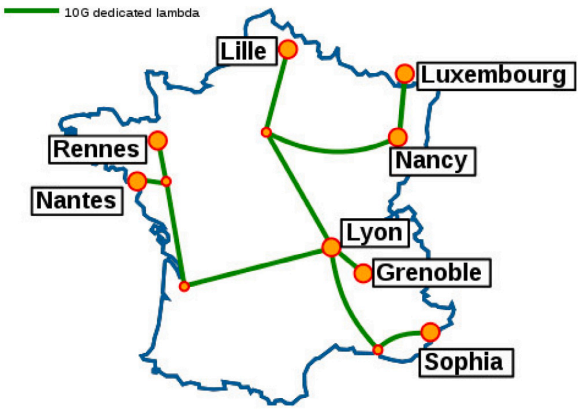


Fig. 4. Platform architecture of Grid'5000.

implementation is carried out with MPI (Message Passing Interface) facilities and the computational codes have been written using C or Fortran language. Matrix and right hand side creation have been implemented by a parallel way. For each simulation, we have considered a cubic domain Ω contained in the 3D space.

In the case of parallel subdomain algorithms without overlapping a block relaxation method is performed on each process and only one subdomain is assigned to each processor. A subdomain is constituted by gathering adjacent blocks. The principle of implementation of parallel asynchronous and synchronous iterative subdomain methods is describe by Algorithm 1.

Algorithm 1: Principle of implementation

```
while not global convergence do
    for each subdomain do
        Perform communications of subdomain boundary values
        Perform block relaxation resolution
    end
end
```

For comparison of synchronous and asynchronous performances, the number τ which is the ratio of elapsed time between synchronous and asynchronous methods given by

$$\tau = \frac{synchronous\ elapsed\ time\ algorithm}{asynchronous\ elapsed\ time\ algorithm}$$

is used; so τ is significant to achieve a comparison between the two communication modes. Indeed, in the case of heterogeneous architecture the notion of speed up and efficiency is not relevant since it is difficult to decide which machine is the reference and also all the more so since for large algebraic systems it is not possible to have the restitution times in sequential mode.

4.1. Parallel experiments on cloud like architecture

In the short paper [12], synchronous and asynchronous parallel algorithms simulation was carried out on a cloud computing like architecture based on a grid platform (called Grid'5000 see [41]). This platform (see Fig. 4), provides access to a large amount of resources (8 sites, 800 compute-nodes), gives an easy access to a wide variety of hardware technologies and is particularly suitable to carry out experiments.

This platform can be used as an infrastructure proposed by a Cloud provider which shares mutualized resources and, a virtual computer corresponds to a quantity of virtual resources. It also allows the execution of virtual computers with hardware virtualization capabilities.

Table 1
Synchronous and asynchronous simulations for solving problem (6) by subdomains method without overlap.

Size	Synchronous		Asynchronous		τ
n^3	Relaxations	Elapsed time (s)	Relaxations	Elapsed time (s)	
360^3	2 866 016	6 076	3 815 801	1 308	4.6
480^3	6 364 544	38 460	7 895 476	7 728	5.0

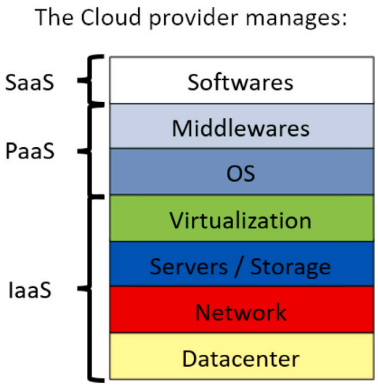


Fig. 5. Cloud services.

In this paper, we simulated problem (6) with several sizes 360^3 and 480^3 on 64 computers resources. In the following Table 1, synchronous and asynchronous results are presented in term of elapsed time and number of relaxations. It can be noticed that for the considered application and the used architecture, the asynchronous computation scheme gives better results than the synchronous scheme; on 64 machines; such algorithm is clearly faster by 4.6 and 5.0 respectively.

4.2. Parallel experiments on cloud architecture

Grid architecture is designed as a distributed system and file manager with the advantage of shared resources but on one hand the disadvantage of a tendency toward complexity for Application Programming Interface (API) which is a way for computers programs to communicate with each other, and on the other hand the deployment of services on this infrastructure.

So the Cloud has emerged from the convergence of several ideas following the maturation of technology for visualization with a simplified API look and feel and also large computing capacity to meet resource requirements.

The Cloud computing (see [42]) is a way of providing access to computer resources, characterized by its self-service availability, elasticity, openness, mutualization and pay-per-use; self-service and on-demand resources of storage capacity and computing power are based on the customer's needs. In the Cloud, the need, automatically detected by the application or at the customer's request, is taken into account and satisfied immediately and we no longer speak of infrastructure components but of services (see Fig. 5).

The different services are :

- IaaS (Infrastructure as a Service) which provide virtualized environments remotely by showing them as physical machines (CPU, disk, memory, network ...). Its advantages are that we can have a customized environment, completely controllable, accessible at any time and usable with a simple API.
- PaaS (Platform as a Service) which provides ready-to-use software and middleware (database, web server, etc.) as, for example, a platform for developing web applications but also an infrastructure for deploying and running applications. Its advantage is that

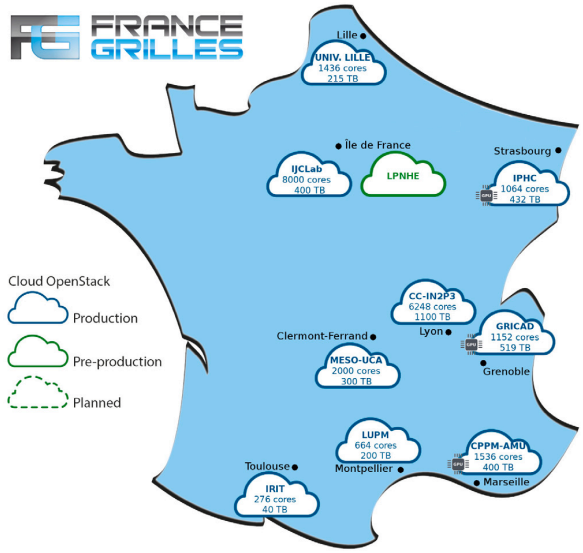


Fig. 6. Cloud resources of FG Cloud.

it provides load balancing and service redundancy features for example. Programmers can develop at a higher level.

- SaaS (Software as a Service) which provides ready-to-use software. It is an application accessible to the web as a hosting service. Its advantages are its simplicity of use with no software deployment, a web interface and its accessibility. The functioning of this service allows to the users to connect and share their applications over internet.

For this new experimental study, computational experiments have been carried out on France Grilles-Cloud (see Fig. 6) which offers to users cloud services that allows computing, storage and network on demand, clusters instanciation, web services instanciation dedicated to the scientific community and tests of algorithms.

Particularly, we use the Computer Center of the National institute of nuclear and particle physics (IN2P3), a french CNRS support and research unit. This institute that pursues and coordinates research on particle physics, nuclear physics and astroparticle physics. Researchers can use servers on which computer calculations are shared through a scheduling system. Calculations may be sequential or parallel. Computing servers, which run a Linux system, mostly feature Intel x86_64 processors and, to a lesser extent, special graphic processors. The resources may be used locally or remotely via a computing infrastructure. The Computing Center have external links of up to 100 Gbit/s and a unique internal network with a 400 Gbit/s backbone.

Our parallel simulations can therefore be run with reserved resources in the platform presented above with different numbers of cores, memory size and CPU. These resources are operating system templates and each instance is it a machine which runs our workloads in the cloud, start as an identical clone of all other instances. Once the client is functional, using a cloud instance is done in 4 steps :

- provide a Secure Shell Protocol (SSH) - which is a network communication protocol that enables two computers to communicate — to access the instances;
- instantiate a server;
- assign a public address IP (Internet Protocol) - which is a series of numbers that identifies the connection to the internet — to the instance;
- connect to the instance.

Table 2
Openstack instance list.

Instance name	Networks	Image (OS)	Flavor
mavm1	Public IP	Ubuntu-20.04	1CPU_8GB-RAM
mavm2	Public IP	Ubuntu-20.04	1CPU_12GB-RAM
mavm3	Public IP	Ubuntu-20.04	2CPU_4GB-RAM
mavm4	Public IP	Ubuntu-20.04	4CPU_8GB-RAM
mavm5	Public IP	Ubuntu-20.04	2CPU_4GB-RAM
mavm6	Public IP	Ubuntu-20.04	4CPU_8GB-RAM
mavm7	Public IP	Ubuntu-20.04	4CPU_8GB-RAM
mavm8	Public IP	Ubuntu-20.04	4CPU_8GB-RAM

In this Computer Center, OpenStack which is a collection of open source software modules and tools that provides a framework to create and manage both public cloud and private cloud infrastructure, is used.

Moreover, OpenStackClient which is a command-line client for OpenStack that brings the command set for the various APIs together in a single shell with a uniform command structure, is used for managing instances.

After creating an SSH key pair, it is possible to establish a SSH session to instances and it is then necessary to provide with Openstack the public key to be injected in the instances for the root account. Then, after consulting the list of available images, we work with the available flavors list. In fact, the flavors represent the available types of virtual machines and determine the virtual machines characteristics. Note that we have virtual characteristics composed of number of CPU (Central Processing Unit) and size of the RAM (Random Access Memory). After creating instances, it is possible to run the instances and realize the experiments (see Table 2). In our project, the network associates a floating IP to our instances in order to have it connected on the network.

For the problems (6) and (13) solved by subdomain method without and with overlap, we have considered the cubic domain Ω with different sizes such as 120^3, 240^3, 360^3 and 480^3.

In the following subsections, we consider two distinct kinds of boundary value problems : on one hand a linear diffusion problem and on the other hand a linear convection–diffusion problem without and with overlapping.

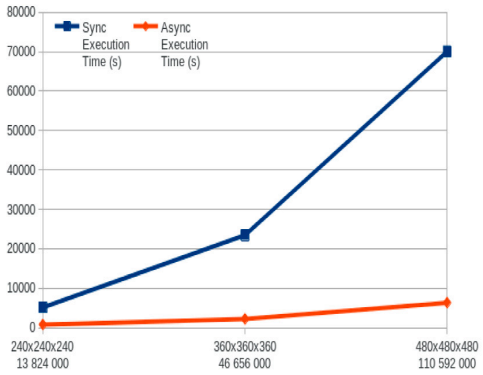
As previously described, we use the Iaas mode. The implementation in the cloud environment is done in a classic way and without real difficulty. Once the resources have been requested, we obtain them without knowing where they come from, their location and their hardware configuration. Moreover, we do not have any information about the possible sharing of physical machines between several users and the latency in the network connections between the 8 resources is not well known at a given time. Moreover, we did not have access to all 22 CPUs mentioned in Table 2, but to a limited number of these CPUs through the use of 8 virtual CPUs.

We started with an installation with a uniform operating system image on the Cloud resources. As mentioned before, our codes were developed in C or Fortran language with the help of the MPI library and scripts were set up to deploy the codes on the cloud resources as well as the libraries necessary for the functioning of the MPI environment. Other scripts were also used to run the simulations and retrieve the results.

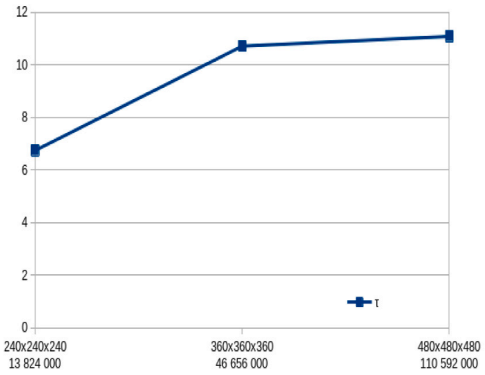
The number of machines used has been limited up to 8 according to the instances reservation; moreover, since the overhead damages the performances of the parallel algorithms, additional machines are not necessary particularly when synchronous scheme are tested.

Numerical solution of the linear diffusion equation implementation.

We consider first the numerical solution of problem (6), for which the results of experiments are summarized in Table 3 for $q = 1000$. We took an high value of q since the system is ill-conditioned knowing that this choice penalizes asynchronous algorithms because the convergence is fast and there is few synchronization. In these table and the



(a) Elapsed time with respect to domain sizes.



(b) Variation of the value τ with respect to the domain sizes.

Fig. 7. Results for problem (6) on 8 cloud machines for $q = 1000$ by subdomains method without overlapping.

Table 3
Synchronous, asynchronous and τ simulation results for solving problem (6) by subdomains method without overlap.

Size	Synchronous		Asynchronous		τ
n^3	Relaxations	Elapsed time (s)	Relaxations	Elapsed time (s)	
240^3	6 128	5 141	42 682	762	6.75
360^3	11 904	23 473	43 706	2 191	10.71
480^3	18 920	70 059	44 956	6 319	11.09

Table 4
Synchronous, asynchronous and τ simulation results for solving problem (13) by subdomains method without overlap.

Size	Synchronous		Asynchronous		τ
n^3	Relaxations	Elapsed time (s)	Relaxations	Elapsed time (s)	
240^3	23 840	20 772	59 243	1 943	10.70
360^3	133 728	271 516	243 584	11 880	22.86
480^3	183 520	965 295	296 317	42 480	22.72

Table 5
Synchronous, asynchronous and τ simulation results for solving problem (13) by subdomains method with overlap of two meshes.

Size	Synchronous		Asynchronous		τ
n^3	Relaxations	Elapsed time (s)	Relaxations	Elapsed time (s)	
120^3	11 475	4 578	131 472	441	10.38
240^3	43 889	110 836	245 576	5 720	19.38
360^3	89 601	496 270	290 256	21 887	22.67
480^3	171 671	1 666 259	348 488	61 747	26.99

following Figs. 7(a) and 7(b), comparison of elapsed time for parallel synchronous and asynchronous methods as well as the parameter τ are presented.

This tables and Fig. 7(a) show clearly the benefits of using parallel asynchronous algorithms in the parallel experiments. Finally, in Fig. 7(b), note that the parameter τ is increasing with the dimension of the discretization matrix.

Numerical solution of the linear convection–diffusion equation implementation without and with overlapping.

We consider first the numerical solution of problem (13), for which the results of experiments without overlapping are summarized in Table 4 and with an overlap of two meshes in Table 5 with both $a = 0.5, b = 1.5$, $c = 1.0$ and $q = 10$. In these tables and the following Figs. 8(a), 8(b), 9(a) and 9(b) comparison of elapsed time for parallel synchronous and asynchronous methods as well as the parameter τ are presented.

In Figs. 8(b) and 9(b), note that the parameter τ can be increasing with the dimension of the discretization matrix. Finally, note again that the use of asynchronous parallel schemes is very interesting in cloud computing with distant and heterogeneous instances for the second problem considered too.

- *Experiments without overlapping* (see Table 4 and Figs. 8(a), 8(b))
- *Experiments with overlapping* (see Table 5 and Figs. 9(a), 9(b))

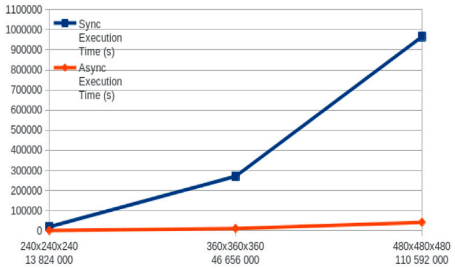
Synthesis of the analysis.

The results obtained confirm our expectations, i.e. that the asynchronous parallel algorithms are more efficient when using cloud resources without knowledge of their positioning, shared usage, network latency and hardware architecture.

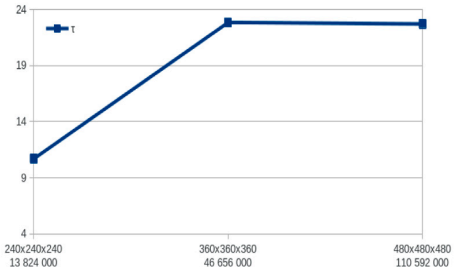
We can noted that, in asynchronous parallel experiments, the number of relaxations necessary to reach convergence is greater than the one necessary when parallel synchronous schemes are used. In fact, this is a well-known drawback of asynchronous iterative schemes, due to the use of possibly delayed components. It turns out that the overhead generated by additional relaxations in the case of asynchronous algorithms is smaller than the synchronization overhead combined with processor idle time of synchronous parallel schemes of computation. In the same way the speed of convergence of parallel asynchronous method is generally slower compared to the synchronous one. Nevertheless the speed of convergence depend of the frequency of updates of calculations and then of the decomposition of the initial problem in subproblems ; so some decomposition may result in a better convergence rate. But generally there is a reduction of the elapsed time to reach convergence. However the increase of the number of relaxations improve the solution's accuracy and the numerical quality of the computations of the numerical solution.

Furthermore, despite higher number of relaxations, elapsed time of asynchronous scheme is less than the one obtained when synchronous scheme is used.

As a consequence the parallel experiments show that on cloud architecture asynchronous parallel methods present real interest when parallel processing requires a large number of synchronizations. This occurs in particular when the convergence is slow, and especially when the communications between the machines are slow particularly when the connections between these machines is not by direct lines. Indeed, communications on a platform are dependent on the latency of the network. This slowdown is accentuated by the distance between machines

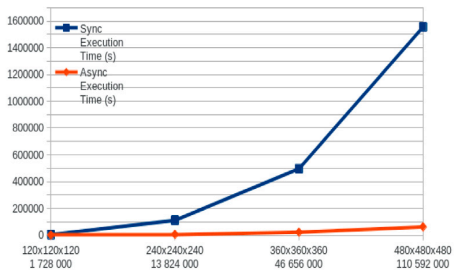


(a) Elapsed time with respect to domain sizes.

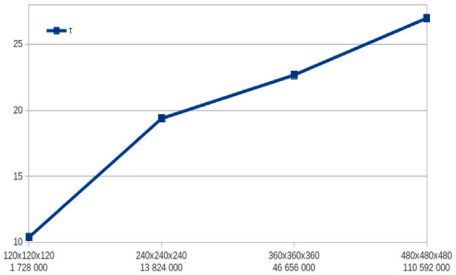


(b) Variation of the value τ with respect to the domain sizes.

Fig. 8. Results for problem (13) on 8 cloud machines for $q = 10$ by subdomains method without overlapping.



(a) Elapsed time with respect to domain sizes.



(b) Variation of the value τ with respect to the domain sizes.

Fig. 9. Results for problem (13) on 8 cloud machines for $q = 10$ with an overlap of 2 meshes.

that are not connected by direct lines. This phenomenon of superiority of the asynchronous parallel methods is thus accentuated when the processors are heterogeneous and distant, which is the case in our simulations on cloud computing. Thus, correlatively, the performance in terms of restitution time of asynchronous parallel methods is much higher than that of synchronous methods. We can thus see here the influence of the architecture of the machines on the performances of the algorithms.

In addition considering the additional software layers, the use of asynchronous parallel algorithms is of interest in a cloud environment compared to the synchronous equivalent insofar as inactivity times are eliminated. Thus asynchronism is an efficient way to deal with communication overhead and load unbalance.

Moreover we obtain also good performances by using parallel asynchronous methods when low granularity is considered (see Table 5 when $n = 120$). Finally using asynchronous methods has a positive impact on environment and allows electrical energy saving.

5. Conclusion

Similar performances could be obtained with other boundary conditions (Neumann conditions, Robin conditions, mixed boundary conditions, etc.). According to our experience and the performances obtained for the model problem (6) and (13) the presented study allows us to hope for the solution of nonlinear problems constituted by :

- other large pseudo-linear single-valued systems with for example application for the simulation of solar oven or situations involved in plasma study,
- large pseudo-linear multi-valued systems with for example application in financial application or in image processing corresponding in this last case to the solution of Hamilton–Jacobi-Bellman equation.

Other possible applications particularly the solution of Navier–Stokes equation corresponding in numerous formulations to the coupling of a diffusion equation to a convection–diffusion equation.

The results obtained during the simulations on the Cloud open up other work perspectives. For example, it would be interesting to use tools that allow us to modify the network speeds and thus decrease or increase the network latency in order to verify its influence on the results of our asynchronous algorithms.

CRediT authorship contribution statement

M.A. Rahhali: Conceptualization, Methodology, Development, Simulation, Writing. **T. Garcia:** Conceptualization, Methodology, Development, Simulation, Writing. **P. Spiteri:** Conceptualization, Methodology, Development, Simulation, Writing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

This study has been made possible with the support of France Grilles- cloud thanks to whom we have access to cloud resources LPNHE which is a UMR Sorbonne University/CNRS-IN2P3. In addition, the previous study was made possible by the use of Grid'5000.

References

[1] Spiteri P. Parallel asynchronous algorithms: a survey. Adv Eng Softw 2020;149. http://dx.doi.org/10.1016/j.advengsoft.2020.102896, Elsevier.
[2] Garcia T, Spiteri P, Ziane-Khodja L, Couturier R. Solution of univalued and multivalued pseudo-linear problems using parallel asynchronous multisplitting methods combined with krylov methods. Adv Eng Softw 2020;153. http://dx.doi.org/10.1016/j.advengsoft.2020.102929, Elsevier.
[3] Garcia T, Spiteri P, Ziane-Khodja L, Couturier R. Coupling parallel asynchronous multisplitting methods with Krylov methods to solve pseudo- linear evolution 3D problems. J Comput Sci 2021;51. http://dx.doi.org/10.1016/j.jocs.2021.101303, Elsevier.
[4] Saad Y. Iterative methods for linear systems of equations: a brief historical journey. In: Brenner Susanne C, Shparlinski Igor E, Shu Chi-Wang, Szyld Daniel, editors. 75 years of mathematics of computation. Contemporary mathematics, vol. 754, 2020, p. 197–214.
[5] Glusa C, Boman EG, Chow E, Rajamanickan S, Szyld D. Scalable asynchronous domain decomposition solvers. SIAM J Sci Comput 2020;42(6):C384–409. http://dx.doi.org/10.1137/19M1291303.
[6] Gbikpi-Benissan G, Magoules F. Asynchronous multiplicative course-space correction. SIAM J Sci Comput 2022;44(3):C237–259. http://dx.doi.org/10.1137/21M1432107.
[7] Nayak P, Cojean T, Anzt H. Evaluating asynchronous Schwarz solvers on GPUs. Int J High Perform Comput 2021;35(3):226–36. http://dx.doi.org/10.1177/1094342020946814.
[8] Yamazaki I, Chow E, Bouteiller A, Dongara J. Performance of asynchronous optimized Schwarz with one-sided communication. Parallel Comput 2019;86:66–81.
[9] Gbikpi-Benissan G, Magoules F. Asynchronous multisplitting-based primal Schur method. J Comput Appl Math 2023;425:115060.
[10] Chaouqui F, Chow E, Szyld D. Asynchronous domain decomposition methods for nonlinear PDES. Electron Trans Numer Anal 2023;58:22–42.
[11] Cai X-C, Sarkis M. A restricted additive Schwarz preconditioner for general sparse linear systems. SIAM J Sci Comput 1999;21:792–7.
[12] Rahhali MA, Garcia T, Spiteri P. Behaviour of asynchronous parallel iterative algorithms on cloud computing type architectures. In: Topping BHV, Iványi P, editors. Proceedings of the eleventh international conference on engineering computational technology. Edinburgh, UK: Civil-Comp Press; 2022, http://dx.doi.org/10.4203/ccc.2.8.2, Online volume: CCC 2, Paper 8.2.
[13] Miellou JC, El Baz D, Spiteri P. A new class of asynchronous iterative algorithms with order interval. Math Comp 1998;67–221:237–55.
[14] Giraud L, Spiteri P. Résolution parallèle de problèmes aux limites non linéaires. M2AN 1991;25:579–606.
[15] Chazan D, Miranker W. Chaotic relaxation. Linear Algebra Appl 1969;2:199–222.
[16] Miellou J-C. Algorithmes de relaxation chaotique à retards. RAIRO Anal Numér 1975;1:55–82.
[17] Miellou J-C. Itérations chaotiques à retards, étude de la convergence dans le cas d'espaces partiellement ordonnés. CRAS Paris 1975;280:233–6.
[18] Bertsekas DP, Tsitsiklis JN. Parallel and distributed iterative algorithms: a selective survey. Automatica 1991;25:3–21.
[19] Bertsekas D, Tsitsiklis J. Parallel and distributed computation. Numerical methods, Prentice Hall Englewood Cliffs N.J.; 1989.
[20] Baudet G. Asynchronous iterative methods for multiprocessors. J Assoc Comput Mach 1978;25:226–44.
[21] Frommer A, Szyld D. On asynchronous iterations. J Comput Appl Math 2000;123:201–16.
[22] Bahi JM, Contassot-Vivier S, Couturier R. Parallel iterative algorithms: From sequential to grid computing. Chapman & Hall/CRC; 2007.
[23] El Baz D, Frommer A, Spitéri P. Asynchronous iterations with flexible communications : contracting operators. J Comput Appl Math 2005;176:91–103, Elsevier.
[24] El Tarazi M. Some convergence results for asynchronous algorithms. Numer Math 1984;39:325–40.
[25] Miellou J-C, Spiteri P. Un critère de convergence pour des méthodes générales de point fixe. M2AN 1985;19:645–69.
[26] Partimbene V, Garcia T, Spiteri P, Marthon P, Ratsifrandrihana L. Asynchronous multi-splitting method for linear and pseudo-linear problems. Adv Eng Softw 2019;133:76–95.
[27] Varga RS. Matrix iterative analysis. Prentice - Hall; 1962.
[28] Ortega J, Rheinboldt W. Iterative solution of nonlinear equations in several variables. New York: Academic Press; 1970.
[29] Evans DJ, Deren W. An asynchronous parallel algorithm for solving a class of nonlinear simultaneous equations. Parallel Comput 1991;17:165–80.
[30] Hoffmann KH, Zou J. Parallel efficiency of domain decomposition methods. Parallel Comput 1993;19:1375–91.
[31] Magoules F, Gbikpi-Benissan G. Distributed convergence detection based on global residual error under asynchronous iterations. IEEE Trans Parallel Distrib Syst 2018;29(4):819–29.
[32] Magoules F, Gbikpi-Benissan G. JACK2: An MPI-based communication library with non-blocking synchronization for asynchronous iterations. Adv Eng Softw 2018;11:116–33.
[33] Gbikpi-Benissan G, Magoules F. Protocol-free asynchronous iterations termination. Adv Eng Softw 2020;146:102827.
[34] Chow E, Frommer A, Szyld DB. Asynchronous Richardson iterations: theory and practice. Numer Algorithms 2021;87:1635–51.
[35] Dijkstra EW, Scholten CS. Termination detection for diffusing computation. Process Lett 1980;11:1–4.
[36] Savari SA, Bertsekas DP. Finite termination of asynchronous iterative algorithms. Parallel Comput 1996;22(1):39–56.
[37] Chau M, Laouar A, Garcia T, Spitéri P. Grid solution of problem with unilateral constraints. Numer Algorithms 2017;75-4:879–908, Springer-Verlag.
[38] Chau M, El Baz D, Guivarch R, Spitéri P. MPI implementation of parallel subdomain methods for linear and nonlinear convection–diffusion problems. J Parallel Distrib Comput 2007;67(5):581–91.
[39] Miellou JC, Spitéri P, El Baz D. A new stopping criterion for linear perturbed asynchronous iterations. J Comput Appl Math 2008;219(2):471–83, Elsevier.
[40] Spitéri P. Finite precision computation for linear fixed point methods of parallel asynchronous iterations. In: Topping BHV, Ivanyi P, editors. Techniques for parallel, distributed and cloud computing in engineering. Saxe-Coburg Publications; 2015, p. 163–96.
[41] Bolze R, Cappello F, Caron E, Daydé M, Desprez F, Jeannot E, et al. Grid'5000: A large scale and highly reconfigurable experimental grid testbed. Int J High Perform Comput Appl 2006;20(4):481–94.
[42] Magoules F, Pan J, Teng F. Cloud computing data-intenbsive computing and scheduling. Chapman & Hall/CRC Numer Anal Sci Comput 2013.

5.3 Article 3

Advances in Engineering Software 153 (2021) 102929

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Research paper

Solution of univalued and multivalued pseudo-linear problems using parallel asynchronous multisplitting methods combined with Krylov methods

T. Garcia [a,*], P. Spiteri [b], L. Ziane-Khodja [c], R. Couturier [d]

[a] University of Toulouse, IRIT-INPT, 2 rue Charles Camichel, B.P. 7122, 31071 Toulouse - Cedex 7, France
[b] University of Toulouse, IRIT-INPT-ENSEEIHT, 2 rue Charles Camichel, B.P. 7122, 31071 Toulouse - Cedex 7, France
[c] ANEO, 122 Avenue du Général Leclerc, 92100 Boulogne-Billancourt, France
[d] University of Bourgogne Franche Comté, CNRS, FEMTO-ST Institute, 19 Av. du Marchal Juin, BP 527, 90016 Belfort cedex, France

ARTICLE INFO

Keywords:
multisplitting methods
synchronous parallel algorithms
asynchronous parallel algorithms
sparse non-linear systems
Krylov methods
pseudo - linear problem
boundary value problem

ABSTRACT

The paper improves a preliminary experimental study on a cluster by adding both theoretical results and experimental tests on a grid platform. These algorithms solve univalued and multivalued pseudo-linear problems using parallel asynchronous multisplitting methods combined with Krylov's methods. This paper also analyses these algorithms using contraction techniques. Two distinct applications, with discretized boundary value problems, are analyzed and simulated. First, a univalued convection-diffusion problem perturbed by an increasing diagonal operator is presented. Then, follows the description of a diffusion problem whose solution is constrained. This situation classically leads to the solution of a multivalued pseudo-linear problem in which the linear part is perturbed by an increasing diagonal multivalued operator. Parallel asynchronous and synchronous algorithms were implemented and tested on a grid platform composed of physically adjacent or geographically distant machines. In addition, the simulation results are detailed and show that the elapsed times obtained for the asynchronous algorithms are significantly less than those obtained for the synchronous algorithms.

1. Introduction and motivation.

The present study is related to the analysis and application of mixed multisplitting methods for solving pseudo - linear stationary problems. These problems are stationary either intrinsically or as a result of the discretization of time evolution problems by implicit or semi-implicit time marching schemes. The considered problems are defined as an affine application $AU - F$ perturbed by an increasing diagonal operator Φ as follows

$$AU - G + \Phi(U) = 0, \tag{1}$$

where in the sequel A has the property of being a large scale M-matrix, G a vector, U is the unknown vector. Note that this type of problem occurs when solving elliptic, parabolic or hyperbolic second order boundaries values problems and that the M-matrix property is well verified after discretization by classical finite differences schemes, finite volumes schemes or finite elements methods provided that, in this last case, the angle condition is verified. In fact two distinct cases are then considered; in the first case corresponding to problem (1) $U \to \Phi(U)$ is a strongly non-linear univalued operator; while in the second case Φ is a multivalued application, for example $\Phi \equiv \partial\Psi$ sub-differential of the indicator function Ψ of the convex $\mathcal{K}$ taking into account constraints on the unknown U to be determined; so in this latter case, the problem to solve is defined by

$$AU - G + \partial\Psi(U) \ni 0, \tag{2}$$

where for example the convex set is defined by

$$\mathcal{K} = \{U \text{ such that } U \geq \varrho\}$$

where ϱ is a given application. Two different kinds of sets can be considered: a convex set with a one sided inequality constraint like the previous one or a set where the solution U must be between two extreme values. In both previous cases each problem will be solved by a specific method; for the univalued problem (1) a local linearization

* Corresponding author.
E-mail addresses: thierry.garcia@irit.fr (T. Garcia), pierre.spiteri@enseeiht.fr (P. Spiteri), lzianekhodja@aneo.fr (L. Ziane-Khodja), raphael.couturier@univ-fcomte.fr (R. Couturier).

https://doi.org/10.1016/j.advengsoft.2020.102929
Received 18 November 2019; Received in revised form 8 October 2020; Accepted 12 October 2020
Available online 15 January 2021

corresponding to the implementation of the iterative Newton method will be carried out; while for the multivalued problem (2), one operates in an iterative alternative way by solving the linear equation without taking into account the constraints we obtain an intermediate value and then projecting this intermediate value on the convex set $\mathcal{K}$ until convergence. In each case, the computation method consists in solving a large sparse linear system. Each system is then associated with a fixed point problem and we intend to solve it by asynchronous parallel iterations [1]-[4]. Taking into account the properties of the matrix A and the operator's monotony property of the perturbed diagonal operator, it is proven in the following that the fixed point application is contractive with respect to a uniform weighted norm [5], which ensures on the one hand the existence and uniqueness of the solution of the algebraic system to be solved and on the other hand the convergence of parallel asynchronous iterations towards the solution of the target problem.

Using the fixed point equation, the parallel iterative asynchronous algorithms are then classically defined in [1] for the solution of large linear algebraic systems and [2] for large algebraic systems. In the first work on asynchronous parallel iterations, delays modeling the asynchronism between the processors were bounded (see [1] and [2]); in fact in [3] G. Baudet has extended the framework of the study to cover cases where delays are no longer bounded, allowing for cases of failure of one or more processors to be taken into account.

In addition, in order to unify the presentation and analysis of algorithm behavior, we consider multisplitting methods that unify the presentation of subdomain methods, either to model subdomain methods without overlap, or to model subdomain methods with overlap such as Schwarz's alternating method. The multisplitting method was introduced by O'Learry and White and also White (see [6]-[7]) in order to give a unified presentation of subdomain methods. Several contributions have been developed by many authors such as J. Arnal, Z.Z. Bai, R. Bru, A. Frommer, V. Migalon, J. Penades. D. Szyld,... etc... in collaboration with several co-authors (see [8]-[9]) for the solution of linear and nonlinear problems. Nevertheless, note that these previous works do not concern problem solution of multivalued problem except for the work of J. Bahi et all [10] developed in an Hilbertian context. These multisplitting methods are then applied to determine the solution to the two previous target problems as well as to the univalued problem (1) and to the multivalued problem (2). The convergence analysis is carried out by contraction techniques with respect to a weighted uniform norm. To solve the model problems, efficient methods are used to find the solution of each of the subproblems handled by each processor. More precisely, a coupling between asynchronous parallel methods and Krylov methods [11] is considered, since each diagonal subproblem obtained by the decomposition of problem (1) or (2) is solved by this last type of algorithm.

As an application, a convection-diffusion problem, perturbed by an increasing diagonal operator [12], is considered, the problem then being solved by a mixed Newton-multisplitting method. On the other hand, there is a diffusion problem whose solution is subjected to constraints of an inequality type, this problem being solved by a relaxation method with projection after each update of the iterative process, each linearized or linear subsystems being solved by the generalized minimal residual method (GMRES). Thus the results of parallel simulation achieved on a grid are presented and discussed in this paper.

The present study completes a preliminary study both theoretically and experimentally. Indeed, theoretically, in relation to [13] this paper is not limited to the single-valued problem but the resolution of multi-valued problems of the form (2) is also considered. On the experimental level, in [13] results on a cluster were presented while in the presented results come from grid, composed of distant and heterogeneous clusters.

In this paper there are two original contributions. The first contribution is in the theoretical part, especially with regard to the resolution of pseudo-linear univalued or multivalued problems by multisplitting methods. From an algorithmic point of view for the resolution of univalued problems the Newton's method is for example used. As previously said multivalued problems are formulated when the solution to be determined is subject to inequality constraints where, from an algorithmic point of view, one must update several components (additive type algorithm) or component by component (multiplicative type algorithm), in both cases without taking into account the constraint and thus obtain an intermediate value which is projected onto the convex set to take into account the inequality constraints to which the solution is subject. The common point between the univalued and multivalued aspects is that the operator which perturbs the affine application $AU - G$ is in both univalued and multivalued cases (See books [14] and [15]) diagonal monotone, which facilitates the analysis of the behavior of the parallel asynchronous algorithms used to solve the target problems. In particular, the forthcoming Proposition 2 recalls a convergence result for any sub-problem decomposition due to the fact that matrix A is an M-matrix (see [16]). This has a consequence for the use of multisplitting methods where, usually, it is assumed that fixed point applications associated with model problem to solve are contractant. However, in the working framework considered in this paper, given the result of Proposition 2, this assumption of contraction of fixed point applications associated with problem to be solved is not necessary since this assumption is of course verified for every decomposition of the problem. In particular, this assumption is applicable for multi-valued problems which are not generally studied in paper concerning multisplitting methods and which are completely solved in this paper (see in particular the result of the forthcoming Corollary 1). Thus, in brief, the contribution at the theoretical level concerns the convergence of multisplitting methods applied to pseudo-linear problems (affine applications perturbed by monotone diagonal operators) in particular for problems whose solution is constrained and classically formulated under a multivalued formulation, since it is well known that the subdifferential of the indicator function is monotone (See book V. Barbu [14]). The second contribution is at the experimental level where, for the solution of pseudo-linear problems, on the one hand we combine a coordination iteration constituted by an asynchronous parallel iteration and on the other hand we solve local sub-problems by a very efficient conjugated gradient method.

The present paper is organized as follows. Section 2 presents the formulation of synchronous and, more generally, asynchronous parallel algorithms and some results allowing to analyze the convergence by contraction techniques are given. In the next section the parallel asynchronous multisplitting algorithms applied to pseudo linear problems are detailed. Section 4 is devoted to present boundary value problems, which, after appropriate discretization leads to solve pseudo linear algebraic systems. Thus the following section is devoted to present the principle of implementation of the studied parallel numerical methods. Section 6 presents the results of parallel experiments achieved on a grid. Finally a conclusion, some future studies and an appendix conclude the paper. This appendix presents the values of the maximum error of the correction obtained in both synchronous and asynchronous mode for the Newton method and these results are commented, both on an experimental and a theoretical level.

2. Parallel asynchronous algorithms

2.1. Discretized model problems

In the sequel, the set of natural integers will denoted by $\mathbb{N}$. Let $M \in \mathbb{N}$. Let us consider the following pseudo - linear problem, issued from the discretization of various boundary value problems

$$AU + \Phi(U) = G, \tag{3}$$

where $A \in \mathcal{L}(\mathbb{R}^M)$ is the discretisation matrix, $U \in \mathbb{R}^M$, $G \in \mathbb{R}^M$, and Φ and G result from the discretisation of a diagonal maximal monotone operator Φ, which means that $i-$ th component of Φ is equal to $\Phi_i(u_i)$

where u_i is the $i-$ th component of U and g the right-hand side of the problem, modelling the source term. Let us assume

A is an M-matrix (4)

Φ is a diagonal increasing operator. (5)

Note that problem (3) is singlevalued.

Nevertheless when the solution U is submitted to some inequality constraints such that

$$U_{min} \leq U \text{ or } U \leq U_{Max} \text{ or } U_{min} \leq U \leq U_{Max}, \tag{6}$$

then, classically, the problem can be expressed by a multivalued formulation (see for example [14], [15]) as follows

$$AU + \partial\Psi(U) - G \ni 0, \tag{7}$$

where $\partial\Psi(U)$ results from the discretization of $\partial\psi(u)$, ψ being the indicator mapping of the convex set defining the constraints and $\partial\psi(u)$ denotes the subdifferential mapping of ψ; classically $\partial\Psi(U)$ satisfies the following property

$\partial\Psi(U)$ is a diagonal increasing or more generally monotone operator. (8)

Let us denote by $\mathcal{E} = \mathbb{R}^M$; note that $\mathcal{E}$ is an Hilbert space. Consider also that the space $\mathcal{E} = \prod_{i=1}^{\alpha} \mathcal{E}_i$ is a finite product of $\alpha > 0$ subspaces denoted $\mathcal{E}_i = \mathbb{R}^{m_i}$, where $\sum_{i=1}^{\alpha} m_i = M$; note that $\mathcal{E}_i$ is also an Hilbert space where $\langle . , . \rangle_i$ denotes the scalar product and $|.|_i$ the associated norm, for all $i \in \{1,\ldots,\alpha\}$. Then for all $U, V \in \mathcal{E}$ let us denote by $\langle U, V \rangle = \sum_{i=1}^{\alpha} \langle U_i , V_i \rangle_i$ the scalar product on $\mathcal{E}$ and $\| . \|$ its associated norm.

We consider now a block-decomposition of problem (3) or (7) in α blocks, which leads for the first one to solve

$$\sum_{j=1}^{\alpha} A_{ij} U_j + \Phi_i(U_i) = G_i, \text{ for all } i \in \{1,\ldots,\alpha\}, \tag{9}$$

and for the second one (7)

$$\sum_{j=1}^{\alpha} A_{i,j}.U_j + \partial\Psi_i(U^i) - G_i \ni 0, \text{ for all } i \in \{1,\ldots,\alpha\}, \tag{10}$$

where $U_i \in \mathcal{E}_i$, $A = (A_{ij})$, and $\partial\Psi_i$ (or Φ_i) result from the associated block decomposition; the subdifferential mapping being multivalued, the previous relation (10) can also be written as follows

$$\sum_{j=1}^{\alpha} A_{i,j}.U_j + w_i(U_i) - G_i = 0, w_i(U_i) \in \partial\Psi_i(U_i), \forall i \in \{1,\ldots,\alpha\}. \tag{11}$$

Then, the following fixed point mapping, at its fixed point $U^\star$ if it exists, is associated to the problem (9) or (11) (in the following, this property is verified to be true):

$$\begin{cases} \text{Find } U^* \in \mathcal{E} \text{ such that} \\ U^* = F(U^*) \end{cases} \tag{12}$$

where $V \mapsto F(V)$ applies from $\mathcal{E}$ to $\mathcal{E}$. The mapping F is implicitly related to problem (3) to solve as follows

$$A_{i,i}.U_i + \Phi_i(U_i) = G_i - \sum_{j \neq i}^{\alpha} A_{ij} V_j, \text{ for all } i \in \{1,\ldots,\alpha\},$$

while for the multi-valued problem (10) the associated fixed point problem is

$$A_{i,i}.U_i + w_i(U_i) = G_i - \sum_{j \neq i}^{\alpha} A_{ij} V_j, w_i(U_i) \in \partial\Psi_i(U_i), \text{ for all } i \in \{1,\ldots,\alpha\},$$

2.2. Parallel fixed point methods

According to the decomposition of $\mathcal{E}$, let us consider the corresponding block decomposition of F and V

$$\begin{array}{rcl} F(V) & = & (F_1(V),\ldots,F_\alpha(V)). \\ V & = & (V_1,\ldots,V_\alpha) \end{array}$$

The parallel asynchronous fixed point iterations $\{U^p\}_{p\in\mathbb{N}}$, which are formally defined by using the following concepts, are then considered.

Definition 1. A steering of block-components of the iterate vector is a sequence $\{s(p)\}, p \in \mathbb{N}$, such that $s(p) \subset \{1,\ldots,\alpha\}, s(p) \neq \emptyset$, for all $p \in N$, where in addition the steering satisfies

$$\{p \in N | i \in s(p)\} \text{ is infinite, for all } i \in \{1,\ldots,\alpha\}. \tag{13}$$

A sequence of delayed iteration numbers $\{\rho(p)\}$ of vectors $\rho(p) = (\rho_1(p), \ldots, \rho_i(p), \ldots, \rho_\alpha(p)) \in \mathbb{N}^\alpha$ is such that for all $p \in \mathbb{N}$ and $i \in \{1,\ldots,\alpha\}$ we have $0 \leq \rho_i(p) \leq p$ and $\rho_i(p) = p$ if $i = s(p)$. Moreover the delayed iteration numbers satisfy

$$\lim_{p\to\infty} \rho_i(p) = +\infty, \text{ for all } i \in \{1,\ldots,\alpha\}. \tag{14}$$

Definition 2. The general class of asynchronous iterative methods is defined recursively as follows. U^0 being given, for all $p \in \mathbb{N}$ and $i \in \{1,\ldots,\alpha\}$:

$$U_i^{p+1} = \begin{cases} F_i\left(\tilde{U}^p\right) \text{ if } i \in s(p), \\ U_i^p \text{ if } i \notin s(p), \end{cases} \tag{15}$$

where

$$\tilde{U}^p = \left\{U_1^{\rho_1(p)}, \ldots, U_i^{\rho_i(p)}, \ldots, U_\alpha^{\rho_\alpha(p)}\right\} \in \mathcal{E}, \text{ if } p \geq 1 \tag{16}$$

$s(p)$ and $\rho(p)$ being defined according to Definition 1.

Remark 1. Asynchronous iterations defined recursively by (15)-(16) are general iterative methods whereby iterations are carried out in parallel by up to α processors without any order nor synchronization. The main feature of this class of iterative methods is to allow flexible communications between the processors and to minimize the weight of synchronisations between the parallel processes. In such computational method, to make the most of computing power by eliminating idle times due to blocking expectations, synchronizations are not necessarily required which avoids having to wait for the communications of the values computed by the other processors; thus each process performs its own calculations using available data calculated by other processors. Then each processor advances its own calculations at its own pace, with communications taking place in no pre-established order. The choice of the relaxed components is performed using a selection of several components of the strategy $s(p)$ at each step p of the calculation; this strategy is in fact a non-empty subset of the set $s(p) \subset \{1, 2, \ldots, \alpha\}$ which well models parallelism between the processes, since each element of the strategy is not limited to a single element. In addition, theoretically, each block component of the iterate vector is continuously updated; but in practice the parallel iterative method is ended by a stopping criterion, which in the asynchronous context, is very hard to perform. For a fixed process, the asynchronism between updates is modelled by the introduction of delayed components calculated by other processors to take into account the necessary coupling between the various processes. Thus, in a typical update of the i-th block-component of the iterate vector at iteration $p+1$, all the values $\tilde{U}_j^p$ of the block-components of the iterate vector are taken equal to $\tilde{U}_j^p$ corresponding to the value computed in the last update of the j-th block-component; so $\tilde{U}_j^p \equiv U_j^{\rho_j(p)}$ models the

nondeterministic behavior of the iterative scheme. Practically, one will choose update corresponding to the last available value of each component. It should also be noted that in the asynchronous parallel iteration model the information exchanged between processors is no longer time-limited and can have unlimited communication delays, which makes it possible to take into account possible temporary failures of multiprocessors, clusters or grids. Note that parallel synchronous iterations correspond to the particular case where $\rho_j(p) = p$ for all $p \in \mathbb{N}$; moreover, in this case, if for all $p \in \mathbb{N}$, $s(p) = \{1, \ldots, \alpha\}$ (respectively $s(p) = (1 + p \bmod \alpha)$) then algorithm (15)-(16) model the sequential block-Jacobi (respectively Gauss-Seidel) method.

For simplicity's sake the study of the convergence is limited in this paper to the use of contraction techniques. The convergence of parallel asynchronous iterations for the solution of the multivalued problem (7) is considered; the case of singlevalued problem (3) being treated similarly in a straightforward way. First, a contraction result associated with the point decomposition is established; then it can be concluded that the fixed point F is contracting for any decomposition into α large blocks.

Proposition 1. *Consider the problem (7) (or (3)) and assume that the matrix A is an M-matrix and that the affine application AU −G is perturbed by a diagonal monotone maximal operator eventually multivalued satisfying (8) (or (5) in the singlevalued case). Then, we can associate to problem (7) defined in the space $\mathcal{E}$ a contracting fixed point mapping denoted F_P associated to the point decomposition, and there exists one and only one fixed point $U^\star$ also unique solution of the discretized problem (7) (or (3)). Furthermore the parallel asynchronous method associated to the fixed point mapping F_P converges to $U^\star$ whatever be the initial guess U^0.*

Proof. Indeed, for generality's sake, problem (7) should be considered; let us decompose this problem into M subproblems; let us write the i^{th} equation verified on the one hand for the exact solution and on the other hand for a current value of the $(p+1)^{th}$ relaxation component; subtracting member to member, multiplying each of the M relations by $(u_i^\star - u_i^{p+1})$ and taking the absolute value on both side leads to

$$\begin{aligned} 0 &\le a_{i,i}\left(u_i^\star - u_i^{p+1}\right)\left(u_i^\star - u_i^{p+1}\right) + \left(w_i^\star - w_i^{p+1}\right)\left(u_i^\star - u_i^{p+1}\right) \\ &\le \sum_{j\neq i}\left|a_{i,j}\left(u_j^\star - v_j\right)\left(u_i^\star - u_i^{p+1}\right)\right|, \ \forall i \in \{1, \ldots, M\}, \end{aligned}$$

where $w_i^\star \in \partial\Psi_i(u_i^\star)$ and $w_i^{p+1} \in \partial\Psi_i(u_i^{p+1})$ and v_j are the available current values produced by the other processors according to the algorithm (15)-(16); then, since the subdifferential mapping is monotone the following inequality is true $(w_i^\star - w_i^{p+1}).(u_i^\star - u_i^{p+1}) \ge 0$ and the left hand-side of the previous inequality can be minored by $a_{i,i}\left|u_i^\star - u_i^{p+1}\right|^2$, which leads to

$$0 \le a_{i,i}|u_i^\star - u_i^{p+1}|^2 \le \sum_{j\neq i}\left|a_{i,j}\left(u_j^\star - v_j\right)\left(u_i^\star - u_i^{p+1}\right)\right|, \ \forall i \in \{1, \ldots, M\};$$

as in the sum of the right member appears the absolute values the right-hand side of the previous inequality can be majored using a classical and straightforward inequality and since A is a M-matrix, its diagonal entries are negative or null and one can thus write $|a_{i,j}| = -a_{i,j}$ which leads finally to

$$|u_i^\star - u_i^{p+1}| \le \sum_{j\neq i} -\frac{a_{i,j}}{a_{i,i}}\left|u_j^\star - v_j\right|, \ \forall i \in \{1, \ldots, M\}. \tag{17}$$

Note that the matrix J with diagonal entries that are null and off-diagonal entries equal to $-\frac{a_{i,j}}{a_{i,i}}$ is the Jacobi matrix of the matrix A. Since the matrix A is an M-matrix, then J is a nonnegative matrix and all eigenvalues of J have a modulus less than one. Let us denote by ν the spectral radius of J and by Θ the associated eigenvector; then $0 \le \nu < 1$. Classically by the Perron-Frobenius theorem, all the components of Θ are strictly positive and the following inequality $J\Theta \le \nu\Theta$ is valid (see [10]). Then, in a straightforward way, the result is

$$\| U^\star - U^{p+1}\|_{\nu,\Theta} \le \nu \| U^\star - V \|_{\nu,\Theta}, \forall V \in \mathcal{E}, 0 \le \nu < 1, \tag{18}$$

where $\| V \|_{\nu,\Theta}$, is a uniform weighted norm defined as follows

$$\| V \|_{\nu,\Theta} = \max_{i=1,\ldots,M}\left(\frac{|v_i|}{\Theta_i}\right); \tag{19}$$

thus, (18) shows that the fixed point mapping F_P associated to the point-decomposition is a contraction and we obtain a result of existence and uniqueness of both the fixed point $U^\star$ of F_P and of the solution of problem (7), which achieves the proof. In the case of problem (3), since Φ is also monotone, the proof follows accordingly.□ In fact, in parallel computation, the user does not access as many processors as components. Practically, the algebraic system to solve is split into α contiguous blocks, $\alpha < M$, corresponding to a coarser decomposition, the communications consisting in exchanging only the necessary values of the components computed by the neighbouring processors. To solve the model problem (7) or (3) let us consider a parallel asynchronous block relaxation algorithm, associated to the $\alpha -$ block decomposition of the discretized problem. Then several adjacent block components of the discretization matrix, and of the iterate vectors are processed accordingly by each processor. Such an implementation leads to a more multiplicative behavior of the considered subdomain methods. Furthermore, using a result of [17]-[16], if the subdomain decomposition is a point decomposition, the following result is obtained

Proposition 2. *Consider problem (7) or (3) and assume that the assumptions of Proposition 1 hold, particularly that the matrix A is an M-matrix and that the affine algebraic system is perturbed by a monotonous diagonal operator, singlevalued or multivalued. Consider any block decomposition in $\alpha < M$ blocks. Then, for any block decomposition, the associated fixed point mapping F is contractive; then there exists one and only one fixed point $U^\star$ also unique solution to the discretized problem (7) (or (3)). Furthermore, for every subdomain decomposition, the parallel asynchronous methods converge to $U^\star$ whatever the initial guess U^0 is.*

Proof. The proof is not very difficult but essentially technical. It consists of writing the equations considered in the proof of Proposition 1, then grouping them in α parts; in this context of block decomposition, using the concept of regular decomposition of M-matrix (see [18]), the result is established. So the fixed point mapping is contractive and the parallel asynchronous methods associated to each block decomposition converge for every decomposition. For more details the reader is referred to [16].□

Remark 2. Moreover the considered pseudolinear problems can also be solved by subdomain methods with overlapping, like the Schwarz's alternating method; in this case, due to the augmentation process of the Schwarz's method, the pseudo-linear problems (1) are respectively

written as follows

$$\overline{A}U - \overline{G} + \overline{\Phi}\left(U\right) = 0. \tag{20}$$

and for problem (2) in the following way

$$\overline{A}U - \overline{G} + \overline{\Psi}\left(U\right) \ni 0. \tag{21}$$

Using a result of D.J. Evans and Van Deren (see [19]), if A is an M-matrix, then $\overline{A}$ is also an M-matrix. Moreover, by applying the augmentation process, the diagonal operator $\overline{\Phi}$ or $\overline{\Psi}$ are still diagonal monotone operator. So this is in the framework of the study of [17]-[16] and the results of this last papers concerning the convergence of parallel synchronous and asynchronous subdomain methods with or without overlapping can still be applied. In fact the multisplitting method presented below allows to give an unified presentation of subdomain methods with or without overlapping.

3. Parallel asynchronous multisplitting method

3.1. Analysis of the behavior of the method in a general context

Let us consider now the solution of problem (7) (or problem (3)) by the parallel asynchronous multisplitting method when A satisfies assumption (4). Let us also consider the n following regular splittings (see [18]) of matrix A

$$A = \mathcal{M}^l - \mathcal{N}^l, \quad l = 1, ..., n, n \in \mathbf{N}, \tag{22}$$

where in what follows $\mathcal{M}^l$ are block diagonal, i.e. $\mathcal{M}^l = diag(\mathcal{M}_i^l),\ i = 1, \ldots, \alpha, l = 1, ..., n$ and moreover $(\mathcal{M}^l)^{-1} \geq 0$ and $\mathcal{N}^l \geq 0$; note that classically $\mathcal{M}^l$ are M-matrices for all l.

Let $F^l : \mathcal{D}(F^l) \to \mathcal{D}(F^l),\ \mathcal{D}(F^l) \subset \mathcal{E},\ l = 1, ..., n,$ be n fixed point mappings associated with problem (7) (or problem (3)) and defined by:

$$F^l(V) = U,\ l = 1, \ldots, n,\ U, V \in \mathcal{D}\left(F^l\right) \subset \mathcal{E}, \tag{23}$$

such that

$$\mathcal{M}^l U = \mathcal{N}^l V + \overline{G};$$

then, for all $l = 1, ..., n$, the mapping F^l associated with problem (7) (or problem (3)) are implicitly defined by

$$G + \mathcal{N}^l V \in \mathcal{M}^l U + \partial\Psi(U) \Leftrightarrow U = F^l(V),\ \forall V \in \mathcal{E} = \mathcal{R}^M. \tag{24}$$

$$\left(G + \mathcal{N}^l V = \mathcal{M}^l U + \Phi(U) \Leftrightarrow U = F^l(V),\ \forall V \in \mathcal{E} = \mathcal{R}^M\right). \tag{25}$$

Assume that the space $E = \mathcal{E}^n$ is normed by the uniform weighted norm, defined in a similar way to (19) by

$$\| V \|_{\nu,\overline{\vartheta}} = \max_{l=1,\ldots,n} \left(\max_{i=1,\ldots,\alpha} \left(\frac{|V_i^l|}{\left(\overline{\vartheta}_l\right)_i} \right) \right),\ \overline{\vartheta}_l > 0, \tag{26}$$

where here $|V_i|$ denotes any Hilbertian norm of V_i in $\mathcal{E}_i, i = 1, \ldots, \alpha$, subspace of $\mathcal{E}$; assume that for $l = 1, ..., n,\ F^l$ is contractive with respect to U^*, its fixed point, with constant $0 \leq \nu_l < 1, l = 1, ..., n$ so that the following inequality is verified

$$\| F^l(V) - U^* \|_{\nu_l,\overline{\vartheta}_l} \leq \nu_l . \| V - U^* \|_{\nu_l,\overline{\vartheta}_l},\ \text{for } l = 1, ..., n. \tag{27}$$

Remark 3. Note that for problems (7) and (3), according to the result of Proposition 2, due to the fact that A is an M-matrix and that in both cases the affine mapping $AU - G$ is perturbed by a diagonal monotone operator, then the fixed point mapping $F^l, l = 1, \ldots, n$ are contractive.

A formal multisplitting associated with problem (7) (or problem (3)) is defined by the collection of fixed point problems (see [10])

$$U^* = F^l(U^*),\ l = 1, ..., n,\ U^* \in \mathcal{E}. \tag{28}$$

Let us now consider space $E = (\mathcal{E})^m$ like a finite product space of the n spaces $\mathcal{E}$ defined by

$$E = \prod_{l=1}^{n} \mathcal{E}_l,$$

where $\mathcal{E}_l = \mathcal{E}$ and consider the following n − block-decomposition of $\tilde{U} \in E$, defined by

$$\tilde{U} = \left\{U^1, \ldots, U^l, \ldots, U^n\right\} \in \prod_{l=1}^{n} \mathcal{E}_l,$$

where $U^1, \ldots, U^l, \ldots, U^n$ denote n vectors of $\mathcal{E}$.

Definition 3. The extended fixed point mapping $\mathcal{F} : E \to E$ associated with the formal multisplitting is given as follows

$$\mathcal{F}\left(\tilde{V}\right) = \tilde{U},\ \text{such that } U^l = F^l\left(\sum_{k=1}^{n} W_{lk} V^k\right), l = 1, ..., n,$$

where W_{lk} are nonnegative diagonal weighting matrices satisfying for all $l \in \{1, \ldots, n\}$

$$\sum_{k=1}^{n} W_{lk} = I_l,$$

I_l being the identity matrix in $L(E_l)$. Since $F^l(\mathcal{D}(F^l)) \subset \mathcal{D}(F^l)$, then obviously $\mathcal{F}(\mathcal{D}(\mathcal{F})) \subset \mathcal{D}(\mathcal{F})$ where $\mathcal{D}(\mathcal{F}) = \prod_{l=1}^{n} \mathcal{D}(F^l)$.

Note that considerable saving in computational work may be possible by using such a method, since a component of V^k needs not be computed if the corresponding diagonal entry of the weighting matrices is zero; then, in parallel computing, the role of such weighting matrices may be regarded as determining the distribution of the computational work of the individual processors.

Let the following block-decomposition of the mapping $\mathcal{F}$

$$\mathcal{F}\left(\tilde{V}\right) = \left\{\mathcal{F}^1\left(\tilde{V}\right), \ldots, \mathcal{F}^l\left(\tilde{V}\right), \ldots, \mathcal{F}^n\left(\tilde{V}\right)\right\} \in \prod_{l=1}^{n} \mathcal{E}_l.$$

Note that for a particular choice of the weighting matrices W_{lk}, we can obtain various iterative methods and particularly on the one hand a subdomain method without overlapping and on the other hand the classical Schwarz alternating method (see [10]). According to [10] the block - Jacobi method corresponds to the following choice of $\mathcal{M}^l$

$$\mathcal{M}^l = diag\left(I_1, \ldots, I_{l-1}, A_{l,l}, I_{l+1}, \ldots, I_n\right), \tag{29}$$

and to the choice of $W_{lk} \equiv \overline{W}_l$ given by

$$\overline{W}_l = diag(0, \ldots, 0, I_l, 0, \ldots, 0), \tag{30}$$

which in other words means that the entries of the weighting matrices are equal to one or to zero.

For the additive Schwarz alternating method more than one processor computes updated values of the same component, and the matrices W_l have positive entries smaller than one. The reader is referred to reference [6] and to other various references for other choices of weighting diagonal matrices and splittings for the definition of various multisplitting methods.

Let us recall now a result of [10]

Proposition 3. *Let us denote by $\tilde{U}^* = \{U^*, \ldots, U^*\}$ where U^* is the solution of (7) (or problem (3)); then if assumption (27) is verified, $\mathcal{F}$ is $\| . \|_{\nu,\overline{\vartheta}}$ contractive with respect to $\tilde{U}^*$, where $\| . \|_{\nu,\overline{\vartheta}}$ is defined by (26) the associated constant of contraction being*

$$\nu = \max_{1\le l\le n}(\nu_l) < 1.$$

Then, using the result of Proposition 3, the following result is obtained

Corollary 1. *Consider the solution of problem (7) (or problem (3)); then if assumptions (4) - (5) (or (4) - (8)) are verified, in particular if matrix A is an M-matrix, then the formal multisplitting $\mathcal{F}$ is contractive with respect to $\tilde{U}^*$ and any sequential, parallel synchronous or asynchronous multisplitting method starting from $\tilde{U}^0 \in \mathcal{D}(\mathcal{F})$ converge to the solution of problem (7) (or problem (3)).*

Proof. Indeed, if assumption (4) is verified, and if the other assumptions of Proposition 2 concerning the monotony of the disturbing diagonal operator are satisfied, then each fixed point mapping $\widehat{F}^l$ is contractive, for $l = 1, \ldots, n$ and then the proof is complete.□

Then, for $l = 1, \ldots, n$, and for $i = 1, \ldots, \alpha$, for problem (7) such an asynchronous multisplitting method can be given by

$$\begin{cases} \mathcal{M}_i^l U_i^{l,p+1} + \omega_i\left(U_i^{l,p+1}\right) = \left(\mathcal{N}^l\left(\sum_{k=1}^{n} W_{lk} V_i^{k,\rho_k(p)}\right) + G\right)_i & , \text{ if } i \in s(p) \\ U_i^{l,p+1} = U_i^{l,p} & \text{if } \ i \notin s(p) \end{cases}$$

where $\omega_i(U_i^{l,p+1}) \in \partial\Psi_i(U_i^{l,p+1})$ and accordingly for problem (3).

3.2. Application to pseudolinear problem

3.2.1. The case of singlevalued problem and Newton method

For the solution of problem (3), the block-diagonal matrix $C(W)$, derived from Newton's method is introduced, with diagonal blocks $C_i(W)$ given by

$$C_i(W) = A_{ii} + \Phi_i^{'}(W).$$

Obviously, matrix $C_i(W)$ is an M-matrix since matrix A_{ii} is an M-matrix and Φ_i is increasing and then its derivative $\Phi_i^{'}$ is positive.

Let U^0 given and U^k is the k-th iteration of the following algorithm:

$$U^{p+1} = U^p - C^{-1}(U^{\rho^{'}(p)}).\left(AU^k + \Phi\left(U^k\right) - G\right), p = 0, 1, \ldots, \tag{31}$$

where $0 \le \rho^{'}(p) \le p$.

Proposition 4. *Assume that assumptions (4) and (5) hold. Then, the sequence $\{U^p\}$ defined by (31) converges to $U^\star$ solution of problem (3).*

Proof. Obvious since C is an M-matrix.□

3.2.2. The case of multivalued problem

Consider now the solution of the multivalued semi-linear problem (7) when assumptions (4) and (8) are verified. For such a problem, by using the same kind of proof than the previous ones, and mainly due to the monotony of the disturbing multivalued diagonal operator, a result similar to the one presented in Proposition 4 is obtained. Then it can stated that

Proposition 5. *Assume that the assumptions (4) and (8) hold. Then, the sequence $\{U^p\}$ obtained by using the parallel asynchronous multisplitting method converges to $U^\star$ solution of problem (7).*

Proof. The convergence of the considered method is analyzed by combining the monotony of the diagonal operator with the main property of the discretization matrix. Indeed, since the subdifferential of the indicator function is monotone and diagonal, the multivalued formulation allows to prove the convergence of the parallel synchronous and asynchrounous iterations in a similar way than the one used for the proof of Proposition 1.□

Remark 4. In the continuous case, for the solution of the convection - diffusion problem with constraints on the solution, the formulation of the problem is achieved by the perturbation of the continuous convection - diffusion operator by the multivalued increasing diagonal operator $\partial\psi(u)$. This multivalued formulation takes some interest only from a theoretical point of view. For the solution of the discretized problem (7) it is necessary to take into account the constraints and so to project the relaxed updates computed by the previous iterative algorithm on a convex set which defines such constraints as follows:

1. first to compute an intermediate value of a component of the iterate vector by solving for example, one equation of the algebraic linear system without taking into account the constraint,
2. and then to project this intermediate value of the component on the convex set until convergence of the iterative process; more precisely, if the constraint is satisfied, the intermediate value of the component is not changed and if the constraint is saturated, for example, if a component of U is less to the corresponding component of ϱ, then the result of the projection on the convex set is such that the final update considered consists in setting the value of the component to ϱ.

4. Application to nonlinear partial differential equations

There are several kinds of partial differential equations which, after discretization, lead to the solution of pseudo-linear algebraic systems like (3) and (7). In the sequel two distinct kinds of pseudo-linear algebraic systems are considered.

4.1. Newton - multisplitting method for unconstrained boundary values problems

In the sequel Ω will denote an open domain included in $\mathbb{R}^3$, $\partial\Omega$ the boundary of Ω, g a sommable square function and $u \to \phi(u)$ a diagonal monotone increasing, convex and continuously differentiable nonlinear operator. So let us consider the following nonlinear convection diffusion problem

$$\begin{cases} -\nu\Delta u + a\dfrac{\partial u}{\partial x} + b\dfrac{\partial u}{\partial y} + c\dfrac{\partial u}{\partial z} + du + \phi(u) = g, \ \text{everywhere in } \Omega, \\ u = 0, \ \text{everywhere in } \partial\Omega, \end{cases} \tag{32}$$

where ν, a, b, c, d are some constant coefficients, $\nu > 0, d \ge 0$.

For example, problem (32) occurs in plasma physics or to model solar ovens [12]; such a problem arises from the implicit temporal discretization of a parabolic problem that appears in such applications modeled as follows

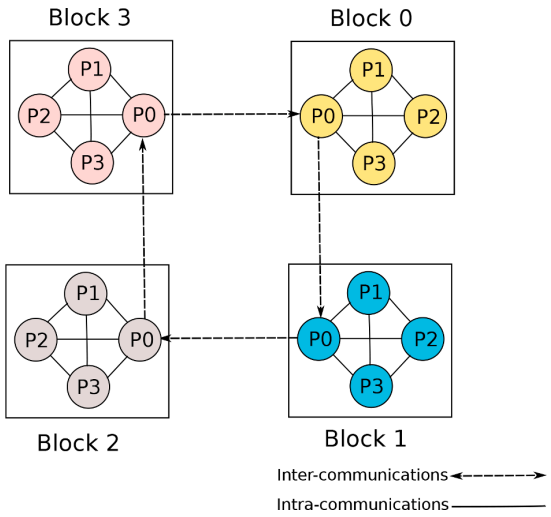


Fig. 1. Example of an interconnection of four blocks.

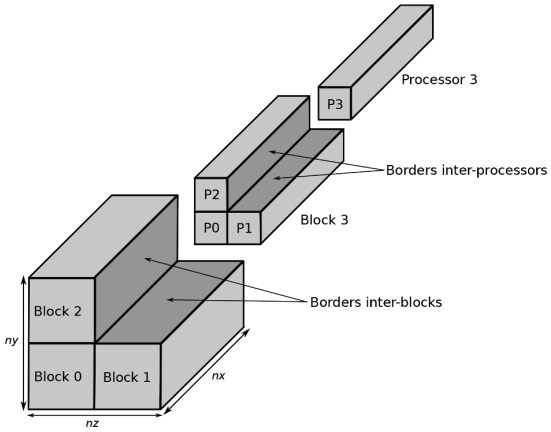


Fig. 2. Example of a decomposition of a 3D problem among blocks of processors.

$$\begin{cases} \dfrac{\partial u(t,x)}{\partial t} - \nu\Delta u(t,x) + Q^t\nabla u + e^{ru} = g(t,x), \text{ everywhere in } [0,T]\times\Omega, b>0, \\ u(t,x) = 0, \text{ everywhere in } [0,T]\times\partial\Omega, \\ u(0,x) = u_0(x), \text{ everywhere in } \Omega, \end{cases} \tag{33}$$

where $T>0$, $u_0 : \Omega\to\mathbb{R}$ is the initial condition, Q is a vector with components (a,b,c).

After temporal discretization the stationary problem associated to the implicit time marching scheme, is defined as follows

$$\begin{cases} -\nu\Delta u + Q^t\nabla u + \dfrac{u}{\delta_\tau} + e^{ru} = g, \text{ everywhere in } \Omega\subset R^3, \\ u = 0, \text{ everywhere in } \partial\Omega, \end{cases} \tag{34}$$

where δ_τ is the time step arising in the implicit scheme.

After spatial discretization we have to solve a pseudo-linear algebraic systems of kind (1) by combining the Newton method with the parallel asynchronous multisplitting method. Note that, by choosing appropriate finite difference approximation, particularly for the convection term where according to the sign of the coefficients a,b,c, forward or backward schemes are considered so that the discretization matrix is an M-matrix; moreover since $\phi(u)$ is a diagonal monotone increasing nonlinear operator, assumptions (4) and (5) are well verified. Thus the previous parallel synchronous or asynchronous multisplitting studied method for the parallel solution of this problem can be applied.

4.2. Multisplitting method for constrained boundary values problems

Lastly note also that in the previous mathematical models, the solution u can be subject to some constraints, for example $u_{min} < u$ or $u < u_{max}$ or $u_{min} < u < u_{max}$; in such situations, the associated boundary value problem is classically formulated as the following multivalued problem (see [14])

$$\begin{cases} -\Delta u + \phi(u) + \partial\psi(u) - g \ni 0, \text{ everywhere in } \Omega, \\ u = 0, \text{ everywhere in } \partial\Omega, \end{cases} \tag{35}$$

where $\partial\psi(u)$ is the subdifferential of the indicator function $\psi(u)$ of the convex set $\mathcal{K}$ defining the constraints. It can be noted that this multivalued formulation expresses the fact that the problem is subject to some constraints. This approach is very convenient and is mainly applied to analyze the behavior of iterative algorithms used for the numerical solution of problems like (35) since the continuous diffusion operator is perturbed by the multivalued increasing operator $\partial\psi(u)$. From a practical point of view, once the problem is discretized, after each update of a block component, a projection on the convex set $\mathcal{K}$ is processed.

5. Implementation of the multisplitting method

The algorithms of the Newton-multisplitting method and the multisplitting method with projection have been implemented, both used to solve univalued and multivalued pseudo-linear problems (3) and (7) respectively.

In this paper the 3D nonlinear problems (34) presented in Section 4 are solved on a computing platform composed of n blocks, each of them composed of q processors, physically adjacent or geographically distant. Fig. 1 illustrates an example of such a computing platform composed of four blocks, each of them composed of four processors. In each block a master processor is designed and the inter-communications between blocks are performed via the masters.

A 3D nonlinear problem is split into n blocks and each block (which represents a splitting in the previous mathematical description) is in turn split into q portions, as it is presented in the example of Fig. 2. In this case each block of data is assigned to a block of processors and each portion to a processor.

5.1. Case of Newton-multisplitting method

In this section the multisplitting algorithm to solve nonlinear stationary convection-diffusion problems (34) presented in sub-section 4.1 is described. It should be noted that no difference is made between processors or cores.

For each splitting $l, l = 1,\ldots,n$, starting with an initial guess $U^{l,(0)}$ the following equations should be solved

$$AU^l + \Phi\left(U^l\right) - G = 0,$$

by the Newton method; then, globally, at step i of the Newton method it follows that

$$\left(A + \frac{\partial\Phi\left(U^{l,(i)}\right)}{\partial U}\right)\delta U^{l,(i)} = G - AU^{l,(i)} - \Phi\left(U^{l,(i)}\right)$$

Algorithm 1: Parallel nonlinear multi-splitting method performed on a block of processors

Output: Solution $U^{l,(i+1)}$
1 Set initial solution: $U^{l,(0)} = 1.0$
2 **while** $\underline{\|\delta\mathbf{U}^{(i+1)}\|_2 \geq \varepsilon_{Newton}}$ **do**
3 Compute the right-hand side of Newton $\hat{G}(U^{l,(i)})$: Formula (39)
4 Update local sub-matrix $\hat{C}_{l,l}$: Formula (36)
5 **while** $\|\delta U^{l,(i)} - \delta U^{l,(i-1)}\|_\infty \geq \varepsilon_{Multisplitting}$ **do**
6 Compute local right-hand side: B_l Formula (38)
7 Parallel GMRES to solve $\delta U^{l,(i)}$: (Formula 37)
8 Exchange local shared values of $\delta U^{l,(i)}$ with neighbor blocks
9 **end**
10 Compute the local solution on block l: $U^{l,(i+1)} = U^{l,(i)} + \delta\ U^{l,(i)}$
11 Compute the global correction $\delta\mathbf{U}^{(i+1)}$
12 **end**

Algorithm 1. Parallel nonlinear multi-splitting method performed on a block of processors

and then

$$U^{l,(i+1)} = U^{l,(i)} + \delta U^{l,(i)},$$

until convergence of the iterative method.

Let $\widehat{C}(U^{l,(i)}) = A + \frac{\partial\Phi(U^{l,(i)})}{\partial U}$ and according to the choice of the weighting matrices $W_{l,k}$ let us consider a block decomposition of the matrix $\widehat{C}(U^{l,(i)})$ such that

$$\widehat{C}_{l,l}\left(U^{l,(i)}\right) = \left(A + \frac{\partial\Phi\left(U^{l,(i)}\right)}{\partial U}\right)_{l,l} \tag{36}$$

denotes the block diagonal of the matrix $\widehat{C}(U^{l,(i)})$, and since the operator $U \to \Phi(U)$ is diagonal increasing, then the Jacobian matrix of Φ, given by $\frac{\partial\Phi(U^{l,(i)})}{\partial U}$, is a positive diagonal matrix; for the same reason, due to the fact that $\Phi(U)$ is diagonal the off-diagonal blocks of $\widehat{C}(U^{l,(i)})$ are reduced to the blocks $A_{l,k}$ of the matrix A. Consequently $\widehat{C}(U^{l,(i)})$ is an M-matrix.

So the implementation of the Newton method requires the solution of the following linear system

$$\widehat{C}\left(U^{l}\right)\delta U^{l} = \widehat{G}\left(U^{l}\right),$$

which will therefore be solved by a multisplitting method; since the matrix $\widehat{C}(U^{l})$ is an M-matrix, the multisplitting method will converge according to the results stated in subsection 3.1.

For the solution of problem (1) by the Newton method and considering for example the block Jacobi method obtained by choosing $\mathcal{M}^l$ and $\tilde{W}_l$ given by (29)-(30), the implemented multisplitting method associated to the iteration number i of the Newton method leads to solve iteratively in parallel for $l = 1, \ldots, n$, the algebraic sub-systems

$$\widehat{C}_{l,l}\left(U^{l,(i)}\right)\delta U_l^{l,(i)} = B_l,\ l = 1, \ldots, n, \tag{37}$$

where B_l is given by

$$B_l = \widehat{G}_l\left(U^{l,(i)}\right) - \sum_{k \neq l} A_{l,k}\delta U_k^{k,(j(k))},\ l = 1, \ldots, n \tag{38}$$

and $\widehat{G}(U^{l,(i)})$ is the right hand side resulting from the Newton process, i.e.

$$\widehat{G}\left(U^{l,(i)}\right) = G - \Phi\left(U^{l,(i)}\right) - AU^{l,(i)}, \tag{39}$$

and the values of the components of the local vectors $\delta U_k^{k,(j(k))}$ come from the computation performed on splitting number $k, k \neq l$, and performed by other processors by using the iterate number $j(k)$ of the iterative method.

Then, in other words, each sub-system (37) is solved independently by a block (or splitting) of processors and communications are required to update the right-hand side of each sub-system. The local vectors δU_k updated according to (38) by each block represent the data dependencies between the different blocks. In the multisplitting method there are two level of iterations; outer and inner parallel iterations. In fact, since the matrices $\widehat{C}_{l,l}$ are also sparse, it is highly recommended to solve the subsystems (37) by an iterative method. It is well known that iterative methods scale well. In our implementation a Krylov method was chosen to solve each block (37). It should be noted that the outer-iteration fits well within the general formulation of parallel asynchronous iterations described in sub-section 2.2, since the inter-block communications can be either synchronous or asynchronous.

The focus was put on solving 3D nonlinear single-valued systems of equations which can be formulated as shown in (1). Algorithm 1 presents the main key-points of the Newton-multisplitting method. The algorithm uses the Newton method to linearize the nonlinear system to solve (lines from 2 to 12). Then it applies the parallel multisplitting method to each linear system issued from the linearization (lines from 5 to 9), such that each system is associated to n splittings as shown in (37).

In the previous description, it should be noticed that all variables indexed with l are local to a block. First, a loop on the Newton iteration is computed. When starting a new Newton iteration, the right-hand side of the Newton process is updated. And the local sub-matrix $\widehat{C}_{l,l}$ is updated. Then the second loop based on the Multisplitting iteration is computed. In the multisplitting loop, first the local right-hand side B_l is involved, taking into account the neighbor's values computed by the other processors. Then each sub-system is solved in parallel by using the well-known Krylov method GMRES [20] (line 7 in Algorithm 1). GMRES iterations represent the inner-iteration of the multisplitting method. It should be noticed that the iterations and the communications performed within the GMRES solver are synchronous inside a block of processors. At the end of the Multisplitting iteration, each local solution of the

Algorithm 2: Parallel projected multi-splitting method

Output: Solution U^l

1 Compute local sub-matrix $A_{l,l}$
2 norm=1
3 **while** $norm \geq \varepsilon_{global}$ **do**
4 Compute local right-hand side: $B_l = G_l - \sum_{k \neq l} A_{l,k} U_k$
5 Parallel GMRES to solve $\bar{U}^l$ from $A_{l,l}\bar{U}_l = B_l$
6 $\bar{U}_l$=max($\bar{U}_l$,constraint)
7 $norm_l = |(U_l - \bar{U}_l)|$;
8 $norm = max_l(norm_l)$
9 $U_l = \bar{U}_l$
10 Exchange local shared values of U^l with neighbor blocks
11 **end**

Algorithm 2. Parallel projected multi-splitting method

sub-system is exchanged to all neighbor blocks (see line 8). On line 10, the local solution on block l is updated, then on line 11, the global correction $\delta \mathbf{U}^{(i+1)}$ is performed.

The global convergence of the synchronous Newton-multisplitting method is detected when the global value of $\delta \mathbf{U}^{(i+1)}$ is stabilized corresponding to the following stopping test

$$\| \delta \mathbf{U}^{(i+1)} \|_2 \geq \varepsilon_{Newton} \tag{40}$$

where ε_{Newton} is the tolerance threshold for the computation of the Newton method.

The parallel algorithms described in this section are implemented in two ways: synchronous and asynchronous. For the parallel *synchronous* algorithms, we use *MPI_Send*() and *MPI_Recv*() routines to perform the communications. In contrast, for the parallel asynchronous algorithms, we use the MPI non-blocking communications *MPI_Isend*(), *MPI_Irecv*() and *MPI_Test*(). The reduction operations are implemented using *MPI_Allreduce*().

In the asynchronous version, the global convergence is detected when all blocks have locally converged. The convergence detection implemented in [21] was used. In fact, as is shown in Fig. 1, a virtual unidirectional ring network is implemented between masters of the blocks of processors. At the beginning of the resolution, a Boolean token is set to *False* by the master of block 0. During the computation, the Boolean token circulates around the virtual ring network until the global convergence. Locally at the end of the resolution of a sub-system, each master of block i updates the value of the token by evaluating the following logical expression:

```
token = token && is_locally_converged_i()
```

where *is_locally_converged*$_i$() returns *True* if the local convergence is achieved or *False* otherwise. Then the token is sent to the master of block $i+1$ if $i < n-1$ or to the master of block 0 otherwise. The global convergence is detected when the master of block 0 receives from the master of block $n-1$ a token set to *True*. To this effect, the master of block 0 broadcasts a message to all masters of other blocks of processors in order to stop the computation.

5.2. Case of multisplitting method with projection

This sub-section shows the implementation of the multisplitting method for solving a boundary value problem of the kind (35) where the solution is subject to inequality constraints. The principle of implementation is the same as the one considered in the previous sub-section. In fact, the implementation is even simpler since we have to solve the linear system associated to the sub-problem without taking into account

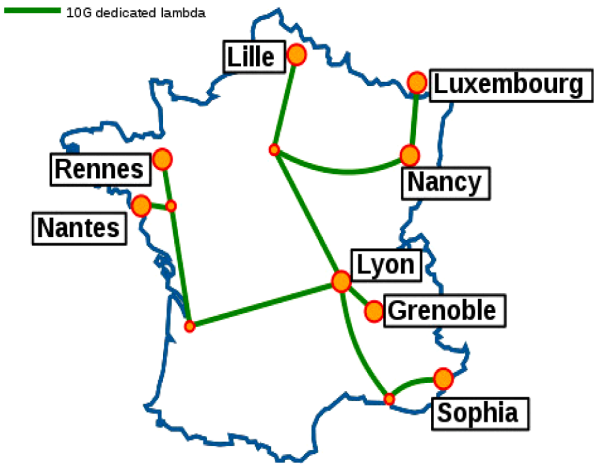


Fig. 3. Grid'5000 architecture.

Table 1
Size of the algebraic systems.

Size of the algebraic systems	
$240^3 = 13824000$	$420^3 = 74088000$
$300^3 = 27000000$	$480^3 = 110592000$
$360^3 = 46656000$	

the constraints and then project the intermediate solution $\overline{U}_l$ on the convex set defining the constraints as follows.

$$A_{l,l}.\overline{U}_l = G_l - \sum_{k \neq l} A_{l,k} U_k \tag{41}$$

and then $\overline{U}_l$ is obtained by a projection

$$\overline{U} = Proj\left(\overline{U}_l\right) \tag{42}$$

It should also be noted that the formulation (35) is not used directly to code the algorithms, because this last formulation is a formal formulation of the problem to be solved. In particular, it is not necessary to discretize the sub-differential operator of the indicator function of the convex set defining the constraints on the solution. It is therefore

Table 2
Characteristics of machines on each site.

Site	Cluster	Processors type	GHz	CPU	cores/CPU	RAM size	more information
Nantes	ecotype	Intel Xeon E5-2630L v4	1.8	2	10	128 GB	[23]
Rennes	parapide	Intel Xeon X5570	2.9	2	4	25 GB	[24]
Rennes	paravance	Intel Xeon E5-2630 v3	2.4	2	8	128 GB	[25]
Sophia	suno	Intel Xeon E5520	2.27	2	4	32 GB	[26]

Table 3
Simulations on GRID'5000.

Experimentations					Newton - multisplitting			Multisplitting with projection		
Clusters			Cores	Machines	Tables		Figures	Tables		Figures
					Sync	Async		Sync	Async	
paravance			600	about 38	4	5	4			
ecotype	parapide		600	about 40	6	7	5	14	15	9
ecotype	parapide	suno	600	about 48	8	9	6	16	17	10
ecotype	paravance	suno	1000	about 64	10	11	7	18	19	11

sufficient to discretize by classical methods, on the one hand the partial derivative operator taking into account the boundary conditions, and on the other hand the second member of the partial derivative equation.

Algorithm 2 presents the main key-points of our synchronous or asynchronous multisplitting method in order to solve the discrete problem subject to constraints. All variables are local to a block, except *norm* which represents the global norm defined as the max of all the local norms. As Algorithm 2 has many similarities to Algorithm 1 except that there is no Newton iteration, all the previous explanations and remarks are valid.

6. Parallel experiments

In paper [13], experiments were achieved on a single cluster. In present version, larger experiments are made on a grid. The reason is that making parallel computation on a distributed grid with geographically distant sites is more difficult and challenging than using a supercomputer. In addition, when one have to solve very large algebraic systems, impossible to solve on an unique cluster with few processors, grid environment is very interesting to use.

So, parallel experiments have been performed on Grid'5000 a French grid platform which is a large-scale and flexible testbed for experiment-driven research in all areas of computer science, with a focus on parallel and distributed computing [22]. Actually this environment provides access to a large amount of resources: 15358 cores, 840 compute-nodes grouped in 37 homogeneous clusters on 8 sites, connected with a dedicated 10-Gbps backbone network and featuring various technologies for example GPU, different CPU type, ssd, 10G and 25G Ethernet.

Our parallel simulations for the solution of the different problems have been performed on various distant clusters connected by highspeed communication of the Grid 5000 platform. At that time, on Grid'5000, the speed of communication was about a Gigabit Ethernet network for local machine on each cluster and links between different sites range from 10-Gbps.

Fig. 3 illustrates the sites of the Grid'5000 architecture.

Let us consider problems (34) and (35), solved respectively by parallel asynchronous Newton-multisplitting method and parallel asynchronous multisplitting algorithm with projection. For the parallel simulations performed on distant clusters of the Grid'5000 platform, several sizes of the algebraic systems to solve have been considered. Indeed the problems are defined on a cube $\Omega \equiv [0,1[^3$ and we consider in the achieved simulations on each axis 240, 300, 360, 420 and/or 480 discretization points. So the size of the algebraic systems to solve are respectively 240^3, 300^3, 360^3, 420^3 and/or 480^3 (see Table 1).

The parallel experiments were carried out by using the cluster named "parapide" or "paravance" of Rennes, the cluster named "ecotype" of Nantes and the cluster named "suno" of Sophia. Table 2 gives the characteristics of each machine used.

So, readers can find below the results for 240^3, 300^3, 360^3, 420^3 and/or 480^3 size of algebraic systems discretized by a finite difference scheme, on a grid composed of two or three distant clusters. The code written in C language is parallelized with facilities provided by Open MPI v2. The MPI machine file alternates dedicated computers on clusters to reach the desired number of cores to perform the experiments.

In these simulations, the number of blocks was fixed to $n = 2$ since it has been observed that this enables one to obtain the best performances. So in all the results, $n \times q$ configuration were used. Moreover, the cores of each block q are randomly deployed on the distant clusters used.

In order to measure the efficiency of the asynchronous methods

Table 4
Domain size, iterations, elapsed and communication time on cluster (parapide) with synchronous parallel algorithm.

Synchronous results				
Size	Iterations		Elapsed time (sec)	Communication time (sec)
	Multi (Nwt)	GMRES		
240^3	4229 (3)	21 145	251	63
300^3	6387 (3)	31 935	426	103
360^3	8827 (3)	44 135	687	157
420^3	11785 (3)	58 925	1 053	224
480^3	15025 (3)	75 125	1 657	363

Table 5
Domain size, iterations, elapsed and communication time on cluster (parapide) with asynchronous parallel algorithm.

Asynchronous results					
Size	Iterations		Elapsed time (sec)	Communication time (sec)	τ
	Multi (Nwt)	GMRES			
240^3	3191 (17)	15 957	150	38	1.67
300^3	4125 (5)	20 625	237	55	1.79
360^3	6006 (26)	30 030	385	86	1.78
420^3	7641 (39)	38 207	677	137	1.56
480^3	9745 (33)	48 725	1 150	205	1.44

compared to the synchronous ones, the values of τ which is the ratio of the synchronous and asynchronous computation time is used.

For each simulation, the tolerance thresholds are used: for GMRES 10^{-12}, for the Multisplitting 10^{-7} and for the Newton method 10^{-4}. The values for problem (32) of a, b, c and d are respectively $a = 0.1, b = 0.2, c = 0.3$ and $d = 0$.

6.1. Parallel simulations for synchronous and asynchronous Newton - multisplitting algorithm and multisplitting algorithm with projection.

Several simulations have been carried out for each problem. The following Table 3 resumes cluster and grid tests on Grid'5000 platform.

The results of each grid simulations for synchronous and asynchronous versions are summarized in Tables 4–11 for the Newton - multisplitting algorithm and in Tables 14–19 for the multisplitting algorithm with projection. In these Tables, the values of domain size, average iteration number by core to reach convergence followed by GMRES number of iterations, elapsed and communication times are detailed. For the Newton - multisplitting algorithm, the number of iterations necessary to reach convergence for linear systems derived from the Newton method is mentioned and the Newton iterations is indicated in parentheses. In addition, in Tables which contain asynchronous results, the values of τ for each problem is given.

The results of the synchronous and asynchronous elapsed times and communications times with respect to the different domain sizes are presented on Figs. 4–7 for the first problem and on Figs. 9–11 for the problem with projection.

Table 6
Domain size, iterations, elapsed time on grid (ecotype, parapide) with synchronous and asynchronous parallel algorithm.

Synchronous results			
Size	Iterations		Elapsed time (sec)
	Multi (Nwt)	GMRES	Total
240^3	4229 (3)	21 145	4 213
360^3	8827 (3)	44 135	16 999
480^3	15025 (3)	75 125	29 425

Table 7
Domain size, iterations, elapsed time on grid (ecotype, parapide) with asynchronous parallel algorithm.

Asynchronous results				
Size	Iterations		Elapsed time (sec)	
	Multi (Nwt)	GMRES	Total	τ
240^3	3 197 (10)	15 987	2 779	1.52
360^3	6 036 (39)	30 180	6 959	2.44
480^3	9 001 (53)	45 005	10 689	2.75

Table 8
Domain size, iterations, elapsed and communication time on grid (ecotype, parapide, suno) with synchronous parallel algorithm.

Synchronous results				
Size	Iterations		Elapsed time (sec)	Communication time (sec)
	Multi (Nwt)	GMRES		
240^3	4229 (3)	21 145	4 368	1 099
300^3	6387 (3)	31 935	6 713	1 711
360^3	8827 (3)	44 135	14 332	3 217
420^3	11785 (3)	58 925	18 331	4 833
480^3	15025 (3)	75 125	24 855	6 933

Table 9
Domain size, iterations, elapsed and communication time on grid (ecotype, parapide, suno) with asynchronous parallel algorithm.

Asynchronous results					
Size	Iterations		Elapsed time (sec)	Communication time (sec)	τ
	Multi (Nwt)	GMRES			
240^3	3272 (8)	16 360	3 025	800	1.44
300^3	5809 (9)	29 045	4 553	1 150	1.47
360^3	5375 (9)	26 875	7 828	1 760	1.83
420^3	6836 (19)	34 180	9 495	2 541	1.93
480^3	8147 (39)	40 735	12 287	3 446	2.02

Table 10
Domain size, iterations, elapsed and communication time on grid (ecotype, paravance, suno) with synchronous parallel algorithm.

Synchronous results				
Size	Iterations		Elapsed time (sec)	Communication time (sec)
	Multi (Nwt)	GMRES		
240^3	2626 (20)	13 130	2 998	652
300^3	3756 (14)	18 780	4 355	944
360^3	4964 (63)	24 820	6 816	1 372
420^3	6710 (62)	33 550	9 395	1 846
480^3	8939 (7)	44 695	14 813	2 849

Table 11
Domain size, iterations, elapsed and communication time on grid (ecotype, paravance, suno) with asynchronous parallel algorithm.

Asynchronous results					
Size	Iterations		Elapsed time (sec)	Communication time (sec)	τ
	Multi (Nwt)	GMRES			
240^3	2276 (393)	11 382	2 565	558	1.17
300^3	3158 (415)	15 792	3 505	773	1.24
360^3	4324 (843)	21 622	5 036	1 085	1.35
420^3	5603 (1235)	28 017	6 787	1 421	1.38
480^3	6313 (1064)	31 567	9 374	1 792	1.58

Table 12
Domain size, iterations, elapsed and communication time on grid (ecotype, paravance, suno) with synchronous parallel algorithm.

Synchronous results				
Size	Iterations		Elapsed time (sec)	Communication time (sec)
	Multi (Nwt)	GMRES		
240^3	3186 (7)	15 930	3 483	782
300^3	4586 (5)	22 930	5 035	1 132
360^3	6412 (10)	32 065	7 100	1 599
420^3	8400 (10)	42 000	10 261	2 284
480^3	10491 (8)	52 455	16 812	3 508

6.1.1. Newton - multisplitting algorithm

Results with 64 cores on single cluster

As previously said, in paper [13], experiments for Newton -

Table 13
Domain size, iterations, elapsed and communication time on grid (ecotype, paravance, suno) with asynchronous parallel algorithm.

Asynchronous results					
Size	Iterations		Elapsed time (sec)	Communication time (sec)	τ
	Multi (Nwt)	GMRES			
240^3	15664 (17)	78 322	2 437	676	1.43
300^3	21210 (24)	106 051	3 431	953	1.47
360^3	25563 (71)	132 817	4 613	1 267	1.54
420^3	32612 (448)	163 061	6 604	1 681	1.55
480^3	42609 (597)	213 047	9 864	2 430	1.70

Table 14
Domain size, iterations, elapsed time on grid (ecotype, parapide) with synchronous parallel algorithm.

Synchronous results			
Size	Iterations		Elapsed time (sec)
	Multi	GMRES	Total
240^3	7 481	37 405	7 496
360^3	15 814	79 070	30 484
480^3	28 538	142 690	55 725

Table 15
Domain size, iterations, elapsed time on grid (ecotype, parapide) with asynchronous parallel algorithm.

Asynchronous results				
Size	Iterations		Elapsed time (sec)	
	Multi	GMRES	Total	τ
240^3	6 501	32 507	5 823	1.29
360^3	12 844	64 222	15 452	1.97
480^3	21 221	106 105	26 642	2.09

Table 16
Domain size, iterations, elapsed and communication time on grid (ecotype, parapide, suno) with synchronous parallel algorithm.

Synchronous results				
Size	Iterations		Elapsed time (sec)	Communication time (sec)
	Multi	GMRES		
240^3	74 81	37 405	7 759	1 984
300^3	11 249	56 245	11 734	2 994
360^3	15 814	79 070	25 808	5 828
420^3	21 030	105 150	33 727	7 898
480^3	28 538	142 690	45 997	12 936

multisplitting algorithm were achieved on a single cluster where the dimension of the 3D problem was discretized in 150 elements, with a size of algebraic system to solve of 150^3 using up to 64 cores. Experiments have been achieved on the mesocentre of the University of Franche-ComtĘ and machines was composed of Xeon(R) CPU E5-2640 v3 @ 2.60GHz processors.

In the present experiments, we have performed additional tests on single cluster constituting one cluster of Grid'5000.

Results with 600 cores on single cluster with $n = 2$ *and* $q = 300$

Table 17
Domain size, iterations, elapsed and communication time on grid (ecotype, parapide, suno) with asynchronous parallel algorithm.

Asynchronous results					
Size	Iterations		Elapsed time (sec)	Communication time (sec)	τ
	Multi	GMRES			
240^3	6 483	32 415	6 185	1 583	1.25
300^3	9 429	47 145	9 080	2 339	1.29
360^3	11 241	6 205	17 114	3 639	1.51
420^3	13 430	67 150	19 903	5 123	1.69
480^3	16 733	83 665	25 437	7 026	1.81

Table 18
Domain size, iterations, elapsed and communication time on grid (ecotype, paravance, suno) with synchronous parallel algorithm.

Synchronous results				
Size	Iterations		Elapsed time (sec)	Communication time (sec)
	Multi	GMRES		
240^3	5 838	29 190	6 707	1 451
300^3	8 551	42 755	9 820	2 151
360^3	11 898	59 490	16 298	3 272
420^3	15 296	76 480	21 368	3 847
480^3	19 448	97 240	33 272	6 335

Table 19
Domain size, iterations, elapsed and communication time on grid (ecotype, paravance, suno) with asynchronous parallel algorithm.

Asynchronous results					
Size	Iterations		Elapsed time (sec)	Communication time (sec)	τ
	Multi	GMRES			
240^3	5 286	26 432	5 654	1 261	1.19
300^3	7 680	38 400	8 227	1 858	1.19
360^3	9 330	46 715	12 537	2 066	1.30
420^3	13 427	67 135	15 056	3 306	1.42
480^3	18 749	93 347	19 943	5 465	1.67

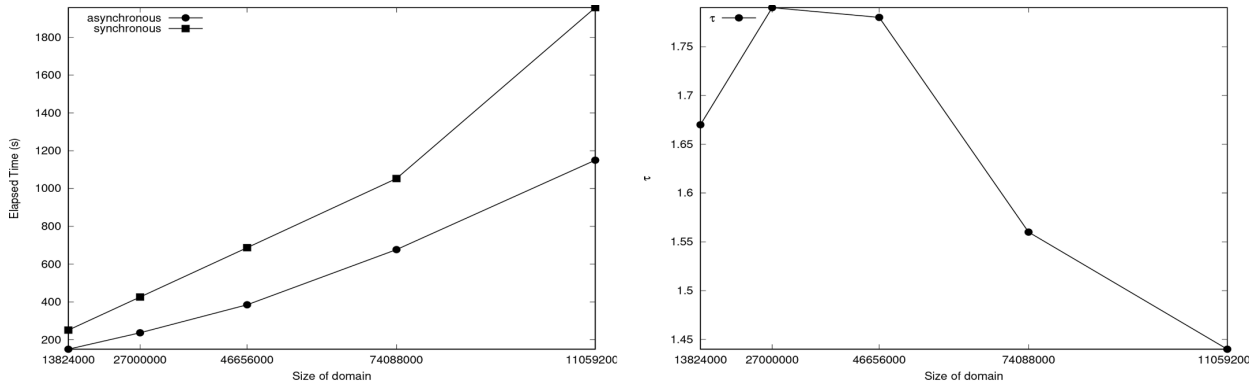
Results with 600 cores on 2 clusters with $n = 2$ *and* $q = 300$
Results with 600 cores on 3 clusters with $n = 2$ *and* $q = 300$
Results with 1000 cores on 3 clusters with $n = 2$ *and* $q = 500$
Results with 1000 cores on 3 clusters with $n = 4$ *and* $q = 250$

6.1.2. Multisplitting problem with projection
Results with 600 cores on 2 clusters with $n = 2$ *and* $q = 300$
Results with 600 cores on 3 clusters with $n = 2$ *and* $q = 300$
Results with 1000 cores on 3 clusters with $n = 2$ *and* $q = 500$

6.2. Experiments analysis.

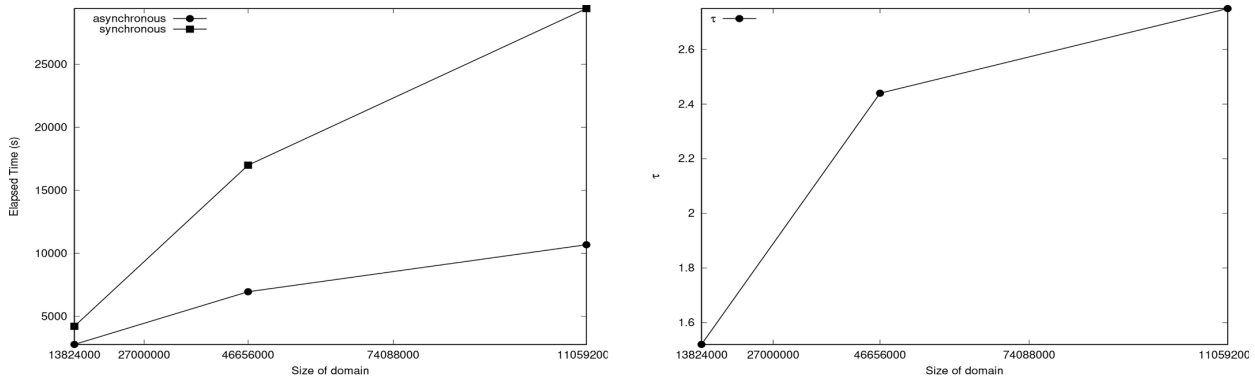
Results on a single cluster

For the experiment on a single cluster, the asynchronous elapsed time is better than the synchronous one but the ratio τ shows a degradation when the size increases (see Tables 4, 5 and Fig. 4). The communications and the execution times are very fast between the cores of a



(a) Elapsed time with respect to domain sizes. (b) Variation of the value τ with respect to the domain sizes.

Fig. 4. Newton - multisplitting algorithm : Results with 600 cores on 1 cluster



(a) Elapsed time with respect to domain sizes. (b) Variation of the value τ with respect to the domain sizes.

Fig. 5. Newton - multisplitting algorithm : Results with 600 cores on 2 clusters

cluster which removes the benefit of the asynchronous compared to the synchronous. Indeed, the different simulations show that the best performances for asynchronous algorithms are in the context of grid use with remote and heterogeneous configurations. In view of these results, no further experiments were carried out on a single cluster because of degraded communications and limited simulation size. So, the simulations focused on grid experiments for the two problems considered.

Results on grid

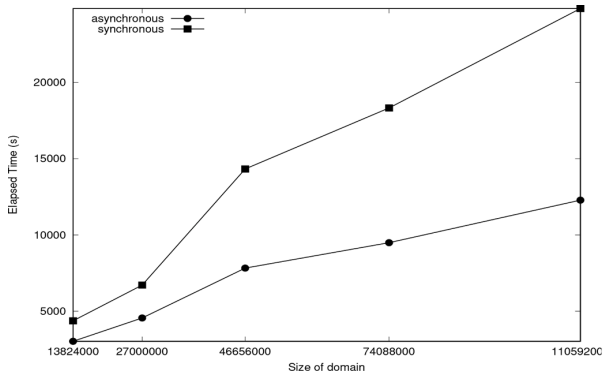
The experiments were carried out on grids with machines distributed over several remote sites using 600 and 1000 cores on 2 and 3 clusters.

In order to verify our assertion concerning the choice of $n = 2$, a simulation on 3 clusters was carried out on 1000 cores and $n = 4$ blocks (see Tables 12, 13 and Fig. 8).

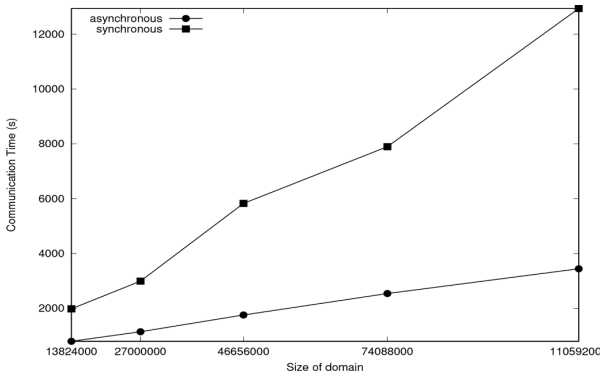
This simulation shows good results for the asynchronous algorithm. Concerning the synchronous algorithm, $n = 4$ blocks increases the execution time and the communication time compared to a $n = 2$ blocks. For the asynchronous, this configuration increases the number of iterations to reach convergence and the communication time. In view of these results, the simulations were carried out by taking $n = 2$ blocks (see Table 3).

In our experiments, the elapsed times obtained for the asynchronous method is largely better than the ones obtained for the synchronous method; this is essentially due to the latency of the network which implies costly communications and also due, as previously said, to the choice of a number of blocks equal to 2. It can be observed that the parameter τ, the ratio of the synchronous and asynchronous computation time which allows to measure suitably the efficiency of the asynchronous methods compared to the synchronous ones, shows us good performances.

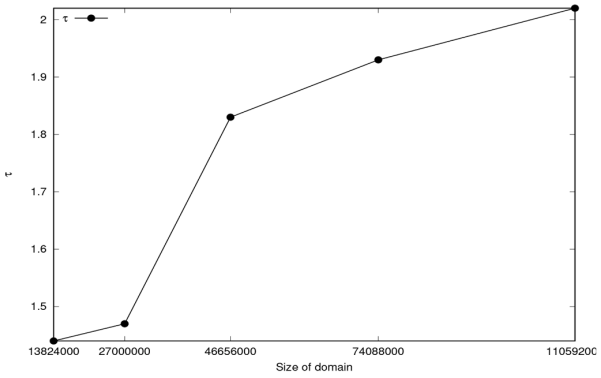
In Table 6 and Table 7, parallel simulations for the Newton - multisplitting algorithm on grid composed of two clusters with 600 cores show that the asynchronous mode is more efficient than the synchronous one. The parallel asynchronous simulation times are clearly inferior to those obtained in the synchronous mode and it can be observed that the values of τ vary between 1.52 and 2.75. So for the size of 480^3 the asynchronous simulation works almost three times faster than the



(a) Elapsed time with respect to domain sizes.



(b) Communication time with respect to domain sizes.



(c) Variation of the value τ with respect to the domain sizes.

Fig. 6. Newton - multisplitting algorithm : Results with 600 cores on 3 clusters

synchronous version. In view of the results indicated in Table 6 and Table 7, it appears that the Newton method converges quickly. It is therefore the resolution of linear systems resulting from the Newton method that requires a large amount of computation (see Tables 6, 7 and Fig. 5).

Similarly, in Table 14 and Table 15, the parallel simulations for the multisplitting algorithm with projection achieved on grid also show that the asynchronous mode is, once again, more efficient than the synchronous mode and the values of τ vary between 1.29 and 2.09. In this case, the size of the algebraic system to solve is equal to 480^3 and the asynchronous simulation works almost twice as fast as the synchronous version (see Tables 14, 15 and Fig. 9).

The results obtained for 600 cores on 3 clusters in Tables 8 and 9 show that the execution times increase due to the increase of time communication. In the end, the asynchronous algorithm is more efficient than the synchronous algorithm and the values of τ vary between 1.44 and 2.02, i.e. for 480^3 the asynchronous simulation is twice as fast as the synchronous one. In Tables 16, 17, the simulations also show a degradation of the results due to the increased communications when increasing the number of the remote site but confirms the good performance of the asynchronous versus synchronous results, the value of τ varying between 1.25 and 1.81. (see Tables 8, 9, 16, 17 and Figs. 6, 10).

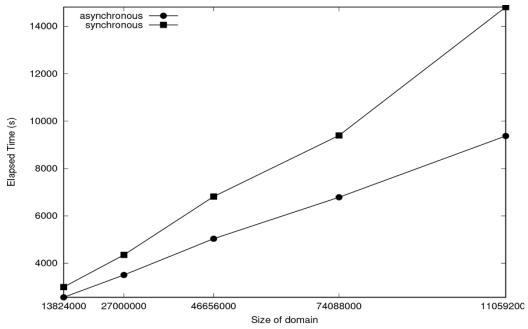
Finally, for the results obtained for 1000 cores on 3 clusters in Tables 10, 11, 18 and 19, the execution times are lower due to the finer cutting date on cores. The asynchronous method is still interesting compared to the synchronous method with values of τ between 1.17 and 1.58 for the first problem and 1.19 to 1.67 for the second problem with projection (Tables 10, 11, 18, 19 and Figs. 7, 11)

It should be noted that the asynchronous version of the calculation code requires fewer iterations than the synchronous one. In fact, this reduction of iterations is related to the process in which the global problem is divided into sub-problems, particularly for the resolution of the linear systems derived from the Newton linearization. With the splitting used, the asynchronous version has a more marked multiplicative behavior than the synchronous version. Thus, in the asynchronous mode, the updates of interaction values performed by processors lead to an acceleration convergence.

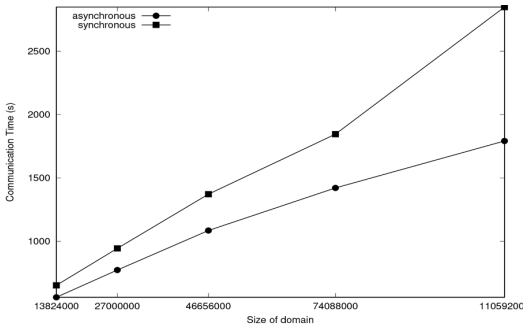
Thus, for the two non-linear problems tested, there is a difference in behaviour in sequential mode between the Gauss-Seidel method and the Jacobi method to solve a Poisson equation; in this case the Gauss-Seidel method converges twice as fast as the Jacobi method. Thus, when relaxation methods are parallelized, this type of algorithm behavior tends to be preserved for the resolution of pseudo-linear problems with or without constraints.

7. Conclusion

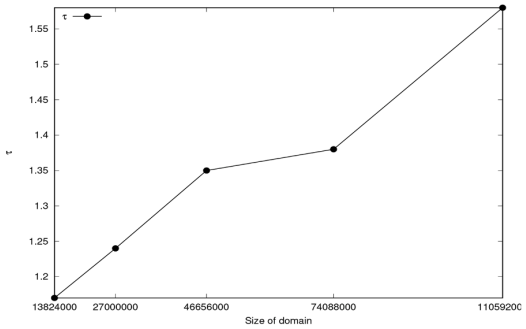
In the present study we have presented a formulation of multi-



(a) Elapsed time with respect to domain sizes.

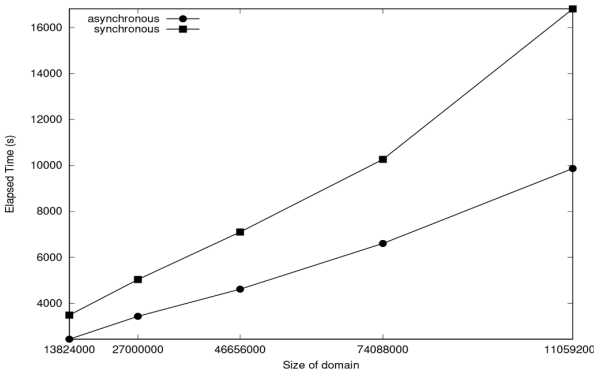


(b) Communication time with respect to domain sizes.

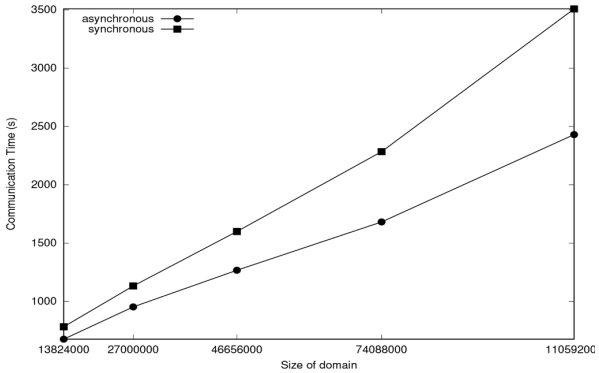


(c) Variation of the value τ with respect to the domain sizes.

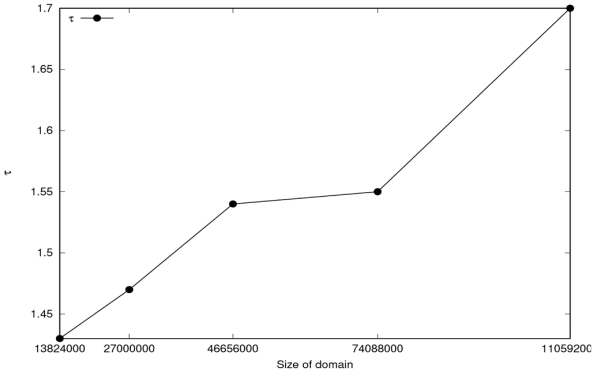
Fig. 7. Newton - multisplitting algorithm : Results with 1000 cores on 3 clusters



(a) Elapsed time with respect to domain sizes.

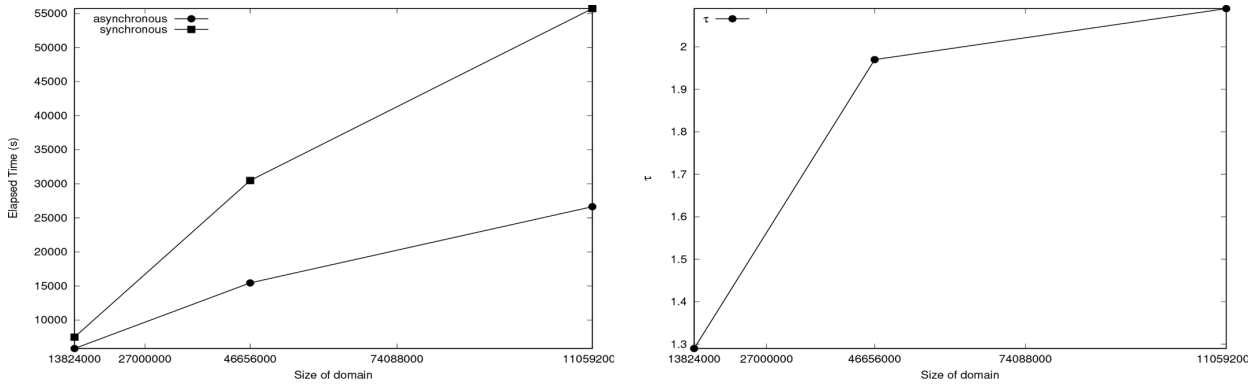


(b) Communication time with respect to domain sizes.



(c) Variation of the value τ with respect to the domain sizes.

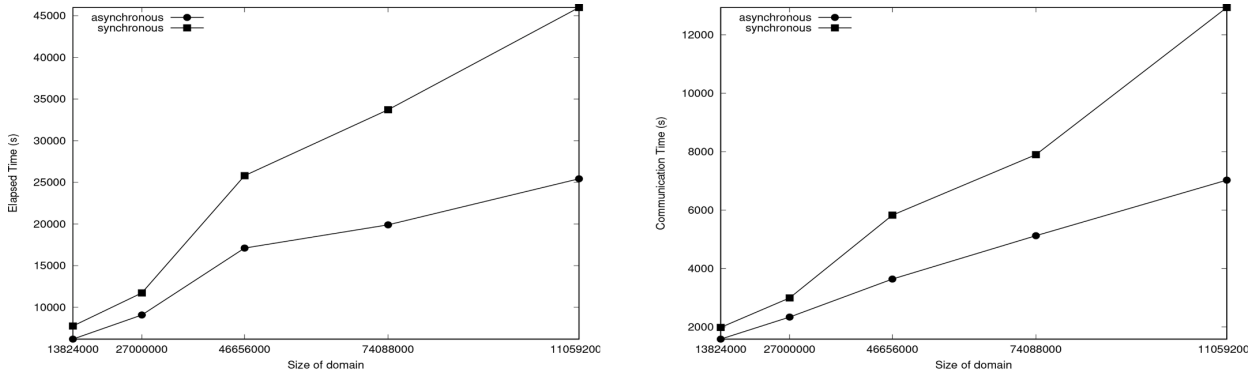
Fig. 8. Newton - multisplitting algorithm : Results with 1000 cores on 3 clusters



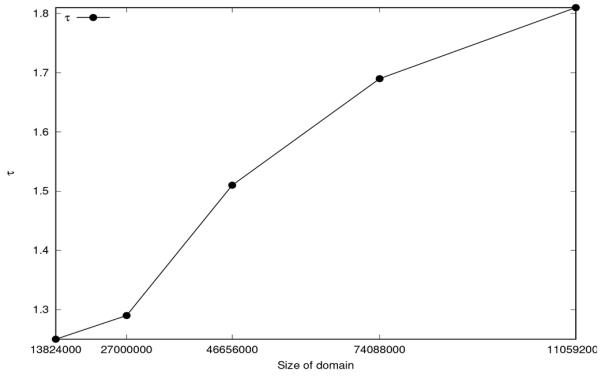
(a) Elapsed time with respect to domain sizes. (b) Variation of the value τ with respect to the domain sizes.

Fig. 9. Multisplitting problem with projection : Results with 600 cores on 2 clusters

splitting algorithms to find the solution of diagonal subproblems. Such a calculation method has been used for the solution of univalued or multivalued pseudo-linear stationary problems and implemented in a computing platform composed of n blocks, each of them composed of q processors, physically adjacent or geographically distant. The performance of asynchronous parallel iterative methods are better than the synchronous ones. In the following appendix, we will present an experimental and some theoretical arguments concerning the behavior

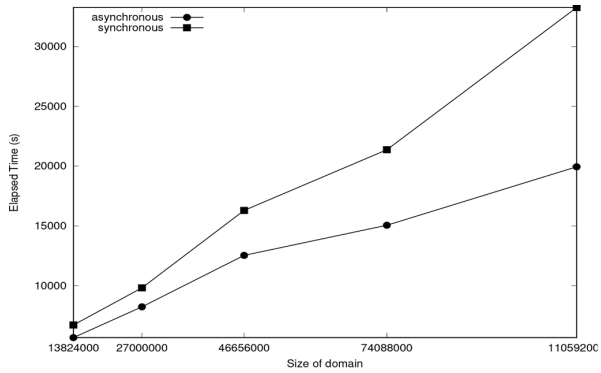


(a) Elapsed time with respect to domain sizes. (b) Communication time with respect to domain sizes.

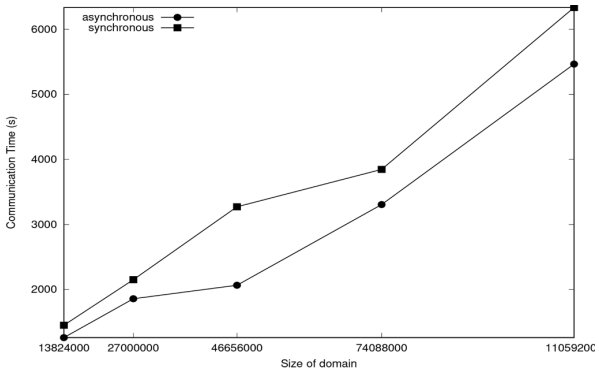


(c) Variation of the value τ with respect to the domain sizes.

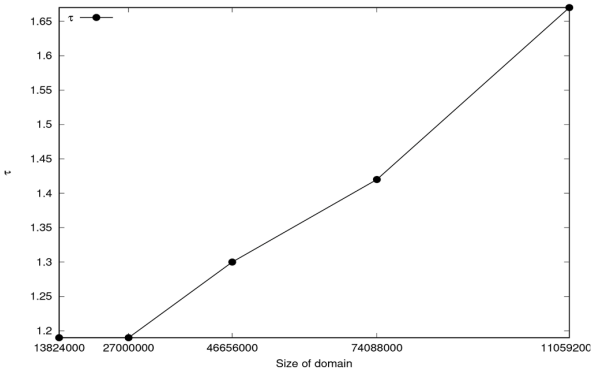
Fig. 10. Multisplitting problem with projection : Results with 600 cores on 3 clusters



(a) Elapsed time with respect to domain sizes.



(b) Communication time with respect to domain sizes.



(c) Variation of the value τ with respect to the domain sizes.

Fig. 11. Multisplitting problem with projection : Results with 1000 cores on 3 clusters

of the parallelized Newton method. For synchronous implementation, the quadratic behavior of Newton's method is preserved. Conversely, for an asynchronous implementation due to chaotic communication between processes, the quadratic convergence is not preserved; however, the additional computation has the positive effect of improving the numerical quality of the solution. In future work, from an experimental point of view, the use of such mixed method for the solution of univalued pseudo-linear stationary problem, arising in boundary value, on cloud architecture, will be considered. Besides, from a theoretical point of view, we intend to analyze non-linear multisplitting methods, using the discrete maximum principle.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationshipsthat could have appeared to influence the work reported in this paper.

Acknowledgment

This study has been made possible with the support of Grid'5000 platform. This work was partially supported by the EIPHI Graduate School (contract "ANR-17-EURE-0002").

Appendix A. Appendix

In this appendix we have therefore plotted the maximum error obtained in both synchronous and asynchronous mode of Newton method, as a function of the iteration number, which makes it possible to visualize the behavior of the parallelized Newton method.

In the following the results of these experiments are commented, both on an experimental and a theoretical level, although on the latter point the question seems to be largely open. In view of the experimental results, it can be seen that the number of Newton iterations is not the same in synchronous and asynchronous modes.

The Newton method being a fixed point iteration of the type $\delta U = H(\delta V)$, what regulates the speed of convergence is the contraction constant μ defined by

$\| H(\delta V) - H(\delta U^*) \| \leq \mu \| \delta V - \delta U^* \|, \forall \delta V, 0 < \mu < 1$, where δU^* is the fixed point.

In this case, a linear convergence is achieved. Note that this type of inequality in the result of Proposition 3 and Corollary 1 has been obtained.

Classically in the sequential case, it is known that the Newton method converges in a quadratic way towards the solution of the problem (see book [27] in particular the 10.7 theorem). The quadratic convergence of the Newton process, corresponding to the fixed point equation $\delta U = H(\delta V)$, comes in particular from the fact that the partial first derivatives of H with respect to the components of δV are zero at the fixed point δU^* according to the following theorem of [27].

Theorem 1. *Suppose δU^* is a solution of $H(\delta U) = \delta U$, where $H(\delta U) = \delta U - C^{-1}(U)(AU + \Phi(U) - G)$, for some function $H = (h_1, \cdots h_M)$ mapping $\mathbb{R}^M$ into $\mathbb{R}^M$.*

If a number $\tau > 0$ exists with the property that

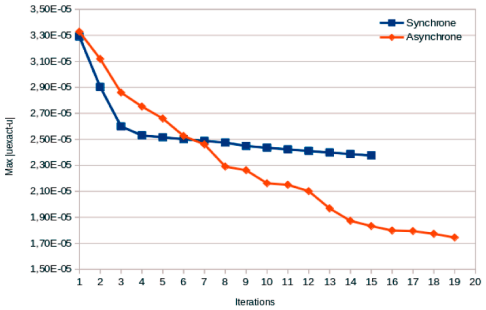
- *$\frac{\partial h_k}{\partial \delta v_l}$ is continuous on $N_\tau = \{\delta V | \parallel \delta V - \delta U^* \parallel < \tau\}$ for each $k, l = 1, \cdots M$,*
- *$\frac{\partial^2 h_k}{\partial \delta v_l \partial \delta v_j}$ is continuous and $|\frac{\partial^2 h_k}{\partial \delta v_l \partial \delta v_j}| \leq P$ for some constant P, whenever $\delta V \in N_\tau$ for each $j, k, l = 1, \cdots M$,*
- *$\frac{\partial h_k(\delta U^*)}{\partial \delta u_l} = 0$ for each $k, l = 1, \cdots M$,*

then the sequence generated by $\delta U^{(i+1)} = H(\delta U^{(i)})$ converges quadratically to δU^* for any initial guess $\delta U^{(0)}$ provided that $\parallel \delta U^{(0)} - \delta U^* \parallel < \tau$. Moreover $\parallel \delta U^{(i+1)} - \delta U^* \parallel_\infty \leq \frac{M^2 P}{2} \parallel \delta U^{(i)} - \delta U^* \parallel_\infty^2$ for $i > 0$.

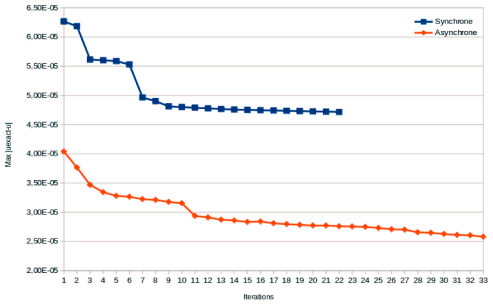
Considering the synchronous parallel implementation of the Newton method, which in fact reproduces the sequential implementation of this method, a quadratic convergence is logically expected. Note that in this case the convergence will be faster than announced in the result of Proposition 3 and Corollary 1.

On the other hand, if we now consider an asynchronous parallel implementation of the Newton method, the Jacobian matrix will be calculated with component values delayed because of asynchronous communications. However, the asynchronous algorithm being non-deterministic, due to the fact that on the one hand the vector to be computed is decomposed into blocks and on the other hand, when asynchronous iterations are performed, the updates are chaotic contrary to the sequential or the synchronous methods, it is not obvious that the partial derivatives of H with respect to its components are continuous in a neighborhood of the solution. It should be noted that in a calculation process the component blocks may well be continuous but that there might be several discontinuities from one component block to another. This discontinuity leads to a discontinuity in the values of the derivatives of H with respect to its components. In these conditions, we are no longer in the hypotheses of the previous theorem and we are therefore not sure that the first and second derivatives of H with respect to its components are continuous.

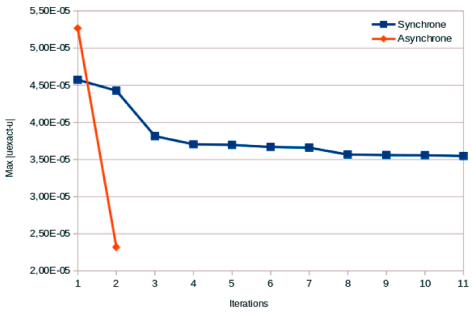
Thus directly determining the order of convergence (quadratic or not) of the asynchronous parallel Newton method is a difficult problem and a widely open question that is outside the initial objectives of the submitted paper.



(a) 240^3 = 13 824 000



(b) 360^3 = 46 656 000



(c) 480^3 = 110 592 000

Fig. 12. Newton Multisplitting problem : Results with 8 cores on 1 cluster

Moreover, in the general asynchronous case, it has been established in the case of the asynchronous multisplitting method that the fixed point application is a contraction, which corresponds to a linear and no longer quadratic convergence (see Proposition 3 and Corollary 1); this explains why the number of Newton iterations is not the same in synchronous or asynchronous mode. Thus, asynchronous communications between computation processes lead to the loss of the quadratic character of the Newton convergence, which is not the case in synchronous mode.

So, in conclusion, the order of the iterative process impacts the speed of convergence. This speed of convergence is quadratic in the sequential or synchronous case, and is linear in the asynchronous case in accordance with the results of Proposition 3 and Corollary 1. The character of quadratic convergence is therefore no longer retained.

Results with 8 cores on 1 cluster with $n = 2$ *and* $q = 4$

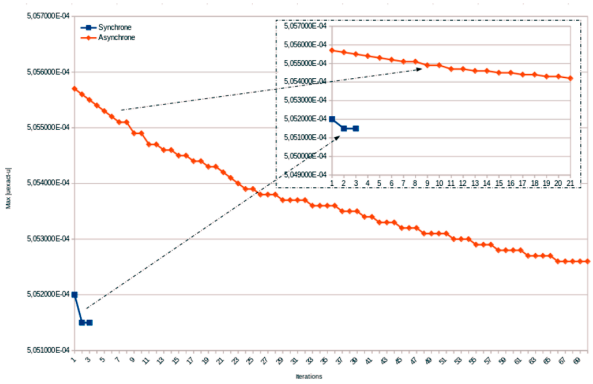
Results with 600 cores on 1 cluster with $n = 2$ *and* $q = 300$

At the experimental level, in our parallel experiments, in both synchronous and asynchronous modes, the same calculation codes were used. The only differences lie that in the synchronous mode the MPI_SEND and MPI_RECEIVE routines were used, while in the asynchronous mode the MPI_ISEND and MPI_IRECEIVE routines were used. This justifies the comparison between both versions. Otherwise the comparison of synchronous mode versus asynchronous mode would make no sense.

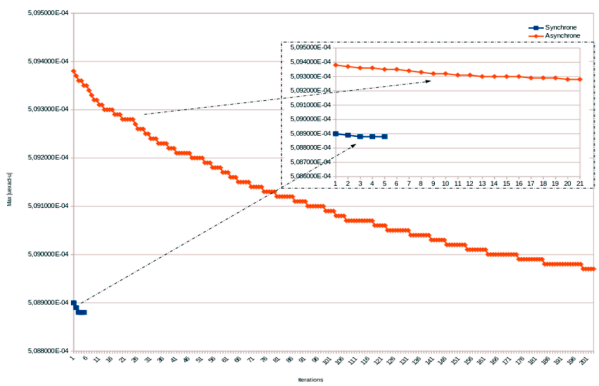
However, considering that the suno cluster on the Sophia site is no longer in the Grid'5000 network, we have limited our tests to the use of

- a local cluster with an 8-core Intel Xeon E5-2630 v4 @ 2.2 GHz processor and shared memory to exchange messages,
- the cluster named paravance (see Table 2) of Grid'5000 composed of 600 cores where the machines are connected to each other by a network,
- two distant clusters geographically distant named paravance and ecotype (see Table 2) composed of 600 cores connected by a network. There are several machines in these clusters and all the cores of each machine are used. The cores on the same machine use shared memory to exchange messages.

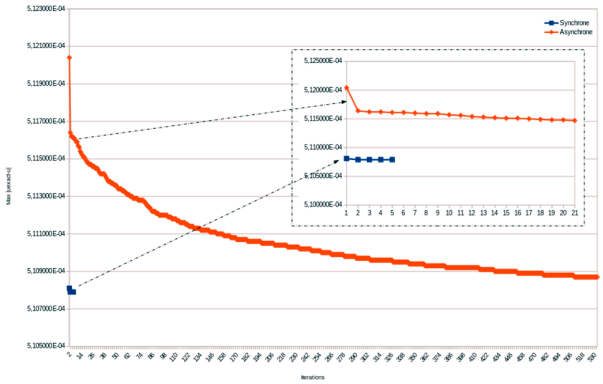
In each case the problem sizes corresponding to 240^3; 360^3 and 480^3 are considered and for each data set the maximum error is plotted according to the Newton iteration number (see Figs. 12–14 where synchronous and asynchronous communications mode are considered).



(a) $240^3 = 13\,824\,000$

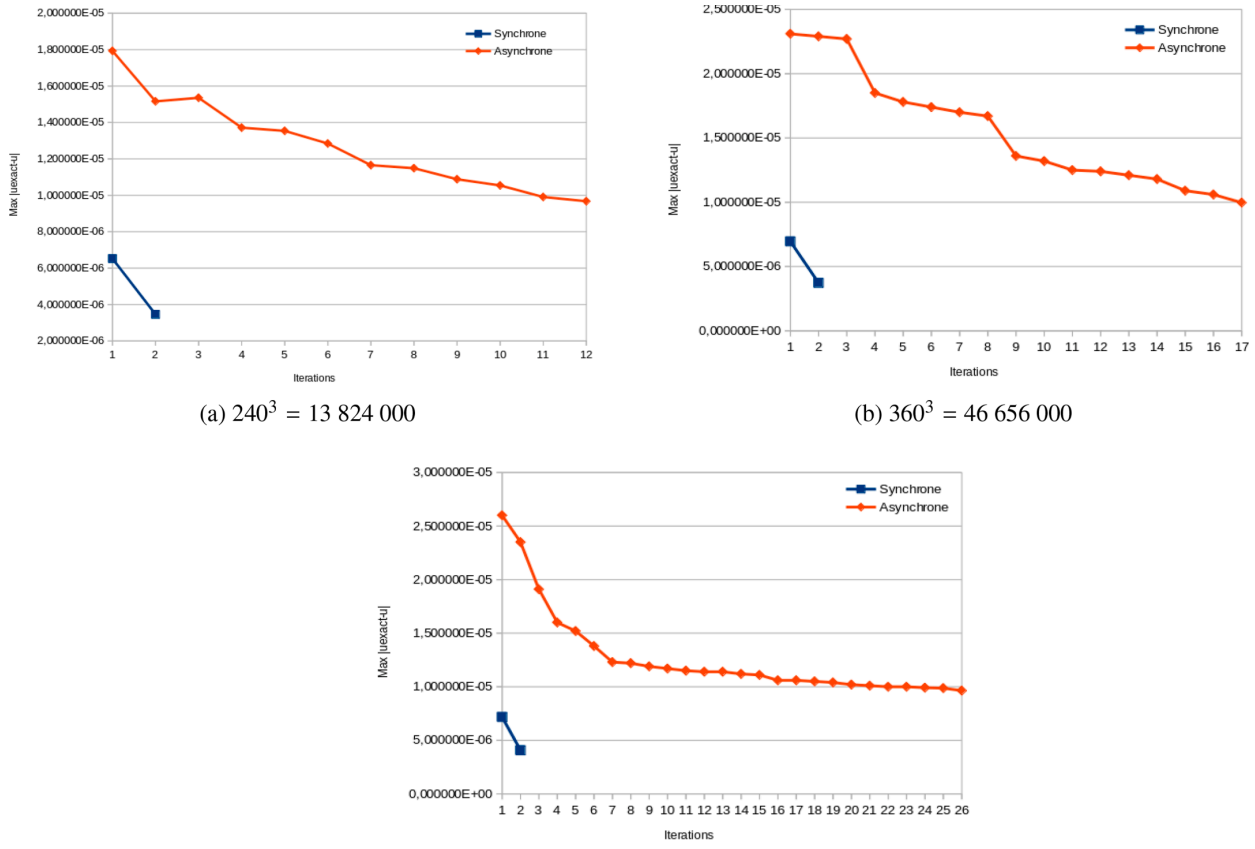


(b) $360^3 = 46\,656\,000$



(c) $480^3 = 110\,592\,000$

Fig. 13. Newton Multisplitting problem : Results with 600 cores on 1 cluster



(a) $240^3 = 13\,824\,000$

(b) $360^3 = 46\,656\,000$

(c) $480^3 = 110\,592\,000$

Fig. 14. Newton Multisplitting problem : Results with 600 cores on 2 clusters

There is a different behavior when switching from one machine to another, the architecture of the machine having an influence on the result obtained. The essential elements influencing the behavior being the speed of communication and the updates of the components. It is clear that the synchronous version converges faster than the asynchronous version, which is consistent with the heuristic explanation given above. However, the number of synchronous iterations on the local cluster is higher than on the Grid'5000 cluster or on the two geographically distant clusters of Grid'5000. Moreover, on the local cluster, except in the case 360^3, the error in asynchronous communication mode is lower than that obtained in synchronous mode. Additionally, we can see from the numerical values of the uniform error norm that these values are very close to the order of $1.0E-05$ on the local cluster and to the order of $1.0E-04$ on the Grid'5000 cluster.

When two geographically remote clusters connected by a network are used, the numerical values of the uniform error norm are ten times higher in the asynchronous case than in the synchronous one. It is therefore normal under these conditions that convergence in asynchronous mode is slower.

Finally, in the synchronous case the decrease of the uniform norm of the error is uniform, whereas in the asynchronous case it is chaotic insofar as one notes alternatively increasing and then decreasing fluctuations of the graph. This is due to the non-deterministic mode of asynchronous communications and the limitation due to the distance between both clusters in the latter case. We note that the number of Newton iterations increases with the number of cores in the asynchronous mode. However, the restitution time is better in the asynchronous mode and this is what we are trying to do by eliminating idle time due to asynchronous mode expectations. Finally, the extra computation time in asynchronous mode improves the numerical quality of the resulting solution. It should also be noted that the asynchronous mode is not intended to accelerate convergence.

Results with 600 cores on 2 clusters with $n = 2$ and $q = 300$

References

[1] Chazan D, Miranker W. Chaotic relaxation. Linear Algebra Appl 1969;2:199–222.
[2] Miellou JC. Algorithmes de relaxation chaotique à retards. RAIRO Analyse numérique 1975;1:55–82.
[3] Baudet G. Asynchronous iterative methods for multiprocessors. Journal Assoc Comput Mach 1978;25:226–44.
[4] Frommer A, Szyld D. On asynchronous iterations. Journal of Computational and Applied Mathematics 2000;123:201–16.
[5] Tarazi ME. "some convergence results for asynchronous algorithms. Numerische Mathematik 1984;39:325–40.
[6] O'Leary DP, White RE. Multi-splittings of matrices and parallel solution of linear systems. SIAM:Journal on Algebraic Discrete Methods 1985;6:630–40.
[7] White R.E., Meth. J.A.D.. Parallel algorithms for nonlinear problems, SIAM. 1986. 7, 137–149.
[8] Arnal J, Migallón V, Penadés J. Parallel newton two-stage multisplitting iterative methods for nonlinear systems. BIT Numerical Mathematics 2003;43:849–61.
[9] Szyld D. Different models of parallel asynchronous iterations with overlapping block. Computational and Applied Mathematics 1998;17:101–15.
[10] Bahi JM, Miellou JC, Rhofir K. Asynchronous multisplitting methods for nonlinear fixed point problems. Numerical Algorithms 1997;15:315–45.
[11] Saad Y. Iterative methods for sparse linear systems. SIAM 2003.
[12] Mossino J. Sur certaines inéquations quasi-variationnelles apparaissant en physique. CRAS Paris 1976;282:187–90.

[13] Spiteri P, Ziane-Khodja L, Couturier R. Asynchronous parallel multi-splitting mixed methods. Civil-Comp Press, Stirlingshire, UK, Paper 15 2019. https://doi.org/10.4203/ccp.112.15.in P. Ivnyi, B.H.V. Topping, (Editors), "Proceedings of the Sixth International Conference on Parallel, Distributed, GPU and Cloud Computing for Engineering
[14] Barbu V. Nonlinear semigroups and differential equations in Banach spaces. Noordhoff International Publishing; 1976.
[15] Glowinski R, Lions JL, Tremolières R. Analyse numérique des inéquations variationnelles. DUNOD, tome 1 and 2 1976.
[16] Miellou J-C, Spiteri P. Un critère de convergence pour des méthodes générales de point fixe. M2AN 1985;19:645–69.
[17] Chau M, Laouar A, Garcia T, Spitri P. Grid solution of problem with unilateral constraints. Numerical Algorithms, Springer-Verlag 2017;(75 - 74):879–908.
[18] Ortega J, Rheinboldt W. Iterative Solution of Nonlinear Equations in Several Variables. New York: Academic Press; 1970.
[19] Evans DJ, Deren W. An asynchronous parallel algorithm for solving a class of nonlinear simultaneous equations. Parallel Computing 1991;17:165–80.
[20] Saad Y, Schultz MH. "GMRES: A Generalized Minimal Residual Algorithm for Solving Nonsymmetric Linear Systems. SIAM Journal on Scientific and Statistical Computing 1986;7(3):856–69.
[21] Ramamonjisoa CE, KhodjaZiane L, Laiymani D, Giersch A, Couturier R. Simulation of asynchronous iterative algorithms using simgrid. In 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS) 2014:890–5.
[22] Bolze R, Cappello F, Caron E, Daydé M, Desprez F, Jeannot E, Jégou Y, Lanteri S, Leduc J, Melab N, Mornet G, Namyst R, Primet P, Quetier B, Richard O, Talbi E-G, Touche I. Grid5000: A large scale and highly reconfigurable experimental grid testbed. International Journal of High Performance Computing Applications 2006;20(4):481494.
[23] https://www.grid5000.fr/w/Nantes:Hardware#ecotype.
[24] https://www.grid5000.fr/w/Rennes:Hardware#parapide.
[25] https://www.grid5000.fr/w/Rennes:Hardware#paravance.
[26] https://www.grid5000.fr/w/Sophia:Hardware#suno.
[27] Burden RL, Faires JD. Numerical analysis. BROOKS/COLE, CENGAGE Learning, Ninth edition 2011:639–44.

5.4 Article 4

Advances in Engineering Software 133 (2019) 76–95

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Asynchronous multi-splitting method for linear and pseudo-linear problems

V. Partimbene[*,a,c], T. Garcia[b], P. Spiteri[a], P. Marthon[a], L. Ratsifandrihana[c]

[a] University of Toulouse, IRIT-INPT-ENSEEIHT, 2 rue Charles Camichel, Toulouse 31071, France
[b] University of Toulouse, IRIT-INPT, 2 rue Charles Camichel, Toulouse 31071, France
[c] SEGULA Aerospace & Defence, 24 Boulevard Déodat de Séverac, Colomiers 31770, France

ARTICLE INFO

Keywords:
Multi-splitting method
Fluid-structure interaction
Finite volume
Parallel algorithms
Message passing interface

ABSTRACT

This paper deals with the solution of fluid-structure interaction problems using an algorithm consisting in the coupling between two solvers, each one related to the fluid and the structure respectively. Due to the difficult decomposition of the domain into sub-domains, we consider for each environment a parallel multi-splitting algorithm which corresponds to an unified presentation of sub-domain methods with or without overlapping. Such method combines several contractant fixed point mappings and, we show that, under appropriate assumptions, each fixed point mapping are contractive in finite dimensional spaces normed by Hilbertian and non-Hilbertian norms. Moreover, we show in a novel way that such study is valid for the parallel synchronous and more generally asynchronous large scale linear systems arising in the solution of fluid-structure interaction problems and can be extended to the case where the displacement of the structure is submitted to some constraints. We perform parallel simulations for a fluid-structure test case on different clusters, considering blocking and non-blocking communications. The performances of the parallel simulations are presented and analyzed.

1. Introduction

Fluid-structure interaction (FSI) occurs in many areas such as aeronautics, automotive, biomechanics, railway industry, shipbuilding, civil engineering, etc. It involves the coupling of the Navier–Stokes equations for the fluid and the Navier equation of linear elasticity for the structure. The main advantage of fluid-structure interaction studies is to accurately predict the aerodynamic or hydrodynamic effects on a structure submitted to flow due to phenomena such as flutter or vortex induced vibrations which can possibly have destructive consequences.

In the present study, the selected solution of the fluid-structure problem consists in solving distinctly the fluid model and the structure model; the coupling between these two models is done by exchange of informations at the interfaces between the two domains, Ω_f, the fluid domain and Ω_s, the domain in which the structure is located. More precisely, for the fluid-structure interaction, the pressure and the shear stresses due to the fluid viscosity (and computed with the fluid velocity components) obtained by the solution of the Navier–Stokes equations are transferred during the treatment of the solid part as boundary conditions; besides the velocities components and the displacement are then transferred during the treatment of the fluid part. During the transitional phase, simply adding the transient term is not sufficient due to the deforming mesh. In order to account for the mesh movement, we use the arbitrary Lagrangian Eulerian (ALE) formulation in which the convective flux through a cell face, constituting the mesh in the fluid region, is computed relatively to the velocity of the cell face.

Obviously, for the computation of the fluid-structure interaction it is necessary to solve on one hand the Navier–Stokes equations describing the flow behavior and on the other hand the Navier equation modeling the behavior of the structure. For such classical equations, the more efficient numerical algorithms used are the pressure implicit with splitting of operator (PISO) algorithm for the fluid [1] and, in order to avoid instabilities, according to a suggestion from Jasak and Weller [2], a semi-implicit time marching scheme where a part of the evolution equation is discretized by an implicit time marching scheme while the rest of the equation is discretized explicitly. Note that, in order to avoid errors due to differences in discretization, the spatial part of both partial differential equations are discretized by the finite volume method; then, such procedure makes it easier to connect the two meshes defined in Ω_f and Ω_s. Taking into account the properties of the obtained discretization matrices, the large sparse algebraic linear systems are solved by block methods using different appropriate iterative numerical solvers allowing on one hand software modularity and on the other hand the use of more efficient methods in each sub-domain like the preconditioned conjugate gradient methods, or variants of the algebraic multi-grid methods.

[*] Corresponding author at: University of Toulouse, IRIT-INPT-ENSEEIHT, 2 rue Charles Camichel, 31071 Toulouse, France.
E-mail addresses: vincent.partimbene@enseeiht.fr (V. Partimbene), thierry.garcia@irit.fr (T. Garcia), pierre.spiteri@enseeiht.fr (P. Spiteri), philippe.marthon@enseeiht.fr (P. Marthon), leon.ratsifandrihana@segula.fr (L. Ratsifandrihana).

https://doi.org/10.1016/j.advengsoft.2019.03.001
Received 18 September 2017; Received in revised form 5 December 2018; Accepted 7 March 2019
Available online 09 May 2019
0965-9978/

However, due to the use of implicit or semi-implicit time marching schemes which have the advantage of being unconditionally stable, in order to obtain accurate results, we have to use very fine meshes on both the fluid and the structure sub-domains. In order to efficiently solve the discretized systems, we are drawn to parallelize the solution algorithms on multi-processor clusters, which leads to a kind of sub-domain method for the solution of the Navier–Stokes equations on one hand and of the Navier equation otherwise. The objective of this paper is to solve the fluid-structure interaction problem by asynchronous parallel iterative methods ; the same asynchronous frame is used for the coupling itself as well as for the solution of individual problems. The use of such methods is well justified on the fluid part where this type of method has already been successfully used in [3] and on the solid part considering the properties of strong diagonal dominance obtained by implicit discretization of the Navier equation, linked as well for the fluid part as for the structural part to the properties of the M-matrices. It should be noted that solving such problems by synchronous parallel iterative methods is also feasible. However, given the slow convergence speed of the iterative methods used, these latter synchronous methods require many synchronizations that would penalize the performance of these algorithms. Moreover, asynchronous parallel iterative methods are largely possible and applicable for this type of application. Nevertheless, when the meshes obtained by discretization of the domain Ω_f and Ω_s are structured, the partitioning of each previous domain into sub-domains is very easy. However, in reality, for the considered fluid-structure application, the mesh is in general unstructured; under these conditions, an algebraic decomposition of the matrix involved in the algebraic systems to be solved is made rather than a real geometric decomposition of each domain in sub-domains. This leads to describe the multi-splitting methods that can also take into account algebraic overlaps between the sub-domains. In such multi-splitting method, we have to solve several sub-problems derived from the global problem; each sub-problem can be solved independently by a direct or an iterative method. Thus the computations of all sub-problems may be performed in parallel by a set of processors. So the multi-splitting method splits the problem into several blocks in order to reduce the large amount of communications and to implement both an inner and an outer iterations obtained using adapted iterative methods, allowing to improve the convergence of the basic multi-splitting method.

Among the many authors who have worked on chaotic parallel methods are Chazan and Miranker [4] who introduced these methods to solve large linear systems in 1969. This work was extended by Miellou in 1975 for non-linear problems, the analysis of the behavior of these methods being carried out by using contraction techniques in vectorial norm [5] as well as order interval techniques [6]. Miellou's results were then extended to the case of asynchronous parallel iterations by Baudet in 1978 [7]. Bertsekas et al. [8] analyzed these methods using nested techniques centered on the solution of the problem. In 1982, El Tarazi [9] published a convergence result based on the use of a weighted uniform norm. Many other important works have been done on the difficult question of the termination of these methods, which use both numerical considerations like Miellou, Spiteri et al. [10] as well as computing considerations like Magoulès [11] and Bahi et al. [12]. Since we cannot mention them all, we also refer to other works such as [13–15]. In addition, studies on the implementation of these methods [16] and on the numerous applications of the latter on various fields have been carried out, for example in [3,17]. Moreover, these methods and their analysis have been extended to the multi-splitting methods by many authors such as O'Leary and White [18], Szyld [19], Frommer [20], Arnal et al. [21], Bahi, Miellou et al. [22]. With regard more specifically to the resolution of fluid-structure interaction by sub-domain methods, we refer to the work of Magoulès [23,24]. In [23], the authors use sub-domain methods of the Shur complement type distinct from the point of view considered in the submitted paper ; and in [24] they consider Krylov methods combined with domain decomposition methods implemented on GPUs.

In most scientific works related to the mathematical behavior of the multi-splitting method, it is assumed that each splitting leads to a contractant fixed point mapping. Note that such property is not obvious to verify practically. In the present study, for the large algebraic systems and more generally for particular classes of pseudo-linear single-valued and multi-valued algebraic systems, these latter corresponding to problems where the solution is submitted to some constraints, we show that under appropriate assumptions, each fixed point mapping arising in the multi-splitting decomposition is contractive; note that the assumptions providing such contraction property are well verified for the FSI application and moreover, such result allowing to precise classical assumption is new. Moreover, we extend the topological context considered in [22] limited to the Hilbertian context, to the case where the considered finite dimensional space where the discrete equations are defined is normed by non-Hilbertian norms; such extension avoids the numerical computations of the eigenvalues being expensive and not easy to obtain. Moreover, since classically the discretization matrices are strictly diagonally dominant or irreducibly diagonally dominant, such extension to the non-Hilbertian context provides very easy to apply practical criteria, in order to show the convergence of the multi-splitting method.

For the solution of our target application, a wall submitted to a cross flow, parallel numerical experiments have been carried out on a local cluster and also on Grid'5000 [25] in France. In view of synchronization and communication weight, we considered both blocking and non-blocking exchanges between processors. It's worth noting that, however, the non-blocking method was difficult to implement because the messages are delivered in a chaotic order by all processors. The implementation of the non-blocking method is an addition to the work presented in [26].

The present paper is organized as follows. In Section 2, we present the mathematical models which govern the physics of fluid-structure interaction. In Section 3, the numerical procedure for the fluid-structure interaction solution is presented. In Section 4, the parallel numerical solution is detailed. In Section 5, we discuss the parallelization schemes used to solve the problem and finally, in Section 6, we present some results on a fluid-structure application.

2. Mathematical models

In the present section, we show the governing equations for the fluid flow and the structure. In the sequel, the fluid is assumed to be isothermal, incompressible and laminar.

2.1. Fluid flow

2.1.1. Governing equations

Let $\Omega_f(t) \subset \mathbb{R}^\eta$, $\eta = 2, 3$, be the spatial fluid mechanics domain with boundary $\partial\Omega_f$ at time $t \in (0,T)$. The fluid flow is governed by the incompressible Navier–Stokes equations on $\Omega_f(t)$ and $\forall t \in (0,T)$

$$\rho_f\left(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\nabla\mathbf{u} - \mathbf{f}_f\right) + \nabla p - \mu_f \nabla^2\mathbf{u} = 0, \tag{1}$$

$$\nabla\cdot\mathbf{u} = 0, \tag{2}$$

where ρ_f, $\mathbf{u}$ and $\mathbf{f}_f$ are the fluid density, velocity and the external force (per unit mass), p is the pressure and μ_f the dynamic viscosity. All bold variables refer to vectors. Eqs. (1) and (2) represent the momentum and mass conservation respectively.

2.1.2. Boundary conditions

The different boundary conditions at the solid surface, at the inflow and outflow and at the lateral surfaces are described as follows.

Solid surface. As a matter of convenience, the velocity vector can be split into its normal u_n and tangential u_t components along the normal vector $\mathbf{n}$ and two orthonormal tangential vectors $\mathbf{t}_1$ and $\mathbf{t}_2$ on $\partial\Omega_f$. The

no-penetration boundary condition becomes

$$u_n = g_n, \tag{3}$$

where g_n is the normal velocity of the solid surface. In the case of viscous flows, the remaining velocity components are set to

$$u_{t_1} = g_{t_1}, \tag{4}$$

$$u_{t_2} = g_{t_2}, \tag{5}$$

where g_t's are the tangential velocities of the solid surface.

Inflow. The entire velocity vector is prescribed:

$$\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \end{Bmatrix} = \begin{Bmatrix} U_1 \\ 0 \\ 0 \end{Bmatrix}. \tag{6}$$

where u_i ($i = 1, 2, 3$) are the components of the velocity and U_1 the prescribed velocity.

Outflow. Free-stream traction boundary conditions are prescribed:

$$\boldsymbol{\sigma}^f \cdot \mathbf{n} = \mathbf{0}, \tag{7}$$

where $\boldsymbol{\sigma}^f$ is the stress tensor defined by

$$\nabla \cdot \boldsymbol{\sigma}^f(\mathbf{u}, p) = -\nabla p + \mu_f \nabla^2 \mathbf{u}. \tag{8}$$

Assuming the normal vector is $\mathbf{n} = (1, 0, 0)^T$, the component form is

$$\begin{Bmatrix} \sigma^f_{11} \\ \sigma^f_{21} \\ \sigma^f_{31} \end{Bmatrix} = \begin{Bmatrix} 0 \\ 0 \\ 0 \end{Bmatrix}. \tag{9}$$

Lateral boundary. Zero normal velocity and tangential traction are prescribed:

$$\mathbf{u} \cdot \mathbf{n} = 0, \tag{10}$$

$$\mathbf{t}_1 \cdot \boldsymbol{\sigma}^f \cdot \mathbf{n} = 0, \tag{11}$$

$$\mathbf{t}_2 \cdot \boldsymbol{\sigma}^f \cdot \mathbf{n} = 0. \tag{12}$$

Boundary conditions (3) to (6), (9) and (10) are Dirichlet boundary conditions and the remaining ones are Neumann boundary conditions.

2.1.3. Arbitrary Lagrangian–Eulerian description

The equations governing the fluid behavior are derived with respect to a stationary (or Eulerian) reference frame : the domain does not follow the motion of the fluid itself. To account for the movement of the mesh during the simulation, the arbitrary Lagrangian–Eulerian (ALE) description is used and the convective flux through a cell face in the fluid region is calculated relative to the velocity of the cell face.

Let $\hat{\Omega}(t) \subset \mathbb{R}^\eta$, $\eta = 2, 3$, be the reference domain and $\hat{\mathbf{x}}$ its coordinates. The convective form of the momentum conservation equation of incompressible flows in the ALE description is

$$\rho_f \left(\left. \frac{\partial \mathbf{u}}{\partial t} \right|_{\hat{\chi}} + (\mathbf{u} - \hat{\mathbf{u}}) \cdot \nabla \mathbf{u} - \mathbf{f}_f \right) + \nabla p - \mu_f \nabla^2 \mathbf{u} = 0, \tag{13}$$

where $|_{\hat{\chi}}$ denotes the time derivative taken holding $\hat{\mathbf{x}}$ fixed and $\hat{\mathbf{u}}$ is the computational domain velocity. Note that $(\mathbf{u} - \hat{\mathbf{u}})$ represents the speed between the fluid and the computational grid, also called convective velocity.

2.2. Structure

2.2.1. Governing equations

Let $\Omega_s(t) \subset \mathbb{R}^\eta$, $\eta = 2, 3$, be the spatial solid mechanics domain with boundary $\partial\Omega_s$ at time $t \in (0,T)$. $\Omega_s(t)$ and $\Omega_f(t)$ have some boundaries in common. The behavior of the structure is described by the Navier displacement equations for linear elasticity on $\Omega_s(t)$ and $\forall t \in (0,T)$

$$\rho_s \frac{\partial^2 \mathbf{D}}{\partial t^2} - \nabla \cdot \left[\mu(\nabla \mathbf{D} + (\nabla \mathbf{D})^T) + \lambda \mathbf{I}(\nabla \cdot \mathbf{D}) \right] - \rho_s \mathbf{f}_s = 0, \tag{14}$$

where ρ_s, $\mathbf{D}$ and $\mathbf{f}_s$ are the solid density, displacement and the body force (per unit mass), $\mathbf{I}$ is the identity tensor and (μ, λ) are the Lamé coefficients related to the Young's modulus E and the Poisson ratio ν such as

$$\mu = \frac{E}{2(1+\nu)},$$

and

$$\lambda = \begin{cases} \dfrac{\nu E}{(1+\nu)(1-\nu)} & \text{for plane stress,} \\ \dfrac{\nu E}{(1+\nu)(1-2\nu)} & \text{for plane strain and 3D.} \end{cases}$$

Due to thermodynamics laws:

$$\mu > 0 \text{ and } 2\mu + 3\lambda > 0.$$

2.2.2. Boundary conditions

Displacement. The displacement is specified on the boundary, as a Dirichlet boundary condition:

$$\mathbf{D}|_{\partial\Omega_s} = \mathbf{D}_1,$$

with $\mathbf{D}_1$ being the prescribed displacement vector.

Stress. The traction is specified on the boundary, as a Neumann boundary condition:

$$\boldsymbol{\sigma}^s \cdot \mathbf{n}|_{\partial\Omega_s} = \tau,$$

where τ is the traction applied to the boundary, $\mathbf{n}$ is the outer normal vector and $\boldsymbol{\sigma}^s$ is the Cauchy stress tensor defined by the Hooke law

$$\boldsymbol{\sigma}^s = 2\mu\varepsilon^s + \lambda(\nabla \cdot \mathbf{D})\mathbf{I},$$

where ε^s is the strain tensor.

3. Numerical procedure for FSI

We present hereafter the Navier–Stokes solution and the Navier solution which consist in fact in solving convection–diffusion equations or diffusion equations discretized by the finite volume method and, finally we present the fluid-structure procedure.

3.1. Navier–Stokes solution – PISO algorithm

PISO (i.e. pressure-implicit with splitting of operator) [27] is a time marching predictor-corrector algorithm based on the splitting of the solution of velocity equations and pressure equations. It splits the operators into an implicit predictor and two explicit corrector steps and is designed to satisfy mass conservation using predictor-corrector steps and may be seen as an extension of the SIMPLE algorithm [1] with an additional corrector step to enhance it.

The governing transport equations (momentum and continuity) in their incompressible form are given by Eqs. (1) and (2). In order to explain the process of the PISO method, the transport equation can be expressed in finite-difference form with the Euler semi-implicit difference scheme for an easy and clear presentation. However, the method of solution is not restricted to the use of that particular scheme.

If n and $n+1$ denote successive time levels, the system of governing equations for an incompressible fluid may be expressed in difference form for each mesh point as

$$\frac{\rho_f}{\delta t}(\mathbf{u}^{n+1} - \mathbf{u}^n) = H(\mathbf{u}^{n+1}) - \Delta p^{n+1}, \tag{15}$$

and

$$\Delta \mathbf{u}^{n+1} = 0. \tag{16}$$

In Eqs. (15) and (16), the operator Δ is the finite-difference equivalent of $\partial/\partial\mathbf{x}$, and in Eq. (15) H is the discretization matrix of the convective-diffusive operator resulting from the semi-implicit time marching scheme. It usually takes the form

$$H(\mathbf{u}) = \mathcal{A}_m \mathbf{u}_m,$$

where suffix m is a grid identifier and the summation is over all the nodes involved in the formulation of the finite-difference representation of the spatial fluxes. The $\mathcal{A}$ coefficient is function of the velocities, densities, etc. Therefore H is non-linear, but assuming that $\mathcal{A}$ remains constant over the interval δt, allows to linearize the operator H.

The pressure equation is derived by taking the divergence of Eq. (15) and substituting in Eq. (16) is

$$\Delta^2 p^{n+1} = \Delta H(\mathbf{u}^{n+1}) + \frac{\rho_f}{\delta t}\Delta \mathbf{u}^n. \tag{17}$$

Let the superscripts *, **, and *** denote intermediate field values obtained during the splitting process.

3.1.1. Predictor step

The pressure field prevailing at t^n is used in the discretized momentum Eq. (15) to get the velocity field $\mathbf{u}^*$:

$$\frac{\rho_f}{\delta t}(\mathbf{u}^* - \mathbf{u}^n) = H(\mathbf{u}^*) - \Delta p^n. \tag{18}$$

This algebraic system can be solved by a standard techniques to yield the $\mathbf{u}^*$ field which may not satisfy the continuity Eq. (16); thus, it needs to be corrected.

Due to the fact that the value of the pressure is fixed at its previous value at t^n, then, if for example, these momentum equations are solved by a semi-implicit time marching scheme where the non-linearity in the convection term $\mathbf{u}\cdot\nabla\mathbf{u}$ is handled by taking $\mathbf{u} = \mathbf{u}^n$ on one hand for the first argument and $\mathbf{u} = \mathbf{u}^*$ on the other hand for the second one, which is a valid procedure when the time discretization step is chosen small, we have to solve three decoupled convection–diffusion equations.

3.1.2. First corrector step

A new velocity field $\mathbf{u}^{**}$ and pressure field p^* are sought to satisfy the continuity condition

$$\Delta \mathbf{u}^{**} = 0. \tag{19}$$

The corresponding momentum system is

$$\frac{\rho_f}{\delta t}(\mathbf{u}^{**} - \mathbf{u}^n) = H(\mathbf{u}^*) - \Delta p^*, \tag{20}$$

which is of explicit type since H operates on $\mathbf{u}^*$. This explicit form is useful to solve the following algebraic system to find the pressure p^*:

$$\Delta^2 p^* = \Delta H(\mathbf{u}^*) + \frac{\rho_f}{\delta t}\Delta \mathbf{u}^n, \tag{21}$$

since the right-hand side contains terms in the known field $\mathbf{u}^*$. The p^* field is then inserted in Eq. (20) to get $\mathbf{u}^{**}$ which now satisfies Eq. (19).

3.1.3. Second corrector step

Similarly to the first corrector step, a new velocity field $\mathbf{u}^{***}$ and pressure field p^{**} are sought to satisfy the continuity condition

$$\Delta \mathbf{u}^{***} = 0, \tag{22}$$

with the corresponding momentum explicit-type system

$$\frac{\rho_f}{\delta t}(\mathbf{u}^{***} - \mathbf{u}^n) = H(\mathbf{u}^{**}) - \Delta p^{**}, \tag{23}$$

and the corresponding pressure system

$$\Delta^2 p^{**} = \Delta H(\mathbf{u}^{**}) + \frac{\rho_f}{\delta t}\Delta \mathbf{u}^n. \tag{24}$$

In a similar way as in the first corrector step, the p^{**} field can be easily determine since the right-hand side of the algebraic system (24) is known, and, once introduced in system (23), the $\mathbf{u}^{***}$ field can be evaluated.

More corrector steps can be used but, as shown by Issa [27], the accuracy with which $\mathbf{u}^{***}$ and p^{**} approximate the exact solution $\mathbf{u}^{n+1}$ and p^{n+1} is sufficient for most practical purposes.

Note that (21) and (24) lead to solve two diffusion equations.

1: **while** not converged **do**
2: solve discretized momentum system (18)
3: solve first pressure correction system (21)
4: correct pressure and velocities (20)
5: solve second pressure correction system (24)
6: correct pressure and velocities (23)
7: solve all other discretized transport systems
8: **end do**

Algorithm 1. The PISO algorithm.

3.2. Discretization of linear elasticity equations

The finite volume method uses the integral form of Eq. (14) over an arbitrary control volume V_P and with application of the Gauß integral theorem, it follows:

$$\int_{V_P} \rho_s \frac{\partial^2 \mathbf{D}}{\partial t^2} \mathrm{d}V - \oint_{\partial V_P} \mathbf{n} dS \cdot \left[\mu \nabla \mathbf{D} + \mu (\nabla \mathbf{D})^T + \lambda \mathbf{I} \mathrm{tr}(\nabla \mathbf{D}) \right] = \int_{V_P} \rho_s \mathbf{f}_s \mathrm{d}V. \tag{25}$$

Following the method developed in [2], the temporal derivative is calculated using two old-time levels of $\mathbf{D}$:

$$\frac{\partial^2 \mathbf{D}}{\partial t^2} = \frac{\mathbf{D}(t+\delta t) - 2\mathbf{D}(t) + \mathbf{D}(t-\delta t)}{\delta t^2}. \tag{26}$$

The volume integrals are evaluated using the mid-point rule

$$\int_{V_P} \phi \mathrm{d}V = \phi_P V_P. \tag{27}$$

The Div-Grad term is split into sum of integrals over cell faces and also evaluated using the mid-point rule:

$$\int_{V_P} \nabla\cdot(\mu \nabla \mathbf{D}) \mathrm{d}V = \oint_{\partial V_P} (\mu \nabla \mathbf{D}) \mathbf{n} \mathrm{d}S = \sum_c \mu (\nabla \mathbf{D})_c \mathbf{n} S_c, \tag{28}$$

where S_c is the face area. Two types of discretization are recognized, an implicit discretization which leads to:

$$\oint_{\partial V_P} (\mu \nabla \mathbf{D}) \mathbf{n} \mathrm{d}S = \bar{a}_P \mathbf{D}_P + \sum_C \bar{a}_C \mathbf{D}_C, \tag{29}$$

where

$$\bar{a}_C = -\mu \frac{|\mathbf{n} S_c|}{|\mathbf{d}_C|}, \quad \bar{a}_P = -\sum_C a_C \quad \text{and} \quad \mathbf{d}_C = \overline{PC}, \tag{30}$$

and an explicit discretization:

$$(\nabla \mathbf{D})_c = f_x (\nabla \mathbf{D})_P + (1 - f_x)(\nabla \mathbf{D})_C, \tag{31}$$

where f_x is the interpolation coefficient.

The cell center gradient is calculated using least-square fit. Assuming a linear variation of a general variable ϕ and minimizing the error at C leads to the following expression:

$$(\nabla \phi)_P = \sum_C w_C^2 \mathbf{G}^{-1} \cdot \mathbf{d}_C (\phi_C - \phi_P), \tag{32}$$

where $w_C = 1/|\mathbf{d}_C|$ and $\mathbf{G}$ is a 3×3 symmetric matrix

$$\mathbf{G} = \sum_C w_C^2 \mathbf{d}_C \mathbf{d}_C^T. \tag{33}$$

Assembling Eq. (25) using Eqs. (26)–(29), (31) and (32) produces

the following:

$$\bar{a}_P \mathbf{D}_P + \sum_C \bar{a}_C \mathbf{D}_C = \mathbf{r}_P, \tag{34}$$

where $\bar{a}_P$ and $\mathbf{r}_P$ include the contributions from the temporal term and the boundary conditions. $\mathbf{D}_P$ depends on the values in the neighboring cells.

The Navier displacement has been discretized using the method developed in [2] in the following way:

$$\rho_s \frac{\partial^2 \mathbf{D}}{\partial t^2} - \underbrace{\nabla\cdot(\mu\nabla\mathbf{D})}_{\text{implicit}} - \underbrace{\nabla\cdot[\mu(\nabla\mathbf{D})^T + \lambda\mathbf{I}\mathrm{tr}(\nabla\mathbf{D})]}_{\text{explicit}} = \rho_s \mathbf{f}_s. \tag{35}$$

Unfortunately, this split is only marginally convergent because the explicit terms carry more informations than their implicit counterparts. The solution proposed by Jasak et al. [2] is to rewrite Eq. (35) as

$$\rho_s \frac{\partial^2 \mathbf{D}}{\partial t^2} - \underbrace{\nabla\cdot[(2\mu+\lambda)\nabla\mathbf{D}]}_{\text{implicit}} - \underbrace{\nabla\cdot[\mu(\nabla\mathbf{D})^T + \lambda\mathbf{I}\mathrm{tr}(\nabla\mathbf{D}) - (\mu+\lambda)\nabla\mathbf{D}]}_{\text{explicit}} = \rho_s \mathbf{f}_s. \tag{36}$$

The discretization matrix has been "over-relaxed" and the convergence speed of the method is improved. In this case, μ is changed in $2\mu + \lambda$ in (30); thus the entries of the linear system to solve are given by:

$$a_C = -(2\mu+\lambda)\frac{|\mathbf{n}S_c|}{|\mathbf{d}_C|} \text{ and } a_P = -\sum_C a_C \tag{37}$$

thus creating the system of algebraic equations:

$$[A][\mathbf{D}] = [\mathbf{r}], \tag{38}$$

where $[A]$ is a sparse matrix, with coefficients a_P on the diagonal and a_C off the diagonal, $[\mathbf{D}]$ is the vector of $\mathbf{D}$ for all CVs and $[\mathbf{r}]$ is the right-hand side vector. Note that the matrix $[A]$ is symmetric with $a_{i,i} > 0$ and $a_{i,j} \leq 0$ $(i \neq j)$, $[A]$ is strictly diagonally dominant, since the diagonal elements of the spatial discretization matrix are perturbed by $\rho_s/(\delta t)^2$, where δt is the time step. The fact that the matrix is symmetric will induce in the forthcoming paragraph 4.4, Algorithm 3, the choice of a conjugate gradient algorithm preconditioned by Choleski incomplete revisited.

3.3. Finite volume method

We saw that the PISO method and Jasak et al.'s method consist in solving convection–diffusion and diffusion equations. In the sequel we will consider the finite volume method for discretization.

The finite volume method (FVM) requires the decomposition of the physical domain into a number of control volumes (CV) also called cells, defined by a numerical grid [1]. The governing equations are integrated over each CV; the terms appearing after integration are approximated with finite-difference type relations. This process yields a system of N algebraic equations, where N is the number of central nodes of each CV. One of the advantages of this type of discretization is that it ensures the conservation of variables over each CV. Fig. 1 represents a three-dimensional control volume.

3.3.1. Discretization of convection diffusion equations

The discretization of a steady convection–diffusion equation of a variable ϕ results in an algebraic equation of the form:

$$a_P\phi_P + \sum_C a_C\phi_C = b_P, \tag{39}$$

where the index C runs over all neighboring points that are involved in the approximation as a result of the discretization scheme employed and:

$$a_P = -\sum_C a_C$$

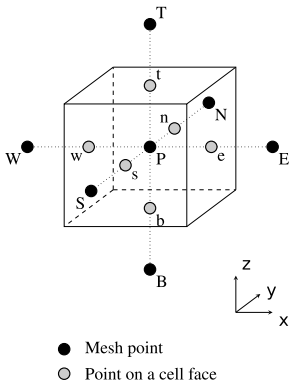


Fig. 1. Mesh points around a control volume.

denotes the conservative property of the finite volume method.

The coefficients are given by [28]:

$$\begin{cases} a_E = -\dfrac{\Gamma}{(x_E - x_P)(x_e - x_w)}, & a_W = -\dfrac{\rho_f v_1}{x_e - x_w} - \dfrac{\Gamma}{(x_P - x_W)(x_e - x_w)}, \\ a_N = -\dfrac{\Gamma}{(y_N - y_P)(y_n - y_s)}, & a_S = -\dfrac{\rho_f v_2}{y_n - y_s} - \dfrac{\Gamma}{(y_P - y_S)(y_n - y_s)}, \\ a_T = -\dfrac{\Gamma}{(z_T - z_P)(z_t - z_b)}, & a_B = -\dfrac{\rho_f v_3}{z_t - z_b} - \dfrac{\Gamma}{(z_P - z_B)(z_t - z_b)}, \\ a_P = \dfrac{\rho_f v_1}{x_e - x_w} + \dfrac{\Gamma(x_E - x_W)}{(x_P - x_W)(x_E - x_P)(x_e - x_w)} + \dfrac{\rho_f v_2}{y_n - y_s} \\ \quad + \dfrac{\Gamma(y_N - y_S)}{(y_P - y_S)(y_N - y_P)(y_n - y_s)} \\ + \dfrac{\rho_f v_3}{z_t - z_b} + \dfrac{\Gamma(z_T - z_B)}{(z_P - z_B)(z_T - z_P)(z_t - z_b)}, \end{cases} \tag{40}$$

where x, y and z are the coordinates of each point involved and Γ is a diffusion coefficient. Note that the coefficients a_p are positive and the coefficients a_C are negative. Therefore, the discretization matrix is diagonally dominant : it's an M-matrix.

3.4. Fluid-structure coupling procedure

The coupling between the two sub-domains is done by a transfer of variables. Pressures and viscous forces (shear stress) at the fluid side of the fluid-structure interface are transferred to the solid side of the interface. Whereas, displacements and velocities at the solid side are transferred to the fluid side of the interface.

3.4.1. FSI algorithm

The general procedure for solving a transient fluid-structure interaction problem is shown in Algorithm 2, where t_f is the final time step.

```
for t = 1 to t_f do
    while FSI is not converged do
        update mesh
        solve Navier–Stokes on Ω_f
        solve Navier on Ω_s
        compute mesh motion
    end while
end do
```

Algorithm 2. The general FSI procedure.

3.4.2. Forces on the interface

The solution of the Navier–Stokes equations yields the fluid velocities and pressure at each fluid node; they are used to calculate the tractions which apply on the solid boundary. These are then used as boundary conditions in the solid solver as fixed tractions and fixed pressures. Two forces are acting on the fluid-structure interface : wall pressure and wall shear stress.

Wall pressure. The wall pressure due to the fluid acts normal to the solid surface. It is directly available from the solution of the fluid flow, the pressure values at the cell faces constitutive of the fluid-structure interface are directly transferred between the fluid and the solid. This is one of the benefits of using the same discretization method on both domains.

Wall shear stress. The wall shear stress acts tangentially to the solid surface. The no-slip condition at a wall dictates that the speed of the fluid relative to the boundary is zero. However, at some distance from the boundary, the flow speed must be equal to that of the rest of the fluid; the region between those two points is called the boundary layer.

For a Newtonian fluid, the shear stress is proportional to the strain rate in the fluid and the dynamic viscosity μ_f is the constant of proportionality. The shear stress is imparted onto the wall surface as a consequence of the loss of velocity. The shear stress at the wall τ_{wall} is given by:

$$\tau_{wall} = \mu_f \left.\frac{\partial u_t}{\partial x_n}\right|_{wall}, \tag{41}$$

where u_t is the wall tangential velocity and x_n is the wall normal coordinate.

3.5. Mesh adaptation

The solution of the Navier equations of elasticity leads to the displacement vector **D** being known at each node in the solid domain and therefore, these displacement can be added to the nodal coordinates in order to determine the new geometry shape. For the fluid solver to take into account this deformation, the fluid sub-domain mesh must be adapted. This is achieved by considering the mesh as a solid body.

A Laplacian equation for the domain velocity $\hat{\mathbf{u}}$:

$$\nabla^2(\Gamma_f \hat{\mathbf{u}}) = 0, \tag{42}$$

with Γ_f a diffusive coefficient, is therefore solved on the fluid sub-domain with the calculated displacement along the fluid-structure interface as a boundary condition. Zero displacement conditions are applied to the other boundaries of the sub-domain. The properties Γ_f of the constitutive "material" of the mesh are usually the same as those in the solid sub-domain. The calculated displacement in the fluid sub-domain are then added to the nodal coordinates to produce the deformed mesh. The mesh is deformed during the course of the fluid-structure simulation while keeping the same node distribution.

4. Parallel numerical solution for FSI

In order to numerically solve the fluid-structure interaction problem, we saw in paragraph 3 that we need to solve 7 linear systems at each time-step, namely the linear algebraic systems (18), (21), (24), (38) and (42) for the domain velocity $\hat{\mathbf{u}}$. In paragraph 3, we noted that the matrices involved in the linear systems, symmetrical or not, are M-matrices. This property is interesting and plays an important part algorithmically in what follows. This is why, in the sequel, we present an original result for the solution of linear systems by multi-splitting methods, avoiding the difficult formal decomposition in sub-domains. Note that however, this analysis is also appropriate for single valued or multivalued (when constraints are exerced on the solution) pseudo-linear problems.

The main contributions developed in the present section are precised below:

- in paragraphs 4.1 and 4.2, for linear problems, if the matrix of the linear system is an M-matrix, we analyze the convergence of asynchronous parallel general methods for the FSI application regardless of the decomposition of the problem into large blocks. Particularly in paragraph 4.2.2, under suitable assumptions, we show a contraction result for fixed point applications associated with a realistic decomposition in large blocks in parallel computation which makes it possible to apply and extend a result from Bahi et al. [22] concerning the multi-splitting method;
- in paragraph 4.3, we use the multi-splitting method, which avoid going through effective sub-domain decomposition in an unified context using the classic Hilbertian or non-Hilbertian norms that are more convenient to use (cf. Remark 8); the Jacobi asynchronous block method appearing thus as a particular case;
- the results are extended to the case of univalued or multivalued pseudo-linear problems;
- as previously said, concerning the FSI application, we consider in this paper a global discretization of the fluid and the structure domains by the finite volume method, ensuring the property of M-matrices in the discretized systems. Another consequence is the coherence in the numbering of grid points which limits the approximation errors when one passes from the points in the fluid sub-domain to the structure sub-domain.

4.1. Preliminaries: asynchronous iterations

Let us first consider the solution $\tilde{X}^*$ of the following linear system

$$A\tilde{X}^* = \tilde{B}, \tag{43}$$

where A results from the discretization of the Navier–Stokes equations or of the Navier equation or Eq. (42) arising in the fluid-structure coupling and is any M-matrix.

Remark 1. In system (43), we note $\tilde{X}$ the unknown of the problem, this notation will be kept in paragraphs 4.1 and 4.2. On the other hand, in paragraph 4.3 we will have to introduce m vectors $\tilde{X}$ to define a multi-vector X arising in the multi-splitting method.

Let us denote by $M \in \mathbb{N}$ the size of the matrix A. Let also $\mathcal{E} = \mathbb{R}^M$ be a Banach space normed by a non-Hilbertian norm and α be another positive integer, corresponding to the number of blocks associated to the natural decomposition, such that the space $\mathcal{E} = \prod_{i=1}^{\alpha} \mathcal{E}_i$ is decomposed in a finite product of α sub-spaces denoted $\mathcal{E}_i = \mathbb{R}^{m_i}$, with $\sum_{i=1}^{\alpha} m_i = M$, where m_i denotes the size of the $i^{th}-$ space $\mathcal{E}_i$; note that $\mathcal{E}_i$ is also a Banach space.

In the sequel, we consider the general following fixed point problem associated to Eq. (43)

$$\begin{cases} \text{Find } \tilde{X}^* \in \mathcal{E} \text{ such that} \\ \tilde{X}^* = F(\tilde{X}^*). \end{cases} \tag{44}$$

where F: $\mathcal{D}(F) \subset \mathcal{E} \mapsto \mathcal{D}(F)$, $\tilde{X} \mapsto F(\tilde{X})$ is a fixed point mapping associated to a particular splitting of the matrix A. To be interesting from an algorithmic point of view, such fixed point mapping must be contractive; thus, according to analogous works in the literature [18,22], we will consider in the sequel a regular splitting of matrix A defined by $A = \Delta - (L + R)$, where in our case Δ^{-1} and L and R are non-negative matrices if A is an M-matrix. Particularly, in the following sub-sections, we will show how the mapping F is related to the problem (43) to solve. According to the decomposition of $\mathcal{E}$, let us consider the corresponding $\alpha-$ block decomposition of F and $\tilde{Y}$

$$\begin{aligned} \tilde{Y} &= (\tilde{Y}_1, ..., \tilde{Y}_\alpha), \\ F(\tilde{Y}) &= (F_1(\tilde{Y}), ..., F_\alpha(\tilde{Y})), \end{aligned}$$

where for all $\tilde{Y} \in \mathcal{E}$, we denote for $l \in \{1, ...,\alpha\}$ by $\tilde{Y}_l \in \mathcal{E}_l$ the associated block components of $\tilde{Y}$.

In order to solve the fixed point problem (44), let us now consider the parallel asynchronous iterations defined as follows: let $\tilde{X}^0 \in \mathcal{E}$ be given; then for all $r \in \mathbb{N}$, $\tilde{X}^{r+1}$ is recursively defined by

$$\tilde{X}_l^{(r+1)} = \begin{cases} F_l(\tilde{X}_1^{(\kappa_1(r))},...,\tilde{X}_k^{(\kappa_k(r))},...,\tilde{X}_\alpha^{(\kappa_\alpha(r))}) \text{ if } l \in s(r), \\ \tilde{X}_l^r \text{ if } l \notin s(r), \end{cases} \tag{45}$$

where

$$\begin{cases} \forall\, r \in \mathbb{N},\ s(r) \subset \{1,...,\alpha\} \text{ and } s(r) \neq \varnothing, \\ \forall\, l \in \{1,...,\alpha\},\ \{r \mid l \in s(r)\} \text{ is countable}, \end{cases} \tag{46}$$

and $\forall\, k \in \{1, ...,\alpha\}$,

$$\begin{cases} \forall\, r \in \mathbb{N},\ \kappa_k(r) \in \mathbb{N},\ 0 \leq \kappa_k(r) \leq r, \\ \kappa_k(r) = r \text{ if } k \in s(r) \text{ and } \lim_{r\to\infty} \kappa_k(r) = +\infty. \end{cases} \tag{47}$$

The previous asynchronous iterative scheme models computations that are carried out in parallel without order nor synchronization; such algorithms describe a sub-domain method. Particularly, it allows one to consider distributed computations whereby processors go at their own pace according to their intrinsic characteristics and computational load. The parallelism between the processors is well described by the set $s(r)$ which contains the number of components relaxed by each processor on a parallel way at each relaxation while the use of delayed components in (45) allows one to model nondeterministic behavior and does not imply inefficiency of the considered distributed scheme of computation. Note that, theoretically, each component of the vector must be relaxed an infinity of time. The choice of the relaxed components may be guided by any criterion, and, in particular, a natural criterion is to pick-up the most recently available values of the components computed by the processors.

Remark 2. Such asynchronous iterations describe various classes of parallel algorithms, such as parallel synchronous iterations if $\forall\, k \in \{1, ...,\alpha\}$, $\forall\, r \in \mathbb{N}$, $\kappa_k(r) = r$. In the synchronous context, for a particular choice of $s(r)$, then (45)–(47) describe classical sequential algorithms; among them, the Jacobi method if $\forall\, r \in \mathbb{N}$, $s(r) = \{1, ...,\alpha\}$ and the Gauss–Seidel method if $\forall\, r \in \mathbb{N}$, $s(r) = \{1 + r \bmod \alpha\}$ (see [5]).

4.2. Notion of M-minorants, a powerful tool

4.2.1. Basic definitions

Let us consider the solution of problem (43) and consider the decomposition of the matrix A into α blocks; by using the same notations than the ones used in Section 4.1, this block decomposition is defined as follows

$$A_{l,k} \in \mathcal{L}(\mathcal{E}_k; \mathcal{E}_l), \text{ with } \sum_{l=1}^{\alpha} m_l = M.$$

Moreover, for $l = 1, \cdots,\alpha$, let us denote by $< .,. >_l$ the pairing between $\mathcal{E}_l$ and $\mathcal{E}_l'$ its topological dual space; let $|.|_l$ be the associated norm in $\mathcal{E}_l$, such that $G_l(\tilde{X}_l)$ is the duality map, we have

$$G_l(\tilde{X}_l) = \{g_l(\tilde{X}_l) \in \mathcal{E}_l' \mid <\tilde{X}_l, g_l>_l = |\tilde{X}_l|_l \text{ and } |g_l|_l^* = 1\},$$

where $|g_l|_l^*$ denotes the norm of $g_l \in \mathcal{E}_l'$.

Remark 3. Recall that the pairing between $\mathcal{E}$ and $\mathcal{E}'$ is a bilinear form, from $\mathcal{E} \times \mathcal{E}'$ onto $\mathbb{R}$. If $\mathcal{E}$ is an Hilbert space, then the pairing is the inner product of $\mathcal{E}$.

Remark 4. Even if, in finite dimension spaces, all Holder norms are mathematically equivalent, when the space dimension M becomes large the measurement of any distance ξ between two elements using these norms poses, numerically, some coherence problems. For example, if using the uniform norm, one has the condition $|\xi|_\infty \leq \varepsilon$, as classically for the Euclidean norm we have $|\xi|_2 \leq M^{1/2}|\xi|_\infty$, then $|\xi|_2 \leq M^{1/2}\varepsilon$. Similarly, when using the l_1–norm, we will obtain $|\xi|_1 \leq M|\xi|_\infty \leq M\varepsilon$. Therefore, even if the uniform norm leads to an acceptable proximity result, the distances measured by the l_1–norm or the Euclidian norm can be substantial. For this reason and for sake of generality, in the sequel we will use Hilbertian or non-Hilbertian norms.

Then we can recall the notions of L-minorant (or Z-minorant) and M-minorant (see [13,29–36]) as follows

Definition 1. The matrix $A \in \mathcal{L}(\mathcal{E})$ admits an L-minorant (or Z-minorant) denoted $N(A)$ with respect on one hand to the above α–block decomposition and on the other hand with respect of the norms defined in $\mathcal{E}_l$, $l = 1, ...,\alpha$ if and only if $\exists\, \eta_{l,l} > 0$ such that

$$< A_{l,l}\tilde{X}_l, g_l >_l \geq \eta_{l,l} |\tilde{X}_l|_l \ , \ \forall\, \tilde{X}_l \in \mathcal{E}_l,\ l = 1, ...,\alpha, \tag{48}$$

and for all $l, k \in \{1, ...,\alpha\}$, $l \neq k$, $\forall\, \tilde{X}_l \in \mathcal{E}_l$, $\forall\, \tilde{X}_k \in \mathcal{E}_k$, $\exists\, \eta_{l,k} \geq 0$ such that

$$|<A_{l,k}\tilde{X}_k, g_l>_l| \leq \eta_{l,k} |\tilde{X}_k|_k, \tag{49}$$

where $N(A)$ is defined by

$$N(A) = \begin{pmatrix} \eta_{1,1} & & \cdots & \cdots & -\eta_{1,\alpha} \\ \vdots & \ddots & & & \\ \vdots & & \ddots & & \\ & -\eta_{l,k} & \eta_{l,l} & \cdots & \\ \vdots & & & \ddots & \\ \vdots & & & & \ddots \\ -\eta_{\alpha,1} & & \cdots & \cdots & \eta_{\alpha,\alpha} \end{pmatrix}.$$

The L-minorant $N(A)$ is said to be an M-minorant of A if and only if $N(A)$ is an M-matrix.

Definition 2. A fixed point mapping F: $\mathcal{D}(F) \subset \mathcal{E} \mapsto \mathcal{D}(F)$ associated to the algebraic linear system $A\tilde{X} = \tilde{B}$, is said to admit the majorant matrix $J_\alpha \in \mathcal{L}(\mathbb{R}^\alpha)$, where J_α is a non-negative matrix ($J_\alpha \geq 0$) with respect to the α–block decomposition of A and to the norm defined in $\mathcal{E}_l$ if and only if for $l = 1, ...,\alpha$ and for all $\tilde{X}, \tilde{X}' \in \mathcal{E}$

$$|F_l(\tilde{X}_1,...,\tilde{X}_k,...,\tilde{X}_\alpha) - F_l(\tilde{X}_1',...,\tilde{X}_k',...,\tilde{X}_\alpha')|_l \leq \sum_{\substack{k=1 \\ k\neq l}}^{\alpha} \frac{\eta_{l,k}}{\eta_{l,l}} |\tilde{X}_k - \tilde{X}_k'|_k, \tag{50}$$

with

$$\tilde{X} = \{\tilde{X}_1,...,\tilde{X}_k...,\tilde{X}_\alpha\} \in \mathcal{E},$$

and

$$\tilde{X}' = \{\tilde{X}_1',...,\tilde{X}_k',...,\tilde{X}_\alpha'\} \in \mathcal{E},$$

where $\tilde{X}_k$ and $\tilde{X}_k'$, $k = 1, ...,\alpha$, are the block-components of X and X' and the matrix J_α associated with the L-minorant $N(A)$ is defined by

$$J_\alpha = J_{N(A)} = \begin{pmatrix} 0 & \frac{\eta_{1,2}}{\eta_{1,1}} & & \cdots & \cdots & \frac{\eta_{1,\alpha}}{\eta_{1,1}} \\ \frac{\eta_{2,1}}{\eta_{2,2}} & \ddots & & & & \vdots \\ \vdots & & \ddots & & & \vdots \\ & \frac{\eta_{l,k}}{\eta_{l,l}} & 0 & \cdots & \cdots & \vdots \\ \vdots & \cdots & \cdots & & \ddots & \vdots \\ \vdots & \cdots & \cdots & & & \vdots \\ \frac{\eta_{\alpha,1}}{\eta_{\alpha,\alpha}} & \cdots & \cdots & \cdots & \cdots & 0 \end{pmatrix}.$$

Definition 3. If in addition $N(A)$ is an M-minorant, then the spectral radius of J_α is less than one, and J_α is a contraction matrix for the α–decomposition in $\mathcal{E}$ normed by any norm.

Thus, in the framework of the α–block decomposition, let us

consider the vectorial norm $q(.)$ defined on $\mathcal{E} = \prod_{l=1}^{\alpha} \mathcal{E}_l$ by

$$q(\tilde{X}) = \{|\tilde{X}_1|_1, ..., |\tilde{X}_\alpha|_\alpha\}. \tag{51}$$

Then, using this notation, (50) is written as follows

$$q(F(\tilde{X}) - F(\tilde{X}')) \leq J_\alpha \cdot q(\tilde{X} - \tilde{X}'). \tag{52}$$

Remark 5. The notion of vectorial norm corresponds to the introduction of a mathematical tool well adapted to the decomposition of the global problem into interconnected sub-problems. Given the large size of the problems to solve and the sparse nature of the matrices, parallel iterative methods are well suited for the solution of the algebraic systems involved. Thus, the vectorial norms make it possible to analyze the behavior of these parallel iterative methods; this mathematical tool, inseparable from the notion of minorante, has been used by many authors such as [5,13,30,32,33,35]. The condition (52) corresponds to a vector Lipschitz condition that becomes a vector contraction property with respect to the vectorial norm if J_α is a non-negative matrix with a spectral radius strictly less than one.

Remark 6. Clearly for the $\alpha-$block decomposition $J_\alpha = J_{N(A)}$ is the Jacobi's matrix of $N(A)$.

Remark 7. Note also that, instead of the $\alpha-$block decomposition, we can consider also a point by point decomposition; then if A is an M-matrix it is always its own M-minorant with respect to the "by-point"-decomposition, which corresponds to the limit case in which we take

$$\begin{cases} A_{i,j} = (a_{i,j}) \in \mathcal{L}(\bar{\mathcal{E}}_i; \bar{\mathcal{E}}_j) \text{ where } \bar{\mathcal{E}}_i \equiv \bar{\mathcal{E}}_j \equiv \mathbb{R} \\ \text{with in this case } i \equiv l;\ j \equiv k;\ i, j \in \{1, ..., M\}. \end{cases}$$

Remark 8. Note that the spaces $\mathcal{E}_l$, $l = 1, ..., \alpha$, can be normed by all classical norms, such as the Euclidean norm but also by the uniform norm or by the l_1-norm. In this two last topological contexts, i.e. when $\mathcal{E}_l$ is normed by non-Hilbertian norms, we have to use more sophisticated mathematical tools. Indeed, in a Hilbertian context, note that $\tilde{X}$ is the Frechet derivative of the Euclidean norm $\frac{1}{2}|\tilde{X}|_2^2$ and the pairing between $\mathcal{E}$ and its topological dual space is replaced by the inner product more commonly used. When the space is normed by the uniform norm or by the l_1-norm, since the mapping $z \mapsto |z|$, the absolute value, $\forall\, z \in \mathbb{R}$, is not differentiable, for g_l we have to consider an element of the sub-differential of $|\tilde{X}|_\infty$ or $|\tilde{X}|_1$, respectively, instead of $\tilde{X}$. Such situation is more complex from a mathematical point of view but leads to practical criteria easier to compute for the diagonal entries $\eta_{l,l}$, $l = 1, ..., \alpha$ of the L-minorant. Indeed, note that the uniform norm and the l_1-norm, are dual norms each other. According to the results developed in [14], when in each case, the diagonal block matrices $A_{l,l}$ are strictly diagonal dominant matrices and have also strictly diagonal entries greater than $\eta_{l,l}$, then $\eta_{l,l} = a_{l,l} - \sum_{k \neq l} |a_{l,k}|$, $\eta_{l,l} > 0$, where $a_{l,k}$ denote the off-diagonal entries of the diagonal sub-matrices $A_{l,l}$. The computation of $\eta_{l,k}$, $k \neq l$ is performed by the computation of the matricial norms of the off-diagonal blocks $A_{l,k}$, $k \neq l$ with respect to the subordinate norm used in $\mathcal{E}_k$, and such computation of matricial norms are achieved by adding the absolute values of the entries of the row or the column of $A_{l,k}$ according to the choice of the non-Hilbertian norm. By this way, we can compute all the entries of the L-minorant. On the contrary, if the spaces $\mathcal{E}_l$, $l = 1, ..., \alpha$, are normed by the Euclidean norm, then the diagonal entries of the L-minorant, are the smaller eigenvalues of the block diagonal sub-matrices $A_{l,l}$, $l = 1, ..., \alpha$, computed by the reverse power method, while the computation of the matrix norms of the block off-diagonal needs the use of the power method; such computations are very expensive compared to the computation using non-Hilbertian norms.

Let us now denote by $\bar{\eta}_{l,l} = \eta_{l,l}$ and $\bar{\eta}_{l,k} = -\eta_{l,k}$, $l \neq k$; then we have the equivalent property of a L-minorant for the $\alpha-$block decomposition of the matrix A, constituting in fact a characterization of the L-minorant.

Proposition 1. *Both assumptions (48) and (49) are equivalent to*

$$< \sum_{k=1}^{\alpha} A_{l,k}\tilde{X}_k, g_l >_l \geq \sum_{k=1}^{\alpha} \bar{\eta}_{l,k} |\tilde{X}_k|_k \ , \ \forall\, l \in \{1, ..., \alpha\}. \tag{53}$$

Proof. We can verify by a direct way that (48) and (49) involve (53); indeed

$$< A_{l,l}\tilde{X}_l, g_l >_l \geq \eta_{l,l} |\tilde{X}_l|_l \ , \ \forall\, \tilde{X}_l \in \mathcal{E}_l,\ l = 1, ..., \alpha,$$

and, for $l, k = 1, ..., \alpha,\ l \neq k$

$$-\eta_{l,k}|\tilde{X}_k|_k \leq |< A_{l,k}\tilde{X}_k, g_l >_l| \leq \eta_{l,k}|\tilde{X}_k|_k;$$

then, by summing, one obtains (53). For the reciprocal one refers to [15]. □

Remark 9. Let us consider the solution of problem (43) and consider an $\alpha-$block decomposition of the matrix A; then, with such decomposition, we can write for all $l \in \{1, ..., \alpha\}$

$$< \sum_{k=1}^{\alpha} A_{l,k}(\tilde{X}_k - \tilde{X}'_k), g_l(\tilde{X}_k - \tilde{X}'_k) >_l = 0. \tag{54}$$

The vectorial norm $q(.)$ associated to the $\alpha-$block decomposition, being defined by (51), then, if the matrix A admits an M-minorant, thanks to (53) and (54), we obtain obviously by a direct way a property of contraction with respect of the vectorial norm $q(.)$ like (52).

4.2.2. A general convergence result for a decomposition in large blocks

In fact, in parallel computation, the user does not access to as many processors as components. Practically, the algebraic system to solve is split into β contiguous blocks, $\beta \leq \alpha$ (in fact $\beta \ll \alpha$), corresponding to a coarser sub-domain decomposition of the problem to solve; thus the communications consist in exchanging only the values of the components computed by the neighboring processors. Under appropriate assumptions, such situation, corresponding to consider for a given decomposition a coarser decomposition, has been studied in [15] where it was proved that if the fixed point mapping F is contractive for a given decomposition, then the fixed point mapping $\hat{F}$ is also contractive for a coarser decomposition; then, in both cases the parallel asynchronous iterations converge.

So, in the sequel, we will consider that $N(A) \in \mathcal{L}(\mathbb{R}^\alpha)$ is an M-minorant, so that the fixed point mapping F associated to the decomposition into α blocks is contractive according to (50) or (52) and we will show that, in this context, the fixed point mapping $\hat{F}$ associated to the coarser decomposition in β blocks, $\beta \leq \alpha$ is also contractive.

$$\begin{cases} \text{Let } \beta \in \mathbb{N} \text{ and } \{\alpha_i\} \text{ for } i \in \{1, ..., \beta\} \\ \text{a set of integers such that } \alpha_i \neq 0 \text{ and } \sum_{i=1}^{\beta} \alpha_i = \alpha, \end{cases}$$

where, in fact, for $i = 1, ..., \beta$, α_i will denote the number of blocks of the finer decomposition, i.e. the $\alpha-$decomposition, these previous blocks being gathered for defining the $i-th$ large block of the coarser decomposition, i.e. the $\beta-$decomposition.

$$\begin{cases} \text{Let } \forall\, i \in \{2, ..., \beta + 1\},\ \beta_i = \sum_{j=1}^{i-1} \alpha_j \text{ such that } \beta_1 = 0 \\ \text{and } \beta_{\beta+1} = \alpha, \text{ and } \forall\, i \in \{1, ..., \beta\},\ \hat{\mathcal{E}}_i = \prod_{l=\beta_i+1}^{\beta_{i+1}} \mathcal{E}_l, \end{cases}$$

where the ith large block of the $\beta-$decomposition is constituted by gathering the $(\beta_i + 1)$th block until the β_{i+1}th block. Obviously

$$\mathcal{E} = \prod_{i=1}^{\beta} \hat{\mathcal{E}}_i = \prod_{i=1}^{\alpha} \mathcal{E}_i,$$

and

$$\tilde{X} = \left\{\hat{\tilde{X}}_1, ..., \hat{\tilde{X}}_\beta\right\} \in \prod_{i=1}^{\beta} \hat{\mathcal{E}}_i,$$

where $\hat{\tilde{X}}_i, i = 1, ..., \beta$ denote the block components of $\tilde{X}$ expressed in the $\beta-$decomposition.

Remark 10. The considered block hierarchy makes it possible to show that under suitable assumptions, verified if the discretization matrix is an M-matrix, which is the case for linear systems (18), (21), (24), (38), (42) the fixed point applications $\hat{F}$ associated with the decomposition into large blocks are contractive regardless of the decomposition into large blocks considered. This result is set out below in Proposition 2 and Corollary 1. This then makes it possible to apply and usefully extend a result of Bahi et al. [22] for the analysis of multi-splitting methods.

Let us denote by Λ: $D(\Lambda) \subset \mathcal{E} \mapsto \mathcal{E}$ the following mapping defined by

$$\Lambda_l(\tilde{X}_1, ..., \tilde{X}_{l-1}, \tilde{X}_l, \tilde{X}_{l+1}, ..., \tilde{X}_\alpha) = \sum_{k=1}^{\alpha} A_{l,k}\tilde{X}_k, \tag{55}$$

such that we can write the initial problem (43) as follows, $\forall\, l \in \{1, ..., \alpha\}$,

$$\Lambda_l(\tilde{X}_1, ..., \tilde{X}_{l-1}, \tilde{X}_l, \tilde{X}_{l+1}, ..., \tilde{X}_\alpha) - \tilde{B}_l = 0.$$

A coarser decomposition of this previous problem (43) is defined by, $\forall\, i \in \{1, ..., \beta\}$,

$$\hat{\Lambda}_i(\hat{\tilde{X}}_1, ..., \hat{\tilde{X}}_{i-1}, \hat{\tilde{X}}_i, \hat{\tilde{X}}_{i+1}, ..., \hat{\tilde{X}}_\beta) - \hat{\tilde{B}}_i = 0,$$

where, $\forall\, i \in \{1, ..., \beta\}$,

$$\hat{\Lambda}_i(\hat{\tilde{X}}_1, ..., \hat{\tilde{X}}_{i-1}, \hat{\tilde{X}}_i, \hat{\tilde{X}}_{i+1}, ..., \hat{\tilde{X}}_\beta) = \sum_{j=1}^{\beta} \hat{A}_{i,j}\hat{\tilde{X}}_j,$$

the matrices $\hat{A}_{i,j}, i, j = 1, ..., \beta$ being obtained by grouping in a suitable manner the blocks $A_{l,k}$ of the $\alpha-$block decomposition. Then, now we consider the decomposition of problem (43) into β sub-problems, $\forall\, i \in \{1, ..., \beta\}$,

$$\hat{\Lambda}_i(\hat{\tilde{Y}}_1, ..., \hat{\tilde{Y}}_{i-1}, \hat{\tilde{X}}_i, \hat{\tilde{Y}}_{i+1}, ..., \hat{\tilde{Y}}_\beta) - \hat{\tilde{B}}_i = 0,$$

where, to simplify the notations, $\hat{\tilde{Y}}_j$ for $j \neq i$, $j = 1, ..., \beta$ represent the delayed interaction values defined by $\hat{\tilde{Y}}_j = \hat{\tilde{X}}_j^{(kj(r))}$ and where, in such decomposition $\hat{\tilde{X}}_i$ is defined by

$$\hat{\tilde{X}}_i = \{..., \tilde{X}_l, ...\}, \text{ for } l \in \{\beta_i + 1, ..., \beta_{i+1}\}.$$

Consider also the same problem for a distinct vector $\tilde{X}'$, so that, $\forall\, i \in \{1, ..., \beta\}$,

$$\hat{\Lambda}_i(\hat{\tilde{Y}}'_1, ..., \hat{\tilde{Y}}'_{i-1}, \hat{\tilde{X}}'_i, \hat{\tilde{Y}}'_{i+1}, ..., \hat{\tilde{Y}}'_\beta) - \hat{\tilde{B}}_i = 0,$$

where similarly

$$\hat{\tilde{X}}'_i = \{..., \tilde{X}'_l, ...\}, \text{ for } l \in \{\beta_i + 1, ..., \beta_{i+1}\}.$$

Then, starting from the $\beta-$decomposition, in fact the coarser decomposition, we can reformulate the same problem in the $\alpha-$decomposition, the finer decomposition; by subtracting and multiplying by $g_l(\tilde{X}_l - \tilde{X}'_l)$, we obtain, for $l \in \{\beta_i + 1, ..., \beta_{i+1}\}$,

$$< \Lambda_l(\tilde{Y}_1, ..., \tilde{Y}_{l-1}, \tilde{X}_l, \tilde{Y}_{l+1}, ..., \tilde{Y}_\alpha) - \Lambda_l(\tilde{Y}'_1, ..., \tilde{Y}'_{l-1}, \tilde{X}'_l, \tilde{Y}'_{l+1}, ..., \tilde{Y}'_\alpha), g_l(\tilde{X}_l - \tilde{X}'_l) >_l = 0,$$

and since (53) is verified, then

$$\left(\sum_{k=1}^{\beta_i} \bar{\eta}_{l,k}|\tilde{Y}_k - \tilde{Y}'_k|_k + \sum_{k=\beta_i+1}^{\beta_{i+1}} \bar{\eta}_{l,k}|\tilde{X}_k - \tilde{X}'_k|_k + \sum_{k=\beta_{i+1}+1}^{\alpha} \bar{\eta}_{l,k}|\tilde{Y}_k - \tilde{Y}'_k|_k\right) \leq 0;$$

so, due to the fact that $\bar{\eta}_{l,k} \leq 0$, for $k \neq l$ we obtain, for $l \in \{\beta_i + 1, ..., \beta_{i+1}\}$

$$\sum_{k=\beta_i+1}^{\beta_{i+1}} \bar{\eta}_{l,k}|\tilde{X}_k - \tilde{X}'_k|_k \leq \left(-\sum_{k=1}^{\beta_i} \bar{\eta}_{l,k}|\tilde{Y}_k - \tilde{Y}'_k|_k - \sum_{k=\beta_{i+1}+1}^{\alpha} \bar{\eta}_{l,k}|\tilde{Y}_k - \tilde{Y}'_k|_k\right). \tag{56}$$

Let us consider a block splitting of the M-minorant $N(A) = \Delta - (L + R)$ into β blocks $\hat{N}(A)_{i,j}$, $i, j = 1, ..., \beta$ such that for $i, j \in \{1, ..., \beta\}$, the entries of the block $\hat{N}(A)_{i,j}$ are $\bar{\eta}_{l,k}$ for $l \in \{\beta_i + 1, ..., \beta_{i+1}\}$ and $k \in \{\beta_j + 1, ..., \beta_{j+1}\}$. So, let us define the block diagonal matrix $\hat{\Delta}$ with diagonal blocks $\hat{N}(A)_{i,i} = (\hat{\Delta}_{i,i})$, by $\hat{\Delta}_{i,i} = (\bar{\eta}_{l,k})$, $l, k \in \{\beta_i + 1, ..., \beta_{i+1}\}$, $i \in \{1, ..., \beta\}$. Let us also define the strictly lower block part $L = (L_{i,j})$ by $\hat{L}_{i,j} = (-\bar{\eta}_{l,k})$, $l \in \{\beta_i + 1, ..., \beta_{i+1}\}$, $k \in \{\beta_j + 1, ..., \beta_{j+1}\}$ for $i \in \{2, ..., \beta\}$ and $j \in \{1, ..., i - 1\}$, i.e. $j < i$ and, similarly, the strictly upper block part $\hat{R} = (\hat{R}_{i,j})$ by $\hat{R}_{i,j} = (-\bar{\eta}_{l,k})$, $l \in \{\beta_i + 1, ..., \beta_{i+1}\}$, $k \in \{\beta_j + 1, ..., \beta_{j+1}\}$, for $i \in \{1, ..., \beta - 1\}$ and $j \in \{i + 1, ..., \beta\}$, i.e. $j > i$; in other words we have

$$\hat{L}_{i,j} = \begin{cases} -\hat{N}(A)_{i,j} \text{ if } i > j \\ 0 \text{ if } i \leq j \end{cases}, \quad \hat{R}_{i,j} = \begin{cases} -\hat{N}(A)_{i,j} \text{ if } i < j \\ 0 \text{ if } j \leq i \end{cases}.$$

Then, the matrix $\hat{\Delta}$ have a size equal to α, we can write the inequality (56) as follows

$$\hat{\Delta}{\cdot}q(\tilde{X} - \tilde{X}') \leq (\hat{L} + \hat{R}){\cdot}q(\tilde{Y} - \tilde{Y}'). \tag{57}$$

Let us recall the following definition (see [37] on pages 56 to 58).

Definition 4. Let $N(A)$, $\hat{\Delta}$, $\hat{L}$, $\hat{R} \in \mathcal{L}(\mathcal{R}^\alpha)$. Then $\hat{\Delta} - (\hat{L} + \hat{R})$ is a regular splitting of $N(A)$ if $\hat{\Delta}$ is invertible, $\hat{\Delta}^{-1} \geq 0$, and $(\hat{L} + \hat{R}) \geq 0$.

Moreover, according to [37] we have the following Lemma.

Lemma 1. *Let $N(A) \in \mathcal{L}(\mathcal{R}^\alpha)$ and assume that $\hat{\Delta} - (\hat{L} + \hat{R})$ is a regular splitting of $N(A)$. Then, the spectral radius $\hat{\delta}$ of $(\hat{\Delta}^{-1}{\cdot}(\hat{L} + \hat{R}))$ satisfies $\hat{\delta} < 1$ if and only if $\hat{\Delta}^{-1}$ exists and is non-negative.*

Since $N(A)$ is an M-matrix, we can apply the previous Lemma; indeed $\hat{\Delta}$ is an M-matrix and $\hat{\Delta} - (\hat{L} + \hat{R})$ is a regular splitting of $N(A)$; then the spectral radius $\hat{\delta}$ of the matrix $\hat{J} = \hat{\Delta}^{-1}(\hat{L} + \hat{R})$ satisfies

$$\hat{\delta} < 1;$$

thus, as $F(\tilde{Y}) \equiv \hat{F}(\hat{\tilde{Y}})$ because we only rewrite the same relations associated to different decompositions, where F is the fixed point mapping associated to the $\alpha-$decomposition and $\hat{F}$ is the fixed point mapping associated to the $\beta-$decomposition, for the splitting corresponding to the coarser decomposition, the block matrix $\hat{J}$ is a contraction matrix with respect to the vectorial norm $q(.)$ and we obtain (52) with $J_\alpha = \hat{J}$. So, we summarize this result as follows.

Proposition 2. *Let us assume that $N(A)$ is an M-minorant for the $\alpha-$block decomposition. Consider a coarser block decomposition in β blocks such that $\beta \leq \alpha$. Then, the inequality (52) is still valid for the $\beta-$decomposition where $\hat{J} = \hat{\Delta}^{-1}(\hat{L} + \hat{R})$ is a contraction matrix.*

Note that the matrix $\hat{J}$ with diagonal entries null and off-diagonal entries equal to $-\frac{\eta_{l,k}}{\eta_{l,l}}$ is the Jacobi's matrix of the matrix $N(A)$. Since the matrix $N(A)$ is an M-matrix, then $\hat{J}$ is a non-negative matrix, and all eigenvalues of $\hat{J}$ have a modulus smaller than one. Let us denote by Θ

the eigenvector associated to the spectral radius $\hat{\delta}$ of $\hat{J}$. Classically by the Perron–Frobenius theorem, all the components of Θ are strictly positive and the following inequality $\hat{J}\Theta \leq \hat{\delta}\Theta, 0 \leq \hat{\delta} < 1$ is valid. Then, by a straightforward way, we obtain, $\forall\ \hat{\hat{Y}}, \hat{\hat{Y}}' \in \mathcal{E}, 0 \leq \hat{\delta} < 1$

$$\|\hat{\hat{F}}(\hat{\hat{X}}) - \hat{\hat{F}}(\hat{\hat{X}}')\|_{\Theta} \leq \hat{\delta}\|\hat{\hat{Y}} - \hat{\hat{Y}}'\|_{\Theta}, \tag{58}$$

where $\|\cdot\|_{\Theta}$, is a uniform weighted norm defined as follows

$$\left\|\hat{\hat{X}}\right\|_{\Theta} = \max_{l=1,\ldots,\alpha}\left(\frac{|\hat{\hat{X}}_l|_l}{\Theta_l}\right). \tag{59}$$

Then, using the last inequality (58), F associated to the α−decomposition and any fixed point mapping associated to the β−decomposition are contractions and, in each case, we obviously obtain a result of existence and uniqueness of both the fixed point of F and of the solution of problem (43). Then, whatever be the initial guess X^0, the convergence of the parallel asynchronous, synchronous and sequential iterations described by (45)–(47), result now either from [7] or [5] associated to the property of contraction with respect to the vectorial norm (52) of the fixed point mappings or directly from the property of contraction with respect to the usual sense (58) by applying the result of [9].

The previous result obtained in Proposition 2 can be usefully applied in the case where the matrix A is an M-matrix and the considered α−decomposition is the point decomposition. Thus, in the sequel, we consider the following assumption

A is an M-matrix. (60)

In this case, according to the Definition 1 and to the Remark 7, clearly, for the point decomposition, the matrix A is its own M-minorant. Thus, by a straightforward and similar way than the one used to prove the result of Proposition 2, we can establish the following result.

Corollary 1. *Consider the problem (43). Assume that the assumption (60) is verified, i.e. A is an M-matrix. Consider any coarser block decomposition in β blocks such that $\beta < M$; let $\hat{\hat{F}}$ be the associated fixed point mapping associated to the β-decomposition. Then, for any block decomposition of X denoted by $\hat{\hat{X}}$, $\hat{\hat{F}}$ satisfies the vectorial norm inequality, $\forall\ \hat{\hat{Y}}, \hat{\hat{Y}}' \in \mathcal{E}$*

$$|\hat{\hat{F}}(\hat{\hat{X}}) - \hat{\hat{F}}(\hat{\hat{X}}')| \leq (\hat{\Delta})^{-1}\cdot(\hat{L} + \hat{R})\cdot|\hat{\hat{Y}} - \hat{\hat{Y}}'|, \tag{61}$$

in which $\hat{J}$ is a contraction matrix for the fixed point mapping $\hat{\hat{F}}$, where $\hat{J}$ is given by

$$\hat{J} = (\hat{\Delta})^{-1}(\hat{L} + \hat{R}),$$

and we have, $\forall\ \hat{\hat{X}}, \hat{\hat{X}}' \in \mathcal{E}$

$$|\hat{\hat{F}}(\hat{\hat{X}}) - \hat{\hat{F}}(\hat{\hat{X}}'))| \leq \hat{J}|\hat{\hat{Y}} - \hat{\hat{Y}}'|. \tag{62}$$

Moreover, $\hat{\hat{F}}$ is also contractive in $\mathcal{E}$ normed by the weighted uniform norm, $\forall\ l = 1, \ldots, M$

$$\left\|\hat{\hat{X}}\right\|_{\Theta} = \max_{l=1,\ldots,M}\left(\frac{|\tilde{x}_l|}{\bar{\theta}_i}\right),\ \bar{\theta} > 0,\ \bar{\theta}_l \in \mathbb{R}. \tag{63}$$

Then, whatever be the initial guess $\tilde{X}^0$, the parallel asynchronous methods associated with any block decomposition converge to $\tilde{X}^{\star}$, fixed point of $\hat{\hat{F}}$.

Remark 11. Note that, the smaller the value of $\hat{\delta}$ is compared to 1, the faster the iterative method converges. Therefore, we can consider that $\hat{\delta}$ gives us an indication of how quickly the iterative process converges.

4.3. The multi-splitting method

To get a good accuracy during the fluid-structure simulation, it is necessary to define very fine mesh; so the discretization of the boundary value problems modeling the fluid-structure equations leads to solve large scale linear sparse systems. Solution of such large algebraic systems are challenging and there is a vast literature addressing this important issue concerning sub-domain methods. In the present subsection, we present the multi-splitting method which allows to give an unified presentation of most sub-domain methods and consequently allows to analyze in an unified way the behavior of all these methods.

We now consider the solution of problem (43) where A satisfies assumption (60) and in the sequel, for sake of generality, we will consider only the point decomposition. For $m \in \mathbb{N}$, let the following collection of regular splittings of matrix A:

$$A = M^l - N^l,\ l = 1,\ \ldots, m, \tag{64}$$

where $(M^l)^{-1} \geq 0$ and $N^l \geq 0$; let $\hat{\hat{F}}^l : \mathcal{D}(\hat{\hat{F}}^l) \mapsto \mathcal{D}(\hat{\hat{F}}^l),\ \mathcal{D}(\hat{\hat{F}}^l) \subset \mathcal{E}$, $l = 1,\ \ldots, m$, be m fixed point mappings associated with problem (43) and defined by

$$\begin{cases} \hat{\hat{F}}^l(\tilde{Y}) = \tilde{X},\ l = 1, \ldots, m,\ \tilde{Y} \in \mathcal{E},\ \tilde{X} \in \mathcal{E} \\ \text{such that } M^l\tilde{X} = N^l\tilde{Y} + \tilde{B}. \end{cases} \tag{65}$$

Assume that the space $\mathcal{E}$ is normed by the uniform weighted norm, defined similarly to (63) by

$$\left\|\tilde{Y}\right\|_{\Theta} = \max_{i=1,\ldots,M}\left(\frac{|\tilde{y}_i|}{\bar{\theta}_i}\right),\ \bar{\theta}_i > 0,\ \bar{\theta}_i \in \mathbb{R},$$

where here, for the point decomposition, $|\tilde{y}_i|$ denotes the absolute value of $\tilde{y}_i$, $\tilde{y}_i \in \mathbb{R}$; assume that for $l = 1,\ \ldots, m$, $\hat{\hat{F}}^l$ is contractive with respect to $\tilde{X}^*$, its fixed point, for every decomposition according to the result of Corollary 1, so that the following inequality is verified

$$\left\|\hat{\hat{F}}^l(\tilde{Y}) - \tilde{X}^*\right\|_{\Theta_l} \leq \hat{\delta}_l\cdot\left\|\tilde{Y} - \tilde{X}^*\right\|_{\Theta_l},\ l = 1,\ \ldots, m, \tag{66}$$

where $0 \leq \hat{\hat{\delta}}_l < 1$.

A formal multi-splitting associated with problem (43) is defined by the collection of fixed point problems (see [22])

$$\tilde{X}^* = \hat{\hat{F}}^l(\tilde{X}^*),\ l = 1,\ \ldots, m,\ \tilde{X}^* \in \mathcal{E}. \tag{67}$$

Let now $E = \mathcal{E}^m = \mathcal{E} \times \mathcal{E} \times \ldots \times \mathcal{E}$ and consider the following block-decomposition of E

$$E = \prod_{l=1}^{m} E_l,$$

where $E_l = \mathcal{E}$. Let $X \in E$, defined by

$$X = \{\tilde{X}^1, \ldots, \tilde{X}^l, \ldots, \tilde{X}^m\} \in \prod_{l=1}^{m} E_l,$$

where $\tilde{X}^1,\ \ldots, \tilde{X}^l,\ \ldots, \tilde{X}^m$ denote m vectors of $\mathcal{E}$.

Definition 5. The extended fixed point mapping H: $E \mapsto E$ associated with the formal multi-splitting is given as follows

$$\begin{cases} H(Y) = X, \text{ such that } \tilde{X}^l = \hat{\hat{F}}^l(\tilde{Z}^l) \\ \text{with } \tilde{Z}^l = \sum_{k=1}^{m} W_{lk}\tilde{Y}^k,\ l = 1, \ldots, m, \end{cases}$$

where W_{lk} are non-negative diagonal weighting matrices satisfying, $\forall\ l \in \{1,\ \ldots, m\}$

$$\sum_{k=1}^{m} W_{lk} = I_l,$$

I_l being the identity matrix in $L(E_l)$.

Since $\hat{\hat{F}}^l(\mathcal{D}(\hat{\hat{F}}^l)) \subset \mathcal{D}(\hat{\hat{F}}^l)$, then obviously $H(\mathcal{U}(H)) \subset \mathcal{U}(H)$ where $\mathcal{U}(H) = \prod_{l=1}^{m} \mathcal{D}(\hat{\hat{F}}^l)$.

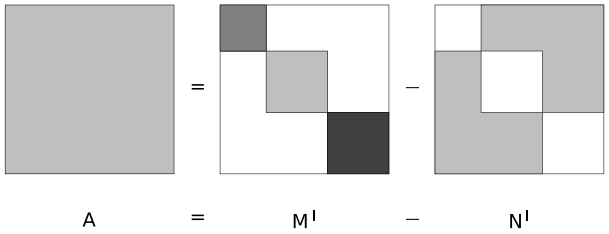


Fig. 2. Decomposition of matrix A.

Note that considerable saving in computational work may be possible, since a component of $\tilde{Y}^k$ needs not be computed if the corresponding diagonal entry of the weighting matrices are zero; the role of such weighting matrices may be regarded as determining the distribution of the computational work of the individual processors.

Fig. 2 schematically represents a decomposition of matrix A.

Let the following block-decomposition of the mapping H

$$H(X) = \{H^1(X), ..., H^l(X), ..., H^m(X)\} \in \prod_{l=1}^{m} E_l.$$

Note that for a particular choice of the weighting matrices W_{lk}, we can obtain various iterative methods and particularly on one hand a sub-domain method without overlapping and on the other hand the classical Schwarz alternating method (see [22]). Indeed in accordance with [22], the block Jacobi method corresponds to the following choice of M^l

$$M^l = diag(I_1, ..., I_{l-1}, A_{l,l}, I_{l+1}, ..., I_m),$$

and to the choice of $W_{lk} \equiv \bar{W}_l$ given by

$$\bar{W}_l = diag(0, ..., 0, I_l, 0, ..., 0),$$

according to the scheme on Fig. 3.

Note that for the parallel block Jacobi method, there is no overlap and the weighting diagonal matrices only have entries with values of zero or one.

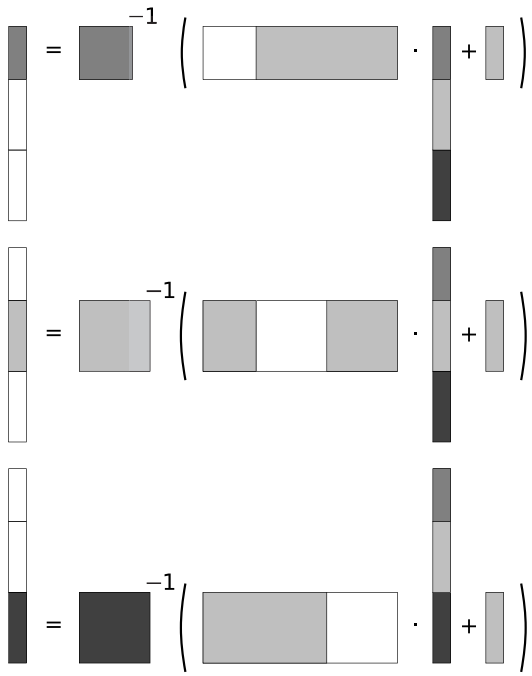


Fig. 3. Decomposition of matrix A.

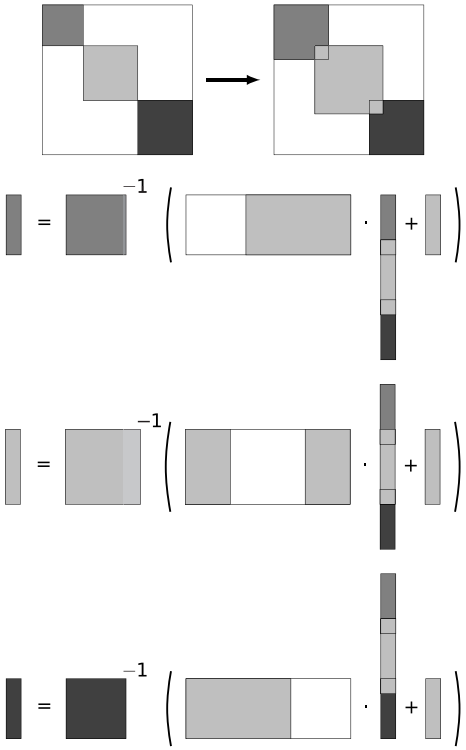


Fig. 4. Decomposition of matrix A with overlapping.

For the Schwarz alternating method, there is overlap and consequently some entries of the weighting diagonal matrices have values strictly between zero and one as illustrated by the scheme on Fig. 4.

The reader is referred to Appendix A where the weight matrices are presented for both cases corresponding to block Jacobi on Fig. A.16 and to Schwarz alternating method on Fig. A.17.

For the O'Leary and White multi-splitting method [18], $W_{lk} = \bar{W}_k$ satisfies

$$\sum_{k=1}^{m} \bar{W}_k = \mathcal{I} \text{ and } (\bar{W}_k)_{j,j} = 0 \ , \ \forall\, j \notin \mathcal{J}_k, \tag{68}$$

where for $k \in \{1, ..., m\}$, $\mathcal{J}_k$ is a subset of $\{1, ..., M\}$ and $\alpha = m$, where α has been considered in the block decomposition described in Section 4.1, this choice corresponding to the fact that each processor manage one of the m splitting; in this case the diagonal weighting matrices $\bar{W}_k$ have only zero and one entries, corresponding to the fact that the sub-vectors to compute are disjoint sub-vectors, and the multi-splitting method is non-overlapping (see [22]).

Let us recall now a result of [22].

Proposition 3. *Let us denote by $X^* = \{\tilde{X}^*, ..., \tilde{X}^*\}$ where $\tilde{X}^*$ is the solution of (43); then if assumption (66) is verified, the extended fixed point mapping H is $\|.\|_{\bar{\Theta}}$ contractive with respect to X^*, where $\|.\|_{\bar{\Theta}_l}$ is defined by*

$$\|X\|_{\bar{\Theta}_l} = \max_{1 \le l \le m} \left(\max_{i=1,...,M} \left(\frac{|\tilde{x}_i|}{(\bar{\Theta}_l)_i} \right) \right);$$

the associated constant of contraction being

$$\hat{\delta} = \max_{1 \le l \le m} (\hat{\delta}_l) < 1.$$

Then, combining the results of Corollary 1 and of Proposition 3, we

have the following result.

Corollary 2. *Consider the solution of problem (43); then if assumption (60) is verified, i.e. the matrix A is an M-matrix, then for any decomposition of the problem, the formal multi-splitting H is contractive with respect to X* and any sequential, parallel synchronous or asynchronous multi-splitting method starting from* $\tilde{X}^0 \in \mathcal{U}$ *converge to the solution of (43).*

Proof. Indeed, if assumption (60) is verified, in Corollary 1 it was stated that each fixed point mapping $\hat{\hat{F}}^l$ is contractive, for $l = 1, ...,m$; then the assumptions of Proposition 3 are verified and the proof is complete. □

Remark 12. In fact, from an implementation point of view, in multi-splitting methods, we can obtain two level of parallelism; indeed

- the first level of parallelism can be obtained by assigning one of the m splitting (64) to a processor in such a way that each of the m splittings is processed independently of the others;
- and the second level of parallelism corresponds to the fact that each set of equations associated to each splitting is decomposed into β blocks solved by an asynchronous parallel algorithm identical to the one previously presented in (45)–(47).

4.4. Solution of the diagonal sub-problems

In order to achieve a good computing precision, it is necessary to consider a very fine mesh. This leads to having to solve very large algebraic systems such as $A\tilde{X} = \tilde{B}$ at each time-step. In this context, the solution of the linear systems involved after discretization of the Navier–Stokes equations, the Navier equation and Eq. (42), can lead to very large computation times. In order to overcome this issue, we are led to decompose the problem into large β blocks and, using a synchronization process described in the next paragraph 5, to parallelize the algebraic systems solution by using the multi-splitting method which models in fact various sub-domain methods.

The implemented multi-splitting method proceeds iteratively to solve the linear system in such a way that each sub-system like

$$\hat{A}_{l,l}\cdot\hat{\tilde{X}}_l = \hat{\tilde{B}}_l - \sum_{j\neq l}\hat{A}_{l,j}\cdot\hat{\tilde{X}}_j,\ l = 1,...,\beta, \tag{69}$$

where the matrix A is decomposed in $\hat{A}_{i,j}$ blocks with $i, j = 1, ...,\beta$, is solved in parallel. The solution $\tilde{X}$ and the right-hand side $\tilde{B}$ are decomposed accordingly in sub-vectors respectively denoted $\hat{\tilde{X}}_l$ and $\hat{\tilde{B}}_l$. Each sub-system (69) is solved independently by a set of processors and communications are required to update the right-hand side of each sub-system, such that the vectors updated by the other processors represent the data dependencies between the blocks. In the implemented multi-splitting method, due to the sparsity of the sub-matrices, we actually have a two-level iteration; an external parallel iteration involving blocking or non-blocking communications and an inner iteration where each system of type (69) can be solved by a variant of the conjugated gradient method or another algorithm such as a multi-grid method. It should be noted that for external parallel iteration, with blocking or non-blocking communications, the calculation method considered fits well within the algorithm model (45) to (47) because inter-processor communications can be synchronous or asynchronous.

In addition, with regard to the two corrective steps of the PISO method:

- the amount of momentum is then calculated explicitly by (20) and (23);
- we calculate the pressure correction by (21) and (24) which requires the use of solvers and we chose to use a multi-grid algebraic method which proves more effective.

Finally we will see in Section 6 presenting the experimental results the effectiveness of this coupling of the external parallel iteration with the internal iteration.

Nevertheless, for the solution of the Navier–Stokes equations, the matrices involved in the PISO method are typically symmetric for the pressure equations and asymmetric for the momentum equations; in order to avoid computation issues due to the boundary layer, we specify that for the latter equations we have chosen upwind schemes. The matrices arising after discretization of the Navier equation are symmetric; therefore, for each type of systems, we will have to consider slightly different solution methods.

In the case of a symmetric matrix, a classical preconditioner consists in considering the incomplete Cholesky conjugate gradient method. But in order to reduce the storage needed, it is possible to introduce a simplified version of this previous method, called the diagonal incomplete Cholesky preconditioned conjugate gradient method (DICPCG). In such method the fill-in arising from the factorization, due to the off-diagonal entries is eliminated and only the diagonal entries are modified; in other words the upper and lower parts of the matrix to invert are kept unchanged. Such preconditioning can be used for the solution of the Navier equations and also to solve the pressure equation. The following structured code (see Algorithm 3) summarizes the implementation of the DICPCG method, where $\mathcal{P}$ is the preconditionning matrix.

1: $\mathbf{r}^{(0)} = \mathbf{b} - A\mathbf{\Phi}^{(0)}$
$\mathbf{d}^{(0)} = \mathcal{P}^{-1}\mathbf{r}^{(0)}$
choose starting direction
2: iterate starting at (n) until convergence
3: $\mathsf{a}^{(n)} = \dfrac{\left(\mathbf{r}^{(n)}\right)^T\mathcal{P}^{-1}\mathbf{r}^{(n)}}{\left(\mathbf{d}^{(n)}\right)^T A\mathbf{d}^{(n)}}$
choose factor in $\mathbf{d}$ *direction*
4: $\mathbf{\Phi}^{(n+1)} = \mathbf{\Phi}^{(n)} + \mathsf{a}^{(n)}\mathbf{d}^{(n)}$
obtain new $\mathbf{\Phi}$
5: $\mathbf{r}^{(n+1)} = \mathbf{r}^{(n)} - \mathsf{a}^{(n)}A\mathbf{d}^{(n)}$
calculate new residual
6: $\mathsf{b}^{(n+1)} = \dfrac{\left(\mathbf{r}^{(n+1)}\right)^T\mathcal{P}^{-1}\mathbf{r}^{(n+1)}}{\left(\mathbf{r}^{(n)}\right)^T\mathcal{P}^{-1}\mathbf{r}^{(n)}}$
calculate coefficient to conjugate residual
7: $\mathbf{d}^{(n+1)} = \mathcal{P}^{-1}\mathbf{r}^{(n+1)} + \mathsf{b}^{(n+1)}\mathbf{d}^{(n)}$
obtain new conjugated search direction

Algorithm 3. The PCG method.

For an asymmetric matrix, we have to use the bi-conjugate gradient method preconditioned by the diagonal incomplete LU factorization (DILUPBiCG). The bi-conjugate gradient (BiCG) is a method used to solve the following system:

$$\begin{bmatrix} 0 & \mathbf{A} \\ \mathbf{A}^T & 0 \end{bmatrix}\begin{bmatrix} \hat{\mathbf{\Phi}} \\ \mathbf{\Phi} \end{bmatrix} = \begin{bmatrix} \mathbf{b} \\ 0 \end{bmatrix}, \tag{70}$$

where $\hat{\Phi}$ is an auxiliary variable non really used practically.

This method leads to symmetric global matrix and thus enable the direct application of the conjugate gradient method to the asymmetric algebraic systems. Computational work is not significantly greater than the conjugate gradient method. The great advantage of the BiCG method is its ease of encoding and the moderate size of the stored informations. However, this method has the disadvantage of being unstable. Moreover, when the bi-conjugate gradient method is properly preconditioned, it rapidly converges to the solution of the linear system. In a similar way as in the symmetric case, and for the same reasons, we consider here a preconditioner by the diagonal incomplete LU

factorization using the block-wise form of the matrices, leading then to define the DILUPBiCG method. The convergence is classically greatly improved. We present hereafter a structured code (see Algorithm 4) showing the implementation of the DILUPBiCG method.

$\mathbf{r}^{(0)} = \hat{\mathbf{r}}^{(0)} = \mathbf{b} - A\mathbf{\Phi}^{(0)}$
$\mathbf{d}^{(0)} = \mathcal{P}^{-1}\mathbf{r}^{(0)}$
$\hat{\mathbf{d}}^{(0)} = \mathcal{P}^{-T}\hat{\mathbf{r}}^{(0)}$
choose starting direction
iterate starting at (n) until convergence
$\mathsf{a}^{(n)} = \dfrac{\left(\hat{\mathbf{r}}^{(n)}\right)^T \mathcal{P}^{-1}\mathbf{r}^{(n)}}{\left(\hat{\mathbf{d}}^{(n)}\right)^T A\mathbf{d}^{(n)}}$
choose factor in $\mathbf{d}$ *direction*
$\mathbf{\Phi}^{(n+1)} = \mathbf{\Phi}^{(n)} + \mathsf{a}^{(n)}\mathbf{d}^{(n)}$
obtain new $\mathbf{\Phi}$
$\mathbf{r}^{(n+1)} = \mathbf{r}^{(n)} - \mathsf{a}^{(n)}A\mathbf{d}^{(n)}$
calculate new $\mathbf{r}$ *residual*
$\hat{\mathbf{r}}^{(n+1)} = \hat{\mathbf{r}}^{(n)} - \mathsf{a}^{(n)}A^T\hat{\mathbf{d}}^{(n)}$
calculate new $\hat{\mathbf{r}}$ *residual*
$\mathsf{b}^{(n+1)} = \dfrac{\left(\hat{\mathbf{r}}^{(n+1)}\right)^T \mathcal{P}^{-1}\mathbf{r}^{(n+1)}}{\left(\hat{\mathbf{r}}^{(n)}\right)^T \mathcal{P}^{-1}\mathbf{r}^{(n)}}$
calculate coefficient to conjugate residual
$\mathbf{d}^{(n+1)} = \mathcal{P}^{-1}\mathbf{r}^{(n+1)} + \mathsf{b}^{(n+1)}\mathbf{d}^{(n)}$
obtain new search $\mathbf{d}$ *direction*
$\hat{\mathbf{d}}^{(n+1)} = \mathcal{P}^{-T}\hat{\mathbf{r}}^{(n+1)} + \mathsf{b}^{(n+1)}\hat{\mathbf{d}}^{(n)}$
obtain new search $\hat{\mathbf{d}}$ *direction*

Algorithm 4. The PBiCG method.

This iterative method is used three times in the first step of the PISO algorithm, one for each spatial component of the velocity.

From a theoretical point of view, we must ask ourselves if the incomplete factorization is always possible and if this is not always possible in which case it is? In fact such incomplete factorization is not possible for any matrix and we have now to take into account the properties of the discretization matrices highlighted in the paragraph 3.3. Indeed, for both cases, the matrices arising from the discretization have a common property; in agreement with the expression of the entries of these matrices viewed in (40) and in (37), it can be noted that the diagonal entries are strictly positive, the off-diagonal entries are negative or zeros and such matrices are strictly diagonally dominant due the temporal discretization, and then are M-matrices [37]. Concerning the preconditionning of the conjugate gradient method by incomplete decomposition, several works by various authors have been conducted on this type of matrices and, for more details, we refer to [38,39]. Taking into account the sizes of the systems to solve, due to sparsity property, the iterative conjugate gradient method and its preconditioned variants are very efficient and more accurate compared to direct methods. Indeed, the use of iterative methods does not require modification of the matrices and involve few arithmetic operations when the matrices are sparse; which considerably limits the propagation of round-off errors, especially when the matrices are badly conditioned, as it is the case for those intervening in the pressure correcting steps of the PISO method. This avoids distortion of the calculated solution. On the other hand concerning the use of direct methods, if the matrix has a bandwidth structure with $(2L + 1)$ entries in the band, the Gauss method requires the order of ML^2 arithmetic operations, when M the dimension of the matrix is large, which corresponds to an important computation cost. This can lead to significant losses in accuracy, especially if the matrix is ill conditioned. Moreover, the use of the Gauss method does not preserve the sparse character of the matrices to the extent that a filling phenomenon occurs which increases the number of arithmetic operations and which constitutes an additional difficulty due to the amount of supplementary storage.

In order to accelerate the convergence of the method, in relation with the property of M-matrices, preconditioning techniques can be used rigorously. Among the classical preconditioners, the incomplete LU factorization, derived from the Gaussian factorization, was shown to be one of the most effective. According to a result in [38], we know that if the matrix arising in the linear system is an M-matrix, then the incomplete factorization is feasible and, moreover the preconditioned conjugate gradient method using the incomplete factorization works very well. Note also that, for any M-matrix A, one characteristic property is that the block diagonal matrix $\hat{A}_{l,l}$ is also an M-matrix [37–39], and consequently the preconditioning by diagonal incomplete factorization can be applied successfully. In these circumstances, the solution of the sub-systems (69) by conjugate gradient-like methods is possible to the extent that the DICPCG method is used for the structure and the DILUPBiCG method is used to solve the Navier–Stokes equations. Thus, for the solution of the sub-problems (69) we can also, in the outer iteration SISC or SIAC defined in paragraph 5, use an inner iteration constituted by the preconditioned incomplete factorizations.

Lastly, note that, instead of using the incomplete Cholesky conjugate gradient method to solve the ill-conditioned pressure equation, it is better to use the geometric algebraic multi-grid method (GAMG) [40] since such method is more efficient. This approach can be used to build highly efficient and robust linear solver for both highly anisotropic grids and/or problems with large changes in the coefficients of their equations. For more details the reader is referred to [40].

4.5. Extension to the pseudo-linear problem

The previous analysis and the previous results are still valid when the linear algebraic system is perturbed by an increasing continuous diagonal nonlinear operator $\tilde{X} \mapsto \Phi(\tilde{X})$

$$A\tilde{X} + \Phi(\tilde{X}) = \tilde{B},$$

and also by a diagonal monotone multivalued operator $\partial\Gamma$ taking into account some constraints on the solution

$$A\tilde{X} + \Phi(\tilde{X}) + \partial\Gamma(\tilde{X}) - \tilde{B} \ni 0,$$

where $\partial\Gamma(\tilde{X})$ is the sub-differential of the indicator function of the convex set defining the constraints on the solution.

Indeed, since the linear system is perturbed in each of the previous cases by a diagonal monotone maximal operator then, in this non-linear situation, inequalities like (48) are still valid for the diagonal part due to the monotony of the perturbation [41], the off-diagonal part being unchanged. Therefore, the results of Propositions 1 and 2 and Corollary 1 are still valid for the analysis of parallel asynchronous iterations. In such situations, where constraints on the solution can appear when we deal with fluid-structure interactions, it is the displacement in the Navier equation which is submitted to constraints of type:

$$\mathbf{D}_{min} < \mathbf{D} < \mathbf{D}_{max}.$$

Moreover, in the case of analysis of parallel asynchronous multi-splitting methods, the result of Corollary 2 is still valid for the solution of pseudo-linear problems perturbed by a single or a multi-valued diagonal operator (cf. [22]).

Remark 13. The use of subdifferential notion is a formal and convenient expression of the mathematical formulation of a problem with constraint allowing to analyze the behavior of the parallel algorithms in this case. This formalism is absolutely not involved in the implementation of the method. From an algorithmic point of view, we do not have to discretize the subdifferential operator. The problem with constraints is a non-linear problem but does not have to be

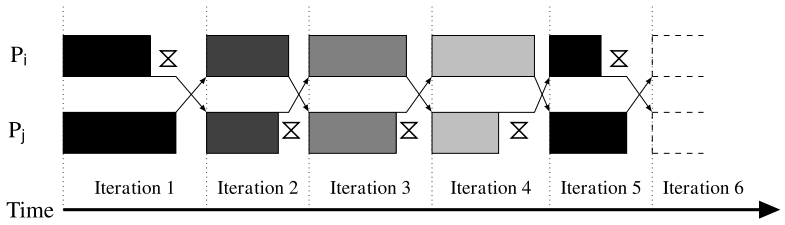


Fig. 5. Example of a SISC scheme on two processors.

linearized by a Newton like method. We simply have to project the intermediate iterated vectors on a set that defines the constraints.

5. Parallelization schemes

5.1. Synchronous iterations and synchronous communications

The most common iterative solvers are the one using synchronous iterations and synchronous communications (SISC). The iterations synchronism is due to the fact that each processor can only start to compute the new iteration when it receives the data computed at the previous iteration from all of its neighbors. Thus, all processors begin the computation of the same iteration at the same time and exchange data at the end of each iteration through synchronous global communications.

Fig. 5 represents the execution of a SISC parallel algorithm. The synchronism of iterations in SISC algorithms implies the exact same number of iterations as their serial algorithms counterparts. Therefore, the conditions of their convergence are the same as those of serial algorithms. However, synchronous communications can often cause processors to experience idle time due to them waiting to communicate or due to the speed of the communications being to slow on the network.

5.2. Synchronous iterations and asynchronous communications

Iterative solvers using synchronous iterations and asynchronous communications (SIAC) have been developed in order to improve solution performances on networks with slow and/or heterogeneous inter-connexions. They manage to reduce idle times on processors between two iterations. Similarly to SISC algorithms, a processor still waits for the receipt of all shared data computed by its neighbors at the previous iteration. However, the global synchronous communications are replaced with asynchronous dispatches and blocking receptions.

Fig. 6 represents the execution of a SIAC parallel algorithm. The convergence conditions are the same as for SISC and serial algorithms. However, asynchronous communications allow for overlaps between computations and data exchanges. A processor can send shared data to its neighbor as soon as they are ready to be used in the new iteration computation, which reduces the time waiting for data to be received between two iterations.

It must be emphasized that the non-blocking method for the considered problem has been difficult to implement because on one hand, the asynchronous messages being delivered in a chaotic order by all processors, and on the other hand, asynchronous communications happen on the inter-processor boundaries in order to update the mesh. Therefore, to ensure consistency of the mesh, it has been necessary to get all of outstanding communications for a given processor and to blocks this processor until all communication operations are finished.

5.3. Stopping criterion

We must solve sub-systems of the type (69) where in the second member appears the interactions denoted $\hat{X}_j$. If we consider GC methods, we must calculate the residue from this second member of the sub-system which, therefore, is frozen once and for all when we start the execution by GC, which does not allow asynchronous iterations in the usual sense. On this subject it should be noted that the synchronous case appears as a special case in the model (45)–(47) (see Remark 2). So at the sub-system level, a classical stopping test is managed for a GC method corresponding to the fact that the residue is smaller than a given threshold, the internal iterations being performed synchronously, including in the non-blocking case whereas in this case communications are asynchronous. Similarly, when we use the GAMG method for the pressure correction, each inner iteration is stopped when the residue is smaller than a given threshold.

At the external iteration level, the situation is simplified, considering the choice of fluid-structure solvers used. Indeed, the process is stopped when all internal iterations converged, because it is well indicated in paragraph 5.2 that we are dealing with synchronous iterations, namely SISC and SIAC methods, and not asynchronous iterations.

5.4. Parallel fluid-structure procedure

The parallel fluid-structure algorithm consists in adapting the serial algorithm described in Section 3.4. Steps 4, 5 and 6 of Algorithm 2 are modified to be performed in parallel using the MPI library facilities.

We first solve the fluid sub-domain on a number of processors P_i and then we store the data at the interface in a predefine variable. Then, we solve the structure problem on the same number of processors P_i using the interface values previously stored by the fluid solver. The values of **D** are calculated by the solid solver and stored into an other variable

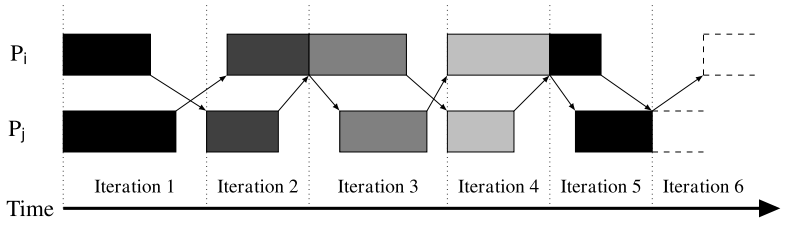


Fig. 6. Example of a SIAC scheme on two processors.

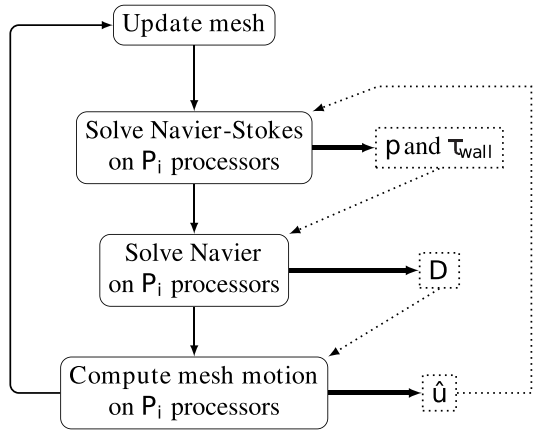


Fig. 7. Parallel fluid-structure procedure.

which constitutes part of boundary values to compute the mesh velocity $\hat{\mathbf{u}}$ (see Eq. (42)). Note that for the rest of boundary conditions for this last equation, we consider the mesh to be clamped, which corresponds to $\hat{\mathbf{u}} = 0$. This procedure is illustrated by Fig. 7.

The method consisting in using variables independent from the solvers to store data at the interface, allows for using completely independent fluid and solid sub-domains decompositions; so we can use two different decomposition methods for the fluid and the solid.

6. Application

6.1. Test case description

The fluid-structure interaction test case consists in a small wall submitted to a cross flow. The wall is 20cm high, 40cm long and 10cm thick. It is clamped at its base in a channel which is twice its height and width that is: 40cm high and 80cm wide (see Fig. 8). The channel is filled with water and the flow goes in the direction perpendicular to the width of the wall. It's a simple case study already presented in sequential in [42], but it allows us to easily refine the mesh in order to get large discretization matrices and benchmark the parallel performances of the fluid-structure solver.

6.2. Numerical and physical parameters

The experiments have been performed on one hand on a local cluster and on the other hand on the Grid'5000 platform located in France; in this network, every site hosts clusters and all sites are

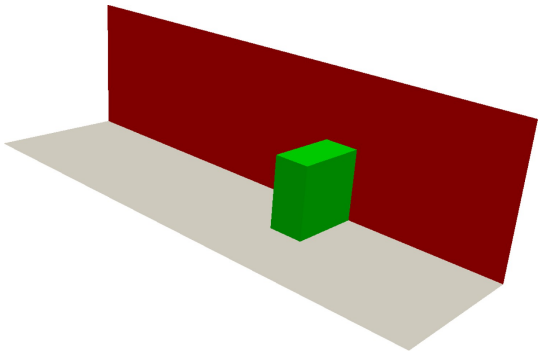


Fig. 8. Wall in green and symmetry plane in red.

connected to each other by high-speed communication. Grid'5000 is a large and versatile testbed for experiment-driven research in all areas of computer science, with a focus on parallel and distributed computing including Cloud, HPC and Big Data. Grid'5000 provides access to a large amount of resources: 1000 nodes, 8000 cores, grouped in homogeneous clusters, and featuring various technologies: 10Gbps Ethernet, Infiniband, GPUs, Xeon Phi. Fig. 9 illustrates the sites of the Grid'5000 architecture. The local cluster includes 10 cores and a cluster located in Nancy is composed of 51 nodes; their characteristics are summarized in Table 1.

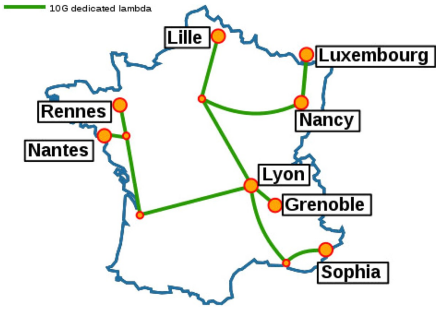


Fig. 9. Grid'5000 architecture.

Table 1
Clusters specifications.

Cluster	CPUs	Cores	RAM
Local	Intel Xeon E5-2630 v4	10 cores	64GB
Nancy	2x Intel Xeon E5-2630 v3	8 cores/CPU	126GB

The fluid and solid physical and numerical parameters are summarized in Table 2. In order to satisfy the Courant–Friedrichs–Lewy condition: $CFL = \Delta t \sum_{i=1}^{3} (u_i/\Delta x_i) < CFL_{max}$, we chose to take a time step of 0.005s.

Table 2
Numerical and physical parameters.

Fluid parameters			
Number of CV	ν_f (m^2s^{-1})	ρ_f (kg m^{-3})	U_{max} (m s^{-1})
3,151,872	0.001	1000	0.2
Solid parameters			
Number of CV	ν	ρ_s (kg m^{-3})	E (Pa)
55,296	0.4	1000	10,000
Computational parameters			
Max FSI iterations	End time (s)	Δt (s)	
20	0.4	0.005	

6.3. Results

The simulation on a coarser mesh allows us to visualize the deformation of the wall and therefore of the mesh as shown on Fig. 10. For this simulation, the size of the systems to solve is given by the number of CVs in Table 2. Fig. 11 shows the displacement of a point at the top of the wall. We can see that the maximum displacement is obtained at t = 0.9s and the stationary state is reached at $t = 4$s.

The computation times for both SISC and SIAC methods for the parallel simulations on the local cluster as well as the corresponding speed-up and efficiencies on a range of 1 to 8 processors are summarized in Table 3. The results for the Nancy cluster using 8, 10, 12, 14 and 16 nodes are summarized in Table 4; note that due to exploitation constraints, we didn't have the opportunity to run a sequential

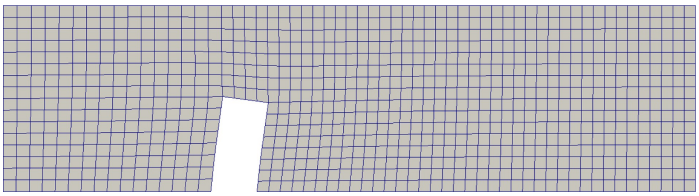


Fig. 10. Mesh deformation at t = 0.9s.

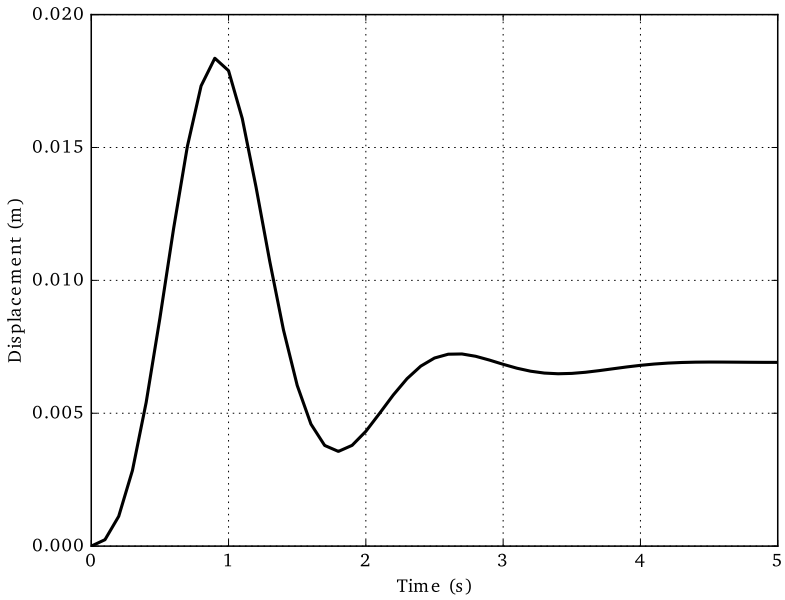


Fig. 11. Displacement at the top of the wall.

Table 3
Computation time (s), speed-up and efficiency for SISC and SIAC on the local cluster.

Number of cores	SISC	Speed-up	Efficiency	SIAC	Speed-up	Efficiency
1	85,440			85,440		
2	47,595	1.80	0.90	46,440	1.84	0.92
3	50,443	1.69	0.56	35,648	2.40	0.80
4	41,115	2.08	0.52	28,826	2.96	0.74
5	35,097	2.43	0.49	25,192	3.39	0.68
6	32,246	2.65	0.44	22,022	3.88	0.65
7	29,800	2.87	0.41	21,005	4.07	0.58
8	27,176	3.14	0.39	20,941	4.08	0.51

Table 4
Computation time (s) and gain between SISC and SIAC on the Nancy cluster.

Number of nodes	SISC	SIAC	Gain
8	27,952	21,648	22.5%
10	23,324	20,513	12.1%
12	21,232	19,596	7.7%
14	21,040	18,062	14.2%
16	24,240	23,520	3%

simulation on the cluster located in Nancy; note also that the experiments carried out on Grid'5000 were restricted due to the large computation times on the cluster. Moreover, parts of the system were under maintenance during the experiments which made it difficult to access the clusters.

Given the time required to perform the calculations, around 24 h in sequential mode, for operational reasons we have chosen a relatively modest size of algebraic systems to solve. On the local cluster, for this particular FSI test case, we have good efficiencies from 2 to 6 processors, but we notice that beyond 6 processors, the acceleration is constant, therefore, it doesn't seem useful to add additional processors. This observation is confirmed by the tests on Grid'5000. However, during these parallel experiments, the main focus was on a comparison of restitution times between the SISC and SIAC methods. The SIAC method has thus been shown to save around 30% of restitution time. Moreover, and even beyond 6 processors, parallel mode playback times, compared to sequential time, decrease even more.

On the Nancy cluster, the improvement of the SIAC method gives 22.5% more effective simulations on 8 nodes. The gain is lower with more nodes because the size of the problem isn't large enough to denote better improvements, also because the grid architecture is not always reliable and computations can be slowed down for a period of time. The fluid-structure problem is highly dependent on the number of time-steps in order to obtain consistent results. The large computation times are not only due to the number of CV but also to the number of time-steps and FSI iterations which are here capped to 20 iterations.

Figs. 12–14 represent the computational times, the speed-up and the efficiencies on the local cluster respectively.

Fig. 15 represents the computation times with respect to the number of nodes used on the Nancy cluster. We note that with 16 nodes, the computation time goes up and is actually larger than with 10 nodes. This is due to the size of the problem: not enough CV and to many communications.

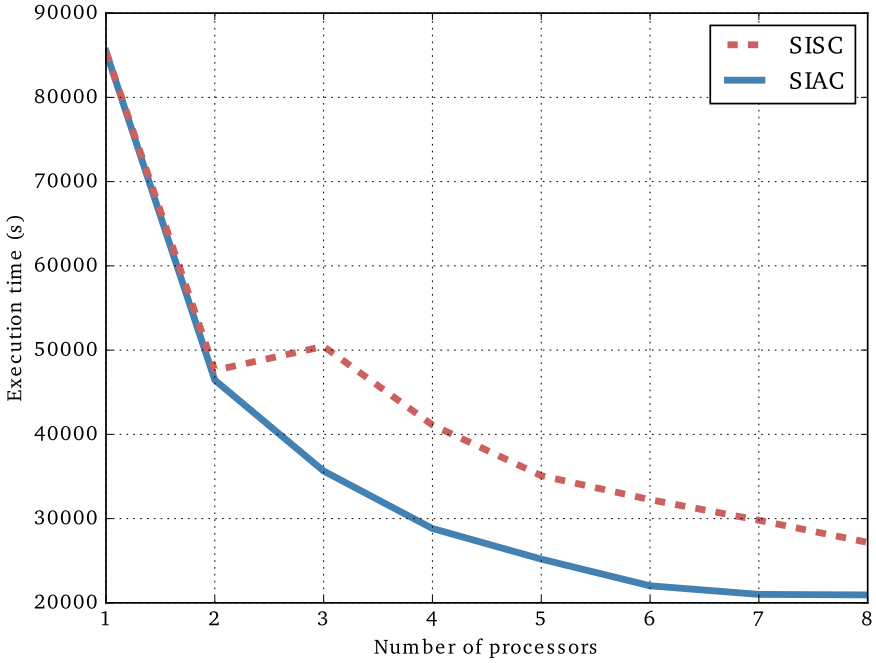


Fig. 12. Execution time on the local cluster.

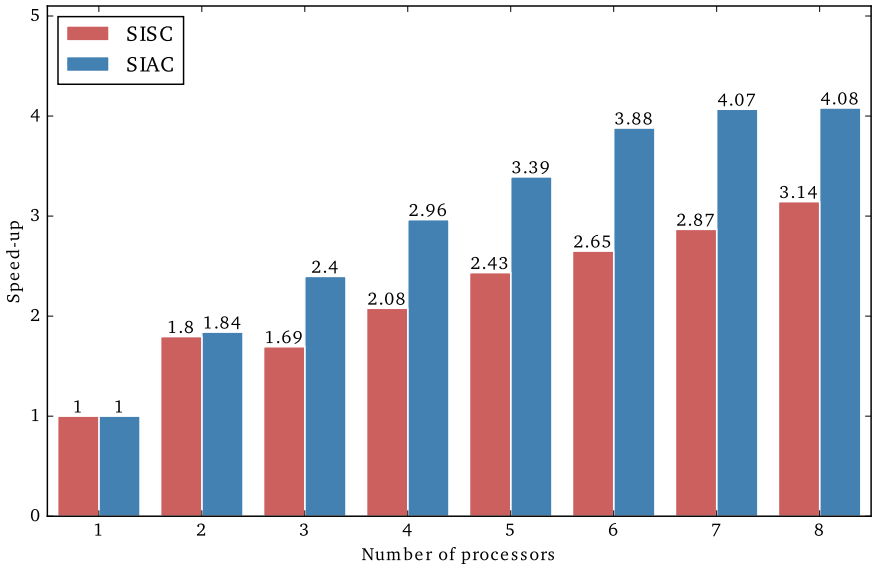


Fig. 13. Speed-up on the local cluster.

In this application the convergence speed is slow, in particular to solve the linear systems appearing during the two corrective steps of the PISO method as well as for the solution of Navier's equation, while for the momentum equations and the domain velocity equation the convergence is fast. So there will be a lot of synchronizations and in this case asynchronous parallel methods are of great interest. In this study, even for relatively small systems, asynchronous methods are efficient; and given our experience in the field, we have reason to believe that by increasing the size of systems we will improve the performances. These good performances in asynchronous parallel computing will further improve given the number of iterations which will be important.

Remark. We also performed experiments on two geographically distant sites of the Grid'5000 architecture (Nancy and Lyon), but since the network is slow, the performances are still better for the SIAC

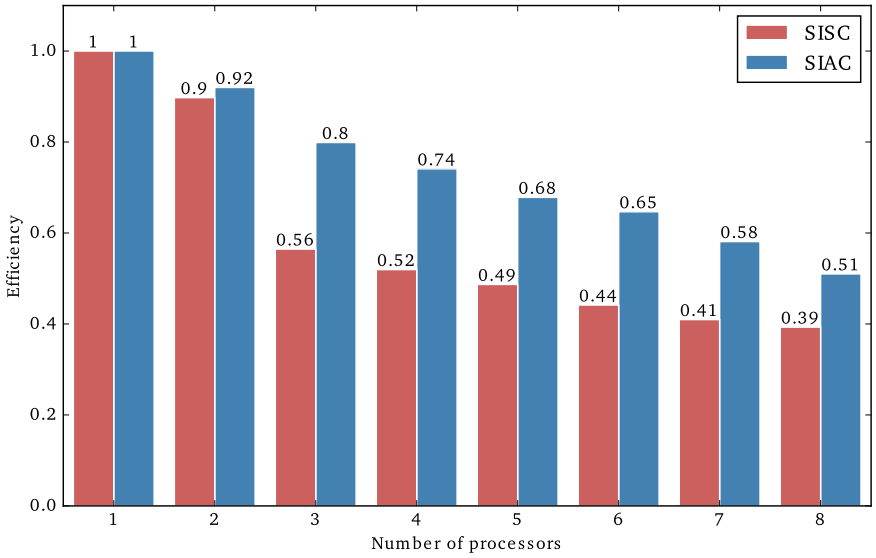


Fig. 14. Efficiency on the local cluster.

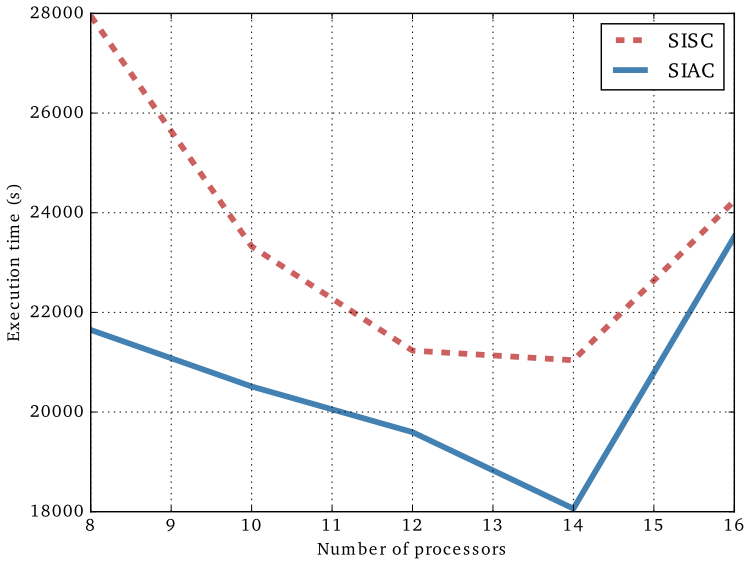


Fig. 15. Execution time on the Nancy cluster.

method but the gain is not significant enough to be presented in this paper.

7. Conclusion

In this study, we considered a target industrial application and it led us to suggest new theoretical results which allow the analysis of the multi-splitting method on the implementation level in order to remove idle times due to synchronizations. We considered two versions of the numerical parallel method implementation: blocking (SISC) and non-blocking (SIAC). It is worth noting that for the application, we considered a relatively simple test case which allowed us to illustrate and benchmark the performances of the fluid-structure interaction solver.

Acknowledgments

Experiments presented in this paper were carried out using the Grid'5000 testbed, supported by a scientific interest group hosted by Inria and including CNRS, RENATER and several Universities as well as other organizations.

Appendix A. Multi-splitting matrices

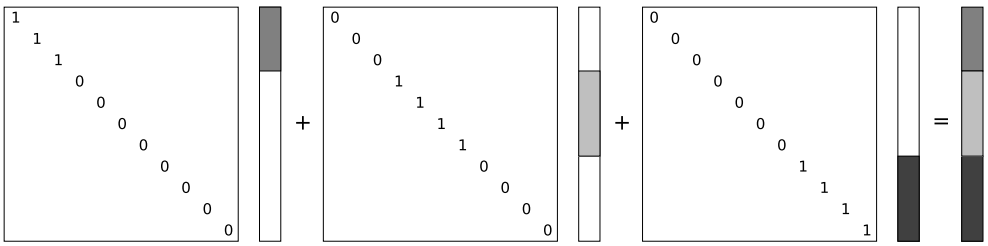


Fig. A1. Multi-splitting matrices without overlapping.

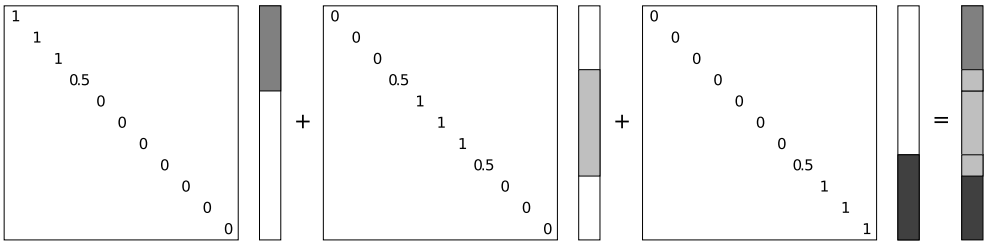


Fig. A2. Multi-splitting matrices with overlapping.

References

[1] Patankar SV. Numerical heat transfer and fluid flow. Series in computational methods in mechanics and thermal science New-York: Hemisphere Publishing Corporation; 1980.
[2] Jasak H, Weller HG. Application of the finite volume method and unstructured meshes to linear elasticity. Int J Numer Methods Eng 2000;48:267–87.
[3] Chau M, Garcia T, Spiteri P. Asynchronous grid computing for the simulation of the 3D electrophoresis coupled problem. Adv Eng Softw 2013;60–61:111–21.
[4] Chazan D, Miranker W. Chaotic relaxation. Linear Algebra Appl 1969;2:199–222.
[5] Miellou J-C. Algorithmes de relaxation chaotique à retards. ESAIM 1975;9:55–82.
[6] Miellou J-C. Asynchronous iterations and order intervals. In: Cosnard M, Trystram D, editors. Parallel algorithms and architectures. North Holland; 1986. 85–96.
[7] Baudet G. Asynchronous iterative methods for multiprocessors. J Assoc Comput Mach 1978;25:226–44.
[8] Bertsekas DP, Tsitsiklis JN. Some aspects of parallel and distributed iterative algorithms–a survey. Automatica 1991;27:3–21.
[9] El Tarazi MN. Some convergence results for asynchronous algorithms. Numer Math 1982;39. 325–240
[10] Miellou J-C, Spiteri P, El Baz D. A new stopping criterion for linear perturbed asynchronous iterations. J Comput Appl Math 2008;219:471–83.
[11] Magoulès F, Gbikpi-Benissan G. Distributed convergence detection based on global residual error under asynchronous iterations. IEEE Trans Parallel Distr Syst 2018;29:819–29.
[12] Bahi J, Contassot-Vivier S, Couturier R. Parallel iterative algorithms: from sequential to grid computing. Numerical analysis and scientific computing Chapman & Hall/CRC; 2007.
[13] Comte P, Miellou J-C, Spiteri P. La notion de H-accrétivité, applications. CRAS 1976;283:655–8.
[14] Giraud L, Spiteri P. Résolution parallèle de problèmes aux limites non linéaires. ESAIM 1991;25:579–606.
[15] Miellou J-C, Spiteri P. Un critère de convergence pour des méthodes générales de point fixe. ESAIM 1985;19:645–69.
[16] Magoulès F, Gbikpi-Benissan G. JACK2: an MPI-based communication library with non-blocking synchronization for asynchronous iterations. Adv Eng Softw 2018;119:116–33.
[17] Chau M, Garcia T, Spiteri P. Asynchronous Schwarz methods applied to constrained mechanical structures in grid environment. Adv Eng Softw 2014;74:1–15.
[18] O'Leary DP, White RE. Multi-splittings of matrices and parallel solution of linear systems. SIAM 1985;6:630–40.
[19] Szyld DB. Different models of parallel asynchronous iterations with overlapping blocks. Comput Appl Math 1998;17:101–15.
[20] Frommer A, Mayer G. Convergence of relaxed parallel multisplitting methods. Linear Algebra Appl 1989;119:141–52.
[21] Arnal J, Migallón V, Penadés J. Non-stationary parallel multisplitting algorithms for almost linear systems. Numer Linear Algebra Appl 1999;6:79–92.
[22] Bahi J, Miellou J-C, Rhofir K. Asynchronous multisplitting methods for nonlinear fixed point problems. Numer Algorithms 1997;15:315–45.
[23] Magoulès F, Roux F-X. Sub-structuring method for fluid-structure interaction problems with non-matching grids. In: Topping B, Montero G, Montenegro R, editors. Innovation in engineering computational technology Stirlingshire, UK: Saxe-Coburg Publications; 2006. p. 269–85. https://doi.org/10.4203/csets.15.13. Chap. 13
[24] Lai LS, Lai CH, Cheik Ahamed A-K, Magoulès F. Coupling and simulation of fluid-structure interaction problems for automotive sun-roof on graphics processing unit. 2014 IEEE international conference on high performance computing and communications. 2014. p. 137–44. https://doi.org/10.1109/HPCC.2014.26.
[25] Bolze R, Cappello F, Caron E, Dayde M, Desprez F, Jeannot E, et al. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. Int J High Perform Comput Appl 2006;20:481–94.
[26] Partimbene V, Garcia T, Spiteri P, Marthon P, Ratsifandrihana L. A parallel method for the solution of fluid-structure interaction problems. Proceedings of the fifth international conference on parallel, distributed, grid and cloud computing for engineering. Iványi P, Topping BHV, Várady G, editors. Stirlingshire, UK: Civil-Comp Press; 2017, https://doi.org/10.4203/ccp.111.20.
[27] Issa RI. Solution of the implicitly discretised fluid flow equation by operator-splitting. J Comput Phys 1985;62:40–65.
[28] Schafer M. Computational engineering –introduction to numerical methods. Berlin, Heidelberg: Springer; 2006.
[29] Dahlquist G. On matrix majorants and minorants, with applications to differential equations. Linear Algebra Appl 1983;52–53:199–216.
[30] Feingold DG, Varga RS. Block diagonally dominant matrices and generalizations of the Gerschgorin circle theorem. Pac J Math 1962;12:1241–50.
[31] Fiedler M, Ptak V. On matrices with non-positive off-diagonal elements and positive principal minors. Czechoslov Math J 1962;12:382–400.
[32] Miellou J-C. Méthodes de Jacobi, Gauss-Seidel, sur-relaxation par blocs, appliquées à une classe de problèmes non linéaires. CRAS 1971;273:1257–60.
[33] Miellou J-C. Sur une variante de la méthode de relaxation, appliquée à des problèmes comportant un opérateur somme d'un opérateur différentiable et d'un opérateur maximal monotone diagonal. CRAS 1972;275:1107–10.
[34] Ostrowski AM. On some metrical properties of operator matrices and matrices partitioned into blocks. J Math Anal Appl 1961;2:161–209.

[35] Robert F. Recherche d'une matrice parmi les minorants d'un opérateur linéaire. Numer Math 1966;9:189–99.
[36] Schröder J. Nichtlineare majoranten beim verfahren der schrittweisen näherung. Arch Math 1957;7:471–84.
[37] Ortega JM, Rheinboldt WC. Iterative solution of nonlinear equations in several variables. Computer science and applied mathematics New-York: Academic Press; 1970.
[38] Saad Y. Iterative methods for sparse linear systems. Other titles in applied mathematics SIAM; 2003.
[39] Meurant G. Computer solution of large linear systems. Studies in mathematics and its applications North Holland; 1999.
[40] Moukalled F, Mangani L, Darwish M. The finite volume method in computational fluid dynamics. Fluid mechanics and its applications Springer; 2015.
[41] Barbu V. Nonlinear semigroups and differential equations in banach spaces. Noordhoff International Publishing; 1976.
[42] Partimbene V, Spiteri P, Marthon P, Ratsifandrihana L. A two-dimensional numerical case study of fluid-structure interaction. In: Pombo J, editor. Proceedings of the third international conference on railway technology: research, development and maintenance. Stirlingshire, UK: Civil-Comp Press; 2016, https://doi.org/10.4203/ccp.110.43.

5.5 Article 5

Advances in Engineering Software 131 (2019) 116–142

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Research paper

Behavior of parallel two-stage method for the simulation of steel solidification in continuous casting

T. Garcia[a], P. Spiteri[*,b], G. Khenniche[b,c]

[a] IRIT-INPT, University of Toulouse, 2 rue Charles Camichel, B.P. 7122, Toulouse 31071 Cedex 7, France
[b] IRIT-INPT-ENSEEIHT, University of Toulouse, 2 rue Charles Camichel, B.P. 7122, Toulouse 31071 Cedex 7, France
[c] Faculty of Sciences, Department of Mathematics, Laboratory LAMAHIS, Skikda university, Algéria

ARTICLE INFO

Keywords:
Continuous casting
Sparse non-linear systems
Synchronous parallel algorithms
Asynchronous parallel algorithms
Multisplitting methods
Two-stage methods

ABSTRACT

This paper presents the behavior of general parallel synchronous and asynchronous multisplitting and two-stage methods for the numerical simulation of steel solidification in continuous casting. Thanks to the mathematical analysis and the implementation of these methods one can show the results of parallel experiments for the target application. The mathematical model is constituted by coupled nonlinear boundary value problems, namely the heat equation taking into account, on part of the boundary, a radiation phenomenon described by the Stefan law. For the numerical solution of such partial differential equations we consider, depending on whether the coefficient of thermal conductivity is constant or temperature-dependent, both an implicit or a semi-implicit discretization with respect to the time of the studied evolution problem, while the spatial discretization is carried out by adapted finite difference schemes. Then large scale discretized algebraic systems are solved by sequential and synchronous or asynchronous iterative algorithms; comparison of these various previous methods implemented on clusters and grid are achieved in both cases when the thermal conductivity is constant and more generally dependent of the temperature.

1. Introduction and motivation

In steel industry, the continuous casting is a process between the metal making and rolling. This process, shortly described in Fig. 1, allows the transformation of a liquid metal into a solid metal in a continuous way. This casting can last as long as we can feed a repartitor by liquid steel; note that it is also possible to feed several parallel moulds. This industrial process is, by far, the most efficient for solidifying a great amount of metal. During the process the liquid metal is water cooled during the progress in the machine. As long as the steel progresses into the machine, the solidified metal goes down at constant speed. Hence slabs or billets are obtained by the passage of liquid steel through several cooling zones. The transformation of the liquid phase into the solid phase takes place in an intermediary zone the so called mushy zone. To summarize the cooling solidification process, we can describe it in the following manner: the steel goes through three phases in three areas denoted by Ω_1 for the liquid area, Ω_2 for the mushy area and Ω_3 for the solid area. The initial temperature in Ω_1 is about 1500^o C, while it remains equal to 800^o C and 600^o C, respectively in Ω_2 and Ω_3. This phenomenon describes a thermo-mechanical problem.

According to the previous description, we consider a steel portion in the transverse direction of the continuous casting. The liquid steel poured into the mould cooled by water, after getting cold enough penetrates the cooling zones receiving optimal flow water quantity. This allows to take into account the primary and secondary cooling, in order to compute the thermal evolution of the steel. The temperature satisfies the heat equation with appropriate boundary conditions describing the physical phenomenon. During its development in the machine, the steel will be either cooled upon contact with the mould, either by the water jet or simultaneously with both. To describe this operation, it is assumed that the exchange of heat in the mould is uniform and the water jet cooling and thermal contact with the rollers constituted by cylinders will be modeled by the condition of convection and / or radiation. It is customary to assume that the contact surfaces with the ambient air are subject to conditions of the same type. The heat exchange that occurs between the subdomains liquid-mushy and mushy-solid is modeled by a condition at the interfaces. Every cooling zone is characterised by its temperature i.e, we have to find successively the temperature field in the liquid zone Ω_1, then in the mushy zone Ω_2 and finally in the solid zone Ω_3. For more details the reader is referred to Refs. [1–4].

The mathematical model used is based on the conservation of energy resulting in the equation of diffusion of heat in three dimensional

* Corresponding author.
E-mail addresses: thierry.garcia@irit.fr (T. Garcia), pierre.spiteri@enseeiht.fr (P. Spiteri).

https://doi.org/10.1016/j.advengsoft.2018.11.012
Received 30 August 2017; Received in revised form 23 October 2018; Accepted 30 November 2018
Available online 30 December 2018
0965-9978/

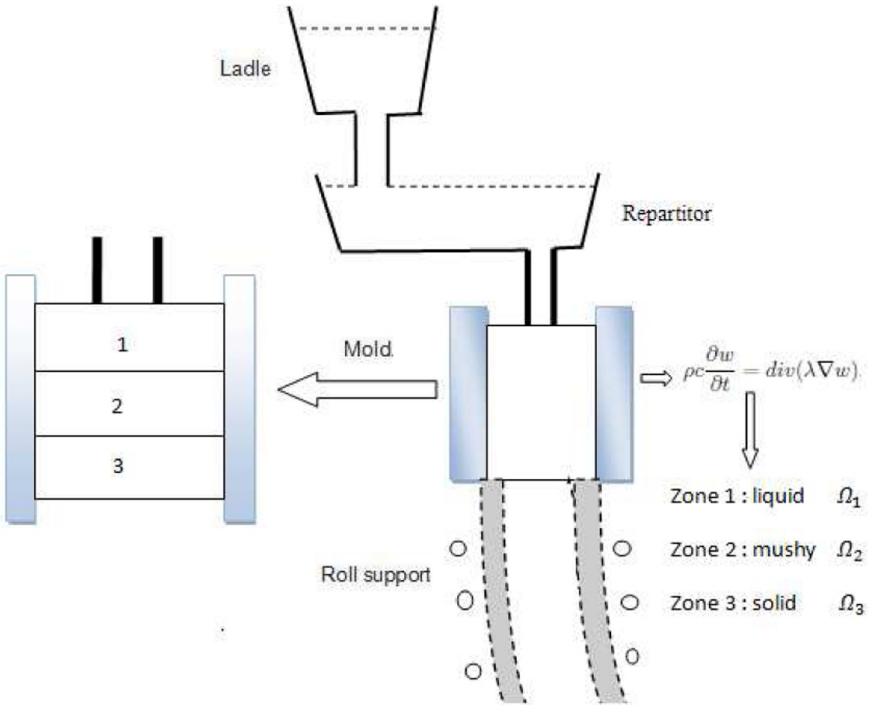


Fig. 1. General scheme of continuous casting.

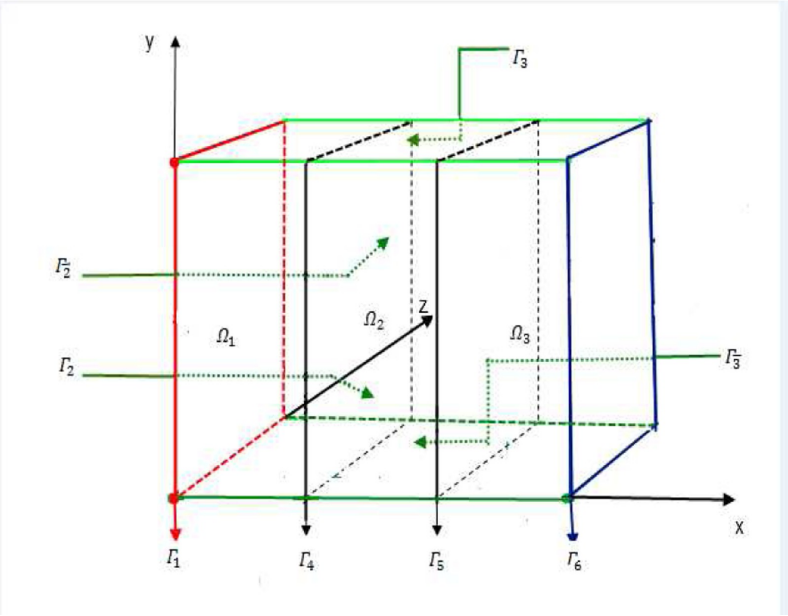


Fig. 2. Global representation of the domain.

space. The solution of this equation requires the knowledge of initial condition and six boundary conditions for each subdomain Ω_I, $I = 1, 2, 3$ (see Fig. 2).

The solidification of steel in continuous casting is well described by considering the previous boundary value problems equipped with non-homogeneous linear and nonlinear mixed Dirichlet–Neumann and Fourier linear and nonlinear boundary conditions. Indeed for the continuous casting problem, due to radiation phenomenon, well described by the Stefan law on some parts of the boundaries, the linear part of the continuous operator modeling the heat diffusion is perturbed by a diagonal increasing non-linear continuous operator, this last property resulting from the positivity of the temperature in each area. Nevertheless note that the positivity of the temperature in each zone Ω_I, $I = 1, 2, 3$ is not obvious to prove; then we consider this positivity as an assumption and the mathematical model is then changed as three coupled variational inequalities defined in three convex sets.

Consequently we have to solve successively three coupled strong nonlinear boundary value problems on each zone $\Omega_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$ in order to find the temperature $u_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$, in each subdomain of the boundary value problems governing the temperature. Besides taking into account both the constraint of temperature positivity and the nonlinearity derived from the radiation phenomenon are not convenient for the analysis of the resolution algorithms. The more convenient formulation of these nonlinearities is constituted by a multivalued formulation of the problem, obtained by a perturbation of the continuous operators arising in the physical model by a diagonal multivalued monotone continuous operator; this last operator is in fact the subdifferential of the indicator function of the convex sets taking account of the positivity of the temperature; for more details the reader is referred to Ref. [5]. After discretization at each time, on one hand of the evolution part of the problems by an implicit time marching scheme when the thermal conductivity coefficient is constant or by a semi-implicit time marching scheme otherwise, respectively, and on the other hand by using classical or adapted finite difference scheme for the spatial discretization with an uniform mesh, respectively, we have to solve, by a numerical way, a sequence of stationary multivalued nonlinear problems. So, finally, at each time step, we have to solve three sparse large coupled strongly nonlinear systems.

Owing to the great size of such system, due to the difficulty of storage of large matrices, an iterative algorithm is more appropriate for the solution of the algebraic systems to solve and may be used in order to reduce the computation time at each time step. In the sequential context, we have implemented sequential projected point Newton - relaxation algorithms, corresponding to a coupling of the projected point Newton method for points belonging to the part of the boundary where the Stefan law occurs, with the projected relaxation method, like the point Gauss–Seidel method, for the interior points of the zones.

Besides, in order to obtain accurate results, it is necessary to choose very small spatial discretization step-size, which consequently leads to solve very large multivalued algebraic systems; such numerical solution is then time consuming. Thus, in order to reduce the elapsed time and taking into account the properties of the discretization matrices on one hand and, due to the positivity of the temperature on each zone $\Omega_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$, those of monotony for the nonlinear discretized operators modeling the Stefan law on some parts of the boundary and also the well-known monotony of the subdifferential of the indicator function of the convex sets on the other hand, we can state sufficient conditions for the convergence of the considered parallel synchronous and asynchronous multisplitting and two-stage methods for the solution of multivalued nonlinear algebraic systems in a general topological context extending the initial results of [6] to the case where the finite dimensional space is normed by nonhilbertian norms more convenient to use. Note that such kind of methods, based on multisplitting method, allows a unified presentation and analysis of classical subdomain methods obtained on one hand by considering nonoverlapping subdomains and on the other hand by considering the implementation of the Schwartz alternating method; for more details the reader is referred to Refs. [7–17]. Note that, in these previous works the algebraic systems are singlevalued; nevertheless in the present study, taking into account the positivity constraint of the solution by the perturbation of a diagonal multivalued continuous operator extends many previous works in this case. Moreover the paper [6] is also extended to a nonhilbertian context.

The present paper is organized as follows. In Section 2, we present the mathematical model of the solidification of steel in continuous casting in both cases where the thermal conductivity coefficient is constant and when it is dependent of the temperature; this section continues with a brief presentation of a multivalued formulation of the problem taking into account the positivity constraints of the temperature; for more details see [5]. In Section 3, we present the tools allowing the numerical solution, particularly the discretization of the linear and nonlinear continuous operator governing the heat diffusion and also the numerical sequential and parallel methods used for the solution of large scale algebraic systems; particularly the general parallel synchronous and asynchronous multisplitting and two-stage methods are presented and analyzed for the solution of algebraic systems perturbed by increasing diagonal multivalued discretized operators in the nonhilbertian context. This section is followed by the presentation of the parallel implementation of the algorithms. In the next section we will present the results of sequential and parallel experiments. The last section is devoted to the presentation of a general conclusion.

2. Mathematical model

2.1. Physical formulation

Let $\Omega \subset \Re^d$, $d = 3$, be a bounded open domain with boundary denoted by $\partial\Omega$. We consider that $\Omega = \bigcup_{\mathcal{I}=1}^{3} \Omega_{\mathcal{I}}$ such that $\Omega_{\mathcal{I}} \cap \Omega_{\mathcal{J}} = \emptyset$, $\mathcal{I} \neq \mathcal{J}$. For the target industrial application note that it is sufficient to consider that Ω has a parallelepipedic shape; so let

$\mathbf{x} \equiv (x, y, z)$

and for $\mathcal{I} = 1, 2, 3$, let

$\gamma_{\mathcal{I}} \equiv \Gamma_{\mathcal{I},2} \cup \Gamma_{\mathcal{I},3} \cup \Gamma_{\mathcal{I},\bar{2}} \cup \Gamma_{\mathcal{I},\bar{3}}$,

where for $\mathcal{I} = 1, 2, 3$,

$\Gamma_1 = \{0 \le y,\ z \le 1,\ x = 0\}$,

$$\Gamma_{\mathcal{I},2} = \{z = 0,\ 0 \le y \le 1,\ \frac{\mathcal{I}-1}{3} < x < \frac{\mathcal{I}}{3}\};\ \Gamma_{\mathcal{I},3} = \{y = 1,\ 0 \le z \le 1,\ \frac{\mathcal{I}-1}{3} < x < \frac{\mathcal{I}}{3}\},$$

$$\Gamma_{\mathcal{I},\bar{2}} = \{z = 1,\ 0 \le y \le 1,\ \frac{\mathcal{I}-1}{3} < x < \frac{\mathcal{I}}{3}\};\ \Gamma_{\mathcal{I},\bar{3}} = \{y = 0,\ 0 \le z \le 1,\ \frac{\mathcal{I}-1}{3} < x < \frac{\mathcal{I}}{3}\}.$$

Remark 1. Note that clearly $\Gamma_{\mathcal{I},2}$ (respectively $\Gamma_{\mathcal{I},\bar{2}}$) shows the front (respectively backward) of Ω; similarly $\Gamma_{\mathcal{I},3}$ (respectively $\Gamma_{\mathcal{I},\bar{3}}$) represents the upper side (respectively under side) of Ω. The symbols $\bar{2}$ and $\bar{3}$, are a convenient notation to simply define borders $\gamma_{\mathcal{I}}$, $\mathcal{I} = 2, 3$, and allows a unified notation of these; without this notation, the forthcoming description of the mathematical model (1)–(3) would be more complex to write.

Let also Γ_4 and Γ_5 be the interfaces between Ω_1 and Ω_2 on one hand, and between Ω_2 and Ω_3 on the other hand; thus Γ_4 and Γ_5 are defined by

$$\Gamma_4 = \{0 \le y,\ z \le 1,\ x = \frac{1}{3}\},\ \Gamma_5 = \{0 \le y,\ z \le 1,\ x = \frac{2}{3}\}.$$

The last frontier delimiting the domain Ω is defined by

$\Gamma_6 = \{0 \le y,\ z \le 1,\ x = 1\}$.

Then we have to find successively the temperature $u_{\mathcal{I}}$, solution in each subdomain $\Omega_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$ of the following boundary value problems:

in the liquid zone Ω_1

$$\begin{cases} \rho_1 c_1 \frac{\partial u_1}{\partial t} - div(\lambda_1 \nabla u_1) = 0, & in\ \ [0, t_{final}] \times \Omega_1, \\ u_1(x, t = 0) = u_{1,0}(x), & I.\ C. \\ -\lambda_1 \frac{\partial u_1}{\partial n} = \Phi_{imp}, & on\ \ \Gamma_1, \\ -\lambda_1 \frac{\partial u_1}{\partial n} = \psi_1(u_1), & on\ \ \gamma_1, \\ -\lambda_1 \frac{\partial u_1}{\partial n} = \Upsilon_1(u_1), & on\ \ \Gamma_4 \end{cases} \tag{1}$$

in the mushy zone Ω_2

$$\begin{cases} \rho_2 c_2 \frac{\partial u_2}{\partial t} - div(\lambda_2 \nabla u_2) = 0, & in \ \ [0, t_{final}] \times \Omega_2, \\ u_2(x, t = 0) = u_{2,0}(x), & I.\ C. \\ u_2 = u_{1|\Gamma_4}, & on \ \ \Gamma_4, \\ -\lambda_2 \frac{\partial u_2}{\partial n} = \psi_2(u_2), & on \ \ \gamma_2, \\ -\lambda_2 \frac{\partial u_2}{\partial n} = \Upsilon_2(u_2), & on \ \ \Gamma_5 \end{cases} \tag{2}$$

and *in the solid zone* Ω_3

$$\begin{cases} \rho_3 c_3 \frac{\partial u_3}{\partial t} - div(\lambda_3 \nabla u_3) = 0, & in \ \ [0, t_{final}] \times \Omega_3 \\ u_3(x, t = 0) = u_{3,0}(x), & I.\ C. \\ u_3 = u_{2|\Gamma_5}, & on \ \ \Gamma_5, \\ -\lambda_3 \frac{\partial u_3}{\partial n} = \psi_3(u_3), & on \ \ \gamma_3 \\ u_3 = u_{imp}, & on \ \ \Gamma_6 \end{cases} \tag{3}$$

where *I.C.* denotes the initial conditions, and

$$\psi_{\mathcal{I}}(u_{\mathcal{I}}) = h_{cv,\mathcal{I}}(u_{\mathcal{I}} - u_{ext}) + \xi\delta(u_{\mathcal{I}}^4 - u_{ext}^4), \quad \mathcal{I} = 1, 2, 3, \tag{4}$$

describe the radiation phenomenon, modeled by the Stefan law and for $\mathcal{I} = 1, 2$,

$$\Upsilon_{\mathcal{I}}(u_{\mathcal{I}}) = \frac{1}{R}(u_{\mathcal{I}} - u_{cst}).$$

The significance of the various parameters is given below: for $\mathcal{I} = 1, 2, 3$, $c_{\mathcal{I}}$ is the specific heat capacity, $\rho_{\mathcal{I}}$ is the metal density, $h_{cv,\mathcal{I}}$ is the coefficient of exchange by convection, $u_{\mathcal{I}}$ is the temperature, Φ_{imp} is the flux, u_{imp} is the temperature imposed, u_{ext} is the temperature of environment, u_{cst} is the constant temperature defined at the interface Γ_4 and Γ_5 between two consecutive subdomains where the thermal exchange is performed, R is the thermal resistance, ξ is the emissivity, δ is the constant of Stefan–Boltzman and $\vec{n}$ is the outward normal vector. In the sequel, such physical coefficients are assumed to be constant.

In the previous physical model $\lambda_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$ is the thermal conductivity coefficient which possibly depends of the temperature; in this last situation where $\lambda_{\mathcal{I}}$ is not constant, after experimental measurement, the thermal conductivity coefficient is approximated, at each time step p, by a piecewise linear function given by $\lambda_{\mathcal{I}}^p(u_{\mathcal{I}}) = a_{\mathcal{I}}^p + b_{\mathcal{I}}^p u_{\mathcal{I}}^p$, where $u_{\mathcal{I}}^p \in [u_{\mathcal{I}}^{p,min}, u_{\mathcal{I}}^{p,max}]$ (see Table 1). Such an approximation can provide a better accuracy for the numerical simulation. Table 1 displays the values of the coefficient $a_{\mathcal{I}}^p$ and $b_{\mathcal{I}}^p$. We show, in Fig. 3, the piecewise linear approximation of the thermal conductivity coefficient.

2.2. Multivalued formulation

Taking into account the physical reality of the continuous casting problems, we impose the realistic assumption that the temperature $u_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$ is a nonnegative function; then we can write the following assumption

$$u_{\mathcal{I}}(\mathbf{x}, t) \geq 0, \ \text{for} \ \mathcal{I} = 1, 2, 3; \tag{5}$$

while the problems (1)–(3) are unconstrained, we consider now in what follows constrained problems. In this case, we have to solve a PDE's inequality corresponding to an obstacle problem (see [5]). The solutions of these previous problems are therefore characterized by various formulations. Moreover, we consider also the temporal discretization of such problems (1)–(3) by an implicit time marching scheme if the conductivity coefficients are constant or by a semi-implicit time marching scheme if these last coefficients are dependent on the temperature in each area. Let us denote by $w_{\mathcal{I}}(\mathbf{x})$, for $\mathcal{I} = 1, 2, 3$ the solution of the stationary obstacle problems where according to assumption (5) we have

$$w_{\mathcal{I}}(\mathbf{x}, t) \geq 0, \ \text{for} \ \mathcal{I} = 1, 2, 3, \tag{6}$$

and in both cases, in the problems (1)–(3), the discretization with respect to the time, at each time $t_{p+1} = (p + 1)\Delta t$, is carried out classically by

$$\frac{\partial u_{\mathcal{I}}}{\partial t} \approx \frac{u_{\mathcal{I}}^{p+1} - u_{\mathcal{I}}^p}{\Delta t} + O(\Delta t), \ \mathcal{I} = 1, 2, 3. \tag{7}$$

So, at each time step, we have to solve the following stationary obstacle problems on each subdomain $\Omega_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$

$$\begin{cases} -div(\lambda_1 \nabla w_1) + \frac{\rho_1 c_1}{\Delta t} w_1 - g_1 \geq 0, \quad w_1 \geq 0, & e.\,w. \ \ in \ \Omega_1 \\ (-div(\lambda_1 \nabla w_1) + \frac{\rho_1 c_1}{\Delta t} w_1 - g_1) w_1 = 0, & e.\,w. \ \ in \ \Omega_1 \\ -\lambda_1 \frac{\partial w_1}{\partial n} = \Phi_{imp}, & on \ \ \Gamma_1 \\ -\lambda_1 \frac{\partial w_1}{\partial n} = \psi_1(w_1), & on \ \ \gamma_1 \\ -\lambda_1 \frac{\partial w_1}{\partial n} = \Upsilon_1(w_1), & on \ \ \Gamma_4 \end{cases} \tag{8}$$

$$\begin{cases} -div(\lambda_2 \nabla w_2) + \frac{\rho_2 c_2}{\Delta t} w_2 - g_2 \geq 0, \quad w_2 \geq 0, & e.\,w. \ \ in \ \Omega_2 \\ (-div(\lambda_2 \nabla w_2) + \frac{\rho_2 c_2}{\Delta t} w_2 - g_2) w_2 = 0, & e.\,w. \ \ in \ \Omega_2 \\ w_2 = w_{1|\Gamma_4}, & on \ \ \Gamma_4 \\ -\lambda_2 \frac{\partial w_2}{\partial n} = \psi_2(w_2), & on \ \ \gamma_2 \\ -\lambda_2 \frac{\partial w_2}{\partial n} = \Upsilon_2(w_2), & on \ \ \Gamma_5 \end{cases} \tag{9}$$

and

$$\begin{cases} -div(\lambda_3 \nabla w_3) + \frac{\rho_3 c_3}{\Delta t} w_3 - g_3 \geq 0, \quad w_3 \geq 0, & e.\,w. \ \ in \ \Omega_3 \\ (-div(\lambda_3 \nabla w_3) + \frac{\rho_3 c_3}{\Delta t} w_3 - g_3) w_3 = 0, & e.\,w. \ \ in \ \Omega_3 \\ w_3 = w_{2|\Gamma_5}, & on \ \ \Gamma_5 \\ -\lambda_3 \frac{\partial w_3}{\partial n} = \psi_3(w_3), & on \ \ \gamma_3 \\ w_3 = w_{imp}, & on \ \ \Gamma_6 \end{cases} \tag{10}$$

where, e.w. means everywhere, Δt is the step time, $g_{\mathcal{I}} = \frac{\rho_{\mathcal{I}} c_{\mathcal{I}}}{\Delta t} w_{\mathcal{I}}^{prec}$, and $w_{\mathcal{I}}^{prec}$, $\mathcal{I} = 1, 2, 3$ is given thanks to the result obtained at the previous time step.

We will now consider a different formulation of the previous stationary variational inequalities in which the notion of subdifferential mapping, recalled hereafter, will play a main role in the sequel for taking into account the constraints (6) on $w_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$ and the necessary projection on the closed convex set $\mathcal{K}_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$ associated to the constraint (6). Indeed, classically in convex optimization (see [18,19]) the problem (8)–(10) can be formulated by a multivalued problem in each subdomain $\Omega_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$, defined as follows, where

Table 1
Piecewise linear approximation of the thermal conductivity coefficient.

$\lambda_{\mathcal{I}}$ variable	$\lambda_{\mathcal{I}}^p(u_{\mathcal{I}}) = a_{\mathcal{I}}^p + b_{\mathcal{I}}^p u_{\mathcal{I}}^p$, $\mathcal{I} = 1, 2, 3$	
$\Omega_{\mathcal{I}}$	$[u_{\mathcal{I}}^{p,min}, u_{\mathcal{I}}^{p,max}]$	$(a_{\mathcal{I}}^p, b_{\mathcal{I}}^p)$
Ω_1	[1508, 1627]; [1418, 1508]	(19.6240, 0.0134); (−85.7667, 0.0833)
	[1400, 1418]; [1000, 1400]	(0.8889, 0.0222); (14.2000, 0.0130)
Ω_2	[800., 1000]; [700., 800.]	(20.7000, 0.0065); (73.1000, −0.0590)
Ω_3	[400., 700.]; [300., 400.]	(57.0000, −0.0360); (49.8000, -0.0180)
	[200., 300.]; [100., 200.]	(57.0000, −0.0420); (53.4000, −0.0240)
	[0., 100.]	(51.8000, −0.0080)

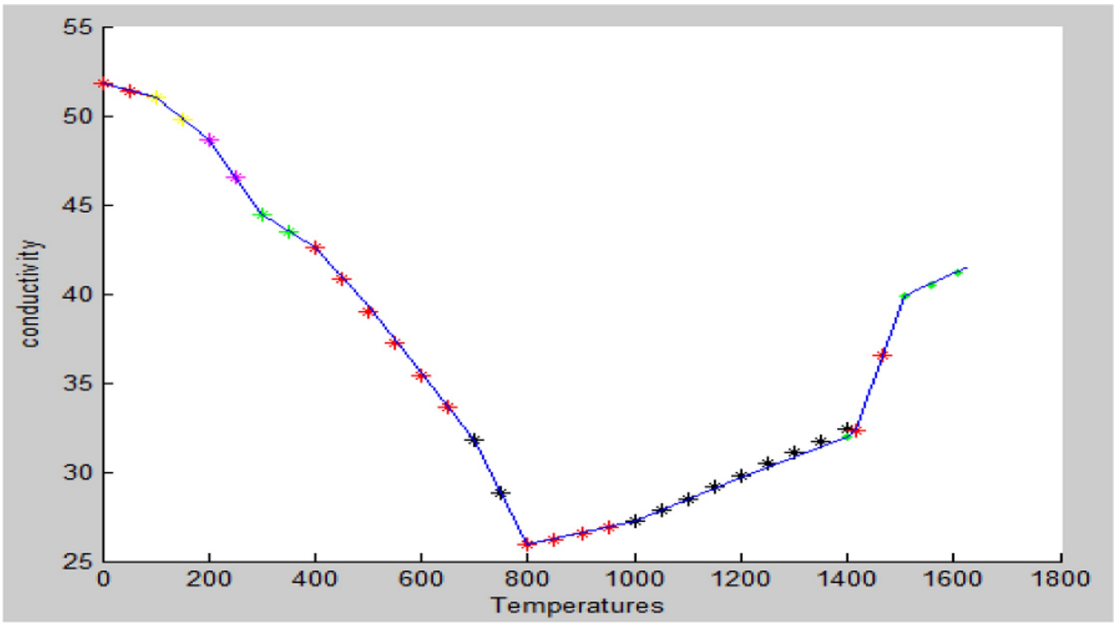


Fig. 3. Piecewise linear approximation of the thermal conductivity coefficient.

$\mathbf{E}_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$ denote appropriate vector spaces:
in the liquid zone Ω_1, we have to find $w_1 \in \mathbf{E}_1$

$$\begin{cases} -div(\lambda_1 \nabla w_1) + \frac{\rho_1 c_1}{\Delta t} w_1 - \bar{g}_1 + \partial\chi_{\mathcal{K}_1}(w_1) \ni 0 \\ -\lambda_1 \frac{\partial w_1}{\partial n} = \Phi_{imp}, & on \ \ \Gamma_1, \\ -\lambda_1 \frac{\partial w_1}{\partial n} = \psi_1(w_1), & on \ \ \gamma_1, \\ -\lambda_1 \frac{\partial w_1}{\partial n} = \Upsilon_1(w_1), & on \ \ \Gamma_4 \end{cases} \tag{11}$$

in the mushy zone Ω_2, we have to find $w_2 \in \mathbf{E}_2$

$$\begin{cases} -div(\lambda_2 \nabla w_2) + \frac{\rho_2 c_2}{\Delta t} w_2 - \bar{g}_2 + \partial\chi_{\mathcal{K}_2}(w_2) \ni 0 \\ w_2 = w_{1|\Gamma_4}, & on \ \ \Gamma_4 \\ -\lambda_2 \frac{\partial w_2}{\partial n} = \psi_2(w_2), & on \ \ \gamma_2, \\ -\lambda_2 \frac{\partial w_2}{\partial n} = \Upsilon_2(w_2), & on \ \ \Gamma_5 \end{cases} \tag{12}$$

in the solid zone Ω_3, we have to find $w_3 \in \mathbf{E}_3$

$$\begin{cases} -div(\lambda_3 \nabla w_3) + \frac{\rho_3 c_3}{\Delta t} w_3 - \bar{g}_3 + \partial\chi_{\mathcal{K}_3}(w_3) \ni 0 \\ w_3 = w_{2|\Gamma_5}, & on \ \ \Gamma_5 \\ -\lambda_3 \frac{\partial w_3}{\partial n} = \psi_3(w_3), & on \ \ \gamma_3, \\ w_3 = w_{imp}, & on \ \ \Gamma_6 \end{cases} \tag{13}$$

where $\partial\chi_{\mathcal{K}_{\mathcal{I}}}(w_{\mathcal{I}})$, $\mathcal{I} = 1, 2, 3$, are the sub differentials of the indicator functions $\chi_{\mathcal{K}_{\mathcal{I}}}$ of the convex subsets $\mathcal{K}_{\mathcal{I}}$, defined by

$$\chi_{\mathcal{K}_{\mathcal{I}}}(v_{\mathcal{I}}) \geq \chi_{\mathcal{K}_{\mathcal{I}}}(w_{\mathcal{I}}) + <v_{\mathcal{I}} - w_{\mathcal{I}}, w'_{\mathcal{I}}>_{\mathbf{E}_{\mathcal{I}} \times \mathbf{E}_{\mathcal{I}}\star}, \text{ for every } v_{\mathcal{I}} \in \mathbf{E}_{\mathcal{I}}, \tag{14}$$

where $<\ ,\ >_{\mathbf{E}_{\mathcal{I}} \times \mathbf{E}_{\mathcal{I}}\star}$ denotes the pairing[1] between $\mathbf{E}_{\mathcal{I}}$ and $\mathbf{E}^{\star}_{\mathcal{I}}$ the dual space of $\mathbf{E}_{\mathcal{I}}$ and where the indicator function of the convex subset $\mathcal{K}_{\mathcal{I}} \subset \mathbf{E}_{\mathcal{I}}$ for $\mathcal{I} = 1, 2, 3$, is given by,

$$\chi_{\mathcal{K}_{\mathcal{I}}}(w_{\mathcal{I}}) = \begin{cases} 0 \text{ if } w_{\mathcal{I}} \in \mathcal{K}_{\mathcal{I}} \\ +\infty \text{ otherwise} \end{cases}$$

[1] i.e. a bilinear form, from a normed vector space $\mathbf{E}_{\mathcal{I}} \times \mathbf{E}_{\mathcal{I}}\star$ onto $\Re$. Recall that if $\mathbf{E}_{\mathcal{I}}$ is an Hilbert space, then the pairing is the inner product of $\mathbf{E}_{\mathcal{I}}$.

Remark 2. Taking into account the values of the temperature in each area then obviously it seems not reasonable to think just one time that the temperature can be negative. Consequently the temperature is intuitively positive and even strictly positive. But the positivity of the temperature in each area $\Omega_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$ is not obvious to prove mathematically even if it is true physically; this last point must be proved rigorously, and in the considered application, it is not obvious. The maximum principle seems very hard to apply in this context due to dominant Neumann or Fourier boundary conditions, with on some parts non linear formulation due to the Stefan law. Moreover, we have also to take into account the coupling between the different area liquid, mushy and solid. Thus in this non standard formulation, even if we can consider in advance an analogous result to the one stated when we have Dirichlet boundary condition concerning the minimum and / or the maximum value of the temperature, due to the Neumann or Fourier boundary condition on some part of the boundary, we have no information on the value and also on the sign of the temperature. Consequently, the question to use the maximum principle to show the positivity of the temperature in each area is an open and very interesting problem. Thus, instead of using the maximum principle, we consider an additional assumption allowing to make complete the mathematical model of solidification of steel in continuous casting. Particularly such property ensures that the mapping $w_{\mathcal{I}} \to \psi_{\mathcal{I}}(w_{\mathcal{I}})$, $\mathcal{I} = 1, 2, 3$, are diagonal increasing continuous operators. Moreover, the multivalued formulation is of some interest only from a theoretical point of view, due to the fact that the sub-differential of the indicator function is a monotone mapping and classically to take into account the constraints arising in the mathematical model (see [18–20]); from a practical point of view, such formulation does not play any role in the implementation of the numerical algorithm, since we have simply to carry out a projection on the convex set defining the constraint to respect the positivity of the temperature. Thus, classically due to the perturbation of the diffusion operator by the sub-differential continuous operator of the indicator function of the convex sets $\mathcal{K}_{\mathcal{I}}$, $\mathcal{I} = 1, 2, 3$, we have to solve a multivalued problem; due both to the monotony of the sub-differential operator, combined to the fact that the mapping $w_{\mathcal{I}} \to \psi_{\mathcal{I}}(w_{\mathcal{I}})$, $\mathcal{I} = 1, 2, 3$, are diagonal increasing continuous

operators, and also to the properties of the discretization matrices presented below in the following Section 3.1, the analysis of the behavior of the general parallel relaxation algorithms is very easy and we will conclude in the sequel to the convergence of the considered numerical parallel iterative methods, well adapted to this kind of problem. Note that another analysis using the orthogonal projection operator does not work here due to the nonlinearity appearing in the model problem because of the radiation phenomenon modeled by Stefan's law since the associated fixed point mapping is hard to define correctly in such situation. Finally note also that the formulation of the model problem by a problem where the solution is subject to some constraint is as difficult to solve as the use of the maximum principle.

In (11)–(13), the projection on the convex sets $\mathcal{K}_I$, $I = 1, 2, 3$ being classically and formally formulated by the perturbation of the continuous operators ψ_I by a multivalued increasing diagonal continuous operator, so, in the sequel, we will mainly use these multivalued formulations (11)–(13) of the model problems in order to analyze the behavior of the iterative algorithms used for the solution of these problems. Note that for $I = 1, 2, 3$, the ith component of the subdifferential is given by:

$$(\partial\chi_{\mathcal{K}_I}(w_i))_i = \begin{cases} \varnothing, & \text{if } w_i < 0, \\ ]-\infty, 0], & \text{if } w_i = 0, \\ 0, & \text{if } w_i \in \mathcal{K}_I, \end{cases}$$

and the corresponding graph is presented in Fig. 4. Recall that the subdifferential $\partial\chi_{\mathcal{K}_I}(\ .\ )$ is a monotone continuous operator, in general multivalued, from $\mathbf{E}_I$ to $\mathbf{E}_I{}^{\star}$; for more details the reader is referred to Ref. [20]. Note that in Fig. 4 the graph of the subdifferential of $\chi_{\mathcal{K}_I}$ is effectively monotone. Note also that, thanks to assumption (6), the graph of $w_I \to \Psi_I(w_I)$ is also monotone (see Fig. 5).

3. Numerical solution.

3.1. Discretization.

In a general framework, and in order to properly model the physical aspects, we must distinguish two cases; firstly the case where the thermal conductivity coefficient is constant and secondly the case where it depends of the temperature. Note that, in both cases, the discretization with respect to the time, at each time $t_{p+1} = (p+1)\Delta t$, is carried out classically by using the scheme (7).

In the first case, the evolution problems (1)–(3) are discretized at each time $t_p = p\Delta t$, by an implicit time marching scheme and the spatial part of the linear operator describing the heat diffusion is approximated by using the classical seven points scheme with an uniform mesh h, the spatial discretization step-size. Such combined discretizations leads to the solution of a multivalued algebraic system described in (16) below.

In the second case where the thermal conductivity coefficient is not constant, for sake of simplicity, we consider the discretization with respect to the time by a semi-implicit time marching scheme; so, in order to take into account the variation of the thermal conductivity coefficient we will in the sequel establish spatial approximation schemes (see [21]). We consider a spatial discretization of the domain Ω with an uniform mesh h, where h denotes once again the spatial discretization step-size. For the spatial discretization of the linear continuous operators arising in (1)-(3), it is necessary to take into account that the coefficient of conductivity can not be constant in the domain Ω_I, $I = 1, 2, 3$. In order to obtain an estimate of the truncation error, assume the classical assumption $u_I \in C^4(\Omega_I)$; assume also that $\lambda_I(t, \mathbf{x}, u_I)$ is continuous. It should be noted that in order to define the scheme of spatial discretization by finite difference, we ask for more regularity in the solution than it actually has; however, such an assumption is classic and allows to obtain discretization error estimates. The discretization scheme is carried out by taking the mean of two intermediate schemes, called in the sequel forward-backward scheme and backward-forward scheme [21]. Let us consider first the discretization of $-\frac{\partial}{\partial x}\left(\lambda\frac{\partial u}{\partial x}\right)$ for $y = y_j$ and $z = z_k$ fixed. In order to simplify the notations, let us denote by $u(t_p, x_i, y_j, z_k) = u^p_{i,j,k}$ and $\lambda(t_p, x_i, y_j, z_k, u^p_{i,j,k}) = \lambda^p_{i,j,k}$.

• *Forward-backward scheme*

$$\begin{aligned} -\frac{\partial}{\partial x}\left(\lambda^p\frac{\partial u}{\partial x}\right)^{p+1}_{x_i} &\approx -\frac{1}{h}\left[\lambda^p_{i+1,j,k}\left(\frac{\partial u}{\partial x}\right)^{p+1}_{i+1,j,k} - \lambda^p_{i,j,k}\left(\frac{\partial u}{\partial x}\right)^{p+1}_{i,j,k}\right] + O(h) \\ &\approx -\frac{1}{h}\left[\lambda^p_{i+1,j,k}\left(\frac{u^{p+1}_{i+1,j,k} - u^{p+1}_{i,j,k}}{h}\right)\right. \\ &\quad \left. - \lambda^p_{i,j,k}\left(\frac{u^{p+1}_{i,j,k} - u^{p+1}_{i-1,j,k}}{h}\right)\right] + O(h) \\ &\approx -\frac{\lambda^p_{i,j,k}}{h^2}u^{p+1}_{i-1,j,k} + \left(\frac{\lambda^p_{i,j,k}}{h^2} + \frac{\lambda^p_{i+1,j,k}}{h^2}\right)u^{p+1}_{i,j,k} \\ &\quad - \frac{\lambda^p_{i+1,j,k}}{h^2}u^{p+1}_{i+1,j,k} + O(h) \end{aligned}$$

Note that the truncation error of the forward-backward scheme is $O(h)$.

• *Backward-forward scheme*

$$\begin{aligned} -\frac{\partial}{\partial x}\left(\lambda^p\frac{\partial u}{\partial x}\right)^{p+1}_{x_i} &\approx -\frac{1}{h}\left[\lambda^p_{i,j,k}\left(\frac{\partial u}{\partial x}\right)^{p+1}_{i,j,k} - \lambda^p_{i-1,j,k}\left(\frac{\partial u}{\partial x}\right)^{p+1}_{i-1,j,k}\right] + O(h) \\ &\approx -\frac{1}{h}\left[\lambda^p_{i,j,k}\left(\frac{u^{p+1}_{i+1,j,k} - u^{p+1}_{i,j,k}}{h}\right)\right. \\ &\quad \left. - \lambda^p_{i-1,j,k}\left(\frac{u^{p+1}_{i,j,k} - u^{p+1}_{i-1,j,k}}{h}\right)\right] + O(h) \\ &\approx -\frac{\lambda^p_{i-1,j,k}}{h^2}u^{p+1}_{i-1,j,k} + \left(\frac{\lambda^p_{i-1,j,k}}{h^2} + \frac{\lambda^p_{i,j,k}}{h^2}\right)u^{p+1}_{i,j,k} \\ &\quad - \frac{\lambda^p_{i,j,k}}{h^2}u^{p+1}_{i+1,j,k} + O(h) \end{aligned}$$

Note also that the truncation error of the backward-forward scheme is $O(h)$.

Then, the final discretization scheme is obtained by making the mean of the two previous schemes. Thus the second derivative with respect to x is then approximated by

$$-\frac{\partial}{\partial x}\left(\lambda^p\frac{\partial u}{\partial x}\right)^{p+1}_{x_i} \approx B^p_{i-1,j,k}u^{p+1}_{i-1,j,k} + \bar{B}^p_{i,j,k}u^{p+1}_{i,j,k} + B^p_{i+1,j,k}u^{p+1}_{i+1,j,k} + O(h^2) \tag{15}$$

with

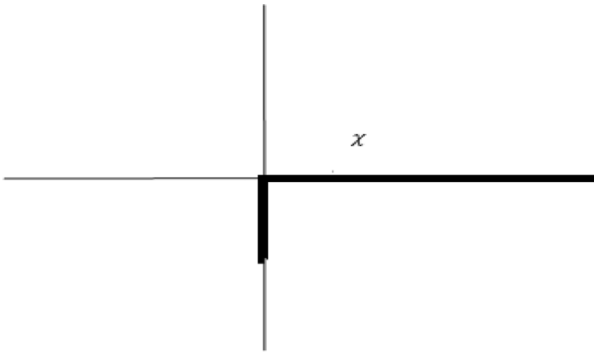


Fig. 4. Graph of the subdifferential of the function $\chi_{\mathcal{K}_I}$.

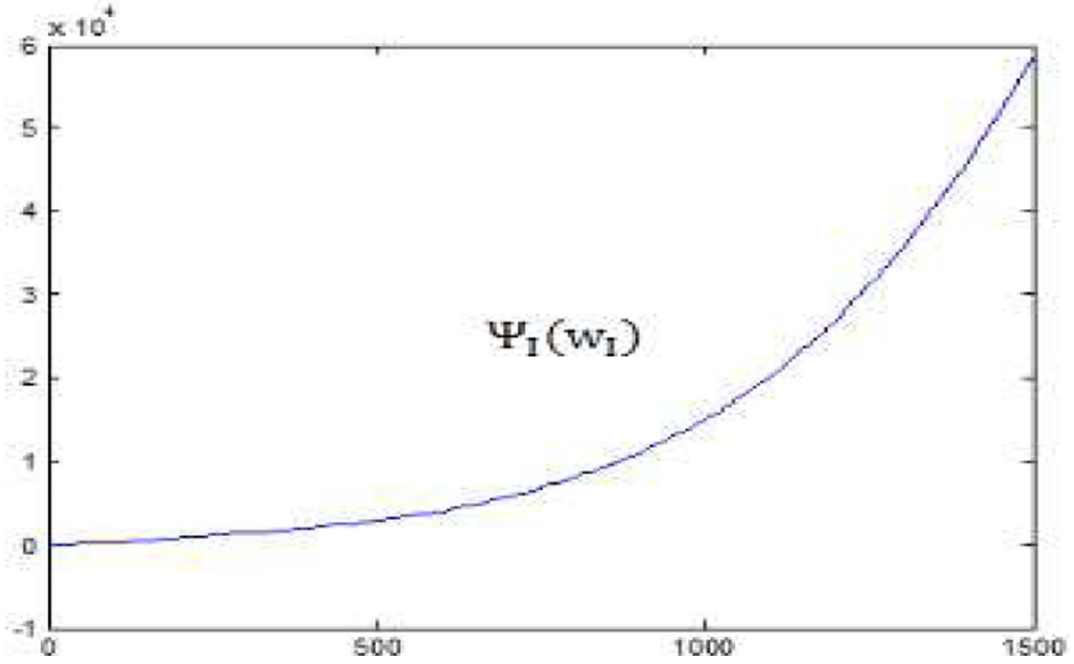


Fig. 5. Graph of function Ψ_I.

$$B^p_{i-1,j,k} = -\frac{1}{2h^2}(\lambda^p_{i-1,j,k} + \lambda^p_{i,j,k}), \quad \bar{B}^p_{i,j,k} = \frac{1}{2h^2}(\lambda^p_{i-1,j,k} + 2\lambda^p_{i,j,k} + \lambda^p_{i+1,j,k}), \quad B^p_{i+1,j,k} = -\frac{1}{2h^2}(\lambda^p_{i,j,k} + \lambda^p_{i+1,j,k}),$$

where $\bar{B}^p_{i,j,k}$ represents the approximation of the second derivative with respect to x. Note that, due to the fact that the truncation error of the forward-backward and of the backward-forward schemes are equal in absolute value, but with opposite signs, by making the mean of the two previous schemes, then the truncation error of the final scheme is $O(h^2)$.

Similarly the other second derivative with respect to y and z are approximated in the same way. For $\mathcal{I} = 1, 2, 3$, let us denote by $\mathcal{B}_I{}^p$ the heptadiagonal spatial discretization matrix defined by

$$\begin{cases} B^p_{i-1,j,k} = -\frac{1}{2h^2}(\lambda^p_{i-1,j,k} + \lambda^p_{i,j,k}); B^p_{i+1,j,k} = -\frac{1}{2h^2}(\lambda^p_{i+1,j,k} + \lambda^p_{i,j,k}); \\ B^p_{i,j-1,k} = -\frac{1}{2h^2}(\lambda^p_{i,j-1,k} + \lambda^p_{i,j,k}); B^p_{i,j+1,k} = -\frac{1}{2h^2}(\lambda^p_{i,j+1,k} + \lambda^p_{i,j,k}); \\ B^p_{i,j,k-1} = -\frac{1}{2h^2}(\lambda^p_{i,j,k-1} + \lambda^p_{i,j,k}); B^p_{i,j,k+1} = -\frac{1}{2h^2}(\lambda^p_{i,j,k+1} + \lambda^p_{i,j,k}); \\ B^p_{i,j,k} = \frac{1}{2h^2}(\lambda^p_{i-1,j,k} + \lambda^p_{i,j-1,k} + \lambda^p_{i,j,k-1} + 6\lambda^p_{i,j,k} + \lambda^p_{i+1,j,k} + \lambda^p_{i,j+1,k} + \lambda^p_{i,j,k+1}). \end{cases}$$

Finally, at each time step, using an implicit or a semi-implicit time marching scheme, depending on whether or not the thermal conductivity coefficient depends of temperature, we have to solve three sparse strongly nonlinear systems where for $\mathcal{I} = 1, 2, 3$, the global spatial discretization matrix $\mathcal{A}^p_I$ in each subdomain Ω_I is defined by

$$\mathcal{A}^p_I = (\frac{\rho_I c_I}{\Delta t} \cdot Id + \mathcal{B}_I{}^p), \ \mathcal{I} = 1, 2, 3. \tag{16}$$

For the continuous casting problem, the Stefan condition is also taken into account by the perturbation of the linear part of the discretized system by a diagonal increasing discretized operator Ψ_I; since U_I representing the values of the temperature in the area Ω_I, $\mathcal{I} = 1, 2, 3$, is assumed to be nonnegative, then at each time step p, we have to solve the following algebraic systems

$$\Psi_I(U^{p+1}_I) + \mathcal{A}^p_I U^{p+1}_I + \partial\chi_{K_I}(U^{p+1}_I) - G_I \ni 0, \quad \mathcal{I} = 1, 2, 3, \tag{17}$$

where for $\mathcal{I} = 1, 2, 3$, $\mathcal{A}^p_I \in \mathcal{L}(\Re^{\aleph_I})$, the space of linear operator in $\Re^{\aleph_I}$, are strictly diagonally dominant M-matrices, $\aleph_I$, $\mathcal{I} = 1, 2, 3$, denotes the dimension of the matrices $\mathcal{A}^p_I$, $U^{p+1}_I \in \Re^{\aleph_I}$ is the value of U_I at time step $(p + 1)$, $\partial\chi_{K_I}(U^{p+1}_I)$ results from the discretization of the subdifferential of the indicator function χ_{K_I} and $G_I \in \mathcal{R}^{\aleph_I}$ are derived from the time marching scheme.

3.2. General information on the numerical algorithms used

As previously said, owing to the great size of such systems (17), due to the propagation of rounding errors which can perhaps distort the numerical results, iterative algorithm is prefered and may be used in order to reduce the computation time at each time step. Moreover, since the matrices are sparse, using iterative methods leads to a low cost of storage.

For the numerical solution we distinguish in the sequel the sequential and the parallel methods which will be presented in an unified way.

In the sequential context, we have implemented sequential point projected Newton-relaxation algorithms, corresponding to a coupling of the projected Newton method for the points belonging to the boundaries γ_I with the projected relaxation method, like the projected Gauss-Seidel method, for the interior points of Ω_I, $\mathcal{I} = 1, 2, 3$ in which the grid points are numbered using the lexicographical ordering.

Moreover, very small spatial discretization step-size, leads to obtain very large and sparse algebraic systems like (17). Due to possible very large computation times, elapsed time reduction can be obtained by considering the parallelization of the numerical algorithms used. We can consider synchronous or asynchronous numerical parallel methods.

More precisely, in a parallel context, we are in the present study in a specific situation which can be summarized as follows: we have to consider a global iteration called “outer iteration”, resulting in a distribution of subproblems between processors and a local iteration resulting from the numerical iterative solution of each subproblem entrusted to it, specific to each processor. So, in our case, each subproblem entrusted to one processor, is solved by an iterative algorithm, e.g. a point projected Newton method for the point belonging to

the discretization points of γ_I and a point projected relaxation method for the interior points of Ω_I, $I = 1, 2, 3$; these iterative processes that are not necessarily carried out until its convergence, are called two-stage methods; for an algebraic linear system defined with an M-matrix [22] perturbed by a diagonal monotone discretized operator such algorithms have been analysed both for the synchronous and asynchronous case by several authors and the reader is referred to Refs. [6–17]. In the present work we extend the results of [6], stated in an hilbertian context, to the case where the finite dimensional spaces where the algebraic equations are defined, are normed by nonhilbertian norms, more convenient to use.

Taking into account the properties of the matrices $\mathcal{A}_I$, $I = 1, 2, 3$, on one hand and the monotony of the discretized operators $\partial\chi_{K_I}(U_I)$ and $\Psi_I(U_I)$, $I = 1, 2, 3$ on the other hand, for both previous iterative methods, we present an analyze of the behaviour of the sequential and parallel synchronous and asynchronous two-stage method applied to the solution of (17).

3.3. Multisplitting method

3.3.1. Preliminaries

In the present section we consider the general following problem

$$\Psi(U) + \mathcal{A}U + \partial\chi_K(U) - G \ni 0, \tag{18}$$

similar to the problem (17), in which Ψ is an increasing nonlinear discretized diagonal operator describing the Stefan law on some part of the boundaries, $\mathcal{A} \in \mathcal{L}(\mathfrak{R}^{\aleph})$ is a strictly diagonally dominant M-matrix, $\aleph \in \mathbb{N}$, denoting the dimension of the matrix $\mathcal{A}$, $U \in \mathfrak{R}^{\aleph}$, $\partial\chi_K(U)$ results from the discretization of the subdifferential of the indicator function χ_K of the convex set $\mathcal{K}$ and $G \in \mathcal{R}^{\aleph}$ is derived from the time marching scheme. For a given vector V, let us associate to problem (18) an implicit fixed point mapping $V \to F(V)$, defined as follows

$$G + \mathcal{N}V \in \mathcal{M}U + \Psi(U) + \partial\chi_K(U) \Leftrightarrow U = F(V)\ ,\ \forall\ V \in E = \mathcal{R}^{\aleph}, \tag{19}$$

where

$$\mathcal{A} = \mathcal{M} - \mathcal{N}$$

denotes a splitting of the matrix $\mathcal{A}$. In the sequel we recall the following classical definition (see [22])

Definition 1. Let $E = \mathfrak{R}^{\aleph}$ and $\mathcal{A} \in L(E)$; the decomposition $\mathcal{A} = \mathcal{M} - \mathcal{N}$ is called a splitting if $\mathcal{M}$ is nonsingular. It is called a convergent splitting if the spectral radius of $\mathcal{M}^{-1}\mathcal{N}$, denoted in what follows $\nu(\mathcal{M}^{-1}\mathcal{N})$ satisfy $\nu(\mathcal{M}^{-1}\mathcal{N}) < 1$. A splitting $\mathcal{A} = \mathcal{M} - \mathcal{N}$ is called

(i) regular if $\mathcal{M}^{-1} \geq 0$ and $\mathcal{N} \geq 0$,
(ii) weak regular if $\mathcal{M}^{-1} \geq 0$ and $\mathcal{M}^{-1}\mathcal{N} \geq 0$.

Remark 3. Clearly, a regular splitting is a weak regular splitting, but the converse is not true.

The following result will be very usefull in the sequel.

Lemma 1. *Assume that $\mathcal{A}$ is an M-matrix admitting a weak regular splitting $\mathcal{A} = \mathcal{M} - \mathcal{N}$; then, there exists a strictly positive vector $\vartheta \in \mathfrak{R}^{\aleph}$ and a scalar $\nu \in [0, 1]$ such that the following inequality holds*

$$\mathcal{M}^{-1}\mathcal{N}\vartheta = \mathcal{J}\vartheta \leq \nu\vartheta. \tag{20}$$

Proof. $\mathcal{A}$ being an M-matrix, so there exists a strictly positive vector ϑ such that $r = \mathcal{A}\vartheta = (\mathcal{M} - \mathcal{N})\vartheta > 0$. Note that $\mathcal{M}^{-1}r \geq 0$. Moreover if $\mathcal{M}^{-1}r = 0$, for all $k \in \{1, ..., \aleph\}$, such that $\sum_{i=1}^{\aleph} (\mathcal{M}^{-1})_{k,i} r_i = 0$, so that all lines of $\mathcal{M}$ are null, in contradiction with the nonsingularity of $\mathcal{M}$. Thus $\mathcal{M}^{-1}r > 0$; therefore

$$\mathcal{M}^{-1}(\mathcal{M} - \mathcal{N})\vartheta > 0 \Rightarrow \mathcal{M}^{-1}\mathcal{N}\vartheta < \vartheta,$$

and, since $\mathcal{A}$ is an M-matrix, there exist both a strictly positive vector ϑ and a positive real number $\nu < 1$ such that (20) is valid, which achieves the proof. □

Thanks to the previous result, we have the following Proposition:

Proposition 1. *Let us assume that*

$$\mathcal{A} \text{ is an M-matrix,} \tag{21}$$

$$U \to \Psi(U) \text{ is an increasing discretized operator,} \tag{22}$$

$$U \to \partial\chi_K(U) \text{ is a multivalued monotone maximal discretized operator,} \tag{23}$$

then the fixed point mapping $V \to F(V)$, defined by (19) is contractive, with constant of contraction equal to ν. So there exists one and only one fixed point U^*, also unique solution of the discretized problem (18).

Proof. For sake of generality we consider that the space E is normed by any not necessarily hilbertian norm; let us denote by $< ., . >$ the pairing between E and $E^{\star}$ its topological dual space; $\tilde{G}(U)$ being the duality map, we have

$$\tilde{G}(U) = \{\tilde{g}(U) \in E^{\star}\ |\ < U, \tilde{g}(U) > = \|U\| \text{ and } \|\tilde{g}\|^{\star} = 1\},$$

where $\|U\|$ and $\|\tilde{g}\|^{\star}$ denote the norms of U and of $\tilde{g}(U)$ in E and $E^{\star}$ respectively. For a point decomposition of U, let $|.|$ be the absolute value in $\mathbb{R}$ such that the previous definition can be written as follows

$$\bar{G}(u) = \{\bar{g}(u) \in \mathbb{R}\ |\ < u, \bar{g}(u) > = |u| \text{ and } |\bar{g}|^{\star} = 1\}.$$

In the sequel let us denote by u_i and v_i, for $i = 1, ..., \aleph$ the components of the vectors U and V; in the same way we shall denote by $m_{i,\ j}$ and $n_{i,\ j}$ the entries of matrices $\mathcal{M}$ and $\mathcal{N}$. Considering any $U = F(V)$ and $\acute{U} = F(\acute{V})$ we can write

$$\mathcal{M}U + \Psi(U) + \omega(U) = \mathcal{N}V + G, \text{ where } \omega(U) \in \partial(\chi_K)(U), \tag{24}$$

and similarly

$$\mathcal{M}\acute{U} + \Psi(\acute{U}) + \omega(\acute{U}) = \mathcal{N}\acute{V} + G, \text{ where } \omega(\acute{U}) \in \partial(\chi_K)(\acute{U}), \tag{25}$$

which involve, by subtracting the two previous relations

$$\mathcal{M}(U - \acute{U}) + \Psi(U) - \Psi(\acute{U}) + \omega(U) - \omega(\acute{U}) = \mathcal{N}(V - \acute{V}).$$

By multiplying the previous system by $\tilde{g}(U - \acute{U}) \in \tilde{G}(U - \acute{U})$ we obtain

$$< \mathcal{M}(U - \acute{U}) + \Psi(U) - \Psi(\acute{U}) + \omega(U) - \omega(\acute{U}), \tilde{g}(U - \acute{U}) > = < \mathcal{N}(V - \acute{V}), \tilde{g}(U - \acute{U}) > .$$

Consider now the $\aleph$ following scalar equations resulting from the point decomposition; due to assumptions (22) and (23) we have for all $i \in \{1, ..., \aleph\}$

$$< \Psi_i(u_i) - \Psi_i(\acute{u}_i), \bar{g}_i(u_i - \acute{u}_i) > \geq 0, \quad \text{and} \quad < \omega_i(u_i) - \omega_i(\acute{u}_i), \bar{g}_i(u_i - \acute{u}_i) > \geq 0,$$

so that, we obtain the following inequality for all $i \in \{1, ..., \aleph\}$

$$< m_{i,i}(u_i - \acute{u}_i) + \sum_{\substack{j=1 \\ j \neq i}}^{\aleph} m_{i,j}(u_j - \acute{u}_j), \bar{g}_i(u_i - \acute{u}_i) > \leq < \sum_{j=1}^{\aleph} n_{i,j}(v_j - \acute{\mathbf{v}}_j), \bar{g}_i(u_i - \acute{u}_i) > ;$$

since, on one hand

$$< (u_i - \acute{u}_i), \bar{g}_i(u_i - \acute{u}_i) > = |u_i - \acute{u}_i|$$

and also, due to the fact that $m_{i,\ j} \leq 0$ for $i \neq j$

$$< m_{i,j}(u_j - \acute{u}_j), \bar{g}_i(u_i - \acute{u}_i) > \geq m_{i,j}|u_j - \acute{u}_j|$$

and on the other hand since classically

$$< (v_j - \acute{\mathbf{v}}_j), \bar{g}_i(u_i - \acute{u}_i) > \leq |v_j - \acute{v}_j|,$$

then finally we have

$$\sum_{j=1}^{\aleph} m_{i,i}|u_j - \acute{u}_j| \leq \sum_{j=1}^{\aleph} n_{i,j}|v_j - \acute{v}_j|,\ \forall\ i \in \{1, ..., \aleph\};$$

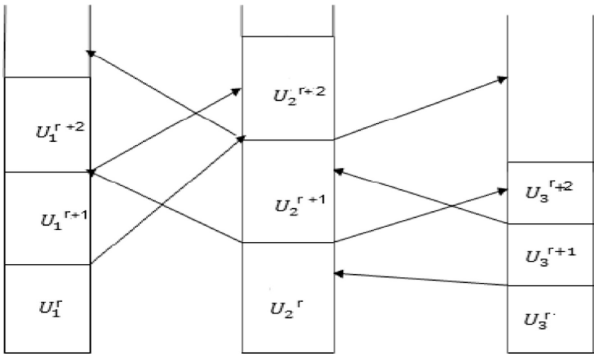


Fig. 6. Behavior of parallel asynchronous iterations.

so, we obtain in a vectorial norm formulation

$$\mathcal{M}|U - \acute{U}| \leq \mathcal{N}|V - \acute{V}|, \ \forall \ V, \acute{V} \in \mathcal{R}^{\aleph}, \tag{26}$$

which can still be written as follows

$$|U - \acute{U}| \leq \mathcal{M}^{-1}\mathcal{N}|V - \acute{V}| = \mathcal{J}|V - \acute{V}|, \ \forall \ V, \acute{V} \in \mathcal{R}^{\aleph};$$

Let us write componentwise this inequality

$$|u_i - \acute{u}_i| \leq \sum_{j=1}^{\aleph} \mathcal{J}_{i,j}|v_j - \acute{v}_j| = \sum_{j=1}^{\aleph} \mathcal{J}_{i,j}\vartheta_j \frac{|v_j - \acute{v}_j|}{\vartheta_j},$$

inequality that can be overestimated by

$$|u_i - \acute{u}_i| \leq \|V - \acute{V}\|_{\nu,\vartheta} \sum_{j=1}^{\aleph} \mathcal{J}_{i,j}\vartheta_j = \nu\vartheta_i\|V - \acute{V}\|_{\nu,\vartheta},$$

where $\|V\|_{\nu,\vartheta}$ is a uniform weighted norm defined by

$$\|V\|_{\nu,\vartheta} = \max_{i=1,\ldots,\aleph} \left(\frac{|v_i|}{\vartheta_i}\right), \tag{27}$$

with $\nu < 1$, according to the result of Lemma 1. Therefore, we obtain the following estimation

$$\|U - \acute{U}\|_{\nu,\vartheta} \leq \nu\|V - \acute{V}\|_{\nu,\vartheta}, \ \forall \ V, \ \acute{V} \in \mathcal{R}^{\aleph}, \ 0 \leq \nu < 1. \tag{28}$$

Then, using inequality (28), F is a contraction and we obtain a result of existence and uniqueness of both the fixed point mapping of F and of the solution of the discretized problem (18), which achieves the proof. □

3.3.2. Parallel asynchronous method

In the present section, we recall the formulation of the parallel asynchronous method associated to the solution of the following fixed point problem

$$\begin{cases} \text{Find } U^{\star} \in E \text{ such that} \\ U^{\star} = F(U^{\star}) \end{cases} \tag{29}$$

where $U \longrightarrow F(U)$ is a mapping from $D(F) \subset E$ to $D(F)$. So, we consider parallel algorithm, related to the natural block decomposition of the problem. For a general formulation of the algorithm, let α be a positive integer representing the number of blocks. In such algorithmic context the space E is now a finite product of α subspaces E_i such that $E = \prod_{i=1}^{\alpha} E_i$. Then let us consider the following block decomposition of U and $F(U)$

$$U = (U_1, U_2, \ldots, U_{\alpha}),$$

and

$$F(U) = F(U_1, U_2, \ldots, U_{\alpha}) = (F_1(U), F_2(U), \ldots, F_{\alpha}(U)).$$

In order to solve problem (29), let us consider now the iterative method defined as follows: let $U^0 \in D(F) \subset E$ be given, and then for all $r \in \mathcal{N}$ assume that we could get $U^1, \ldots, U^r$; U^{r+1} is recursively defined by

$$U_i^{r+1} = \begin{cases} F_i\left(U_1^{\rho_1(r)}, \ldots, U_j^{\rho_j(r)}, \ldots, U_{\alpha}^{\rho_{\alpha}(r)}\right) & \text{if } i \in s(r) \\ U_i^r & \text{if } i \notin s(r) \end{cases} \tag{30}$$

where

$$\begin{cases} \forall \ r \in \mathbf{N}, \ s(r) \subset \{1, \ldots, \alpha\} \text{ and } s(r) \neq \varnothing \\ \forall \ i \in \{1, 2, \ldots, \alpha\}, \ \text{the set } \{r \in \mathbf{N} | i \in s(r)\} \text{ is infinite} \end{cases} \tag{31}$$

and for all $j \in \{1, \ldots, \alpha\}$

$$\begin{cases} \forall \ r \in \mathbf{N}, \ \rho_j(r) \in \mathbf{N}, 0 \leq \rho_j(r) \leq r \text{ and } \rho_j(r) = r \text{ if } j \in s(r) \\ \lim_{r \to \infty} \rho_j(r) = \infty. \end{cases} \tag{32}$$

Note that $\mathcal{S} = (s(r))_{r \in \mathbf{N}}$ models the parallelism between the processors, since $s(r)$ contains the number of components relaxed at each step r while $\mathcal{R} = (\rho(r))_{r \in \mathbf{N}}$ models the asynchronism between the processors. The choice of the relaxed components may be guided by any criterion; the more efficient criterion is then to pick-up the most recently available values of the components just computed by the other processors.

The great interest of asynchronous parallel methods over synchronous ones lies in the fact that they eliminate processor latency when there are many unnecessary synchronizations, which causes periods of idle time of these latter. This situation is summarized in the following Figs.6 and 7; in Fig. 7 the grey area represents a phase of processor inactivity.

Remark 4. Such asynchronous iterations describe various classes of parallel algorithms, such as parallel synchronous iterations if $\forall \ j \in \{1, \ldots, \alpha\}, \ \forall \ r \in \mathbb{N}, \rho_j(r) = r$. In the synchronous context, for particular choice of $s(r)$, the previous fixed point method describes classical sequential methods; indeed, if $s(r) = \{1, \ldots, \aleph\}$, i.e. $\mathcal{S} = \{\{1, \ldots, \aleph\}, \ldots \{1, \ldots, \aleph\}, \ldots\}$ then (30)–(32) describes the sequential point Jacobi method while if $s(r) = r.\ mod(\aleph) + 1$, i.e. $\mathcal{S} = \{\{1\}, \{2\}, \ldots, \{\aleph\}, \{1\}, \ldots, \{\aleph\}, \ldots\}$ then (30)–(32) models the sequential point Gauss-Seidel method. In addition, if $\mathcal{S} = \{\{1\}, \ldots, \{\aleph\}, \{\aleph\}, \{\aleph - 1\}, \ldots, \{1\}, \{1\}, \ldots, \{\aleph\}, \{\aleph\}, \ldots, \{1\}, \ldots\}$ then (30)–(32) models the alternating direction method (ADI).

We can now recall the classical result

Proposition 2. *Let F: $D(F) \subset E \to D(F)$ and assume that*

(i) $D(F) = \prod_{i=1}^{\alpha} D_i(F)$, where $D_i(F)$ are closed and convex sets,

(ii) there exists $U^{\star}$ such that $U^{\star} = F(U^{\star})$,

(iii) for all $V \in D(F)$ we have $\|F(V) - U^{\star}\|_{\nu,\vartheta} \leq \nu\|V - U^{\star}\|_{\nu,\vartheta}$, $0 \leq \nu < 1$, where here $\|V\|_{\nu,\vartheta}$ is the weighted uniform norm defined analogously to (27) but, here associated to the α–block

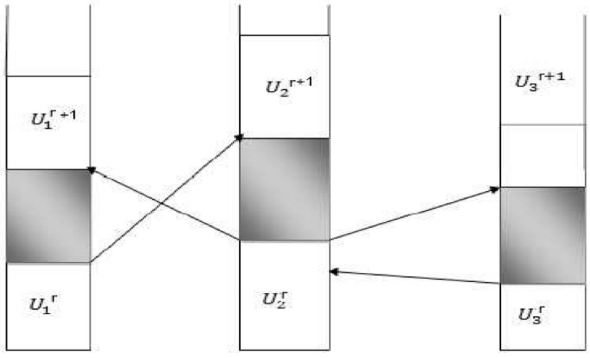


Fig. 7. Behavior of parallel synchronous iterations.

decomposition and then given by

$$\|V\|_{\nu,\tilde{\vartheta}} = \max_{i=1,\ldots,\alpha} \left(\frac{|\mathbf{v}_i|}{\tilde{\vartheta}_i}\right).$$

Then every parallel asynchronous algorithm (30)–(32) associated with F and starting from $U^0 \in D(F)$ converges to the fixed point $U^\star$ of F.

Proof. Since F is a contraction the proof follows by a straitghforward way from the application of a result of [23]. □

3.3.3. Parallel asynchronous multisplitting method

We consider now the solution of the problem (18) by parallel asynchronous multisplitting method when $\mathcal{A}$ satisfy assumption (21). Let us also consider the $m \in \mathbb{N}$ following regular splittings of $\mathcal{A}$

$$\mathcal{A} = \mathcal{M}^l - \mathcal{N}^l, \quad l = 1, \ldots, m,\ m \in \mathbf{N}, \tag{33}$$

where in what follows $\mathcal{M}^l$ are block diagonal, i.e. $\mathcal{M}^l = diag(\mathcal{M}^l_i)$, $i = 1, \ldots, \alpha$, $l = 1, \ldots, m$ and moreover $(\mathcal{M}^l)^{-1} \geq 0$ and $\mathcal{N}^l \geq 0$, corresponding to a regular splitting of $\mathcal{A}$. Let F^l: $\mathcal{D}(F^l) \to \mathcal{D}(F^l)$, $\mathcal{D}(F^l) \subset E$, $l = 1, \ldots, m$, be m corresponding fixed point mappings associated with problem (18) and defined by:

$$F^l(V) = U,\ l = 1, \ldots, m,\ U, V \in \mathcal{D}(F^l) \subset E, \tag{34}$$

by an analogous way than (24) and (25) and such that

$$\mathcal{M}^l U = \mathcal{N}^l V + B;$$

then, for all $l = 1, \ldots, m$, the mapping F^l associated with problem (18) are implicitly defined by

$$G + \mathcal{N}^l V \in \mathcal{M}^l U + \Psi(U) + \partial\chi_K(U) \Leftrightarrow U = F^l(V)\ ,\ \forall\ V \in E. \tag{35}$$

A formal multisplitting associated with problem (18) is defined by the collection of fixed point problems (see [6])

$$U^* = F^l(U^*),\quad l = 1, \ldots, m,\ U^* \in E. \tag{36}$$

Let now consider the space $\mathcal{E} = E^m$ like a finite product space of the m spaces E defined by

$$\mathcal{E} = \prod_{l=1}^{m} E_l,$$

where $E_l = E$ and consider the following $m-$block-decomposition of $\tilde{U} \in \mathcal{E}$, defined by

$$\tilde{U} = \{U^1, \ldots, U^l, \ldots, U^m\} \in \prod_{l=1}^{m} E_l,$$

where $U^1, \ldots, U^l, \ldots, U^m$ denote m vectors of E.

Assume that the space $\mathcal{E}$ is normed by the uniform weighted norm, defined by a similar way to (27) by

$$\|V\|_{\nu,\tilde{\vartheta}} = \max_{l=1,\ldots,m}\ (\max_{i=1,\ldots,\alpha} (\frac{|V_i^l|}{(\tilde{\vartheta}_l)_i})),\ \tilde{\vartheta}_l > 0, \tag{37}$$

where here $|V_i|$ denotes any hilbertian or nonhilbertian norm of V_i in E_i, $i = 1, \ldots, \alpha$ subspaces of E; assume that for $l = 1, \ldots, m$, F^l are contractive with respect to U^*, its fixed point, with constant $0 \leq \nu_l < 1$, so that the following inequality is verified

$$\|F^l(V) - U^*\|_{\nu_l,\tilde{\vartheta}_l} \leq \nu_l.\ \|V - U^*\|_{\nu_l,\tilde{\vartheta}_l},\ \text{for}\ l = 1, \ldots, m. \tag{38}$$

Definition 2. The extended fixed point mapping $\mathcal{F}$: $\mathcal{E} \to \mathcal{E}$ associated with the formal multisplitting is given as follows

$$\mathcal{F}(\tilde{V}) = \tilde{U},\ \text{ such that }\ U^l = F^l(\sum_{k=1}^{m} W_{lk} V^k),\ l = 1, \ldots, m,$$

where W_{lk} are nonnegative diagonal weighting matrices satisfying for all $l \in \{1, \ldots, m\}$

$$\sum_{k=1}^{m} W_{lk} = Id_l,$$

Id_l being the identity matrix in $L(E_l)$.

Since $F^l(\mathcal{D}(F^l)) \subset \mathcal{D}(F^l)$, then obviously $\mathcal{F}(\mathcal{D}(\mathcal{F})) \subset \mathcal{D}(\mathcal{F})$ where $\mathcal{D}(\mathcal{F}) = \prod_{l=1}^{m} \mathcal{D}(F^l)$.

Note that considerable saving in computational work may be possible, since a component of V^k needs not be computed if the corresponding diagonal entry of the weighting matrices are zero; the role of such weighting matrices may be regarded as determining the distribution of the computational work of the individual processors.

Let the following block-decomposition of the mapping $\mathcal{F}$

$$\mathcal{F}(V) = \{\mathcal{F}^1(V), \ldots, \mathcal{F}^l(V), \ldots, \mathcal{F}^m(V)\} \in \mathcal{E} = \prod_{l=1}^{m} E_l.$$

Note that for a particular choice of the weighting matrices W_{lk}, we can obtain various iterative methods and particularly on one hand a subdomain method without overlapping and on the other hand the classical Schwarz alternating method (see [6]). According with [6] the block Jacobi method corresponds to the following choice of $\mathcal{M}^l$

$$\mathcal{M}^l = diag\,(I_1, \ldots, I_{l-1}, \mathcal{A}_{l,l}, I_{l+1}, \ldots, I_m)\ ,$$

and to the choice of $W_{lk} \equiv \bar{W}_l$ given by

$$\bar{W}_l = diag\,(0, \ldots, 0, I_l, 0, \ldots, 0),$$

which in other words means that the entries of the weighting matrices are equal to one or to zero.

For the additive Schwarz alternating method more than one processor computes updated values of the same component, and the matrices W_l have positive entries smaller than one.

Let us recall now a result of [6]

Proposition 3. *Let us denote by $\tilde{U}^* = \{U^*, \ldots, U^*\}$ where U^* is the solution of (18); then if assumption (38) is verified, $\mathcal{F}$ is $\|\ .\ \|_{\nu,\tilde{\vartheta}}$ contractive with respect to $\tilde{U}^*$, where $\|\ .\ \|_{\nu,\tilde{\vartheta}}$ is defined by (37), the associated constant of contraction being*

$$\nu = \max_{1 \leq l \leq m} (\nu_l) < 1.$$

Then, using the result of Proposition 3, we have the following result

Corollary 1. *Consider the solution of problem (18); then if assumptions (21) –(23) are verified, in particular if the matrix A is an M-matrix, then the formal multisplitting $\mathcal{F}$ is contractive with respect to $\tilde{U}^*$ and any sequential, parallel synchronous or asynchronous multisplitting method starting from $\tilde{U}^0 \in \mathcal{D}(\mathcal{F})$ converge to the solution of (18).*

Proof. Indeed, if assumption (21) is verified, and if the other assumption of Proposition 1 are satisfied then each fixed point mapping F^l is contractive, for $l = 1, \ldots, m$ and then the proof is complete. □

Then, for $l = 1, \ldots, m$, and for $i = 1, \ldots, \alpha$, such asynchronous multisplitting method can be given by

$$\begin{cases} \mathcal{M}^l_i U_i^{l,r+1} + \Psi_i(U_i^{l,r+1}) + \omega_i(U_i^{l,r+1}) \ , \text{ if } i \in s(r) \\ \quad = \left(\mathcal{N}^l\left(\sum_{k=1}^{m} W_{lk} V_i^{k,\rho_k(r)}\right) + G\right)_i \\ U_i^{l,r+1} = U_i^{l,r} \qquad\qquad \text{if } \ i \notin s(r), \end{cases}$$

where $\omega_i(U_i^{l,r+1}) \in \partial(\chi_K)_i(U_i^{l,r+1})$

Now, from the discretization of the solidification steel problem, we have to solve, for $\mathcal{I} = 1, 2, 3$, the multivalued nonlinear algebraic systems (17). Each system (17) verifies all assumptions of Proposition 1 and Corollary 1. So we can conclude as follows

Corollary 2. *Consider the solution of the constrained algebriac systems (17); then, if assumptions (21)–(23) are verified, the sequential and parallel synchronous and asynchronous multisplitting method starting from every initial guess U_I^0 converge to the solution U_I^*, $I = 1, 2, 3$, of the problems.*

3.4. Two-stage multisplitting methods for pseudolinear problem

For pseudolinear problem such (18), we give in this section several particular classes of multisplitting methods called in the literature as two-stage methods; such two-stage methods corresponds to the fact that each subsystem is solved by an iterative relaxation method. Let us assume that we have m regular splitting $\mathcal{A} = \mathcal{M}^l - \mathcal{N}^l, \quad l = 1, ..., m$, of the matrix $\mathcal{A}$; in such splittings for $l = 1, ..., m$, $\mathcal{M}^l$ refers as previously said to a $\alpha-$block diagonal matrix. Then we have α subproblems related to the decomposition of $\mathcal{M}^l$ in α blocks. When the subsystems are solved iteratively, we consider in addition the splitting of $\mathcal{M}^l$ defined by

$$\mathcal{M}^l = \mathcal{P}^l - \mathcal{Q}^l, \ l = 1, ..., m,$$

where $\mathcal{P}^l$ are diagonal matrices. From an algorithmic point of view we can perform a fixed number q of inner iterations or alternatively, under the previous appropriate assumptions, perform iterations until convergence. Using such new decomposition, we can define an asynchronous two-stage method as follows

$$\mathcal{P}^l U^{l,r+1} + \Psi(U^{l,r+1}) + \omega(U^{l,r+1}) = \mathcal{Q}^l U^{l,r} + \mathcal{N}^l \tilde{U}^l + G, \ \omega(U^{l,r+1}) \in \partial(\chi_K)(U^{l,r+1})$$

where $\tilde{U}^l = (U_1^{l,\rho_1(r)}, ..., U_\alpha^{l,\rho_\alpha(r)})$. From a theoretical point of view, in the sequel, let us also assume that for $l = 1, ..., m$, $\mathcal{M}^l = \mathcal{P}^l - \mathcal{Q}^l$ is a weak regular splitting while, as previously said, $\mathcal{A} = \mathcal{M}^l - \mathcal{N}^l$ is a regular splitting.

Then , for $l = 1, ..., m$, and for $i = 1, ..., \alpha$, such asynchronous two-stage multisplitting method is given by

$$\begin{cases} \mathcal{P}_i^l U_i^{l,r+1} + \Psi_i(U_i^{l,r+1}) + \omega_i(U_i^{l,r+1}) = (\mathcal{Q}^l U^{l,r})_i + \left(\mathcal{N}^l \left(\sum_{k=1}^{m} W_{lk} V^{k,\rho_k(r)} \right) + G \right)_i & , \text{ if } i \in s(r) \\ U_i^{l,r+1} = U_i^{l,r} & \text{if } \ i \notin s(r), \end{cases}$$

where $\omega_i(U_i^{l,r+1}) \in \partial(\chi_K)_i(U_i^{l,r+1})$.

In order to prove the convergence of the two-stage method, it is necessary to state a similar result than the one obtained in Lemma 1.

Lemma 2. *Assume that $\mathcal{A}$ is an M-matrix admitting a regular splitting $\mathcal{A} = \mathcal{M} - \mathcal{N}$, where $\mathcal{M}$ is a block diagonal matrix; moreover, assume also that the matrix $\mathcal{M}$ admits a weak regular splitting $\mathcal{M} = \mathcal{P} - \mathcal{Q}$. Let $\tilde{\mathcal{J}} = \mathcal{P}^{-1}(\mathcal{Q} + \mathcal{N})$. Then, there exists a strictly positive vector $\tilde{\mathfrak{d}} \in \mathfrak{R}^{\aleph}$ and a scalar $\tilde{\nu} \in [0, 1]$ such that the following inequality*

$$\tilde{\mathcal{J}}\tilde{\mathfrak{d}} = \mathcal{P}^{-1}(\mathcal{Q} + \mathcal{N})\tilde{\mathfrak{d}} \leq \tilde{\nu}\tilde{\mathfrak{d}} \tag{39}$$

is valid.

Proof. $\mathcal{A}$ being an M-matrix, then the block diagonal matrix $\mathcal{M}$ is classically also an M-matrix, and $\mathcal{M}^{-1} \geq 0$. Besides, since $\mathcal{A}$ is an M-matrix, there exists a strictly positive vector $\tilde{\mathfrak{d}}$ such that

$$\tilde{r} = \mathcal{A}\tilde{\mathfrak{d}} = (\mathcal{M} - \mathcal{N})\tilde{\mathfrak{d}} = (\mathcal{P} - \mathcal{Q} - \mathcal{N})\tilde{\mathfrak{d}} > 0.$$

Note that $\mathcal{P}^{-1}\tilde{r} \geq 0$. Moreover if $\mathcal{P}^{-1}\tilde{r} = 0$, for all $k \in \{1, ..., \aleph\}$, such that $\sum_{i=1}^{\aleph} (\mathcal{P}^{-1})_{k,i}\tilde{r}_i = 0$, so that all lines of $\mathcal{P}$ are null, in contradiction with the nonsingularity of $\mathcal{P}$. Thus $\mathcal{P}^{-1}\tilde{r} > 0$; therefore, by a straightforward way, we obtain

$$\mathcal{P}^{-1}(\mathcal{P} - \mathcal{Q} - \mathcal{N})\tilde{\mathfrak{d}} > 0 \Rightarrow \mathcal{P}^{-1}(\mathcal{Q} + \mathcal{N})\tilde{\mathfrak{d}} < \tilde{\mathfrak{d}};$$

moreover since $\mathcal{M} = \mathcal{P} - \mathcal{Q}$ is a weak regular splitting of $\mathcal{M}$, then $\mathcal{P}^{-1}\mathcal{Q}$ is a nonnegative matrix; furthermore since $\mathcal{A} = \mathcal{M} - \mathcal{N}$ is a regular splitting of $\mathcal{A}$ then $\mathcal{N}$ is nonnegative and $\mathcal{P}^{-1}$ is a nonnegative matrix; it follows that $\mathcal{P}^{-1}\mathcal{N}$ is also a nonnegative matrix and finally $\mathcal{P} - \mathcal{Q} - \mathcal{N}$ is a weak regular splitting of $\mathcal{A}$; since $\mathcal{A}$ is an M-matrix, there exist both a strictly positive vector $\tilde{\mathfrak{d}}$ and a positive real number $\tilde{\nu} < 1$ such that (39) is valid, which achieves the proof. □

So, similarly to Corollary 1, thanks to the result of Lemma 2 which ensures that the fixed point mapping associated to the two-stage method is contractive, we can conclude briefly on the convergence of the sequential and parallel synchronous and asynchronous two-stage methods applied to the model problem as follows.

Corollary 3. *Consider the solution of the three constrained algebriac systems (17); then, since assumptions (21)–(23) are well verified, the sequential and parallel synchronous and asynchronous two-stage method starting from every initial guess U_I^0 converge to the solution U_I^*, $I = 1, 2, 3$, of the problems.*

4. Parallel implementation

In the present study the numerical solution of constant and variable coefficient models have been implemented in Fortran and use message passing programming on many independent computers, called nodes or processors. Due to their sparsity, the matrices are not globally stored. When using the finite difference method, in the first case where the coefficients of the diffusion equation are constant, it is sufficient to perform the calculation of the matrix coefficients only once at the begining of the numerical treatment. On the other hand, in the second case where the coefficients of the diffusion equation are not constant, due to the use of the semi-implicit scheme in selected time marching scheme, the matrix coefficients must be recalculated at each time step. Then, in each subdomain Ω_I seven subvectors are used in each processor for the storage of the matrix. Thus for each subdomain we have to solve algebraic systems with about the same number of unknowns. During parallel process, we divide then this size of each algebraic system by the number of processors. We can then measure the interest of using an iterative method that does not require the storage of the whole matrix. Indeed considering its sparse character, this one is here represented in memory at most by seven subvectors that minimize memory occupation. Thus, using iterative methods allows clearly to minimize the storage memory. Each processor starts its own program and communicates with other nodes by sending and receiving messages. Our parallel synchronous and asynchronous algorithms use the message-passing system called MPI (Message Passing Interface). MPI is a communication protocol which proposes for example point-to-point or collective communications and barrier synchronization. MPI can use TCP-based communications over IP interfaces. Our implementation is based on SPMD (single program, multiple data) paradigm, which is a commonly used approach for parallel and distributed applications. In SPMD, multiple autonomous processors simultaneously execute the same program with different data in order to obtain results faster.

Let n be the number of interior discretization points on each axis; then we have $n \times n \times n = n^3$ points on the complete domain Ω. This domain Ω is split into 3 subdomains Ω_I, $I = 1, 2, 3$ with everyone $n/3 \times n \times n$ discretization points . In our case, for the parallel simulation on α processors, each subdomain Ω_I, $I = 1, 2, 3$ is split into subdomains $\omega_{I,j}$, $j = 1, ..., \alpha$ with size about $(n/3)/\alpha \times n \times n$ points, such that $\Omega_I = \bigcup_j^{\alpha} \omega_{I,j}$, and the data are split accordingly into regular sets and each processor initializes its own data set.

In the first case where the coefficients of thermal conductivity are constant, the matrix and the right hand side creation have been implemented sequentially since this part of computation is not very intensive. In other words, only intensive computations have been parallelized. In the case where the coefficients of thermal conductivity are not constant, then each processor is going to work with a part of the

matrix and the associated right hand side, so they can be seen as a matrix and a vector filler that feeds a parallel linear system solver. This programming method greatly simplifies the solution of boundary value problems.

Two distinct parallel solutions are implemented by considering at each time step on one hand a subdomain method without overlapping and on the other hand a subdomain method with overlapping like the Schwarz alternating method; thus each zone Ω_I, $I = 1, 2, 3$ is divided into parallelepiped, each parallelepiped being identified by a pointer index system which refers to the number of points positioned on one of the axes and for the subdomain method with overlapping refers to the number of points considering the value of overlapping. Similarly, each of the zones Ω_I, $I = 1, 2, 3$ is identified by a similar pointer index system referencing either the first point or the last point of the zone on the same axis, respectively the first point minus the value of overlap or the last point plus the value of overlap of the zone on the same axis for the subdomain method with overlapping.

Firstly, the subdomain method with or without overlapping and also the synchronous or asynchronous method is chosen at the beginning of the solution. Then, each processor updates its assigned subblocks of components of the iterate vector. Thereafter, the transmission of the boundary blocks assigned to each processor to the contiguous blocks is realized.

Note that on each zone Ω_I, $I = 1, 2, 3$ a two-stage iterative method is implemented and note that the size of each subdomain is enough large in order to get to an efficient multiplicative behavior of the subdomain method.

Stopping criterion. An efficient stopping criterion is implemented using a supplementary convergence manager processor devoted to this task. We prefer use another processor to avoid delays in communication between computing processors. We have used the stopping criterion proposed by Bahi et al. [24] based on the use of a centralized method.

The convergence manager processor centralizes the local convergences of all the computing processors. Every processor determines its local convergence; a local convergence corresponds to the fact that the uniform norm of the difference between two successive updates is less than a given fixed threshold (see [25–27] for justifications). Note that under assumptions similar to those considered in the submitted study that considering the norm of the difference between two successive steps provides convenient stopping criteria of iterations; note also, that in the three previous references, when considering such stopping criterion, we have stated some estimate of the value of corresponding residue by using a property of approximate contraction induced by taking into account the propagation of round-off errors. In fact, it was stated in Refs. [25–27], that each step r belongs to a set $\mathcal{T}^r$ centered at the fixed point solution $U^\star$, these sets $\mathcal{T}^r$ being embedded when the index r increases, so that when r tends to infinity, the sets $\mathcal{T}^r$ tend to a limit set with small diameter. Indeed, in these previous references, both in the linear case (see [25,27]), when taking into account round off errors, and non-linear case [26], it is shown that this type of stopping criterion where the norm of two successive iterate vectors is considered is suitable; in addition, in these previous studies, when the convergence criterion is satisfied, an error estimate is drawn up which subsequently justifies the termination of iterations by considering the norm of the difference between two successive updates. Finally, from a numerical behavior, due to the strong diagonal dominance of the matrices, the speed of convergence is high, so that in such situation two successive steps cannot be identical, and in this case, the numerical criterion used is robust. On the contrary when the speed of convergence is low, two successive steps could be slightly identical and far from the searched solution, which is not the case in the present study due to strong diagonal dominance induced by physical parameters as it will be seen in the numerical experiments.

So, for a current time step, when a local convergence is reached by a computing processor, a message is sent to the convergence manager processor; nevertheless, after such local convergence, the process continues to iterate. Of course, if there is divergence after a convergence, a convergence cancellation message is sent. When the convergence manager processor detects the global convergence in all processors, it sends a stop message to computing processors.

This process is realized on each zone Ω_I, $I = 1, 2, 3$ successively. Then the next time step is performed for a given number of time steps for all Ω_I, $I = 1, 2, 3$. When all time steps are performed, the computation is ended; in this case, a final notification message is sent by one computing processors to the convergence manager processor. Note that the principle of the centralized stopping criterion is the same as well as for the synchronous algorithm than the asynchronous one.

Barrier synchronization. In order to use intrinsic MPI barrier synchronization and to permit the independence of the convergence manager processor, we implement a synchronization barrier using broadcast function. In fact, it is the convergence manager processor which takes care of this barrier.

So, at the end of a time step, all the processors are synchronized thanks to this barrier and send a message to the convergence manager processor. Then this processor restarts each computing processor.

In a similar way, we use this synchronization barrier when the convergence is reached in all subdomains constituting one of the zone Ω_I, $I = 1, 2, 3$.

The algorithm of the convergence manager processor can be summarized as follows:

```
Algorithm convergence manager processor
    receive command of processors
    if barrier then
        wait all computing processors
        send notification restart to computing processors
    else
        if convergence then
            receive local convergence
            send global convergence
        end if
    end if
```

Parallel implementation without and with overlapping for the problem with constants coefficients and variable coefficients. The algorithms of the numerical solution of constant coefficient model and variable coefficient model are identical except the compute of variable coefficients at each time step (see box below) and can be summarized as follows:

Algorithm on computing processor for constant coefficient model

compute initials conditions with physical parameters for each subdomain Ω_I
choose between without or with overlapping method
start a timer to measure computational time
for each time step t **do**
  for each subdomains $\Omega_I, I = 1, 2, 3$ **do**
    compute right hand side
    while no convergence **do**
      for each subdomain assigned to the processor **do**
        // by synchronous or asynchronous algorithm
        solve all linear systems
      end for
      send and receive boundary values from previous and next processors
      send local convergence to convergence manager processor
      receive global convergence from convergence manager processor
      barrier
    end while
    last processor send last boundary values of current subdomain Ω to first processor
    first processor receive last boundary values of previous subdomain Ω from last processor
    barrier
  end for
  barrier
end for
send stopping message to convergence manager processor
end the timer and get elapsed wall-clock since timerstart

Algorithm on computing processor for variable coefficient model

compute initials conditions with physical parameters for each subdomain Ω_I
choose between without or with overlapping method
start a timer to measure computational time
for each time step t **do**
  compute variable coefficients for each subdomain Ω_I
  for each subdomains $\Omega_I, I = 1, 2, 3$ **do**
    compute right hand side
    while no convergence **do**
      for each subdomain assigned to the processor **do**
        // by synchronous or asynchronous algorithm
        solve all linear systems
      end for
      send and receive boundary values from previous and next processors
      send local convergence to convergence manager processor
      receive global convergence from convergence manager processor
      barrier
    end while
    last processor send last boundary values of current subdomain Ω to first processor
    first processor receive last boundary values of previous subdomain Ω from last processor
    barrier
  end for
  barrier
end for
send stopping message to convergence manager processor
end the timer and get elapsed wall-clock since timerstart

5. Sequential and parallel experiments.

Experiments for the numerical simulation of the solidification of steel in continuous casting have been performed on the Grid'5000 platform located in France [28]; in this network every site hosts clusters and all sites are connected by highspeed communication. Grid'5000 is a large-scale and versatile testbed for experiment-driven research in all areas of computer science, with a focus on parallel and distributed computing including Cloud, HPC and Big Data. Grid'5000 provides access to a large amount of resources: 1000 nodes, 8000 cores, grouped in homogeneous clusters, and featuring various technologies: 10G Ethernet, Infiniband, GPUs, Xeon PHI. Fig. 8 illustrates the sites of the Grid'5000 architecture.

Sequential simulations have been performed on one machine and parallel simulations on single cluster and on distant coupled clusters of the Grid'5000 platform for different size of domains. The parallel experiments were carried out by coupling the grisou cluster of Nancy and the parasilo cluster of Rennes.

In Table 2, characteristics of each machines are summarized. Note

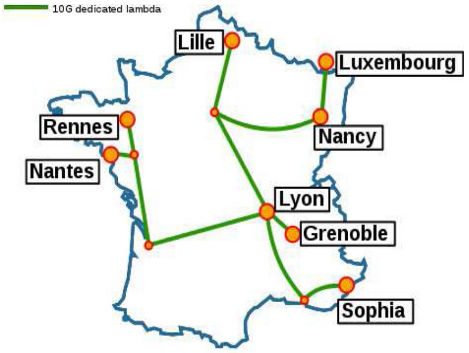


Fig. 8. Grid'5000 architecture.

Table 2
Characteristics of machines on each site.

Site	Cluster	Processors type	speed	cpu	core	RAM size
Nancy	grisou	Intel Xeon E5-2630v3	2.4 GHz	2	8	126 GB
Rennes	parasilo	Intel Xeon E5-2630v3	2.4 GHz	2	8	128 GB

that only machines having large RAM size could be used for large applications (see second paragraph of Section 5.2).

In each subdomain we have implemented a point relaxation method coupled with the point Newton method for the discretization points belonging to $\Gamma_{I,2} \cup \Gamma_{I,2}$ and $\Gamma_{I,3} \cup \Gamma_{I,3}$. For the solution of the linear systems we use the point Gauss–Seidel method, more performant in term of convergence speed compared to the point Jacobi method. The associated stopping criterion is implemented by considering the uniform norm of the difference between two successive steps, according to [25–27].

Taking into account the physical application, the domain Ω is split into three equal subdomains, such that $\Omega_1 = [0, \frac{1}{3}] \times [0, 1]^2$, $\Omega_2 = [\frac{1}{3}, \frac{2}{3}] \times [0, 1]^2$ and $\Omega_3 = [\frac{2}{3}, 1] \times [0, 1]^2$.

The physical parameters are given by Table 4, particularly for $I = 1, 2, 3$, the coefficients ρ_I,c_I, $h_{cv,I}$ and λ_I,: the unit of the metal density is $(kg.\ m^{-3})$, the specific heat is $(J.\ kg^{-1}.\ K^{-1})$, and the thermal conductivity is $(W.\ m^{-1}.\ K^{-1})$; the parameters $u_{ext} = 40.\ (C)$, $u_{imp} = 100.\ (C)$, $\Phi_{imp} = 50.$, $R = 15.$, $\sigma = 5.67e-08\ W.\ m^{-2}.\ K^{-4}$, $\xi = .18$, u_{cst} being given in each area by : $u_{in,1} = 1500.\ (C)$, $u_{in,2} = 900.\ (C)$ and $u_{in,3} = 600.\ (C)$, the common parameters in each subdomain.

We have chosen 100 times steps. Moreover in the numerical simulations, many size of the complete domain Ω are taken and summarized in Table 3. For the model problems, we have to solve large algebraic systems, and due to large value of ρ_I and c_I, the matrices $\mathcal{A}_I$ are strongly diagonal dominant.

Table 4
Physical parameters in each subdomain.

Subdomain	Ω_1	Ω_2	Ω_3	unit
Parameters	$c_1 = 688.$	$c_2 = 745.$	$c_3 = 485.$	$J.\ kg^{-1}.\ K^{-1}$
	$\rho_1 = 7340.$	$\rho_2 = 7659.$	$\rho_3 = 7834.$	$kg.\ m^{-3}$
	$h_{cv,1} = 25.$	$h_{cv,2} = 15.$	$h_{cv,3} = 5.$	$W.\ m^{-2}.\ K^{-1}$
λ_I constant	$\lambda_1 = 32.$	$\lambda_2 = 35.6$	$\lambda_3 = 51.$	$W.\ m^{-1}.\ K^{-1}$

Table 5
Elapsed time (sec) and relaxations with sequential algorithm with constant physical parameters and $\epsilon = 10^{-3}$.

	Sequential results on 1 machine	
n	Elapsed time	Relaxations
454	843.23	51 044 387 879
622	2 342.18	152 385 850 898
766	4 396.04	294 567 855 811
1051	14 463.79	854 073 499 394

5.1. Sequential method.

Numerical sequential experiments have been carried out on one machine of Rennes for different size of problem $n = 454$, 622, 766 and 1051. The performance of the algorithms are summarized in the following Tables 5 and 6.

We remark that the rate of the convergence of both the point Gauss–Seidel and point Newton methods is very fast respectively between 6 and 8 iterations per time step for Gauss–Seidel method and 2 iterations per time step for the Newton method (see [29]). For the point Newton method the fast convergence is due to the fact that the corresponding rate of convergence of this method is quadratic. For the point Gauss–Seidel method, this fast convergence is due, as previously said, to the large values of ρ_I and c_I, for $I = 1, 2, 3$. Indeed, in the present situation, the matrices $\mathcal{A}_I^p$ are strongly diagonal dominant, and since the diagonal discretized operators taking into account the radiation phenomenon and the constraint on the solution are monotone, the contraction constant associated to each fixed point operator in each area Ω_I, $I = 1, 2, 3$ are given by

Table 6
Elapsed time (sec) and relaxations with sequential algorithm with variable physical parameters and $\epsilon = 10^{-3}$.

	Sequential results on 1 machine	
n	Elapsed time	Relaxations
454	498.88	59 228 680 811
622	1 418.96	175 563 995 758
766	2 799.16	350 979 353 787
1051	8 421.29	1 078 465 869 190

Table 3
Different sizes of domains for the problems.

n	Number of points in Ω	Number of points in Ω_1
454	$454 \times 454 \times 454 = 93\ 576\ 664$	$152 \times 454 \times 454 = 31\ 329\ 632$
622	$622 \times 622 \times 622 = 240\ 641\ 848$	$208 \times 622 \times 622 = 80\ 471\ 872$
766	$766 \times 766 \times 766 = 449\ 455\ 096$	$256 \times 766 \times 766 = 150\ 209\ 536$
1051	$1051 \times 1051 \times 1051 = 1\ 160\ 935\ 651$	$350 \times 1051 \times 1051 = 386\ 610\ 350$
n	Number of points in Ω_2	Number of points in Ω_3
454	$151 \times 454 \times 454 = 31\ 123\ 516$	$150 \times 454 \times 454 = 30\ 917\ 400$
622	$207 \times 622 \times 622 = 80\ 084\ 988$	$206 \times 622 \times 622 = 79\ 698\ 104$
766	$255 \times 766 \times 766 = 149\ 622\ 780$	$254 \times 766 \times 766 = 149\ 036\ 024$
1051	$349 \times 1051 \times 1051 = 385\ 505\ 749$	$348 \times 1051 \times 1051 = 384\ 401\ 148$

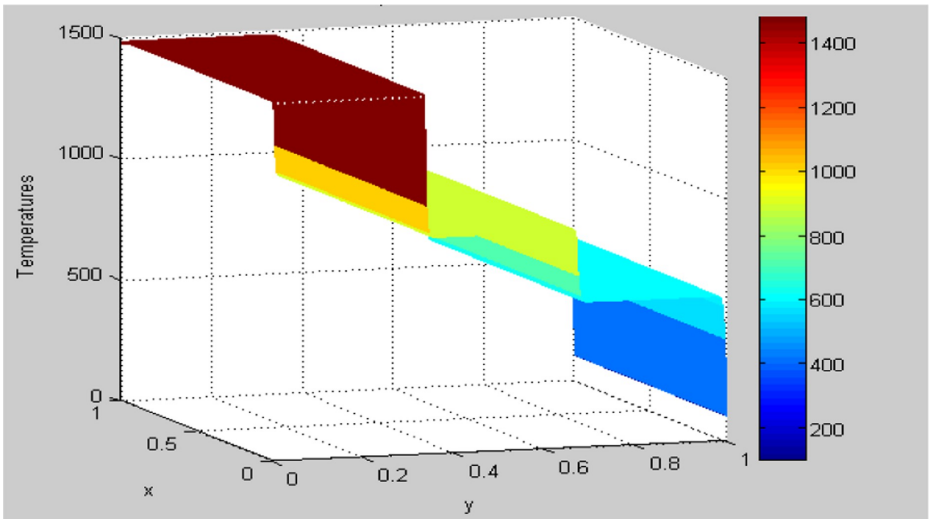


Fig. 9. Evolution of temperature with constant coefficients.

$$\nu_{\mathcal{I}} = \max_{\mathcal{I}}\Big(\sum_{k\neq i} |\frac{a^{\mathcal{I}}_{i,k}}{a^{\mathcal{I}}_{i,i}}|\Big) < 1,\ \mathcal{I} = 1,\ 2,\ 3,$$

where $a^{\mathcal{I}}_{i,k}$ are the entries of each matrix $\mathcal{A}^p_{\mathcal{I}}$, $\mathcal{I} = 1,\ 2,\ 3$. Consequently the rate of convergence is classically given by

$$R_\infty = -Ln(\nu_{\mathcal{I}}),\ \mathcal{I} = 1,\ 2,\ 3,$$

and thus, the convergence is fast. Note that, according to a result stated in [30], such estimate of the rate of convergence is still valid in the context of asynchronous parallel iterations. Consequently the elapsed time is short. Moreover we can notice that in the situation where the diffusion coefficient is variable, the elapsed time is significantly smaller than in the case where such parameter is constant; this situation is very difficult to analyze theoretically, since in this case the numerous and varied values of the entries of the matrices are difficult to visualize.

The Figs. 9 and 10 show the distribution of the temperature with constant or variable coefficients respectively. Note that the evolution of the temperature is slightly the same in both cases where the coefficients are constant or variable. In fact, such concordance is due to strongly diagonal dominance of the matrices; indeed in such matrices, due to strict diagonal dominance, the coefficient $\frac{\rho_{\mathcal{I}} \cdot c_{\mathcal{I}}}{\Delta t}$ is preponderant compared to the values of the entries of the spatial discretization matrices. Consequently the global matrices $\mathcal{A}^p_{\mathcal{I}}$ can be considered like diagonal matrices $\frac{\rho_{\mathcal{I}} \cdot c_{\mathcal{I}}}{\Delta t}$. *Id* perturbed by the spatial discretization matrices admiting low entries. This fact will have major consequences in the behavior of the parallel simulations presented below.

We can notice that in the simulation, the temperature is also slightly constant in *xoy* plane. This fact is intuitively physically predicable since the variation of the temperature is very small in each area.

5.2. Parallel method.

As previously said, in the considered parallel numerical simulations, many sizes of the complete domain Ω are taken (see Table 3) for the subdomain method without on one hand and with overlapping on other hand; each size is then characterized by the value of *n*.

In the target application, at the beginning of the parallel running, for each MPI process values of subvectors representing the matrix and

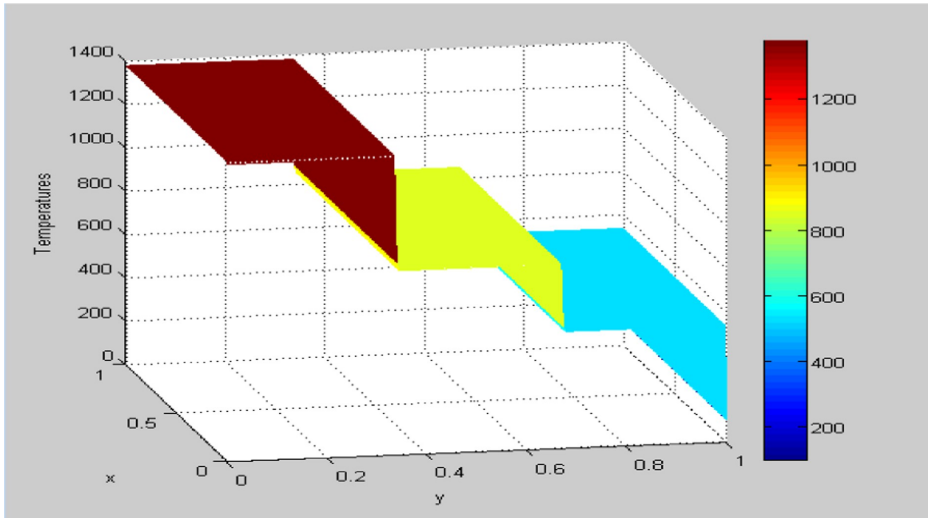


Fig. 10. Evolution of temperature with variable coefficients.

also the associated right hand side of the algebraic system, send and receive buffers containing exchanged values are stored. Grid'5000 provides 32 bits and 64 bits machines. We have chosen 64 bits machines which have the capability to use RAM with size greater than 4 Gigabytes. Indeed as one bit in CPU register reference a byte in memory, 32 bits machines can use 2^{32} register values and 64 bits machines 2^{64} register values (in fact 64 bits machine is 2^{32} larger than 32 bits machine). So 32 bit machines can address a maximum of 4 Gigabytes of RAM and 64 bit machines can theoretically address 16 Exabytes of RAM. Moreover all cores are used in order to avoid disc swapping which, perhaps, could make decrease the performances of parallel algorithms. This decrease in performance in Grid'5000 is also aggraved by the fact that there is no direct link between Rennes and Nancy. So, for the solution of large scale discretized systems, only machines having large memory are used.

We present below the results for $n = 454$, 622, 766 and 1051 on one cluster and on a grid composed of two distant clusters when we use four and eight processors in synchronous and asynchronous mode.

For the stopping criterion with a value of threshold $\epsilon = 10^{-3}$ and with an overlap which is 3 elements thick, the results of the cluster simulations for synchronous and asynchronous modes are summarized in Tables 7–10 for the subdomain method without and with overlapping when the coefficients are constant, and in Tables 11–14 for the subdomain method without and with overlapping when the coefficients are variable. The results of the grid simulations for synchronous and asynchronous modes are summarized in Tables 15–18 for the subdomain method without and with overlapping when the coefficients are constant, and in Tables 19–22 for the subdomain method without and with overlapping when the coefficients are variable. On these tables are indicated the values of n, the number of machines used, the simulation elapsed times, for all time steps and for the three domains Ω_l, $l = 1, 2, 3$, the accumulated average relaxation number by processor to reach convergence, the communication times during the calculations and the synchronization times due to the synchronization barriers, the percentage of the sum of the times necessary to communications and synchronisations and finally for the grid, the values of τ which is the ratio of the synchronous and asynchronous computation time and which allows to measure the efficiency of the asynchronous methods compared to the synchronous ones. For the grid simulations, these tables are completed by Figs. 11–22 representing the elapsed times, the percentage of communications and synchronizations on grid as well as the value of τ as a function of n.

On clusters

5.2.1. Parallel simulation without overlapping for the problem with constants coefficients on cluster.

Table 7
Elapsed time (sec), relaxations and communications on cluster (parasilo cluster) with synchronous parallel algorithms without overlapping and $\epsilon = 10^{-3}$.

Synchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	251.31	14 767 021 548	21.73	13.89	14.17
622	796.08	37 194 973 863	49.20	30.39	10.00
766	1 319.10	59 278 668 933	82.36	57.60	10.61
1051	3 411.53	146 982 570 362	187.42	129.71	9.30
Synchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	193.88	7 312 353 574	41.34	16.07	29.61
622	523.28	21 748 818 304	94.53	40.17	25.74
766	973.25	37 250 094 453	159.06	82.49	24.82
1051	2 460.60	80 705 408 598	365.20	157.76	21.25

Table 8
Elapsed time (sec), relaxations and communications on cluster (parasilo cluster) with asynchronous parallel algorithms without overlapping and $\epsilon = 10^{-3}$.

Asynchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	730.07	41 765 926 816	1.21	25.42	3.65
622	2 468.83	95 460 312 639	1.89	64.59	2.69
766	3 690.04	157 983 117 530	2.45	153.52	4.23
1051	10 555.37	484 882 141 547	4.29	320.40	3.08
Asynchronous results on 8 machines					
			Communication times		
n	Elapsed Time	Relaxations	Compute	Barrier	% Comms
454	478.49	25 845 178 762	1.20	34.62	7.49
622	1 284.29	52 967 005 140	1.88	99.87	7.92
766	2 499.15	95 242 392 077	2.62	160.81	6.54
1051	6 326.46	242 032 746 961	4.44	543.65	8.66

5.2.2. Parallel simulation with overlapping of 3 meshes for the problem with constants coefficients on cluster.

Table 9
Elapsed time (sec), relaxations and communications on cluster (parasilo cluster) with synchronous parallel algorithms with overlapping and $\epsilon = 10^{-3}$.

Synchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	268.40	14 766 909 057	21.79	12.43	12.75
622	772.76	37 295 110 160	50.33	30.94	10.52
766	1 476.11	60 144 741 664	85.09	72.43	10.67
1051	3 615.83	147 442 415 395	188.13	165.54	9.78
Synchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	220.02	7 312 241 083	41.58	21.17	28.52
622	592.82	21 848 928 125	96.19	57.18	25.87
766	1 051.21	37 684 314 768	162.03	81.63	23.18
1051	2 510.00	80 922 125 125	366.49	197.79	22.24

Table 10
Elapsed time (sec), relaxations and communications on cluster (parasilo cluster) with asynchronous parallel algorithms with overlapping and $\epsilon = 10^{-3}$.

Asynchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	887.04	43 987 976 712	1.30	30.41	3.57
622	2 435.52	95 498 161 020	1.93	83.00	5.92
766	4 953.17	189 162 260 078	2.72	250.46	5.11
1051	11 637.81	495 993 116 809	4.68	449.55	3.90
Asynchronous results on 8 machines					
			Communication Times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	569.00	25 861 444 388	1.28	40.81	7.40
622	1 517.71	53 424 870 465	1.99	100.97	6.78
766	2 806.72	90 740 442 454	2.69	193.38	6.98
1051	7 034.31	258 501 403 878	4.92	406.31	5.85

5.2.3. Parallel simulation without overlapping for the problem with variable coefficients on cluster.

Table 11
Elapsed time (sec), relaxations and communications on cluster (parasilo clusters) with synchronous parallel algorithms, variable coefficients, without overlapping and $\epsilon = 10^{-3}$.

Synchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	548.35	12 449 607 741	18.56	11.19	5.43
622	1 530.28	32 776 125 189	40.04	24.21	4.20
766	2 816.04	49 013 373 900	64.06	33.07	3.45
1051	9 219.70	110 084 918 497	167.59	182.33	3.80
Synchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	452.92	6 173 189 940	34.90	18.26	11.74
622	1 236.83	17 441 738 776	76.54	35.27	9.04
766	2 229.31	32 095 019 404	122.99	59.46	8.18
1051	7 491.51	62 368 548 279	284.27	585.56	11.61

Table 12
Elapsed time (sec), relaxations and communications on cluster (parasilo cluster) with asynchronous parallel algorithms, variable coefficients, without overlapping and $\epsilon = 10^{-3}$.

Asynchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	518.88	12 453 523 945	0.78	6.80	1.46
622	1 458.14	32 776 125 189	1.32	18.20	1.34
766	2 707.23	49 013 373 900	1.89	24.14	0.96
1051	9 028.81	110 084 918 497	4.17	263.37	2.96
Asynchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	403.84	6 173 189 940	0.78	12.71	3.34
622	1 131.88	17 441 738 776	1.34	24.85	2.31
766	2 048.30	32 095 019 404	1.91	31.98	1.65
1051	6 984.60	62 368 548 279	4.85	482.51	6.98

5.2.4. Parallel simulation with overlapping of 3 meshes for the problem with variable coefficients on cluster.

Table 13
Elapsed time (sec), relaxations and communications on cluster (parasilo cluster) with synchronous parallel algorithms, variable coefficients, with overlapping and $\epsilon = 10^{-3}$.

Synchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	608.91	12 449 607 741	18.69	16.93	5.85
622	1658.07	32 817 134 913	40.52	30.59	4.29
766	3014.77	49 013 373 900	65.63	62.54	4.25
1051	9986.47	110 085 477 077	144.12	482.82	6.28
Synchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	511.13	6 173 189 940	34.92	27.21	12.16
622	1351.79	17 481 974 732	76.73	38.36	8.51
766	2396.51	32 095 019 404	123.54	67.62	7.98
1051	8056.07	62 369 106 859	281.91	658.97	11.68

Table 14
Elapsed time (sec), relaxations and communications on cluster (parasilo cluster) with asynchronous parallel algorithms, variable coefficients, with overlapping and $\epsilon = 10^{-3}$.

Asynchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	582.13	12 449 607 741	0.77	15.70	2.83
622	1594.67	32 817 134 913	1.32	27.27	1.79
766	2903.03	49 013 373 900	1.93	50.95	1.82
1051	9637.56	110 085 477 077	4.76	382.44	4.02
Asynchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	460.96	6 173 189 940	0.78	17.16	3.89
622	1256.23	17 481 974 732	1.33	29.30	2.44
766	2226.73	32 095 019 404	1.93	48.44	2.26
1051	7514.89	62 369 106 859	4.73	532.34	7.15

On grid

5.2.5. Parallel simulation without overlapping for the problem with constants coefficients on grid.

Table 15
Elapsed time (sec), relaxations and communications on grid (grisou and parasilo clusters) with synchronous parallel algorithms without overlapping and $\epsilon = 10^{-3}$.

Synchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	5 955.97	14 767 021 548	4 741.19	976.44	95.99
622	9 132.76	37 194 973 863	6 961.00	1 460.45	92.21
766	19 004.89	59 278 668 933	14 709.80	3 045.77	93.43
1051	41 144.79	146 982 570 362	31 250.89	6 540.48	91.85
Synchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	6 644.41	7 312 353 574	4 769.30	1 741.87	97.99
622	13 273.07	21 748 818 304	9 554.95	3 349.80	97.22
766	20 914.63	37 250 094 453	15 044.00	5 185.73	96.72
1051	75 953.86	80 705 408 598	58 307.05	15 680.12	97.41

Table 16
Elapsed time (sec), relaxations, communications, and τ on grid (grisou and parasilo clusters) with asynchronous parallel algorithms without overlapping and $\epsilon = 10^{-3}$.

Asynchronous results on 4 machines						
			Communication times			
n	Elapsed time	Relaxations	Compute	Barrier	% Comms	τ
454	1 525.64	52 970 504 781	154.86	404.46	36.66	3.90
622	2 470.52	79 629 301 202	192.46	406.64	24.25	3.70
766	2 973.57	116 046 621 407	68.20	294.66	12.21	6.39
1051	6 460.73	234 481 794 735	388.15	987.49	21.29	6.37
Asynchronous results on 8 machines						
			Communication times			
n	Elapsed time	Relaxations	Compute	Barrier	% Comms	τ
454	1 580.60	35 328 315 313	146.23	752.44	56.85	4.20
622	3 237.46	80 906 705 721	185.29	978.56	35.94	4.09
766	4 808.17	138 818 706 436	187.60	999.70	24.69	4.34
1051	6 287.62	237 240 794 600	115.82	628.88	11.84	12.08

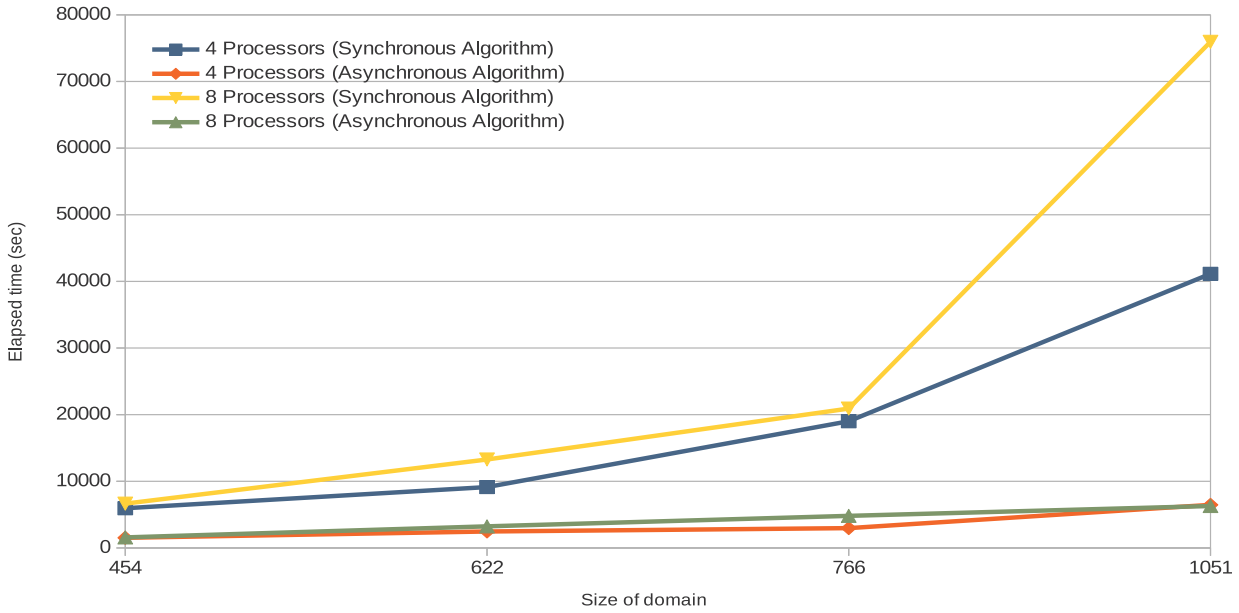


Fig. 11. Comparison between synchronous and asynchronous elapsed time on grid for the subdomain method with constant coefficients without overlapping.

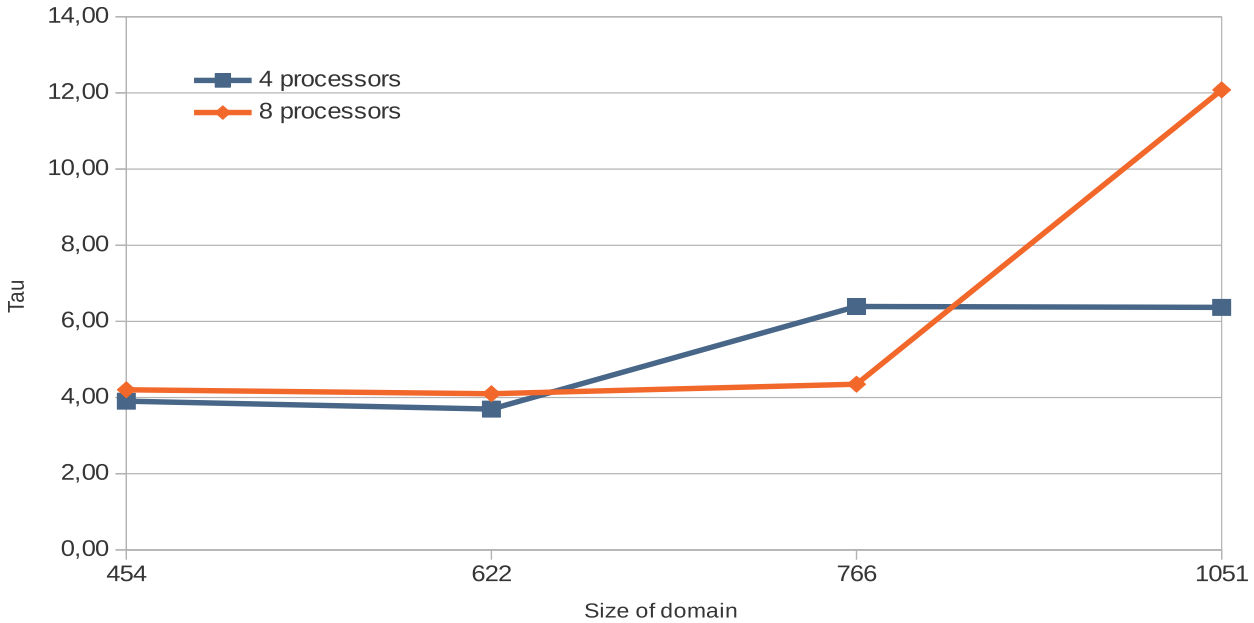


Fig. 12. Evolution of τ for the subdomain method with constant coefficients without overlapping.

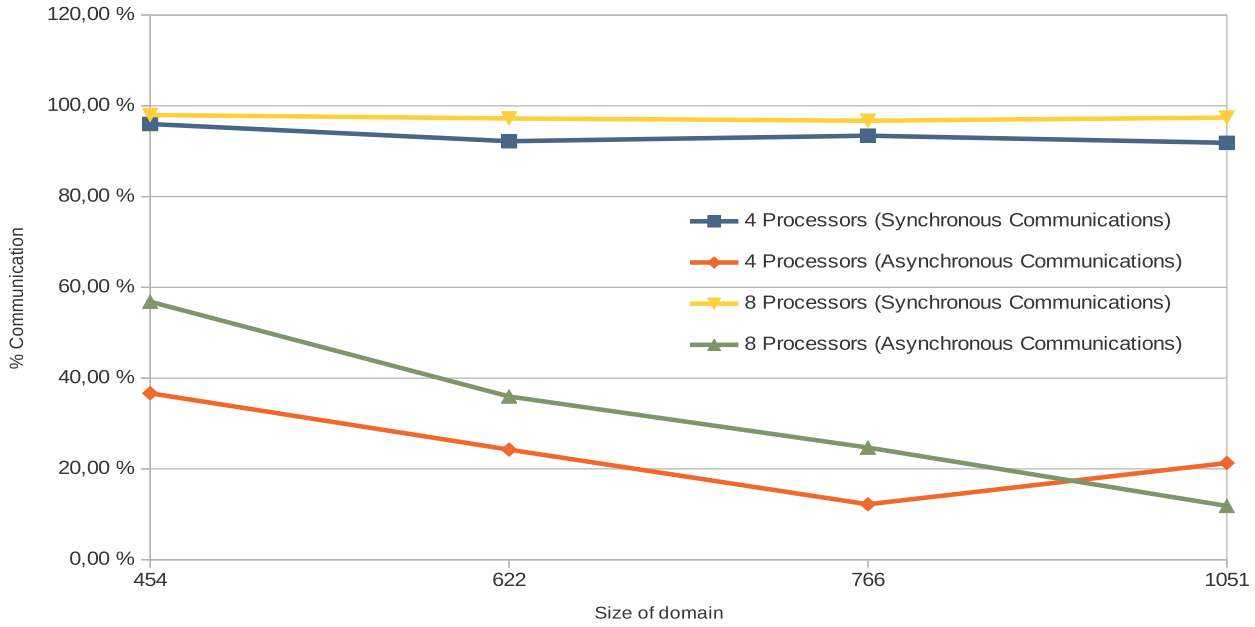


Fig. 13. Percentage of communications and synchronizations on grid for the subdomain method with constant coefficients without overlapping.

5.2.6. Parallel simulation with overlapping of 3 meshes for the problem with constants coefficients on grid.

Table 17
Elapsed time (sec), relaxations and communications on grid (grisou and parasilo clusters) with synchronous parallel algorithms with overlapping and $\epsilon = 10^{-3}$.

Synchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	3 039.57	14 766 909 057	2 390.31	418.19	92.40
622	6 327.52	37 295 110 160	4 853.87	799.43	89.34
766	19 463.53	60 144 741 664	14 988.62	3 122.70	93.05
1051	21 631.98	147 442 415 395	13 744.80	4 717.03	85.34
Synchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	6 662.29	7 312 241 083	4 767.37	1 740.10	97.67
622	13 391.76	21 848 928 125	9 592.78	3 363.38	96.74
766	21 426.57	37 684 314 768	15 331.79	5 285.58	96.22
1051	38 108.99	80 704 225 454	27 204.27	9 168.87	95.44

Table 18
Elapsed time (sec), relaxations, communications, and τ on grid (grisou and parasilo clusters) with asynchronous parallel algorithms with overlapping and $\epsilon = 10^{-3}$.

Asynchronous results on 4 machines						
		Z	Communication times			
n	Elapsed time	Relaxations	Compute	Barrier	% Comms	τ
454	1 540.36	52 368 821 965	115.89	354.30	30.52	1.97
622	2 540.62	76 663 195 956	185.72	541.93	28.64	2.49
766	2 180.89	69 907 713 376	219.24	422.44	29.42	8.92
1051	7 951.71	339 619 187 499	85.81	387.34	5.95	2.72
Asynchronous results on 8 machines						
			Communication times			
n	Elapsed time	Relaxations	Compute	Barrier	% Comms	τ
454	1 713.20	35 711 264 519	149.15	751.52	52.57	3.88
622	3 985.08	88 959 003 614	207.54	1 048.40	31.51	3.36
766	4 864.81	126 913 337 728	170.62	892.19	21.84	4.40
1051	5 397.60	189 143 234 440	92.24	539.26	11.70	7.06

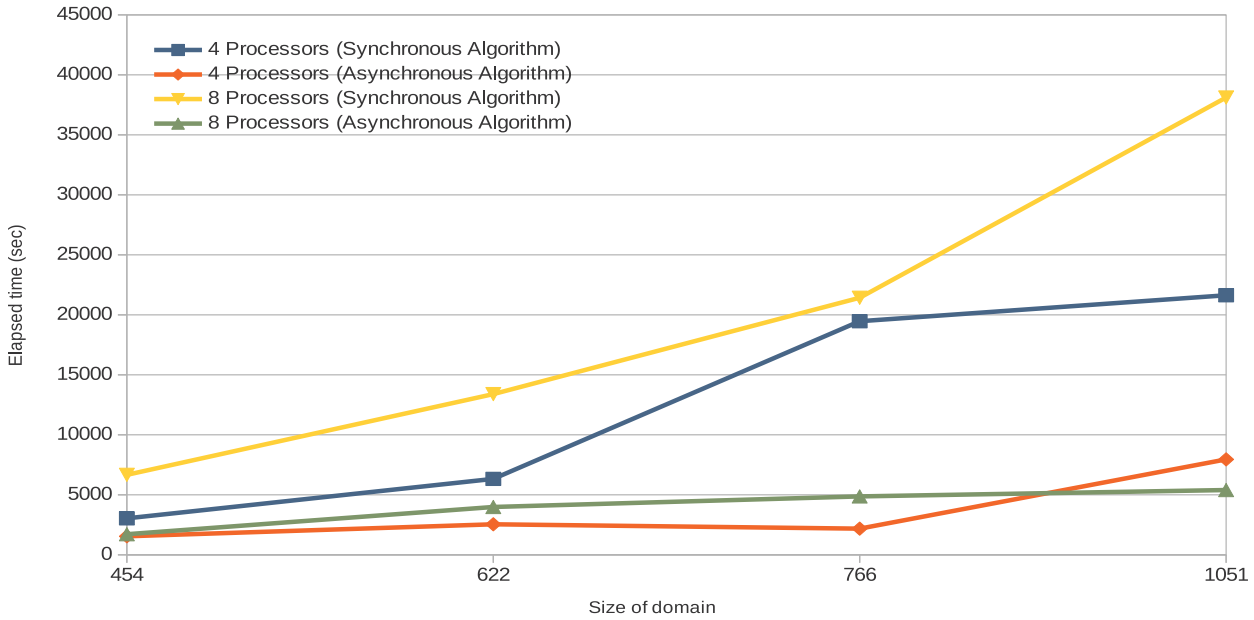


Fig. 14. Comparison between synchronous and asynchronous elapsed time on grid for the subdomain method with constant coefficients with overlapping.

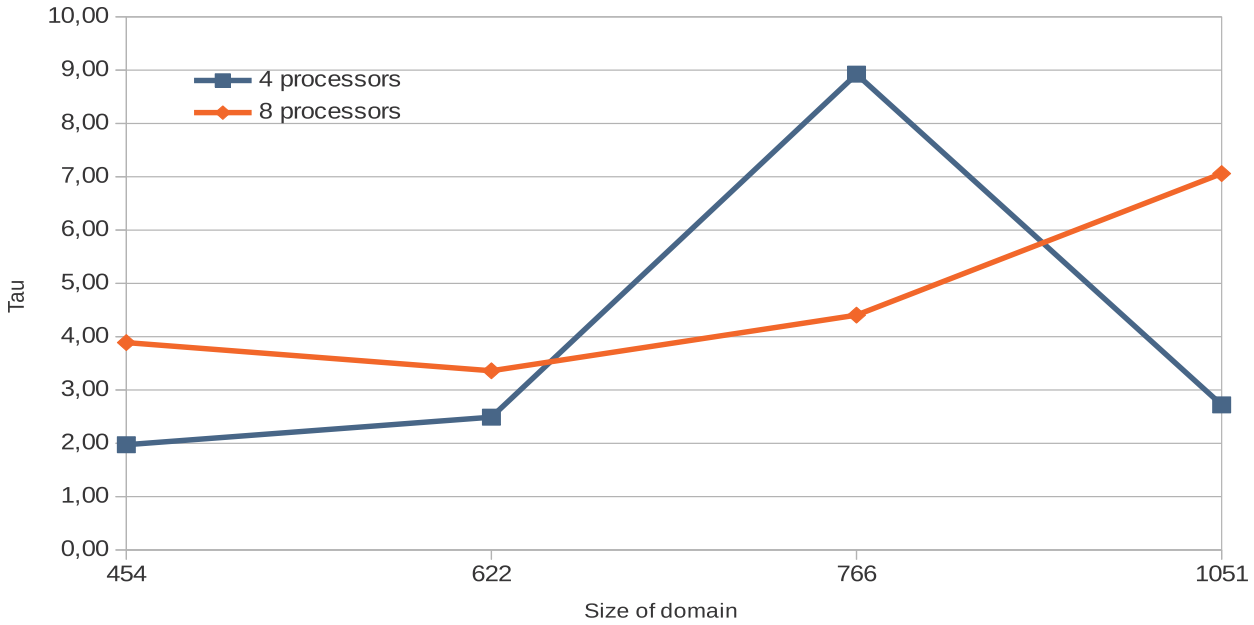


Fig. 15. Evolution of τ for the subdomain method with constant coefficients with overlapping.

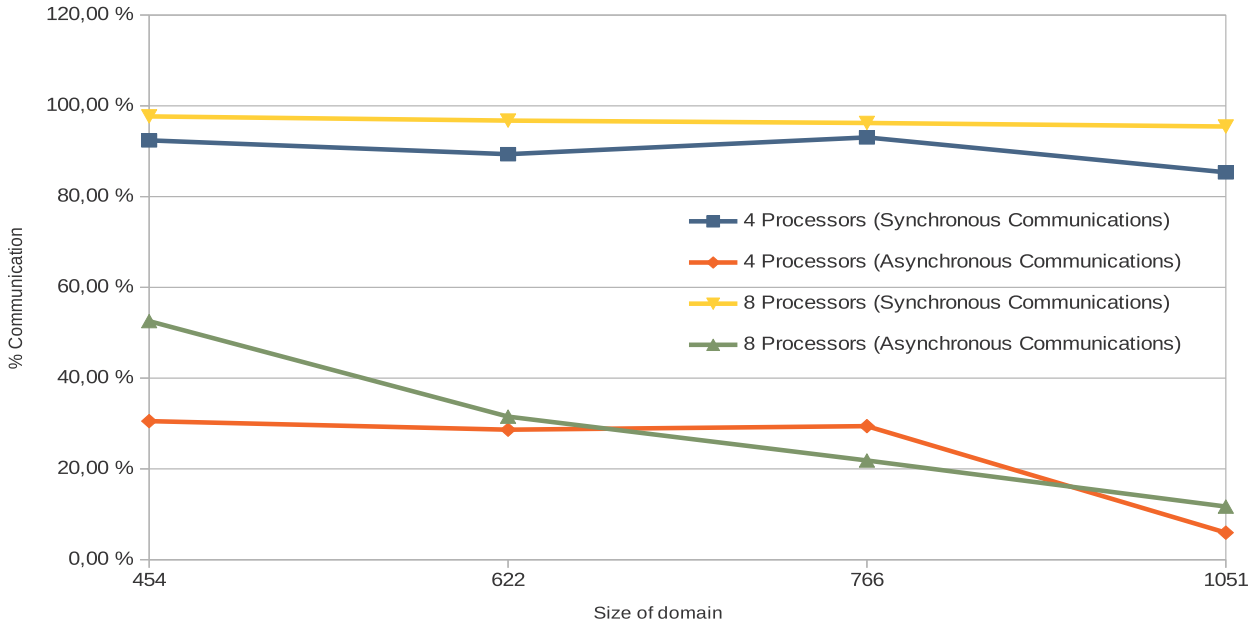


Fig. 16. Percentage of communications and synchronizations on grid for the subdomain method with constant coefficients with overlapping.

5.2.7. Parallel simulation without overlapping for the problem with variable coefficients on grid.

Table 19
Elapsed time (sec), relaxations and communications on grid (grisou and parasilo clusters) with synchronous parallel algorithms, variable coefficients, without overlapping and $\epsilon = 10^{-3}$.

Synchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	2 955.97	12 449 607 741	1 768.98	673.06	82.61
622	5 971.21	32 776 125 189	3 871.03	640.22	75.55
766	8 554.80	48 974 958 795	5 091.06	1 773.72	80.25
1051	23 294.51	110 084 918 497	9 931.96	3 702.60	58.53
Synchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	5 876.25	6 173 189 940	4 015.14	1 467.96	93.31
622	11 450.45	17 441 738 776	7 654.27	2 682.68	90.27
766	17 571.09	32 095 019 404	11 554.26	3 976.72	88.38
1051	39 179.38	62 368 548 279	23 190.04	8 219.18	80.16

Table 20
Elapsed time (sec), relaxations, communications, and τ on grid (grisou and parasilo clusters) with asynchronous parallel algorithms, variable coefficients, without overlapping and $\epsilon = 10^{-3}$.

Asynchronous results on 4 machines						
			Communication times			
n	Elapsed time	Relaxations	Compute	Barrier	% Comms	τ
454	605.29	12 449 607 741	22.96	73.78	15.98	4.88
622	1 546.09	32 776 125 189	25.43	73.86	6.42	3.86
766	2 815.97	49 013 373 900	26.51	93.12	4.25	3.04
1051	9 274.76	110 084 918 497	30.58	381.25	4.44	2.51
Asynchronous results on 8 machines						
			Communication times			
n	Elapsed time	Relaxations	Compute	Barrier	% Comms	τ
454	554.67	6 173 189 940	27.03	133.74	28.98	10.59
622	1 283.19	17 441 738 776	29.71	147.78	13.83	8.92
766	2 209.71	32 095 019 404	30.93	157.71	8.53	7.95
1051	7 195.77	62 368 548 279	36.08	667.37	9.77	5.44

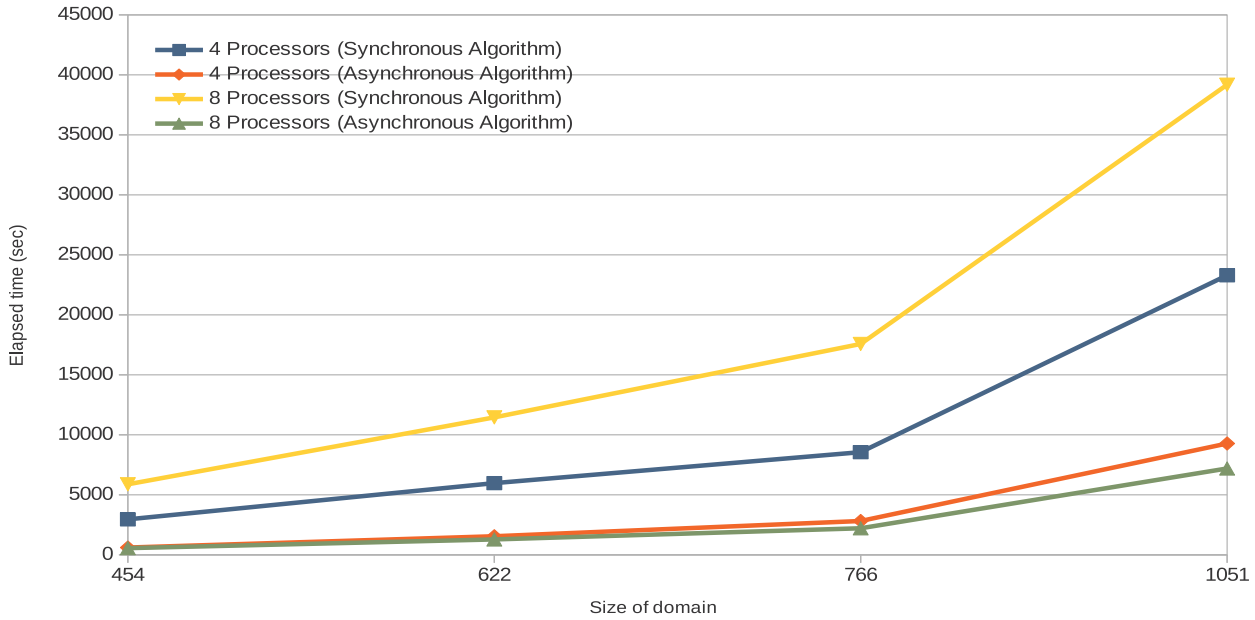


Fig. 17. Comparison between synchronous and asynchronous elapsed time on grid for the subdomain method with variable coefficients without overlapping.

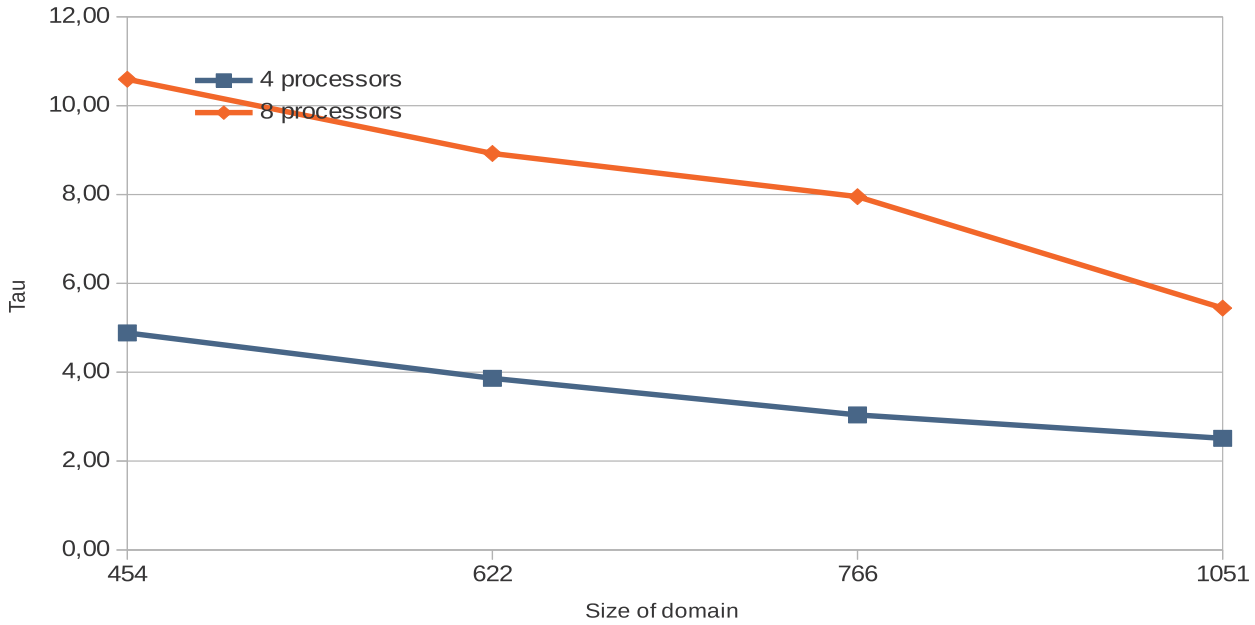


Fig. 18. Evolution of τ for the subdomain method with variable coefficients without overlapping.

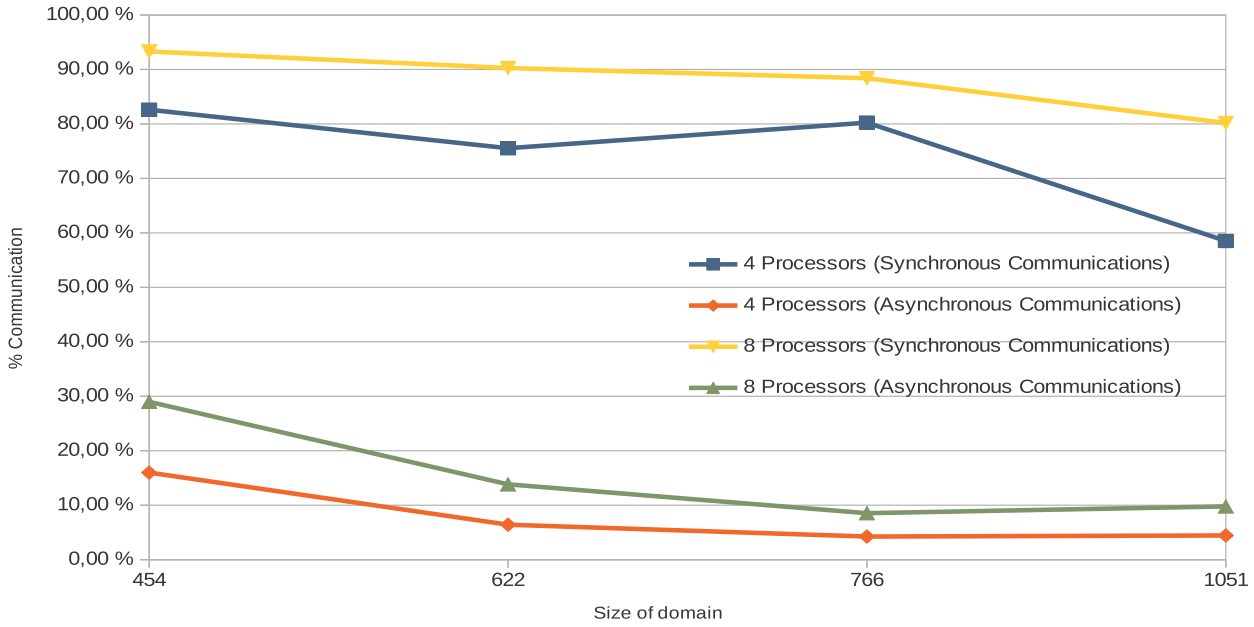


Fig. 19. Percentage of communications and synchronizations on grid for the subdomain method with variable coefficients without overlapping.

5.2.8. Parallel simulation with overlapping of 3 meshes for the problem with variable coefficients on grid.

Table 21
Elapsed time (sec), relaxations and communications on grid (grisou and parasilo clusters) with synchronous parallel algorithms, variable coefficients, with overlapping and $\epsilon = 10^{-3}$.

Synchronous results on 4 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	1 836.47	12 449 607 741	1 165.55	113.22	69.63
622	4 090.21	32 817 134 913	2 377.13	162.62	62.09
766	9 597.78	49 013 373 900	4 962.66	1 767.95	70.13
1051	23 706.00	110 085 477 077	9 917.64	3 715.67	57.51
Synchronous results on 8 machines					
			Communication times		
n	Elapsed time	Relaxations	Compute	Barrier	% Comms
454	5 937.25	6 173 189 940	4 019.59	1 476.03	92.56
622	11 597.31	17 481 974 732	7 668.46	2 699.80	89.40
766	15 402.60	32 095 019 404	9 825.47	3 381.33	85.74
1051	34 336.39	62 369 106 859	19 724.83	7 022.03	77.89

Table 22
Elapsed time (sec), relaxations, communications, and τ on grid (grisou and parasilo clusters) with asynchronous parallel algorithms, variable coefficients, with overlapping and $\epsilon = 10^{-3}$.

Asynchronous results on 4 machines						
			Communication times			
n	Elapsed time	Relaxations	Compute	Barrier	% Comms	τ
454	746.71	12 449 607 741	84.84	96.72	24.31	2.46
622	1 817.54	32 817 134 913	126.98	141.22	14.76	2.25
766	2 982.18	49 013 373 900	26.42	109.56	4.56	3.22
1051	9 624.07	110 085 477 077	30.59	438.84	4.88	2.46
Asynchronous results on 8 machines						
			Communication times			
n	Elapsed time	Relaxations	Compute	Barrier	% Comms	τ
454	610.22	6 173 189 940	27.02	134.11	26.40	9.73
622	1 409.37	17 481 974 732	29.77	146.20	12.48	8.22
766	2 384.75	32 095 019 404	30.94	158.93	7.96	6.46
1051	7 370.00	62 369 106 859	31.32	528.21	7.59	4.66

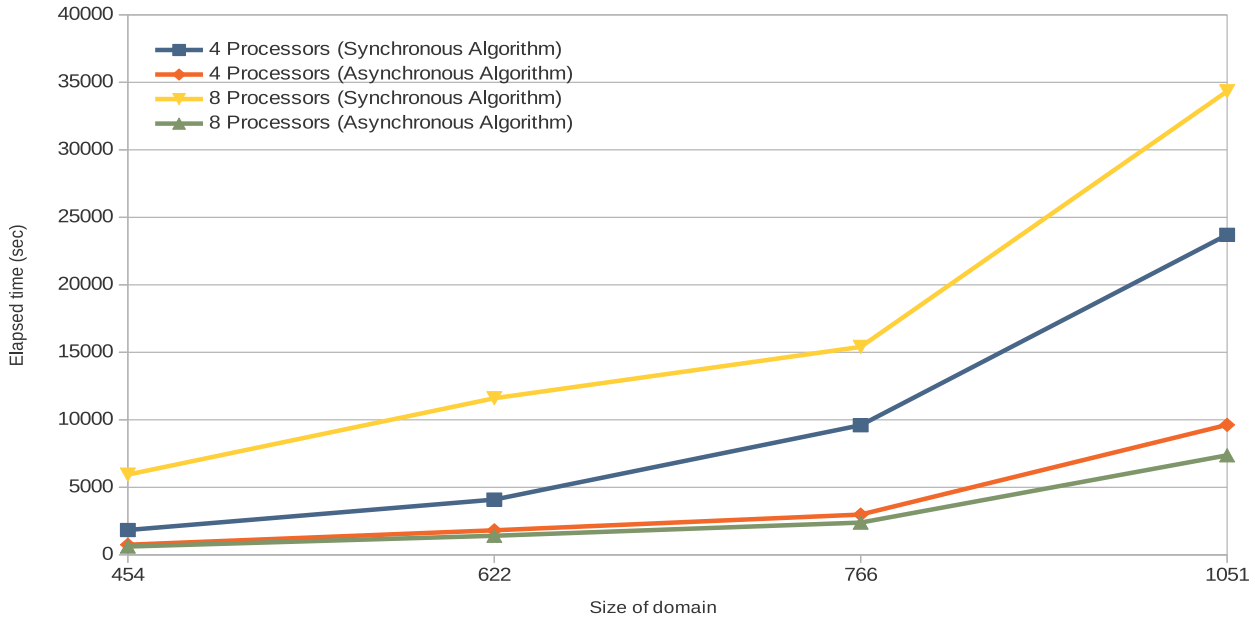


Fig. 20. Comparison between synchronous and asynchronous elapsed time on grid for the subdomain method with variable coefficients with overlapping.

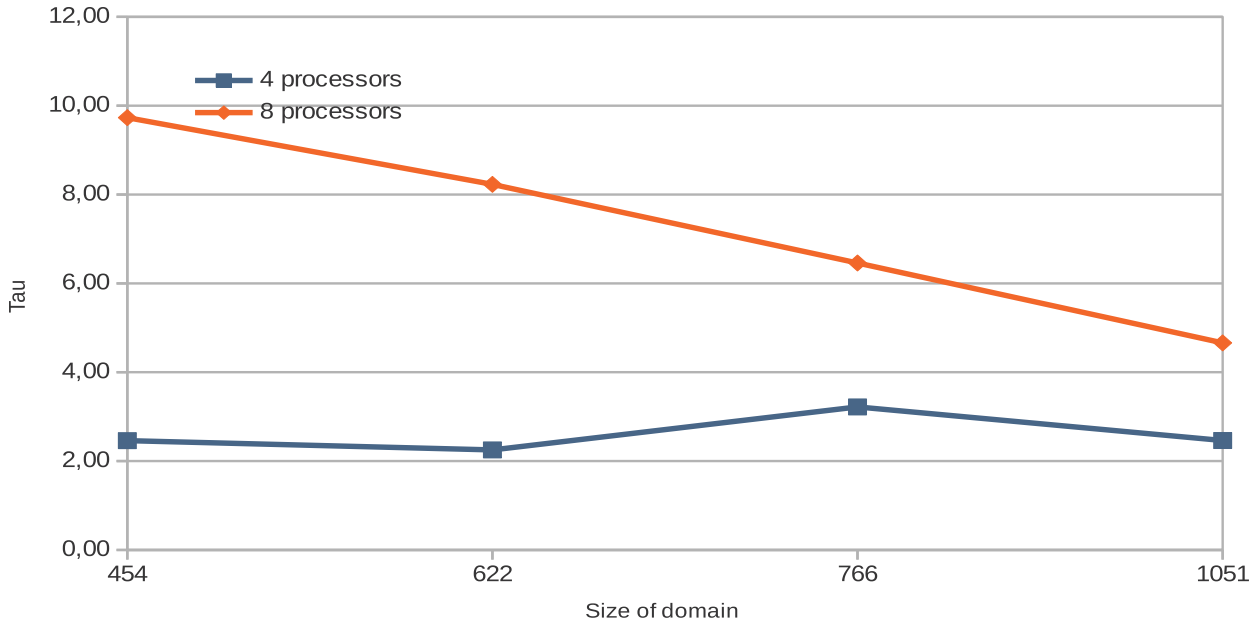


Fig. 21. Evolution of τ for the subdomain method with variable coefficients with overlapping.

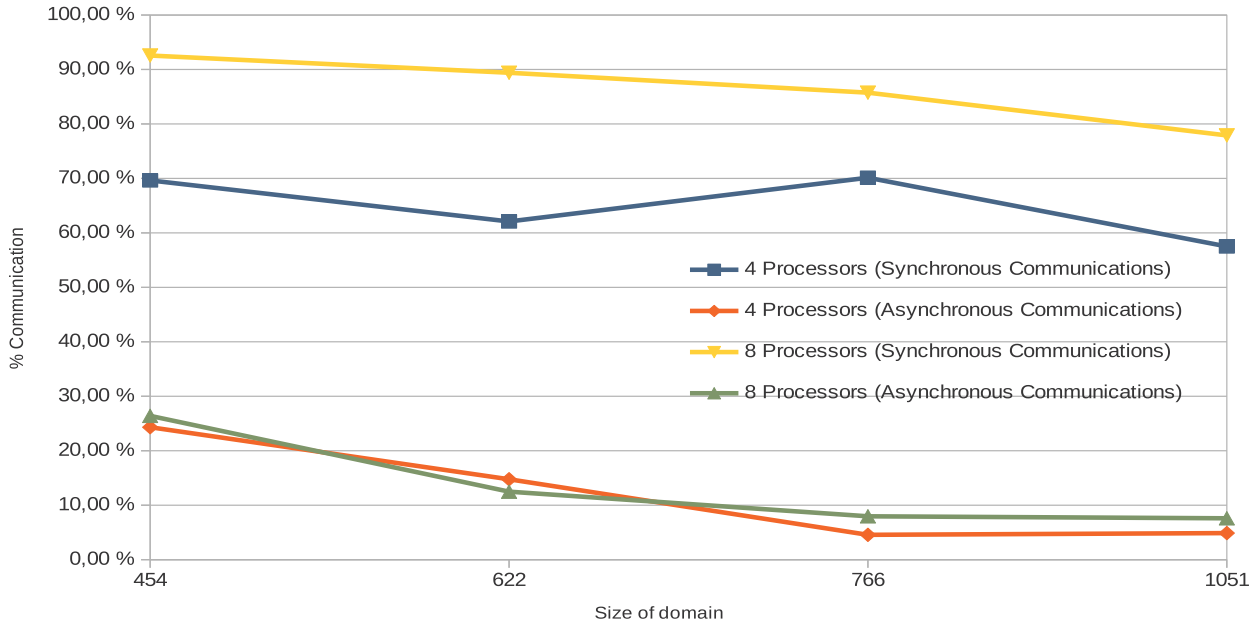


Fig. 22. Percentage of communications and synchronizations on grid for the subdomain method with variable coefficients with overlapping.

5.2.9. Parallel simulations synthesis.

For this kind of specific application, the analysis of the performances of the synchronous and asynchronous parallel methods is not simple to perform because the influence of the parameters describing the behavior of the iterative methods must be considered in relation to the computer aspects; it can be noted in the specific present case that, due on one hand to the high speed of convergence of the iterative methods and on the other hand taking into account the architecture of the machine, particularly the speed of the interconnection network between the machines, the behavior of parallel synchronous and asynchronous algorithms are not the same than the ones obtained in our previous works (see for example, but not only [5,21]). However, in this paragraph we have attempted to draw a number of conclusions.

On clusters

In the case of cluster when the coefficients are constant, synchronous results has better elapsed time than asynchronous results due to the numeric behavior (convergence speed); nevertheless, in the case of variable coefficient, asynchronous elapsed time are slightly better than synchronous ones. Regarding the number of relaxations we can notice that, for all problems with overlapping and non overlapping subdomain when the coefficients are constant, asynchronous mode works about three times slower than synchronous mode. On the other hand, in the case of variable coefficients with overlapping and non overlapping subdomain, the number of relaxations are slightly the same. These results seem difficult to explain since, in the situation where the coefficients are not constant, it is difficult to know the numerous values of the coefficients of the matrices.

The asynchronous parallel mode does not allow for good performance since on one hand the convergence rate is high due to a strong diagonal dominance of the matrices and on the other hand the interprocessor interconnection network is extremely fast at 10 GB Ethernet per second; however, in asynchronous mode, the communication times due to exchanges between processors are small and the time losses due to the synchronization barriers required at each time step and when passing from a sub-domain Ω_i to the following subdomain are time consuming.

In the synchronous mode, an inverse effect is obtained for these communication times: the communication times due to the exchanges between processors are important while the synchronization barriers are not very time consuming.

In spite of this notable gain, it can be seen that in the case of using clusters, the parallel asynchronous methods have performances that are generally lower than those obtained in synchronous mode. Thus, the combined effects of these two characteristics specific to the resolution of our model problem do not make the parallelization of asynchronous methods efficient since, given the few iterations, there is little synchronization and communication between the processors; thus, in contrast to what happens when the convergence is slow (see [5,21]), synchronous methods are more efficient on clusters in our case.

Nevertheless, compared to the times obtained in sequential mode, it is observed that the times obtained in parallel synchronous and asynchronous mode have decreased significantly. Then, the parallelization of the sequential method on cluster in both cases is interesting.

In terms of convergence speed, it can be seen that the necessary number of relaxations to converge in asynchronous mode is always greater than the ones obtained in synchronous mode; this decrease in the convergence rate comes from the fact that in asynchronous mode the parallel algorithm no longer has a behavior close to the Gauss–Seidel method implemented in sequential whereas in synchronous mode this behavior is preserved.

It is finally to note that given the speed of convergence it is not necessary to use much more processors for this application.

On grid

In the case of grid, we use two and four processors on each clusters in order to obtain good load balancing. In fact when only two processors are used, given the high speed of convergence, it is substantially a behavior similar to that observed on a single cluster. But, as will be seen below, beyond the use of two processors, the behavior of asynchronous methods changes completely; the speed of convergence has less influence and the asynchronous methods on a grid are of interest due to the use of a slow network and the few synchronizations.

Moreover, on a grid the notion of speed-up and efficiency, usually used to compare the performances of parallel algorithms to the

sequential ones, lose all their senses insofar as the machines are heterogeneous and, consequently, it is necessary to ask the question of knowing with respect to which machine the time of sequential computation must be chosen; it is therefore difficult to choose a particular machine and the parameter τ, the ratio of the synchronous and asynchronous computation time, will allow to measure the efficiency of the asynchronous methods compared to the synchronous ones.

Asynchronous method is generally better than synchronous method essentially due to the high latency of the network which imply costly communications. Moreover the results are interesting for large systems that have to be split into sub-tasks because they are not treatable on a single computer due to memory size limitation.

In Tables 15 and 16, parallel simulations without overlapping for the problem with constant coefficients on grid show that asynchronous mode is more efficient than synchronous mode. When four and eight processors are used, we show that the parallel asynchronous simulation times are clearly less than those obtained in synchronous mode and that the asynchronous mode elapsed time obtained in asynchronous mode is practically the same on four or eight processors. Note that the values of τ varie between 3.70 and 6.37 when four processors are used and between 4.09 and 12.08 on eight processors.

Similarly, in Tables 17 and 18, parallel simulations with overlapping for the problem with constant coefficients on grid also show that asynchronous mode is more efficient than synchronous mode and, when four and eight processors are used, that the parallel asynchronous simulation times are clearly less than those obtained in synchronous mode. Note that the values of τ varie between 1.97 and 8.92 when four processors are used and between 3.36 and 7.06 on eight processors.

In Tables 19 and 20, we remark the same phenomenon. Note that the values of τ varie between 2.51 and 4.88 when four processors are used and between 5.44 and 10.59 on eight processors and in Tables 21 and 22, the values of τ varie between 2.25 and 3.22 when four processors are used and between 4.66 and 9.73 on eight processors.

Moreover, in overlapping solutions, the elapsed time decreases due to less communications and more little restitution time.

This gain in performance is due to the weight of the synchronizations as can be seen by examining the communication times; indeed in asynchronous mode the wait times of the processors is negligible compared to those observed in synchronous mode (see Fig. 13 to Fig. 22). This is the main advantage of asynchronous algorithms when running a program on an architecture with many heterogeneous processors remote from each other and connected by a conventional slow network.

The number of relaxations to achieve convergence is related to the mode of updating the components. If, during the parallelization of the algorithm, the behavior of the sequential Gauss–Seidel algorithm is preserved, we will have a number of relaxations close to what was obtained in sequential mode. The algorithm will then have a multiplicative behavior. Otherwise it will have an additive behavior, close to the behavior of the Jacobi algorithm; in this case the number of relaxations will tend to double. This is linked to the large blocks decomposition of the global problem. However, given the imprecision of the stop test, a possible additional number of relaxations will improve the numerical quality of the calculated solution.

6. Conclusion.

The purpose of this paper was numerical simulation to compute the temperature involved in an industrial iron and steel problem. To carry out this work, we have developed both modeling studies, theoretical studies on the behavior of parallel synchronous and asynchronous algorithms studied in a unified context and simulations on a cluster and on a grid computing. At the end of this interesting study we can draw a number of remarks.

Indeed, several lessons emerge on parallel simulations. The first remark concerns the interaction between the behavior of the iterative numerical algorithm used with respect to the architecture of the machine. Given the dominant diagonal properties of the matrices, the relaxation method converges very quickly, i.e. in a few iterations. However, the parallel performances are not the same according to whether the calculations are performed on a cluster or on a grid computing containing remote and heterogeneous machines. This is essentially due to the interconnection network which is fast on a cluster and slow on a calculation grid. Under these conditions, and for this target application where the convergence is fast, the weight of the synchronizations plays an important role on the performances of the algorithm. It is therefore clear that, given this high speed of convergence, on cluster, synchronous methods are interesting to use given the few synchronizations and the fast network used. On the other hand, on a calculation grid, asynchronous methods show all their interest because of the heterogeneity of the distant machines used and especially the slowness of the network.

The second observation relates to the influence of the behavior of the sequential algorithm and the weight of this latter, the behavior of sequential algorithms is irreducible on the parallel method. In particular, the parameters allowing to measure the efficiency of the iterative numerical methods implemented, notably the speed of convergence, can guide the choice of synchronous or asynchronous communications during the parallelization of the algorithm.

The third remark concerns the choice stopping test. In parallel synchronous mode, the implementation of this stopping test does not pose a huge implementation problem. On the other hand, in parallel asynchronous mode, our experience led us to implement a more efficient centralized test.

The fourth remark concerns, on one hand, the splitting into parallel tasks, i.e. the granularity of the tasks and, on the other hand, the number of processors used, given the size of the problem and the performance of the algorithms. In this target application where the weight of the calculations is relatively small given the speed of convergence, it is preferable to have a high granularity and not a very large number of processors. This significantly reduces the overhead. Experimentally, one can determine the number of processors to use; indeed, when the elapsed time curve remains constant when the number of processors increases, additional processors is not necessary.

This application also needs to include additional synchronization barriers to respect the logic of the sequential processing and to guarantee the logic of the processing in parallel mode.

Moreover, asynchronous methods do not require rigorous load balancing.

Finally, in this specific case, this is the main advantage of these asynchronous methods, which will be relevant for very large applications. One can also think of their interest to a future investigation of exploitation in cloud computing.

Acknowledgment

This study has been made possible by support of Grid5000. Also Miss Ghania Khenniche gratefully acknowledges support provided by the Institute of National Polytechnic of Toulouse and the Institute of Research and computer science of Toulouse during its interships in France.

References

[1] Costes F. Modélisation thermomécanique tridimensionnelle par éléments finis de la coulée continue d'aciers. ENSM de Paris: Thèse de doctorat; 2004.
[2] Heinrich A. Modélisation bidimensionnelle de la coulée continue d'acier. Thèse de doctorat; 1979. Institut national polytechnique de Toulouse
[3] Henri M. Modélisation tridimensionnelle par éléments finis du refroidissement primaire lors de la coulée continue d'aciers. ENSM de Paris: Thèse de doctorat; 2009.
[4] Kandeil AY, Tag IA, Hassab MA. Solidification of steel billets in continuous casting. Eng J Qatar Uni 1991;4:103–20.
[5] Chau M, Garcia T, Spiteri P. Asynchronous schwarz methods applied to constrained

mechanical structures in grid environment. Adv Eng Softw Elsevier Sci 2014;74:1–15.
[6] Bahi JM, Miellou JC, Rhofir K. Asynchronous multisplitting methods for nonlinear fixed point problems. Numer Algorithms 1997;15:315–45.
[7] Couturier R, Denis C, Jézéquel F. GREMLINS: a large sparse linear solver for grid environment. Parallel Comput 2008;34(6–8):380–91.
[8] Couturier R, Ziane L. Khodja a scalable multisplitting algorithm to solve large sparse linear systems. J Supercomput 2014;69(1):200–24. https://doi.org/10.1007/s11227-014-1367-7.
[9] Bai ZZ, Migallón V, Penadés J, Szyld DB. Block and asynchronous two-stage methods for mildly nonlinear systems. Numer Math 1999;82(1):1–20.
[10] Jézéquel F, Couturier R, Denis C. Solving large sparse linear systems in a grid environment: the GREMLINS code versus the PETSc library. J Supercomput 2012;59(3):1517–32.
[11] Arnal J, Migallón V, Penadés J. Parallel newton two-stage multisplitting iterative methods for nonlinear systems. BIT Numer Math 2003;43:849–61.
[12] Bru R, Migallón V, Penadés J, Szyld DB. Parallel, synchronous and asynchronous two-stage multisplitting methods. Electron Trans Numer Anal 1995;3:24–38.
[13] Frommer A. Parallel nonlinear multisplitting methods. Numer Math 1989;56:269–82.
[14] Jones MT, Szyld DB. Two-stage multisplitting methods with overlapping blocks. Numer Linear Algebra Appl 1996;3:113–24.
[15] Frommer A, Szyld DB. H -Splittings and two-stage iterative methods. Numer Math 1992;63:345–56.
[16] Arnal J, Migallón V, Penadés J. Non-stationary parallel multisplitting algorithms for almost linear systems. Numer Linear Algebra Appl 1999;6:79–92.
[17] White RE. Parallel algorithms for nonlinear problems. SIAM J Alg Discrete Meth 1986;7:137–49.
[18] Duvaut G, Lions JL. Les inéquations en mécanique et Physique. Dunod; 1972.
[19] Glowinski R, Lions JL, Tremolieres R. Analyse numérique des Inéquations variationnelles. Dunod, Tome 1 and 2; 1976.
[20] Barbu V. Nonlinear semigroups and differential equations in Banach spaces. Noordhoff International Publishing; 1976.
[21] Chau M, Garcia T, Spiteri P. Asynchronous grid computing for the simulation of the 3D electrophoresis coupled problem. Adv Eng Software 2013;60–61:111–21.
[22] Ortega J, Rheinboldt W. Iterative solution of nonlinear equations in several variables. New York: Academic Press; 1970.
[23] El Tarazi M. Some convergence results for asynchronous algorithms. Numer Math 1982;39:325–40.
[24] Bahi JM, Contassot-Vivier S, Couturier R. Parallel iterative algorithms: from sequential to grid computing. Chapman & HallCRC; 2007.
[25] Miellou JC, Spitéri P, Baz DE. A new stopping criterion for linear perturbed asynchronous iterations. J Comput Appl Math Elsevier 2008;219(2):471–83.
[26] Miellou JC, Spitéri P. Stopping criteria for parallel asynchronous iterations for fixed point methods. In: Topping BHV, Ivanyi P, editors. Developments in parallel, distributed, grid and cloud computing for engineering. Saxe-Coburg Publications; 2013. p. 277–308.
[27] Spitéri P. Finite precision computation for linear fixed point methods of parallel asynchronous iterations. In: Topping BHV, Ivanyi P, editors. Techniques for parallel, distributed and cloud computing in engineering. Saxe-Coburg Publications; 2015. p. 163–96.
[28] Bolze R, Cappello F, Caron E, Daydé M, Desprez F, Jeannot E, Jégou Y, Lanteri S, Leduc J, Melab N, Mornet G, Namyst R, Primet P, Quetier B, Richard O, Talbi EG, Touche I. Grid5000: a large scale and highly reconfigurable experimental grid testbed. Int J High Perform Comput Appl 2006;20(4):481–94.
[29] Khenniche G, Garcia T, Spiteri P. Parallel simulation of steel solidification. In: Iványi P, Topping B, Várady G, editors. Proceedings of the fifth international conference on parallel, distributed, grid and cloud computing for engineering CivilComp. Press; 2017. https://doi.org/10.4203/ccp.111.22.
[30] Miellou JC. Algorithmes de relaxation chaotique à retards. ESAIM, Math Model Numer Anal 1975;9:55–82.

5.6 Article 6

Numer Algor (2017) 75:879–908
DOI 10.1007/s11075-016-0224-6

ORIGINAL PAPER

Grid solution of problem with unilateral constraints

M. Chau[1] · A. Laouar[2] · T. Garcia[3,4] · P. Spiteri[3]

Received: 4 March 2014 / Accepted: 12 October 2016 / Published online: 2 November 2016

Abstract The present study deals with the solution of a problem, defined in a three-dimensional domain, arising in fluid mechanics. Such problem is modelled with unilateral constraints on the boundary. Then, the problem to solve consists in minimizing a functional in a closed convex set. The characterization of the solution leads to solve a time-dependent variational inequality. An implicit scheme is used for the discretization of the time-dependent part of the operator and so we have to solve a sequence of stationary elliptic problems. For the solution of each stationary problem, an equivalent form of a minimization problem is formulated as the solution of a multivalued equation, obtained by the perturbation of the previous stationary elliptic operator by a diagonal monotone maximal multivalued operator. The spatial discretization of such problem by appropriate scheme leads to the solution of large

✉ P. Spiteri
Pierre.Spiteri@enseeiht.fr

M. Chau
ming.chau@irt-systemx.fr

A. Laouar
laouar.abdelhamid@univ-annaba.org

T. Garcia
thierry.garcia@uvsq.fr

[1] IRT SystemX, 8 avenue de la Vauve, 91 120 Palaiseau, France

[2] Faculté des Sciences, Département de Mathématiques, Université d'Annaba, Laboratoire LANOS, BP 12, 23 000, Annaba, Algérie

[3] IRIT - ENSEEIHT, UMR CNRS 5505, 2 rue Camichel - BP 7122, 31 071 Toulouse, France

[4] Present address: UVSQ-PRISM, 45 avenue des états-unis, 78035, Versailles Cedex, France

scale algebraic systems. According to the size of these systems, parallel iterative asynchronous and synchronous methods are carried out on distributed architectures; in the present study, methods without and with overlapping like Schwarz alternating methods are considered. The convergence of the parallel iterative algorithms is analysed by contraction approaches. Finally, the parallel experiments are presented.

Keywords Variational inequality · Parallel iterative algorithms · Asynchronous iterations · Unilateral constraints problem · Grid computing · Fluid mechanics

Mathematics Subject Classification (2010) 68U10 · 65F10 · 65Y05 · 65N22 · 65C20 · 65Y20 · 65M12

1 Introduction

In this study, we present a parallel numerical solution for solving a problem with unilateral constraints; this problem corresponds to a strongly nonlinear problem, the nonlinear nature belonging to the constraints subject to the solution of the model problem on one part of the boundary of the domain Ω in which the problem is defined; so for the solution of the model problem, the goal of the paper is to test the efficiency of parallel asynchronous methods compared to the synchronous ones. Such problems arise in mechanics, more particularly in elasticity and in fluid mechanics, for example, in problems of hydrodynamics arising in semipermeable environment (see [18, 24, 46]). In a general formulation, the problem is an evolution problem; by using an implicit time marching scheme, the problem is finally reduced to the solution of a sequence of stationary problems. Generally, each stationary problem is formulated thanks to an elliptic operator equipped with mixed boundary conditions. Classically, each stationary problem can be formulated as an equivalent form by a minimization problem on a closed convex set. Then, using the Euler inequality, the minimization problem is characterized by a stationary variational inequality. The solution of each stationary variational inequality needs, at each time step, the projection on a convex set of the solution of the corresponding unconstrained stationary problem.

In convex optimization, it is well known that the minimization of a functional J on a convex subset U_{ad} included in a vectorial space E (i.e. $U_{ad} \subset E$) is equivalent to the solution of a multivalued equation (see [6]); more precisely, consider the following constrained optimization problem

$$\begin{cases} \text{Find } u \in U_{ad} \text{ such that} \\ J(u) \leq J(v), \forall v \in U_{ad}. \end{cases} \tag{1}$$

Then, an equivalent formulation of the optimization problem is to solve the following multivalued equation

$$0 \in \bar{B}u \tag{2}$$

where $\bar{B}$ is the subdifferential mapping of $(J + \phi_{U_{ad}})$, where $\phi_{U_{ad}}$ denotes the indicator function of the convex set U_{ad}. So classically, $\bar{B}$ is a monotone operator from E into the dual space of E, denoted in the sequel by E' (see [6]). It can be noted that

this formalism expresses the projection on the convex set U_{ad}. This approach is also very convenient and will be mainly applied to the analysis of the behaviour of iterative algorithms used for the numerical solution of the corresponding unconstrained optimization problem.

As previously said, for the discretization of the evolution part of the mathematical problem, an implicit time marching scheme is carried out; then we have to solve a sequence of stationary problems, where each one corresponds to the solution at each time step of a problem of the kind (1) or equivalently (2). Furthermore, well adapted schemes are used in order to achieve the spatial discretization of the remaining operators present in the mathematical model, i.e. the spatial part of the partial differential equation. Consequently, we have to solve large scale algebraic systems associated to each stationary problem. Taking into account the large size of the systems to solve, iterative numerical algorithms on the one hand and parallel numerical methods on the other hand are necessary to use, in order to reduce the elapsed time of computation. In parallel computation, synchronizations between the processors seem to be necessary; nevertheless, for the solution of large fixed point problems, if the operators arising in the problem to solve verify appropriate properties, parallel asynchronous iterative algorithms can be processed successfully. These algorithms have been studied by many authors (see for example [7, 15, 19, 20, 23, 34–36, 42]) for various applications. We recall that parallel asynchronous iterative schemes describe parallel computations carried out without order nor synchronization; in such algorithms, processors go at their own paces according to their intrinsic characteristics and computational load. Practically, it appears that asynchronous algorithms, compared to the synchronous ones, reduce idle times due to less synchronizations between the processors; this kind of algorithms is well adapted to the use of distributed computation on grid environments. Moreover, strict load balancing of the charge of the machines is not necessary. Finally, from a theoretical point of view, the mathematical model of iterative parallel synchronous methods corresponds to a particular case of the asynchronous ones.

More precisely we concentrate in the present study to the use of algorithms corresponding to the fact that the domain Ω is split in several parts. So, parallel iterative numerical methods will be used for the solution of the considered boundary value problem. Classically, one considers the methods without and with overlapping like the Schwarz alternating method. For such parallel synchronous or asynchronous iterative methods, the convergence is ensured when the spatial part of the operators arising in the mathematical problem are discretized by some classical appropriate schemes.

Then, by using such appropriate discretization and thanks to the formulation (2) of the problem to solve, it can be shown that at each time step, we have to invert M-matrices (see [44]) perturbed by a multivalued diagonal increasing (or monotone) operator. Then, the convergence of parallel asynchronous and synchronous iterative fixed point methods applied to the considered problem can be shown by contraction techniques (see [23, 36]) in a general theoretical framework well adapted to distributed computation.

More precisely, for nonlinear multivalued problems, the convergence of the considered parallel iterations, in the framework of a block decomposition, is obtained

by using the concept of regular splitting of an M-matrix (see [38]). Indeed, we show in Subsection 3.4 of the current paper that the associated block Jacobi's matrix associated to a block decomposition is a contraction matrix. As a consequence, if an M-matrix is involved, the fixed point mapping corresponding to the point decomposition is contracting and moreover the contraction property still holds for any block decomposition. It is worth noting that the result presented using only the notion of regular splitting in the forthcoming Proposition 2 of Subsection 3.4 is a very particular case of the main result of [36]. This result is presented in the present paper in a simplified framework compared to [36], without the help of M-minorant or H-minorant notions (see [16, 17, 21, 22, 32, 33, 39–41]). Moreover, the use of the Schwarz alternating method reduces in fact to solve an augmented system constituted also by an M-matrix perturbed by a multivalued diagonal monotone operator; so the previous analysis can again be achieved similarly to the convergence analysis of parallel synchronous and asynchronous Schwarz method applied to the solution of the augmented problem (see [35, 42]).

From an experimental point of view, the implementation of the considered algorithms is carried out on a distributed memory multiprocessor. Communications are managed with MPI facilities[1]. More specifically, computational experiments are performed on GRID 5000 (see [9]), the French national grid. Asynchronous and synchronous versions of the parallel algorithms are compared when distant and heterogeneous clusters are used; their efficiency is analysed.

The problem with unilateral constraints has been studied by several authors; so we refer for example to [29], [46] and to the book of G. Duvaut and J.L. Lions [18]. In [29], J.L. Lions states a general result of existence and uniqueness of the solution for the stationary and the evolutive problems with unilateral constraints; these statements are established in both cases when the operator arising in the problem is on one hand linear and on the other hand nonlinear and monotone. In [18], we can find an unified approach concerning the problem with unilateral constraints and the classical obstacle problem; the main difference between the two distinct problems resides in the choice of a particular functional defined by an integral either on the boundary for the problem with unilateral constraints either on the domain for the obstacle problem. We refer to this last very useful reference [18] for a basis of the present paper, particularly for the multivalued formulation of the studied problem.

The obstacle problem has been studied in many contributions and by several authors; it appears in many applications such as mechanics, free boundary value problem and finance. Classically, this problem can be formulated by equivalent ways such as variational inequality, complementary problem or multivalued problem. After appropriate discretizations, these formulations lead to the solution of nonlinear algebraic systems. For example, in [31], sequential methods are considered for the Hamilton-Jacobi-Bellman problem, this latter problem being related to the obstacle problem formulated thanks to a complementary problem. In [2], the linear

[1]Message passing interface.

convergence is proved for a sequential multiplicative Schwarz method, applied to the solution of variational inequalities and in [3] a geometric convergence rate is established for sequential additive Schwarz method for the same problem. In [26], the block relaxation methods for obstacle problems with M-matrices are considered while in [27] multiplicative and additive Schwarz methods for obstacle problems with convection-diffusion operators are considered on two subdomains; on the first subdomain, the solution equals to a given obstacle function, and on the second one, the solution satisfies a linear equation. In [28], the sequential convergence analysis of the generalized Schwarz method for solving the obstacle problem with T-monotone operator is studied. In the previous papers, note that, mainly, the study of numerical solutions is considered by using sequential subdomain methods. Furthermore, in the previous studies, when the parallel solution is considered for the solution of the obstacle problem, it can be noticed that only parallel synchronous methods are studied. Nevertheless, some works studying the behaviour of parallel asynchronous subdomain methods have also been considered. In [4, 5], Z.Z. Bai and D.J. Evans studied the behaviour of iterative parallel asynchronous multisplitting methods for solving linear complementary problems in the case where the linear part of the problem is constituted by an M-matrix.

Besides, other studies related to the implementation of parallel asynchronous iterations for the solution of the obstacle problem have been developed; in [12], the authors analyse the behaviour of the parallel asynchronous Richardson method. In [13], the parallel asynchronous projected block relaxation method without overlapping is analysed. In [14, 43], the authors consider parallel asynchronous Schwarz alternating method for the obstacle problem in a complementary formulation; in [14], the classical formulation of this method is considered, while in [43] the parallel asynchronous methods with flexible communications is studied. We refer also to [42] where the parallel asynchronous multisplitting methods with flexible communications are presented and analysed for the solution of a nonlinear diffusion problem. In all previous studies, the parallel synchronous and asynchronous methods are implemented on CPU architectures whereas in [45] a comparison of performances of these previous parallel methods implemented on CPU and GPU architectures is presented.

The paper is organized as follows: in Section 2, the model problem and the related application of unilateral constraints in fluid mechanics are briefly presented, afterwards a characterization of the solution is given. The last subsection is devoted to recall classical mathematical background and a formulation of the model problem is given; this new formulation leads to the solution of a multivalued problem. In Section 3, the parallel synchronous and more generally asynchronous iterative algorithms are described; in particular, thanks to appropriate discretization schemes presented in Subsection 3.1, the convergence analysis of the parallel asynchronous and synchronous algorithms is studied, both for the parallel methods without and with overlapping and for every decomposition. Finally, in the Subsection 4.1, the implementation of the algorithms on GRID 5000 platform is briefly described and the experimental results on grid environment are presented and analysed in Subsection 4.2. Some concluding remarks summarize the paper in Section 5.

2 Model problem

2.1 Formulation of the problem

Consider a flow in a bounded domain $\Omega \subset \mathbb{R}^3$. Let us denote by Γ the boundary of Ω and assume that $\Gamma = \Gamma_0 \bigcup \Gamma_1$; in the physical application Γ_0 is assumed to be permeable and Γ_1 semipermeable, in the following way: the fluid (assumed to be less compressible) is free to enter in Ω through Γ_1. If $u(\mathbf{x}, t)$ denotes the pressure of the fluid at the point $\mathbf{x} = \{x_1, x_2, x_3\}$ and at time t, then $u = u(\mathbf{x}, t)$ is the solution of the following parabolic boundary value problem

$$\frac{\partial u(\mathbf{x}, t)}{\partial t} - \Delta u(\mathbf{x}, t) = f(\mathbf{x}, t), \text{ everywhere in } \Omega \text{ and for } 0 < t \leq T,$$

where $f \in \mathcal{L}^2(\Omega \times [0, T])$, T is the final time and Δ is the Laplacian operator defined in the three-dimensional space. The boundary condition associated to the previous parabolic equation can be formulated as follows [18] : if a known pressure $\varphi(\mathbf{x})$ is applied on the boundary $\Gamma = \Gamma_0 \bigcup \Gamma_1$, then we have

$$u(\mathbf{x}, t) = \varphi(\mathbf{x}), \forall \mathbf{x} \in \Gamma_0 \text{ and } \forall t > 0;$$

on Γ_1 we have two distinct cases:

– First case: if

$$\varphi(\mathbf{x}) < u(\mathbf{x}, t), \forall \mathbf{x} \in \Gamma_1 \text{ and } \forall t > 0,$$

the fluid is inclined to come out of Ω, which is impossible; so the output is zero and then

$$\frac{\partial u(\mathbf{x}, t)}{\partial n} = 0, \forall \mathbf{x} \in \Gamma_1 \text{ and } \forall t > 0,$$

where $\frac{\partial}{\partial n}$ is the outward normal derivative with respect to Γ_1; thus, the normal derivative is equal to zero if the constraint is not active.

– Second case: if

$$\varphi(\mathbf{x}) \geq u(\mathbf{x}, t), \forall \mathbf{x} \in \Gamma_1 \text{ and } \forall t > 0,$$

the fluid is inclined to come in Ω, which is possible since it is free to enter in Ω; then we have a continuous flow, and consequently, there is no jump of pressure. Thus, we have

$$\varphi(\mathbf{x}) = u(\mathbf{x}, t) \text{ and } \frac{\partial u(\mathbf{x}, t)}{\partial n} \geq 0, \forall \mathbf{x} \in \Gamma_1 \text{ and } \forall t > 0.$$

Let us denote by $u(\mathbf{x}, 0) = u_0(\mathbf{x})$ the initial pressure. Then, we can summarize the problem of hydrodynamic in semipermeable environment as follows

$$\begin{cases} \frac{\partial u(\mathbf{x},t)}{\partial t} - \Delta u(\mathbf{x}, t) = f(\mathbf{x}, t), \text{ everywhere in } \Omega \text{ and for } 0 < t \leq T, \\ u(\mathbf{x}, t) \geq \varphi(\mathbf{x}), \frac{\partial u(\mathbf{x},t)}{\partial n} \geq 0, (u(\mathbf{x}, t) - \varphi(\mathbf{x})) \frac{\partial u(\mathbf{x},t)}{\partial n} = 0, \forall \mathbf{x} \in \Gamma_1 \text{ and } \forall t > 0, \\ u(\mathbf{x}, t) = \varphi(\mathbf{x}), \forall \mathbf{x} \in \Gamma_0 \text{ and } \forall t > 0, \\ u(\mathbf{x}, 0) = u_0(\mathbf{x}), \forall \mathbf{x} \in \Omega. \end{cases} \tag{3}$$

Problem (3) is currently solved by an implicit time marching scheme. Consider now the associated stationary problem and let us denote by $v = v(\mathbf{x})$ the solution of this problem; then we have to solve the following problem

$$\begin{cases} -\Delta v(\mathbf{x}) + \sigma v(\mathbf{x}) = g(\mathbf{x}), \text{ everywhere in } \Omega, \sigma > 0, \\ v(\mathbf{x}) \geq \varphi(\mathbf{x}), \frac{\partial v(\mathbf{x})}{\partial n} \geq 0, (v(\mathbf{x}) - \varphi(\mathbf{x})) \frac{\partial v(\mathbf{x})}{\partial n} = 0, \forall \mathbf{x} \in \Gamma_1, \\ v(\mathbf{x}) = \varphi(\mathbf{x}), \forall \mathbf{x} \in \Gamma_0, \end{cases} \tag{4}$$

where $g \in \mathcal{L}^2(\Omega)$ is derived from the implicit time marching scheme and from the discretization of the right-hand side $f(\mathbf{x}, t)$ of (3) and σ is a positive parameter, in fact the inverse of the time step.

For the stationary problem, consider the following functional

$$J(v) = \frac{1}{2} \int_\Omega (\mid \nabla v \mid^2 + \sigma v^2) d\mathbf{x} - \int_\Omega g v d\mathbf{x}$$

and let us denote by $\|v\|_{0,\Omega} = (\int_\Omega v^2 d\mathbf{x})^{\frac{1}{2}}$ the classical norm of $\mathcal{L}^2(\Omega)$ and define the set

$$U_{ad} = \{v | v \in \mathcal{H}^1(\Omega), \text{ such that } v(\mathbf{x}) \geq \varphi(\mathbf{x}) \text{ on } \Gamma_1, v(\mathbf{x}) = \varphi(\mathbf{x}) \text{ on } \Gamma_0\},$$

where $\mathcal{H}^1(\Omega)$ is the classical Sobolev space, defined by

$$\mathcal{H}^1(\Omega) = \{v | v \in \mathcal{L}^2(\Omega), \frac{\partial v}{\partial x_i} \in \mathcal{L}^2(\Omega), i = 1, 2, 3\},$$

in which the partial derivatives are considered in the distributional sense; recall that classically $\mathcal{H}^1(\Omega)$ normed by

$$\|v\|^2_{1,\Omega} = \int_\Omega (\mid \nabla v \mid^2 + v^2) d\mathbf{x} = \|\nabla v\|^2_{0,\Omega} + \|v\|^2_{0,\Omega}$$

is an Hilbert space.

Then, we have to solve the following problem

$$\begin{cases} \text{Find } u \in U_{ad} \text{ such that} \\ J(u) \leq J(v), \forall v \in U_{ad}. \end{cases} \tag{5}$$

Let

$$a(u, v) = \int_\Omega (\nabla v \nabla u + \sigma\, uv) d\mathbf{x},$$

and

$$L(v) = < g, v > = \int_\Omega g v d\mathbf{x},$$

where $< .,. >$ denotes the inner product defined in $\mathcal{L}^2(\Omega)$. Then, we can write $J(u)$ as follows

$$J(u) = \frac{1}{2} a(u, u) - L(u),$$

and if the minimization problem has one solution satisfying (4), that will be verified in the next subsection, then the Euler inequality applied to problem (5) leads to the classical equivalent formulation

$$\begin{cases} \text{Find } u \in U_{ad} \text{ such that} \\ a(u, v - u) \geq L(v - u), \forall v \in U_{ad}. \end{cases} \tag{6}$$

Remark 1 The boundary value problem (3) models a problem of semipermeable wall in which the thickness of the wall is very small. When the thickness of the semipermeable wall is finite then, the following mathematical problem must be considered (see [18])

$$\begin{cases} \frac{\partial u(\mathbf{x},t)}{\partial t} - \Delta u(\mathbf{x},t) = f(\mathbf{x},t), \text{ everywhere in } \Omega, \text{ for } 0 < t \leq T, \\ \text{if } u(\mathbf{x},t) \geq \varphi(\mathbf{x}) \text{ then } \frac{\partial u(\mathbf{x},t)}{\partial n} = 0, \forall \mathbf{x} \in \Gamma_1, \forall t > 0, \\ \text{elsewhere } \frac{\partial u(\mathbf{x},t)}{\partial n} + k(u(\mathbf{x},t) - \varphi(\mathbf{x})) = 0, \forall \mathbf{x} \in \Gamma_1, \forall t > 0, k \in \mathbb{R}, k > 0, \\ u(\mathbf{x},t) = \varphi(\mathbf{x}), \forall \mathbf{x} \in \Gamma_0 \text{ and } \forall t > 0, \\ u(\mathbf{x},0) = u_0(\mathbf{x}), \forall \mathbf{x} \in \Omega. \end{cases}$$

The difference between the two models lies on the fact that in the previous boundary value problem, a Fourier (or Robin) boundary condition appears when $u(\mathbf{x},t) < \varphi(\mathbf{x})$; such condition expresses the fact that the flow through the wall is proportional to the difference of pressure $(u(\mathbf{x},t) - \varphi(\mathbf{x}))$. Note that, for such previous problem, the study considered hereafter in the following sections still holds.

2.2 Multivalued formulation of the problem

Consider the formulation (6) of the model problem. Note that obviously U_{ad} is a convex set. Moreover, classically, by using the classical Poincaré-Friedrich inequality, we can show by a direct way that U_{ad} is a closed convex set of the Sobolev space $\mathcal{H}^1(\Omega)$. Furthermore, since classically

- the space $\mathcal{H}^1(\Omega)$ normed by $\| \, . \, \|_{1,\Omega}$ is an Hilbert space and U_{ad} is a closed convex set
- the mapping $(u, v) \to a(u, v)$ is obviously a symmetric bilinear continuous and elliptic form
- the mapping $v \to L(v)$ is obviously a linear continuous form,

by applying the Stampacchia's theorem [24], the variational inequality (6) has a unique solution. In the case of the evolution problem (3), the reader is referred to the results of [29] and [30] for the existence and uniqueness of the solution

Furthermore, the notion of subdifferential mapping, recalled hereafter, will play a main role in the sequel; so we recall the main properties associated (see [6]).

Definition 1 Let E be a real Banach space with topological dual space E'. Given a proper[2] convex function χ on E and a point $z \in E$, we denote by $\partial\chi(z)$ the set of all $z' \in E'$ such that

$$\chi(y) \geq \chi(z) + < y - z, z' > \text{, for every } y \in E, \tag{7}$$

where $< .,. >$ denotes the pairing [3] between E and E'.

[2] That is, a proper function from E to $]-\infty, +\infty]$ is not identically equal to $+\infty$.

[3] That is, a bilinear form, from $E \times E'$ onto $\mathbb{R}$.

Such element z' is called subgradient of χ at z, and the set $\partial\chi(z)$ is called the subdifferential of χ at z.

Note that, if E is an Hilbert space, then E is identified with its own dual space and the pairing is the inner product of E. Moreover, if χ is Gateaux differentiable (or Frechet differentiable) at z, then $\partial\chi(z)$ consists in a single element, namely the Gateaux (or Frechet) differential of χ at z (see [6]). Lastly, $\partial\chi(z)$ is a closed convex set, possibly empty (see [6]).

Recall also that classically, $z_0 \in E$ is such that $\chi(z_0) = \min_{z\in E}(\chi(z))$ if and only if $0 \in \partial\chi(z_0)$. Moreover, the subdifferential $\partial\chi(z)$ is a monotone operator, in general multivalued, from E to E' (see [6]). In the sequel, we have also to consider the indicator function of the convex subset U_{ad} defined by

$$\phi_{U_{ad}}(z) = \begin{cases} 0 \text{ if } z \in U_{ad} \\ +\infty \text{ otherwise} \end{cases}$$

Clearly, $\phi_{U_{ad}}(z)$ is convex. By using the definition of the subdifferential mapping we have (see [6])

$$\partial\phi_{U_{ad}}(z) = \{z' \in E' | < z - y, z' > \geq 0, \text{for every } y \in U_{ad}\}.$$

This shows that $D(\partial\phi_{U_{ad}}) = D(U_{ad}) = U_{ad}$, where $D(\partial\phi_{U_{ad}})$ denotes the domain of the subdifferential mapping, and that $\partial\phi_{U_{ad}}(z) = \{0\}$ for each $z \in \text{int}(U_{ad})$, the interior of U_{ad}. Moreover, if z lies on the boundary of U_{ad}, then $\partial\phi_{U_{ad}}(z)$ coincides with the cone of the normal to U_{ad} at point z. Besides since

- the mapping $(u, v) \to a(u, v)$ is a bilinear continuous form
- the mapping $v \to L(v)$ is a linear continuous form,

then we can apply the Riesz representative theorem. So, there exists one and only one element denoted $\bar{A}u \in E'$ (E' being identified with the Hilbert space $E = \mathcal{H}^1(\Omega)$), where $\bar{A}$ is a continuous linear mapping from $E \to E'$, such that

$$a(u, v) =< \bar{A}u, v >_{E'\times E}, \forall v \in E = \mathcal{H}^1(\Omega);$$

similarly, there exists also one and only one element, denoted $\bar{g} \in E'$ such that

$$L(v) =< \bar{g}, v >_{E'\times E}, \forall v \in E = \mathcal{H}^1(\Omega);$$

then the stationary variational inequality (6) can be written as follows

$$\begin{cases} \text{Find } u \in U_{ad} \text{ such that} \\ < \bar{A}u - \bar{g}, v - u >_{E'\times E} \geq 0, \forall v \in U_{ad}. \end{cases} \tag{8}$$

In [18], the multivalued formulation of the problem with unilateral constraints is formulated and clarified; then, the model problem (4) can be written as follows

$$\begin{cases} -\Delta u(\mathbf{x}) + \sigma u(\mathbf{x}) = g(\mathbf{x}), \text{everywhere in } \Omega, \\ u(\mathbf{x}) = \varphi(\mathbf{x}), \forall \mathbf{x} \in \Gamma_0 \\ \frac{\partial u(\mathbf{x})}{\partial n} + \partial\phi_{U_{ad}}(u(\mathbf{x})) \ni 0, \forall \mathbf{x} \in \Gamma_1, \end{cases} \tag{9}$$

where $\partial\phi_{U_{ad}}(u)$ is the subdifferential of the indicator function of the convex subset U_{ad} of the admissible values, $U_{ad} \subset \mathcal{H}^1(\Omega)$.

So, in the sequel, we will mainly use this multivalued formulation (9) of the model problem in order to analyse the behaviour of the parallel iterative synchronous and asynchronous algorithms used for the solution of this problem. Note that this formulation includes both the inhomogeneous Dirichlet's condition in a part of the boundary and the inequality constraint on the other part.

3 Numerical solution of the model problem

3.1 Discretization

In the particular considered model problem, note that we have to solve, by implicit time marching scheme, the nonstationary problem (3). Thus, the problem consists in solving by a numerical way a sequence of stationary nonlinear problems like problem (4), the nonlinearity belonging to the fact that we have to project on the convex set U_{ad}.

In the sequel, we will consider that $\Omega \subset \mathbb{R}^3, \Omega = [0,1]^3$, and that Γ_1 is the square $[0,1]^2$ constituting by the lower face of the cube, i.e. the horizontal square corresponding to the value zero on the z-axis, the rest defining Γ_0. We consider also that Ω is discretized with an uniform mesh constituted by $M = (q+1)q^2, q \in \mathbb{N}$, discretization points, where q is related to the spatial discretization step by $h = \frac{1}{q+1}$, so that the complete discretization of the stationary boundary value problem (4) leads, at each time step, to the solution of a large multivalued nonlinear algebraic system. In order to write appropriately this algebraic system, let us denote by $\mathcal{I}_1 \subset \{1,\ldots,M\}$ the set of indices corresponding to the discretization points belonging to Γ_1 and $\mathcal{I}_0 \subset \{1,\ldots,M\}$ the set of indices of the other points of the mesh; thus, $\mathcal{I}_0 \bigcup \mathcal{I}_1 = \{1,\ldots,M\}$. Then, we can define a diagonal operator denoted by $\Psi(U)$ as follows

$$\Psi_i(u_i) \equiv \begin{cases} 0 & \text{if } i \in \{1,\ldots,M\} \setminus \mathcal{I}_1 \equiv \mathcal{I}_0, \\ (\partial\Phi_{U_{ad}})_i & \text{if } i \in \mathcal{I}_1, \end{cases}$$

where $U \in \mathbb{R}^M$ denotes the discretization vector on the mesh having M components denoted u_i; due to the fact that the subdifferential $\partial\phi_{U_{ad}}(u)$ of the indicator function $\phi_{U_{ad}}(u)$ of the convex subset U_{ad} is multivalued, then the discretization of this last operator is also multivalued and maximal monotone. Moreover, the components of $\Psi(U)$ belonging to $\{1,\ldots,M\} \setminus \mathcal{I}_1$ are zero; since the mapping identically zero is obviously monotone, then the operator $\Psi(U)$ is also multivalued diagonal maximal monotone. Using this operator $\Psi(U)$, the multivalued nonlinear algebraic system to solve is

$$(A + \sigma I)U - G + \Psi(U) \ni 0, \tag{10}$$

where $\sigma > 0$, $A \in \mathcal{L}(\mathbb{R}^M)$ is the spatial block discretization matrix obtained by using the classical seven points scheme of the finite difference method, I is the identity matrix, $G \in \mathbb{R}^M$ is derived from the time marching scheme and from the discretization of the right-hand side, and $U \in \mathbb{R}^M$. Furthermore, the matrix arising

in this system is an M-matrix (see [44]); so, in the sequel we consider the following assumption

$$A \text{ is an M-matrix.} \tag{11}$$

Thus, classically, if A is an M-matrix then the matrix $(A+\sigma I)$ is also an M-matrix. Note that, this previous property will play a main role for ensuring the convergence of the studied parallel asynchronous or synchronous algorithms, recalled and presented hereafter, for the numerical solution of the model problem. Indeed, taking into account on one hand this property of the matrix A and on the other hand, due to the monotony of the subdifferential of the indicator function of the convex set U_{ad}, we can prove by a theoretical way the convergence of such parallel methods presented below in Subsections 3.2 to 3.5. So we obtain a multivalued formulation of the problem to solve. Nevertheless, this fact corresponds to a theoretical point of view allowing only the analysis of the convergence of the parallel synchronous and asynchronous methods. In fact, in the implementation of the algorithm, for all indices $i \in \mathcal{I}_1$, we have to project an intermediate value of the iterate vector on the convex set. Then, from a practical point of view, concerning the implementation associated to the model problem, for all $i \in \mathcal{I}_1$, this process is described below:

- Firstly, compute an intermediate value of a component of the iterate vector by solving, for example, one equation of the algebraic linear system.
- And then project this intermediate value of this component on he convex set; more precisely, if the constraint is satisfied, the intermediate value of the component is not changed and if the constraint is saturated, then we project in the convex U_{ad}

Note that the block decomposition of the problem is derived from a decomposition of the domain Ω into slices; so, according to the chosen ordering, the iterate vector is decomposed into q^2 tridiagonal blocks of size $(q+1)$. In parallel computation, such decomposition simplifies and minimizes the communications between the processors.

3.2 Parallel asynchronous algorithms

Let α be a positive integer. Assume that $\mathcal{E} = \mathbb{R}^M$; note that $\mathcal{E}$ is an Hilbert space. Consider also that the space $\mathcal{E} = \prod_{i=1}^{\alpha} \mathcal{E}_i$ is a finite product of α subspaces denoted $\mathcal{E}_i = \mathbb{R}^{m_i}$, where $\sum_{i=1}^{\alpha} m_i = M$; note that $\mathcal{E}_i$ is also an Hilbert space where $\langle . \, , \, . \rangle_i$ denotes the scalar product and $|\, . \, |_{i,2}$ the associated Euclidean norm in $\mathcal{E}_i$, for all $i \in \{1, \ldots, \alpha\}$.

Then, for all $U, V \in \mathcal{E}$, if U_i and V_i are the block components of U and V, let us denote by $\langle U \, , \, V \rangle = \sum_{i=1}^{\alpha} \langle U_i \, , \, V_i \rangle_i$ the scalar product on $\mathcal{E}$ and $\| \, . \, \|_2$ its associated Euclidean norm.

In the sequel, we consider the general following fixed point problem

$$\begin{cases} \text{Find } U^* \in \mathcal{E} \text{ such that} \\ U^* = F(U^*) \end{cases} \tag{12}$$

where $V \mapsto F(V)$ applies from $\mathcal{E}$ to $\mathcal{E}$. Particularly in forthcoming Subsections 3.3, 3.4 and 3.5.2, we will show how the mapping F is related to the problem (10) to solve. According to the decomposition of $\mathcal{E}$, let us consider the corresponding block decomposition of F and V

$$\begin{aligned} V &= (V_1, \ldots, V_\alpha) \\ F(V) &= (F_1(V), \ldots, F_\alpha(V)). \end{aligned}$$

In order to solve the problem (12), let us consider now the parallel asynchronous iterations defined as follows: let $U^0 \in \mathcal{E}$ be given; then for all $p \in \mathbb{N}$, U^{p+1} is recursively defined by

$$U_i^{p+1} = \begin{cases} F_i(U_1^{\rho_1(p)}, \ldots, U_j^{\rho_j(p)}, \ldots, U_\alpha^{\rho_\alpha(p)}) \text{ if } i \in s(p) \\ U_i^p \text{ if } i \notin s(p) \end{cases} \tag{13}$$

where

$$\begin{cases} \forall p \in \mathbb{N}, s(p) \subset \{1, \ldots, \alpha\} \text{ and } s(p) \neq \emptyset \\ \forall i \in \{1, \ldots, \alpha\}, \{p \mid i \in s(p)\} \text{ is countable} \end{cases} \tag{14}$$

and $\forall j \in \{1, \ldots, \alpha\}$,

$$\begin{cases} \forall p \in \mathbb{N}, \rho_j(p) \in \mathbb{N}, 0 \leq \rho_j(p) \leq p \text{ and } \rho_j(p) = p \text{ if } j \in s(p) \\ \lim\limits_{p \to \infty} \rho_j(p) = +\infty. \end{cases} \tag{15}$$

The previous asynchronous iterative scheme models computations that are carried out in parallel without order nor synchronization; such algorithms describe a method without overlapping. Particularly, it permits one to consider distributed computations whereby processors go at their own pace according to their intrinsic characteristics and computational load. The parallelism between the processors is well described by the set $s(p)$ which contains the number of components relaxed by each processor on a parallel way at each relaxation while the use of delayed components in (13) permits one to model nondeterministic behaviour and does not imply inefficiency of the considered distributed scheme of computation. Note that, theoretically, each component of the vector must be relaxed an infinity of time. The choice of the relaxed components may be guided by any criterion, and in particular, a natural criterion is to pick-up the most recently available values of the components computed by the processors.

Remark 2 Such asynchronous iterations describe various classes of parallel algorithms, such as parallel synchronous iterations if $\forall j \in \{1, \ldots, \alpha\}$, $\forall p \in \mathbb{N}$, $\rho_j(p) = p$. In the synchronous context, for particular choice of $s(p)$, then (13)–(15) describe classical sequential algorithms; among them, the Jacobi method if $\forall p \in$

$\mathbb{N}$, $s(p) = \{1, \ldots, \alpha\}$ and the Gauss-Seidel method if $\forall p \in \mathbb{N}$, $s(p) = \{1+p \bmod \alpha\}$ (see[34]).

3.3 A property of contraction for the fixed point mapping associated to the point decomposition

Let us define in what follows, a fixed point mapping associated to the point decomposition of problem (10). So, we complete the previous formalism as follows; let us denote by capital letters the vectors of $\mathbb{R}^M$ and by capital letters with an index the block components of a vector, while lower letters with an index denote the point components; then g_i and u_i for $i = 1, \ldots, M$, are the components of the vectors G and U. So, for the point decomposition, we state the following result:

Proposition 1 *Consider the problem (10) and assume that the matrix A is an M-matrix (see assumption (11)) and Ψ is a monotone maximal operator. Then, we can associate to the problem (10) defined in the space $\mathcal{E}$ a contracting fixed point mapping denoted F_P, associated to the point decomposition, and there exists one and only one fixed point $U^\star$ also unique solution of the discretized problem (10).*

Proof For the considered point decomposition, we consider the M following scalar problems

$$(a_{i,i} + \sigma)u_i^\star - g_i + \sum_{\substack{j=1 \\ j\neq i}}^{M} a_{i,j}u_j^\star + \Psi_i(u_i^\star) \ni 0, \forall i \in \{1, \ldots, M\}, \tag{16}$$

or

$$(a_{i,i}+\sigma)u_i^\star - g_i + \sum_{\substack{j=1 \\ j\neq i}}^{M} a_{i,j}u_j^\star + w_i(u_i^\star) = 0,\ w_i(u_i^\star) \in \Psi_i(u_i^\star), \forall i \in \{1, \ldots, M\}. \tag{17}$$

So, for any given vector $V \in \mathbb{R}^M$, we first define implicitly the fixed point mapping $F_P(V) = U, \forall\, V \in \mathbb{R}^M$, associated to the point decomposition, where each component u_i of the vector U satisfies

$$(a_{i,i} + \sigma)u_i + w_i(u_i) = g_i - \sum_{\substack{j=1 \\ j\neq i}}^{M} a_{i,j}v_j,\ w_i(u_i) \in \Psi_i(u_i), \forall i \in \{1, \ldots, M\}. \tag{18}$$

Let us consider a vector $V' \in \mathbb{R}^M$ and consider the same relation than (18)

$$(a_{i,i} + \sigma)u_i' + w_i(u_i') = g_i - \sum_{\substack{j=1 \\ j\neq i}}^{M} a_{i,j}v_j',\ w_i(u_i') \in \Psi_i(u_i'), \forall i \in \{1, \ldots, M\}. \tag{19}$$

By subtracting and multiplying each of the M previous relations by $(u_i - u_i')$, then, on one hand, the left-hand side of the previous relations can be underestimated by $(a_{i,i} + \sigma)|u_i - u_i'|^2$, since the operator Ψ is maximal monotone; on other hand,

the right-hand side of the relation can be overestimated by using a straightforward inequality and we obtain

$$|u_i - u_i'| \leq \sum_{\substack{j=1 \\ j \neq i}}^{M} -\frac{a_{i,j}}{(a_{i,i}+\sigma)}|v_j - v_j'|, \forall i \in \{1, \ldots, M\}, \tag{20}$$

or in a vectorial norm formulation

$$|F_P(V) - F_P(V')| \leq J|V - V'|, \forall V, V' \in \mathbb{R}^M. \tag{21}$$

Note that the matrix J with diagonal entries null and off-diagonal entries equal to $-\frac{a_{i,j}}{(a_{i,i}+\sigma)}$ is the Jacobi's matrix of the matrix $A + \sigma I$. Since the matrix A is an irreducible M-matrix and $(A + \sigma I)$ is a strictly diagonal dominant M-matrix, then J is an irreducible and non-negative matrix and all eigenvalues of J have a modulus less than one. Let us denote by ν the spectral radius of J and by Θ the associated eigenvector. Classically, by the Perron-Frobenius theorem, all the components of Θ are strictly positive and the following inequality $J\Theta \leq \nu\Theta$, $\nu < 1$ is valid. Then, by a straightforward way, we obtain

$$\|F_P(V) - F_P(V')\|_{\nu,\Theta} \leq \nu\|V - V'\|_{\nu,\Theta}, \forall V, V' \in \mathbb{R}^M, 0 \leq \nu < 1, \tag{22}$$

where $\|V\|_{\nu,\Theta}$, is a uniform weighted norm defined as follows

$$\|V\|_{\nu,\Theta} = \max_{i=1,\ldots,M}\left(\frac{|v_i|}{\Theta_i}\right).$$

Then, using the last inequality (22), F_P is a contraction and we obtain a result of existence and uniqueness of both the fixed point $U^\star$ of F_P and of the solution of problem (10). Then the proof is achieved. □

3.4 A convergence result for a large blocks decomposition without overlapping

For the sake of generality, in order to apply the results of [36], presented in the present section in a simplified framework, we will solve problem (10) by a projected parallel asynchronous relaxation algorithm, related to a large block decomposition. Indeed, in parallel computation, the user does not access to as much processors as many components. Practically, the algebraic system to solve is split into $\alpha < M$ contiguous large blocks; thus the communications consist in exchanging only the necessary values of the components computed by the neighbouring processors. For simplicity, in what follows, for the problem (10), we will study the behaviour of parallel asynchronous algorithms when the space $\mathcal{E} = \mathbb{R}^M$ decomposed in α subspaces is normed by the Euclidean norm. Note that we can also consider other classical norms; but the use of such previous norms needs sophisticated mathematical tools that will complicate unnecessarily the presentation of the analysis below. So, in the sequel, we will briefly show that, under the assumptions and notations of Subsections 3.2 and 3.3, the fixed point mapping $\hat{F}$ associated to the large α-block decomposition is contractant. To prove this property, we also need to use the point decomposition which will

enable us to obtain a convergence result whatever be the large block decomposition considered in the algorithm. So let

$$\begin{cases} \alpha \in \mathbb{N} \text{ and } \{m_i\} \text{ for } i \in \{1, \ldots, \alpha\} \text{ a set of integers such that } m_i \neq 0 \\ \text{and } \sum_{i=1}^{\alpha} m_i = M; \end{cases}$$

in fact, with such notation, m_i denotes the number of components of the i-th large block. Let us also consider the numbers $r_i, i = 1, \ldots, \alpha + 1$ defined by

$$\begin{cases} r_1 = 0 \text{ and } \forall i \in \{2, \ldots, \alpha + 1\}, r_i = \sum_{j=1}^{i-1} m_j \text{ and } r_{\alpha+1} = M, \\ \text{and } \forall i \in \{1, \ldots, \alpha\}, \hat{\mathcal{E}}_i = \prod_{l=r_i+1}^{r_{i+1}} \mathcal{E}_l, \text{ with } \mathcal{E}_l = \mathbb{R}, \forall l \in \{1, \ldots, M\}, \end{cases}$$

where the i-th large block $\hat{V}_i$ is constituted by gathering the $(r_i + 1)$-th component until the $r_{i+1} - th$ component of the vector V of the point decomposition; in fact the numbers $r_i, i = 1, \ldots, \alpha + 1$, are pointers and allow to find the beginning and the end of each block $\hat{V}_i$. Obviously,

$$\mathcal{E} = \prod_{i=1}^{\alpha} \hat{\mathcal{E}}_i, \text{ and } V = \{\hat{V}_1, \ldots, \hat{V}_\alpha\} \in \prod_{i=1}^{\alpha} \hat{\mathcal{E}}_i.$$

Thus, let us consider (18) and (19); by a similar way than the one considered in the proof of Proposition 1, we obtain for all $l \in \{1, \ldots, M\}$

$$(a_{l,l}+\sigma)|u_l-u'_l|^2+ < w_l(u_l)-w_l(u'_l), u_l-u'_l > + \sum_{\substack{k=1 \\ k\neq l}}^{M} a_{l,k} < u_k-u'_k, u_l-u'_l >= 0, \tag{23}$$

where once again $|u_l|$ denotes here the absolute value of u_l, $w_l(u_l) \in \Psi_l(u_l)$ and $w_l(u'_l) \in \Psi_l(u'_l)$; A being an M-matrix then we have

$$a_{l,l} < u_l - u'_l, u_l - u'_l > \geq a_{l,l}|u_l - u'_l|^2, \forall l \in \{1, \ldots, M\},$$

and since $a_{l,k} \leq 0$ then

$$a_{l,k} < u_l - u'_l, u_k - u'_k > \geq a_{l,k}|u_l - u'_l|.|u_k - u'_k|, \forall l \neq k;$$

then, by adding all the previous inequalities and taking into account on one hand the monotony of Ψ and on the second hand (23), it follows that $\forall l \in \{1, \ldots, M\}$

$$< \sigma(u_l-u'_l)+\sum_{k=1}^{M} a_{l,k}(u_k-u'_k), u_l-u'_l > \geq \sigma.|u_l-u'_l|^2+\sum_{k=1}^{M} a_{l,k}.|u_l-u'_l|.|u_k-u'_k|. \tag{24}$$

A large block decomposition of problem (10) is defined by

$$\hat{W}_i(\hat{U}_i) + \sigma.\hat{U}_i + \sum_{j=1}^{\alpha} \hat{A}_{i,j}\hat{U}_j - \hat{G}_i = 0,\ \hat{W}_i(\hat{U}_i) \in \hat{\Psi}_i(\hat{U}_i), \forall i \in \{1,\ldots,\alpha\}; \quad (25)$$

such block decomposition is obtained by gathering the components of the point decomposition in agreement with (18) and can be expressed with respect to the point decomposition as follows

$$\hat{U}_i = \{\ldots, u_l, \ldots\},\ \text{for } l \in \{r_i + 1, \ldots, r_{i+1}\}, \quad (26)$$

in which $\hat{A}_{i,j}$ correspond to the block decomposition of the matrix A and $\hat{G}_i, i = 1,\ldots,\alpha$, denotes the i-th block component of the vector G defined by a similar way as (26).

Let us consider a block splitting of $A = \hat{D} - (\hat{L} + \hat{R})$ into $\alpha \times \alpha$ blocks $(\hat{A}_{i,j})$, such that for $i, j \in \{1,\ldots,\alpha\}$, the entries of $\hat{A}_{i,j}$, are $a_{l,k}$ for $l \in \{r_i + 1, \ldots, r_{i+1}\}$ and $k \in \{r_j + 1, \ldots, r_{j+1}\}$. Let us also define the block diagonal matrix $\hat{D}$ with diagonal blocks $\hat{D}_{i,i} = \hat{A}_{i,i}, i \in \{1,\ldots,\alpha\}$ defined by $\hat{D}_{i,i} = (a_{l,k}), l, k \in \{r_i + 1, \ldots, r_{i+1}\}$. The strictly lower block part $\hat{L} = (\hat{L}_{i,j})$ is defined by $\hat{L}_{i,j} = (-a_{l,k}), l \in \{r_i + 1, \ldots, r_{i+1}\}, k \in \{r_j + 1, \ldots, r_{j+1}\}$ for $i \in \{2,\ldots,\alpha\}$ and $j \in \{1,\ldots,i-1\}$, i.e. $j < i$ and, similarly, the strictly upper block part $\hat{R} = (\hat{R}_{i,j})$ by $\hat{R}_{i,j} = (-a_{l,k}), l \in \{r_i + 1, \ldots, r_{i+1}\}, k \in \{r_j + 1, \ldots, r_{j+1}\}$, for $i \in \{1,\ldots,\alpha-1\}$ and $j \in \{i + 1, \ldots, \alpha\}$, i.e. $j > i$; in other words, we have

$$\hat{L}_{i,j} = \begin{cases} -\hat{A}_{i,j} \text{ if } i > j \\ 0 \text{ if } i \leq j \end{cases} \text{ and } \hat{R}_{i,j} = \begin{cases} -\hat{A}_{i,j} \text{ if } i < j \\ 0 \text{ if } j \leq i \end{cases}$$

So, using theses notations we have the following result.

Proposition 2 *Consider the problem (10). Assume that the assumptions of Proposition 1 hold, particularly that Ψ is a diagonal monotone maximal operator and (11) is verified, i.e. A is an M-matrix. Consider any block decomposition in α blocks such that $\alpha < M$; let $\hat{F}$ the associated fixed point mapping. Then, for any block decomposition, $\hat{F}$ satisfies the following vectorial norm inequality*

$$|\hat{F}(V) - \hat{F}(V')| \leq (\hat{D} + \sigma.I)^{-1}.(\hat{L} + \hat{R}).|V - V'|, \forall V, V' \in \mathbb{R}^M, \quad (27)$$

in which

$$\hat{J} = (\hat{D} + \sigma.I)^{-1}(\hat{L} + \hat{R})$$

is a contraction matrix for the fixed point mapping $\hat{F}$. Furthermore, whatever be the initial guess U^0, the parallel asynchronous methods associated to any block decomposition converge to $U^\star$, fixed point of $\hat{F}$.

Proof For all $i \in \{1,\ldots,\alpha\}$, by a similar way as (18), we can define implicitly the fixed point mapping $\hat{F}$ associated to the block decomposition of problem (10) and such that $\hat{F}(\hat{V}) = \hat{U}, \forall \hat{V} \in \mathcal{E}$ as follows

$$(\sigma.I + \hat{A}_{i,i})\hat{U}_i + \hat{W}_i(\hat{U}_i) = \hat{G}_i - \sum_{\substack{j=1 \\ j \neq i}}^{\alpha} \hat{A}_{i,j}\hat{V}_j,\ \hat{W}_i(\hat{U}_i) \in \hat{\Psi}_i(\hat{U}_i), \forall i \in \{1,\ldots,\alpha\},$$

where, in such decomposition $\hat{U}_i$ is defined by (26).

Consider also the same problem associated to a distinct vector V', so that

$$(\sigma.I + \hat{A}_{i,i})\hat{U}'_i + \hat{W}_i(\hat{U}'_i) = \hat{G}_i - \sum_{\substack{j=1 \\ j\neq i}}^{\alpha} \hat{A}_{i,j}\hat{V}'_j,\ \hat{W}_i(\hat{U}'_i) \in \hat{\Psi}_i(\hat{U}'_i), \forall i \in \{1,\ldots,\alpha\}.$$

Then, starting from the block decomposition (25), we can reformulate the same problem with respect to the point decomposition; by substracting and multiplying by $u_l - u'_l$, we obtain for $l \in \{1,\ldots,M\}$

$$< w_l(u_l) - w_l(u'_l) + (\sigma + a_{l,l})(u_l - u'_l) + \sum_{\substack{k=1 \\ k\neq l}}^{M} a_{l,k}(v_k - v'_k), u_l - u'_l >= 0;$$

since the mapping Ψ is monotone, and adapting the notations used herein with those of the inequality (24), we obtain for $l \in \{r_i + 1,\ldots,r_{i+1}\}$,

$$\sigma.|u_l - u'_l| + \sum_{k=r_i+1}^{r_{i+1}} a_{l,k}|u_k - u'_k| \leq -\sum_{k=1}^{r_i} a_{l,k}|v_k - v'_k| - \sum_{k=r_{i+1}+1}^{M} a_{l,k}|v_k - v'_k|. \quad (28)$$

Then, using the same notations as the ones used in paragraph 3.3, we can write the inequality (28) in the vectorial norm formulation similar to the inequality (27) valid for any block decomposition. Since A is an M-matrix, then $\hat{D}$ is also an M-matrix and obviously $(\hat{D} + \sigma.I)$ is an M-matrix, so that $(\hat{D} + \sigma.I)^{-1} \geq 0$; moreover $(A + \sigma.I)$ being an M-matrix, then $A = (\hat{D} + \sigma.I) - (\hat{L} + \hat{R})$ is a regular splitting of $(A + \sigma.I)$ (see [38] on pages 56 to 58) so that the spectral radius $\hat{\nu}$ of the non-negative matrix $\hat{J} = (\hat{D} + \sigma.I)^{-1}(\hat{L} + \hat{R})$ satisfies

$$\hat{\nu} < 1;$$

thus, for the splitting corresponding to any block decomposition, the block matrix $\hat{J}$ is a contraction matrix with respect to the vectorial norm $|\,.\,|$. Then, for any block decomposition, whatever be the initial guess U^0, the convergence of the parallel asynchronous, synchronous and sequential iterations described by (13)–(15), applied to the numerical solution of problem (10), results now either from [7] or [34] associated to the property of contraction with respect to a vectorial norm of the mapping $\hat{F}$ or directly to the property of contraction of $\hat{F}$ in the usual sense (22) by applying the result of [19], which achieves the proof. □

Remark 3 In fact, the result of Proposition 2 is a corollary of a more general result presented in [36] (see particularly the result of Proposition 6) in a general framework where we have to consider a large α-block decomposition and another β-block decomposition such that $\alpha < \beta < M$: thus in this case the $\beta-$decomposition is not necessarily the point decomposition but any sharper block decomposition. Then, in such block decomposition situation, it is necessary to use a technical tool, the so-called notion of M-minorant, which is an M-matrix associated to the considered β-block decomposition; for bibliographical details, the reader is referred to [16, 17, 21,

22, 32, 33, 39–41]. Thus, if we consider now for example the natural block decomposition of the problem to solve, by a similar process as the one previously used to state the result of Proposition 2, we can obtain a similar result; moreover, when we consider a general nonlinear problem with once again a block decomposition like the one considered in [36], the notion of M-minorant is necessary to state a similar result to the one stated in Proposition 2 (for more details see Proposition 6 in [36]). Note that for the previous point decomposition considered in the proof of Proposition 2, the matrix $A+\sigma I$ being an M-matrix, this discretization matrix plays also the role of M-minorant defined by (24), this case corresponding to a limit case. So, in the present subsection, the situation is simplified since the discretization matrix $(A+\sigma I)$ is its own M-minorant; in such case, the introduction of the M-minorant is not necessary, and in order to obtain the convergence of the parallel asynchronous algorithms for any block decomposition of the problem, it is only sufficient to use the notion of regular splitting. Moreover, this process is still applicable with appropriate assumptions, i.e. the determination of a M-minorant, even if A is not an M-matrix. Besides, in [36], the space $\mathbb{R}^M$ is normed by any classical norms, i.e. the l_1-norm, the Euclidean norm and the uniform norm; using l_1 and uniform norms result in the loss of the Hilbertian structure in $\mathbb{R}^M$ and require the utilization of other mathematical sophisticated notions. In this theoretical context, we can obtain more practical results and criteria allowing to determine the M-minorant in various topological contexts. For more details, the reader is referred to [36] and to [23].

Remark 4 The approach previously developed is directly used to deal with specific block structure related to structured matrices obtained from finite difference and finite volume discretizations. In fact, the interesting property arising in the assumptions of Proposition 2 is that the spatial discretization matrix is an M-matrix. Such situation occurs also when particular finite element method is used for the spatial discretization. Indeed, consider the more general elliptic boundary value problem defined in an open bounded set $\Omega \subset \mathbb{R}^2$ equipped with Dirichlet boundary condition on one part Γ_0 of the boundary and Fourier boundary condition on the other part Γ_1

$$\begin{cases} -\frac{\partial}{\partial x_1}(p_1\frac{\partial u}{\partial x_1}) - \frac{\partial}{\partial x_2}(p_2\frac{\partial u}{\partial x_2}) + \bar{\sigma}u = g, \text{ in } \Omega, \sigma \geq 0, \\ u = \gamma \text{ on } \Gamma_0, \\ \frac{\partial u}{\partial n} + \theta u = \xi, \theta > 0, \text{ on } \Gamma_1, \end{cases}$$

where $p_i = p_i(x_1, x_2), i = 1, 2$ and $\bar{\sigma} = \bar{\sigma}(x_1, x_2) \geq 0$ are continuous, g, γ and ξ satisfying classical assumptions. If we consider the discretization of this general diffusion problem by a finite element method using piecewise linear basis functions P1 or bilinear basis functions Q1, then the discretization matrix is an M-matrix when any vertex angle of the elements constituting the mesh is smaller than $\frac{\pi}{2}$ in the case of piecewise linear interpolation and the classical conforming condition in the case of bilinear interpolation, corresponding in fact to the condition of non-degeneration of the angles of the elements (see [1], Theorem 5.2 on page 203). Note that the piecewise linear interpolation and the bilinear interpolation are the most used in finite element discretization and give similar accuracy to the discretization by finite difference method. Then, classical finite element discretizations with appropriate

assumptions lead to spatial discretization matrices that are M-matrices. Note that finite element discretization is well appropriate for any domain Ω and does not lead necessarily to structured matrices. In conclusion, in such context of finite element approximation, the discretization matrix being an M-matrix, the result of Proposition 2 is still valid for any block decomposition of the underlying system of linear equations.

Remark 5 Note that, since the unilateral constraint is valid only on Γ_1, for a subdomain Ω_i such that $\bar{\Omega}_i \bigcap \Gamma_1 = \emptyset$, (10) reduces to an algebraic single-valued equation; consequently, (16)–(17) are valid only on the points belonging to Γ_1. Nevertheless, for the other points of Ω which do not belong to Γ_1, the previous analysis of the convergence is still valid since, in this case, $\Psi(U) \equiv 0$, this last operator being, as previously said, obviously monotone.

3.5 Parallel asynchronous methods with overlapping

3.5.1 Schwarz alternating method in the continuous case

The Schwarz alternating algorithm is well suited to the parallel solution of boundary values problems (see [25]). In these kind of methods, in order to parallelize the computation, the domain where a partial differential equation is defined is split into overlapping subdomains. Thus, sequences of smaller subproblems are solved on each processor of a parallel computer in order to compute a solution of the global problem; practically more accuracy is obtained. Consider a boundary value problem $\Lambda u = f$ defined in a domain Ω with boundary condition $Bu = g$ on Γ. For the sake of simplicity, we consider a decomposition in two subdomains Ω_1 and Ω_2 and we denote $\partial\Omega_i$, $i = 1, 2$ the boundary of each Ω_i, $i = 1, 2$.

Parallel asynchronous Schwarz algorithms for two processors consist in solving at each iteration

$$\left\{\begin{array}{l} \Lambda_1 u_1^{p+1} = f_1 \text{ in } \Omega_1 \\ B_1 u_1^{p+1} = g_1 \text{ on } \Gamma \cap \Omega_1 \\ \quad u_1^{p+1} = \tilde{u}_2^p \text{ on } \partial\Omega_1 \cap \Omega_2 \end{array}\right. \text{ and } \left\{\begin{array}{l} \Lambda_2 u_2^{p+1} = f_2 \text{ in } \Omega_2 \\ B_2 u_2^{p+1} = g_2 \text{ on } \Gamma \cap \Omega_2 \\ \quad u_2^{p+1} = \tilde{u}_1^p \text{ on } \partial\Omega_2 \cap \Omega_1 \end{array}\right. \tag{29}$$

where Λ_i, $i = 1, 2$ and B_i, $i = 1, 2$ correspond to the decomposition of the operators Λ and B according to the considered decomposition and $\tilde{u}_1^p$ and $\tilde{u}_2^p$ denote the available values of the components of the iterate vector (u_1, u_2) at the current iteration. In the synchronous algorithm, $\tilde{u}_1^p = u_1^p$ and $\tilde{u}_2^p = u_2^p$. Besides, in the classical asynchronous algorithm (see [15, 34]), these components may be delayed as follows $\tilde{u}_1^p = u_1^{\rho_1(p)}$ and $\tilde{u}_2^p = u_2^{\rho_2(p)}$, with $\rho_i(p) \geq 0$, $i = 1, 2$. Finally, in the case of asynchronous algorithm with flexible communications (see [35]), $\tilde{u}_i^p$ are not necessarily associated to components that are labelled by an outer iteration number as communications may occur at any time. Then, in this class of method, partial updates, i.e. the current values of any component of the iterate vector, can be used at any time in the computation; as a consequence, the coupling between communications and computations can be improved.

In the sequel, we will focus on the solution by parallel asynchronous Schwarz method applied to the particular nonlinear multivalued operators Λ, constituted by an affine operator perturbed by a multivalued diagonal operator.

3.5.2 Convergence analysis of parallel asynchronous Schwarz method

We consider the same discretization as the one previously shown in Section 3.1; so using the same notations, we have to solve the nonlinear algebraic system (10). The numerical solution of (10) by the Schwarz alternating method consists to the solution of the following system

$$(\tilde{\mathcal{A}} + \sigma\tilde{\mathcal{I}})\tilde{\mathcal{U}} - \tilde{\mathcal{G}} + \tilde{\mathbf{\Psi}}(\tilde{\mathcal{U}}) \ni 0, \tag{30}$$

where $\tilde{\mathcal{A}}$, $\tilde{\mathcal{I}}$, $\tilde{\mathcal{U}}$, $\tilde{\mathcal{G}}$ and $\tilde{\mathbf{\Psi}}(\tilde{\mathcal{U}})$ are derived from the augmentation process of the Schwarz alternating method (see [20] and [35]).

Let $\alpha \in \mathbb{N}$ be a positive integer and consider now the following block decomposition of problem (30) into α subproblems

$$(\tilde{\mathcal{A}}_{i,i} + \sigma\tilde{\mathcal{I}}_i)\tilde{\mathcal{U}}_i^{\star} - \tilde{\mathcal{G}}_i + \sum_{\substack{j=1 \\ j\neq i}}^{\alpha} \tilde{\mathcal{A}}_{i,j}\tilde{\mathcal{U}}_j^{\star} + \tilde{\mathbf{\Psi}}_i(\tilde{\mathcal{U}}_i^{\star}) \ni 0, \forall i \in \{1, \ldots, \alpha\}, \tag{31}$$

with $\tilde{\mathcal{U}}_i \in \mathbb{R}^{\tilde{m}_i}$, $\tilde{\mathcal{G}}_i \in \mathbb{R}^{\tilde{m}_i}$, where $\tilde{m}_i$ denotes the size of the i-th block of the previous vectors and $\tilde{\mathcal{I}} = (\tilde{\mathcal{I}}_i)_{1\leq i\leq\alpha}$, $\tilde{\mathcal{A}} = (\tilde{\mathcal{A}}_{i,j})_{1\leq i,j\leq\alpha}$, according to the associated block decomposition. By choosing an element of $\tilde{\mathbf{\Psi}}_i(\tilde{\mathcal{U}}_i^{\star})$, (31) can be written as follows

$$(\tilde{\mathcal{A}}_{i,i}+\sigma\tilde{\mathcal{I}}_i)\tilde{\mathcal{U}}_i^{\star}-\tilde{\mathcal{G}}_i+\sum_{\substack{j=1 \\ j\neq i}}^{\alpha} \tilde{\mathcal{A}}_{i,j}\tilde{\mathcal{U}}_j^{\star}+\tilde{\mathcal{W}}_i(\tilde{\mathcal{U}}_i^{\star}) = 0, \tilde{\mathcal{W}}_i(\tilde{\mathcal{U}}_i^{\star}) \in \tilde{\mathbf{\Psi}}_i(\tilde{\mathcal{U}}_i^{\star}), \forall i \in \{1, \ldots, \alpha\}. \tag{32}$$

Consider now the solution of subproblems (31) by the following asynchronous parallel iteration

$$\begin{cases} (\tilde{\mathcal{A}}_{i,i} + \sigma\tilde{\mathcal{I}}_i)\tilde{\mathcal{U}}_i^{p+1} + \tilde{\mathbf{\Psi}}_i(\tilde{\mathcal{U}}_i^{p+1}) \ni \tilde{\mathcal{G}}_i - \sum_{\substack{j=1 \\ j\neq i}}^{\alpha} \tilde{\mathcal{A}}_{i,j}\tilde{\mathcal{V}}_j, \text{ if } i \in s(p), \\ \tilde{\mathcal{U}}_i^{p+1} = \tilde{\mathcal{U}}_i^{p}, \text{ if } i \notin s(p), \end{cases} \tag{33}$$

where $\{\tilde{\mathcal{V}}_1, \tilde{\mathcal{V}}_2, \ldots, \tilde{\mathcal{V}}_\alpha\}$ are the available values of the components $(\tilde{\mathcal{U}}_j)_{j\neq i}$ defined by $\tilde{\mathcal{V}}_j = \tilde{\mathcal{U}}_j^{\rho_j(p)}$ according to (13)–(15). Moreover, $\mathcal{S} = \{s(p)\}_{p\in\mathbb{N}}$ is defined according to (14).

Remark 6 Similarly to Section 3.2, when $\rho_i(p) = p$ for all $i \in \{1, \ldots, \alpha\}$ and for all $p \in \mathbb{N}$, then (33) models a synchronous Schwarz alternating method.

This process is in fact a theoretical model that represents the solution of the algebraic system (10) by a Schwarz alternating method. In the implementation of the algorithms, $\tilde{\mathcal{A}}$ and $\tilde{\mathcal{G}}$ are not explicitly computed.

Proposition 3 *Consider the problem (30) to solve. Assume that (11) is verified and that Ψ is a diagonal monotone maximal operator. Then, we can associate to problem (30) a contracting fixed point mapping $\mathcal{F}_P$ and there exists one and only one fixed point $\tilde{\mathcal{U}}^\star$ also unique solution of the discretized problem (30). Moreover, $\tilde{\mathcal{V}} = \{\tilde{\mathcal{V}}_1, \ldots, \tilde{\mathcal{V}}_\alpha\}$ being defined according to (13)–(15), the parallel asynchronous and synchronous Schwarz alternating methods starting from any initial guess U^0 and defined by (33), applied to the numerical solution of problem (10), converge to the solution of the system (10).*

Proof Since A is an M-matrix, the augmented matrix $\tilde{\mathcal{A}}$, derived from the augmentation process by the Schwarz alternating method, has the important property of being an M-matrix (see [20]). Since $\sigma > 0$ then σI is a diagonal matrix with strictly positive diagonal entries; then $\tilde{\mathcal{A}} + \sigma\tilde{\mathcal{I}}$ is also an M-matrix. Moreover, $\tilde{\mathbf{\Psi}}(\tilde{\mathcal{U}})$ also derived from the augmentation process is obviously a diagonal monotone maximal operator. Thus, the perturbation of the affine mapping $(\tilde{\mathcal{A}} + \sigma\tilde{\mathcal{I}})\tilde{\mathcal{U}} - \tilde{\mathcal{G}}$ by $\tilde{\mathbf{\Psi}}(\tilde{\mathcal{U}})$ is such that the augmented system (30) has the same properties than those of the initial algebraic system (10). Then, the proof follows in two distinct steps. For the study of the convergence of the parallel asynchronous Schwarz method (33), in a similar way than the one previously used in Proposition 2, we can consider first a point decomposition of problem (30) and conclude that the associated fixed point mapping $\tilde{\mathcal{F}}_P$ is contracting for this particular point decomposition; afterwards, for every coarser block decomposition (33), we use once again the result presented in Subsection 3.4 in an Hilbertian framework (or more generally the results of Proposition 6 of [36] in the general topological framework), and we obtain a property of contraction of the fixed point mapping associated to this large block decomposition. Moreover, for the convergence of the parallel asynchronous and synchronous Schwarz alternating methods (33), we use a similar argument than the one used in the proof of Proposition 2 which achieves the proof. □

Remark 7 We refer to [35] for parallel asynchronous Schwarz alternating method with flexible communications between the processors, corresponding to a more general model of parallel asynchronous iterations. In fact, the Schwarz alternating method is a particular class of subdomain method. More generally, we can consider the parallel synchronous or asynchronous multisplitting method, corresponding to a more general iterative method. The convergence of such parallel asynchronous multisplitting method can also be studied in an analogous way than [35] (see also [42]).

Remark 8 In order to measure the amount of computation required to reach convergence, we will use in the presented results of parallel experiments the number of relaxations instead of the number of iterations. A relaxation corresponds to the update like (13). This definition holds in both sequential, synchronous and asynchronous cases. An iteration corresponds to the update of at least all the components i, for $i = 1, \ldots, \alpha$ in a parallel synchronous or sequential way. As the formulation of parallel synchronous iteration cannot be extended immediately to the asynchronous case, we prefer using relaxation count as an indicator of the amount of floating operations required to reach convergence.

4 Parallel experiments

4.1 Implementation

The implementation of parallel algorithms have been carried out on heterogeneous and distant clusters constituting a grid platform. Matrix and right-hand side creation have been implemented by a parallel way. In the case of parallel Schwarz alternating method, subdomains may overlap each other. The principle of implementation of parallel asynchronous and synchronous iterative methods can be summarized as follows:

Do until global convergence
For each block **do**
- Perform communications of block boundary values
- Perform block relaxation

End For
End Do

For the implementation of the Schwarz alternating method, the domain Ω, where the boundary value problem is defined, is split into overlapping parallelepiped. Minimal values have been chosen for overlapping between them (one mesh point). The method without overlapping corresponds to a straightforward block relaxation method, with communications of the values of the components associated to the boundaries of a block; so such method does not present any difficulty. Thus, sequences of smaller subproblems are solved on each processor of the parallel computer in order to compute a solution of the global problem; practically more accuracy is obtained. Several blocks are assigned to each processor in order to implement a strategy which is close to the multiplicative one. To obtain a faster convergence of the parallel computations, each processor handles contiguous blocks, numbered according to red-black or lexicographic ordering ; note that such red-black ordering is more appropriate for parallel computations, and in this case, the convergence results of Propositions 2 and 3 still hold by a straightforward way.

For the parallel asynchronous methods, the stopping criterion used a decentralized algorithm where asynchronous convergence detection is achieved with a token circulation technique. Each processor updates the components of the iterate vector associated with each one's own block and computes the residual norm attached to this block in order to participate to the convergence detection. For simplicity, with respect to the block decomposition considered in the Subsections 3.4 and 3.5.2 the block partitioning retained for the assignation of the large blocks to the processors is very simple; in fact, in order to define a large block, we have gathered several adjacent blocks of the natural block decomposition. So, on each large block associated to each process, a classical and sequential block relaxation method is used in order to solve each subproblem; this kind of method is implemented by solving each tridiagonal block by the TDMA method, corresponding to the Thomas's elimination method applied for tridiagonal block; in order to reduce the elapsed time of computation, note that the implementation of the block relaxation method is optimized by elimination of sequences of redundant code. Indeed, in this implementation, we distinguish the mesh points belonging to Γ_0 and to Γ_1; each case is not coded by

Table 1 Characteristics of machines on each site

Site	Cluster	Processor type	Speed	CPU	Core	RAM size
Rennes	Paradent	Intel Quad Xeon L5420	2.5 GHz	2	4	32 GB
Sophia	Suno	Intel Quad Xeon E5520	2.26 GHz	2	4	32 GB
Toulouse	Pastel	AMD Opteron 2218	2.6 GHz	2	2	8 GB

a similar way. For the diagonal blocks corresponding to the points belonging to Γ_0, the LU decomposition of each diagonal block is performed only once in the initialization phase. For the diagonal block corresponding to the points belonging to Γ_1, the LU decomposition of each diagonal block is performed when the diagonal block changes. Then, this optimization allows to decrease the elapsed time.

Convergence detection was performed via a snapshot algorithm (see [8], section 8.2 and [10]), corresponding to a variant of Lamport's method. Convergence occurs when a given predicate on a global state is true. A usual predicate corresponds to the fact that the iterate vector generated by the asynchronous iterative algorithm is sufficiently close to a solution of the problem (see [8], page 580); then, on every process, the norm of the local residual remains under a given threshold after two successive updates of the component (see [8], [37]). Due to the termination and the detection of the global state of the processes, the implementation of each variant of parallel asynchronous method is more complex than the synchronous one.

In the case of asynchronous iterations, point to point communications between two processes have been implemented using nonblocking MPI send and receive subroutines. MPI_TEST is used in order to allow any processor to continue the computations without having to wait for the completion of any pending send or receive operations. Idle times due to synchronizations in message passing are suppressed in this way. On the other hand, the parallel synchronous iterative schemes are implemented by using MPI_WAIT. One has to take care about the deadlock issue when implementing synchronous communications using blocking MPI subroutines.

For more details concerning the implementation and the convergence detection of the considered parallel asynchronous methods, the reader is referred to [11].

Table 2 Elapsed time (sec.) and number of relaxations with sequential algorithm on each site

	Rennes		Sophia		Toulouse	
Mesh	Parallel subdomain method without overlapping					
Very fine	7296 s	1950	4442 s	1950	None	None
Fine	2974 s	1542	1798 s	1542	2964 s	1542
	Parallel subdomain Schwarz alternating method					
Very fine	9324 s	1112	7104 s	1112	None	None
Fine	4475 s	1015	3416 s	1015	5404 s	1015

Table 3 Elapsed time and average number of relaxations for parallel method without overlapping with 400^3 points on two sites

	Asynchronous				Synchronous		
Proc	Time/s	Rel.Min	Rel.Max	Average	Time/s	Relaxations	τ
4	2021	2166	3056	2607	3160	1950	1.56
8	1249	2692	3878	3295	1912	1963	1.53
16	538	2369	3607	2988	1130	1963	2.10
32	313	2696	4402	3457	426	1963	1.36
64	152	2621	4237	3369	767	1976	5.05

4.2 Numerical experiments

Computational experiments have been carried out on the GRID 5000 platform. This French grid platform is composed of 2970 processors with a total of 6906 cores distributed over nine sites in France. Most of them have at least a Gigabit Ethernet. The machines of the GRID 5000 platform used are located in Sophia-Antipolis, Rennes and Toulouse sites and run Linux 64 bits. The characteristics of machines are summarized in Table 1.

For each method presented in Section 3, we have considered a cubic domain $\Omega \subset \mathbb{R}^3$; this domain Ω has been discretized with $M = (q+1)q^2$ points, where q denotes the number of inner points on each edge of the cube. The iterate vector is split into $q(q+1)$ blocks with q components. A set of adjacent sub-blocks is assigned to each node and it is updated using a parallel relaxation method. In the sequential and parallel experiments, we have considered several regular meshes: one fine mesh[4] and one very fine mesh.[5]

The parallel algorithms with and without overlapping have been implemented and the parallelization is achieved using MPI communication library.

The results of experiments are summarized in Tables 2, 3, 4, 5 and 6 and represented by Figs. 1, 2 and 3. The Table 2 displays the sequential elapsed time obtained on each cluster for the two meshes considered. The first line corresponds to the very fine mesh and the second line to the fine mesh. Due to the utilization constraints of the grid platform in dedicated exploitation and taking also into account the size of the linear systems to solve, the tests have been performed on two distant clusters with high memory for very fine mesh and on three distant clusters for the other size of mesh.

The experiment studies use 4, 8, 16, 32, 64, and 128 computing machines for the two meshes, except 128 for the very fine mesh. The parallel computational experiments are summarized in Table 3 and in Table 4 for the parallel method without overlapping. Tables 5 and 6 display the results obtained when parallel

[4] $320 \times 320 \times 320 = 32\,768\,000$ points.
[5] $400 \times 400 \times 400 = 64\,000\,000$ points.

Table 4 Elapsed time and average number of relaxations for parallel method without overlapping with 320^3 points on three sites

	Asynchronous				Synchronous		
Proc	Time/s	Rel.Min	Rel.Max	Average	Time/s	Relaxations	τ
4	962	1807	3092	2239	2316	1542	2.41
8	619	2532	3935	2866	1878	1554	3.03
16	281	2190	3728	2747	1565	1554	5.57
32	175	2779	4776	3513	1416	1554	8.09
64	70	2122	3733	2738	1041	1568	14.87
128	38	2038	3844	2823	260	1577	6.84

Schwarz alternating method is used. In Tables 3, 4, 5 and 6, the number of relaxations is mentioned; note that for only the asynchronous experiments, min, max and average number of relaxations are considered. In these tables and figures, comparison of elapsed time for parallel synchronous and asynchronous methods are presented; the parameter τ measures the ratio of elapsed time between synchronous and asynchronous methods.

In the parallel simulations only one core of each machine could be used. Indeed the amount of data to be stored in each RAM memory, for each MPI process, is very large; consequently, very large memory size is needed, in fact greater than 4 GB. On other hand, the advantage of using 64 bits machines and compilers is the capability to use RAM with size greater than 4 GB. Thus, in the case, where many cores per machine are used, due to limited RAM on each workstation, disc swapping will occur, and consequently the performances of parallel algorithms will decrease. The number of machines used has been limited up to 64 when we consider a mesh with 400^3 mesh points processed on two sites and 128 when we consider a mesh with 320^3 mesh points processed on three sites, according to the duration of the experiments and of

Table 5 Elapsed time and average number of relaxations for parallel Schwarz alternating method with 400^3 points on two sites

	Asynchronous				Synchronous		
Proc	Time/s	Rel.Min	Rel.Max	Average	Time/s	Relaxations	τ
4	3075	1422	1785	1587	4603	1112	1.50
8	1413	1320	1694	1506	2235	1112	1.58
16	740	1325	1806	1549	1084	1112	1.46
32	405	1385	2006	1613	511	1112	1.26
64	185	916	1837	1441	522	1112	2.82

Table 6 Elapsed time and average number of relaxations for parallel Schwarz alternating method with 320^3 points on three sites

	Asynchronous				Synchronous		
Proc	Time/s	Rel.Min	Rel.Max	Average	Time/s	Relaxations	τ
4	1432	1234	1321	1355	2449	1015	1.71
8	648	1144	1476	1267	1257	1015	1.94
16	339	1165	1580	1354	616	1015	1.57
32	194	1144	1878	1458	548	1015	2.82
64	87	841	1661	1256	456	1015	5.24
128	37	517	2037	1251	248	1015	6.70

constraints of exploitation; moreover, since the overhead damages the performances of the parallel algorithms, additional machines are not really necessary. The elapsed times of the methods with and without overlapping, are relatively close; we can see this fact for parallel times execution in Tables 3 and 5, or in Tables 4 and 6. We can notice that considering the obtained elapsed times, the method without overlapping is faster. The comparison of the number of relaxations performed on each large block shows that the method with overlapping converges slightly faster. For the considered application and architecture used, the asynchronous scheme of computation scales better than the synchronous one. For example, with 64 machines, the asynchronous

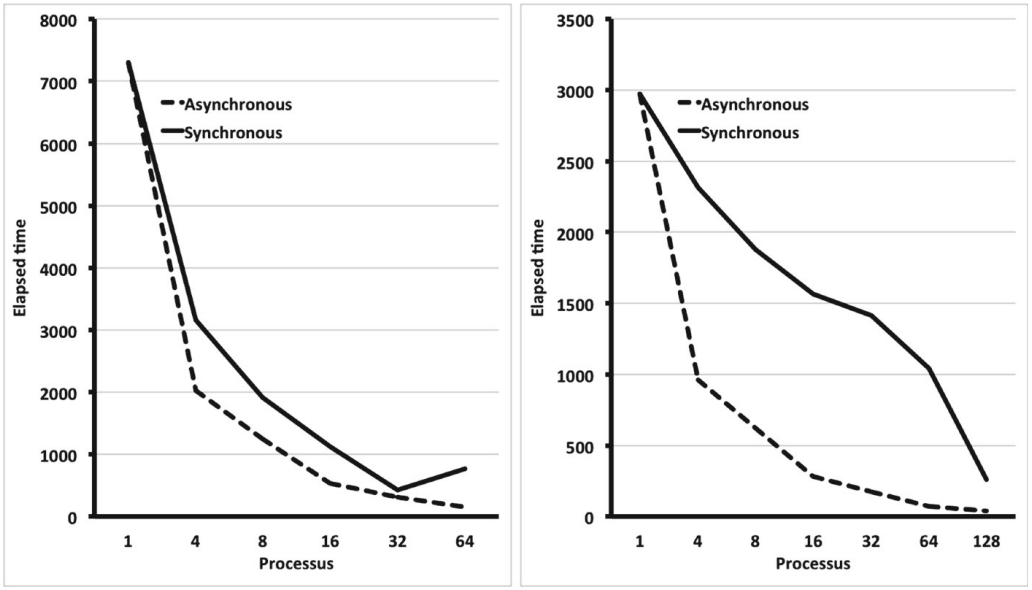


Fig. 1 Elapsed time for parallel method without overlapping with 400^3 points on two sites *(left)* and 320^3 points on three sites *(right)*

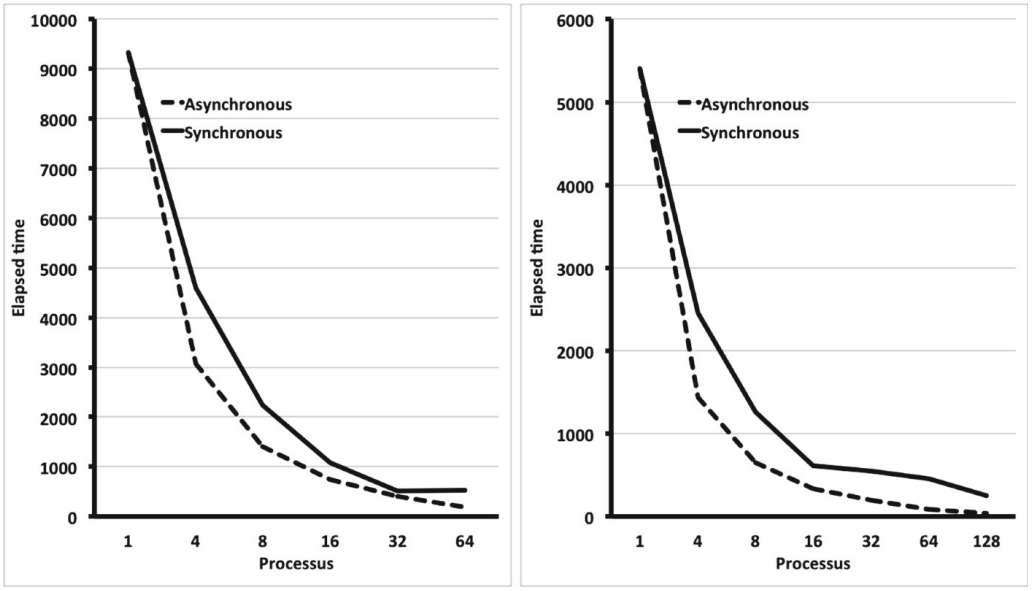


Fig. 2 Elapsed time for parallel Schwarz alternating method with 400^3 points on two sites *(left)* and 320^3 points on three sites *(right)*

computation scheme clearly performs about five times faster on two sites and about 15 times faster on three sites for the parallel method without overlapping than the synchronous one. For the same number of machines, the asynchronous method performs about three times faster on two sites and five times faster on three sites for the parallel Schwarz alternating method than the synchronous scheme. This feature shows

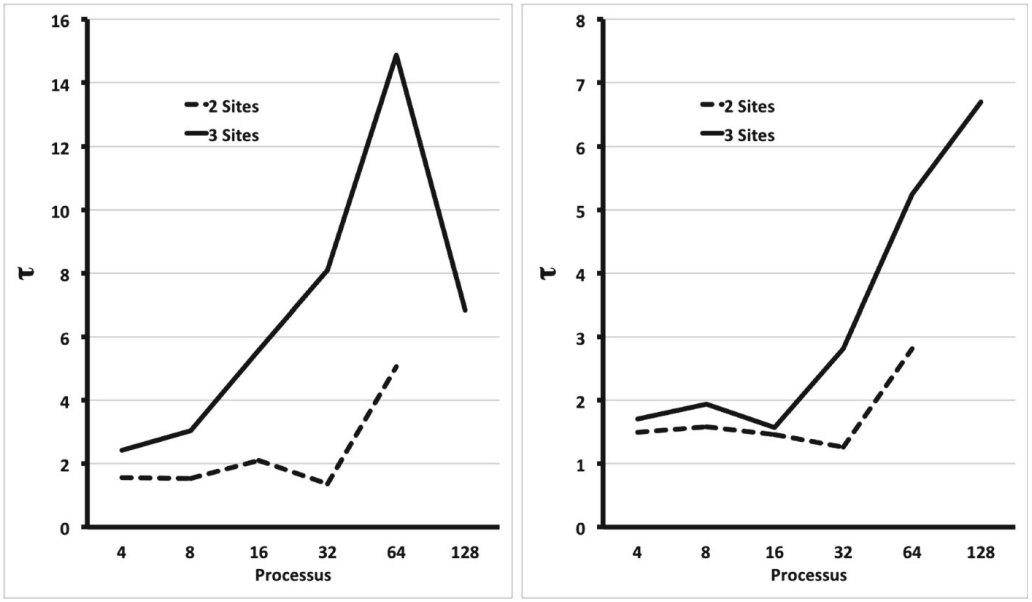


Fig. 3 Comparison of the ratio τ for each method on two and three sites for the methods without *(left)* and with overlapping *(right)* with very fine mesh *(dashed line)* and fine mesh *(continuous line)*

that asynchronous algorithms are less sensitive to granularity and network latency. Moreover the asynchronous algorithms do not generate idle time. Furthermore, load balancing is not necessary in order to decrease the elapsed time. The communication overhead induced by the parallelisation is small enough, so that positive effects of parallelism on memory management can be seen. Indeed, the more machines are used, the less data is involved in the computation on each single machine. Therefore, the proportion of data that fits in cache memory is higher when granularity is fine. In some cases, superlinear scaling has occurred in preliminary tests performed on one site.

5 Concluding remarks

In this paper, we have presented synchronous and asynchronous parallel methods in order to solve a problem arising in fluid mechanics with unilateral constraints on the boundary. Taking into account the properties of the discrete operator, the convergence of these methods is well ensured and experiments on GRID 5000 platform have been performed successfully. These experiments allowed us to compare the behaviour of both synchronous and asynchronous schemes when two or three heterogeneous and geographically distant sites are used and when two different fine meshes are considered. So, in the context of grid computing where the clusters are distant and heterogeneous, we have obtained very good performance of the parallel algorithms, particularly for the asynchronous ones.

Acknowledgments The authors wish to thank the referees for their helpful comments and remarks. This study has been made possible with the support of the GRID 5000 platform.

References

1. Axelson, O., Barker, V.A.: Finite element solution of boundary value problems. Theory and computation. Academic press (1984)
2. Badea, L., Tai, X.C., Wang, J.: Convergence rate analysis of a multiplicative Schwarz method for variational inequalities. SIAM J. on Numer. Anal. **41**(3), 1052–1073 (2004)
3. Badea, L., Wang, J.: An additive Schwarz method for variational inequalities. Math. Comp. **69**, 1341–1354 (2000)
4. Bai, Z.Z., Evans, D.J.: Matrix multisplitting methods with applications to linear complementary problems: parallel asynchronous methods. Intern J. Computer Math. **79**(2), 205–232 (2002)
5. Bai, Z.Z., Evans, D.J.: Chaotic iterative methods for the linear complementary problems: parallel asynchronous methods. J. Comput. Appl. Math. **96**, 127–138 (1998)
6. Barbu, V.: Nonlinear semigroups and differential equations in Banach spaces. Noordhoff International Publishing (1976)
7. Baudet, G.: Asynchronous iterative methods for multiprocessors. J. Assoc. Comput. Mach. **25**, 226–244 (1978)
8. Bertsekas, D.P., Tsitsiklis, J.: Parallel and Distributed Computation. Numerical Methods. Athena Scientific, Englewood Cliffs (1997)
9. Bolze, R., Cappello, F., Caron, E., Daydé, M., Desprez, F., Jeannot, E., Jégou, Y., Lanteri, S., Leduc, J., Melab, N., Mornet, G., Namyst, R., Primet, P., Quetier, B., Richard, O., Talbi, E.-G., Touche, I.: GRID 5000: A large scale and highly reconfigurable experimental grid testbed. International Journal of High Performance Computing Applications **20**(4), 481–494 (2006)

10. Chandy, K.M., Lamport, L.: Distributed snapshots : determining global states of distributed systems. ACM Trans. Comput. Syst. **3**(1), 63–75 (1985)
11. Chau, M., El Baz, D., Guivarch, R., Spitéri, P.: MPI Implementation of parallel subdomain methods for linear and nonlinear convection-diffusion problems. J. Parallel Distrib. Comput. **67**, 581–591 (2007)
12. Chau, M., Couturier, R., Bahi, J., Spiteri, P.: Parallel solution of the obstacle problem in grid environment. International Journal of High Performance Computing Applications **25**(4), 488–495 (2011)
13. Chau, M., Couturier, R., Bahi, J., Spiteri, P.: Asynchronous grid computation for American options derivative. Adv. Eng. Softw. **60-61**, 136–144 (2013)
14. Chau, M., Garcia, T., Spiteri, P.: Asynchronous Schwarz methods applied to constrained mechanical structures in grid environment. Adv. Eng. Softw. **74**, 1–15 (2014)
15. Chazan, D., Miranker, W.: Chaotic relaxation. Linear Algebra Appl. **2**, 199–222 (1969)
16. Comte, P., Miellou, J.C., Spiteri, P.: La notion de h-accrtivité, applications. CRAS **283**, 655–658 (1976)
17. Dahlquist, G.: On matrix majorants and minorants, with applications to differential equations. Linear Algebra Appl. **52/53**, 199–216 (1983)
18. Duvaut, G., Lions, J.L.: Les inéquations en mécanique. Dunod (1972)
19. El Tarazi, M.: Some convergence results for asynchronous algorithms. Numer. Math. **39**, 325–340 (1982)
20. Evans, D.J., Deren, W.: An asynchronous parallel algorithm for solving a class of nonlinear simultaneous equations. Parallel Comput. **17**, 165–180 (1991)
21. Feingold, D., Varga, R.S.: Block diagonally dominant matrices and generalizations of the Gerschgorin theorem, Pacific. J. Math. **12**, 1241–1250 (1963)
22. Fiedler, M., Ptak, V.: On matrices with non-positive off-diagonal elements and positive principal minors. Math. Jounal **12**(87), 382–400 (1962)
23. Giraud, L., Spiteri, P.: Résolution parallèle de problèmes aux limites non linéaires. M2 AN **25**, 579–606 (1991)
24. Glowinski, R., Lions, J.L., Tremolieres, R.: Analyse numérique des inéquations variationnelles, Dunod, tome 1 and 2 (1976)
25. Hoffman, K.H., Zou, J.: Parallel efficiency of domain decomposition methods. Parallel Comput. **19**, 1375–1391 (1993)
26. Kuznetsov, Y.A., Neittaanmaki, P., Tarvainen, P.: Block relaxation methods for algebraic obstacle problems with M-matrices. East-West J. Numer. Math. **4**, 69–82 (1996)
27. Kuznetsov, Y.A., Neittaanmaki, P., Tarvainen, P.: Schwarz methods for obstacle problems with convection-diffusion operators. In: Keyes, D.E., Xu, J.C. (eds.) Proceedings of Domain Decomposition Methods in Scientifical and Engineering Computing, pp. 251–256. AMS (1995)
28. Li, C., Zeng, J., Zhou, S.: Convergence analysis of generalized Schwarz algorithms for solving obstacle problems with T-monotone operator. Computers & Mathematics with Applications **48**(3-4), 373–386 (2004)
29. Lions, J.L.: Sur les problémes unilatéraux, Séminaire N. Bourbaki **350**, 55–77 (1969)
30. Lions, J.L., Stampacchia, G.: Variational inequalities. Comm. Pure Appl. Math. **XX**, 493–519 (1967)
31. Lions, P.L., Mercier, B.: Approximation numérique des équations de Hamilton-Jacobi-Bellman, R.A.I.R.O Analyse numérique **14**, 369–393 (1980)
32. Miellou, J.C.: Méthodes de Jacobi, Gauss-Seidel, sur-relaxation par blocs, appliquées à une classe de problèmes non linéaires. CRAS **273**, 1257–1260 (1971)
33. Miellou, J.C.: Sur une variante de la méthode de relaxation, appliquée à des problèmes comportant un opérateur somme d'un opérateur différentiable et d'un opérateur maximal monotone diagonal. CRAS **275**, 1107–1110 (1972)
34. Miellou, J.C.: Algorithmes de relaxation chaotique à retards. RAIRO Analyse numé,rique **R1**, 55–82 (1975)
35. Miellou, J.C., El Baz, D., Spiteri, P.: A new class of asynchronous iterative algorithms with order interval. Math. Comput. **67**(221), 237–255 (1998)
36. Miellou, J.C., Spiteri, P.: Un critère de convergence pour des méthodes générales de point fixe. M2AN **19**, 645–669 (1985)
37. Miellou, J.C., Spiteri, P., El Baz, D.: Stopping criteria, forward and backward errors for perturbed asynchronous linear fixed point methods in finite precision. IMA J. Numer. Anal. **25**, 429–442 (2005)

38. Ortega, J.M., Rheinboldt, W.C.: Iterative Solution of Nonlinear Equations in Several Variables. Academic Press, New York (1970)
39. Ostrowski, A.M.: On some metrical properties of operator matrices and matrices partitioned into blocks. J. Math. Anal. Appl. **2**, 161–209 (1961)
40. Robert, F.: Recherche d'une M-matrice parmi les minorantes d'un opérateur linéaire. Numer. Math. **9**, 189–199 (1966)
41. Schröder, J.: Nichtlineare Majoranten beim Verfahren der Schrittweisen Näherung. Arch. Math. (Bused) **7**, 471–484 (1956)
42. Spiteri, P., Miellou, J.C., El Baz, D.: Parallel asynchronous Schwarz and multisplitting methods for a nonlinear diffusion problem. Numerical Algorithms **33**, 461–474 (2003)
43. Spiteri, P., Miellou, J.C., El Baz, D.: Asynchronous Schwarz alternating method with flexible communications for the obstacle problem, réseaux et systèmes répartis - Calculateurs parallèles. Hermes **13**(1), 47–66 (2001)
44. Varga, R.: Matrix Iterative Analysis. Prentice Hall (1962)
45. Ziane-Khodja, L., Chau, M., Couturier, R., Bahi, J., Spiteri, P.: Parallel solution of American option derivatives on GPU clusters. Computers and Mathematics with Applications **65**(11), 1830–1848 (2013)
46. Méthodes de décomposition, Méthodes numériques d'analyse de systèmes, tome 2, cahier de l'IRIA 11 (1972). (see http://books.google.fr/books?id=JrkDNQEACAAJ)

5.7 Article 7

Advances in Engineering Software 74 (2014) 1–15

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Asynchronous Schwarz methods applied to constrained mechanical structures in grid environment

M. Chau [a], T. Garcia [b,c], P. Spiteri [b,*]

[a] Advanced Solutions Accelerator, 199 rue de l'Oppidum, 34170 Castelnau le Lez, France
[b] ENSEEIHT-IRIT, 2 rue Charles Camichel, 31071 Toulouse CEDEX, France
[c] UVSQ-PRISM, 45 avenue des états-unis, 78035 Versailles Cedex, France

ARTICLE INFO

Article history:
Received 14 January 2013
Received in revised form 17 March 2014
Accepted 27 March 2014
Available online 8 May 2014

Keywords:
Obstacle problem
Parallel algorithm
Synchronous method
Asynchronous method
Complementary problem
Schwarz alternating method
Constrained mechanical structures

ABSTRACT

This paper deals with the parallel solution of the stationary obstacle problem with convection–diffusion operator. The obstacle problem can be formulated by various ways and in the present study it is formulated like a multivalued problem. Another formulation by complementary problem is also considered. Appropriate discretization schemes are considered for the numerical solution on decentralised memory machines by using parallel synchronous and asynchronous Schwarz alternating algorithms. The considered discretization schemes ensure the convergence of the parallel synchronous or asynchronous Schwarz alternating methods on one hand for the solution of the multivalued problem and on the other hand for the solution of the complementary problem. Finally the implementation of the algorithms is described and the results of parallel simulations are presented.

Introduction

The behavior of mechanical structures is classically modelled by the equations governing the elasticity field [1,2]. The problem consists in studying the displacement of mechanical structure submitted to external forces. For physical reasons, the displacement of the structures may be submitted to some constraints, for example the displacement u cannot be less than a given threshold ϕ. The modelling of such constrained problem is very complex and leads to the solution of nonlinear boundary value problem, called in applied mathematics the obstacle problem. In fact this previous problem leads to the solution of a variational inequality. Variational inequalities arise from many physical and economical applications such as fluid flow in porous media, the behavior of elasto-plastic materials and lubrification phenomena in mechanics like free boundary problems and valuation of American options pricing in modern finance.

The obstacle problem has been studied in many contributions and by many authors. For example in [3] sequential methods are considered for the Hamilton–Jacobi–Bellman problem; in [4] a study concerning the rate of convergence when multilevel domain decomposition and multigrid methods are considered for sequential solution of the obstacle problem. In [5] the linear convergence is proved for sequential multiplicative Schwarz method applied to the solution of variational inequalities and in [6] a geometric convergence rate is established for sequential additive Schwarz method for the same problem; in [7] the convergence rate analysis of domain decomposition methods is also studied in the case of obstacle problem solution. In [8] the block relaxation methods for algebraic obstacle problems with M-matrices are considered while in [9] multiplicative and additive Schwarz methods for obstacle problems with convection–diffusion operators, when two (possibly overlapping) subdomains are considered, one, where the solution is equal to a given obstacle function, and the other, where the solution satisfies a linear equation. In [10] the sequential convergence analysis of the generalized Schwarz method for solving the obstacle problems with T-monotone operator is studied. In the previous contributions, note that, mainly, the study of numerical solutions is considered by using sequential subdomain methods; moreover when the parallel solution is considered for the solution of the obstacle problem, it can be noticed that only parallel synchronous subdomain methods are studied. In [11] the rate of convergence of parallel asynchronous method is studied in the context of convex optimization; the considered work includes as particular cases parallel domain decomposition and multigrid

* Corresponding author. Tel.: +33 0534322166; fax: +33 0534322157.
E-mail addresses: mchau@asa-sas.com (M. Chau), Thierry.Garcia@enseeiht.fr, thierry.garcia@uvsq.fr (T. Garcia), Pierre.Spiteri@enseeiht.fr (P. Spiteri).

http://dx.doi.org/10.1016/j.advengsoft.2014.03.005
0965-9978/

methods for solving elliptic partial differential equations and is closely related to relaxation methods for nonlinear network flow.

In the present study we will solve the model problem by the Schwarz alternating method. Thus, it is necessary to take into account the constraints and so to project the relaxed updates computed by the previous iterative algorithm on a convex set which defines such constraints. Thus, this projection has been achieved by various ways and in the proposed study we will consider two distinct ways allowing to project the relaxed updates. On other side, it can be noted that the discretization of boundary value problems arising from the formulation of the obstacle problem, leads to the solution of large scale algebraic systems. In order to compute efficiently the solution of such large algebraic systems, supercomputers can be used. The present study is devoted to the parallel solution by synchronous or asynchronous relaxation algorithms implemented on distributed architectures, mainly in Grid computing environment [12]. Recall that the asynchronous relaxation method corresponds to a general scheme of computation where the computations are performed in parallel without order nor synchronization among the processors [13]. Then idle times due to the synchronizations among the processors are suppressed; consequently, since the synchronizations are time consuming, the elapsed time of computation decreases when parallel asynchronous relaxation algorithms are performed.

In our previous works, the solution of the obstacle problem has been computed by various parallel synchronous or asynchronous algorithms implemented on various architectures. In [14–17] the parallel synchronous and asynchronous projected Richardson's method has been studied and implemented; since the projection operator is contracting, if appropriate additional assumptions hold, the convergence of the projected Richardson's method is ensured. Nevertheless, in the general model of the obstacle problem, when the gradient of the displacement u occurs, i.e. when a convection–diffusion operator models the studied problem, this previous algorithm cannot be applied directly in the context of the projected Richardson's method. Indeed the use of the Richardson's method needs a formulation of the obstacle problem as a constrained optimization problem on a convex set and is not directly suitable, since in such formulation the cost function must be defined thanks to a symmetric stationary operator. Moreover, the projected Richardson's method is not very efficient.

Besides domain decomposition methods are well suited to the parallel solution of boundary values problems. In [16,17], we have considered for the solution of the obstacle problem an iterative parallel asynchronous projected subdomain method without overlapping among the subdomains; the subdomains are here defined by gathering several adjacent blocks of the discretization matrix.

For both previous mathematical formulations, the convergence of parallel synchronous and asynchronous algorithms applied to the solution of the discretized problem, can be proved, either for every splitting of the problem to solve, by contracting techniques [18,19] or by partial ordering techniques [20,21]. The reader is referred to the previous works for more details.

The present study is based on various different formulations of the model problem taking into account the constraints on the expected solution. Indeed, we concentrate on the obstacle problem modelled with convection–diffusion operator. In a distinct mathematical context we consider in the present paper the use of parallel synchronous or asynchronous Schwarz alternating method for the solution of the obstacle problem. Recall that, in the considered Schwarz method the domain is decomposed into subdomains which overlap each other. So the present work is distinct from previous studies concerning the solution of the obstacle problem. A general approach is considered, linked to the numerical parallel solution of the obstacle problem, well adapted and very different than the ones considered from [3–11]. By using the asynchronous Schwarz alternating method we hope to obtain the more multiplicative possible behavior of the domain decomposition method; recall that the additive and the multiplicative Schwarz alternating methods may be viewed respectively as Jacobi and Gauss–Seidel type methods applied to partial differential equations.

A main property ensuring the convergence of parallel asynchronous iterative method is related to the fact that after appropriate discretization, the spatial part of the operator leads to a discretization matrix which is an M-matrix [22]. On the other hand, the projection on a convex set can be also formulated by the perturbation of the continuous convection–diffusion operator by a multivalued increasing diagonal operator [23]; so, the convergence of the considered method can also be analyzed by combining the monotony of the diagonal operator with the main property of the discretization matrix (see [19]). Another distinct theoretical approach consists in considering the obstacle problem written by a complementary formulation of this problem (which in fact corresponds to the projection of the intermediate values on the convex set which defines the constraints) and in solving the linearized algebraic system obtained by the Howard process, by the parallel asynchronous Schwarz alternating method. Note also that the present study is distinct to the one considered in [24], since in this previous work we have considered the parallel asynchronous Schwarz alternating method with flexible communications analyzed by partial ordering techniques, this last kind of communication mode among the processors which, possibly, consumes time when the rate of convergence is low; here, the convergence of the parallel asynchronous method follows from the fact that the matrix of the discretized and linearized problem is an M-matrix. Moreover, in order to obtain good efficiencies, in the presented work we consider only parallel experiments with the classical asynchronous schemes. Then we have to solve either a multivalued problem either a linearized problem by using the Howard process. So we obtain very interesting properties of discretized problems which ensure easily the convergence of parallel synchronous or asynchronous Schwarz alternating algorithms (see [18,20,21,25]).

Implementation of the considered algorithms is carried out on Grid'5000, the French national grid (see [12]). This architecture allows us to study the behavior of parallel algorithms on distant and heterogeneous clusters. Communications are managed with MPI facilities. Asynchronous and synchronous efficiencies of the studied parallel algorithms are compared. Nevertheless, in such experimental context of distant and heterogeneous clusters, the classical notions of speed-up and efficiency are not relevant. Thus, we define a new parameter which permit us to measure the real efficiency of the studied asynchronous parallel algorithms. Moreover, in the parallel experiments, we have also considered the impact of the communications on the elapsed time of the iterative parallel algorithms for both synchronous and asynchronous mode.

The present study is organized as follows. In Section 'The obstacle problem' several useful formulations of the obstacle problem are presented. In Section 'Numerical solution of the model problem' appropriate discretization schemes ensuring the convergence of the parallel synchronous or asynchronous Schwarz alternating method are considered for both formulations of the model problem. In Section 'Parallel simulation' the implementation of the algorithms and the results of the parallel experiments are presented. Finally some concluding remarks are given.

The obstacle problem

The classical obstacle problem [1–11] occurs in many applications such as mechanics, free boundary problems and financial derivatives. In the present study we concentrate mainly on the obstacle problem occurring in civil engineering. The obstacle

problem consists in computing a function u, whose characterization is given by the following time dependent nonlinear problem

$$\begin{cases} \frac{\partial u}{\partial t} - b\cdot\Delta u + c.u - f \geqslant 0, \quad u \geqslant \phi, \quad \text{everywhere in } [0,T]\times\Omega, \quad b>0, \quad c\geqslant 0, \\ \left(\frac{\partial u}{\partial t} - b\cdot\Delta u + c\cdot u - f\right)(u-\phi) = 0, \quad \text{everywhere in } [0,T]\times\Omega, \\ u(0,x,y,z) = u_0(x,y,z), \\ \text{B.C. for } u(t,x,y,z) \text{ defined on } \partial\Omega, \end{cases} \tag{1}$$

where u_0 is the initial condition, b is a physical parameter, T is the final time, $u = u(t,x,y,z)$ is the displacement of the constrained mechanical structure submitted to external strains, f is the external force imposed to the structure, B.C. describes the boundary conditions on the boundary $\partial\Omega$ of the domain Ω and ϕ models a constraint imposed to u. Practically, the Dirichlet condition (where u is fixed on $\partial\Omega$) or the Neumann condition (where the normal derivative of u is fixed on $\partial\Omega$) are classically considered.

Remark 1. When the materials constituting the structure are not homogeneous, then the coefficients arising in the mathematical model are not constant. Consequently, we can also consider the general formulation of the obstacle problem in which, after appropriate derivation with respect to the spatial variables, some terms of convection may appears. Then, we consider hereafter a simplified formulation of the obstacle problem, using a convection–diffusion operator.

$$\begin{cases} \frac{\partial u}{\partial t} + a^t\cdot\nabla u - b\cdot\Delta u + c\cdot u - f \geqslant 0, \quad u \geqslant \phi, \quad \text{everywhere in } [0,T]\times\Omega, \quad b>0, \quad c\geqslant 0, \\ \left(\frac{\partial u}{\partial t} + a^t\cdot\nabla u - b\cdot\Delta u + c\cdot u - f\right)(u-\phi) = 0, \quad \text{everywhere in } [0,T]\times\Omega, \\ u(0,x,y,z) = u_0(x,y,z), \\ \text{B.C. for } u(t,x,y,z) \text{ defined on } \partial\Omega, \end{cases} \tag{2}$$

where a is a vector describing physical parameters. Consequently, we consider two situations in which on one hand the diffusion operator appears like that in (1) and, on the other hand, the convection–diffusion operator is considered in the formulation of the problem. This last situation is more general and our study can take into account the two situations.

For the sake of generality, we consider, in what follows the more general formulation of the obstacle problem, expressed with the convection term. The previous time dependent problem is solved numerically by considering an implicit or semi-implicit time marching scheme, where at each time step the following stationary nonlinear problem is solved.

$$\begin{cases} a^t\cdot\nabla u - b\cdot\Delta u + (c+\delta)\cdot u - g \geqslant 0, \quad u \geqslant \phi, \quad \text{everywhere in } [0,T]\times\Omega, \quad b>0, \\ (a^t\cdot\nabla u - b\cdot\Delta u + (c+\delta)\cdot u - g)(u-\phi) = 0, \quad \text{everywhere in } [0,T]\times\Omega, \\ \text{B.C. for } u(t,x,y,z) \text{defined on } \partial\Omega, \end{cases} \tag{3}$$

where δ is the inverse of the time step and $g = f + \delta u^{prec}$, where u^{prec} is the solution obtained at the previous time step.

Remark 2. In Fig. 1 we can see a mechanical interpretation of the obstacle problem. Indeed, in Fig. 1(a), a thread is fixed at its own extremities and is subjected to an external force; due to their effect, the thread has a displacement submitted to some constraints. So we can see that there exists a contact area $\mathcal{O}$ in which the value of the displacement u is identical to the value of the constraint ϕ whereas the displacement is free outside of this contact area. Indeed, classically, if $a^t\cdot\nabla u - b\cdot\Delta u + (c+\delta)\cdot u - g < 0$, then $u = \phi$; on the contrary if $a^t\cdot\nabla u - b\cdot\Delta u + (c+\delta)\cdot u - g \geqslant 0$, then the constraints are not saturated and the solution of (3) is free. So, classically, due to the lack of constraints, in this last case where $u > \phi$ and the operator $a^t\cdot\nabla u - b\cdot\Delta u + (c+\delta)\cdot u - g$ is null; thus the physical problem is well described. From an industrial point of view, the Fig. 1(b) shows an example of application concerning the study of crash test arising in the self-propelling construction.

For each stationary problem many equivalent formulations of the obstacle problem exist. In the present study, we concentrate on the variational formulation associated with problem (3); classically the corresponding variational formulation is given as follows

$$\begin{cases} \text{Find } u \in K \text{ such that} \\ a(u, w-u) \geqslant L(w-u), \quad \forall w \in K, \end{cases} \tag{4}$$

where

$$a(u,w) = \int_\Omega (b\cdot\nabla u\cdot\nabla w + a^t\cdot\nabla u\cdot w + (c+\delta)\cdot u\cdot w)\,dx\,dy\,dz,$$

$$L(w) = (f,w) = \int_\Omega f\cdot w\,dx\,dy\,dz,$$

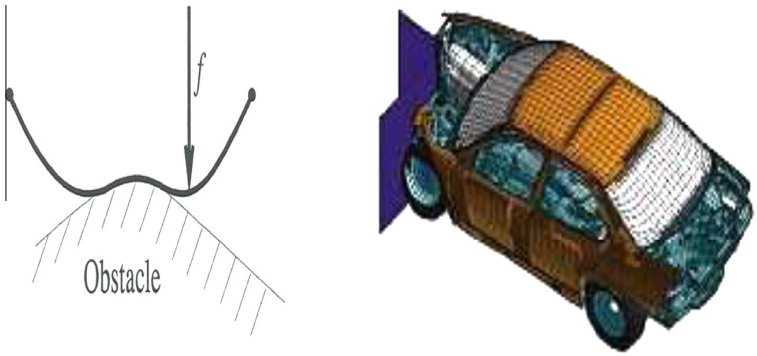
and

$$K = \{w | w \text{ given in } \Omega, \text{ such that } w(x,y,z) \geqslant \phi(x,y,z) \text{ everywhere on } \Omega\}.$$

Let us consider the stationary variational inequality (4). Then, since classically

1. the space $\mathcal{H}^1(\Omega)$ normed by $\|w\|^2_{1,\Omega} = \int_\Omega (\nabla w\cdot\nabla w + w\cdot w)\,dx\,dy\,dz$, is an Hilbert space and K is a closed convex set,
2. the mapping $(u,w) \to a(u,w)$ is obviously a bilinear, continuous and elliptic form,
3. the mapping $w \to L(w)$ is obviously a linear continuous form,

then, by applying the Stampacchia theorem, the variational inequality (4) has a unique solution.



(a) displacement of a thread (b) crash test

Fig. 1. Mechanical interpretation of the obstacle problem for the constrained displacement of a thread (a) and of a crash-building of a self-propulsing vehicle (b).

In the case of the evolution problem (1) or (2), the reader is referred to [26] for a result of existence and uniqueness of the solution.

We can also apply the Riesz representative theorem. Let us denote by $E = \mathcal{H}^1(\Omega)$ the real vector space and by E' its dual space; in our case E' is identified with E, since $\mathcal{H}^1(\Omega)$ is an Hilbert space. Then, there exists a unique element denoted by $\overline{A}u \in E'$, where $\overline{A}$ is a continuous linear mapping from $E \to E'$, such that

$$a(u,w) = \langle w, \overline{A}u\rangle_{E\times E'}, \quad \forall w \in \mathcal{H}^1(\Omega);$$

similarly, there exists also a unique element, denoted by $\bar{g} \in E'$ such that

$$L(w) = \langle w, \bar{g}\rangle_{E\times E'}, \quad \forall w \in \mathcal{H}^1(\Omega);$$

then the stationary variational inequality (4) can be written as follows

$$\begin{cases} \text{Find } u \in K \text{ such that} \\ \langle w-u, \overline{A}u - \bar{g}\rangle_{E\times E'} \geqslant 0, \quad \forall w \in K. \end{cases} \tag{5}$$

Remark 3. From a practical point of view, note that for the studied stationary problem (4) $\overline{A}u(x,y,z) = -b\cdot\Delta u(x,y,z) + a^t\cdot\nabla u + (c+\delta)\cdot u(x,y,z)$ and $\bar{g} = g(x,y,z)$.

Sub-differential mapping

The notion of sub-differential mapping will play a major role in the following study. So we recall hereafter this notion and the main properties associated (see [1,23]).

Definition 1. Let be χ a given convex function on E and a point $u \in E$; we denote by $\partial\chi(u)$ the set of all $u' \in E'$ such that

$$\chi(v) \geqslant \chi(u) + \langle v-u, u'\rangle_{E\times E'}, \quad \text{for every } v \in E, \tag{6}$$

where $\langle,\rangle_{E\times E'}$ denotes the pairing between E and E'. Such element u' is called sub-gradient of χ at u, and $\partial\chi(u)$ is called the sub-differential of χ at u.

Remark 4. Recall that the pairing between E and E' is a bilinear form, from $E\times E'$ onto $\mathbb{R}$ (or possibly $\mathbb{C}$). If E is an Hilbert space, then the pairing is the inner product of E.

Remark 5. Let χ be a Gateaux differentiable (or Frechet differentiable) convex mapping at u. Then $\partial\chi(u)$ consists of a single element, namely the Gateaux (or Frechet) differential of χ at u (see [23]). From (6), it is obvious that $\partial\chi(u)$ is a closed convex set (possibly empty, see [23]).

In the sequel, we will use a multivalued formulation of the model problem (4) which playes a major role for the studied nonlinear obstacle problem.

Lemma 1. *Let $u \in E$ be such that $\chi(u) = \min_{v\in E}(\chi(v))$; then u is a minimum if and only if $0 \in \partial\chi(u)$.*

Proof. Indeed, let $u \in E$ such that $\chi(u) \leqslant \chi(v)$, $\forall v \in E$; then we have

$$\chi(v) \geqslant \chi(u) + \langle v-u, 0\rangle_{E\times E'},$$

and then $0 \in \partial\chi(u)$. □

Lemma 2. *The sub-differential $\partial\chi(u)$ is a monotone operator (in general multivalued) from E to E'.*

Proof. Let $w' \in \partial\chi(w)$; then $\chi(v) \geqslant \chi(w) + \langle v-w, w'\rangle_{E\times E'}$, $\forall v \in E$. Let also $u' \in \partial\chi(u)$; then $\chi(v) \geqslant \chi(u) + \langle v-u, u'\rangle_{E\times E'}$, $\forall v \in E$. Let us consider the first inequality for $v = u$ and the second for $v = w$; then by adding, we obtain $\langle w-u, w'-u'\rangle_{E\times E'} \geqslant 0$, and the proof is achieved. □

The indicator function of the convex subset K will also play an important role in the sequel and is defined as follows.

Definition 2. Let K be a closed convex subset of E. The indicator function of the convex subset K denoted ψ_K is then defined by

$$\psi_K(v) = \begin{cases} 0 & \text{if } v \in K \\ +\infty & \text{otherwise.} \end{cases}$$

Clearly, $\psi_K(v)$ is convex. By definition of the sub-differential we have (see [23])

$$\partial\psi_K(v) = \{v' \in E' | \langle v-w, v'\rangle_{E\times E'} \geqslant 0, \text{ for every } w \in K\}.$$

This shows that the domain of the subdifferential is $D(\partial\psi_K) = D(\psi_K) = K$ and $\partial\psi_K(v) = \{o\}$ for each $v \in int(K)$. Moreover, if v lies on the boundary of K, then $\partial\psi_K(v)$ coincides with the cone of normal to K at point v. So we can summarize by giving bellow the expression of the sub-differential indicator function of the convex subset K

$$\partial\psi_K(v) = \begin{cases} \emptyset & \text{if } v < \phi, \\ ]-\infty, 0] & \text{if } v = \phi, \\ 0 & \text{if } v \in K, \end{cases}$$

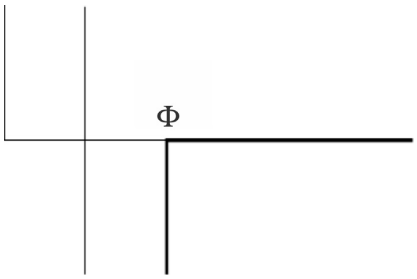


Fig. 2. Graph of the sub-differential of the indicator function.

and the corresponding graph presented on the following Fig. 2.

Equivalent formulation of the obstacle problem

Then, thanks to the notions presented in the previous subsection, the formulation (5) of the continuous stationary variational inequality (4) is equivalent to the following multivalued problem

$$\begin{cases} \text{Find } u \in E \text{ such that} \\ \overline{A}u - \bar{g} + \partial\psi_K(u) \ni 0, \end{cases} \tag{7}$$

where $\partial\psi_K(u)$ is the sub-differential of the indicator function of the convex subset K. Note that, in (7), the projection on the convex set K can be also formulated by the perturbation of the continuous convection–diffusion operator by a multivalued increasing diagonal operator. Indeed, the definition of the sub-differential of the indicator function implies

$$\partial\psi_K(u) = \{\omega' \in E' | \psi_K(w) - \psi_K(u) \geqslant \langle w - u, \omega' \rangle_{E\times E'},\ \forall w \in K\}.$$

Let us assume that (5) holds and let us verify that the following inequality is true

$$\psi_K(w) - \psi_K(u) \geqslant \langle w - u, \bar{g} - \overline{A}u \rangle_{E\times E'}, \quad \forall w \in K. \tag{8}$$

If $w \in K$, then $\psi_K(w) = 0$ and since $\psi_K(u) = 0$, then we obtain

$$0 \geqslant \langle w - u, \bar{g} - \overline{A}u \rangle_{E\times E'},$$

which is valid by considering (5).

If $w \notin K$, then $\psi_K(w) = +\infty$; since, $\psi_K(u) = 0$, then we obtain

$$+\infty - 0 \geqslant \langle w - u, \bar{g} - \overline{A}u \rangle_{E\times E'},$$

and the obtained inequality is always true.

Conversely, consider two distinct situations:

1. if there exists $u \in E$ such that $\bar{g} - \overline{A}u \in \partial\psi_K(u)$ then we have to verify that there exists $u \in E$ such that (8) holds for all $w \in K$. If $u \in K$ then $\psi_K(u) = 0$ and (8) can be written as follows

   $$\psi_K(w) \geqslant \langle w - u, \bar{g} - \overline{A}u \rangle_{E\times E'}, \quad \forall w \in K.$$

   Since $w \in K$ then $\psi_K(w) = 0$ and (8) can be written as follows

   $$0 \geqslant \langle w - u, \bar{g} - \overline{A}u \rangle_{E\times E'}, \quad \forall w \in K,$$

   inequality well verified by (5);
2. besides, let us show that the assertion $u \notin K$ is impossible. Indeed, in this case $\psi_K(u) = +\infty$, so that

   $$\psi_K(w) - \infty \geqslant \langle w - u, \bar{g} - \overline{A}u \rangle_{E\times E'}, \quad \forall w \in K,$$

   which involves, since $\psi_K(w) = 0$, that

   $$-\infty \geqslant \langle w - u, \bar{g} - \overline{A}u \rangle_{E\times E'}, \quad \forall w \in K,$$

inequality never verified.

In conclusion, the problems (7) and (8) are equivalent. So the numerical solution of the obstacle problem leads to the solution of the multivalued problem (7).

Numerical solution of the model problem

Discretization

Let us consider the classical stationary linear convection–diffusion operator similar to the one arising in (2) and defined in a bounded domain Ω included in the 3D space. For the sake of simplicity, let us assume that the discretization grid of the domain Ω is uniform. In the sequel h will denote the discretization step-size. The discretization of the operators is made according to the following rules: the Laplacian is discretized with the classical seven points scheme; the first derivatives, for example the derivative with respect to x, is discretized as follows according to the sign of the component a_1 of the vector a

$$\frac{\partial u}{\partial x} = \begin{cases} \frac{u(x,y,z)-u(x-h,y,z)}{h} + \mathcal{O}(h), & \text{if the coefficient } a_1 \text{ is strictly positive,} \\ \frac{u(x+h,y,z)-u(x,y,z)}{h} + \mathcal{O}(h), & \text{if the coefficient } a_1 \text{ is strictly negative,} \end{cases} \tag{9}$$

and accordingly for the other first derivatives.

Let A denote the discretization matrix of the considered convection–diffusion operator; let us also denote by $\overline{G}$ the corresponding right hand side of the discretized system. Since $c + \delta$ is strictly positive, then regardless the sign of the components of the vector a, it follows from (9) that the off-diagonal entries of the matrix A are nonpositive and the diagonal entries of the matrix A are positive. Thus, the matrix A is strictly (and irreducibly by using the characterization of irreducible matrices (see [22])) diagonally dominant; thus, A is a nonsingular M-matrix and $A^{-1} \geqslant 0$ i.e. all the entries of the inverse of the matrix A are nonnegative (see [22]).

Let us now consider the discretization of the global problem (7); taking into account the nonlinear part of the problem due to the perturbation of the linear algebraic system by the multivalued sub-differential operator $\partial\Psi_K$ of the indicator function Ψ_K, then the discretized problem can be written as follows

$$AU - G + \partial\Psi_K(U) \ni 0, \tag{10}$$

where $\partial\Psi_K(U)$ is the discretized sub-differential operator of the indicator function.

Remark 6. Since the sub-differential is a diagonal operator, the components of $\partial\Psi_K(U)$ are constituted by all the discretized sub-gradients belonging to the sub-differential of the indicator function.

The Schwarz alternating method

The use of a purely implicit time marching scheme and the considered discretization of the spatial part of the operators lead, at each time step, to the numerical solution of very large algebraic systems. Owing to the great size and also of the sparsity of such a system, iterative algorithm may be used in order to reduce the computation time at each time step. Moreover the parallelization of such iterative algorithm can reduce again the elapsed time of computation.

In the present study, we concentrate on the implementation of the iterative Schwarz alternating algorithm for the solution of the stationary obstacle problem. In such method, at each time step, we consider the classical projected Schwarz alternating method applied to the solution of each stationary variational inequality.

Note that the Schwarz alternating algorithm with overlapping is well suited for efficient parallel computation (see [27]). Note also that we can consider parallel synchronous and asynchronous Schwarz alternating algorithms. The interest of asynchronous algorithm, compared to the synchronous one, is the reduction of idle times due to synchronizations among the processors.

In this kind of method, in order to parallelize the computation, the domain Ω, is split into parallelepiped subdomains which overlap each other. Thus, for the computation of the solution of the global problem, smaller subproblems are solved on each processor of a parallel computer and then more accuracy can be obtained since the problem to be solved is ill-conditioned. For a considered subdomain, the restriction of the available values of the computed solution obtained on the neighboring subdomains are taken as Dirichlet boundary conditions. The synchronous algorithm consists in performing a block Jacobi like method. Besides, in the classical asynchronous algorithm, the components computed by the other processors may be delayed (see [18,20,25]) and the available values are used at the end of each relaxation.

More precisely, let us consider the system of algebraic Eq. (10). The numerical solution of (10) by the Schwarz alternating method leads to the solution of the following multivalued problem

$$\mathcal{A}\cdot\mathcal{U}-\mathcal{G}+\partial\overline{\Psi}_K(\mathcal{U})\ni 0, \tag{11}$$

where $\mathcal{A}$, $\mathcal{V}$, $\mathcal{G}$ and $\overline{\Psi}_K$ are derived from the augmentation process associated with the Schwarz alternating method [28]. This process is a theoretical model that represents the solution of the algebraic system (10) by a Schwarz domain decomposition method. In the implementation of the algorithms $\mathcal{A}$, $\mathcal{G}$ and $\overline{\Psi}_K$ are not explicitly computed. According to a result of Evans and Deren [28], the matrix $\mathcal{A}$ is also an M-matrix. So, the system (11) derived from the augmentation process, has the same property as the initial algebraic system (10), i.e. $\mathcal{A}$ is an M-matrix and $\partial\overline{\Psi}_K(\mathcal{U})$ is a monotone operator.

Let us consider the system of algebraic Eq. (10) to solve. Let $\alpha\in\mathbb{N}$ be a positive integer and consider now the following block decomposition of problem (11) into α subproblems associated to the subdomain decomposition

$$\sum_{j=1}^{\alpha}\mathcal{A}_{ij}\cdot\mathcal{U}_j+\partial\overline{\Psi}_{K_i}(\mathcal{U}_i)-\mathcal{G}_i\ni 0,\quad \forall i\in\{1,\dots,\alpha\} \tag{12}$$

where $\mathcal{U}_i\in\mathbb{R}^{n_i}$, $\mathcal{G}_i\in\mathbb{R}^{n_i}$ and $\partial\overline{\Psi}_{K_i}(\mathcal{U}_i)$ is the i-th block of the sub-differential mapping, where n_i denotes the size of the i-th block of the previous vectors and $\mathcal{A}=(\mathcal{A}_{ij})_{1\leqslant i,j,\leqslant\alpha}$, is defined according to the associated block decomposition.

Let us now consider the solution of subproblems (12) by the classical asynchronous parallel iteration which can be written as follows

$$\begin{cases}\mathcal{A}_{ii}\cdot\mathcal{U}_i^{r+1}+\partial\overline{\Psi}_{K_i}(\mathcal{U}_i^{r+1})\ni\mathcal{G}_i-\displaystyle\sum_{j\neq i}\mathcal{A}_{ij}\cdot\mathcal{W}_j, & \text{if } i\in s(r),\\ \mathcal{U}_i^{r+1}=\mathcal{U}_i^{r}, & \text{if } i\notin s(r),\end{cases} \tag{13}$$

where $\{\mathcal{W}_1,\mathcal{W}_2,\dots,\mathcal{W}_\alpha\}$ are the available values of the components $(\mathcal{U}_j)_{j\neq i}$, computed by the other processors and corresponding to an iterative method where the updates are performed at the end of any block relaxation by an asynchronous way, each component being defined by $\mathcal{W}_j=\mathcal{U}_j^{\rho_j(r)}$ and $\mathcal{S}=\{s(r)\}_{r\in\mathbb{N}}$ is a sequence of non-empty subsets of $\{1,2,\dots,\alpha\}$. In other words, $s(r)$ is the subset of the subscripts of the components updated at the r-th relaxation. In the sequel, we have also to consider the vector

$$\mathcal{R}=\{\rho_1(r),\dots,\rho_\alpha(r)\}_{r\in\mathbb{N}},$$

which denotes a sequence of integer vectors from $\mathbb{N}^\alpha$. In order to take into account the asynchronism among the processors, $\mathcal{R}$ models the delays between the parallel updates of each component, at the r-th relaxation. Furthermore $\mathcal{S}$ and $\mathcal{R}$ satisfy the following assumptions

$$\forall i\in\{1,2,\dots,\alpha\},\quad \text{the set } \{r\in\mathbb{N}|i\in s(r)\} \text{ is unbounded},$$

$$\forall i\in\{1,2,\dots,\alpha\},\qquad \forall r\in\mathbb{N},\quad 0\leqslant\rho_i(r)\leqslant r,$$

$$\forall i\in\{1,2,\dots,\alpha\},\quad \lim_{r\to\infty}\rho_i(r)=+\infty.$$

Remark 7. Classically, when $\rho_i(r)=r$ for all $i\in\{1,\dots,\alpha\}$ and for all $r\in\mathbb{N}$, then (13) models a synchronous Schwarz alternating method.

This formulation (10)–(12) by a multivalued sub-problem (12) takes some interest only from a theoritical point of view. Indeed, since the sub-differential of the indicator function is monotone and diagonal, the previous formulation allows to prove the convergence of the parallel synchronous and asynchrounous iterations [18–21].

Recall that the matrix arising in the algebraic linear system, is an M-matrix; the corresponding algebraic linear system is perturbed by a monotone diagonal operator. Then, for any subdomain decomposition, according to theoretical results obtained in [18,19,25], since it can be proved that the uniform weighted norm of the error at the step r is decreasing, the numerical solution of problem (10) by the classical parallel asynchronous Schwarz alternating method associated with (13) and starting from any initial guess U^0, converges to the solution of the obstacle problem. Reference is made to these three last papers for more details concerning these results. Note also that the same result can also be obtained by partial ordering analysis (see [20,21]), since the mapping $AU-G+\partial\Psi_K(U)$ is obtained by the perturbation of a linear system with an M-matrix by a monotone diagonal operator, and then is an M-function, i.e. a mapping off-diagonally antitone and inverse isotone (see [22]).

Sometimes, the implementation of parallel algorithms requires the use of lexicographical ordering or red–black ordering of the columns of the mesh. The previous convergence results still hold, in both cases of natural and red–black ordering of the subdomains. Let us denote by A and $\overline{A}$ the corresponding matrices associated with the two considered orderings. Let us assume that if A is an M-matrix, then $\overline{A}$ is also an M-matrix since $\overline{A}$ is obtained from A by a permutation matrix P which preserves the sign of the entries and particularly the sign of the diagonal entries. Furthermore we have $\overline{A}=P\cdot A\cdot P^t$; thus we have $\overline{A}^{-1}=P\cdot A^{-1}\cdot P^t$. The matrix A being an M-matrix, it follows that A^{-1} is a nonnegative matrix [22], i.e. the entries of A^{-1} are nonnegative. Thus $\overline{A}^{-1}$ is also a nonnegative matrix obtained from A^{-1} by the same permutation as the one considered for A. Then $\overline{A}$ is an M-matrix. Moreover when a red–black ordering of the blocks is considered, since the sub-differential mapping is a diagonal operator, this is again a monotone mapping. This remark proves that the asynchronous parallel methods converge also in the context of the red–black ordering of the subdomains.

Remark 8. For the solution of system (10), we can consider instead of the classical parallel asynchronous Schwarz alternating method, the parallel asynchronous Schwarz alternating method with flexible communications which corresponds to a more general model of parallel asynchronous iterations. This kind of method consists in using partial ordering techniques linked with the discrete maximum principle [21]. In the present context, the values of the

components of the iterate vector generated by the other process, can be accessed while the computations are still in progress. In such method the current value of any component of the iterate vector is not necessarily labelled by an outer iteration number as communication may occur at any time. Then, in this class of methods, partial updates can be used at any time in the computation. Thus, flexible data exchanges among processors are allowed; as a consequence, the coupling between communications and computations can be improved. This method consists in starting the iteration (13) with an initial guess U^0 such that

$$A \cdot U^0 + \Lambda(U^0) - G \geqslant 0, \quad \Lambda(U^0) \in \partial\overline{\Psi}_K(\mathcal{U}^0). \tag{14}$$

In practice U^0 is chosen such that $U^0 = \Phi$. In this case, for $r \geqslant 1$, the vector $\mathcal{W} = \{\mathcal{W}_1, \mathcal{W}_2, \ldots, \mathcal{W}_\alpha\}$ belongs to the order interval $\langle \mathcal{U}^r, \min(\mathcal{U}^{\rho(r)}, \mathcal{U}^q)\rangle$, where $\mathcal{U}^{\rho(r)}$ denotes the vector with block components $\mathcal{U}_j^{\rho_j(r)}$, $j \in \{1, 2, \ldots, \alpha\}$ and $q = \max_{k \in K^r_{s(r)}}(k)$, where the set K_i^r contains all the iteration numbers lower than r associated with the computation of the i^{th} block component (see [21]). Note that the values of the components $\mathcal{W}_j$ are also the available values of the iterate vector. In this case, if (14) is satisfied, it can be proved (see [21]) that the sequence of iterate vectors satisfies the maximum discrete principle; indeed $(U^r)_{r\geqslant 0}$ satisfies the following inequalities $U \leqslant \ldots \leqslant U^r \leqslant \ldots \leqslant U^0$ (see also [29] for an analysis by using contraction technics).

In conclusion for the first way of projection, since the discretization scheme applied to the numerical approximation of the boundary value problem modelling the obstacle problem, leads to the solution of an algebraic system constituted by an M-matrix perturbed by a monotone diagonal operator, then, the parallel synchronous and asynchronous alternating methods with or without flexible communications converge to the solution of the considered discretized boundary value problem for any initial guess U^0 and for any ordering of the subdomains. From a practical point of view, concerning the implementation associated to the model problem, we have:

1. first to compute an intermediate value of a component of the iterate vector by solving for example, one equation of the algebraic linear system,
2. and then to project this intermediate value of the component on the convex set; more precisely, if the constraint is satisfied, the intermediate value of the component is not changed and if the constraint is saturated, for example, if the component of U is less than the component of Φ ($U < \Phi$), then the intermediate value of the component is equal to Φ.

Linearization of the problem by the Howard method

Note also that the projection on the discretized convex set can be performed by using the Howard method; such mathematical approach avoids using the notion of sub-differential mapping. In fact this second method is a Newton like method allowing the linearization of the stationary problem associated with (3). Indeed, after appropriate discretization we have to solve at each time step

$$\max(AV - G, V - \Phi) = 0$$

and we associate it with the linearized system

$$B(V) \cdot V = C(V), \tag{15}$$

constructed as follows:

1. if the i-th component of $(AV - G)$ is greater than the i-th component of $(V - \Phi)$, then the i-th line of $B(V)$ is the i-th line of A and the i-th component of $C(V)$ is the i-th component of G,
2. else all the entries of the i-th line of $B(V)$ are null except the diagonal entry equal to one, and the i-th component of $C(V)$ is the i-th component of Φ.

Since the discretization matrix A is strictly (and irreducibly) diagonally dominant and regardless the sign of the entries of the matrix A, obviously the matrix $B(V)$ is strictly diagonally dominant. Note that, the matrix $B(V)$ is not necessarily irreducible, since in the linearization process some entries equal to zero can appear in all the off-diagonal entries of a same line; this happens when the previous case 2) occurs during the building of the linearized system. Nevertheless, at each step of the linearization process, the global matrix $B(V)$ has strictly positive diagonal entries and nonpositive off-diagonal entries; furthermore $B(V)$ is strictly diagonally dominant. Consequently, $B(V)$ is an M-matrix and at each step of the Howard method we have to solve a linear algebraic system where the matrix to be inverted is an M-matrix. So, for solving such linear system, we can use the parallel synchronous or asynchronous Schwarz alternating method described in subSection 'The Schwarz alternating method', and, due to the property of $B(V)$, this last algorithm converges for each step of the linearization process. We refer to [24] for a direct convergence analysis of the parallel asynchronous Howard algorithm by partial ordering analysis, including parallel asynchronous method with flexible communications.

Remark 9. It can be noted that the Howard method can be used for a more general class of complementary problems. Indeed consider the Hamilton–Jacobi–Bellman equation arising for example in image processing. Such stationary problem can be formulated as follows

$$\begin{cases} \text{Find } v \in E \text{ such that} \\ \sup(\overline{A}_1 v - f_1, \overline{A}_2 v - f_2) = 0, \quad \text{everywhere in } \Omega, \\ v = 0, \quad \text{everywhere on } \partial\Omega, \end{cases} \tag{16}$$

where $\overline{A}_i,\ i = 1, 2$ are two elliptic operators; after appropriate discretization we have to solve the discrete complementary problem

$$\max(A_1 V - F_1, A_2 V - F_2) = 0, \tag{17}$$

where $A_i,\ i = 1, 2$ are the discretization matrices. Then, for the previous problem, we can consider an associated linearized problem obtained by the Howard method. The linearization is the same as the one considered in (15), except the identity matrix is replaced by the matrix A_2 and Φ is replaced by F_2 in the second argument. Now, if $A_i,\ i = 1, 2$ are two M-matrices, the matrix $B(V)$ arising in the linearized system is also an M-matrix and consequently, the parallel synchronous or asynchronous Schwarz alternating methods converge.

In conclusion, in the proposed study, we present two distincts ways for achieving the projection on the convex set. The first way consists classically to perturbe the linear algebraic systems by adding the sub-differential of the indicator function. Then we obtain a multivalued formulation of the problem to solve. This way corresponds to a theoretical way and allows to analyze the convergence of the parallel synchronous and asynchronous subdomain methods. In the implementation of the algorithm, we have to project the intermediate value of the iterate vector on the convex set as explained below.

The second way is distinct to the first one and is more global. Indeed we have to linearize the problem by the Howard process. Then we obtain a linearized system (15), and the convergence of the parallel synchronous and asynchronous subdomain methods follows from the fact that the matrix $B(V)$ issued from the Howard

process is an M-matrix. Then, the proposed paper presents two methods for the solution of the obstacle problem.

Parallel simulation

This section is devoted to the presentation of the parallel algorithms implementation and the results of parallel experiments.

Principle of implementation

The simulation algorithm is based on master/slave paradigm which is a commonly used approach for parallel and distributed applications. In our case, a master process controls the distribution of work to a set of slave processes.

Algorithm 1. General algorithm

```
Algorithm
  if (master)
    compute discretization matrix and right hand side
    send input data to slaves
    compute SPMD parallel algorithm to solve the algebraic
  system
    receive output data from slaves
  else
    receive input data from master
    compute SPMD parallel algorithm to solve the algebraic
  system
    send output data to master
  end if
```

This process is used for the solution of the system that arises in the numerical simulation of the obstacle problem. Matrix and right hand side creation have been implemented sequentially since this part of computation is not very intensive. In this way, the master process can be seen as a matrix and vector filler that feeds a parallel solver. In other words, only intensive computations have been parallelized with MPI facilities.[1] Note that only the parallel synchronous and classical asynchronous Schwarz alternating methods have been implemented; the asynchronous Schwarz alternating method with flexible communications has not been considered in the present study. In addition, the principle of parallel asynchronous iterative subdomain algorithms implementation for on one hand the Schwarz method for the multivalued problem and on the other hand for the Schwarz method with Howard linearization can be summarized as follows:

Algorithm 2. Schwarz for multivalued problem

```
do until global convergence
  for each subdomain assigned to the processor do
   if local convergence is not reached then
     receive the latest boundary values
     solve with projection the subsystems
     send the boundary values to the neighbors
   end if
  end for
end do
```

[1] Message Passing Interface.

Algorithm 3. Schwarz with Howard linearization

```
do until global convergence
  for each subdomain assigned to the processor do
   if local convergence is not reached then
     receive the latest boundary values
     solve the subsystems:
      – compute matrix and right hand side according to
        Section 'Linearization of the problem by the Howard
        method'
      – solve the linearized subsystem
     send the boundary values to the neighbors
   end if
  end for
end do
```

Subdomains are assigned to each processor following a red–black ordering. For the solution of the obstacle problem, note that in each subdomain we have considered either the implementation of the block Gauss–Seidel method with projection of the iterate vector on the convex set or the block Gauss–Seidel method applied to the solution of the linearized problem by the Howard process. For the first considered method, this kind of algorithm is faster than a point relaxation method with projection.

Parallel experiments

Parallel simulations for the solution of the obstacle problem have been performed on various distant clusters of the Grid'5000 platform. This French grid platform was composed of 2970 processors with a total 6906 cores distributed over 9 sites in France. In this network every site hosts clusters and all sites are connected by highspeed communication; nowadays the communication network is Gigabit Ethernet for local machines. Bandwidth among the different sites is about 10 Gbps. Most of Grid'5000 sites are composed of several clusters with different computing architectures and performances. Fig. 3 illustrates the sites of the Grid'5000 architecture.

The parallel experiments have been performed using an heterogeneous and dedicated machines running Linux 64 bits. The machines of the Grid'5000 platform used are located in Sophia-Antipolis, Rennes, Nancy and Toulouse sites. The characteristics of these machines are summarized in Table 1.

In the parallel experiments we have only considered the performances obtained by computing on one time step which in fact corresponds to the solution of a stationary obstacle problem. Indeed in a previous study [16,17] where a time-dependent obstacle problem has been solved by using projected Richardson method and subdomain without overlapping algorithm, we have shown that all time steps require similar execution time for the solution of each stationary problem. Thus, experiments with Schwarz alternating methods allow us to estimate the performances of the considered parallel algorithm for the solution of time dependent problem.

In the parallel experiments, we consider a benchmark where a regular mesh is defined on $\Omega = [0,1] \times [0,1] \times [0,1]$; in numerical simulations, we have considered two meshes constituted on one hand of 64,000,000 mesh points[2] and on other hand of 32,768,000 mesh points[3]; due to utilization constraints of the grid platform in dedicated exploitation, the number of mesh points is limited to the previous values; for the same reason the number of cores is also

[2] $400 \times 400 \times 400 = 64,000,000$ points.
[3] $320 \times 320 \times 320 = 32,768,000$ points.

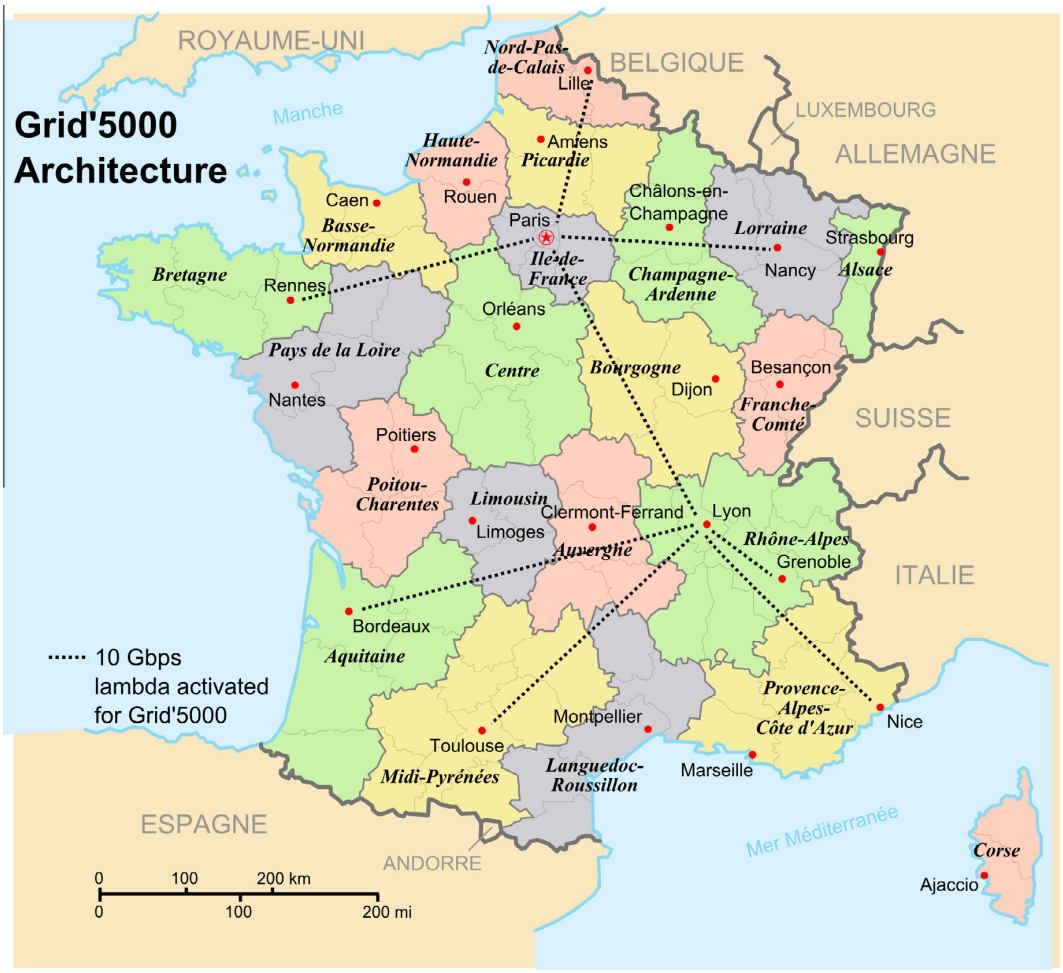


Fig. 3. The Grid'5000 architecture.

Table 1
Characteristics of machines on each site.

Site	Cluster	Processors type	Speed (GHz)	cpu	Core	RAM size (GB)
Rennes	Paradent	Intel Quad Xeon L5420	2.5	2	4	32
Sophia	Suno	Intel Quad Xeon E5520	2.26	2	4	32
Toulouse	Pastel	AMD Opteron 2218	2.6	2	2	8
Nancy	Griffon	Intel Quad Xeon L5420	2.5	2	4	16

limited. The diffusion coefficient is set to $b = 0.2$ and the components of the convection vector are equal to -0.9; furthermore $c = 0$ and $\phi = 0$. The domain is decomposed into 256 overlapping subdomains numbered using a red–black ordering, well adapted to parallel computation. In the implementation of the Schwarz alternating methods, two or more subdomains are assigned to each processor; thus, on each processor, this assignation allows a behavior of the parallel algorithm as close as possible of a multiplicative Schwarz alternating method. From a practical point of view, parallel synchronous and asynchronous algorithms are more efficient on such subdomain distribution.

The numerical algorithms have been implemented using Fortran 90, BLAS subroutines, no other external libraries have been used; the remainder of the implementation is coded by ourself. In particular, a block relaxation method has been implemented for the solution of the obstacle problem. The parallelization is achieved using MPI facilities. Message passing are implemented with MPI persistant communication requests, in both synchronous and asynchronous cases. Thus, the proposed MPI implementation of the Schwarz alternating method in decentralized memory multiprocessor is more efficient. The implementations of message passing are described in Algorithms 4 and 5.

Algorithm 4. Implementation of synchronous message passing using MPI facilities.

Synchronous message passing:

```
!!!!!!!!!!!!!!!!
! SEND REQUESTS!
!!!!!!!!!!!!!!!!
! Wait for the completion of the former send
  requests
call MPI_WAITALL (mlpt%NSEND,sndt,starray,ierr)
! Pack the current messages
do i = 1, mlpt
  badr = mlpt%SND_BT (i)
  call MSGPK (PK_P,mlpt%SND_HD (i),
  handl%LOCFV,X,buf (badr))
end do
! Start sending messages
call MPI_WAITALL (mlpt%NRECV,rcvt,starray,ierr)
!!!!!!!!!!!!!!!!!!!
! RECEIVE REQUESTS!
!!!!!!!!!!!!!!!!!!!
! Wait for the completion of the former receive
  requests
call MPI_WAITALL (mlpt%NRECV,rcvt,starray,ierr)
! Unpack the current messages
do i = 1, mlpt%NRECV
  badr = mlpt%RCV_BT (i)
  call MSGPK (PK_U,mlpt%RCV_HD
  (i),handl%LOCFV,X,buf (badr))
end do
! Start receiving messages
call MPI_STARTALL (mlpt%NRECV,rcvt,ierr)
```

Algorithm 5. Implementation of asynchronous message passing using MPI facilities.

Asynchronous message passing:

```
!!!!!!!!!!!!!!!!
! SEND REQUESTS!
!!!!!!!!!!!!!!!!
! Nonblocking check for the completion
! of the former send requests
call MPI_TESTSOME
  (mlpt%NSEND,sndt,nout,outarray,starray,ierr)
do i = 1, nout
  ! Pack the current messages for completed
  requests only
  mpos = outarray (i)
  badr = mlpt%SND_BT (mpos)
  call MSGPK (PK_P,mlpt%SND_HD
  (mpos),handl%LOCFV,X,buf (badr))
  ! Start sending messages
  call MPI_START (sndt (mpos),ierr)
end do
!!!!!!!!!!!!!!!!!!!
! RECEIVE REQUESTS!
!!!!!!!!!!!!!!!!!!!
! Nonblocking check for the completion
! of the former receive requests
call MPI_TESTSOME
  (mlpt%NRECV,rcvt,nout,outarray,starray,ierr)
do i = 1, nout
  ! Unpack the current messages for completed
  requests only
  mpos = outarray (i)
  badr = mlpt%RCV_BT (mpos)
  call MSGPK (PK_U,mlpt%RCV_HD
  (mpos),handl%LOCFV,X,buf (badr))
  ! Start receiving messages
  call MPI_START (rcvt (mpos),ierr)
end do
```

For all considered decomposition of the problem to solve, convergence of the proposed sequential and parallel algorithms is ensured. Concerning the rate of convergence, when the convergence analysis is done by contraction technics, an estimate of the rate of convergence is obtained by considering on one hand the spectral radius of the Jacobi matrix associated with the discretization matrix A and on the other hand, the associated subdomain decomposition (see [19]).

Many experiments have been performed to exhibit elapsed times of parallel Schwarz alternating method in both synchronous and asynchronous iterative schemes and using two or three distant sites. The latter case corresponds to an heterogeneous architecture since computing processors are different; indeed the machines located in Sophia-Antipolis and Nancy used Intel processors and the machines located in Toulouse used AMD processors.

In the parallel simulations only one core of each machine could be used. Indeed the amount of data to be stored in each RAM memory, for each MPI process, is very large; consequently very large memory size is needed, in fact greater than 4 GB. On other hand, the advantage of using 64 bits machines and compilers is the capability to use RAM with size greater than 4 GB. Thus, in the case where many cores per machine are used, due to memory swapping, the performances of parallel algorithms will decrease. Note that, due to the size of the algebraic systems to solve, only machines having large memory could be used.

The results of the experiments are summarized in Tables 2–7 and Figs. 4–9, for the obstacle problem formulated by a multivalued algebraic system [30] and for the linearized and discretized obstacle problem, called in the sequel Schwarz–Howard method. In the following tables, P denotes the number of processors.

In fact, the two considered formulations of the discretized obstacle problem lead to similar performances of the associated

Table 2
Elapsed time (seconds) and number of relaxations for sequential algorithm with Schwarz alternating method for the multivalued problem on cluster suno (Sophia), paradent (Rennes), griffon (Nancy) and pastel (Toulouse).

32,768,000 mesh points					
suno		griffon		pastel	
7494	2105	8748	2105	11,780	2105
64,000,000 mesh points					
suno			paradent		
16,099	2380		21,214	2380	

Table 3
Elapsed time (seconds) and number of relaxations for sequential algorithm with Schwarz–Howard method on cluster suno (Sophia), paradent (Rennes), griffon (Nancy) and pastel (Toulouse).

32,768,000 mesh points					
suno		griffon		pastel	
7269	2066	8915	2066	9611	2066
64,000,000 mesh points					
suno			paradent		
15,820	2337		21,752	2337	

Table 4
Elapsed time (s), number of relaxations and communication time with Schwarz alternating method for the multivalued problem on cluster suno (Sophia) and paradent (Rennes) for 64,000,000 mesh points.

P	Async.	Relaxations			Comm	Sync.	Relaxations	Comm	τ
		Min	Max	Average					
4	8409	3617	4424	4014	521 (6.2%)	11,300	2380	5255 (46%)	1.34
8	3977	3234	4290	3729	249 (6.3%)	6033	2380	2411 (40%)	1.51
16	1709	2334	3963	3269	68 (3.9%)	2625	2380	830 (32%)	1.53
32	934	2208	4317	3355	60 (6.4%)	1599	2380	397 (25%)	1.67
64	428	1310	4621	2876	37 (8.6%)	1120	2380	261 (13%)	2.61

Table 5
Elapsed time (s), number of relaxations and communication time with Schwarz–Howard method on cluster suno (Sophia) and paradent (Rennes) for 64,000,000 mesh points.

P	Async.	Relaxations			Comm	Sync.	Relaxations	Comm	τ
		Min	Max	Average					
4	8254	3341	4132	3741	628 (7.6%)	11,447	2337	3880 (34%)	1.39
8	4141	2910	4362	3694	329 (7.9%)	6098	2337	2778 (45%)	1.47
16	1818	2474	4098	3351	72 (3.9%)	2705	2337	872 (32%)	1.49
32	956	2313	4328	3326	58 (6.0%)	1607	2337	398 (24%)	1.68
64	414	1720	4021	2928	28 (6.7%)	1162	2337	281 (24%)	2.80

Table 6
Elapsed time (s), number of relaxations and communication time with Schwarz alternating method for the multivalued problem on cluster suno (Sophia), griffon (Nancy) and pastel (Toulouse) for 32,768,000 mesh points.

P	Async.	Relaxations			Comm	Sync.	Relaxations	Comm	τ
		Min	Max	Average					
4	3198	2885	3443	3041	46 (1.4%)	5257	2105	2284 (43%)	1.64
8	1822	2971	3405	3201	78 (4.0%)	2494	2105	873 (34%)	1.37
16	773	2176	3424	2802	25 (3.2%)	1156	2105	278 (26%)	1.50
32	442	2059	3712	2931	33 (7.4%)	958	2105	239 (24%)	2.17
64	202	1139	4265	2594	22 (11.0%)	815	2105	189 (23%)	4.03
128	57	683	3500	2267	8 (14.0%)	549	2105	133 (24%)	9.63

Table 7
Elapsed time (s), number of relaxations and communication time with Schwarz–Howard method on cluster suno (Sophia), griffon (Nancy) and pastel (Toulouse) for 32,768,000 mesh points.

P	Async.	Relaxations			Comm	Sync.	Relaxations	Comm	τ
		Min	Max	Average					
4	3283	2702	3497	3010	61 (1.9%)	5323	2066	2282(43%)	1.62
8	1830	2893	3483	3099	73 (3.9%)	2697	2066	955 (35%)	1.47
16	833	2158	3585	2860	29 (3.4%)	1164	2066	280 (24%)	1.40
32	443	2058	3761	2908	29 (6.5%)	913	2066	224 (24%)	2.06
64	211	1359	4479	2625	23 (10.9%)	788	2066	184 (23%)	3.73
128	58	718	3360	2131	8 (13.8%)	493	2066	122 (25%)	8.5

parallel computational methods. Nevertheless, considering the more general Hamilton–Jacobi–Bellman problem (17), the multivalued formulation is not relevant, while the linearized formulation by the Howard method is interesting.

Tables 2 and 3 display the sequential elapsed times obtained on each cluster. The number of machines used has been limited up to 64 or 128 according to the duration of the exploitation of the computational codes. Moreover, for the considered size of the model problem, since the overhead damages the performances of the parallel algorithms, additional machines are not necessary particularly when synchronous scheme are tested; indeed more processors are not really interesting and the performances will not be improved.

In these tables and figures, comparison of elapsed time for parallel synchronous and asynchronous methods are presented; since the notion of speed-up and efficiency are not relevant when distant and heterogeneous clusters are used, the parameter τ measures the ratio of elapsed time between synchronous and asynchronous methods; this parameter is more appropriate for the comparison between the previous two kinds of parallel methods.

Figs. 4 and 5 (Figs. 6 and 7, respectively) present the elapsed time obtained when the mesh is constituted by 64,000,000 (32,768,000, respectively) mesh points and when 2 (3, respectively) sites are used. Figs. 8 and 9 show the evolution of the parameter τ when the number of processors increases and when 2 or 3 sites are used.

Tables 4–7, Figs. 4–7 show clearly the benefit of using parallel algorithms. Note that the asynchronous parallel iterative algorithms are more efficient than the synchronous ones; so the advantage of the first method compared to the second one is clearly showed.

In Tables 4–7, it can be noted, that in asynchronous parallel experiments, the number of relaxations necessary to reach convergence is greater when few processors are used than the one necessary when parallel synchronous schemes are used. Furthermore,

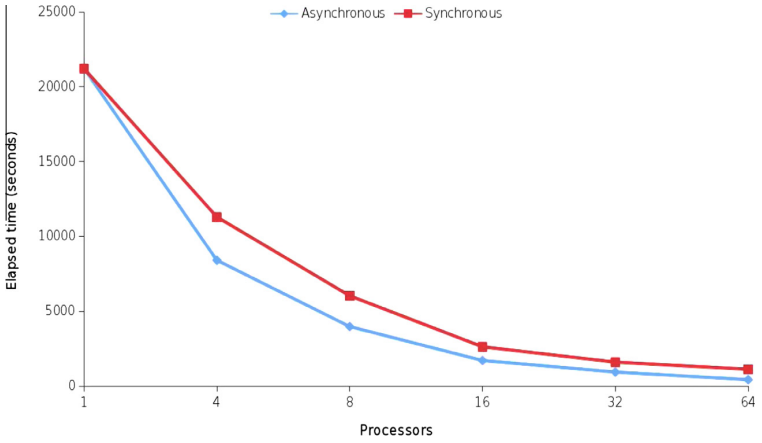


Fig. 4. Elapsed time (s) with Schwarz alternating method for the multivalued problem on cluster suno (Sophia) and paradent (Rennes) for 64,000,000 mesh points.

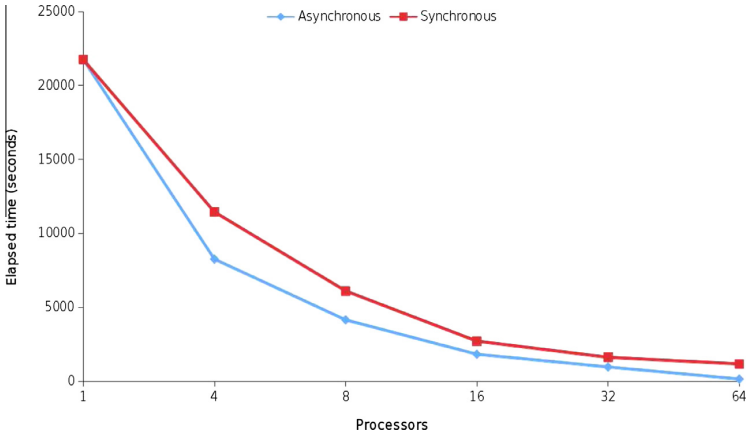


Fig. 5. Elapsed time (s) with Schwarz–Howard method on cluster suno (Sophia) and paradent (Rennes) for 64,000,000 mesh points.

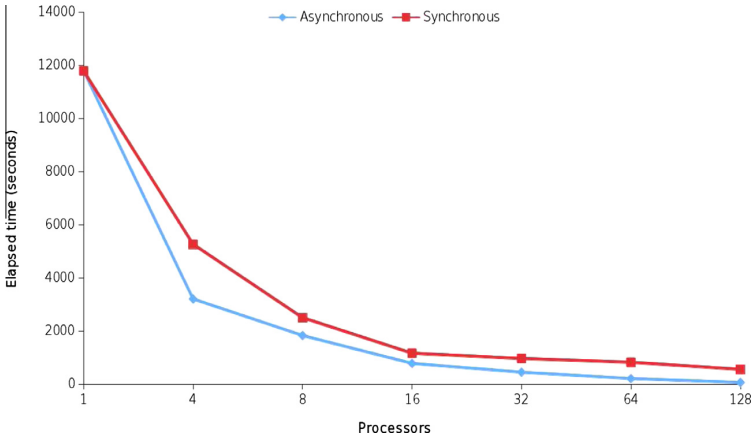


Fig. 6. Elapsed time (s) with Schwarz alternating method for the multivalued problem on cluster suno (Sophia), griffon (Nancy) and pastel (Toulouse) for 32,768,000 mesh points.

despite higher number of relaxations, elapsed time of asynchronous scheme is less than the one obtained when synchronous scheme is used. This asynchronism is an efficient way to deal with communication overhead and load unbalance. It turns out that the overhead generated by additional relaxations in the case of asynchronous algorithms is smaller than the synchronization overhead

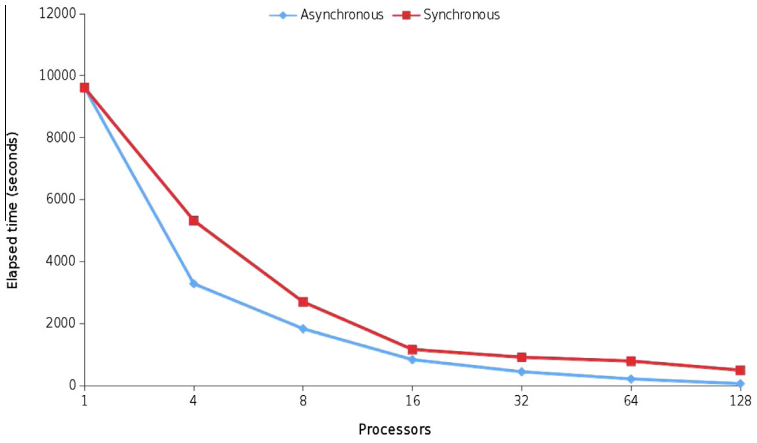


Fig. 7. Elapsed time (s) with Schwarz–Howard method on cluster suno (Sophia), griffon (Nancy) and pastel (Toulouse) for 32,768,000 mesh points.

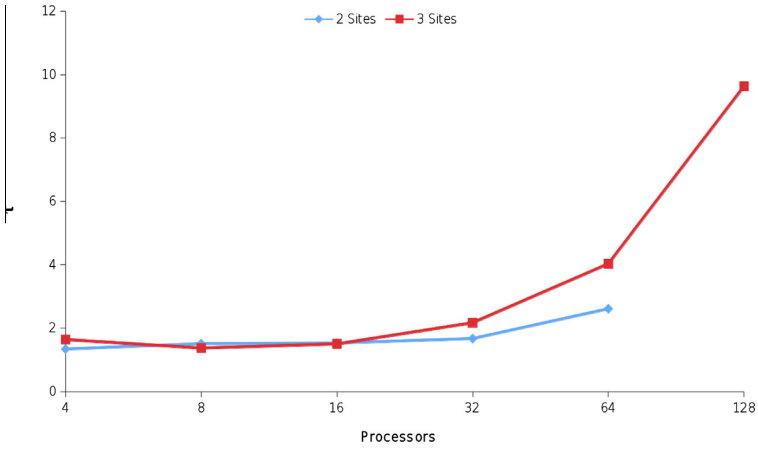


Fig. 8. Comparison of the values of the parameter τ with Schwarz alternating method for the multivalued problem when the number of processors increases and when 2 and 3 sites are used.

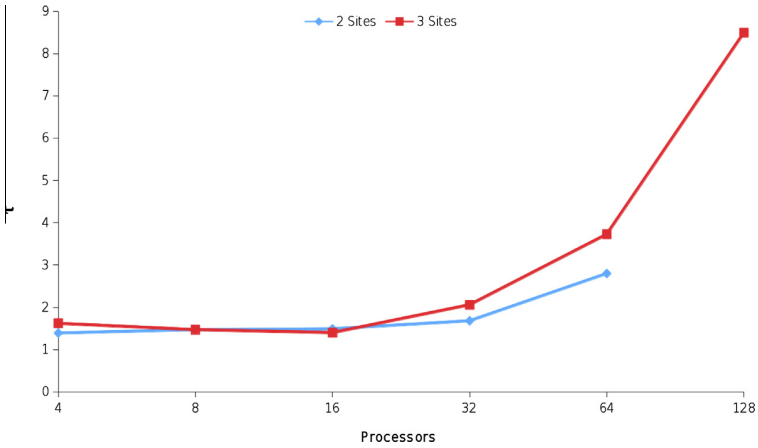


Fig. 9. Comparison of the values of the parameter τ with Schwarz–Howard method when the number of processors increases and when 2 and 3 sites are used.

combined with processor idle time of parallel synchronous schemes of computation. Nevertheless these additional relaxations improve the solution's accuracy and the numerical quality of the computations. Moreover, the efficiency of the synchronous algorithm decreases faster than the efficiency of asynchronous algorithm when the number of processors increases. The lack of

synchronization points and the use of current values of the components of the iterate vector induce better performances for the asynchronous Schwarz alternating methods when several processors are used.

We focus now on experiments performed for the same problem with the same parameters on two and three distant sites.

Results are summarized in Tables 4–7. Many interesting issues can be pointed out from these tables:

1. it can be noticed that with both Schwarz alternating methods, we obtain good results; indeed this is due to the fact that the discretization matrix is nonsymmetric and it is well known that the block relaxation method is well adapted to this case,
2. the results on two or three distant sites show that the synchronous version of both Schwarz alternating methods is slower than the asynchronous one. The ratio τ obviously highlights the power of the asynchronous version compared to the synchronous one.

In Tables 4–7, note that another feature of the parallel experiments consists in the fact that the parameter τ is increasing with the increasing number of machines used (see Figs. 8 and 9). This means again that the efficiency of asynchronous schemes is clearly better than the efficiency of the synchronous one. Then clearly, asynchronous algorithms are well-adapted to the use of distant and heterogeneous machines.

Finally, note that the use of asynchronous parallel schemes is very interesting, since the weight of synchronizations is low and, as a consequence, the obtained elapsed times are shorter. Indeed, since non-blocking MPI communication subroutines are used in asynchronous algorithm, the low values of communication ratio in this case can only illustrate the absence of idle time. On the other hand, a blocking communication subroutine is used in the synchronous algorithm; so, the communication time measurements represent the idle times due to synchronizations and the amount of CPU operation dedicated to communications. Moreover Tables 4–7 and Figs. 4–7 show clearly the impact of the communications on the behavior of the iterative parallel algorithm and the fact that such communications degrade the performances of the synchronous methods. Due to the weight of synchronizations, parallel asynchronous experiments on the grid platform show clearly the interest of such methods on distant clusters. So parallel asynchronous iterative algorithms, more particularly on grid platform, are appropriate ways to solve complex problems.

Conclusion

In this study we have presented the parallel numerical solution of the stationary obstacle problem arising for example in a model problem of civil-engineering, solved by the parallel synchronous or asynchronous Schwarz alternating method. Due to the size of the algebraic systems to be solved and also to the fact that such system is very ill-conditioned and finally that the computational time is very long, this kind of problem is very difficult to solve by sequential algorithms. For the numerical solution of the considered problem, the stability of the time marching scheme is classically ensured; the convergence of the implemented methods has been ensured by appropriate spatial discretization scheme. Experiments with the Grid'5000 architecture allow us to compare the behavior of both synchronous and asynchronous schemes of computation in various configurations with geographically distant and heterogeneous sites. We have obtained very good performances of the parallel algorithms, particularly for the asynchronous ones. Due to the weight of synchronizations, parallel asynchronous experiments on the grid platform show clearly the interest of such method on distant sites. So parallel asynchronous iterative algorithms, more particularly on grid platform, are appropriate ways to solve this complex problem. In conclusion, it appears from our experience that parallel asynchronous algorithms are robust, efficient methods of computation and do not need load balancing between the processors.

Acknowledgement

This study has been made possible with the support of Grid'5000 platform.

References

[1] Glowinski R, Lions JL, Tremolieres R. Analyse numérique des inéquations variationnelles. Dunod, tome 1 and 2; 1976.
[2] Ciarlet PG. Numerical analysis of the finite element method. Presses de l'université de Montréal; 1975.
[3] Lions P, Mercier B. Approximation numérique des équations de Hamilton–Jacobi–Bellman. RAIRO Anal Numér 1980;14:369–93.
[4] Tai XC. Rate of convergence for some constraint decomposition methods for nonlinear variational inequalities. Numer Math 2003;93:755–86.
[5] Badea L, Tai XC, Wang J. Convergence rate analysis of a multiplicative Schwarz method for variational inequalities. S I AM J Numer Anal 2004;41–3:1052–73.
[6] Badea L, Wang J. An additive Schwarz method for variational inequalities. Math Comput 2000;69:1341–54.
[7] Tai XC. Convergence rate analysis of domain decomposition methods for obstacle problems. East-West J Numer Anal 2001;9(3):233–52.
[8] Kuznetsov YA, Neittaanmaki P, Tarvainen P. Block relaxation methods for algebraic obstacle problems with M-matrices. East-West J Numer Math 1996;4:69–82.
[9] Kuznetsov YA, Neittaanmaki P, Tarvainem P. Schwarz methods for obstacle problems with convection–diffusion operators. In: Keyes DE, Xu JC, editors. Proceedings of domain decomposition methods in scientifical and engineering computing, AMS; 1995. p. 251–6.
[10] Li C, Zeng J, Zhou S. Convergence analysis of generalized Schwarz algorithms for solving obstacle problems with T-monotone operator. Comput Math 2004;48:373–86.
[11] Tai XC, Tseng P. Convergence rate analysis of an asynchronous space decomposition method for convex minimization. Math Comput 2002;71–239:1105–35.
[12] Bolze R, Cappello F, Caron E, Daydé M, Desprez F, Jeannot E, et al. Grid'5000: a large scale and highly reconfigurable experimental grid testbed. Int J High Perform Comput Appl 2006;20(4):481–94.
[13] Baudet G. Asynchronous iterative methods for multiprocessors. J Assoc Comput Mach 1978;25:226–44.
[14] Chau M, Couturier R, Bahi J, Spiteri P. Parallel solution of the obstacle problem in Grid environment. High Perform Comput Appl 2011;25(4):488–95.
[15] Spiteri P, Chau M. Parallel asynchronous Richardson method for the solution of obstacle problem. In: Almhana JN, Bhavsar VC, editors. Proceedings of the 16th annual international symposium on HPCS, 2002 IEEE; 2002. p. 133–8.
[16] Chau M, Couturier R, Bahi J, Spiteri P. Parallel solution of the sequence of obstacle problems in a grid environment. In: Topping BHV, Ivânyi P, editors. Proceedings of the second international conference on parallel, distributed, grid and cloud computing for engineering. Stirlinshire, Scotland: Civil-Comp. Press; 2011.
[17] Chau M, Couturier R, Bahi J, Spiteri P. Asynchronous grid computation for american options derivatives. Adv Eng Softw 2013:136–44.
[18] Miellou JC, Spiteri P. Two criteria for the convergence of asynchronous iterations. In: Chenin P et al., editors. Computers and computing. Paris: Wiley Masson; 1985. p. 91–5.
[19] Miellou JC, Spiteri P. Un critère de convergence pour des méthodes générales de point fixe. M2 AN 1985;19(4):645–69.
[20] Miellou JC, Spiteri P. Itérations asynchrones avec segments d'ordre. Méthodes numériques performantes, Contrat ATP – CNRS n 8096; 1984.
[21] Miellou JC, ElBaz D, Spiteri P. A new class of asynchronous iterative algorithms with order interval. Math Comput 1998;67(221):237–55.
[22] Ortega JM, Rheinboldt WC. Iterative solution of nonlinear equations in several variables. New York: Academic Press; 1970.
[23] Barbu V. Nonlinear semigroups and differential equations in Banach spaces. Noordhoff International Publishing; 1976.
[24] Spiteri P, Miellou JC, El Baz D. Asynchronous Schwarz alternating methods with flexible communication for obstacle problem. Réseaux Syst Répartis 2001;13(1):47–66.
[25] Giraud L, Spiteri P. Résolution parallèle de problèmes aux limites non linéaires. M2 AN 1991;25:73–100.
[26] Lions JL, Stampacchia G. Variational inequalities. Commun Pure Appl Math 1967;XX:493–519.
[27] Hoffman KH, Zou J. Parallel efficiency of domain decomposition methods. Parallel Comput 1993;19:1375–91.
[28] Evans DJ, Deren W. An asynchronous parallel algorithm for solving a class of nonlinear simultaneous equations. Parallel Comput 1991;17:165–80.

[29] El Baz D, Frommer A, Spitéri P. Asynchronous iterations with flexible communication: contracting operators. J Comput Appl Math 2005;176:91–103.
[30] Chau M, Garcia T, Spiteri P. Parallel asynchronous Schwarz alternating method for the obstacle problem on grid computing. In: Topping BHV, Tsompanakis Y, editors. Proceedings of the thirteenth international conference on civil, structural and environmental engineering computing, Chania, Crete, Greece, Civil-Comp Press, Stirlingshire, UK, paper 321; 2011.

5.8 Article 8

Advances in Engineering Software 60-61 (2013) 111–121

Contents lists available at SciVerse ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Asynchronous grid computing for the simulation of the 3D electrophoresis coupled problem

M. Chau [a], T. Garcia [b,c], P. Spiteri [b,*]

[a] Advanced Solutions Accelerator, 199 rue de l'Oppidum, 34170 Castlenau le Lez, France
[b] ENSEEIHT-IRIT, 2 rue Charles Camichel, 31071 Toulouse Cedex, France
[c] UVSQ-PRISM, 45 Avenue des États-Unis, 78035 Versailles Cedex, France

ARTICLE INFO

Article history:
Received 13 November 2012
Accepted 13 November 2012
Available online 19 December 2012

Keywords:
Parallel computing
Schwarz alternating method
Continuous-flow electrophoresis
Grid computing
Asynchronous iteration
PISO algorithm
Mass transfer

ABSTRACT

This paper deals with the implementation on the Grid'5000 network, the French grid platform, of parallel simulations for the solution of a 3D coupled boundary value problem. This problem models continuous flow electrophoresis which is governed by Navier Stokes equations coupled with convection–diffusion and potential equations. The coupled boundary value problems are discretised by appropriate schemes involving the convergence of the parallel synchronous and asynchronous Schwarz alternating algorithms. Finally, parallel experiments on grid computing are presented and analyzed.

1. Introduction

The electrophoresis problem occurs in many scientific fields such as agronomy, medicine and more generally in biological applications. It is a process for separating protein mixtures currently used for analysis but difficult to extend to a preparative scale, with a continuous water flow which transports the mixture through the electrical field. Previous works have been developed on the modelling of this process by coupled boundary value problems. Among such physical models we refer to the one developed in [1,2]. In these works the model consists in the coupling of Navier Stokes equations for the hydrodynamic part with electrical field potential equation and as many convective transport equations as the number of proteins to be separated.

The spatial and the temporal discretization of this system of non-linear or linear coupled partial differential equation leads to the solution of large scale algebraic systems. If an explicit time marching scheme is used, a stability condition is necessary. The numerical stability of time marching scheme is well ensured if implicit or semi-implicit schemes are considered. So we have to solve a sequence of stationary algebraic systems. Moreover, taking into account the large size of the sparse systems to solve, iterative methods are well suited. Furthermore parallel iterative processing is an appropriate answer in order to minimize the computational cost. Unfortunately, in parallel computations, synchronizations occur during the communications at the beginning of each iteration, when a processor waits for a message sent by the others processors. Thus synchronizations of concurrent parallel tasks induce idle times between processors and consequently increase elapsed time. The aim of the present study is to present a particular class of parallel methods, the parallel asynchronous iterative algorithms, that do not require synchronization between processors. Parallel asynchronous algorithms are aimed at suppressing idle times without using load balancing techniques. As the processors can work concurrently without any order nor synchronization, such algorithms seem well suited to distributed and grid platform.

Furthermore subdomains methods are well suited to parallel computations. Among the classical subdomain methods one of the most efficient [3] for solving boundary value problem is the Schwarz alternating method with overlapping between the subdomains. The main difficulty that arises in the implementation of domain decomposition methods is the handling of the computational overhead due to the synchronizations between processors. This feature is critical in grid environments. Thus, in order to solve the electrophoresis problem we propose in the present study to implement and analyze a variant of parallel asynchronous Schwarz alternating method with or without flexible communications between the processors derived from previous studies (see [4,5]).

* Corresponding author. Tel.: +33 534322166; fax: +33 534322157.
E-mail addresses: mchau@advancedsolutionsaccelerator.com (M. Chau), thierry.garcia@enseeiht.fr, thierry.garcia@uvsq.fr (T. Garcia), pierre.spiteri@enseeiht.fr (P. Spiteri).

0965-9978/$ - see front matter © 2012 Civil-Comp Ltd and Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.advengsoft.2012.11.010

From a mathematical point of view, all the linear systems derived from appropriate discretization of the linear and non-linear boundary values problems have a common property. Indeed, if suitable discretization schemes are selected, the diagonal entries of the matrices are strictly positive, the off-diagonal entries are non-positive and the matrices are strictly or irreducibly diagonal dominant. Then, all these matrices are symmetric or nonsymmetric M-matrices [6]. According to theoretical results (see [4,5,7,8]) the previous properties ensure the convergence of the parallel synchronous and asynchronous Schwarz alternating method with or without flexible communications between the processors.

Then, at each time step, since the spatial discretization step size must be chosen very small for obtaining good accuracy, the considered method consists in solving seven large sparse linear systems. Taking into account of the current development of the parallel architectures, the computation on grid environment seems a suitable choice. Then, in the present study, we present numerical simulations of continuous flow electrophoresis performed on such parallel distributed architectures by means of parallel synchronous and asynchronous Schwarz alternating method. In order to compare and analyze the behaviour of the parallel synchronous and asynchronous algorithms, when distant and heterogeneous clusters are used for parallel experiments on a grid, we define a new parameter corresponding to the ratio between the elapsed time of synchronous and asynchronous algorithms. This parameter permits us to measure the gain of the studied parallel asynchronous algorithm. Moreover, in the parallel experiments, we have also considered the impact of the communications on the elapsed time of the iterative parallel algorithms for both synchronous and asynchronous mode.

Lastly the main goal of the present study is to produce an interesting application to illustrate our previous theoretical papers (see [4,5,7,8]). Consequently, in the present paper, parallel experiments have been highly developed, particularly by considering on one hand very large mesh points and consequently the solution of very large scale algebraic systems and on the other hand by using heterogeneous and distant clusters in order to test in reality the impact of asynchronism with respect to the use of architectures like Grid. Taking into account the number of our theoretical papers published, this study is not a supplementary fundamental contribution but an experimental one.

The present paper is organized as follows. In Section 2 we present the physical model describing the continuous flow electrophoresis. The next section is devoted to the numerical solution of the studied problem; then we present appropriate discretizations of the coupled boundary value problems ensuring the convergence of the parallel synchronous and asynchronous Schwarz alternating method; a comparison with some other theoretical works is also achieved. In Section 4 the experimental methodology is described and the parallel experiments on Grid'5000, the French grid platform, are presented and analyzed.

2. The physical problem and the associated model

Continuous flow electrophoresis is a process for separating protein mixture (see Fig. 1). Currently used for analysis, its extension to a preparative scale is difficult due to the size of the electrophoresis chamber (see [1,2]). Its resolution is determined by the migration distance at the collection plane and the fineness of the filament occupied by each protein species. Filaments undergo spreading due to a number of different phenomena, among which electrokinetics and electrohydrodynamics are known to be important. Firstly, differences in migration velocity between the ionic species give rise to local variations in electrical conductivity near the protein filament. And secondly, the local change in electrical conductivity distorts the electrical field, thus inducing shear stress

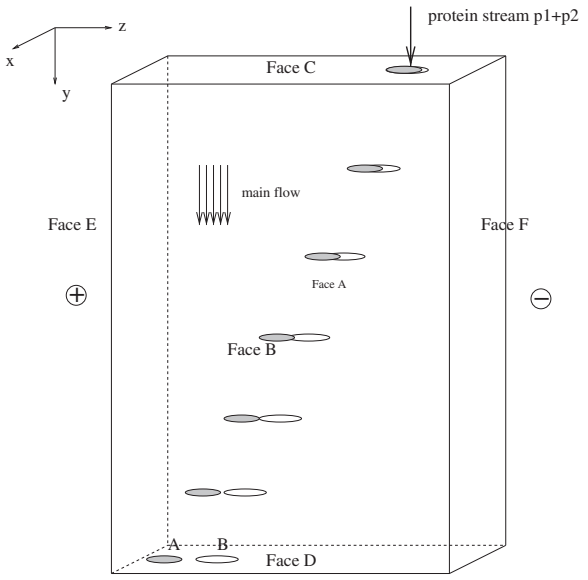


Fig. 1. The principle of continuous flow electrophoresis.

in the liquid and creating a local flow pattern. More precisely density coupling phenomena involving thermal and solutal connexion are likely induce strong instability effect.

A physical model has been developed (see [1,2]) in order to describe these phenomena when proteins are being separated. As the current study is aimed at evaluating parallel efficiency of studied algorithms, we only consider in the sequel the migration of one protein. This model consists in the coupling of three boundary value problems defined in a bounded domain Ω included in the three dimensional space. According to the usual shape of the electrophoresis chamber, we will consider in the sequel that Ω included in the three dimensional space, is a parallelepiped. The coupled equations describing the considered phenomena are:

- the Navier–Stokes equations with mixed boundary conditions, the Dirichlet boundary conditions being preponderant,
- an evolutive convection–diffusion equation with mixed boundary conditions, more precisely the Dirichlet boundary conditions arising on three faces,
- a potential equation which corresponds to a generalized Laplacian with Dirichlet boundary conditions.

The Navier–Stokes equations describe the behaviour of the flow; more precisely at each point $M = (x, y, z)$ of the bounded domain Ω the flow of an incompressible viscous fluid is modelled by the classical equation which takes into account the external forces field, as follows:

$$\frac{\partial \vec{V}}{\partial t} + \overline{\overline{\nabla \vec{V}}} \cdot \vec{V} = \nu \Delta \vec{V} - \frac{1}{\rho} \nabla p + S, \tag{1}$$

where $\vec{V} = (u, v, w)$ is the velocity field, p the pressure, ν the kinematic viscosity of the fluid, ρ the volumetric mass of the fluid. $S = (S_x, S_y, S_z)$ is the source term defined by:

$$S_\xi = \epsilon \mathrm{div}(E_\xi \cdot \vec{E}) = \epsilon \left(\frac{\partial}{\partial x} E_\xi E_x + \frac{\partial}{\partial y} E_\xi E_y + \frac{\partial}{\partial z} E_\xi E_z \right), \quad \text{for } \xi = x, y, z,$$

in which ϵ is the dielectric permittivity of the fluid and $\vec{E} = (E_x, E_y, E_z)$ is the electrical field; the following quantity $\overline{\overline{\nabla \vec{V}}}$ is defined by:

$$\overline{\overline{\nabla \vec{V}}} = \begin{pmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} & \frac{\partial u}{\partial z} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} & \frac{\partial v}{\partial z} \\ \frac{\partial w}{\partial x} & \frac{\partial w}{\partial y} & \frac{\partial w}{\partial z} \end{pmatrix}.$$

Eq. (1) is completed by the following mass conservation law:

$$\frac{\partial \rho}{\partial t} + \mathrm{div}(\rho \vec{V}) = 0,$$

which is reduced here to

$$\mathrm{div}(\vec{V}) = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0, \tag{2}$$

since the volumetric mass of the fluid ρ is constant.

The boundary conditions are given hereafter. The fluid comes in the cell by the upper face C and comes out by the lower face D. So we consider that the velocity fulfils nonhomogeneous Dirichlet boundary condition on the face C and homogeneous Neumann boundary condition on the face D: $u_{/C} = w_{/C} = 0$, $v_{/C} = v^0$, where v^0 is the given value of v on the face C; $\frac{\partial u}{\partial n_{/D}} = \frac{\partial w}{\partial n_{/D}} = \frac{\partial v}{\partial n_{/D}} = 0$. Furthermore the velocities u and w are zero on the other four faces; so they fulfil homogeneous Dirichlet boundary conditions on the faces A, B, E and F: $u_{/A} = w_{/A} = 0$; $u_{/B} = w_{/B} = 0$; $u_{/E} = w_{/E} = 0$; $u_{/F} = w_{/F} = 0$. The axial velocity v is zero on the faces A and B: $v_{/A} = v_{/B} = 0$. On the thin faces E and F, v fulfils homogeneous Neumann boundary condition: $\frac{\partial v}{\partial n_{/E}} = \frac{\partial v}{\partial n_{/F}} = 0$.

In order to describe the transport of one protein m, the following time dependent convection–diffusion boundary value problem is considered

$$\frac{\partial c_m}{\partial t} + u\frac{\partial c_m}{\partial x} + v\frac{\partial c_m}{\partial y} + w\frac{\partial c_m}{\partial z} - D_m \Delta c_m = \varphi, \tag{3}$$

where c_m is the concentration of the protein m, D_m is the diffusion coefficient of the protein m and φ is the source term.

The boundary conditions are specified as follows: the proteins come in the cell by the face C; so on this upper face the concentration c^0 is known and it fulfils nonhomogeneous Dirichlet boundary condition: $c_{/C} = c^0$. Furthermore the concentration is free on the other five faces of the cell; so we can consider that on these five faces the concentration fulfils homogeneous Neumann boundary conditions: $\frac{\partial c}{\partial n_{/A}} = \frac{\partial c}{\partial n_{/B}} = \frac{\partial c}{\partial n_{/E}} = \frac{\partial c}{\partial n_{/D}} = \frac{\partial c}{\partial n_{/F}} = 0$.

The electrical phenomena are described by the behaviour of the potential Φ governed by a generalized Poisson equation:

$$-\mathrm{div}(K\ \mathrm{grad}\ \Phi) = \Delta Q, \tag{4}$$

where K is the electrical conductivity of the fluid, Φ the potential and $Q = Q_0 + RT\sum \mu_m c_m$, where R is the constant arising in the law of the perfect gas, T the temperature and $K = K_0 + \sum \lambda_m c_m$, where $\forall m$, $\lambda_m > 0$ and $\sum_{m=1}^{n_p} \lambda_m = 1, \lambda_m$ being the mean ionic conductivity of the protein and μ_m is the electrophoretic mobility of the protein. Eq. (4) can also be written as follows:

$$-\frac{\partial}{\partial x}\left(K\frac{\partial \Phi}{\partial x}\right) - \frac{\partial}{\partial y}\left(K\frac{\partial \Phi}{\partial y}\right) - \frac{\partial}{\partial z}\left(K\frac{\partial \Phi}{\partial z}\right) = \Delta Q. \tag{5}$$

The equation governing the flow (1) is coupled with the above relation by:

$$\vec{E} = -\overrightarrow{\mathrm{grad}}\ \Phi. \tag{6}$$

The potential fulfils Dirichlet boundary conditions on all the faces. As the potential is known and constant at every point of the electrodes, i.e. on the two lateral faces E and F we have: $\Phi_{/E} = \Phi^0$; $\Phi_{/F} = 0$. Furthermore the boundary conditions on faces A and B are obtained by a linear interpolation between the values of the potential defined on the electrodes: $\Phi_{/A} = \Phi_{/B} = \frac{x_1}{L}\Phi^0$, where L is the width of the cell. On the horizontal faces C and D, the boundary conditions are obtained by the solution of the potential equation restricted to each upper and lower face; these boundary conditions are preliminarily fixed. As the concentration on the face C is constant, the potential on this face is computed only once: $\Phi_{/C} = \Phi_C$. On the other hand the concentration on the face D changes when the time increases. Then, at each time step the potential must be computed on this face in order to obtain the boundary condition: $\Phi_{/D} = \Phi_D(t)$.

Numerical simulations can yield insights into the nature of expected effects.

3. The numerical solution

The numerical solution of the electrophoresis problem requires several steps.

3.1. Discretization

The first one consists in the discretization of the global problem. In the present section, we treat the discretization of the boundary value problems which describe the evolution of the electrophoresis problem. Then we derive a common property of the discrete operators allowing to study the convergence of the parallel synchronous and asynchronous domain decomposition methods with overlapping.

The Navier–Stokes equations are discretised on staggered grids by the finite volume method in an implicit approach with respect to the time. Then the global Navier–Stokes problem is solved by the PISO algorithm (Pressure Implicit Splitting of Operators) [9] which is a time marching predictor–corrector algorithm based upon the splitting of the solution of velocity equations and pressure equations. At each time step, PISO method process into three separate steps: one predictor step and two corrector steps where the operations on the velocity are decoupled from those on the pressure. Issa [9] has shown that two corrector steps are sufficient to obtain a suitable accuracy, compatible with the discretization scheme and round off error propagation.

Let u^n, v^n, w^n and p^n the values of velocity and pressure at the nth time step. The predictor step consists in computing one time step for Eq. (1) by leaving the pressure unchanged. An intermediate velocity $\left(u^{n+\frac{1}{3}}, v^{n+\frac{1}{3}}, w^{n+\frac{1}{3}}\right)$ is obtained. Then, each corrector step consists in:

1. solving the mass conservation Eq. (2) where the velocity is substituted by the gradient of pressure,
2. correcting the former approximate velocity with the computed pressure.

Then the velocities $\left(u^{n+\frac{2}{3}}, v^{n+\frac{2}{3}}, w^{n+\frac{2}{3}}\right), (u^{n+1}, v^{n+1}, w^{n+1})$, and the pressures $p^{n+\frac{1}{2}}, p^{n+1}$ are obtained.

The discretization of the equations involved in the PISO method with finite volume method leads to five linear systems (see [10,11]). Note that both matrices are M-matrices [6], i.e. matrices with strictly positive diagonal entries, non-positive off-diagonal entries and strictly diagonally dominant or irreducibly diagonally dominant. Note that the centred discretization scheme applied to the convection terms of Eq. (1) may not necessarily lead to M-matrices.

The evolutive part of the convection–diffusion Eq. (3) is discretised with an implicit scheme; for the spatial part of this equation, the classical seven points scheme is used for the Laplacian and decentred scheme for the convection terms. More precisely, for these last convection terms a forward difference scheme is used if the coefficient of the considered first derivative is strictly positive and a backward difference scheme if this coefficient is

negative. Then, the discretization matrix of the transport equation is always an *M*-matrix, since the diagonal entries of such matrix are strictly positive, the off-diagonal entries are non-positive and the matrix is strictly (irreducibly) diagonally dominant.

Finally, as the conductivity K in the term of diffusion Eq. (4) is not constant, the potential equation is discretised by appropriate finite difference methods. For example, consider that h_i denotes the distance between the nodes number $i-1$ and i in the direction of the abscissa. Consider the discretization of $-\frac{\partial}{\partial x}\left(K\frac{\partial \phi}{\partial x}\right)$ for $y = y_j$ and $z = z_k$ fixed. Then by taking the mean of *forward–backward* and *backward–forward* scheme we obtain easily the following discretization scheme:

$$-\frac{\partial}{\partial x}\left(K\frac{\partial \phi}{\partial x}\right)_{x_i,y_j,z_k} = \frac{1}{2}[-a_i\Phi_{i-1} + b_i\Phi_i - c_i\Phi_{i+1}], \tag{7}$$

where

$$a_i = \left(\frac{K_{i-1}}{h_i^2} + \frac{K_i}{h_i h_{i+1}}\right); \quad b_i = \left(\frac{K_{i-1}}{h_i^2} + \frac{2K_i}{h_i h_{i+1}} + \frac{K_{i+1}}{h_{i+1}^2}\right) \quad \text{and}$$
$$c_i = \left(\frac{K_i}{h_i h_{i+1}} + \frac{K_{i+1}}{h_{i+1}^2}\right).$$

Then, by using classical characterization, it can be shown that the numerical scheme (7) leads also to discretization matrix which is an *M*-matrix.

Reference is made to Refs. [14,15] for more details concerning the values of the entries of the discretization matrices.

Then, note also that the solution of the transport equation and the potential equation lead to two additional large sparse linear systems to solve.

Finally, at each time step, we have to solve seven large sparse linear systems, in which the matrices have the main property to be *M*-matrices.

3.2. Parallel synchronous and asynchronous Schwarz alternating algorithms

Domain decomposition methods, such as the iterative Schwarz algorithm with overlapping are well suited to the parallel solution of boundary values problems (see [3]). In this kind of method, in order to parallelize the computation, the domain Ω, is split into parallelepiped subdomains which overlap each other. Thus, for the computation of the solution of the global problem, sequences of smaller subproblems are solved on each processor of a parallel computer. For a considered subdomain, the restriction of the available values of the computed solution obtained on the neighbouring subdomains are taken as Dirichlet boundary conditions. The synchronous algorithm consists in performing a block Jacobi-like method. Besides, in the classical asynchronous algorithm, the components computed by the other processors may be delayed (see [7,8]). In the case of asynchronous algorithm with flexible communications (see [4,12,13]) the current value of any component of the iterate vector is not necessarily labelled by an outer iteration number as communication may occur at any time. Then, in this class of method, partial updates can be used at any time in the computation. Thus, flexible data exchanges between processors are allowed; as a consequence, the coupling between communications and computations can be improved. Note also, that practically more accuracy can be obtained since smaller systems are to be solved.

In the previous subsection, we have shown that the discretization of the 3D continuous flow electrophoresis problem leads to the solution of seven linear algebraic systems; furthermore the matrices arising in these seven linear systems are all *M*-matrices. Then, according to theoretical results obtained in [4,5,7,8], this last property ensures the convergence of the parallel synchronous and asynchronous (with or without flexible communications) Schwarz alternating algorithms. Reference is made to these four last references for more details concerning these results.

More precisely, consider the following system of algebraic equations:

$$A \cdot X = F, \tag{8}$$

where $A \in \mathcal{L}(\mathbb{R}^n)$, $X \in \mathbb{R}^n$ and $F \in \mathbb{R}^n$. Furthermore, assume that

$$A \text{ is an } M\text{-matrix}. \tag{9}$$

The numerical solution of (8) by the Schwarz alternating method is equivalent to the solution of the following system:

$$\widetilde{\mathcal{A}} \cdot \widetilde{\mathcal{X}} = \widetilde{\mathcal{F}}, \tag{10}$$

where $\widetilde{\mathcal{A}}, \widetilde{\mathcal{X}}$ and $\widetilde{\mathcal{F}}$ are derived from the augmentation process of the Schwarz alternating method [16]. This process is a theoretical model that represents the solution of the algebraic system (8) by a Schwarz domain decomposition method. In the implementation of the algorithms, $\widetilde{\mathcal{A}}$ and $\widetilde{\mathcal{F}}$ are not explicitly computed. According to a result of Evans and Deren [16], the matrix $\widetilde{\mathcal{A}}$ is also an *M*-matrix. So, the system (10), derived from the augmentation process, has the same property as in the initial algebraic system (8).

Let $\alpha \in \mathbb{N}$ be a positive integer and consider now the following block decomposition of problem (10) into α subproblems:

$$\sum_{j=1}^{\alpha} \widetilde{\mathcal{A}}_{ij} \cdot \widetilde{\mathcal{X}}_j = \widetilde{\mathcal{F}}_i, \quad \forall i \in \{1,\ldots,\alpha\}, \tag{11}$$

where $\widetilde{\mathcal{X}}_i \in \mathbb{R}^{n_i}$ and $\widetilde{\mathcal{F}}_i \in \mathbb{R}^{n_i}$, where n_i denotes the size of the ith block of the previous vectors and $\widetilde{\mathcal{A}} = (\widetilde{\mathcal{A}}_{ij})_{1\leqslant i,j,\leqslant\alpha}$, according to the associated block decomposition.

Consider now the solution of subproblems (11) by the asynchronous parallel iteration which can be written as follows:

$$\begin{cases} \widetilde{\mathcal{A}}_{ii} \cdot \widetilde{\mathcal{X}}_i^{r+1} = \widetilde{\mathcal{F}}_i - \sum_{j\neq i} \widetilde{\mathcal{A}}_{ij} \cdot \widetilde{\mathcal{W}}_j, & \text{if } i \in s(r), \\ \widetilde{\mathcal{X}}_i^{r+1} = \widetilde{\mathcal{X}}_i^{r}, & \text{if } i \notin s(r), \end{cases} \tag{12}$$

where $\{\widetilde{\mathcal{W}}_1, \widetilde{\mathcal{W}}_2, \ldots, \widetilde{\mathcal{W}}_\alpha\}$ are the available values of the components $(\widetilde{\mathcal{X}}_j)_{j\neq i}$, which will be specified more precisely in the sequel, and $\mathcal{S} = \{s(r)\}_{r\in\mathbb{N}}$ is a sequence of non-empty subsets of $\{1,2,\ldots,\alpha\}$. In other words, $s(r)$ is the subset of the subscripts of the components updated at the rth relaxation. In the sequel, we will also consider the vector

$$\mathcal{R} = \{\rho_1(r),\ldots,\rho_\alpha(r)\}_{r\in\mathbb{N}},$$

denoting a sequence of integer vectors from $\mathbb{N}^\alpha$. In order to take into account the asynchronism between the processors, $\mathcal{R}$ models the delays between the parallel updates of each component, at the rth relaxation. Furthermore $\mathcal{S}$ and $\mathcal{R}$ verify the following assumptions:

$$\begin{aligned} &\forall i \in \{1,2,\ldots,\alpha\}, \quad \text{the set } \{r \in \mathbb{N} | i \in s(r)\} \text{ is unbounded}, \\ &\forall i \in \{1,2,\ldots,\alpha\}, \quad \forall r \in \mathbb{N}, \ 0 \leqslant \rho_i(r) \leqslant r, \\ &\forall i \in \{1,2,\ldots,\alpha\}, \quad \lim_{r\to\infty} \rho_i(r) = +\infty. \end{aligned}$$

Remark 1. Classically, when $\rho_i(r) = r$ for all $i \in \{1,\ldots,\alpha\}$ and for all $r \in \mathbb{N}$, then (12) models a synchronous Schwarz alternating method.

For the solution of system (8), the classical parallel asynchronous Schwarz alternating method with flexible communications, corresponds to the more general model of parallel asynchronous iterations. In such a case, the access to the available values of the components of the iterate vector $\widetilde{\mathcal{W}}_j$ is given by the following norm constraint [5]:

$$\left\|\widetilde{\mathcal{W}}_j - X_j^{\star}\right\|_{j,\infty} \leqslant \|W^{\rho(r)} - X^{\star}\|_{\infty}, \quad \forall j \in \{1,\ldots,\alpha\}, \tag{13}$$

where $W^{\rho(p)} = \left(W_1^{\rho_1(p)},\ldots,W_\alpha^{\rho_\alpha(p)}\right)$ and $\|.\|_\infty$ denotes a suitable weighted uniform norm [5] and $\|.\|_{j,\infty}$ an analogous weighted uniform norm defined in $\mathbb{R}_j^n, j = 1,\ldots,\alpha$. It can be noted that, in the present context, the values of the components of the iterate vector generated by the other process, can be accessed while the computations are still in progress.

So, if assumption (9) is verified, if $\widetilde{\mathcal{W}} = \{\widetilde{\mathcal{W}}_1,\ldots,\widetilde{\mathcal{W}}_\alpha\}$ is defined according to (13), the numerical solution of problem (8) by the parallel asynchronous Schwarz alternating method with flexible communications associated to (12) and starting from any initial guess X^0, converges to the solution of $A \cdot X = F$. Reference is made to [5] for a proof of this previous result.

Remark 2. Another possibility to analyze the behaviour of parallel asynchronous Schwarz alternating method with flexible communications consists in using partial ordering techniques linked with the discrete maximum principle (see [4,12,13]). This approach, defined in a different mathematical background, is nevertheless more restrictive than the one previously considered. This method consists in starting the iteration (12) with an initial guess X^0 such that

$$A \cdot X^0 - F \geqslant 0. \tag{14}$$

In this case, for $r \geqslant 1$, the vector $\widetilde{\mathcal{W}} = \{\widetilde{\mathcal{W}}_1, \widetilde{\mathcal{W}}_2,\ldots,\widetilde{\mathcal{W}}_\alpha\}$ belongs to the order interval $\langle \mathcal{X}^r, \min(\mathcal{X}^{\rho(r)}, \mathcal{X}^q)\rangle$, where $\mathcal{X}^{\rho(r)}$ denotes the vector with block components $\mathcal{X}_j^{\rho_j(r)}$, $j \in \{1,2,..,\alpha\}$ and $q = \max_{k\in K_{s(r)}^r}(k)$, where the set K_i^r contains all the iteration numbers lower than r associated with the computation of the ith block component (see [4]). Note that the values of the components $\widetilde{\mathcal{W}}_j$ are also the available values of the iterate vector. In this case, if (14) is satisfied, it can be proved (see [4,12,13]) that the sequence of iterate vectors satisfy the maximum discrete principle; indeed $(X^r)_{r\geqslant 0}$ satisfies $X \leqslant \cdots \leqslant X^r \leqslant \cdots \leqslant X^0$.

Remark 3. A particular class of asynchronous Schwarz alternating method corresponds to the one where the updates are performed at the end of any relaxation and defined by $\widetilde{\mathcal{W}}_j = \widetilde{\mathcal{X}}_j^{\rho_j(r)}$. This case corresponds to the classical parallel asynchronous iterations as defined in [7,8]. Then, since A is an M-matrix, according to a result of [8], it can be proved by a straightforward way, since the uniform weighted norm of the error at the step r is decreasing, that, for any subdomain decomposition, the numerical solution of problem (8) by the classical parallel asynchronous Schwarz alternating method defined by (12) and starting from any initial guess X^0 converges to the solution of $A \cdot X = F$.

Remark 4. Sometimes, the implementation of parallel algorithms requires the use of lexicographical ordering or red–black ordering. The previous convergence results still hold, in both cases of natural and red–black ordering of the mesh points, and moreover, for similar ordering of the subdomains. Let us denote respectively by A and $\overline{A}$ the corresponding matrices associated with the two considered ordering. Assume that A is an M-matrix, then $\overline{A}$ is also an M-matrix since $\overline{A}$ is obtained from A by a permutation matrix P which preserves the sign of the entries and particularly the sign of the diagonal entries. Furthermore we have $\overline{A} = P \cdot A \cdot P^t$; thus we have $\overline{A}^{-1} = P \cdot A^{-1} \cdot P^t$. The matrix A being an M-matrix, it follows that A^{-1} is a non-negative matrix [6]. Thus $\overline{A}^{-1}$ is also a non-negative matrix obtained from A^{-1} by the same permutation as the one considered for A. Then $\overline{A}$ is an M-matrix. This proves that the asynchronous parallel methods converge in the context of the red–black ordering of both mesh points and subdomains.

Finally and in conclusion, since the discretization schemes applied to the numerical approximation of the coupled boundary value problems modelling the continuous flow electrophoresis, lead to the solution of seven linear algebraic systems in which the matrices to invert are M-matrices, then, the parallel synchronous and asynchronous Schwarz alternating method with or without flexible communications converge to the solution of the considered discretised boundary value problem for any initial guess X^0 and for any ordering of the subdomains.

3.3. Connection with various works in the literature

The convergence of the Schwarz method has been studied by many authors in several works. In all our published theoretical papers, we have considered the solution of non-linear problems; it is one of the major difference between our approach and for example the papers [17–19]. The solution of linear problems is then considered like a particular case of our more general studies, which will be applied in the present paper. Nevertheless the direct study of the solution of linear algebraic systems give more informations due to the richness of the results obtained for the linear operators. It can be noted that the restriction to the linear case of our approach permits us to be in accordance with most of the direct studies developed in the linear case, this fact being very reassuring. With respect to the other contributions, we can notice that the works of our team have been started since 1974, just after the pioneer work published in 1969 by Chazan and Miranker concerning the solution by chaotic iterations of linear problems [20].

Our approach to solve a non-linear discrete problem is to associate a fixed point mapping to the non-linear operator. Then the convergence of the associated iterative algorithm can be proved by using two distinct ways:

The first approach consists to prove that the fixed point mapping verify a Lipschitz condition with respect to a vectorial norm (practically a vector of norms) in which the Lipschitz constant is replaced by a non-negative matrix J; if furthermore the spectral radius of the matrix J is less than one, then we obtain a property of contraction with respect to the considered vectorial norm. Note also that, thanks to the Perron–Frobenius theorem [6], this previous property leads to a property of contraction with respect to the weighted uniform norm, constructed by using the components of the eigenvector associated to the spectral radius of the matrix J (see [8]). This last point is a common point with the papers [17,19], but it is more general since valid in the non-linear case; it can be also verified that the rate of convergence of the parallel iterative algorithm is linked to the magnitude of this spectral radius (see [8]). In the non-linear case we have also given a characterization of the discrete operators arising in the problem to solve (see [7,8]). Now, when considering the solution of a linear problem, we can verify, by a straightforward way, according to the norm used that the matrix arising in the problem to solve is either a positive definite or a symmetric positive definite matrix either a strictly or an irreducible diagonal dominant matrix either is an M-matrix or an H-matrix (see [7]); then, these properties of the matrices are similar to the properties considered in [17,19]. Lastly, in the non-linear case, in connection with practical applications, we have also considered the solution of quasi-linear problems obtained when a linear operator is perturbed by an increasing diagonal non-linear operator eventually multivalued (corresponding to the situation where the solution is submitted to some constraints).

The second approach consists to consider that the non-linear operator arising in the non-linear problem to solve is an M-function (i.e. an inverse monotone and off-diagonally antitone mapping (see [4,21])), eventually perturbed by a monotone

diagonal multivalued operator; it can be noticed that a linear M-function is an M-matrix. Then, once again, in the linear case the theoretical context of [17,19] can coincide with [4]. If the initial guess satisfies an appropriate condition (see Remark 2, relation (14)), the convergence is monotone according to the fact that the discrete maximum principle is verified.

In these papers [17,19], the authors consider on one hand additive (see [17]) or multiplicative (see [19]) behaviour of the iterative scheme. In our approach, we cannot predict such additive or multiplicative behaviour of the algorithms; we can only consider practically a splitting of the problem in several large scale subproblems solved by a block relaxation method (due to the block structure of the matrices), such that the behaviour of the parallel asynchronous algorithm becomes the more possible multiplicative; this fact follows from the asynchronous updates of the relaxed components.

Another more general situation considered in our previous works is when the parallel asynchronous algorithms are modelled by considering parallel asynchronous schemes with flexible communications between the processors (see [4,5,21]), i.e. the communications are not performed when all the components of a large sub-block are re-actualized but are performed at any time during the computations, as explained previously. This kind of parallel asynchronous schemes with flexible communications have been firstly analyzed by partial ordering techniques (see [4,21]) and then extended to the case where contraction techniques are used (see [5]). Naturally, when considering the classical parallel asynchronous schemes (without flexible communications) which correspond to a particular case, then the behaviour of these particular parallel asynchronous methods is similar to the one considered in [17,19].

We have also studied like a particular case of the parallel asynchronous schemes the behaviour of the Schwarz alternating method. In fact, using our previous theoretical results this study is easy. Indeed, in the most applications considered, we have to solve by the Schwarz alternating method a linear problem perturbed by a monotone diagonal operator. According with a result of Evans and Deren stated in the linear case (see [16]), it is possible to prove when the Schwarz method is considered, and when previous appropriate assumptions are satisfied, that the augmented system to solve satisfy the same properties than the initial problem. Then the behaviour of the parallel asynchronous Schwarz alternating method is similar to the one considered without overlapping between the subproblems.

Note also a distinct work concerning the solution of non-linear problem by multisplitting methods with flexible communications analyzed in the context of monotone convergence, and not considered in the experimental study concerning the model problem but relevant for the solution of such problem (see [21]).

Concerning the numerical solution of the concentration equation, in [18] the author considers the uniform convergence of the Schwarz alternating method for solving singularly perturbed advection–diffusion equations. Note that we have also make a similar study when considering an advection–diffusion operator perturbed by a diagonal monotone eventually multivalued operator corresponding to the case where a constraint on the solution is considered; the results are published in [22,23]. To achieved these two previous studies, we have considered for the discretization of the first derivative forward or backward schemes according to the sign of the coefficients of the first derivative, by a similar way than the one considered in [18]. So the behaviour of the parallel asynchronous Schwarz alternating with flexible communications is achieved by monotone convergence in a similar way than in [18], due to the fact that with such discrete schemes, the discretization matrix of the linear part is an M-matrix. Once again, in the same context, note that the behaviour of parallel asynchronous algorithms with or without flexible communications can also be studied by contraction techniques. Nevertheless, in Ref. [18], the author considers a linear operator and then uses more rich results linked to the theory of such linear mapping; indeed, in [18], according to the linear context, the author can estimate the rate of convergence, linked in fact to the factor of reduction of error. In our case, such estimate of the rate of convergence can be estimated, as previously said, by the spectral radius of the matrix of contraction when contraction techniques are used. In fact in [18] the author considers several case of singular perturbation, and the connection with our situation coincides when we solve the equation of concentration by an implicit time marching scheme; indeed in this case we are in the situation where a zeroth-order term occurs (see [18]), but the situation is more complex in our application, since the coefficients of the first derivatives are unknown and correspond to the components of the speed of the flow. Moreover in [18] the author considers essentially homogeneous Dirichlet boundary conditions, while in our case, the boundary conditions are essentially the mixed ones with predominant Neumann ones. Furthermore, in [18] the experiments are sequential while in our study, the experiments are sequential and parallel synchronous or asynchronous, performed on a Grid.

Then our results of [4,5,7,8], can be applied to the solution of the electrophoresis coupled problem in which we have seven linear algebraic systems to solve.

Lastly, note also that for the numerical solution of the concentration equation, we consider also, instead of a purely implicit time marching scheme, an explicit time marching scheme described hereafter.

3.4. Use of an explicit scheme for the convection–diffusion equation

In the transport equation the diffusion coefficients are very low, so that convection is preponderant. Thus false diffusion may occur. That is the reason why the MPDATA algorithm [24] has also been implemented to solve the transport Eq. (3). This method is based on the classical donor-cell explicit scheme; therefore, the use of a solver is pointless.

The first stage of the method consists in computing one step of the former scheme. Then, the expression of false diffusion, given by a second order Taylor development, is transformed to an *antidiffusive* convection velocity. Thus, donor-cell scheme is applied once again, with the antidiffusive velocity in order to counter the effects of false diffusion. Furthermore, this process can be repeated several times to improve the numerical results. In our simulations, second order MPDATA algorithm have been used. Note that the stability of the MPDATA algorithm is ensured if a CFL condition is satisfied. Nevertheless such CFL condition is not very easy to obtain and to implement in the considered application where the convection coefficients are not constant. Practically, the time step must be chosen small.

4. Parallel simulation

This section is devoted to present the principle of implementation of the parallel algorithms and the results of parallel experiments.

4.1. Principle of implementation

The simulation algorithm is based on master/slave paradigm which is a commonly used approach for parallel and distributed applications. In our case, a master process controls the distribution of work to a set of slave processes.

Algorithm 1.

```
if (master)
  compute discretization matrix and right hand side
  send input data to slaves
  compute SPMD parallel algorithm to solve the linear system
  receive output data from slaves
else
  receive input data from master
  compute SPMD parallel algorithm to solve the linear system
  send output data to master
endif
```

This process is used for the creation and the solution of all linear systems that arise in the numerical simulation of electrophoresis. Matrix and right hand side creation have been implemented sequentially since this part of computation is not very intensive. Indeed, for each considered method in the present study, the complete sequential part of the computation is at most about 6% (respectively about 18%) of the total elapsed time, when two (respectively 32) processors are used. In this way, the master process can be seen as a matrix and vector filler that feeds a parallel linear system solver. In other words, only intensive computations have been parallelized with MPI.[1]

Note also that the principle of implementation of parallel asynchronous iterative subdomain algorithms with flexible communication can be summarized as follows:

```
do until global convergence
  for each subdomain assigned to the processor do
    if local convergence is not reached then
      receive the latest boundary values
      solve the subsystems
      send the boundary values to the neighbours
    end if
  end for
end do
```

Remark 5. In this subsection we have briefly described the principle of implementation. Nevertheless, when considering parallel asynchronous Schwarz alternating method with flexible communications, the overhead induced by the amount of communications increases and performance can be degraded particularly on high latency and low bandwidth networks. It is the reason why, in the implementation we have considered only the implementation of the classical parallel asynchronous Schwarz alternating method.

4.2. Parallel experiments

Parallel simulations for the solutions of the electrophoresis problem have been performed on various single clusters located in France, as well as distant coupled clusters of the Grid'5000 platform; in this network every site hosts clusters and all sites are connected by highspeed communication. In the considered numerical simulations, the size of the domain Ω is $30\,\text{cm} \times 0.5\,\text{cm} \times 10\,\text{cm}$. A regular mesh is defined on Ω. The domain is decomposed into 128 overlapping subdomains numbered using a red–black ordering, well adapted to parallel computation. In the sequel we consider a migration for only one protein injected on the face C of the cell. The Reynolds number is equal to 250, the electrical field is equal to $950\,\text{m}^{-1}$, the electrophoretic mobility is equal to $526 \times 10^{-8}\,\text{m}^2\,\text{v}^{-1}\,\text{s}^{-1}$ and the filament radius is equal to 1.5 mm. Fig. 2 visualizes the computed concentration and the protein filament.

The computational codes have been written using Fortran 90 and the parallelization is achieved using MPI communication library.

The parallel experiments have been performed using an heterogeneous and dedicated Intel processor-based clusters running Linux 64 bits. The clusters of the GRID'5000 platform used are located in Sophia-Antipolis, Rennes and Nancy sites. The characteristics of machines are summarized in Table 1.

In the parallel simulations only one core of each node could be used. Indeed the amount of data to be stored in each RAM memory, for each MPI process, is very large; consequently very large memory size is needed, in fact greater than 8 GB. On the other hand, the advantage of using 64 bits machines and compilers is the capability to use RAM with size greater than 4 GB. Moreover, in the application concerning the electrophoresis problem, at each time step the heterogeneous values of the entries of the matrices and of the right-hand sides must be stored; consequently, the previous constraint is increased. Thus, in the case where many cores per node are used, due to disc swapping, the performances of parallel algorithms will decrease. Note that only machines having large memory could be used.

The results of experiments are summarized in Tables 2–8 and Figs. 3–7. Table 2 displays the sequential elapsed time obtained on each cluster. The number of processors used has been limited up to 16 or to 32 according to the duration of the exploitation of the computational codes; moreover, since the overhead damages the performance of the parallel algorithms, additional machines are not necessary. In these tables and figures, comparison of elapsed time for parallel synchronous and asynchronous methods are presented as well as speed-up and efficiency only when one cluster is used (see Table 3).

We have considered several regular meshes: one large mesh[2] and one very large mesh[3]; due to utilization constraints of the grid platform in dedicated exploitation and taking also into account the size of the linear systems to solve we have performed few (three or five) time steps. Two codes have been tested: one corresponding to the purely implicit scheme using Schwarz alternating method on the one hand, and on the other hand, only for the solution of the transport equation, the explicit MPDATA scheme is used, the other boundary value problems of the mathematical model being solved by the Schwarz method.

In Table 2, for the two considered meshes, `suno` cluster in Sophia is the fastest cluster for implicit scheme but `paradent` cluster in Rennes performs slightly better for MPDATA scheme. Therefore, when distant cluster are used, for the comparison between synchronous and asynchronous algorithms, the notion of speed-up and efficiency are difficult to define since we do not know which cluster to choose in order to measure sequential time. So, in this case, we define the parameter τ which measures the ratio of elapsed time between synchronous and asynchronous methods (see Tables 3–7 except Table 5). In all cases, this parameter is significant in order to achieve the comparison of synchronous and asynchronous parallel methods.

Moreover in order to achieve this comparison in Tables 3–8 P denotes the number of processors used.

In the results of Table 3, only one cluster is used, the `suno` one located in Sophia; so the considered sequential elapsed time is the one obtained by using this previous cluster and so, in this context

[1] Message Passing Interface.

[2] $1040 \times 390 \times 52 = 21\,091\,200$ points.

[3] $1200 \times 450 \times 60 = 32\,400\,000$ points

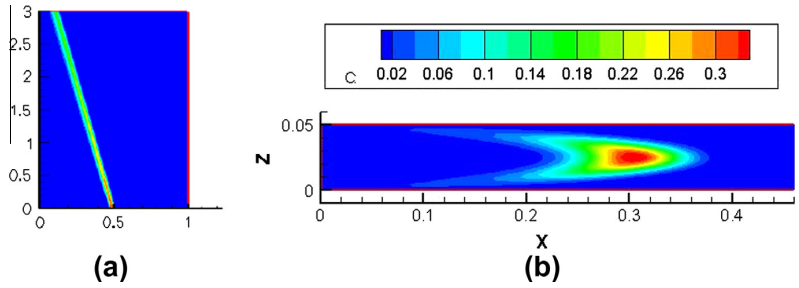


Fig. 2. Computed protein filament (a) and concentration profile (b) in a cut plane.

Table 1
Characteristics of nodes on each site.

Site	Cluster	Processors type	Speed (GHz)	CPU	Core	RAM size (GB)
Sophia	suno	Intel Quad Xeon E5520	2.26	2	8	32
Nancy	griffon	Intel Quad Xeon L5420	2.5	2	8	16
Rennes	paradent	Intel Quad Xeon L5420	2.5	2	8	32

Table 2
Elapsed time (s) on clusters suno (Sophia), griffon (Nancy) and paradent (Rennes) with sequential algorithm.

21 091 200 Mesh points and three time steps		32 400 000 Mesh points and five time steps			
Implicit		Implicit		MPDATA	
suno	griffon	suno	paradent	suno	paradent
159 346	187 150	45 959	58 161	60 648	60 474

Table 3
Elapsed time (s) and number of relaxations on cluster suno (Sophia) for 21 091 200 mesh points with purely implicit algorithm and three time steps.

P	Sync.	Rel.	spdUp	Eff	Async.	Rel.	spdUp	Eff	τ
2	123 706	9 159 332	1.29	0.64	81 637	9 213 658	1.95	0.97	1.51
4	84 208	9 184 454	1.89	0.47	42 303	9 341 774	3.77	0.94	1.99
8	44 071	9 127 594	3.61	0.45	22 501	9 506 778	7.08	0.88	1.96
16	23 561	9 774 530	6.76	0.42	13 960	10 735 172	11.41	0.71	1.69
32	11 230	9 876 916	14.19	0.44	7986	11 191 640	19.95	0.62	1.41

Table 4
Elapsed time (s) on clusters suno (Sophia) and griffon (Nancy) for 21 091 200 mesh points with purely implicit algorithm and three time steps.

P	Synchronous	Asynchronous	τ
2	191 491	97 019	1.97
4	84 917	50 670	1.69
8	53 550	27 605	1.94
16	34 195	23 060	1.48

Table 5
Elapsed time comparison of purely implicit and MPDATA algorithms on clusters suno (Sophia) and griffon (Nancy) for 21 091 200 mesh points and three small time steps.

P	Synchronous		Asynchronous	
	Implicit	MPDATA	Implicit	MPDATA
1	37 537	18 637	37 537	18 637
2	20 281	20 524	11 551	11 596
4	12 288	12 248	7142	7089
8	7638	7551	5462	5362

we can define the speed-up and the efficiency of the parallel algorithms. In this case, as we can see in Fig. 3 and in Table 3, note that another feature of the parallel experiments consists in the fact that

Table 6
Elapsed time (s) on clusters suno (Sophia) and paradent (Rennes) for 32 400 000 mesh points with purely implicit method and five small time steps.

P	Synchronous	Asynchronous	τ
2	61 687	34 847	1.77
4	36 656	20 787	1.76
8	23 630	14 701	1.61
16	16 013	12 519	1.28
32	13 928	12 354	1.13

Table 7
Elapsed time (s) on clusters suno (Sophia) and paradent (Rennes) for 32 400 000 mesh points with explicit MPDATA method for the transport equation and five small time steps.

P	Synchronous	Asynchronous	τ
2	62 729	35 517	1.77
4	36 367	21 492	1.69
8	22 769	14 512	1.57
16	17 959	12 787	1.40
32	12 406	12 353	1

in the asynchronous case, the speed-up and the efficiency are clearly better than in the synchronous case.

Table 8
Comparison of average communication time compared to elapsed time on cluster `suno` (Sophia) and cluster `paradent` (Rennes) for 32 400 000 mesh points with purely implicit algorithm and five time steps.

P	Synchronous			Asynchronous		
	Elapsed time	Comm. time	%	Elapsed time	Comm. time	%
2	63 459	25 130	39.60	34 800	663	1.91
4	37 643	14 322	38.05	21 516	448	2.08
8	24 630	8002	32.50	14 703	294	2.00
16	15 746	2097	13.32	12 764	415	3.25
32	15 287	1563	10.22	11 953	345	2.88

In the explicit MPDATA method, the time step size is taken very small for ensuring numerical stability of the scheme; indeed, in this case the time step size must be chosen 100 time less than the one considered when using the purely implicit scheme. So, in this case, more time steps are performed in the simulation process. For this kind of method, and for the previous value of the time step size the Courant–Friedrich–Levy (CFL) condition is verified (see [14]). On the other hand, when we use the purely implicit scheme, thanks to unconditional numerical stability, we can choose greater step size.

We can note that the global electrophoresis problem is very hard to solve numerically. Indeed, when the implicit scheme is used, among the seven linear algebraic systems to solve, three of them are very ill-conditioned; it is particularly true for the two linear algebraic systems corresponding to the two corrector steps for the solution of the Navier–Stokes equation and also for the solution of the potential equation. This fact is still true when the transport equation is solved by using the explicit MPDATA scheme for the transport equation. Remark also that, for each time step, the solution of each linear system requires similar running times.

In the implementation of the Schwarz alternating method, two or more subdomains are assigned to each processor; thus, on each processor, this assignation allows a behaviour of the parallel algorithm as close as possible to a multiplicative Schwarz alternating method. From a practical point of view, parallel synchronous and asynchronous algorithms are more efficient on such subdomain distribution. Thus, the proposed MPI implementation of the Schwarz alternating method in distributed memory multiprocessor is more efficient.

Tables 3–7 and Figs. 3–6 show clearly the benefits of using parallel algorithms. Note that the asynchronous parallel iterative algorithms are more efficient than the synchronous ones.

Tables 5–7 show that, using three or five identical small time steps, the purely implicit algorithm and the method using the explicit MPDATA scheme for the transport equation practically lead to the same performances. Nevertheless, due to the small size of the considered time step, the use of an explicit time marching scheme increases finally the global elapsed time.

It can be noted (see [25]) that, in asynchronous parallel experiments, the number of relaxations necessary to reach convergence is slightly greater than the one necessary when parallel synchronous schemes are used. This is a well-known drawback of

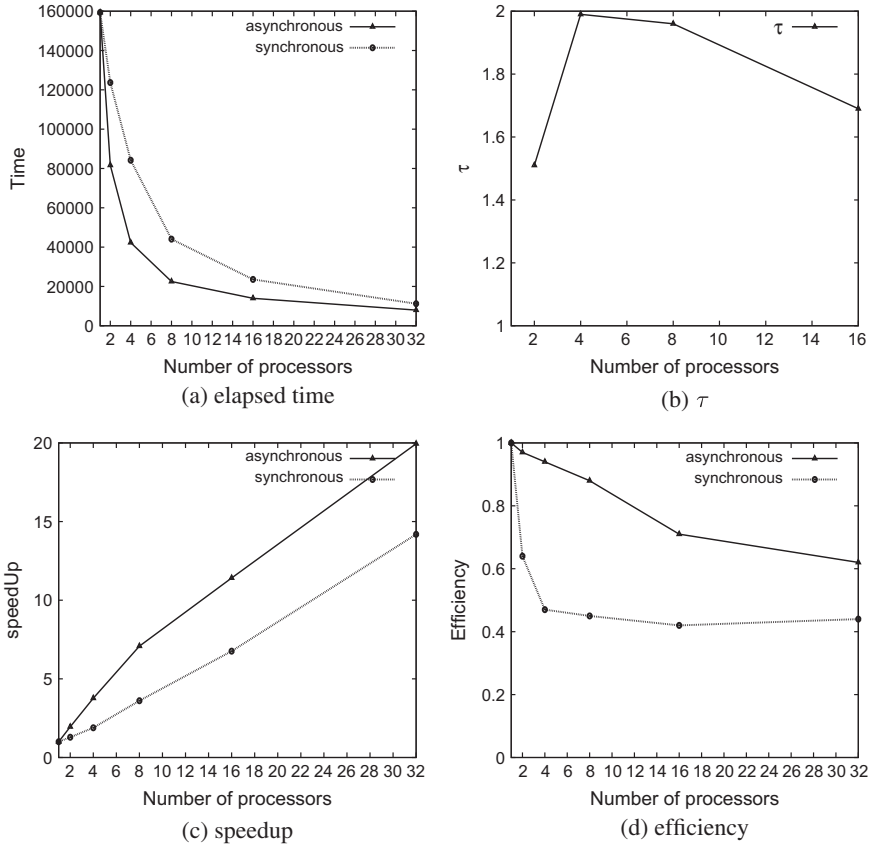


Fig. 3. Elapsed time on cluster `suno` (Sophia) for 21 091 200 mesh points with purely implicit algorithm and three time steps.

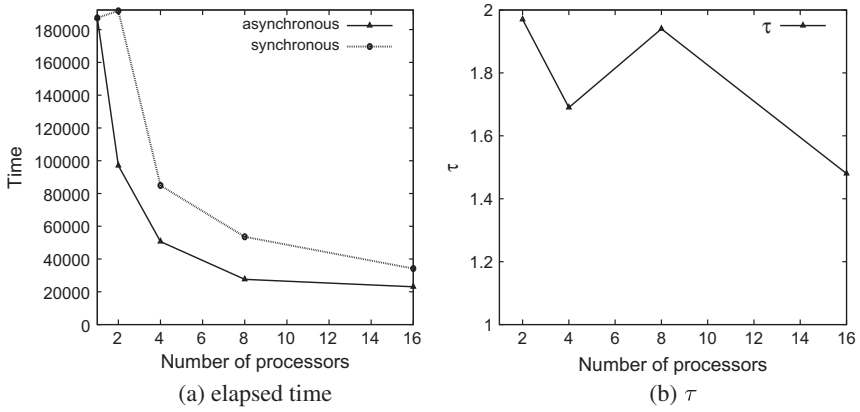


Fig. 4. Results on clusters `suno` (Sophia) and `griffon` (Nancy) for 21 091 200 mesh points with purely implicit algorithm and three time steps.

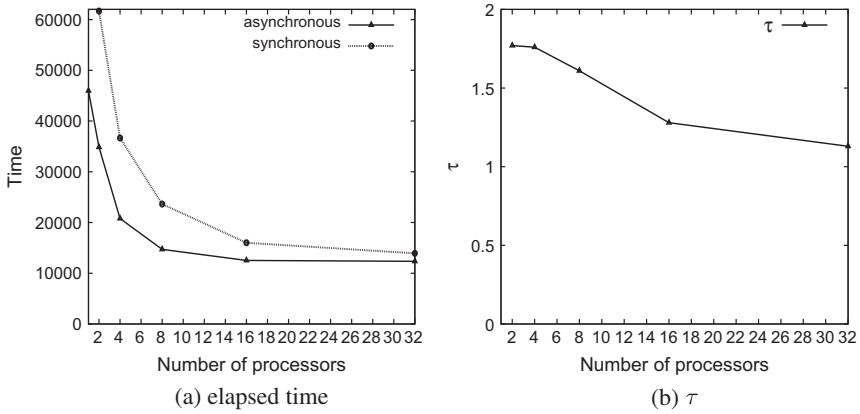


Fig. 5. Results on clusters `suno` (Sophia) and `paradent` (Rennes) for 32 400 000 mesh points with purely implicit method and five small time steps.

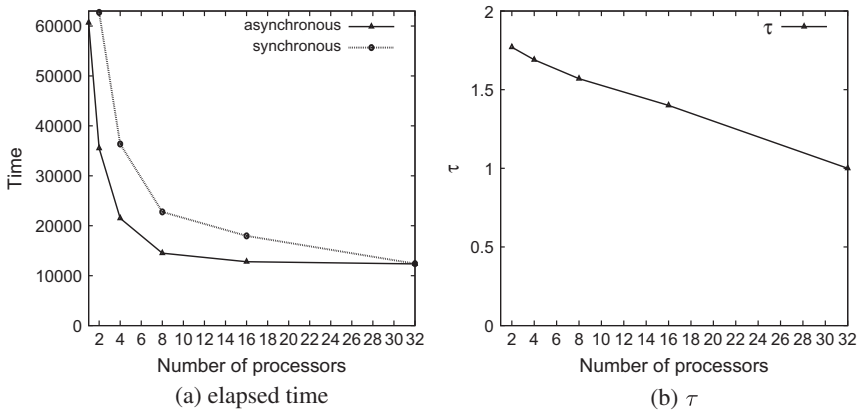


Fig. 6. Results on clusters `suno` (Sophia) and `paradent` (Rennes) for 32 400 000 mesh points with explicit MPDATA method for the transport equation and five small time steps.

asynchronous iterative schemes, due to the use of possibly delayed components. It turns out that the overhead generated by additional relaxations in the case of asynchronous algorithms is smaller than the synchronization overhead combined with processor idle time of parallel schemes of computation. Note that, when the number of processors increases, the number of relaxations necessary to reach convergence increases too. Nevertheless these additional relaxations improve the solution's accuracy and the numerical quality of the computations. Furthermore, despite higher number of relaxations, elapsed time of asynchronous scheme is less than the one obtained when synchronous scheme is used. This asynchronism is an efficient way to deal with communication overhead

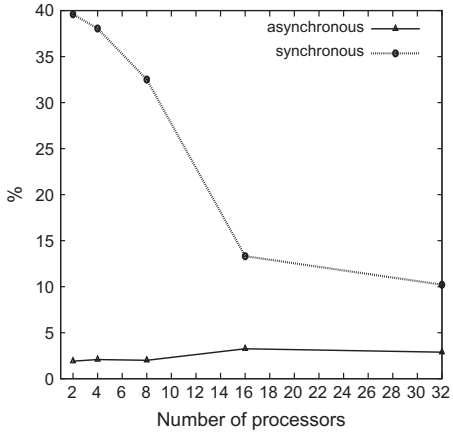


Fig. 7. Communication times on clusters `suno` (Sophia) and `paradent` (Rennes) for 32 400 000 mesh points with purely implicit method for the transport equation and five small time steps.

and load unbalance. Moreover, the efficiency of the synchronous algorithms decreases faster than the efficiency of asynchronous algorithms when the number of processors increases. The lack of synchronization points and the use of current values of the components of the iterate vector induce better performances for the asynchronous Schwarz alternating method when several processors are used.

Furthermore in Figs. 3–6 and in Tables 3–7 (except Table 5), the value of the parameter τ show clearly that the performances of parallel asynchronous algorithms are better than the ones obtained when parallel synchronous algorithms are used. In fact, the behaviour of the parallel schemes expressed in term of number of relaxations, as well as for the synchronous and the asynchronous methods, are similar, in the case of use of a single cluster or two distant different clusters. Finally, note that the use of asynchronous parallel schemes is very interesting, since the weight of synchronizations is low and, as a consequence, the obtained elapsed times are lower. Moreover Table 8 and Fig. 7 shows clearly the impact of the communications on the behaviour of the iterative parallel algorithm and the fact that such communications degrade the performances of the synchronous methods.

5. Conclusion

In the present study we have presented the parallel numerical solution of the electrophoresis problem, modelled by the coupling of several boundary value problems. Due to the fact that the algebraic systems to solve are very ill-conditioned and also that the computational time is very high, this kind of problem is very difficult to solve by sequential algorithms. In the presented study, we have considered two kinds of time marching schemes for the protein transport equation: a purely implicit scheme and an explicit one which give suitable results. The stability and the convergence of both methods have been ensured. Due to the weight of synchronizations, parallel asynchronous experiments on the Grid platform show clearly as well as on local cluster or on distant clusters the interest of such methods. So parallel asynchronous iterative algorithms, more particularly on Grid platform, are appropriate ways to solve this complex problem.

Acknowledgement

This study has been made possible with the support of Grid'5000 platform.

References

[1] Clifton MJ, Roux-de-Balmann H, Sanchez V. Electrohydrodynamic deformation of the sample stream in continuous-flow electrophoresis with an AC electric field. Can J Chem Eng 1992;70:1055–62.
[2] Clifton MJ. Numerical simulation of protein separation by continuous-flow electrophoresis. Electrophoresis 1993;14:1284–91.
[3] Hoffman KH, Zou J. Parallel efficiency of domain decomposition methods. Parallel Comput 1993;19:1375–91.
[4] Miellou JC, El Baz D, Spiteri P. A new class of iterative algorithms with order intervals. Math Comput 1998;67:237–55.
[5] El Baz D, Frommer A, Spitéri P. Asynchronous iterations with flexible communication: contracting operators. J Comput Appl Math 2005;176:91–103.
[6] Ortega JM, Rheinboldt WC. Iterative solution of nonlinear equations in several variables. New York: Academic Press; 1970.
[7] Giraud L, Spiteri P. Résolution parallèle de problèmes aux limites non linéaires. M.2 AN 1991;25:73–100.
[8] Miellou JC, Spiteri P. Un critère de convergence pour des méthodes générales de point fixe. M.2 AN 1985;19(4):645–69.
[9] Issa RI. Solution of the implicitely discretized fluid flow equations by operator splitting. J Comput Phys 1986;62:40–65.
[10] Patankar SV. Numerical heat transfer and fluid flow. McGraw Hill; 1980.
[11] Spitéri P, Boisson HC. Subdomain predictor–corrector algorithms for solving the incompressible Navier–Stokes equation. In: Kaper HG, Garbey M, editors. Asymptotic and numerical methods for partial differential equations with critical parameters. NATO ASI series, vol. 384. Kluwer Academic Publishers; 1993. p. 335–47.
[12] Miellou JC. Asynchronous iterations and order intervals. In: Cosnard M et al., editors. Parallel algorithms. North Holland; 1985. p. 85–96.
[13] Miellou JC, Spiteri P. Itérations asynchrones avec segments d'ordre. Méthodes numériques performantes, Contrat ATP – CNRS no. 8096; 1984.
[14] Chau M, Spiteri P, Boisson HC. Parallel numerical simulation for the coupled problem of continuous flow electrophoresis. Int J Numer Methods Fluids 2007;55:945–63.
[15] Chau M, Spiteri P, Guivarch R, Boisson HC. Parallel asynchronous iterations for the solution of a 3D continuous flow electrophoresis problem. Comput Fluids 2008;37(9):1126–37.
[16] Evans DJ, Deren W. An asynchronous parallel algorithm for solving a class of nonlinear simultaneous equations. Parallel Comput 1991;17:165–80.
[17] Frommer A, Szyld DB. Weighted max norms, splittings, and overlapping additive Schwarz iterations. Numer Math 1999;83:259–78.
[18] Mathew TP. Uniform convergence of the Schwarz alternating method for solving singularly perturbed advection–diffusion equations. SIAM J. Numer Anal 1998;35:1663–83.
[19] Bru R, Pedroche F, Szyld DB. Overlapping additive and multiplicative Schwarz iterations for *H*-matrices. Linear Algebra Appl 2004;393C:91–105.
[20] Chazan D, Miranker W. Chaotic relaxation. Linear Algebra Appl 1969;2:199–222.
[21] Spiteri P, Miellou JC, El Baz D. Parallel asynchronous Schwarz and multisplitting methods for a nonlinear diffusion problem. Numer Algor 2003;65:74–84.
[22] Spiteri P, Guivarch R, El Baz D, Chau M. Parallelization of subdomain methods with overlapping for linear and non linear convection–diffusion problems. In: Proceedings of the 11th int conference PDP 2003, Gennes. IEEE; 2003. p. 341–8.
[23] Chau M, El Baz D, Guivarch R, Spiteri P. MPI implementation of parallel subdomain methods for linear and non linear convection–diffusion problems. JPDC 2007;67:581–91.
[24] Smolarkiewicz PK. A fully multidimensional positive definite advection transport algorithm with small implicit diffusion. J Comput Phys 1984;54:325–62.
[25] Garcia T, Chau M, Spiteri P. Computation of protein separation using a grid environment. In: Ivanyi P, Topping BHV, editors. Proceedings of the second international conference on parallel, distributed, grid and cloud computing for engineering, Ajaccio, Corsica, France. Stirlingshire (UK): Civil-Comp Press; 2011. [Paper 82].

5.9 Article 9

Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 93 (2005) 307–313
Information Processing Letters
www.elsevier.com/locate/ipl

A Coarse-Grained Multicomputer algorithm for the detection of repetitions

Thierry Garcia, David Semé *

LaRIA: Laboratoire de Recherche en Informatique d'Amiens, Université de Picardie Jules Verne, CURI, 5, rue du Moulin Neuf, 80000 Amiens, France

Received 17 October 2003

Available online 11 January 2005

Communicated by F. Dehne

Abstract

The paper presents a Coarse-Grained Multicomputer algorithm that solves the problem of detection of repetitions. This algorithm can be implemented in the CGM model with P processors in $O(N^2/P)$ in time and $O(P)$ communication steps. It is the first CGM algorithm for this problem. We present also experimental results showing that the CGM algorithm is very efficient.

Keywords: Parallel algorithms; Coarse-Grained Multicomputers; Dynamic programming; String matching

1. Introduction

This paper describes a Coarse-Grained Multicomputer solution for the detection of repetitions. Strings of symbols containing no consecutive occurrences of the same pattern have attracted the attention of researchers in diverse fields for a long time. For example, repetitions play a crucial role in many domains [3,19] such as formal language theory, term rewriting, data compression and computational molecular biology. Perhaps their first appearance dates back to the work of Thue [23], who is generally credited with the discovery of arbitrary long streams of symbols from a finite alphabet that do not contain any "square" substrings, i.e., subpatterns formed by the concatenation of some substring with itself.

In recent years, the study of such "square-free" strings has been found relevant to automata and formal language theory, algebraic coding and more generally in systems theory and combinatorics, and we shall make no attempt to refer to the existing copious literature. Suffice it to mention that papers have been devoted to the construction of arbitrarily long square-free (as well as other related repetition-constrained)

* Corresponding author.
E-mail addresses: garcia@laria.u-picardie.fr (T. Garcia), seme@laria.u-picardie.fr (D. Semé).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2004.12.004

strings [18] over alphabets of fixed cardinality. In a related endeavor, the complementary notions of periodicity and overlaps of strings have been extensively investigated and still are an active research subject (see Duval [11] for an extensive bibliography).

In the framework of pattern matching [2], some classic results on string periodicity [13] have been used to develop clever techniques for the detection of assigned patterns in text-strings in time linear in the string length [7].

The problem of the efficient recognition of the occurrence of substring squares in a string X stems quite naturally from the preceding remarks, and it is certainly relevant to a variety of practical applications as well [1]. $O(N^2)$ time algorithms can be quickly developed on the basis of existing pattern matching techniques and tools (N is the length of the string X). Recently, an $O(N \log N)$ algorithm has been proposed to determine whether a given text-string over a finite alphabet contains a repetition [21]. Crochemore [5] developed an $O(N \log N)$ algorithm to determine all repetitions in a text-string x. Crochemore's approach essentially relies on the well-known minimization algorithm for finite state automata [2] and exploits the theoretical bound of $N \log N$ repetitions in a string [20] as a terminating condition. Apostolico and Negro [4] presented a systolic algorithm for the detection of repetitions running in time $O(N)$ (with $4N - 3$ steps) and using N processors, whereas the linear systolic array for the same problem presented in [22] requires $(5N/4) - 1$ steps on $N/4$ processors. The only known algorithm for the PRAM model can be obtained by simulating the linear systolic array on the PRAM. A constant time parallel detection of repetitions with a BSR (Broadcasting with Selective Reduction) solution has been developed by [10]. All known parallel algorithms require a work of $O(N^2)$.

In this paper we show an algorithm which solve this problem in the CGM model but this algorithm is not work optimal. Indeed, the optimality in work to solve this problem is $O(N \log N)$. Here, our algorithm has a work of $O(N^2)$ but our goal is to obtain a CGM algorithm from a systolic solution having a work of $O(N^2)$. This is a contribution to a more larger work on the adaptation of linear systolic solutions in CGM model.

In recent years several efforts have been made to define models of parallel computation that are more realistic than the classical PRAM models. In contrast of the PRAM, these new models are coarse grained, i.e., they assume that the number of processors P and the size of the input N of an algorithm are orders of magnitudes apart, $P \ll N$. By the precedent assumption these models map much better on existing architectures where in general the number of processors is at most some thousands and the size of the data that are to be handled goes into millions and billions.

This branch of research got its kick-off with Valiant [24] introducing the so-called Bulk Synchronous Parallel (BSP) machine, and was refined in different directions, for example, by Culler et al. [6], LogP, and Dehne et al. [9], CGM extensively studied in [8,12, 14–16].

CGM seems to be the best suited for a design of algorithms that are not too dependent on an individual architecture.

We summarize the assumptions of this model:

- all algorithms perform in so-called supersteps, that consist of one phase of interprocessor communication and one phase of local computation,
- all processors have the same size $M = O(N/P)$ of memory ($M > P$),
- the communication network between the processors can be arbitrary.

The goal when designing an algorithm in this model is to keep the individual workload, time for communication and idle time of each processor within $T/s(P)$, where T is the runtime of the best sequential algorithm on the same data and $s(P)$, the speedup, is a function that should be as close to P as possible. To be able to do so, it is considered as a good idea the fact of keeping the number of supersteps of such an algorithm as low as possible, preferably $O(M)$.

As a legacy from the PRAM model it is usually assumed that the number of supersteps should be polylogarithmic in P, but there seems to be no real world rationale for that. In fact, algorithms that simply ensure a number of supersteps that are a function of P (and not of N) perform quite well in practice, see Goudreau et al. [17].

The paper is organized as follows. In Section 2, we present the detection of repetition problem. The CGM solution of the problem is described in Section 3. Section 4 presents some experimental results and the conclusion ends the paper.

2. The detection of repetitions problem

2.1. Basic definitions and notations

Let A be a finite alphabet and A^* be the free monoid generated by A. Let A^+ the free semigroup generated by A. A string $X \in A^+$ is fully specified by writing $X = x_1 \dots x_n$, where $x_i \in A$ $(1 \leqslant i \leqslant n)$.

Definition 1. A factor of X is a substring of X and its starting position in $\{1, 2, \dots, n\}$. The notation $X[k : l]$ is used to denote the factor of $X : x_k x_{k+1} \dots x_l$. A left (right) factor of X is a prefix (suffix) of X.

Definition 2. A string $X \in A^+$ is primitive if setting $X = u^k$ implies $u = X$ and $k = 1$.

Definition 3. A square in a string X is any nonempty substring of X in the form uu.

Definition 4. String X is square-free if no substring of X is a square. Equivalently, X is square-free if each substring of X is primitive.

Definition 5. A repetition in X is a factor $X[k : m]$ for which there are indices l, d $(k < d \leqslant l \leqslant m)$ such that:

(1) $X[k : l]$ is equivalent to $X[d : m]$,
(2) $X[k : d - 1]$ corresponds to a primitive word,
(3) $X_{l+1} \neq X_{m+1}$.

Definition 6. p is a period of a repetition $X[k : m]$ of X if $x_i = x_{i+p}$ $(\forall i = k, k + 1, \dots, |X[k : m]| - p)$.

Therefore, $1 \leqslant p \leqslant |X[k : m]|/2$.

2.2. Dynamic programming approach

Let X be a word of length N and p be a period, our goal is to detect all repetitions in X for a period p using Definitions 5 and 6. Table 1 gives an example of all periods of the word X of length $N = 16$.

The idea is to cut this table in P columns of N/P elements, with $P \ll N$ and $P \neq 0$. Table 2 gives an example of this cut using $P = 4$ processors.

Fig. 2 is composed of square parts of two equal parts of X. Each square part is composed of a low triangular part and a high triangular part.

The dynamic programming approach is based on different systolic solutions [25]. The following section describes two sequential algorithms based on the previous definitions, which computes respectively low triangular parts and high triangular parts.

2.3. Sequential algorithms

Algorithm 1 detects all repetitions for periods p (with $1 \leqslant p \leqslant N - 1$) from the first character to the

Table 1
Values of all periods p for $X = abbbbaacacad$

		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
		a	b	b	b	b	a	a	c	a	c	a	d	d	a	c	a
0	a																
1	b	1															
2	b	2	1														
3	b	3	2	1													
4	b	4	3	2	1												
5	a	5	4	3	2	1											
6	a	6	5	4	3	2	1										
7	c	7	6	5	4	3	2	1									
8	a	8	7	6	5	4	3	2	1								
9	c		8	7	6	5	4	3	2	1							
10	a			8	7	6	5	4	3	2	1						
11	d				8	7	6	5	4	3	2	1					
12	d					8	7	6	5	4	3	2	1				
13	a						8	7	6	5	4	3	2	1			
14	c							8	7	6	5	4	3	2	1		
15	a								8	7	6	5	4	3	2	1	

Table 2
Cut-table for $X = abbbbaacacad$

		0 a	1 b	2 b	3 b	4 b	5 a	6 a	7 c	8 a	9 c	10 a	11 d	12 d	13 a	14 c	15 a
0	a																
1	b	1															
2	b	2	1														
3	b	3	2	1													
4	b	4	3	2	1												
5	a	5	4	3	2	1											
6	a	6	5	4	3	2	1										
7	c	7	6	5	4	3	2	1									
8	a	8	7	6	5	4	3	2	1								
9	c		8	7	6	5	4	3	2	1							
10	a			8	7	6	5	4	3	2	1						
11	d				8	7	6	5	4	3	2	1					
12	d					8	7	6	5	4	3	2	1				
13	a						8	7	6	5	4	3	2	1			
14	c							8	7	6	5	4	3	2	1		
15	a								8	7	6	5	4	3	2	1	

$(N - p)$th character of the word X. At the end, two tables Ic and Lc are created or modified. These tables represent respectively starting and ending positions of a repetition in the word X for a period p. The different repetitions are given dynamically during the execution. So, the function $R(Ic, p, Lc)$ is used to write the repetition starting at the position Ic with a period p of length Lc.

Algorithm 2 detects all repetitions for periods $1 \leqslant p \leqslant N$ computed from the $(N - p + 1)$th character to the Nth character of the word X. At the end, two tables Ic and Lc are created or modified. These tables represent respectively starting and ending positions of a repetition in the word X for a period p. The different repetitions are given in real time during the execution.

2.4. CGM algorithm

Algorithm REPET_CGM presents the CGM solution of the Detection of Repetition problem. Each processor *num* ($1 \leqslant num \leqslant P$) has the *num*th partition of N/P elements of the input sequence X. The following CGM algorithm presents the program of each processor *num*. This CGM algorithm uses sequential Algorithms 1 and 2 described before as local computation parts. Two functions are used for communication rounds:

send (X, num): a vector X is sent to the processor *num*,

```
for (p = 1) to (p = N − 1)
  for (i = 1) to (i = N − p)
    if (X[i] ≠ X[i + p]) then
        Ic[p] = Ic[p] + Lc[p] − p + 1
      Lc[p] = p
    else
        Lc[p] = Lc[p] + 1
      if (Lc[p] ⩾ 2p) then
        R(Ic[p], p, Lc[p])
      endif
    endif
  endfor
endfor
```

Algorithm 1.

```
for (p = 1) to (p = N)
  for (i = N − p + 1) to (i = N)
    if (X[i] ≠ X[i + p]) then
        Ic[p] = Ic[p] + Lc[p] − p + 1
        Lc[p] = p
    else
      Lc[p] = Lc[p] + 1
    if (Lc[p] ⩾ 2p) then
      R(Ic[p], p, Lc[p])
      endif
    endif
  endfor
endfor
```

Algorithm 2.

```
for (ii = 0) to (ii ⩽ (P − 1) − num)
  if (ii = 0) and (num ≠ 0) then send (X, 0)
  if (ii ≠ 0) and (num = 0) then receive (Y, ii)
  if (num ≠ 0) then receive (Y, num − 1)
  if (ii ≠ 0) then
    Local computation using Algorithm 2
    send (Y, num + 1)
    send (Lc, num + 1)
    send (Ic, num + 1)
  endif
  if (num ≠ 0) then
    receive (Lc, num − 1)
    receive (Ic, num − 1)
  endif
  Local computation using Algorithm 1
endfor
```

Algorithm REPET_CGM.

receive (Y, num): a vector Y is received from the processor *num*.

2.5. Communication rounds

The communication strategy is described in Fig. 1. The number of communication rounds is $2(P-1)$.

Proof. Each processor sends one message to the processor 0. So we have $(P-1)$ communication rounds. After that, the message from the last processor received by the processor zero should go through all other processors. This implies $(P-1)$ communication rounds mere. Thus, the total number of communication rounds is $2(P-1)$. This ends the proof. □

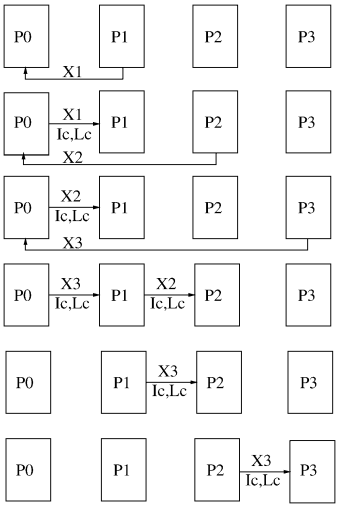


Fig. 1. Communication round (for 4 processors).

2.6. Complexity

The complexity of the CGM algorithm is $O(N^2/P)$.

Proof. The complexity of sequential algorithms used as local computation in our CGM algorithm is $O((N/P)^2)$. A processor must execute $N/2$ periods. Each processor has N/P characters and compares $(N/2 \times P/N) = P/2$ characters. When a processor has finished the execution of its local computations and as the first period begins to the second character, the total complexity is $(P/2 \times 2 + 1) \times N^2/P = (P + 1) \times N^2/P)$. This implies a complexity of $O(N^2/P)$. This ends the proof. □

Then, the work of our CGM solution is $O(N^2)$ as for the systolic solution.

3. Experimental results

We have implemented these programs in C language using MPI communication library and tested them on a multiprocessor Celeron 466 MHz platform running LINUX. The communication between processors is performed through an Ethernet switch.

Tables 3 and 4 present respectively the total running times and the communication times (in seconds), for each configuration of 1, 2, 4, 8 and 16 processors.

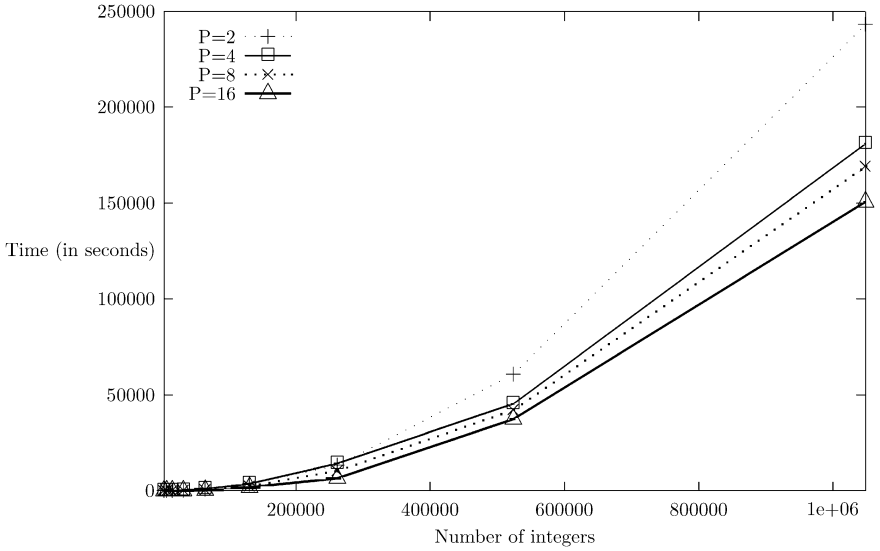


Fig. 2. Total running times (in seconds) for different numbers of data items. The curves represent configurations of 2, 4, 8 and 16 processors, respectively.

Table 3
Total running times (in seconds) for each configuration of 1, 2, 4, 8 and 16 processors, respectively

N	$P=1$	$P=2$	$P=4$	$P=8$	$P=16$
4096	4.15	3.11	3.14	1.90	1.19
8192	16.55	12.41	13.36	7.68	5.13
16384	66.10	49.57	53.27	31.10	20.81
32768	266.09	199.57	213.87	124.46	83.85
65536	1086.78	815.09	868.44	572.94	335.52
131072	4376.22	3282.16	3512.33	2307.40	1624.47
262144	17536.28	13152.21	14162.64	10259.82	6569.23
524288	81061.04	60795.78	45294.67	42242.09	37367.71
1048576	324395.75	243296.81	181163.14	169284.73	150588.20

Table 4
Communication times (in seconds) for each configuration of 2, 4, 8 and 16 processors, respectively

N	$P=2$	$P=4$	$P=8$	$P=16$
4096	0.004	0.010	0.037	0.038
8192	0.007	0.015	0.037	0.042
16384	0.013	0.061	0.039	0.055
32768	0.025	0.088	0.097	0.085
65536	0.053	0.144	0.177	0.179
131072	0.104	0.221	0.508	0.611
262144	0.203	0.429	1.005	2.128
524288	0.404	0.847	2.014	4.367
1048576	0.809	1.686	4.064	8.852

The communication times correspond to send and receive N/P characters by a processor to another using the communication rounds described before. Here, we have $N = 2^k$ where k is an integer such that $12 \leqslant k \leqslant 20$.

Table 4 shows that communication times increase proportionally of the number of processors. But Table 3 shows that total running times, for a fixed number of data items, decrease when the number of processors increase. Then, the communication times are much more lower than the computation times. This leads that there is more important to improve local computations than communication rounds.

4. Concluding remarks

We presented a Coarse-Grained Multicomputer algorithm that solves the detection of repetitions prob-

lem. This algorithm can be implemented in the CGM model with P processors with a time complexity of $O(N^2/P)$ and $O(P)$ communication rounds. This is the first CGM algorithm for this problem.

The paper presents also experimental results showing that the CGM algorithm is very efficient and that it is interesting to increase the number of processors when the input sequence X is very long.

We should now implement this algorithm on another platform in order to show its portability. Our goal was to develop a CGM algorithm from a systolic solution in order to show that a linear systolic solution can be implemented in the CGM model with the same work. It will be interesting to develop another CGM algorithm for the same problem using the optimal sequential solution as local computation.

References

[1] D.G. Lainiotis, A. Apostolico, Fast applications of suffix trees, in: N.S. Tzannes (Ed.), Advances in Control, D. Reidel, Dordrecht, 1974.

[2] A.V. Aho, J.E. Hopcroft, J.D. Ullman, The Design and Analysis of Computer Algorithms, Addison–Wesley, Reading, MA, 1974.

[3] A.V. Aho, Algorithms for finding patterns in strings, in: J. Van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier, Amsterdam, 1990.

[4] A. Apostolico, A. Negro, Systolic algorithm for string manipulations, IEEE Trans. Comput. c-33 (4) (1984) 515–520.

[5] M. Crochemore, An optimal algorithm for computing the repetitions in a word, Inform. Process. Lett. 12 (5) (1981) 244–250.

[6] D. Culler, R. Karp, D. Patterson, A. Sahay, K. Schauser, E. Santos, R. Subramonian, T. Von Eicken, Log "p": towards a realistic model of parallel computation, in: Proc. 4th ACM SIGPLAN Symp. on Principles and Practices of Parallel Programming, 1996, pp. 1–12.

[7] J.H. Morris, D.E. Knuth, V.R. Pratt, Fast pattern matching in strings, SIAM J. Comput. 6 (1) (1977) 323–350.

[8] F. Dehne, X. Deng, P. Dymond, A. Fabri, A. Khokhar, A randomized parallel 3d convex hull algorithm for coarse grained multicomputers, in: Proc. 7th ACM Symp. on Parallel Algorithms and Architectures, 1995, pp. 27–33.

[9] F. Dehne, A. Fabri, A. Rau-Chaplin, Scalable parallel computational geometry for coarse grained multicomputers, Int. J. Comput. Geom. 6 (3) (1996) 379–400.

[10] E. Delacourt, J.F. Myoupo, D. Semé, A constant time parallel detection of repetitions, Parallel Process. Lett. 9 (1) (1999) 81–92.

[11] J.P. Duval, Sur la périodicité des mots, Thèse de Docteur du 3me cycle, Faculty of Sciences, University of Rouen, 1978.

[12] A. Ferreira, N. Schabanel, A randomized bsp/cgm algorithm for the maximal independent set problem, Parallel Process. Lett. 9 (3) (1999) 411–422.

[13] N.J. Fine, H.S. Wilf, Uniqueness theorems for periodic functions, Proc. Amer. Math. Soc. 16 (1965) 109–114.

[14] T. Garcia, J.F. Myoupo, D. Semé, A work-optimal CGM algorithm for the longest increasing subsequence problem, in: Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA'01), 2001.

[15] T. Garcia, J.F. Myoupo, D. Semé, A coarse-grained multicomputer algorithm for the longest common subsequence problem, in: Proc. 11th Euromicro Conference on Parallel Distributed and Network based Processing (PDP'03), 2003.

[16] T. Garcia, D. Semé, A coarse-grained multicomputer algorithm for the longest repeated suffix ending at each point in a word, in: Proc. 2003 Internat. Conf. on Computational Science and its Application (ICCSA'03), 2003.

[17] M. Goudreau, S. Rao, K. Lang, T. Suel, T. Tsantilas, Towards efficiency and portability: Programming with the bsp model, in: Proc. 8th Annual ACM Symp. on Parallel Algorithms and Architectures (SPAA'96), 1996, pp. 1–12.

[18] M.A. Harrison, in: Introduction to Formal Language Theory, Addison–Wesley, Reading, MA, 1978, pp. 36–40.

[19] J. Van Leeuwen (Ed.), Handbook of Theoretical Computer Science, Elsevier, Amsterdam, 1993.

[20] A. Lentin, M.P. Schützenberger, A combinatorial problem in the theory of free monoids, in: Proc. University of North Carolina, 1967, pp. 128–144.

[21] R. Lorentz, M. Main, An $o(n \log n)$ algorithm for finding repetition in a string, T.R. 79-056 Computer Science Department, Washington State University, Pullman, 1979.

[22] J.-F. Myoupo, A. Wabbi, Improved linear systolic algorithms for substrings statistics, Inform. Process. Lett. 61 (1997) 253–258.

[23] A. Thue, Über die gegenseitige Lage gleicher Teile gewisser Zeichenreichen, Skr. Vid.-Kristiana I. Mat. Naturv. Klasse 1 (1912) 1–67.

[24] L.G. Valiant, A bridging model for parallel computation, Comm. ACM 33 (8) (1990) 103–111.

[25] A. Wabbi, Architectures Paralléles systoliques pour la compression de données et la manipulation des sous-chaines, PhD thesis, Université de Picardie Jules Verne, Faculté de Mathématiques et d'Informatique, Amiens, France, 1998.

5.10 Proceeding 1

2011 IEEE International Parallel & Distributed Processing Symposium

Asynchronous peer-to-peer distributed computing for financial applications

Thierry GARCIA
Université de Toulouse
INP – ENSEEIHT – IRIT
BP 7122, 2 Rue Camichel
F-31071 Toulouse Cedex, France
e-mail: thierry.garcia@enseeiht.fr

Ming CHAU
Advanced Solutions Accelerator
199 Rue de l'Oppidum
F-34170 Castelnau le Lez, France
e-mail: mchau@advancedsolutionsaccelerator.com

The Tung Nguyen, Didier El-Baz
CNRS - LAAS 7 avenue du colonel Roche
F-31077 Toulouse, France
Université de Toulouse; UPS, INSA, INP, ISAE.
e-mail: ttnguyen@laas.fr elbaz@laas.fr

Pierre SPITERI
Université de Toulouse
INP – ENSEEIHT – IRIT
BP 7122, 2 Rue Camichel
F-31071 Toulouse Cedex, France
e-mail: pierre.spiteri@enseeiht.fr

Abstract— **This paper deals with the numerical solution of financial applications, more specifically the computation of American and European options derivatives modelled by boundary values problems. In such applications we have to solve large-scale algebraic linear systems. We concentrate on synchronous and asynchronous parallel iterative algorithms carried out on peer-to-peer networks. The properties of the operators arising in the discretized problem ensure the convergence of the parallel iterative synchronous and asynchronous algorithms. Computational experiments performed on peer-to-peer networks are presented and analyzed.**

Keywords-component; parallel asynchronous algorithms; distributed computing; iterative numerical methods; peer-to-peer networks; European options derivatives; American options derivatives.

I. INTRODUCTION

For the last years financial applications have known great developments. In particular, European and American options derivatives modelled by the classical Black and Scholes equation have known great interest [1]. The problem consists of solving a time dependant boundary value problem defined on an unbounded domain generally included in the three dimensional space. A classical artifice consists of solving the Black and Scholes equation on a bounded domain and to increase the size of the domain for ensuring convergence to the exact solution [2]. So, we have to solve boundary value problems numerically. Taking into account the size of the algebraic systems derived from the discretization process of the Black and Scholes equation, parallel iterative algorithms are prefered.

In this paper we concentrate on asynchronous and synchronous iterative methods implemented on peer-to-peer (P2P) architectures. Recent advances in microprocessors architecture and networks permit one to consider new applications like High Performance Computing (HPC). Therefore, we can identify a real stake at developing new protocols and P2P environments for HPC since this can lead to economic and attractive solutions. Task parallel models and distributed iterative methods for large scale parallel numerical simulation on P2P networks gives rise to numerous challenges like communication management, scalability, heterogeneity and peer volatility on the network (see [3]). Some performance issues can be addressed via distributed asynchronous iterative algorithms (see [4] to [8]). The reader is referred to [5] for the solution of the stationary obstacle problem and to [9] for contribution to the development of a new protocol and an environment for P2P high performance computing. The approach presented in [9] is different from MPICH Madeleine [10] in allowing the modification of internal transport protocol mechanism in addition to switch between networks. Recently, middleware like BOINC [11] or OurGrid [12] have been developed in order to exploit the CPU cycles of computers connected to the network. Those systems are generally dedicated to applications where tasks are independent and direct communications between machines are not needed.

In [5], we have considered a stationary problem solved by a Richardson like method. In the present study we consider an evolutive problem closer and related to the studied financial application, more precisely European and American options derivatives in which each stationary problem derived from the numerical time marching scheme is solved by a parallel block relaxation algorithm, which is more efficient than the Richardson's one. This kind of algorithm corresponds in fact to a subdomain method without overlapping. In addition, due to the overhead of synchronizations the use of distributed asynchronous iterative algorithms [6], [7] and [8] seems well suited to P2P computing.

1530-2075/11 $26.00 © 2011 IEEE
DOI 10.1109/IPDPS.2011.292

The present paper is structured as follows: section II deals with the presentation of the financial application (European and American options derivatives). In section III, we present parallel synchronous, and more generally parallel asynchronous iterative schemes. In section IV, environment for peer-to-peer high performance computing is described. In section V, implementation of parallel iterative algorithms is detailed. Computational experiments are displayed and analysed in section VI, in particular the pertinence to apply such parallel asynchronous iterative methods is presented for the solution of the model problems. Finally concluding remarks are presented in section VII.

II. THE PROBLEM OF OPTIONS DERIVATIVES

The classical Black and Scholes equation is a boundary value problem describing the evolution of call or put options in the field of mathematics of financial derivatives [1]. Among the many descriptions of financial option contracts, we can consider:
- the European option, which may only be exercised at expiry, i.e. when the time τ takes the value T of the final expiry date,
- the American option, which may be exercised at any time prior to expiry.

These two kinds of options are modelled by retrograde time dependent convection – diffusion equations. From a mathematical point of view, the main difference between them consists in the fact that European option is defined on a normed vectorial space while the American option is defined on a closed convex set included in a normed vectorial space; consequently, the determination of American option needs the projection of the computed values on the convex set.

Classically, an European option is modeled by the following time dependent linear equation:

$$\begin{cases} \dfrac{\partial v(\tau,x)}{\partial \tau} + (r - \dfrac{\sigma^2}{2}).\nabla v + \dfrac{\sigma^2}{2}\Delta v - rv = 0, \text{ everywhere in } [0,T]xR^n \\ v(T,x) = \psi(x) \end{cases}$$

where $\psi(x)$= Max(x-K,0) (in the case of call option) or $\psi(x)$ = Max(K-x,0) (in the case of put option); in the above equations, K denotes the exercise price – called classically strike, v denotes the value of the considered option, i.e. a "call" or a "put" option; $v = v(\tau,x)$ is a function of the current value of the underlying asset x and of the time τ. Note also that the considered option also depends on the following parameters:
- r the interest rate,
- σ the volatility of the underlying asset, σ being in fact the instantaneous standard deviation of the price with respect to K fixed beforehand; in fact σ characterizes the uncertainly of the behavior of the option.

Using the same previous notations, the American option is modeled by the following nonlinear equation:

$$\begin{cases} \dfrac{\partial v(\tau,x)}{\partial \tau} + (r - \dfrac{\sigma^2}{2}).\nabla v + \dfrac{\sigma^2}{2}\Delta v - rv \geq 0,\ v(\tau,x) \geq \psi(x), \text{ everywhere in } [0,T]xR^n \\ (\dfrac{\partial v(\tau,x)}{\partial \tau} + (r - \dfrac{\sigma^2}{2}).\nabla v + \dfrac{\sigma^2}{2}\Delta v - rv)(v(\tau,x) - \psi(x)) = 0, \text{ everywhere in } [0,T]xR^n \\ v(T,x) = \psi(x) \end{cases}$$

One of the main difficulty lies on the fact that the Black and Scholes equation is not defined on a bounded domain, but is defined on the unbounded domain R^n, $n \geq 1$. This difficulty overcome by considering the problem defined on a bounded domain $\Omega \subseteq R^n$ with appropriate boundary conditions. Then it can be proved that the solution of the equation defined on the bounded domain Ω converges to the solution of the model problem when the measure of Ω tends to infinity [2]. Another particularity of the problem to solve, is that the value of the option is not known at the initial time $\tau = 0$; only the final value v(T,x) is known. So the problem consists of computing v(0,x).

These previous two particularities can be treated, first by considering problems defined on a bounded large domain Ω and secondly by a change of variable, such that the variable τ is replaced by a variable $t = T - \tau$. Thus, the model problem of the European option is replaced by the following linear problem:

$$\begin{cases} \dfrac{\partial v(t,x)}{\partial t} - (r - \dfrac{\sigma^2}{2}).\nabla v - \dfrac{\sigma^2}{2}\Delta v + rv = 0, \text{ everywhere in } [0,T]x\,\Omega \\ v(0,x) = \psi(x) \\ B.C. \text{ of v(t,x) defined on } \partial\Omega \end{cases}$$

where B.C. describes the boundary conditions on the boundary $\partial\Omega$ of the domain Ω. Practically, the Dirichlet condition where v is fixed on $\partial\Omega$ or the Neumann condition where the normal derivative of v is fixed on $\partial\Omega$ is classically considered.

Concerning the American option, by the same way, we have to solve the following nonlinear problem:

$$\begin{cases} \dfrac{\partial v(t,x)}{\partial t} - (r - \dfrac{\sigma^2}{2}).\nabla v - \dfrac{\sigma^2}{2}\Delta v + rv \geq 0,\ v(t,x) \geq \psi(x), \text{ everywhere in } [0,T]x\Omega \\ (\dfrac{\partial v(t,x)}{\partial t} - (r - \dfrac{\sigma^2}{2}).\nabla v - \dfrac{\sigma^2}{2}\Delta v + rv)(v(t,x) - \psi(x)) = 0, \text{ everywhere in } [0,T]x\Omega \\ v(0,x) = \psi(x) \\ B.C \text{ of } v(t,x) \text{ defined on } \partial\Omega \end{cases}$$

III. NUMERICAL SOLUTION OF THE PROBLEMS OF OPTIONS DERIVATIVES

For the numerical solution of the financial problems, we consider that the temporal part is discretized by implicit or semi – implicit scheme. While, for the spatial part of the problem, the bounded domain $\Omega \subseteq R^3$ is discretized with an uniform mesh. Note also that the spatial differential operators are discretized by appropriate schemes as follows:
- the Laplacian is discretized by using the classical seven points scheme,
- the first derivatives, corresponding to the convection phenomenon, are discretized by using decentered scheme; more precisely, if the coefficient of the considered first derivative is strictly positive, then the forward-difference formula is used, otherwise backward–difference formula is considered.

Thus, if good accuracy is wanted, then the full discretization of the Black and Scholes problem leads to the solution of a very large linear algebraic system at each time step.

Let A denote the discretization matrix of the model problem. It follows from the considered appropriate discretization of the Black and Scholes equations, that the diagonal entries of the matrix A are strictly positive and that the off-diagonal entries are nonpositive. Moreover, the matrix A is strictly diagonally dominant. Thus, A is an M – matrix [13]. This very interesting property of the matrices to invert, at each time step, has a consequence regarding to the behaviour of the parallel iterative algorithms considered in the sequel; indeed the convergence of the iterative scheme is ensured (see [4] and [6] to [8]).

Note also, that due to the large size of the linear algebraic system to invert at each time step, it is necessary to use iterative methods. More precisely, we consider in the present study iterative parallel synchronous and asynchronous block relaxation algorithms studied in [6], [7] and [8], implemented in the present study on peer-to-peer distributed architecture. In the sequel, let us recall the formulation of parallel synchronous and more generally asynchronous block relaxation algorithms for the solution of a large linear algebraic system.

Let us consider the solution of the linear algebraic system associated, for example to the solution of the discretized European option

$$A.V=F$$

where V and F are respectively, a vector whose components approximate the values of the exact solution and the right hand side of the partial differential equation respectively, at each point of the mesh. We consider now a block decomposition from the previous linear algebraic system and associate the following fixed-point mapping:

$$V_i = A_{i,i}^{-1}(F_i - \sum_{j \neq i} A_{i,j} V_j) = \Phi_i(V),\ i = 1,\cdots,m,$$

where m is an integer denoting the number of blocks. This kind of fixed point problem can be considered, at each time step, for the solution of the problem of European option.

For the solution of the discretized American option problem, we have to consider now the projection on a convex set (see [1]) as follows:

$$V_i = \text{Proj}(A_{i,i}^{-1}(F_i - \sum_{j \neq i} A_{i,j} V_j)) = \Phi_i(V),\ i = 1,\cdots,m.$$

Then, by considering the appropriate operator as well as, the European option than the American option, the problems consist of solving the following fixed point problem:

$$\begin{cases} Find\ V^* \text{ such that} \\ V^* = \Phi(V^*) \end{cases} \tag{1}$$

where $V \rightarrow \Phi(V)$ is a fixed point mapping defined in a finite dimensional space. For all V, consider the following block-decomposition of the mapping Φ associated to the parallel distributed implementation:

$$\Phi(V) = (\Phi_1(V), \ldots, \Phi_m(V))$$

We consider the distributed solution of the fixed point problem (1) via a parallel asynchronous block relaxation method defined as follows (see [6] to [8]): let the initial guess $V^{(0)}$ be given and for every $p \in N$ assume that we can get $V^{(1)}, \ldots, V^{(p)}$; then $V^{(p+1)}$ is defined recursively by:

$$V_i^{(p+1)} = \begin{cases} V_i^{(p)} \text{ if } i \notin J(p) \\ \Phi_i(\ldots, V_j^{(s_j(p))}, \ldots) \text{ if } i \in J(p) \end{cases} \tag{2}$$

where $\mathbf{J} = \{J(p)\}$, $p \in N$, is a sequence of nonempty sets such that:

$$\begin{cases} J(p) \subset \{1,\ldots,m\},\ J(p) \neq \emptyset,\ \forall p \in N, \\ \forall i \in \{1,\ldots,m\},\ \text{Card}(\{p \in N \mid i \in J(p)\}) = +\infty \end{cases} \tag{3}$$

and

$$\begin{cases} \forall p \in N,\ \forall j \in \{1,\ldots,m\},\ s_j(p) \in N,\ 0 \leq s_j(p) \leq p, \\ \forall p \in N,\ s_i(p) = p \text{ if } i \in J(p), \\ \forall p \in N,\ \forall j \in \{1,\ldots,m\},\ \lim_{p \to \infty}(s_j(p)) = +\infty. \end{cases} \tag{4}$$

The previous asynchronous iterative scheme models computations that are carried out in parallel without order

nor synchronization and describes in fact a subdomain method without overlapping. Particularly, it permits one to consider distributed computations whereby peers go at their own pace according to their intrinsic characteristics and computational load. The parallelism between the peers is well described by J since J(p) contains the number of components relaxed by each processor on a parallel way while the use of delayed components in (2) permits one to model nondeterministic behavior and does not imply inefficiency of the considered distributed scheme of computation. Note that, theoretically, each component of the vector must be relaxed an infinite number of time. The choice of the relaxed components may be guided by any criterion, and, in particular, a natural criterion is to pick-up the most recently available values of the components computed by the processors.

Remark: The algorithm (2) – (4) describes a computational method where the communications between peers can be synchronous or asynchronous. Among them parallel synchronous methods, when $s(p) \equiv p, \forall p \in N$; moreover if $J(p) = \{1,\ldots, m\}$ and $s(p)=p, \forall p \in N$, then (2) – (4) describes the sequential block Jacobi method while if $J(p) =p.\mathrm{mod}(m) +1$ and $s(p)=p, \forall p \in N$, then (2) – (4) models the sequential block Gauss – Seidel method. So, the previous model of parallel asynchronous algorithm appears like a general model.

For the solution of the evolution Black and Scholes equations, a numerical time marching scheme is implemented and, at each time step, we have to solve a large scale algebraic system by using either parallel synchronous or asynchronous algorithms.

Then, for the solution of stationary problems derived from European and American options derivatives, the convergence of classical synchronous or asynchronous relaxation algorithms has been established by various ways, using contraction techniques (see [6] and [7]) or partial ordering techniques (see [8]). To summarize, as previously said, since the discretization matrix is an M-matrix then thanks to various results established in [6], [7] and [8], the iterative process described by (2) – (4) converges to V^*, for every initial guess $V^{(0)}$. The reader is referred to these previous references for more details. Moreover assume that the algebraic system is split into q blocks, $q \leq m$, corresponding to a coarser subdomain decomposition without overlapping; then using results in [7], it can be shown by using the same arguments, that the parallel asynchronous block relaxation methods converge for this coarser decomposition. Furthermore, if the subdomain decomposition associated with m blocks is a point decomposition, then classical parallel asynchronous block relaxation methods converge for every subdomain coarser decomposition and for every numbering (lexicographical or red-black) of the blocks.

IV. Environment for peer to peer high performance computing

P2PDC [5] is an environment for peer to peer high performance computing that was developed at LAAS-CNRS. Contrarily to existing solutions, P2PDC environment allows directed communications between peers by using the P2PSAP Peer To Peer Self Adaptive communication Protocol (see [9]). The P2PSAP protocol chooses dynamically the most appropriate communication mode between any peers according to schemes of computation and elements of context like topology. In the present study, note

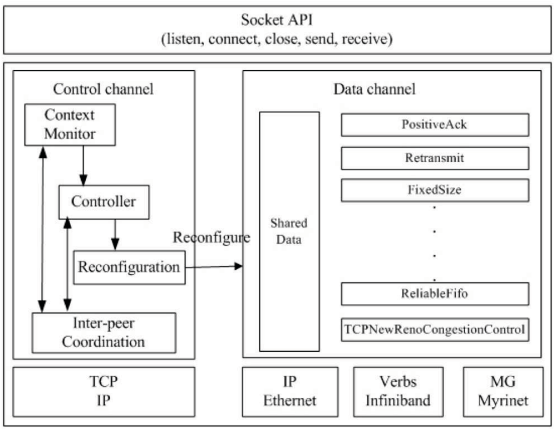


Figure 1. P2PSAP Protocol architecture

that the self adaptive capability is not used. In the following, we shall briefly present the P2PSAP protocol and the P2PDC environment.

A. Protocol P2PSAP

The P2PSAP protocol is based on the Cactus framework (see [14] and [15]) and makes use of micro-protocols. Figure 1 shows the architecture of P2PSAP; the protocol consists of a Socket interface and two channels: a control channel and a data channel.

A **Socket API** is placed on the top of the protocol in order to facilitate programming. Session management commands like listen, open, close, setsockoption and getsockoption are directed to Control channel; while data exchange commands, i.e. send and receive commands are directed to Data channel.

The **Data channel** based on Cactus framework, transfers data packets between peers. It has two levels: the physical layer and the transport layer. The physical layer is

encompassed to support communications on different networks, i.e. Ethernet, InfiniBand and Myrinet; the data channel can be triggered between the different types of networks. The transport layer is made of several micro-protocols; it can be reconfigured by substituting or removing and adding micro-protocols. The behavior of the data channel is triggered by the control channel.

The **Control channel** manages session opening, closure and data channel configuration. The TCP/IP protocol is used to exchange control messages since those messages must not be lost. The control channel has four main components. The *Context monitor* collects context data like schemes of computation, peers location or machine loads. The *Controller* manages session opening and end through TCP connection opening and closure; it also combines and analyzes context information so as to choose the configuration (at session opening) or to take reconfiguration decision (during session operation) for **Data channel**.

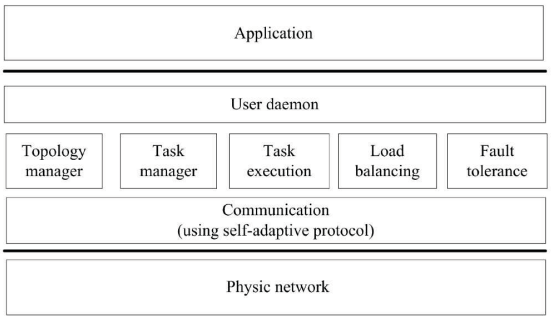


Figure 2. Environment architecture

Reconfiguration of Data channel is carried out via the dedicated Cactus functions by substituting or removing and adding micro-protocols.

B. Environment P2PDC

Figure 2 illustrates the global architecture of the environment P2PDC. We describe now its main components
- *User daemon* is the interaction interface between the application and the environment; it allows users to submit their tasks and retrieve final results.
- *Topology manager* organizes connected peers into clusters and maintains links between clusters and peers.
- *Task manager* is responsible for subtasks distribution and results collection.
- *Task execution* executes subtasks and exchanges intermediate results.
- *Load balancing* estimates peer workload and migrates a part of work from overloaded peer to non-loaded peer.
- *Fault tolerance* ensures the integrity of the calculation in case of peer or link failure.
- *Communication* provides support for directed data exchange between peers using protocol P2PSAP.

In order to allow programmers to develop their own application easily, P2PDC proposes a programming model with a reduced set of communication operations and some deployment activities carried out automatically by the environment.

Communication operations: There are only three classical operations: *P2P_Send*, *P2P_Receive* and *P2P_Wait*. The idea is to facilitate programming of large-scale P2P applications and to hide the complexity of communication management as much as possible. The operation *P2P_Wait* is particular and used in order to wait for the arrival of a message from another peer.

Messages exchanged between peers can be decomposed into two classes: data messages and control messages. Data messages are used to exchange boundary values of block components at interfaces between peers; while control messages are used to exchange computational state like local termination criteria, termination command, etc. A *flags* parameter is added to communication operations in order to distinguish two classes of messages: the *CTRL_FLAG* indicates control messages and the *DATA_FLAG* indicates data messages. The control channel of P2PSAP exchanges not only messages for protocol configuration but also messages from P2PDC application with the so-called *CTRL_FLAG* flag. Then messages from P2PDC application with *DATA_FLAG* will be handled by data channel and messages from P2PDC application with *CTRL_FLAG* will be handled by control channel. Thus, the communication mode for data exchange is chosen according to the context of the user choice; while note that the communication mode for exchange of control messages is always asynchronous and reliable.

Application programming model: In order to develop an application, programmers have to write code for only three functions: *Problem_Definition()*, *Calculate()* and *Results_Aggregation()*. Others deployment activities like subtasks distribution or results collection are carried out automatically by the environment. In the *Problem_Definition()* function, programmers define the problem in indicating the number of subtasks and subtask data, computational scheme and number of peers necessary. In the *Calculate()* function, programmers write subtasks code; they can use *P2P_Send()* and *P2P_Receive()* to send or receive updates. Finally, programmers define how subtasks results are aggregated and the type of output, i.e. a console or a file, in the *Results_Aggregation()* function.

For further details about P2PSAP protocol and P2PDC environment, reference is made to [9] and [5] respectively.

V. IMPLEMENTATION

The implementation of the considered financial application is carried out with the P2PDC environment taking into account the computational scheme on one hand and the stopping criterion of the iterative algorithm, on the other hand. The computation scheme (synchronous, asynchronous) is chosen at the beginning of the solution. Each node updates its assigned sub-blocks of components of the iterate vector in the lexicographical order. Then, the transmission of the boundary blocks assigned to each computational node to the contiguous blocks is delayed so as to reduce the waiting time in the case of the use of synchronous method. The parallel algorithm is based on the SPMD paradigm, which is commonly used for parallel and distributed application. In our case, each process initializes its own data set. The principle of the implementation of the parallel algorithm is summarized below; in this flow chart, r is the time step index, V_0 is the initial condition of the time dependant problem and F_{r+1} is the right hand side computed at time r+1.

In the *Calculate()* function of P2PDC, *P2P_Send()* and *P2P_Receive()* operations will not have the same property according to the computation scheme used. The operation *P2P_Wait()* is used in order to synchronize the peers at each time step before the computation of the right-hand side. In the synchronous (resp. asynchronous) scheme, the *P2P_Send()* and *P2P_Receive()* operations are blocking (resp. not blocking). Note that this implementation of the parallel asynchronous method simplifies the exchange of boundary blocks in the block relaxation method but it is not adapted to the synchronized part at each time step. The P2PDC communication API (Application Programming Interface) does not offer a real synchronization barrier but only a function, which wait for the arrival of a message from another peer.

We specify that in the implementation of the application concerning the computation of European and American options, we use the classical block-relaxation method in which the blocks are numbered using the lexicographical ordering. More precisely, in the considered implementation several adjacent tridiagonal blocks are gathered; so, this kind of method can be viewed as a subdomain method without overlapping between the subdomains. Note also that each tridiagonal algebraic subsystem is solved using the TDMA method well adapted to the solution of tridiagonal subsystems. Moreover, the considered implementation allows having a multiplicative subdomain method. Finally, in the considered implementation of the subdomain relaxation method without overlapping the data are split into regular polygons.

Algorithm: on computing peers

```
// Initial condition V_0
Input V_0
For each time step r do
  // synchronized part
  Compute δt*F_{r+1} + V_0
  While no convergence do
    // by synchronous or asynchronous algorithm
    Solve by block relaxation method A.V=δt*F_{r+1} + V_0
    // Project in the case of American options
    V_0 ← V
    Send local convergence to convergence manager peer
    Receive global convergence from convergence manager peer
  End while
  If on barrier manager peer Then
    Wait all peers
    Send notification restart to computing peer
  Else
    Send notification message to barrier manager peer
    Wait notification restart from barrier manager peer
  End If
End for
Send final notification message to convergence manager peer.
```

For the implementation of the time marching scheme, the parallel synchronous algorithm is easy to implement, since the *P2P_Send* and the *P2P_Receive* are blocking operations. In the case of the asynchronous scheme, the implementation is more difficult. Due to the specificity of the implicit time marching scheme and to P2PDC conception, it is necessary to implement an additional synchronisation barrier when a new time step is considered. At the end of a time step, all the peers are synchronized thanks to the *P2P_Wait* operation and send a message to a barrier manager peer dedicated among all the computing peers. Then this manager barrier peer restarts each computing peer. An additional level of synchronization is performed only when the first relaxation is started.

With respect to [5], the Richardson solver is replaced by the subdomain without overlapping solver and a new efficient stopping criterion is implemented using a supplementary convergence manager peer only devoted to this task. Indeed, we have compared the efficiency of the stopping criterion proposed by [16] based on the use of a decentralized method and the centralized one presented in [17]. The comparison of the two stopping criteria has shown that the number of relaxations are not the same and we have finally chosen the stopping criterion proposed by [17] since more relaxations are performed and better numerical quality of the computed

solution is obtained. For a current time step, when a local convergence is reached by a computing peer, a message is sent to the convergence manager peer; a local convergence corresponds to the fact that the uniform norm of the difference between two successive updates is less than a given threshold, fixed here to 10^{-6}. This last manager peer stops the iterative process when all the computing peers have reached convergence then the next time step is performed.

When all time steps are performed, the computation is ended; in this case, a final notification message is sent by the computing peers to the convergence manager peer. Note that the principle of the centralized stopping criterion is the same as well as for the synchronous algorithm than the asynchronous one; nevertheless, the only difference between synchronous and asynchronous convergence detection lies in the way that the convergence manager answers to computing peers. In the synchronous case, the convergence manager waits for the convergence message of all computing peers at each relaxation; while in the asynchronous case, the answer is sent dynamically after receipt of any convergence message.

VI. PEER-TO-PEER EXPERIMENTS

Computational schemes have been implemented in C language as a parallel algorithm using the P2PDC environment. Computational experiments have been carried out on the Grid5000 platform. This French grid platform is composed of 2970 processors with a total of 6906 cores distributed over 9 sites in France. Most of them have at least a Gigabit Ethernet network for local machines. Nodes between the different sites range from 2.5 Gflops up to 10 Gflops. Several site of Grid5000 have several clusters with different performances.

Table I displays the characteristics of the machine used in the computational experiments.

Site	Cluster	Processors Type	Speed GHz	CPU	Core	RAM
Lille	ChinqChint	Intel Xeon E5440 QC	2,83	2	8	8
Orsay	Gdx	AMD Opteron250	2,4	2	2	2

TABLE I. CHARACTERISTICS OF MACHINES ON EACH SITE.

We have considered a cubic domain Ω contained in the 3D space, Ω being discretized with $S = s^3$ points where s=256 denotes the number of points considered on each edge of the cube. The iterate vector is decomposed into $m = s^2$ sub-blocks with s components. A set of sub-blocks is assigned to each node and is updated using a sequential Gauss-Seidel block relaxation like algorithm performed in the lexicographical order. Concerning the time marching scheme, only three time steps are considered.

The results of the sequential computational experiments are summarized in Table II.

European Options				American Options			
Lille		Orsay		Lille		Orsay	
Time	Relax	Time	Relax	Time	Relax	Time	Relax
7669	1004	12668	1004	8534	1108	14362	1108

TABLE II. ELAPSED TIME AND RELAXATIONS WITH SEQUENTIAL ALGORITHM ON EACH SITE.

The parallel computational experiments are summarized in Table III, IV, V and VI in which 2, 4, 8, 16, 32, 64 and 128 computing machines are used (not including the convergence manager peer). In Table III and in Table V the number of relaxations is mentioned; note that for only the asynchronous experiments, Min, Max and Average number of relaxations are taken over the processors.

	Asynchronous				Synchronous	
Peers	Time/s	Relaxations			Time/s	Relaxations
		Min	Max	Average		
2	5492	996	1439	1217	4904	1004
4	3158	1019	1695	1353	3158	1015
8	1289	995	1572	1309	1370	1015
16	515	1010	2363	1502	589	1030
32	195	822	1866	1486	297	1035
64	90	1074	1823	1478	292	1053
128	65	949	2316	1380	303	1091

TABLE III. ELAPSED TIME AND AVERAGE RELAXATIONS FOR EUROPEAN OPTIONS.

	Asynchronous		Synchronous	
Peers	Speedup	Efficiency	Speedup	Efficiency
2	1.40	0.70	1.56	0.78
4	2.43	0.61	2.43	0.61
8	5.95	0.74	5.60	0.70
16	14.89	0.93	13.02	0.81
32	39.39	1.23	25.82	0.81
64	85.21	1.33	26.26	0.41
128	117.90	0.92	25.31	0.20

TABLE IV. SPEEDUP AND EFFICIENCY FOR EUROPEAN OPTIONS.

For the considered application and architecture used, when the number of machines is greater than 8 for European options and 4 for American options, the asynchronous scheme of computation, scales better than the synchronous one. For example, with 128 peers, the asynchronous computation scheme clearly performs about five times (European options) and about three times (American options) better than the synchronous one. This feature shows that asynchronous algorithms are less sensitive to granularity network latency. Moreover the asynchronous

algorithms do not generate idle time. Therefore load balancing is not necessary for decreasing the elapsed time.

	Asynchronous				Synchronous	
Peers	Time/s	Relaxations			Time/s	Relaxations
		Min	Max	Average		
2	7169	1106	1862	1484	6526	1123
4	3209	1161	1709	1431	3681	1138
8	1464	1096	1720	1400	1659	1138
16	581	1153	2574	1748	626	1138
32	285	1069	2157	1712	361	1145
64	102	495	1469	1049	318	1165
128	109	1284	2685	1704	337	1208

TABLE V. ELAPSED TIME AND AVERAGE RELAXATIONS FOR AMERICAN OPTIONS.

	Asynchronous		Synchronous	
Peers	Speedup	Efficiency	Speedup	Efficiency
2	1.19	0.59	1.31	0.65
4	2.66	0.66	2.32	0.58
8	5.83	0.73	5.14	0.64
16	14.67	0.92	13.63	0.85
32	29.94	0.94	23.64	0.74
64	83.72	1.31	26.84	0.42
128	78.29	0.61	25.32	0.20

TABLE VI. SPEEDUP AND EFFICIENCY FOR AMERICAN OPTIONS.

In both cases of synchronous or asynchronous scheme, the speedup of computation is calculated using the shortest sequential elapsed time (see Table II); note that on a heterogeneous architecture the notion of speedup and efficiency is inappropriate, but this is a good indicator of performance. Nevertheless, note that the obtained values of speedup are high, particularly for the asynchronous scheme with large number of machine. Note also that super-linear acceleration has been obtained for asynchronous schemes of computation but not in the synchronous case. This feature shows that the communication overhead induced by the parallelisation is small enough, so that positive effects of parallelism on memory management can be seen. Indeed, the more peers are used, the less data is involved in the computation on each single peer. Therefore, the proportion of data that fits in cache memory is higher when granularity is fine.

The efficiency of asynchronous scheme of computation is better compared to the efficiency of synchronous scheme of computation. Note that for the considered size of the algebraic system to solve, 128 peers are sufficient.

VII. CONCLUSION

In the presented study, for the solution of the evolution European and American options derivatives, we have studied the use and the implementation of parallel synchronous and asynchronous iterative algorithm [18] on an environment of peer-to-peer architecture. It follows from the computational experiments that the choice of communication mode (synchronous or asynchronous) has important impact on the efficiency of the distributed methods. The computational results show that the use of P2PDC carried out on the Grid5000 platform permits one to obtain good efficiency, particularly when the number of processors is large except for the synchronous method when the number of processors is greater or equal to 64. Moreover, using parallel iterative asynchronous methods, compared to the synchronous ones, is well adapted to the use of heterogeneous and distant distributed large number of machines. Finally, the difference of performance is mainly due to the weight of synchronisations between the processors.

VIII. ACKNOWLEDGMENT

Part of this study has been made possible by ANR grant: ANR-07-CIS7-011 and support of Grid5000.

REFERENCES

[1] Paul Wilmott, Jeff Dewyne, Sam Howison, Option pricing - mathematical models and computation - Oxford financial press – 1993.

[2] P. Jaillet, D. Lamberton, B. Lapeyre, “Variational inequalities and the pricing of american options”, Acta Applicandae Mathematicae, vol. 21, pp. 263 – 289, 1990.

[3] D. El Baz, G. Jourjon, “Some solutions for Peer to Peer Global Computing”, in *Proceedings of 13th Euromicro conference on Parallel, Distributed and Network-Base Processing*, 2005, pp. 49-58.

[4] P. Spitéri, M. Chau, “Parallel asynchronous Richardson method for the solution of obstacle problem” in *Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications*, pp. 133-138, 2002.

[5] T.T. Nguyen, D. El Baz, P. Spiteri, G. Jourjon, M. Chau, “High Performance Peer-to-Peer Distributed Computing with Application to Obstacle Problem”, in *Proceedings of IEEE IPDPS 2010 Conference,* Atlanta, 2010.

[6] L. Giraud, P. Spiteri, “Parallel resolution of non-linear boundary value problems”, Mathematical Modeling and Numerical Analysis, vol. 25, n° 5, pp. 579-606, 1991.

[7] J.C. Miellou. P. Spiteri, “A criterion of convergence for general fixed point methods”, Mathematical Modeling and Numerical Analysis, vol. 19, pp. 645-669, 1985.

[8] J.C. Miellou, D. El Baz, P. Spiteri, “A new class of asynchronous iterative algorithms with order interval”, Mathematics of Computation, vol. 67, n° 221, pp. 237-255, 1998.

[9] D. El Baz, T.T. Nguyen, “A self-adaptive communication protocol with application to high performance peer to peer distributed computing”, in *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, Pisa, pp. 327-333, 2010.

[10] Aumage, G. Mercier, "MPICH/Madeleine: a True Multi-Protocol MPI for High Performance Networks", *15th International Parallel and Distributed Processing Symposium* (IPDPS'01), 2001.

[11] David P. Anderson, “BOINC: A System for Public-Resource Computing and Storage”, *5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, USA, 2004.

[12] N. Andrade, W. Cirne, F. Brasileiro, P. Roisenberg, “OurGrid: An approach to easily assemble grids with equitable resource sharing”, in

Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing, pp. 61-86, 2003.

[13] J.M. Ortega, W. Rheinboldt, Iterative solution of nonlinear equations in several variables, Academic press, 1970.

[14] Matti A. Hiltunen, "The Cactus Approach to Building Configurable Middleware Services", in *DSMGC2000*, Nuremberg, Germany, 2000.

[15] G.T Wong, M.A Hiltunen, R.D Schlichting, "A configurable and extensible transport protocol" in *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, pp. 319–328, 2001.

[16] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and distributed computation: numerical methods, in Prentice Hall, Englewood Cliffs, N.J., 1987.

[17] J.M. Bahi, S. Contassot-Vivier, R. Couturier, Parallel iterative algorithms: from sequential to grid computing, in Chapman & Hall/CRC, 2007.

[18] Pierre Spiteri, "Parallel optimization and financial mathematics", in *Proceedings of COSI'09 and Microsoft summer school*, Annaba, pp. 257 - 270, 2009.

5.11 Proceeding 2

Synchronous and Asynchronous distributed computing for financial option pricing

Thierry GARCIA[1], Ming CHAU[2] and Pierre SPITERI[1]

[1] INP – ENSEEIHT – IRIT BP 7122, 2 Rue Camichel F-31071 Toulouse Cedex, France
[2] Advanced Solutions Accelerator 199 Rue de l'Oppidum F-34170 Castelnau le Lez, France
{thierry.garcia, pierre.spiteri}@enseeiht.fr
mchau@advancedsolutionsaccelerator.com

Abstract. This paper deals with the numerical solution of financial applications, more specifically the computation of American and European options derivatives modeled by boundary value problems. In such applications we have to solve large-scale algebraic linear systems. We concentrate on synchronous and asynchronous parallel iterative algorithms carried out on Grid'5000, by using an experimental peer-to-peer platform. The properties of the operators arising in the discretized problem ensure the convergence of the parallel iterative synchronous and asynchronous algorithms. Different detection convergence algorithms are tested for asynchronous iterations. Computational experiments performed on distributed architectures are presented and analyzed.

Keywords: Parallel asynchronous algorithms; distributed computing; iterative numerical methods; European options derivatives; American options derivatives.

I. INTRODUCTION

Financial applications have known great developments these last years. In particular, European and American options derivatives modeled by the classical Black – Scholes equation have known great interest [1]. The problem consists of solving a time dependant boundary value problem defined on an unbounded domain generally included in the three dimensional space. A classical artifice consists of solving the Black – Scholes equation on a bounded domain and to increase the size of the domain for ensuring convergence to the exact solution [2]. So, we have to solve boundary value problems by a numerical way. Taking into account the size of the algebraic systems derived from the discretization process of the Black – Scholes equation, parallel iterative algorithms are preferred. In this paper we concentrate on comparison between parallel asynchronous and synchronous iterative methods implemented on distributed architectures and also on comparison of different methods for asynchronous global convergence detection. Recent advances in microprocessors architecture and networks permit one to consider new applications like High

Performance Computing (HPC). Therefore, we can identify a real stake at developing new protocols and environments for HPC since this can lead to economic and attractive solutions. Task parallel models and distributed iterative methods for large scale parallel numerical simulation on internet network gives rise to numerous challenges like communication management, scalability, heterogeneity and peer volatility on the network (see [3]). Some performance issues can be addressed via distributed asynchronous iterative algorithms (see [4] to [8]). In the context of an ANR grant, reference is made to [5] for the solution of the stationary obstacle problem on the so-called P2PDC peer-to-peer platform and to [9] for contribution to the development of a new protocol and an environment for distributed high performance computing. The approach presented in [9] is different from MPICH Madeleine [10], middleware like BOINC [11] or OurGrid [12].

Then, the goal of the present study is to consider the solution of financial option pricing derivative performed on the considered peer-to-peer platform.

In [5], we have considered a stationary problem solved by a Richardson like method. In the present study we consider a time dependant problem closer and related to the studied financial application, more precisely European and American options derivatives in which each stationary problem derived from the numerical time marching scheme is solved by a parallel block relaxation algorithm, which is more efficient than the Richardson's one used in [5]. This kind of algorithm corresponds in fact to a subdomain method without overlapping. In addition, due to the overhead of synchronizations the use of asynchronous iterative algorithms [6], [7] and [8] seems well suited to distributed computing.

The present paper is structured as follows: section II deals with the presentation of the financial application (European and American options derivatives). In section III, we present parallel synchronous, and more generally parallel asynchronous iterative schemes. In section IV, implementation of parallel iterative algorithms on an environment for high performance parallel computing is detailed and moreover, different global convergence detection methods are presented and tested. Computational experiments are displayed and analyzed in section V, in particular the pertinence to apply such parallel asynchronous iterative methods is presented for the solution of the model problems. Finally concluding remarks are presented in section VI.

II. THE PROBLEM OF OPTIONS DERIVATIVES

The classical Black – Scholes equation is a boundary value problem describing the evolution of call or put options in the field of mathematics of financial derivatives [1]. Among the many descriptions of financial option contracts, we can consider:
- the European option, which may only be exercised at expiry, i.e. when the time τ takes the value T of the final expiry date,
- the American option, which may be exercised at any time prior to expiry.

These two kinds of options are modeled by retrograde time dependent convection – diffusion equations. From a mathematical point of view, the main difference between them consists in the fact that European option is defined on a normed vectorial space while the American option is defined on a closed convex set included in a normed vectorial space; consequently, the determination of American option needs the projection of the computed values on the convex set.

Classically, an European option is modeled by the following time dependent linear equation:

$$\begin{cases} \dfrac{\partial v(\tau,x)}{\partial \tau} + (r - \dfrac{\sigma^2}{2}).\nabla v + \dfrac{\sigma^2}{2}\Delta v - rv = 0, \text{ everywhere in } [0,T]xR^n \\ v(T,x) = \psi(x) \end{cases}$$

where ψ(x)= Max(x-K,0) (in the case of call option) or ψ(x) = Max(K-x,0) (in the case of put option); in the above equation, K denotes the exercise price – called classically strike, v denotes the value of the considered option, i.e. a "call" or a "put" option; v = v(τ,x) is a function of the current value of the underlying asset x and of the time τ. Note also that the considered option also depends on the following parameters:
- r the interest rate,
- σ the volatility of the underlying asset, σ being in fact the instantaneous standard deviation of the price with respect to K fixed beforehand; in fact σ characterizes the uncertainly of the behavior of the option.

Using the same previous notations, the American option is modeled by the following nonlinear equation:

$$\begin{cases} \dfrac{\partial v(\tau,x)}{\partial \tau} + (r - \dfrac{\sigma^2}{2}).\nabla v + \dfrac{\sigma^2}{2}\Delta v - rv \geq 0, \ v(\tau,x) \geq \psi(x), \text{ everywhere in } [0,T]xR^n \\ (\dfrac{\partial v(\tau,x)}{\partial \tau} + (r - \dfrac{\sigma^2}{2}).\nabla v + \dfrac{\sigma^2}{2}\Delta v - rv)(v(\tau,x) - \psi(x)) = 0, \text{ everywhere in } [0,T]xR^n \\ v(T,x) = \psi(x) \end{cases}$$

One of the main difficulty lies on the fact that the Black – Scholes equation is not defined on a bounded domain, but is defined on the unbounded domain R^n, $n \geq 1$. This difficulty is solved by considering the problem defined on a bounded domain $\Omega \subseteq R^n$ with appropriate boundary conditions. Then it can be proved that the solution of the equation defined on the bounded domain Ω converges to the solution of the model problem when the measure of Ω tends to infinity [2].

Another particularity of the problem to solve, is that the value of the option is not known at the initial time τ = 0; only the final value v(T,x) is known. So the problem consists in computing v(0,x).

These previous two particularities can be treated, first by considering problems defined on a bounded large domain Ω and secondly by a change of variable, which consists in replacing the variable τ by a variable t = T - τ. Thus, the model problem of the European option is replaced by the following linear problem:

$$\begin{cases} \dfrac{\partial v(t,x)}{\partial t} - (r - \dfrac{\sigma^2}{2}).\nabla v - \dfrac{\sigma^2}{2}.\Delta v + rv = 0, \text{ everywhere in } [0,T]x\ \Omega \\ v(0,x) = \psi(x) \\ B.C. \text{ of } v(t,x) \text{ defined on } \partial\Omega \end{cases}$$

where B.C. describes the boundary conditions on the boundary $\partial\Omega$ of the domain Ω. Practically, the Dirichlet condition where v is fixed on $\partial\Omega$ or the Neumann condition where the normal derivative of v is fixed on $\partial\Omega$, is classically considered.

Concerning the American option, by the same way, we have to solve the following nonlinear problem:

$$\begin{cases} \dfrac{\partial v(t,x)}{\partial t} - (r - \dfrac{\sigma^2}{2}).\nabla v - \dfrac{\sigma^2}{2}\Delta v + rv \geq 0,\ v(t,x) \geq \psi(x), \text{ everywhere in } [0,T]x\Omega \\ (\dfrac{\partial v(t,x)}{\partial t} - (r - \dfrac{\sigma^2}{2}).\nabla v - \dfrac{\sigma^2}{2}\Delta v + rv)(v(t,x) - \psi(x)) = 0, \text{ everywhere in } [0,T]x\Omega \\ v(0,x) = \psi(x) \\ B.C \text{ of } v(t,x) \text{ defined on } \partial\Omega \end{cases}$$

III. NUMERICAL SOLUTION OF THE PROBLEMS OF OPTIONS DERIVATIVES

For the numerical solution of the financial problems, we consider that the temporal part is discretized by implicit or semi – implicit scheme. While, for the spatial part of the problem, the bounded domain $\Omega \subseteq R^3$ is discretized with an uniform mesh. Note also that the spatial differential operators are discretized by appropriate schemes as follows:

- the Laplacian is discretized by using the classical seven points scheme,
- the first derivatives, corresponding to the convection phenomenon, are discretized by using decentered scheme; more precisely, if the coefficient of the considered first derivative is strictly positive, then the forward-difference formula is used, else backward–difference formula is considered.

Thus, if good accuracy is required, then the full discretization of the Black – Scholes problem leads to the solution of a very large linear algebraic system at each time step.

Let A denote the discretization matrix of the model problem. It follows from the considered appropriate discretization of the Black - Scholes equations, that the diagonal entries of the matrix A are strictly positive and that the off-diagonal entries of the matrix A are nonpositive. Moreover, the matrix A is strictly diagonally dominant. Thus, A is an M – matrix [13]. This very interesting property of the matrices to invert, at each time step, has a consequence regarding to the behaviour of the parallel iterative algorithms considered in the sequel; indeed the convergence of the iterative scheme is ensured (see [4] and [6] to [8]).

Note also, that due to the large size of the linear algebraic system to invert at each time step, it is necessary to use iterative methods. More precisely, we consider in the present study iterative parallel synchronous and asynchronous block relaxation

algorithms studied in [6], [7] and [8], implemented in the present study on distributed architecture. In the sequel, let us recall the formulation of parallel synchronous and more generally asynchronous block relaxation algorithms for the solution of a large linear algebraic system.

Let us consider the solution of the linear algebraic system associated, for example to the solution of the discretized European option

$$A.V = F$$

where V and F are respectively, a vector whose components approximate the values of the exact solution and the right hand side of the system respectively, at each point of the mesh. We consider now a block decomposition from the previous linear algebraic system and associate the following fixed-point mapping:

$$V_i = A_{i,i}^{-1}(F_i - \sum_{j \neq i} A_{i,j} V_j) = \Phi_i(V),\ i = 1, \cdots, m,$$

where m is an integer denoting the number of blocks. This kind of fixed point problem can be considered, at each time step, for the solution of the problem of European option.

For the solution of the discretized American option problem, we have to consider now the projection on a convex set (see [1]) as follows:

$$V_i = \text{Proj}(A_{i,i}^{-1}(F_i - \sum_{j \neq i} A_{i,j} V_j)) = \Phi_i(V),\ i = 1, \cdots, m.$$

Then, by considering the appropriate operator as well as, the European option than the American option, the problems consist of solving the following fixed point problem:

$$\begin{cases} \textit{Find } V^* \text{ such that} \\ V^* = \Phi(V^*) \end{cases} \tag{1}$$

where $V \rightarrow \Phi(V)$ is a fixed point mapping defined in a finite dimensional space. For all V, consider the following block-decomposition of the mapping Φ associated to the parallel distributed implementation:

$$\Phi(V) = (\Phi_1(V), \ldots, \Phi_m(V))$$

We consider the distributed solution of the fixed point problem (1) via a parallel asynchronous block relaxation method defined as follows (see [6] to [8]): let the initial guess $V^{(0)}$ be given and for every $p \in N$ assume that we can get $V^{(1)}, \ldots, V^{(p)}$; then $V^{(p+1)}$ is defined recursively by:

$$V_i^{(p+1)} = \begin{cases} V_i^{(p)} \text{ if } i \notin J(p) \\ \Phi_i(\ldots, V_j^{(s_j(p))}, \ldots) \text{ if } i \in J(p) \end{cases} \tag{2}$$

where $J = \{J(p)\}$, $p \in N$, is a sequence of nonempty sets such that:

$$\begin{cases} J(p) \subset \{1, \ldots, m\},\ J(p) \neq \emptyset,\ \forall p \in N, \\ \forall i \in \{1, \ldots, m\},\ \text{Card}(\{p \in N \mid i \in J(p)\}) = +\infty \end{cases} \tag{3}$$

and

$$\begin{cases} \forall p \in N,\ \forall j \in \{1,...,m\},\ s_j(p) \in N,\ 0 \le s_j(p) \le p, \\ \forall p \in N,\ s_i(p) = p \text{ if } i \in J(p), \\ \forall p \in N,\ \forall j \in \{1,...,m\},\ \lim_{p \to \infty} (s_j(p)) = +\infty. \end{cases} \quad (4)$$

The previous asynchronous iterative scheme models computations that are carried out in parallel without order nor synchronization and describes in fact a subdomain method without overlapping. Particularly, it permits one to consider distributed computations whereby processors go at their own pace according to their intrinsic characteristics and computational load. The parallelism between the processors is well described by J since J(p) contains the number of components relaxed by each processor on a parallel way while the use of delayed components in (2) permits one to model nondeterministic behavior and does not imply inefficiency of the considered distributed scheme of computation. Note that, theoretically, each component of the vector must be relaxed an infinite number of time. The choice of the relaxed components may be guided by any criterion, and, in particular, a natural criterion is to pick-up the most recently available values of the components computed by the processors.

Remark. The algorithm (2) – (4) describes a computational method where the communications between peers can be synchronous or asynchronous. Among them parallel synchronous methods, when $s(p) \equiv p, \forall\ p \in N$; moreover if $J(p) = \{1,\ldots, m\}$ and $s(p)=p, \forall\ p \in N$, then (2) – (4) describes the sequential block Jacobi method while if $J(p) = p.\mathrm{mod}(m) + 1$ and $s(p)=p, \forall\ p \in N$, then (2) – (4) models the sequential block Gauss – Seidel method. So, the previous model of parallel asynchronous algorithm appears like a general model.

For the solution of the evolution Black - Scholes equations, a numerical time marching scheme is implemented and, at each time step, we have to solve a large scale algebraic system by using either parallel synchronous or asynchronous algorithms.

Then, for the solution of stationary problems derived from European and American options derivatives, the convergence of synchronous or asynchronous relaxation algorithms has been established by various ways, using contraction techniques (see [6] and [7]) or partial ordering techniques (see [8]). To summarize, as previously said, since the discretization matrix is an M-matrix then thanks to various results established in [6], [7] and [8], the iterative process described by (2) – (4) converges to V^*, for every initial guess $V^{(0)}$. The reader is referred to these previous references for more details. Moreover assume that the algebraic system is splitted into q blocks, $q \le m$, corresponding to a coarser subdomain decomposition without overlapping; then using result of [7], it can be shown by using the same arguments, that the parallel asynchronous block relaxation methods converge for this coarser decomposition. Furthermore, if the subdomain decomposition associated with m blocks is a point decomposition, then parallel asynchronous block relaxation methods converge for every subdomain coarser decomposition and for every numbering (lexicographical or red-black) of the blocks.

IV. IMPLEMENTATION

ENVIRONMENT FOR DISTRIBUTED HIGH PERFORMANCE COMPUTING

P2PDC [5] is a peer-to-peer demonstrator for distributed high performance computing that was developed at LAAS (see [9]).

There are only three classical operations: P2P_Send, P2P_Receive and P2P_Wait. The operation P2P_Wait is particular and used in order to wait for the arrival of a message from another processor. For further details about P2PDC environment, reference is made to [5].

PARALLEL IMPLEMENTATION OF FINANCIAL APPLICATIONS

The implementation of the considered financial application is carried out with the P2PDC environment taking into account the computational scheme on one hand and the stopping criterion of the iterative algorithm on the other hand. The communication mode (synchronous, asynchronous) is chosen at the beginning of the solution. Each node updates its assigned sub-blocks of components of the iterate vector in the lexicographical order. Then, the transmission of the boundary blocks assigned to each computational node to the contiguous blocks is delayed so as to reduce the waiting time in the case of the use of synchronous method. The parallel algorithm is based on the SPMD paradigm, which is commonly used for parallel and distributed application. In our case, each process initializes its own data set. The principle of the implementation of the parallel algorithm is summarized below; in this flow chart, r is the time step index, V^0 is the initial condition of the time dependant problem and F^{r+1} is the right hand side computed at time r+1.

In P2PDC, P2P_Send() and P2P_Receive() operations will not have the same property according to the communication method used. This feature is specific to P2PDC. The operation P2P_Wait() is used in order to synchronize the processors at each time step before the computation of the right-hand side. In the synchronous (resp. asynchronous) scheme, the P2P_Send() and P2P_Receive() operations are blocking (resp. non blocking). Note that this implementation of the parallel asynchronous method simplifies the exchange of boundary blocks in the block relaxation method but it is not adapted to the synchronized part at each time step. The P2PDC communication API (Application Programming Interface) does not offer a real synchronization barrier but only a function, which wait for the arrival of a message from another processor.

We specify that in the implementation of the application concerning the computation of European and American options, we use the classical block-relaxation method in which the blocks are numbered using the lexicographical ordering. More precisely, in the considered implementation several adjacent tridiagonal blocks are gathered; so, this kind of method can be viewed as a subdomain method without overlapping between the subdomains. Note also that each tridiagonal algebraic subsystem is solved using the TDMA method well adapted to the solution of

tridiagonal subsystems. Moreover, the considered implementation allows having a multiplicative subdomain method. Finally, in the considered implementation of the subdomain relaxation method without overlapping the data are split into regular polygons.

Algorithm: on computing processors

```
// Initial condition V^0
input V^0
for each time step r do
  // synchronized part
   compute δt*F^(r+1) + V^0
   while no convergence do
       // by synchronous or asynchronous algorithm
       solve by block relaxation method A.V=δt*F^(r+1)+V^0
       // Project in the case of American options
       V^0 ← V
       send local convergence to convergence manager
       processor
       receive global convergence from convergence manager
       processor
   end while
   if on barrier manager processor then
       wait all processors
       send notification restart to computing processors
   else
       send notification message to barrier manager
       processor
       wait notification restart from barrier manager
       processor
   end if
end for
send final notification message to convergence manager
processor.
```

For the implementation of the time marching scheme, the parallel synchronous algorithm is easy to implement, since the P2P_Send and the P2P_Receive functions are blocking operations. In the case of the asynchronous scheme, the implementation is more difficult since the P2P_Send and P2P_Receive functions take into account the asynchronous communication mode. Due to the specificity of the implicit time marching scheme and to P2PDC conception, it is necessary to implement an additional synchronization barrier when a new time step is considered. At the end of a time step, all the processors are synchronized thanks to the P2P_Wait operation and send a message to a barrier manager processor dedicated among all the computing processors. Then this manager barrier processor restarts each computing processor. An additional level of synchronization is performed only when the first relaxation is started.

With respect to [5], a time dependant problem is here considered, the Richardson solver is replaced by the non-overlapping domain decomposition solver and a new

efficient stopping criterion is implemented using a supplementary convergence manager processor only devoted to this task. Indeed, we have compared the efficiency of the stopping criterion proposed by [16] based on the use of a decentralized method and the centralized one presented in [17]. The comparison of the two stopping criteria has shown that the number of relaxations is not the same and we have finally chosen the stopping criterion proposed by [17] since more relaxations are performed and consequently better numerical quality of the computed solution is obtained. Table 1 displays an example of comparison of different stopping detection for the American options: a centralized method (denoted C), a centralized method with a predefined number of successive local convergence detection before deciding termination in order to reduce the probability of a false global convergence detection (denoted C(N), where N is the predefined number) and a decentralized method (denoted D).

	Asynchronous					Synchronous	
Nb. Procs.	Convergence detection	Time/sec	Relaxations			Time/sec	Relaxations
			Min	Max	Average		
2	C	5144	1100	1242	1171	6526	1123
2	C(10)	7353	1133	1909	1521		
2	D	7169	1106	1862	1484		
4	C	2536	1116	1252	1191	3681	1138
4	C(10)	3558	1134	1862	1499		
4	D	3209	1161	1709	1431		
8	C	1200	1103	1253	1151	1659	1138
8	C(10)	1553	1140	1737	1461		
8	D	1464	1096	1720	1400		
16	C	597	1033	1291	1139	626	1138
16	C(10)	587	1124	2609	1594		
16	D	581	1153	2574	1748		
32	C	238	556	1222	1064	361	1145
32	C(10)	313	1132	2517	1999		
32	D	285	1069	2157	1712		
64	C	103	492	1468	1049	318	1165
64	C(10)	108	1291	2077	1700		
64	D	102	495	1469	1049		
128	C	55	357	1408	1020	337	1208
128	C(10)	130	1388	2905	1855		
128	D	109	1284	2685	1704		

Table 1. Comparison of elapsed time and average relaxations for American options

For a current time step, when a local convergence is reached by a computing processor, a message is sent to the convergence manager processor; a local convergence corresponds to the fact that the uniform norm of the difference between two successive updates is less than a given threshold fixed here to 10^{-6}. This last manager processor stops the iterative process when all the computing processors have

reached convergence then the next time step is performed. When all time steps are performed, the computation is ended; in this case, a final notification message is sent by the computing processors to the convergence manager processor. Note that the principle of the centralized stopping criterion is the same as well as for the synchronous algorithm than the asynchronous one; nevertheless, the only difference between synchronous and asynchronous convergence detection lies in the way that the convergence manager answers to computing processors. In the synchronous case, the convergence manager waits for the convergence message of all computing processors at each relaxation; while in the asynchronous case, the answer is sent dynamically after receipt of any convergence message.

V. NUMERICAL EXPERIMENTS

Computational schemes have been implemented in C language as a parallel algorithm using the P2PDC environment. Computational experiments have been carried out on the Grid'5000 platform. The French grid platform is composed of 2970 processors with a total of 6906 cores distributed over 9 sites in France. Most of them have at least a Gigabit Ethernet network for local machines. Nodes between the different sites range from 2.5 Gflops up to 10 Gflops. Several site of Grid'5000 have several clusters with different performances. Table 2 displays the characteristics of the machine used in the computational experiments.

Site	Cluster	Processors Type	Speed GHz	CPU	Core	RAM
Lille	ChinqChint	Intel Xeon E5440 QC	2,83	2	8	8
Orsay	Gdx	AMD Opteron250	2,4	2	2	2

Table 2. Characteristics of machines on each site

We have considered a cubic domain Ω contained in the 3D space, Ω being discretized with $S = s^3$ points where s=256 denotes the number of points considered on each edge of the cube. The iterate vector is decomposed into $m = s^2$ sub-blocks with s components. A set of sub-blocks is assigned to each node and is updated using a sequential Gauss-Seidel block relaxation like algorithm performed using a lexicographical order. Concerning the time marching scheme only three time steps have been considered. The results of the sequential computational experiments are summarized in Table 3. In Table 4 and in Table 6 the number of relaxations is mentioned; note that for only the asynchronous experiments, Min, Max and Average number of relaxations are taken over the processors. The parallel computational experiments are summarized in Tables 4 to 7 in which 2, 4, 8, 16, 32, 64 and 128 computing machines are used (not including the convergence manager processor).

European Options				American Options			
Lille		Orsay		Lille		Orsay	
Time	Relax	Time	Relax	Time	Relax	Time	Relax
7669	1004	12668	1004	8534	1108	14362	1108

Table 3. Elapsed time and relaxations with sequential algorithm on each site

Synchronous and Asynchronous distributed computing for financial option pricing

	Asynchronous				Synchronous	
Peers	Time/s	Relaxations			Time/s	Relaxations
		Min	Max	Average		
2	5492	996	1439	1217	4904	1004
4	3158	1019	1695	1353	3158	1015
8	1289	995	1572	1309	1370	1015
16	515	1010	2363	1502	589	1030
32	195	822	1866	1486	297	1035
64	90	1074	1823	1478	292	1053
128	65	949	2316	1380	303	1091

Table 4. Elapsed time and average number of relaxations for European options

	Asynchronous		Synchronous	
Peers	Speedup	Efficiency	Speedup	Efficiency
2	1.40	0.70	1.56	0.78
4	2.43	0.61	2.43	0.61
8	5.95	0.74	5.60	0.70
16	14.89	0.93	13.02	0.81
32	39.39	1.23	25.82	0.81
64	85.21	1.33	26.26	0.41
128	117.90	0.92	25.31	0.20

Table 5. Speedup and efficiency for European options

	Asynchronous				Synchronous	
Peers	Time/s	Relaxations			Time/s	Relaxations
		Min	Max	Average		
2	7169	1106	1862	1484	6526	1123
4	3209	1161	1709	1431	3681	1138
8	1464	1096	1720	1400	1659	1138
16	581	1153	2574	1748	626	1138
32	285	1069	2157	1712	361	1145
64	102	495	1469	1049	318	1165
128	109	1284	2685	1704	337	1208

Table 6. Elapsed time and average number of relaxations for American options

	Asynchronous		Synchronous	
Peers	Speedup	Efficiency	Speedup	Efficiency
2	1.19	0.59	1.31	0.65
4	2.66	0.66	2.32	0.58
8	5.83	0.73	5.14	0.64
16	14.67	0.92	13.63	0.85
32	29.94	0.94	23.64	0.74
64	83.72	1.31	26.84	0.42
128	78.29	0.61	25.32	0.20

Table 7. Speedup and efficiency for American options

For the two European and American options respectively, Figure 1 and Figure 2 show the elapsed time, Figure 3 and Figure 4 the speedup, Figure 5 and Figure 6 the

efficiency for the parallel schemes of computation, in the case of using one or two distant clusters. Nevertheless, in this case the speedup and the efficiency are given only for giving an evaluation of the performance of the parallel algorithms, since in the case of using an heterogeneous architecture, the notions of speedup and efficiency have no sense. In both cases of synchronous or asynchronous scheme, the speedup is computed using the shortest sequential elapsed time (see Table 3). Moreover, note that the obtained values of speedup are high, particularly for the asynchronous scheme with large number of machines. Note also that super-linear acceleration has been obtained for asynchronous schemes of computation but not in the synchronous case. This feature shows that the communication overhead induced by the parallelization is small enough, so that positive effects of parallelism on memory management can be seen. Indeed, the more processors are used, the less data is involved in the computation on each single machine. Therefore, the proportion of data that fits in cache memory is higher when granularity is fine.

The efficiency of asynchronous scheme of computation is better compared to the efficiency of synchronous scheme of computation. Note that for the considered size of the algebraic system to solve, 128 processors are sufficient.

For the considered application and architecture used, when the number of machines is greater than 8 for the European options and 4 for the American options respectively, the asynchronous scheme of computation scales better than the synchronous one. For example, with 128 processors, the asynchronous computation scheme clearly performs about five times (for European options) and about three times (for American options) better than the synchronous one. This feature shows that asynchronous algorithms are less sensitive to granularity network latency and load balance issues when the number of processors is large.

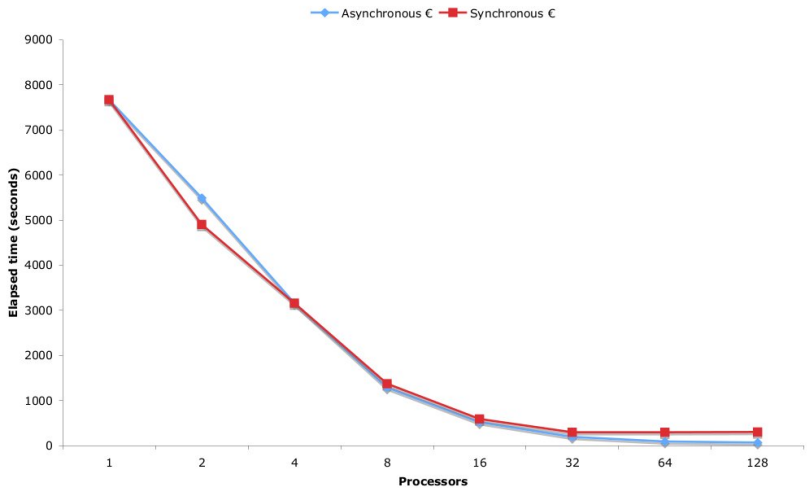


Fig. 1. Elapsed time for synchronous and asynchronous scheme for European options

Synchronous and Asynchronous distributed computing for financial option pricing

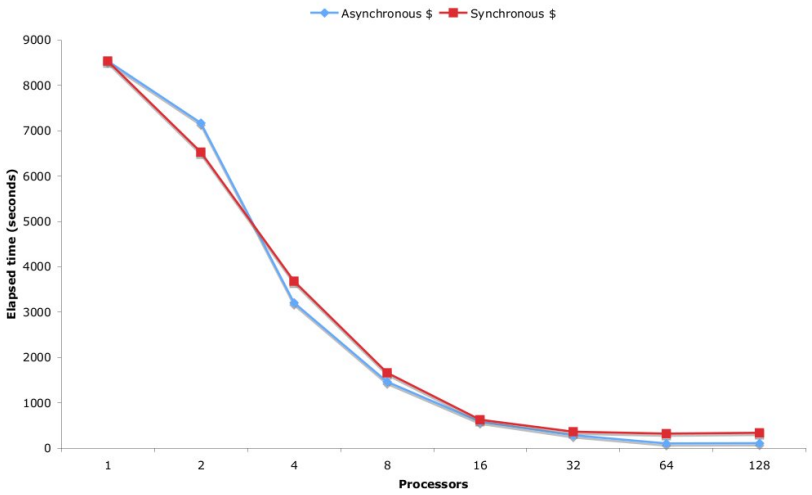


Fig. 2. Elapsed time for synchronous and asynchronous scheme for American options

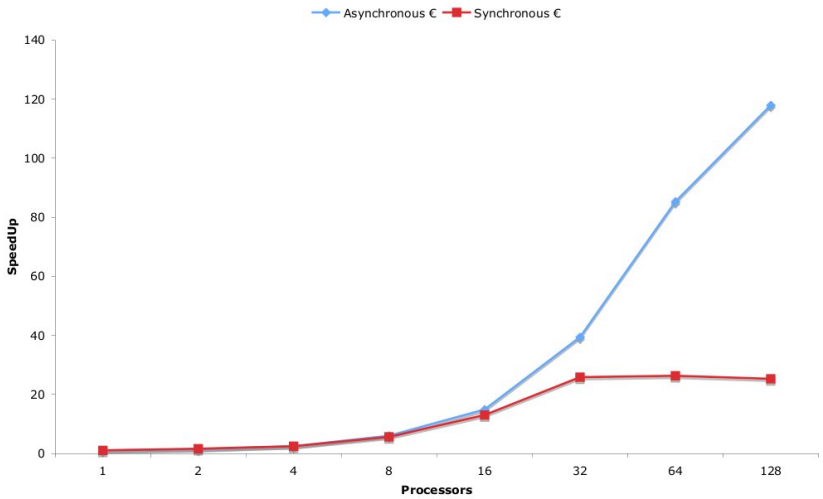


Fig. 3. Speedup for synchronous and asynchronous scheme for European options

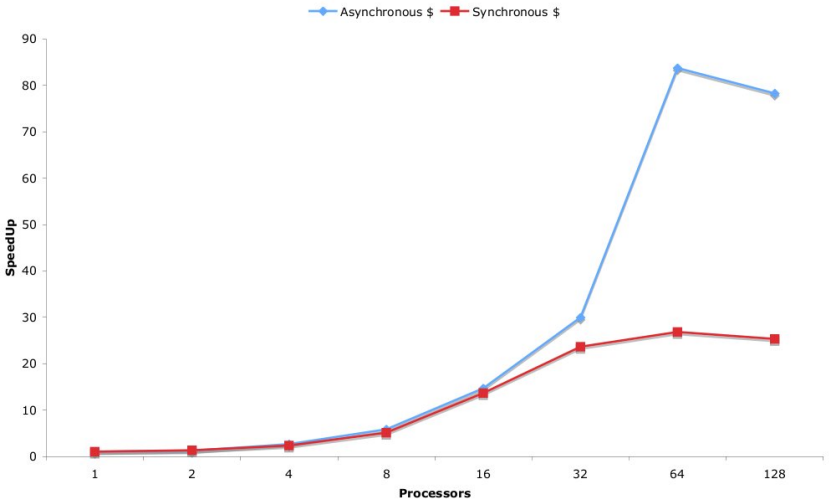


Fig. 4. Speedup for synchronous and asynchronous scheme for American options

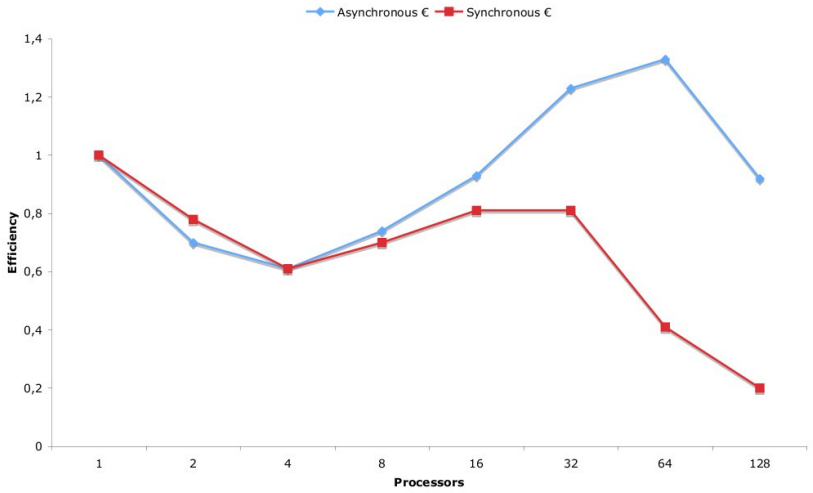


Fig. 5. Efficiency for synchronous and asynchronous scheme for European options

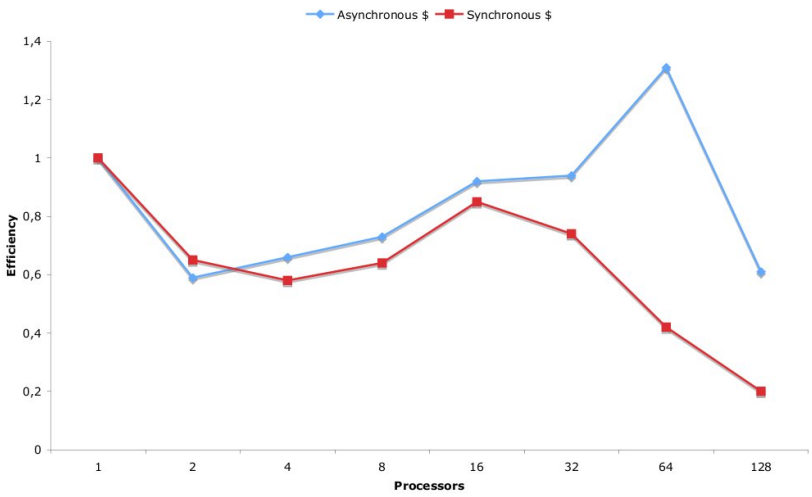


Fig. 6. Efficiency for synchronous and asynchronous scheme for American options

VI. CONCLUSION

In the presented study, for the solution of the evolution European and American options derivatives, we have studied the use and the implementation of parallel synchronous and asynchronous iterative algorithms [18] on an environment of peer-to-peer architecture. It follows from the computational experiments that the choice of the communication mode (synchronous or asynchronous) has important impact on the efficiency of the distributed methods. The computational results show that the use of P2PDC carried out on the Grid'5000 platform permits one to obtain good efficiency, particularly when the number of processors is large except for the synchronous method when the number of processors is greater or equal to 64. Moreover, using the parallel iterative asynchronous methods, compared to the synchronous ones, is well adapted to the use of heterogeneous and distant distributed large number of clusters. Finally, the difference of performance is mainly due to the weight of synchronizations between the processors.

VII. ACKNOWLEDGMENT

This study has been made possible by ANR grant: ANR-07-CIS7-011 and support of Grid'5000. We thank gratefully M. The Tung NGUYEN (LAAS –Toulouse) for providing us the P2PDC framework.

REFERENCES

[1] Paul Wilmott, Jeff Dewyne, Sam Howison, Option pricing - mathematical models and computation - Oxford financial press – 1993.

[2] P. Jaillet, D. Lamberton, B. Lapeyre, "Variational inequalities and the pricing of american options", Acta Applicandae Mathematicae, vol. 21, pp. 263 – 289, 1990.

[3] D. El Baz, G. Jourjon, "Some solutions for Peer to Peer Global Computing," in *Proceedings of 13th Euromicro conference on Parallel, Distributed and Network-Base Processing*, pp. 49-58, 2005.

[4] P. Spitéri, M. Chau, "Parallel asynchronous Richardson method for the solution of obstacle problem" in *Proceedings of the 16th Annual International Symposium on High Performance Computing Systems and Applications*, pp. 133-138, 2002.

[5] T.T. Nguyen, D. El Baz, P. Spiteri, G. Jourjon, M. Chau, "High Performance Peer-to-Peer Distributed Computing with Application to Obstacle Problem", in *Proceedings of IEEE IPDPS 2010 Conference,* Atlanta, 2010.

[6] L. Giraud, P. Spiteri, "Parallel resolution of non-linear boundary value problems", Mathematical Modeling and Numerical Analysis, vol. 25, n° 5, pp. 579-606, 1991.

[7] J.C. Miellou. P. Spiteri, "A criterion of convergence for general fixed point methods", Mathematical Modeling and Numerical Analysis, vol. 19, pp. 645-669, 1985.

[8] J.C. Miellou, D. El Baz, P. Spiteri, "A new class of asynchronous iterative algorithms with order interval", Mathematics of Computation, vol. 67, n° 221, pp. 237-255, 1998.

[9] D. El Baz, T.T. Nguyen, "A self-adaptive communication protocol with application to high performance peer to peer distributed computing", in *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, Pisa, pp. 327-333, 2010.

[10] Aumage, G. Mercier, "MPICH/Madeleine: a True Multi-Protocol MPI for High Performance Networks", *15th International Parallel and Distributed Processing Symposium* (IPDPS'01), 2001.

[11] David P. Anderson, "BOINC: A System for Public-Resource Computing and Storage", *5th IEEE/ACM International Workshop on Grid Computing*. Pittsburgh, USA, 2004.

[12] N. Andrade, W. Cirne, F. Brasileiro, P. Roisenberg, "OurGrid: An approach to easily assemble grids with equitable resource sharing", in *Proceedings of the 9th Workshop on Job Scheduling Strategies for Parallel Processing*, pp. 61-86, 2003.

[13] J.M. Ortega, W. Rheinboldt, Iterative solution of nonlinear equations in several variables, Academic press, 1970.

[14] Matti A. Hiltunen, "The Cactus Approach to Building Configurable Middleware Services", in *DSMGC2000*, Nuremberg, Germany, 2000.

[15] G.T Wong, M.A Hiltunen, R.D Schlichting, "A configurable and extensible transport protocol" in *Proceedings of IEEE INFOCOM '01*, Anchorage, Alaska, pp. 319–328, 2001.

[16] D.P. Bertsekas, J.N. Tsitsiklis, Parallel and distributed computation: numerical methods, in Prentice Hall, Englewood Cliffs, N.J., 1987.

[17] J.M. Bahi, S. Contassot-Vivier, R. Couturier, Parallel iterative algorithms: from sequential to grid computing, in Chapman & Hall/CRC, 2007.

[18] Pierre Spiteri, "Parallel optimization and financial mathematics", in *Proceedings of COSI'09 and Microsoft summer school*, Annaba, pp. 257 - 270, 2009.

Chapitre 6

Curriculum Vitæ

http://pro.garcia64.fr

Thierry GARCIA

Maître de conférences Hors Classe

✉ thierry@garcia64.fr
1 Impasse les Marguerites 64800 Arros de Nay
+33 (0)6 70 48 01 67

Situation de famille : Marié
Date et lieu de naissance : 08/07/1968 à PAU(64)

✉ thierry.garcia@cyu.fr
CY Cergy Paris Université - CY Tech Campus de Pau
2 boulevard Lucien Favre CS 77563 64075 Pau Cedex
Contact : +33 (0)6 09 60 38 83

✉ thierry.garcia@irit.fr
INP-IRIT-ENSEEIHT
2 rue Camichel BP 7122 31071 Toulouse Cedex 7

SITUATIONS PROFESSIONNELLES

- 2024 Soutenance HDR (09 janvier)
- 2021 Détachement à CY Cergy Paris Université (CYU)
  - » Composante : Grande école d'ingénieurs CY Tech Campus de Pau (CY TECH)
- 2018 Mutation Institut National Polytechnique de Toulouse (INPT)
  - » Institut de Recherche en Informatique de Toulouse (IRIT) - ENSEEIHT
  - » Composante : Institut de la Promotion Supérieure du Travail (IPST-CNAM)
- 2016 » Laboratoire d'informatique Parallélisme Réseaux Algorithmes Distribués (LI-PaRAD)
- 2015 **Maître de Conférences Hors-Classe**
- 2011 Université de Versailles Saint-Quentin en Yvelines (UVSQ) - Université Paris-Saclay
  - » Laboratoire Parallélisme, Réseaux, Systèmes, Modélisation (PRiSM)
  - » Composantes : Institut des Sciences et Techniques des Yvelines (École d'ingénieurs ISTY) et UFR des Sciences
- 2009 Disponibilité sur CDD Ingénieur de recherche (contrat ANR)
  - » Institut de Recherche en Informatique de Toulouse (IRIT) - ENSEEIHT
- 2008 Disponibilité sur CDD Ingénieur de recherche (contrat ANR)
  - » Laboratoire d'Informatique Interactive (LII) École Nationale de l'Aviation Civile (ENAC)
- 2006 **Maître de Conférences Classe Normale**
  Université de Versailles Saint-Quentin en Yvelines (UVSQ) - Université Paris-Saclay
  - » Laboratoire Parallélisme, Réseaux, Systèmes, Modélisation (PRiSM)
  - » Composantes : Institut des Sciences et Techniques des Yvelines (École d'ingénieurs ISTY) et UFR des Sciences
- 2005 Institut National des Langues et Civilisations Orientales (INALCO Paris)
- 2004 en détachement à Université de Rennes 1 sur poste Attaché Temporaire d'Enseignement et de Recherche
  - » Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA)
  - » Composantes : Institut de Formation Supérieur en Informatique et Communication (IFSIC) et École Militaire InterArmes de Saint-Cyr de Coëtquidan
- 2003 **Doctorat informatique** de l'Université de Picardie Jules Verne à Amiens (mention très honorable) : « Algorithmique parallèle du texte : du modèle systolique au modèle CGM »
  Président : Pr. Afonso FERREIRA, Rapporteurs : Pr. Frank DEHNE, Pr. Thierry LECROQ, Directeur : Dr David SEME, Examinateurs : Dr Christophe CERIN, Pr. Jean-Frédéric MYOUPO
- 2000 Centre Régional des Œuvres Universitaires et Scolaires (CROUS Amiens)
- 1999 **DEA Informatique, Productique et Imagerie Médicale** de l'Université Blaise Pascal à Clermont-Ferrand : « Étude et développement d'un micro-noyau en langage C, dédié, embarqué, réparti, temps-réel, adapté aux applications parallèles et tolérant aux fautes ». Encadrement : Pr. Kun-Mean HOU
- 1998 Disponibilité pour suivre un DEA
  Corps : Ingénieur d'études ITRF BAP E (démarrage en 1991 en CDD Technicien puis Assistant Ingénieur)
- 1996 Institut d'Administration des Entreprises (IAE Pau) - Faculté Droit, Économie et Gestion (FDEG Pau)

ACTIVITÉS COLLECTIVES

Responsabilités administratives

- 2023 Directeur délégué au développement et au pilotage de CY Tech Campus de Pau
- 2021 Directeur délégué adjoint formation de CY Tech Campus de Pau
- 2018 Fin Responsable des stages en entreprise de l'ISTY
- 2015 Chargé de missions auprès du Directeur de département du cycle ingénieur informatique de l'ISTY
- 2014 Fin mandat Directeur de département
- 2012 Fin Responsabilité de la cinquième année du cycle ingénieur informatique de l'ISTY
- 2011 Directeur de département du cycle ingénieur informatique de l'ISTY
  Responsable de la cinquième année du cycle ingénieur informatique de l'ISTY
  Responsable des stages en entreprise de l'ISTY
- 2005 Conseiller TIC-TICE auprès du Président
- 2000 Directeur du département informatique et technologies de la communication (DSI)
- 1996 Responsable du Service Informatique

Responsabilités dans les projets et la vie collective

- 2022 Co-rédacteur d'un projet CMA PIA4, participation à un projet PUI et de conventions de partenariats avec des entreprises
  Participation au projet de demande d'accréditation d'une formation sous statut étudiant et à la rédaction de lettres d'intention pour l'ouverture de l'apprentissage du cycle ingénieur informatique de CY Tech auprès de la Commission des Titres d'Ingénieur (CTI)
- 2022 Membre du CA et de la Commission Scientifique et Technique (CST – en charge de l'incubation d'entreprises) de la technopôle Hélioparc à Pau
- 2018 Fin Président des jurys VAE (Validation des Acquis de l'Expérience) pour le diplôme d'Ingénieur de l'ISTY
- 2015 Co-rédacteur du projet de demande d'habilitation de formations sous statut étudiant du cycle ingénieur informatique de l'ISTY auprès de la Commission des Titres d'Ingénieur (CTI)
- 2014 Fin mandat Directeur de département
- 2012 Porteur du projet de demande d'habilitation de la formation sous statut étudiant du cycle ingénieur informatique de l'ISTY auprès de la CTI
  Président des jurys VAE (Validation des Acquis de l'Expérience) pour le diplôme d'Ingénieur de l'ISTY
- 2011 Président délégué des jurys des 3 années du diplôme d'ingénieur de l'ISTY
- 2006 Fin mission Co-animateur du comité de pilotage des projets TICE de l'UVSQ
- 2006 Co-animateur du comité de pilotage des projets TICE de l'UVSQ

Responsabilités et activités au sein de sociétés savantes et associations professionnelles

- 2023 Fin Mandats Société Informatique de France
- 2021 Fin Mandat SPECIF-CAMPUS
- 2019 Fin mission Responsabilité éditoriale SIF
- 2016 Membre du Bureau et du Conseil d'Administration de SPECIF-CAMPUS
  Membre du Conseil d'Administration de la Société Informatique de France (SIF)
  Membre du Bureau de la Société Informatique de France (SIF)
- 2014 Responsable éditorial de la rubrique « Du côté de l'entreprise » du bulletin 1024 de la SIF

Autres Responsabilités (expertises, jurys de concours, anciennes responsabilités)

- 2023 Membre du vivier d'experts HCERES
- 2023 Expert pair du cluster Informatique pour l'AEQES[1] (voir détails (a))
- 2020 Fin Jury Bernard Novelli
- 2018 Expert pair du diplôme de Bachelier Finalité «Techniques Graphiques» pour l'AEQES (voir détails (a))
- 2014 Jury pour le prix Bernard Novelli dans le cadre des trophées Tangentes
- 2013 Co-président et expert pair du diplôme de Bachelier Finalité «Techniques Graphiques» pour l'AEQES (voir détails (a))
- 2008 Fin mission expert ITRF
- 2005 Experts mandatés pour l'organisation des jurys des concours ITRF

(a) Cette agence gouvernementale a pour objectif essentiel d'évaluer les programmes, d'énoncer toute recommandation utile en vue d'améliorer la qualité, de soutenir les établissements dans leur dynamique d'amélioration continue, dans la poursuite des actions mises en place et dans le développement d'outils de pilotage. Dans ce cadre, j'ai participé à l'expertise comme expert pair mais aussi co-président du comité d'évaluation d'établissements réalisant des cours en journée mais aussi le soir, à l'élaboration de recommandations suite à l' évaluation de la qualité du diplôme de Bachelier Finalité «Techniques Graphiques» proposé par des Hautes-Écoles et des Établissements de Promotion Sociale belge (2012) ; à la rédaction des rapports d'évaluation, d'un rapport transversal commun à tous les établissements (http://www.aeqes.be/rapports_details.cfm?documents_id=311) (2013) et à la présentation du rapport transversal auprès du comité de gestion de l'AEQES (2013). En 2018, j'ai procédé comme expert pair à l'évaluation du suivi de ces recommandations qualités (2018) pour ce même bachelier. En 2022-2023, j'ai participé comme expert pair à l'évaluation du cursus informatique d'établissements qui a donné lieu à un rapport transversal en décembre 2023 (https://www.aeqes.be/actualites_details.cfm?news_id=251).

Anciennes responsabilités

Missions Ingénieur d'Études : développement et modernisation des Systèmes d'Information (SI) ainsi que l'intégration de ces technologies à l'enseignement (TICE) ; proposition de projets de contractualisation suivies de la gestion budgétaire des dotations (environ 1 000 000 €) ; gestion du budget du service et des RH (notation, fiche de poste, recrutement) ; gestion des ressources logistiques (création de bâtiments, moyens numériques) ; animation de réunions et relation avec les partenaires académiques ; conseils de direction, Comité de Pilotage des Projets Informatiques, conventions avec les universités partenaires ; gestion des moyens informatiques, de l'architecture réseau et des matériels actifs ainsi que la politique de sécurité (responsable de la sécurité des systèmes d'information (RSSI), correspondant Renater) ; conduite de la procédure d'appel d'offres dans le cadre de marchés publics (CCTP, CCAP) ; développement d'applications et modernisation du site internet et extranet (ENT).

[1] *Agence pour l'Évaluation de la Qualité de l'Enseignement Supérieur belge*

ACTIVITÉS PÉDAGOGIQUES

Animation

Les animations liées aux activités pédagogique ont été :

- création des modalités de contrôle des connaissances ;
- création et mise en place de la validation des acquis de l'expérience (VAE), accompagnement des candidats dans la démarche VAE, évaluation des expériences des candidats pour l'attribution du diplôme, soutenance et jury ;
- gestion des offres de stages, d'alternances pour les étudiants, relations avec les entreprises et gestion numérique du processus des stages - de l'installation en entreprise à la soutenance avec les visites obligatoires au sein des entreprises permettant de créer des liens avec les professionnels du secteur ;
- projets de demande d'habilitation, rédaction collaborative, construction de maquettes pédagogique avec l'obligation de respecter les réglementations dans le cadre de demande d'habilitation de la Commission des Titres d'Ingénieurs, amélioration continue de la maquette pédagogique ;
- organisation de campagnes d'évaluation des enseignements ;
- gestion des étudiants (absences, médiation, recrutement des nouveaux élèves ingénieurs) ;
- organisation de jurys, présidence et délivrance des attestations de réussite et des diplômes ;
- animation de réunions avec les étudiants et les enseignants ;
- pilotage de la pédagogie, de l'emploi du temps, des enseignants et du contrôle des charges d'enseignement ;
- pilotage de l'affectation des moyens pédagogiques, en liaison avec les enseignants, la scolarité et les gestionnaires des ressources : salles, moyens matériels, technologiques ;
- participation aux actes de communication (salons, conférences, JPO, JPE, …) ;
- porteur de projets pour une double-diplomation entre l'ISTY et l'Université de Coventry (UK) et entre l'ISTY et l'UQAC (Canada).

Enseignements

Type	***Nom***	***Lieu***
École d'Ingénieur	CY TECH	Pau
	IPST-CNAM	Toulouse
	ISTY	Vélizy
	ENSEEIHT-INP	Toulouse
	IFSIC	Rennes
École Militaire	Ecole Militaire InterArmes de Saint-Cyr	Coëtquidan
Université	UFR Sciences UVSQ	Versailles
	FDEG-IAE	Pau
Entreprises	Crédit Agricole, Préfecture, Ministère Agriculture	Pau

Enseignements et suivis

Supports : Les étudiants ont un support de cours sur leur environnement numérique (ENT, ENF)).

Suivi des stages et des alternants : le tuteur organise le suivi, la visite en entreprise, la lecture du rapport et la soutenance.

Lieu : grande école d'ingénieur CY TECH– Auditeurs : Étudiants, Alternants

Filières	**Niveau**	**Enseignements et suivis**	**Type d'enseignements**	**Plus d'informations**
Informatique	2e année (L2)	Scripts shell	TD, Projet	PréIng2
Informatique	3e année (L3)	Algorithmique procédurale	TD	ING1
		Système d'Exploitation	TD	ING1
		Commande Unix	TP	ING1
Informatique	4e année (M1)	Architecture et Programmation Parallèle	TD	ING2
		Programmation Système et Réseau	Cours, TD/TP, Projet	ING2
Informatique	5e année (M2)	Heterogeneous Programming	Cours, TD, Projet	ING3

Lieu : école d'ingénieur IPST-CNAM – Auditeurs : Alternants

Filières	Niveau	Enseignements et suivis	Type d'enseignements	Plus d'informations
Informatique	3e année (L3)	Système d'exploitation, Linux	Cours, TD/TP	UTC502, NSY103
		Tutorats alternants		
Informatique	4e année (M1)	Modélisation, Optimisation, Complexité	Cours, TD	RCP105, RCP103
		Évaluation de la performance	Cours, TD/TP	
		Tutorats alternants		

Lieu : école d'ingénieur ISTY et université UFR Sciences – Auditeurs : Étudiants, Alternants

Filières	Niveau	Enseignements et suivis	Type	Plus d'informations
Informatique	Cycle prépa 1ère année (L1)	Algorithmique et programmation C	TD/TP	CPI
	Cycle prépa 2e année (L2)	Algorithmique et programmation C	TD/TP	
		Dév Web, Système, BD	TD/TP	
Informatique	3e année (L3)	Mise à niveau UNIX, C et Algorithme	TD/TP	Ingénieur Informatique
		Système d'exploitation	Cours et TD/TP	
		OO C++, JAVA, Python	Cours et TD/TP	
		Base de Données	Cours et TD/TP	
		Algorithmique	Projet	
		OO	Projet	
		Système d'exploitation	Projet	
		Suivis de stagiaires		
	4e année (M1)	Suivis de stagiaires		
	5e année (M2)	Suivis de stagiaires		
Mécatronique	3e année (L3)	Suivis d'alternants		Ingénieur Mécatronique
	4e année (M1)	Suivis d'alternants		
	5e année (M2)	Suivis d'alternants		
Sys Élec Emb	3e année (L3)	Mise à niveau en langage C	Cours et TD/TP	Ingénieur Systèmes Électroniques Embarqués
		Programmation orientée objets C++	Cours et TD/TP	
Informatique	L1	Fondement informatique	TD	
	L3	Algorithmique avancé	TD	
	M1	Programmation parallèle	TD/TP	
	M2 Recherche	Méta-Heuristique Méthodes Exactes	Cours	

Entre 2009 et 2011, en disponibilité - Vacations.

Lieu : école d'ingénieur IPST-CNAM et école d'ingénieur ENSEEIHT – Auditeurs : Étudiants, Alternants

Filières	Niveau	Enseignements et suivis	Type d'enseignements	Plus d'informations
Informatique	3e année (L3)	Système d'exploitation, Linux	Cours, TD/TP	NSY103
Informatique	4e année (M1)	Processus Stochastiques	TP	
		Système d'Exploitation centralisés	TP	

Anciennement, ATER et Ingénieur d'études ITRF BAP E - Vacations

Lieu : école d'ingénieur IFSIC, Ecole Militaire InterArmes de Saint-Cyr et FDEG-IAE – Auditeurs : Étudiants

Filières	Niveau	Enseignements et suivis	Type d'enseignements	Plus d'informations
Informatique	3e année (L3)	Algo progr. JAVA	TD/TP	
	5e année (M2)	Système d'Exploitation avancé	TD/TP	
Informatique	3e année (L3)	Algorithme et programmation	TD/TP	
		Projet Génie Logiciel		
Informatique	4e année (M1)	Processus Stochastiques	TP	
		Système d'Exploitation centralisés	TP	
Informatique	L1 à M2	Algo, Bureautique, Internet	TD/TP	
	M2 Pro	Internet et communication	TP	

Quelques détails des enseignements

Algorithmique, Programmation et Langage C : Bases d'algorithmique et de programmation, fonctionnement logique d'un ordinateur, structures de données de base, type de base, variable, expression, affectation, élément de logique, test (instruction conditionnelle), boucle, structure de données (tableau), structure de programmes, chaîne de caractères, structure de données et programmes, opérations élémentaires associées utilisées pour spécifier un algorithme, algorithmes de recherche, listes chaînées : simples, doubles, circulaires, avec sentinelle, pile, file, arbres, arbres binaires de recherche, complexité, équilibrage, programmation récursive.

Analyse, Conception orientée objet et Langages C^{++}, JAVA, PYTHON : Notion d'objets, encapsulation, surcharge, héritage, polymorphisme, instructions de composition : séquence, conditionnelle, itération, notion d'invariant, sous-programmes et modularité, composants logiciels, « boîtes noires », pré et post-conditions, structures de données séquentielles, tableaux. Définition et utilisation des langages OO, type de base, variable, expression, affectation, élément de logique, test (instruction conditionnelle), boucle, structure de données (tableau), structure de programmes, chaîne de caractères, structure de données et programmes, objets simples et complexes et leurs propriétés, patterns.

Base de Données : Architecture et objectifs des SGBD, Fichiers, hachage et indexation, Modèle relationnel, Création et manipulation d'une base de données, Langage SQL, interrogation et mise à jour, Intégrité et confidentialité des données, Optimisation élémentaire et vues.), SGBD (Mysql, SQL, phpMyAdmin).

Architecture et programmation parallèle : Différentes architectures parallèles, construction d'algorithmes parallèles, librairie MPI, openMP, CUDA, communications synchrones et asynchrones, tests sur HPC, grilles, GPU - Heterogeneous Programming.

Systèmes d'Exploitation : Bases essentielles des systèmes d'exploitation (généralités, complexité, fiabilité, maintenabilité, modularité, portabilité, structure en couches) et mécanismes fondamentaux des systèmes d'exploitation centralisés, répartis et temps réel. Développement d'applications multiprocessus en utilisant des outils de communication, de synchronisation et des primitives "noyau" (processus, fichiers, mémoire virtuelle, gestion des E/S). Scripts, processus, CIP, … noyau Linux et Unix.

Réseaux : Protocoles et normes télécoms, Protocoles IP, Technologies radiofréquences, Technologies numériques, Technologies analogiques, Technologie des fibres optiques, Techniques de multiplexage, Logiciels de modélisation et simulation, Traitement du signal (bases). Architecture réseau, Réseaux de télécommunication, Architectures de plateformes de services, de réseaux de téléphonie fixe, réseaux de téléphonie mobile, réseaux informatiques et télécoms, Internet,de réseaux multi-services.

Algorithmique avancé : Concepts de base de la théorie des graphes, parcours des graphes (en largeur, en profondeur), connexité, forte connexité (algorithme de TARJAN), Eulérien et Hamiltonien, algorithmes de plus courts chemins (Ford, Dijkstra, Bellman, Floyd), définitions et propriétés d'arbres, arbres couvrants de poids minimum (Prim, Kruskal), réseaux de flots : flots maximums, coupes minimales, flots de coût minimal : algorithme de Ford-Fulkerson, fermeture transitive : Algorithme de Roy -Warshall, méthode Diviser pour Régner et méthode gloutonne, Ordonnancements (méthodes PERT et MPM et problèmes d'atelier), Introduction à la complexité des algorithmes et des problèmes, Réseaux de Petri (RdP), exemples de modélisation de systèmes dynamiques à événements discrets.

Développement Web : HTML, PHP, CSS.

Évaluation de performances et sûreté de fonctionnement : Modélisation markovienne, Chaînes de Markov à temps discret (CMTD) et a temps continu (CMTC), chaîne de Markov immergée (EMC), Régime transitoire, régime permanent, ergodicité, distribution stationnaire. Equations de balance globale, Files d'attente : file M/M/S, file M/G/1, Loi de Little, formule de Pollaczek-Khintchine, Les réseaux de file d'attente (RFA) à forme produit (monoclasses/multi-classes, ouverts/fermés) : réseaux de Jackson, Gordon-Newell et BCMP, Equation de trafic, Algorithme de la valeur moyenne (MVA), Réseaux de Petri stochastiques : le modèle GSPN, Évaluation prévisionnelle de la sûreté de fonctionnement : fiabilité, disponibilité, Limites de la modélisation markovienne, Simulations stochastiques (méthodologie, validité, coût).

Méta-Heuristiques et Méthodes Exactes : Présentation de la méthode du recuit simulé.

Processus stochastiques : projet de simulation d'une gare de péage à l'aide d'un langage de simulation de processus stochastiques afin de mettre en pratique le cours qui permet de modéliser et de prévoir l'évolution de phénomènes aléatoires (applications aux phénomènes d'attente, à la fiabilité et aux réseaux).

Projets : Algorithmique et Programmation Orienté Objet, concevoir et implémenter un algorithme pour traiter un problème complexe, Programmation parallèle numérique .

Bureautique, Algorithmique et Internet : Architecture, Système d'exploitation, Réseau, Langage Pascal, Traitement de texte , Tableur , BD, Messagerie, Navigation et Sécurité.

ACTIVITÉS SCIENTIFIQUES

Thématiques de recherche

Mes thématiques de recherche concernent la simulation numérique appliquée à la résolution de problèmes complexes de très grandes tailles pour des applications pluridisciplinaires. Les problèmes abordés relèvent des mathématiques, de la physique, de la bio-informatique, de la mécanique, du génie des procédés, de l'économie et de l'algorithmique du texte. Mes contributions se situent d'une part vers l'implémentation de méthodes calculs parallèles synchrones et asynchrones en particulier sur les tests d'arrêt des itérations dans un cadre de calculs asynchrones et d'autre part sur les enseignements que l'on peut tirer du comportement des expérimentations en liaison avec l'architecture des machines utilisées et la rapidité du réseau d'inter-connexions. L'utilisation de ces méthodes est intéressante pour la simulation numérique d'applications industrielles et les applications traitées ont concerné :

- des problèmes aux dérivées partielles fortement non-linéaire par la méthode de de multi-domaines - voir Publications [J7] et [J8],

- des problèmes couplés en biologie concernant la séparation de protéines par électrophorèse (le modèle est représenté par l'équation de Navier-Stokes couplée à une équation de convection-diffusion et à une équation de diffusion) - voir Publications [J2],

- des problèmes formulés sous forme complémentaire appliqué à l'équation d'Hamilton-Jacobi-Bellman intervenant par exemple en traitement d'images ou à des problèmes avec contraintes - voir Publications [J3],

- des problèmes d'interaction fluide-structure modélisé par l'équation de Navier-Stokes couplée à l'équation de Navier -voir Publications [J3],

- un problème de mécanique de fluide - voir Publications [J4],

- un problème de convection-diffusion appliqué au débruitage et à la segmentation d'images dynamiques TEP – voir Publications [C15],

- un problème de solidification de l'acier modélisé par une équation de la chaleur prenant en compte des phénomènes de rayonnement à la frontière - voir Publications [J6],

- un problème intervenant en mathématiques financières modélisé par l'équation de Black Scholes - voir Publications [C11],

- un problème de détection de répétition intervenant dans l'algorithmique du texte, en biologie moléculaire (séquences ADN) et en compression de données – voir Publications [J1].

A partir des modélisations effectuées par les experts du domaine, j'ai contribué à l'implémentation d'algorithmes parallèles ou distribués, asynchrones ou synchrones permettant la résolution des problèmes citées précédemment et réaliser ainsi des simulations en faisant appel au calcul haute-performance sur des architectures dédiées (HPC, grille, cluster, simulateur peer to peer). Les analyses des expérimentations ont permis de montrer un gain de temps d'exécution non négligeable pour les méthodes asynchrones par rapport à celles synchrones dans le cas de machines éloignées géographiquement.

De plus, du fait de la particularité pluridisciplinaire de cette thématique de recherche, mes travaux peuvent s'intégrer dans de nombreux autres domaines pluridisciplinaires et thèmes scientifiques, pour lesquels des compétences, en calcul intensif, parallèle ou en architectures distribuées seraient nécessaires.

Travaux de recherche antérieurs

2009–2011 Le projet CIP Calcul Intensif Pair à pair – http://www.laas.fr/CIS-CIP/, initié au sein de l'Institut de Recherche en Informatique de Toulouse (IRIT) à l'École Nationale Supérieure d'Électrotechnique, d'Électronique, d'Hydraulique d'Informatique et des Télécommunications (ENSEEIHT) – Institut National de Polytechnique (INP) (équipe TCI – **T**raitement et **C**ompréhension d'**I**mages) (**Pierre Spitéri** – pierre.spiteri@enseeiht.fr), avait pour objectif de proposer des outils et des environnements pour la mise en œuvre de calculs intensifs sur un simulateur d'architecture de réseaux pair à pair. Il a permis de développer un environnement pour la mise en œuvre de calculs intensifs sur une simulation d'architecture pair à pair totalement décentralisée, qui est conçu pour les grandes applications de simulation numérique et d'optimisation présentant un parallélisme de tâche et nécessitant des communications fréquentes entre les machines. On s'est intéressé essentiellement à la résolution de problèmes au moyen de méthodes itératives parallèles ou distribuées. Afin de faciliter la programmation et d'optimiser les performances, l'environnement repose sur un jeu d'opérations de communications réduit ; par ailleurs le programmeur n'a pas à spécifier le mode de communication, ce dernier est fixé par le protocole de manière auto adaptative et dynamique en fonction d'éléments de contexte de la couche réseau et d'indications sur le schéma itératif de calcul préféré du programmeur relevant de la couche application. Des démonstrateurs ont été réalisés pour la recherche opérationnelle et la simulation numérique. Les partenaires du projet étaient le LAAS-CNRS (Toulouse), le LIFC (Montbéliard), le MIS ex LaRIA (Amiens), et EuroMedTextile (association d'industriels).

2008-2009 Le projet Istar : http://www.i-star.fr (2007-2010), initié au sein du Laboratoire d'Informatique Interactive à l'École Nationale de l'Aviation Civile (**Stéphane Chatty** – stephane.chatty@enac.fr), visait à développer et à évaluer une solution pour l'interopérabilité des interfaces graphiques, sur la base d'un « moteur d'exécution » (machine virtuelle) exécutant des programmes décrits selon un modèle sémantique dédié aux composants interactifs. Le modèle sémantique a été conçu sur la base d'un modèle qui organise une application interactive en un arbre de composants dont les feuilles sont des objets graphiques, des comportements, des actions, ou des algorithmes. Les partenaires du projet étaient le LRI (Orsay), IntuiLab (PME-Toulouse) et Anyware Technology (PME-Toulouse).

- Durant mon doctorat, j'ai travaillé sur des problèmes d'algorithmique du texte comme le problème de recherche de la plus longue sous-suite croissante, de la plus longue sous-suite commune à deux mots, du plus long suffixe répété en chaque caractère d'un mot et de répétitions. En utilisant des algorithmes parallèles sur un modèle à grains fins (le modèle systolique), le but était de créer une passerelle entre ce modèle et un modèle à gros grains (le modèle CGM – Coarse Grained Multicomputers) afin de pouvoir utiliser des clusters d'ordinateurs. Un algorithme développé pour le modèle CGM est constitué de calculs locaux utilisant des algorithmes séquentiels optimaux et de rondes de communication dont le nombre doit être indépendant de la taille des données à traiter. Ce modèle est indépendant des architectures réelles et permet de réutiliser des algorithmes séquentiels efficaces. De plus, la charge de travail n'étant pas la même sur chaque processeur lors du traitement des solutions, il a été proposé une solution d'équilibrage de charges. Enfin, une extrapolation des résultats de nos travaux a été proposée afin de prédire quelles sont les adaptations envisageables des architectures systoliques au modèle CGM.

  Durant mon DEA, j'ai travaillé sur un micro-noyau embarqué, temps-réel et distribué. Mon travail a consisté en la gestion des communications entre des capteurs (considérés comme des objets intelligents communicants) modélisés par des processus tolérant aux fautes.

Travaux de recherche en cours et perspectives de travaux

En partenariat avec avec INP | IRIT-ENSEEIHT – Pierre Spitéri et l'Université Badji Mokhtar Annaba – Abdelhamid Laouar et Skikda University | LAMAHIS – Amel Hannache, résolution d'équations aux dérivées partielles soumis à des contraintes modélisant des problèmes de plaques ;

Développement et simulation sur grilles d'algorithmes parallèles à partir d'une modélisation mathématique d'un problème de Recherche Opérationnelle concernant les réseaux de serveurs téléphoniques en partenariat avec INP | IRIT-ENSEEIHT – Pierre Spitéri et l'Université Badji Mokhtar d'Annaba.

Les perspectives de travaux sont orientées vers l'établissement d'un lien entre les grilles de calcul et le Cloud, la gestion des pannes temporaires pour donner un cadre de tolérance aux fautes de nos algorithmes asynchrones, une utilisation optimale du cloud avec son système à couches et des benchmarks pour d'autres types de non-linéarités (navier stokes, problèmes complémentaires, problèmes pseudo linéaires univoques, GMRES sur cloud).

Animation et encadrement

Encadrement

- Co-encadrement d'une future thèse (2021 -) ENSEEIHT-IRIT – Mohammed Amine Rahhali - encadrement 60 %
- Co-encadrement d'un post-doctorat (Skikda University | LAMAHIS - Ghania Khenniche) - encadrement 25 %;
- Co-encadrement du doctorat de (2015 – 2017) Vincent Partimbene avec l'ENSEEIHT-IRIT en partenariat avec l'Entreprise SEGULA Technologies (CIFRE) - encadrement 20 %
- Co-encadrement d'un doctorat (Ghania Khenniche avec l'Université Badji Mokhtar Annaba) - encadrement 20 % ;
- *Depuis 2005*, encadrements d'élèves ingénieurs ou de masters en stage ou en apprentissage en entreprise ;
- *2013* Encadrements d'élèves ingénieurs sur un projet d'analyse du mouvement 3D en collaboration avec la Fondation Garches et l'APHP (Assistance Publique des Hôpitaux de Paris) – CHU Raymond Poincare.

Gestion de projets

- Participation à des groupes de travail avec les partenaires académiques et industriels (LAAS-Laboratoire(31), MIS-Laboratoire(80), EuroMed-PME(80), LIFC-Laboratoire(25)) en relation avec le projet CIP. (2010-2011) ;

- Participation à des groupes de travail avec les partenaires académiques et industriels (LRI-Laboratoire(91), IntuiLab-PME(31), Anyware Technologies-PME(31)) du projet Istar. (2008-2009) ;
- Participation à des réunions techniques et administratives avec les partenaires académiques, industriels (Aerospace Valley (31 et 33), Stantum-PME (33), Thales Avionics (33), IntuiLab-PME (31)) et financiers (Régions Aquitaine et Midi-Pyrénées) du projet Share-It.

Rayonnement

Comité de programme

Membre du comité de programme de la conférence « IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC) »

Membre du comité de programme de la conférence « International Conference on Pattern Recognition Systems (ICPRS) »

Membre du comité de programme et/ou d'organisation de journées de la Société Informatique de France (SIF)

Reviewer

Journal of Parallel and Distributed Computing, Journal of Computational Science, PDSEC, Artificial Intelligence Review (AIRE)

Animation d'ateliers

2018 Observatoire Midi-Pyrénées : **membre invité** pour animer des formations dans le cadre des journées HPCpourTous

2017 CNRS : **membre invité** pour animer des ateliers et un groupe de travail lors de la 4ème édition des Journées nationales du DEVeloppement logiciel http://devlog.cnrs.fr/jdev2017/t8

Bilan statistique des publications / interventions

Type de publication / intervention	**Nombre**
Journaux internationaux	9
Conférences Proceedings	16
Rapports de recherche	2
Textes de vulgarisation	2
Ateliers et groupe de travail	6
Thèse	1

Liste des publications (format : [TypeN°] Détails de la Publication, Liens doi et/ou hal et Impact Factor)

Journaux internationaux

[J9] M.A. Rahhali, T. Garcia, P. Spiteri, Performances analysis of parallel asynchronous and synchronous algorithms on cloud architecture for PDE,. Advances in Engineering Software, Elsevier, 2023, 186 doi : 10.1016/j.advengsoft.2023.103550

[J8] Thierry Garcia, Pierre Spiteri, Lilia Ziane-Khodja, Raphaël Couturier. Coupling parallel asynchronous multisplitting methods with Krylov methods to solve pseudo-linear evolution 3D problems. Journal of Computational Science, Elsevier, 2021, 51. doi : 10.1016/j.jocs.2021.101303

[J7] Thierry Garcia, Pierre Spiteri, Lilia Ziane-Khodja, Raphaël Couturier. Solution of univalued and multivalued pseudo-linear problems using parallel asynchronous multisplitting methods combined with Krylov methods. Advances in Engineering Software, Elsevier, 2020, 153. doi : 10.1016/j.advengsoft.2020.102929

[J6] Thierry Garcia, Ghania Khenniche, Pierre Spiteri. Behavior of parallel two-stage method for the simulation of steel solidification in continuous casting, Advances in Engineering Software, Elsevier, 2019, 131, pp.116-142. doi : 10.1016/j.advengsoft.2018.11.012 hal : hal-02381883

[J5] Vincent Partimbene, Thierry Garcia, Pierre Spiteri, Philippe Marthon, Leon Ratsifandrihana. Asynchronous multi-splitting method for the solution of fluid-structure interaction problems, Advances in Engineering Software, Elsevier, 2019, 133, pp.76-95. doi : 10.1016/j.advengsoft.2019.03.001 hal : hal-02381885

[J4] Ming Chau, Laouar Abdelhamid, Thierry Garcia, Pierre Spiteri. Grid solution of problem with unilateral constraints. *Numerical Algorithms*, Springer Verlag, 2017, 75 (4), pp.879-908. doi : 10.1007/s11075-016-0224-6 hal : hal-01450772

[J3] Ming Chau, Thierry Garcia, Pierre Spiteri. Asynchronous Schwarz methods applied to constrained mechanical structures in grid environment. *Advances in Engineering Software*, Elsevier, 2014, 74, pp.1–15. doi : 10.1016/j.advengsoft.2014.03.005 hal : hal-01588517

[J2] Ming Chau, Thierry Garcia, Pierre Spiteri. Asynchronous grid computing for the simulation of the 3D electrophoresis coupled problem. *Advances in Engineering Software*, Elsevier, 2013, 60–61, pp.111–121. doi : 10.1016/j.advengsoft.2012.11.010 hal : hal-01588518

[J1] Thierry Garcia, David Semé. A Coarse-Grained Multicomputer algorithm for the detection of repetitions. *Information Processing Letters*, Elsevier, 2005, 93 (6), pp.307–313. doi : 10.1016/j.ipl.2004.12.004 hal : hal-01588523

Conférences internationales avec comité de lecture et publication des actes

[C16] M.A. Rahhali, T. Garcia, P. Spiteri , Behaviour of asynchronous parallel iterative algorithms on cloud computing type architectures, in B.H.V. Topping, P. Iványi, (Editors), Proceedings of the Eleventh International Conference on Engineering Computational Technology, Civil-Comp Press, Edinburgh, UK, Online volume: CCC 2, Paper 8.2, 2022, doi: 10.4203/ccc.2.8.2

[C15] P. Gonzalez, Thierry Garcia, Pierre Spiteri, Pierre Tauber. Simultaneous filtering and sharpening of vector- valued images with numerical schemes. *Proceedings of the International Conference on Pattern Recognition Systems*, Tours, France, pp.58-63, 2019, doi : 10.1049/cp.2019.0249 hal : hal-02403752

[C14] Vincent Partimbene, Thierry Garcia, Pierre Spiteri, Philippe Marthon, Leon Ratsifandrihana. A Parallel Method for the Solution of Fluid-Structure Interaction Problems. *Proceedings of the 5th International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Pécs, Hungary, 111 (20), 2017, doi : 10.4203/ccp.111.20 hal : hal-01588515

[C13] Ghania Khenniche, Thierry Garcia, Pierre Spiteri. Parallel Simulation of Steel Solidification. *Proceedings of the 5th International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Pécs, Hungary, 111 (22), 2017, doi : 10.4203/ccp.111.22 hal : hal-01588516

[C12] Thierry Garcia, Ming Chau, Pierre Spiteri. Computation of Protein Separation using a Grid Environment. *Proceedings of the Second International Conference on Parallel, Distributed, Grid and Cloud Computing for Engineering*, Ajaccio, France, 2011, doi : 10.4203/ccp.95.82 hal : hal-00690837

[C11] Thierry Garcia, Ming Chau, Pierre Spiteri. Synchronous and Asynchronous Distributed Computing for Financial Option Pricing. *Proceedings of the Computational Science and Its Applications – ICCSA 2011: International Conference*, Santander, Spain, Springer, Part II, pp.664–679, 2011, doi : 10.1007/978-3-642-21887-3_50 hal : hal-01588521

[C10] Ming Chau, Thierry Garcia, Pierre Spiteri. Parallel asynchronous Schwarz alternating method for obstacle problems on grid computing. *Proceedings of the 13th International Conference on Civil, Structural and Environmental Engineering Computing*, Chania, Greece, 2011, doi : 10.4203/ccp.96.118 hal : hal-01588519

[C9] Ming Chau, Thierry Garcia, Abdelhamid Laouar, Pierre Spiteri. Subdomain Solution of Problem with Unilateral Constraints in Grid Environments. *Proceedings of the Data Management in Grid and Peer-to-Peer Systems: 4th International Conference, Globe*, Toulouse, France, Springer, pp.108–119, 2011, doi : 10.1007/978-3-642-22947-3_10 hal : hal-01588520

[C8] Thierry Garcia, Ming Chau, The Tung Nguyen, Didier El Baz, Pierre Spiteri. Asynchronous peer-to-peer distributed computing for financial applications. *IPDPSW*, Anchorage, Alaska, United States, IEEE, pp.1458-1466, 2011, doi : 10.1109/IPDPS.2011.292 hal : hal-00688400

[C7] Ming Chau, Thierry Garcia, Pierre Spiteri. Proteins Separation in Distributed Environment Computation. Springer. *ICCSA 2011 : 11th International Conference on Computational Science and Its Applications*, Santander, Spain, Springer, 6783, pp.648-663, 2011, LNCS, doi : 10.1007/978-3-642-21887-3_49 hal : hal-00690910

[C6] Ming Chau, Abdelhamid Laouar, Thierry Garcia, Pierre Spiteri. Parallel solution of problem with unilateral constraints. *10th IMACS International Symposium on Iterative Methods in Scientific Computing*, Marrakech, Morocco, 2011.

[C5] Thierry Garcia, David Semé. A Load Balancing Technique for Some Coarse-Grained Multicomputer Algorithms. *Proceedings of the 21st International Conference on Computers and Their Applications, CATA 2006*, Seattle, United States, pp.301–306, 2006, hal : hal-01588522

[C4] Thierry Garcia, Jean Frédéric Myoupo, David Semé. A Coarse-Grained Multicomputer Algorithm for the Longest Common Subsequence Problem. *Proceedings of the 11th Euromicro Conference on Parallel, Distributed and Network-Based Processing, Euro-PDP*, Gênes, Italy, IEEE Computer Society, pp.349–356, 2003, doi : 10.1109/EMPDP.2003.1183610 hal : hal-01588524

[C3] Thierry Garcia, David Semé. A Coarse-Grained Multicomputer Algorithm for the Longest Repeated Suffix Ending at Each Point in a Word. *Proceedings of the Computational Science and Its Applications —- ICCSA 2003: International Conference*, Montreal, Canada, Springer, Part II, 2668, pp.239–248, 2003, doi : 10.1007/3-540-44843-8_26 hal : hal-01588525

[C2] Thierry Garcia, Jean Frédéric Myoupo, David Semé. A work-optimal CGM algorithm for the LIS problem. *Proceedings of the thirteenth annual ACM symposium on Parallel algorithms and architectures – SPAA 01*, Crete Island, Greece. pp.330–331, 2001, doi : 10.1145/378580.378756 hal : hal-01588526

[C1] K.M. Hou, Thierry Garcia, Emmanuel Mesnard, Philippe Kauffmann. Distributed Real-time Micro-kernel with Fault-tolerance: DREAM. *Proceedings of The 4th Annual International Conference on Industrial Engineering Theory, Application and Practice*, San Antonio, United States, 1999, hal : hal-01588527

Conférences nationales avec comité de lecture et publication des actes

[N1] Thierry Garcia. Le problème de la plus longue sous-suite commune à deux mots : du modèle systolique au modèle CGM, *MAnifestation des JEunes Chercheurs du domaine des STIC*, Marseille, France, 2003.

Rapports de recherche

[R2] Thierry Garcia, Ming Chau, The Tung Nguyen, Didier El-Baz, Pierre Spiteri. Asynchronous Peer-to-peer Distributed Computing for Financial Applications LAAS-CNRS 11037, 2011.

[R1] Thierry Garcia, Ming Chau, The Tung Nguyen, Didier El-Baz, Pierre Spiteri. Peer-to-Peer distributed computing with application to European option LAAS-CNRS 10541, 2010.

Texte de vulgarisation

[V2] Thierry Garcia, Rubrique Action : L'informatique, une nécessité dans Tangente Éducation TE40, 2017.

[V1] Thierry Garcia, Brève dans la revue Tangente Éducation TE38, 2016

Ateliers et groupe de travail

[AG6] Thierry Garcia, Pierre Spiteri, Portage d'un code sous MPI, *Journées HPCpourTous* Observatoire Midi-Pyrénées, Toulouse France, 2018

[AG5] Philippe Wautelet, Thierry Garcia, Initiation MPI communication collective, *Journées HPCpourTous* Observatoire Midi-Pyrénées, Toulouse France, 2018

[AG4] Thierry Garcia, Philippe Wautelet, Initiation MPI communication point à point, *Journées HPCpourTous* Observatoire Midi-Pyrénées, Toulouse France, 2018

[AG3] Pierre Spiteri, Thierry Garcia, Retour d'expériences en programmation parallèle asynchrone, applications et modélisation mathématique, *4ème édition des Journées nationales du DEVeloppement logiciel* CNRS, Marseille France, 2017, http://devlog.cnrs.fr/jdev2017/t8.gt09

[AG2] Thierry Garcia, Pierre Spiteri, Portage d'un code sous MPI, *4ème édition des Journées nationales du DEVeloppement logiciel* CNRS, Marseille France, 2017, http://devlog.cnrs.fr/jdev2017/t8.a10

[AG1] Thierry Garcia, Les bases de MPI, *4ème édition des Journées nationales du DEVeloppement logiciel* CNRS, Marseille France, 2017, http://devlog.cnrs.fr/jdev2017/t8.ap03

Thèse - PhD

[T1] Thierry Garcia. Algorithmique parallèle du texte : du modèle systolique au modèle CGM. Université de Picardie

Bibliographie

[1] J.L Afonso and M.J. Clifton. Coupling between transfer phenomena in continuous-flow electrophoresis : effect on the steadiness of the carrier flow. *Chemical Engineering Science*, 56 :3053–3064, 2001.

[2] S.G. Akl. Parallel sorting algorithms. *Academic Press*, 1985.

[3] G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, pages 483–485, New York, USA, 1967. ACM.

[4] N. Antonopoulos and L. Gillam. *Cloud Computing : Principles, Systems and Applications*. Computer Communications and Networks. Springer, Berlin, Heidelberg, 2010.

[5] J. Arnal, V. Migallón, and J. Penadés. Non-stationary parallel multi-splitting algorithms for almost linear systems. *Numerical Linear Algebra with Applications*, 6 :79–92, 1999.

[6] J. Bahi, R. Couturier, and L. Ziane Khodja. Parallel gmres implementation for solving sparse linear systems on gpu clusters. In *Proceedings of the 19th High Performance Computing Symposia*. Society for Computer Simulation International, 2011.

[7] J. Bahi, X. Fang, C. Guyeux, and L. Larger. Fpga design for pseudorandom number generator based on chaotic iteration used in information hiding application. *Applied Mathematics and Information Sciences*, 7 :2175–2188, 2013.

[8] J. Bahi, X. Fang, C. Guyeux, and Q. Wang. Evaluating quality of chaotic pseudo-random generators. application to information hiding. *International Journal On Advances in Security*, 4 :118–130, 2011.

[9] J. Bahi, X. Fang, C. Guyeux, and Q. Wang. Suitability of chaotic iterations schemes using xorshift for security applications. *Journal of Network and Computer Applications*, 37 :282–292, 2014.

[10] J. Bahi, E. Griepentrog, and J.C. Miellou. Parallel treatment of a class of differential-algebraic systems. *SIAM Journal of Numerical Analysis*, 23(5) :and–1996, 1996.

[11] J. Bahi and C. Guyeux. Hash functions using chaotic iterations. *Journal of Algorithms and Computational Technology*, 4 :167–181, 2010.

[12] J. M. Bahi, J. C. Miellou, and K. Rhofir. Asynchronous multisplitting methods for nonlinear fixed point problems. *Numerical Algorithms*, 15 :315–345, 1997.

[13] J.M. Bahi, S. Contassot-Vivier, and R. Couturier. *Parallel iterative algorithms : from sequential to grid computing.* Chapman & Hall/CRC, Boca Raton, 2007.

[14] Z. Z. Bai. The monotone convergence rate of the parallel nonlinear aor method. *Computers and Mathematics with Applications*, 31 :1–8, 1996.

[15] Z. Z. Bai. Asynchronous multisplitting aor methods for a class of systems of weakly nonlinear equations. *Appl. Math*, 98 :49–59, 1999.

[16] ZZ. Bai, V. Migallón, J. Penadés, and DB. Szyld. Block and asynchronous two-stage methods for mildly nonlinear systems. *Numerische Mathematik*, 82 :1–20, 1999.

[17] V. Barbu. *Non linear semi-groups and differential equations in Banach spaces.* Noordhoff international publishing, Grøningen, 1976.

[18] G.M. Baudet. Asynchronous iterative methods for multiprocessor. *J. Assoc. Comput. Mach.*, 2 :226–244, 1978.

[19] D. El Baz. A method of terminating asynchronous iterative algorithms on message passing systems. *Parallel Algorithms and Applications*, 9 :153–158, 1996.

[20] D. El Baz, A. Frommer, and P. Spitéri. Asynchronous iterations with flexible communication : contracting operators. *Journal of Computational and Applied Mathematics*, 176 :91–103, 2005.

[21] D. El Baz, D. Gazen, J.C. Miellou, and P. Spitéri. Mise en œuvre de méthodes itèratives asynchrones avec communication flexible. *Calculateur Parallèles*, 8(4) :393–410, 1996.

[22] D. El Baz and T.T. Nguyen. A self-adaptive communication protocol with application to high performance peer to peer distributed computing. In *Parallel, Distributed and Network-Based Processing (PDP), 2010 18th Euromicro International Conference on*, pages 327–333, 2010.

[23] D. El Baz, P. Spitéri, J.C. Miellou, and M. Jarraya. Mathematical study of perturbated asynchronous iterations designed for distributed termination. *International Mathematical Journal*, 1(5) :491–503, 2002.

[24] S. Benjelloun, P. Spitéri, and G. Authié. Parallel algorithms for solving the obstacle problem. *Computational Mechanics Publ., Springer-Verlag*, 2 :275–281, 1989.

[25] D. Bertsekas. Distributed asynchronous computation of fixed points. *Math. Programming*, 27 :107–120, 1983.

[26] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation.* Numerical Methods, Prentice Hall Englewood Cliffs N.J, 1989.

[27] R. Bolze, F. Cappello, E. Caron, M. Daydè, F. Desprez, E. Jeannot, Y. Jègou, S. Lanteri, J. Leduc, N. Melab, G. Mornet, R. Namyst, P. Primet, B. Quetier, O. Richard, E.G. Talbi, and I. Touche. Grid'5000 : A large scale and

highly reconfigurable experimental grid testbed. *International Journal of High Performance Computing Applications*, 20(4) :481–494, 2006.

[28] R. Buyya, J. Broberg, and A. M. Goscinski. *Cloud Computing : Principles and Paradigms*. Wiley, 2011.

[29] K. M. Chandy and L. Lamport. Distributed snapshots : determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 77(78) :63–75, 1985.

[30] M. Chau. *Algorithmes Parallèles Asynchrones et Applications en Calcul Scientifique*. Habilitation à diriger des recherches, Institut National Polytechnique de Toulouse, Institut de Recherche en Informatique de Toulouse, 2015.

[31] M. Chau, D. El Baz, R. Guivarch, and P. Spitéri. MPI implementation of parallel subdomain methods for linear and nonlinear convection-diffusion problems. *J. Parallel Distrib. Comput.*, 67(5) :581–591, 2007.

[32] M. Chau, R. Couturier, J. Bahi, and P. Spitéri. Parallel solution of the obstacle problem in grid environment. *International Journal of High Performance Computing Applications, SAGE Publications*, 25(4) :488–495, 2011.

[33] M. Chau, R. Couturier, J. Bahi, and P. Spitéri. Asynchronous grid computation for american options derivative. *Advances in Engineering Software, Science, 60-61*, pages 136–144, 2013.

[34] M. Chau, T. Garcia, and P. Spiteri. Asynchronous schwarz methods applied to constrained mechanical structures in grid environment. *Advances in Engineering Software, Elsevier*, 74(0) :1 – 15, 2014.

[35] M. Chau, T. Garcia, and P. Spitéri. Asynchronous grid computing for the simulation of the 3d electrophoresis coupled problem. *Advances in Engineering Software, Elsevier*, pages 60–61, 2013.

[36] M. Chau, A. Laouar, T. Garcia, and P. Spitéri. Grid solution of problem with unilateral constraints. *Numerical Algorithms, Springer Verlag, 879 - 908*, 75(4) :879–908, 2017.

[37] M. Chau, P. Spitéri, and H. C. Boisson. Parallel numerical simulation for the coupled problem of continuous flow electrophoresis. *International Journal for Numerical Methods in Fluids, Interscience Publications, 55*, pages 945–963, 2007.

[38] M. Chau, P. Spitéri, R. Guivarch, and H. C. Boisson. Parallel asynchronous iterations for the solution of a 3d continuous flow electrophoresis problem. *Computers and Fluids, , 37*, pages 1126–1137, 2008.

[39] D. Chazan and W. Miranker. Chaotic relaxation. *Linear Algebra Appl.*, 2 :199–222, 1969.

[40] R. Couturier. *Designing Scientific Applications on GPUs*. Numerical Analysis and Scientific Computing. Chapman & Hall/CRC, 2013.

[41] R. Couturier, P. Canalda, and F. Spies. Iterative algorithms on heterogeneous network computing : Parallel polynomial root extracting. In *Internatio-*

nal Conference on High-Performance Computing. , , Heidelberg, Berlin, 2002. Springer.

[42] R. Couturier, C. Denis, and F. Jézéquel. Gremlins : a large sparse linear solver for grid environment. *Parallel Comput*, 34(68), 2008.

[43] R. Couturier and L. Ziane Khodja. A scalable multisplitting algorithm to solve large sparse linear systems. *The Journal of Supercomputing*, 69(1) :200–224, 2014.

[44] E. W. Dijkstra and C. S. Scholten. Termination detection for diffusing computation. *Processing Letters*, 11 :1–4, 1980.

[45] Didier El Baz. *Contribution à l'algorithmique parallèle, Le concept d'asynchronisme : étude théorique, mise en œuvre et application.* Habilitation à diriger des recherches, INP DE TOULOUSE, 1998.

[46] Nahid Emad, S.-A. Shahzadeh-Fazeli, and Jack Dongarra. An asynchronous algorithm on the netsolve global computing system. *Future Generation Computer Systems*, 22(3) :279–290, 2006.

[47] D.J. Evans and W. Deren. An asynchronous parallel algorithm for solving a class of nonlinear simultaneous equations. *Parallel Computing*, 17 :165–180, 1991.

[48] D. Szyld F. Chaouqui, E. Chow. Asynchronous domain decomposition methods for nonlinear pdes. *ETNA*, 58 :22 – 42, 2023.

[49] F. Teng F. Magoules, J. Pan. *Cloud Computing data-intenbsive computing and scheduling.* Chaman and Hall/ CRC , Numerical Analysis and Scientific Computing, 2013.

[50] X. Fang, Q. Wang, C. Guyeux, and J. Bahi. Fpga acceleration of a pseudo-random number generator based on chaotic iterations. *Journal of Information Security and Applications*, 19 :78–87, 2014.

[51] FG-Cloud. Fg-cloud, 2023.

[52] M. J. Flynn. Very high-speed computing systems. In *Proceedings of the IEEE 54*, pages 1901 – 1909, 1966.

[53] A. Frommer and G. Mayer. Convergence of relaxed parallel multisplitting methods. *Linear Algebra and its Applications*, 119 :141–152, 1989.

[54] A. Frommer and G. Mayer. On the theory and practice of multisplitting methods in parallel computation. *Computing*, 74 :49–63, 1992.

[55] A. Frommer, H. Schwandt, and D. Szyld. Asynchronous weighted additive Schwarz methods. *Electronic Transactions on Numerical Analysis*, 5 :48–61, 1997.

[56] A. Frommer and P. Spitéri. On linear asynchronous iterations when the spectral radius of the modulus matrix is one. *Computing*, 15 :91–104, 2001.

[57] A. Frommer and D. Szyld. On asynchronous two-stage iterative methods. *Numer. Math.*, 69 :141–153, 1994.

[58] A. Frommer and D. Szyld. Weighted max norms, splittings and overlapping additive Schwarz iterations. *Numer. Math.*, 83 :259–278, 1999.

[59] A. Frommer and D. Szyld. On asynchronous iterations. *Journal of Computational and Applied Mathematics*, 123 :201–216, 2000.

[60] A. Frommer and D. B. Szyld. H -splittings and two-stage iterative methods. *Numerische Mathematik*, 63 :345–356, 1992.

[61] F. Magoules G. Gbikpi-Benissan. Asynchronous multiplicative course-space correction. *SIAM journal on scientific computing*, 44-3 :C237 – 259, 2022.

[62] F. Magoules G. Gbikpi-Benissan. Asynchronous multisplitting-based primal schur method. *Journal of Computational and Applied Mathematics*, 425 :115060, 2023.

[63] M.J. Gander. Optimized Schwarz methods. *SIAM Journal of Numerical Analysis*, pages 15–28, 2006.

[64] M.J. Gander, F. Magoulès, and F. Nataf. Optimized schwarz methods without overlap for the helmholtz equation. *SIAM Journal on Scientific Computing*, 24(1) :38–60, 2002.

[65] M. Garbey and D. Tromeur-Dervout. On some Aitken like acceleration of the Schwarz methods. *Int. J. for Numerical Methods in Fluids*, 40(12) :1493–1513, 2002.

[66] T. Garcia, M. Chau, T.T. Nguyen, D. El-Baz, and P. Spitéri. Asynchronous peer-to-peer distributed computing for financial applications. In *Proceedings of the 2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and PhD Forum*, IPDPSW '11, pages 1458–1466, Washington, DC, USA, 2011. IEEE Computer Society.

[67] T. Garcia, M. Chau, and P. Spitéri. Synchronous and asynchronous distributed computing for financial option pricing. In B. Murgante et al., editor, *Computational Science and Its Applications - ICCSA 2011*, volume 6783 of *Lecture Notes in Computer Science*, pages 664–679. Springer Berlin Heidelberg, 2011.

[68] T. Garcia, P. Spitéri, and G. Khenniche. Behavior of parallel two-stage method for the simulation of steel solidification in continuous casting. *Advances in Engineering Software, Elsevier*, 131 :116–142, 2019.

[69] T. Garcia, P. Spitéri, L. Ziane-Khodja, and R. Couturier. Solution of univalued and multivalued pseudo-linear problems using parallel asynchronous multisplitting methods combined with krylov methods. *Advances in Engineering Software, Elsevier*, 153, 2020.

[70] T. Garcia, P. Spitéri, L. Ziane-Khodja, and R. Couturier. Coupling parallel asynchronous multisplitting methods with krylov methods to solve pseudo-linear evolution 3d problems. *Journal of Computational Science, Elsevier*, 51, 2021.

[71] G. Gbikpi-Benissan, Q. Zou, and F. Magoules. Asynchronous iterative methods : programming models and parallel implementations. *Institute of Computer Science Press*, 2018.

[72] M. Gengler, S. Ubéda, and F. Desprez. *Initiation au Parallélisme : Concept, Architectures et Algorithmes.* Masson, 1996.

[73] K. Ghidouche, R. Couturier, and A. Sider. A parallel implementation of the durand-kerner algorithm for polynomial root-finding on gpu. *Advanced Networking Distributed Systems and Applications (INDS), International Conference on. IEEE*, 2014.

[74] K. Ghidouche, A. Sider, R. Couturier, and C. Guyeux. Efficient high degree polynomial root finding using gpu. *Journal of Computational Science*, 18 :46–56, 2017.

[75] L. Giraud and P. Spitéri. Résolution parallèle de problèmes aux limites non linéaires. *M2AN*, 25 :579–606, 1991.

[76] L. Giraud and P. Spitéri. Implementations of parallel solutions for nonlinear boundary value problems. In *Parallel Computing'91*, pages 203–211. D.J. Evans, G.R. Joubert, and H. Liddel, Elsevier Science B.V., North Holland, Amsterdam, 1992.

[77] R. Glowinski, J. L. Lions, and R. Tremolieres. *Analyse numérique des inéquations variationnelles*, volume 1-2. DUNOD, 1976.

[78] C. Glusa, E.G. Boman, E. Chow, S. Rajamanickan, and D. Szyld. Scalable asynchronous domain decomposition solvers. *SIAM Journal on Scientific Computing,*, 42-6 :C384 – 409, 2020.

[79] Grid'5000. Grid'5000, 2023.

[80] R. Guivarch and P. Spitéri. Implantation de mthodes de sous-domaines asynchrones avec pvm et mpi sur ibm-sp2. *Calculateurs Parallles, Editions Hermes Paris*, 10(4) :431–438, 1998.

[81] R. Guivarch, P. Spitéri, H. C. Boisson, and J. C. Miellou. Schwartz alternating parallel algorithm applied to incompressible flow computation in vorticity stream function formulation. *Parallel Algorithm and Application*, 11 :205–225, 1997.

[82] C. Guyeux and J. Bahi. A topological study of chaotic iterations. application to hash functions. *Computational Intelligence for Privacy and Security*, 394 :51–73, 2012.

[83] Christophe Guyeux. Convergence versus divergence behaviors of asynchronous iterations, and their applications in concrete situations. *Mathematical and Computational Applications*, 25 :69, 10 2020.

[84] R. W. Hockney and C. R. Jesshope. *Parallel Computers : Architecture, Programming and Algorithms.* Adam Hilger Ltd, 1981.

[85] K.H. Hoffman and J. Zou. Parallel efficiency of domain decomposition methods. *Parallel Computing*, 19 :1375–1391, 1993.

[86] Idris. Mpi, 2023.

[87] F. Jézéquel, R. Couturier, and C. Denis. Solving large sparse linear systems in a grid environment : the gremlins code versus the petsc library. *Journal of Supercomputing.*, 59(3) :1517–1532, 2012.

[88] F. Jézéquel, R. Couturier, and C. Denis. Solving large sparse linear systems in a grid environment : the GREMLINS code versus the PETSc library. *The Journal of Supercomputing*, 59 :1517 – 1532, 2012.

[89] M. T. Jones and DB. Szyld. Two-stage multisplitting methods with overlapping blocks. *Numerical Linear Algebra with Applications*, 3 :113–124, 1996.

[90] G. Khenniche. *Contribution à l'analyse des systèmes Dynamiques distribués : Application à l'optimisation de la thermique en Aciérie*. Thèse, Université Badji Mokhtar Annaba, Annaba, 2017.

[91] P. L. Lions. On the schwarz alternating method i. In R. Glowinski, G. H. Golub, G. A. Meurant, and J. Periaux, editors, *First International Symposium on Domain Decomposition Methods for Partial Differential Equations*, pages 1–42, Philadelphia, 1988. SIAM.

[92] P. L. Lions. On the schwarz alternating method ii. In T. Chan, R. Glowinski, J. Periaux, and O. Widlund, editors, *Domain Decomposition Methods*, pages 47–70, Philadelphia, 1989. SIAM.

[93] P. L. Lions. On the schwarz alternating method iii : A variant for nonoverlapping subdomains. In T. Chan, R. Glowinski, J. Periaux, and O. Widlund, editors, *Third International Symposium on Domain Decomposition Methods for Partial Differential Equations*, volume 6, pages 202–223, Philadelphia, 1990. SIAM.

[94] F. Magoules. From synchronous to asynchronous substructuring methods. *Advances in Parallel, Distributed, Grid and Cloud Computing, P. Iványi, B.H.V. Topping and G. Várady (Editors), Saxe-Coburg Publication*, 1 :203–215, 2017.

[95] F. Magoules. Asynchronous iterative sub-structuring methods. *Mathematics and Computers in Simulation*, 145 :34–49, 2018.

[96] F. Magoules and G. Gbikpi-Benissan. Jack : An asynchronous communication kernel library for iterative algorithms. *The Journal of Supercomputing*, 73(8) :3468–3487, 2017.

[97] F. Magoules and G. Gbikpi-Benissan. Asynchronous parareal time discretization for partial differential equations. *SIAM J. Sci. Comput.*, 40(6) :704–725, 2018.

[98] F. Magoules and G. Gbikpi-Benissan. Distributed convergence detection based on global residual error under asynchronous iterations. *Parallel and Distributed Systems, IEEE Transactions on*, 29(4) :819–829, 2018.

[99] F. Magoules and G. Gbikpi-Benissan. Jack2 : An mpi-based communication library with non-blocking synchronization for asynchronous iterations. *Advances in Engineering Software*, 11 :116–133, 2018.

[100] F. Magoules, G. Gbikpi-Benissan, and Q. Zou. Asynchronous iterations of parareal algorithm for option pricing models. *Mathematics*, 6(4) :45, 2018.

[101] F. Magoules, G. Gbikpi-Benissan, and Q. Zou. Asynchronous iterations of parareal algorithm for option pricing models. *Mathematics*, 6(4) :45, 2018.

[102] F. Magoulès, F.-X. Roux, and G. Houzeaux. *Parallel Scientific Computing*. Wiley, 2015.

[103] F. Magoules, D. B. Szyld, and C. Venet. Asynchronous optimized schwarz methods with and without overlap. *Numerische Mathematik*, 137(1) :199–227, 2017.

[104] F. Magoules, D. B. Szyld, and C. Venet. Asynchronous optimized schwarz methods with and without overlap. *Numerische Mathematik*, 137(1) :199–227, 2017.

[105] J-C. Miellou, J. Bahi, and M. Laaraj. A survey on asynchronous iterations. In *Parallel and Distributed Computing for Computational Mechanics using High Performance Computing 1999, Weimar, B. H. V. Topping, Editor, Saxe-Coburg Publications, Stirling, Scotland*, pages 95–115. Proceedings of the Euroconference EURO-CM-PAR99, 2002.

[106] J.-C. Miellou, D. El Baz, and P. Spitéri. A new class of asynchronous iterative algorithms with order interval. *Mathematics of Computation*, 67-221 :237–255, 1998.

[107] J. C. Miellou, P. Cortey-Dumont, and M. Boulbrachêne. Perturbation of fixed point iterative methods. *Advances in Parallel Computing, I*, pages 81–122, 1990.

[108] J.-C. Miellou, M. Laaraj, and M. Gander. Overlapping multi-subdomain asynchronous fixed point methods for elliptic boundary value problems. *Proceedings of the ninth Colloquium on differential equation, International Science Publishers Utrecht, The Netherlands,*, pages 261–266, 1999.

[109] J. C. Miellou, P. Spiteri, and D. El Baz. Stopping criteria, forward and backward errors for perturbed asynchronous linear fixed point methods in finite precision. *IMA Journal of Numerical Analysis*, 25 :429–442, 2005.

[110] J.-C. Miellou and P. Spitéri. Un critère de convergence pour des méthodes générales de point fixe. *M2AN*, 19 :645–669, 1985.

[111] J. C. Miellou and P. Spitéri. Stopping criteria for parallel asynchronous iterations for fixed point methods. In grid distributed and B. H. V. Topping P. Ivanyi cloud computing for Engineering, B. H. V. Topping cloud computing for Engineering, editors, *Developments in parallel*, pages 277–308. Saxe-Coburg Publications, 2013.

[112] J. C. Miellou, P. Spitéri, and D. El Baz. A new stopping criterion for linear perturbed asynchronous iterations. *Journal of Computational and Applied Mathematics*, pages 471–483, 2008.

[113] J.C. Miellou. Algorithmes de relaxation chaotique à retards. *RAIRO*, 1 :55–82, 1975.

[114] J.C. Miellou. Variantes synchrones et asynchrones de la méthode alternée de schwarz. Technical Report E.R.A. de mathématiques n° 70654, Université de Besançon, 1982.

[115] J.C. Miellou, P. Cortey-Dumont, and M. Boulbrachêne. Perturbation of fixed-point iterative methods. *Advances in Parallel Computing, AI Press Inc.*, 1 :81–122, 1990.

[116] T. T. Nguyen, D. El Baz, P. Spitéri, G. Jourjon, and M. Chau. High performance peer-to-peer distributed computing with application to obstacle problem. In *Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium*, pages 1–8, 2010.

[117] T.T. Nguyen. *Un environnement pour le calcul intensif pair à pair.* Thèse de doctorat, INP DE TOULOUSE, 2011.

[118] D. P. O'Leary and R. E. White. Multi-splittings of matrices and parallel solution of linear systems. *SIAM :Journal on Algebraic Discrete Methods*, 6 :630–640, 1985.

[119] V. Partimbene, T. Garcia, P. Marthon, L. Ratsifandrihana, and P. Spitéri. Asynchronous multi-splitting method for linear and pseudo-linear problems. *Advances in Engineering Software, Elsevier*, 133 :76–95, 2019.

[120] IDRIS : Dimitri Lecas Rémi Lacroix Serge Van Criekingen Myriam Peyrounette. Mpi, 2023. [En ligne ; Page disponible le 18-juillet-2023].

[121] M.A. Rahhali, T. Garcia, and P. Spitéri. Behaviour of asynchronous parallel iterative algorithms on cloud computing type architectures. *Proceedings of the Eleventh International Conference on Engineering Computational Technology in B.H.V. Topping, P. Iványi, (Editors), Civil-Comp Press, Edinburgh, UK*, CCC2 8.2, 2022.

[122] M.A. Rahhali, T. Garcia, and P. Spitéri. Performances analysis of parallel asynchronous and synchronous algorithms on cloud architecture for pde. *Advances in Engineering Software*, 186 :103550, 2023.

[123] C. E. Ramamonjisoa, L. Ziane Khodja, D. Laiymani, A. Giersch, and R. Couturier. Simulation of asynchronous iterative algorithms using simgrid. *IEEE Intl Conf on High Performance Computing and Communications, IEEE 6th Intl Symp on Cyberspace Safety and Security, IEEE 11th Intl Conf on Embedded Software and Syst (HPCC,CSS,ICESS*, pages 890–895, 2014.

[124] F. Robert. Recherche d'une m-matrice parmi les minorantes d'un opèrateur linèaire, numer. *Math.*, 9 :189–199, 1966.

[125] F. Robert. Blocs h-matrices et convergence des mthodes itratives classiques par blocs. *Linear Algebra and its Applications*, 2 :223–265, 1969.

[126] F. Robert, M. Charnay, and F. Musy. Itérations chaotiques série-parallèle pour des équations non-linéaires de point fixe. *Aplikace matematiky*, 20-1 , 1-38, 1975.

[127] Y. Saad. *Iterative Methods for Sparse Linear Systems*. Other Titles in Applied Mathematics. SIAM, 2003.

[128] Y. Saad. Iterative methods for linear systems of equations : a brief historical journey. *75 Years of Mathematics of Computation, Contemporary Mathematics, Edited by Susanne C. Brenner, Igor E. Shparlinski, Chi-Wang Shu, Daniel Szyld*, 754 :197 – 214, 2020.

[129] S. A. Savari and D. P. Bertsekas. Finite termination of asynchronous iterative algorithms. *Parallel Computing*, 22(1) :39–56, 1996.

[130] J. Schröder. Nichtlineare majoranten beim verfahren der schrittweisen näherung. *Arch. Math. (Bused)*, 7 :471–784, 1956.

[131] J. Schroeder. Computing error bounds in solving linear systems. Technical Report 242, University of Wisconsin, M.R.C, 1961.

[132] P. Spiteri, D. El Baz, J. C. Miellou, and M. Jarraya. Mathematical study of perturbed asynchronous iterations designed for distributed termination. *International Mathematical Journal*, 1(5) :491–503, 2002.

[133] P. Spitéri. Simulation d'exécution parallèles pour la résolution d'inéquations variationnelles stationaires. *Revue E.D.F. Informatique et Mathèmatique Appliquées, Série C*, 1 :149–158, 1983.

[134] P. Spitéri. *Contribution à l'étude de grands systèmes non linéaires*. PhD thesis, Université de Franche Comté, 1984.

[135] P. Spitéri. Parallel asynchronous algorithms for solving boundary value problems. In *Parallel Algorithms and Architectures*, pages 73–84. M. Cosnard et al, North-Holland, 1986.

[136] P. Spitéri. A new characterization of m-matrices and h-matrices. In *Bit Numerical Mathematics*, pages 1019–1032. Springer, 43, 2003.

[137] P. Spitéri. Finite precision computation for linear fixed point methods of parallel asynchronous iterations. In distributed and B. H. V. Topping P. Ivanyi cloud computing in engineering, B. H. V. Topping cloud computing in engineering, editors, *Techniques for parallel*, chapter 8, pages 163–196. Saxe-Coburg Publications, Stirlingshire, UK, 2015.

[138] P. Spitéri. Synthetic presentation of iterative asynchronous parallel algorithms. In P. Ivnyi and B. H. V. Topping and, editors, *Proceedings of the Sixth International Conference on Parallel, Distributed, GPU and Cloud Computing for Engineering*. Civil-Comp Press, Stirlingshire, UK, Paper 14, 2019.

[139] P. Spitéri. Parallel asynchronous algorithms : a survey. *Advances in Engineering Software, Elsevier, 149*, 2020.

[140] P. Spitéri and H.C. Boisson. Subdomain predictor-corrector algorithms for solving the incompressible Navier-Stokes equation. In H.G. Kaper and M. Garbey,

editors, *Asymptotic and numerical methods for partial differential equations with critical parameters*, volume 384 of *NATO ASI series*, pages 335–347. Kluwer Academic Publishers, 1993.

[141] P. Spitéri and M. Chau. Parallel asynchronous Richardson method for the solution of the obstacle problem. In J.N Almhana and V.C. Bhavsars, editors, *HPCS 2002*, High Performance Computing Systems and Applications, pages 133–138, Moncton, 2002. IEEE.

[142] P. Spitéri, J.-C. Miellou, and D. El Baz. Asynchronous schwarz alternating methods with flexible communication for the obstacle problem. *Réseaux et systèmes répartis*, 13 (1) :47–66, 2001.

[143] P. Spitéri, J. C. Miellou, and D. El Baz. Parallel asynchronous schwarz and multisplitting methods for a non linear diffusion problem. *Numerical Algorithms*, 33 :461–474, 2003.

[144] P. Spitéri and J.C. Miellou. Algorithmes parallèles asynchrone i : modélisation et analyse. *Techniques de l'ingénieur, AF 1385, Sciences fondamentales*, 2021.

[145] P. Spitéri and J.C. Miellou. Algorithmes parallèles asynchrone ii : implémentation. *Techniques de l'ingénieur, AF 1385, Sciences fondamentales*, 2021.

[146] P. Spitéri and J.C. Miellou. Algorithmes parallèles asynchrone iii : applications, performances. *Techniques de l'ingénieur, AF 1385, Sciences fondamentales*, 2021.

[147] P. Spitéri, J.C. Miellou, and D. El Baz. Perturbation of parallel asynchronous linear iterations by floating point errors. *Electron. Trans. Numer. Anal*, 13 :38–55, 2002.

[148] D. Szyld. Different models of parallel asynchronous iterations with overlapping block. *Computational and Applied Mathematics*, 17 :101–115, 1998.

[149] M.N. El Tarazi. *Contraction et ordre partiel pour l'étude d'algorithmes synchrones et asynchrones en analyse numérique.* Thèse, Université de Besançon, 1981.

[150] M.N. El Tarazi. Some convergence results for asynchronous algorithms. *Num. Math.*, 39 :325–340, 1982.

[151] C. Tauber, P. Spitéri, and H. Batatia. Iterative methods for anisotropic diffusion of speckled medical images. *Applied Numerical Mathematics*, 60 :1115–1030, 2010.

[152] Open MPI Teams. Open mpi : Open source high performance computing, 2020. [En ligne ; Page disponible le 26-août-2020].

[153] D. R. Wang, Z. Z. Bai, and D. J. Evans. On the monotone convergence of multisplitting method for a class of system of weakly nonlinear equations. *Intern. J. Computer Math*, 60 :229–242, 1996.

[154] Q. Wang, S. Yu, C. Guyeux, J. Bahi, and X. Fang. Theoretical design and circuit implementation of integer domain chaotic systems. *International Journal of Bifurcation and Chaos*, 24 :118–128, 2014.

[155] R. E. White. Parallel algorithms for nonlinear problems. *SIAM J. Alg. Discrete Meth.*, 7 :137–149, 1986.

[156] Wikipédia. Gnu compiler collection — wikipédia, l'encyclopédie libre, 2020. [En ligne ; Page disponible le 19-août-2020].

[157] Wikipédia. Message passing interface — wikipédia, l'encyclopédie libre, 2020. [En ligne ; Page disponible le 18-juillet-2023].

[158] J. Wilkinson. Rounding error in algebraic processes. *Notes on Applied sciences, Prentice-Hall series in automatic computation*, 32, 1963.

[159] P. Wilmott, J. Dewyne, and S. Howison. *Option Pricing-Mathematical Models and Computation*. Oxford financial press, 1993.

[160] L. Ziane Khodja, M. Chau, R. Couturier, J. Bahi, and P. Spitéri. Parallel solution of american option derivatives on GPU clusters. *Computers & Mathematics with Applications*, 65 :1830 – 1848, 2013.