# Amélioration des performances des logiciels d'éléments finis par l'utilisation de méthodes d'agrégation et d'architectures parallèles

Gilbert Pion

# T H E S E

présentée à

L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Pour obtenir

LE TITRE DE DOCTEUR-INGENIEUR

par

MONSIEUR GILBERT PION

* * *

AMELIORATION DES PERFORMANCES DES LOGICIELS D'ELEMENTS FINIS
PAR UTILISATION DE METHODES D'AGREGATION
ET D'ARCHITECTURES PARALLELES

Soutenue le 19 Septembre 1984 devant la Commission d'Examen

## J U R Y

| | | |
|---|---|---|
| Monsieur | SABONNADIERE J.C. | Président |
| Messieurs | AMIET M. | Examinateurs |
| | ROUCAIROL G. | |
| | COULOMB J.L. | |
| Madame | BECKER M. | |

# INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

Président: Daniel BLOCH

Année universitaire 1983-1984

Vice-Présidents: René CARRE
Hervé CHERADAME
Jean-Pierre LONGEQUEUE

## Professeur des Universités

| | | | | | |
|---|---|---|---|---|---|
| ANCEAU | François | E.N.S.I.M.A.G | JOUBERT | Jean-Claude | E.N.S.I.E.G |
| BARIBAUD | Michel | E.N.S.E.R.G | JOURDAIN | Geneviève | E.N.S.I.E.G |
| BARRAUD | Alain | E.N.S.I.E.G | LACOUME | Jean-Louis | E.N.S.I.E.G |
| BAUDELET | Bernard | E.N.S.I.E.G | LATOMBE | Jean-Claude | E.N.S.I.M.A.G |
| BESSON | Jean | E.N.S.E.E.G | LESIEUR | Marcel | E.N.S.H.G |
| BLIMAN | Samuel | E.N.S.E.R.G | LESPINARD | Georges | E.N.S.H.G |
| BLOCH | Daniel | E.N.S.I.E.G | LONGEQUEUE | Jean-Pierre | E.N.S.I.E.G |
| BOIS | Philippe | E.N.S.H.G | LOUCHET | François | E.N.S.E.E.G |
| BONNETAIN | Lucien | E.N.S.E.E.G | MASSELOT | Christian | E.N.S.I.E.G |
| BONNIER | Etienne | E.N.S.E.E.G | MAZARE | Guy | E.N.S.I.M.A.G |
| BOUVARD | Maurice | E.N.S.H.G | MOREAU | René | E.N.S.H.G |
| BRISSONNEAU | Pierre | E.N.S.I.E.G | MORET | Roger | E.N.S.I.E.G |
| BUYLE BODIN | Maurice | E.N.S.E.R.G | MOSSIERE | Jacques | E.N.S.I.M.A.G |
| CAVAIGNAC | Jean-François | E.N.S.I.E.G | PARIAUD | Jean-Charles | E.N.S.E.E.G |
| CHARTIER | Germain | E.N.S.I.E.G | PAUTHENET | René | E.N.S.I.E.G |
| CHENEVIER | Pierre | E.N.S.E.R.G | PERRET | René | E.N.S.I.E.G |
| CHERADAME | Hervé | U.E.R.M.C.P.P | PERRET | Robert | E.N.S.I.E.G |
| CHERUY | Arlette | E.N.S.I.E.G | PIAU | Jean-Michel | E.N.S.H.G |
| CHIAVERINA | Jean | U.E.R.M.C.P.P | POLOUJADOFF | Michel | E.N.S.I.E.G |
| COHEN | Joseph | E.N.S.E.R.G | POUPOT | Christian | E.N.S.E.R.G |
| COUMES | André | E.N.S.E.R.G | RAMEAU | Jean-Jacques | E.N.S.E.E.G |
| DURAND | Francis | E.N.S.E.E.G | RENAUD | Maurice | U.E.R.M.C.P.P |
| DURAND | Jean-louis | E.N.S.I.E.G | ROBERT | André | U.E.R.M.C.P.P |
| FELICI | Noël | E.N.S.I.E.G | ROBERT | François | E.N.S.I.M.A.G |
| FONLUPT | Jean | E.N.S.I.M.A.G | SABONNADIERE | Jean-Claude | E.N.S.I.E.G |
| FOULARD | Claude | E.N.S.I.E.G | SAUCIER | Gabrielle | E.N.S.I.M.A.G |
| GANDINI | Alessandro | U.E.R.M.C.P.P | SCHLENKER | Claire | E.N.S.I.E.G |
| GAUBERT | Claude | E.N.S.I.E.G | SCHLENKER | Michel | E.N.S.I.E.G |
| GENTIL | Pierre | E.N.S.E.R.G | SERMET | Pierre | E.N.S.E.R.G |
| GUERIN | Bernard | E.N.S.E.R.G | SILVY | Jacques | U.E.R.M.C.P.P |
| GUYOT | Pierre | E.N.S.E.E.G | SOHM | Jean-Claude | E.N.S.E.E.G |
| IVANES | Marcel | E.N.S.I.E.G | SOUQUET | Jean-Louis | E.N.S.E.E.G |
| JALINIER | Jean-Michel | E.N.S.I.E.G | VEILLON | Gérard | E.N.S.I.M.A.G |
| JAUSSAUD | Pierre | E.N.S.I.E.G | ZADWORNY | François | E.N.S.E.R.G |

## Professeurs Associés

| | | | | | |
|---|---|---|---|---|---|
| BLACKWELDER | Ronald | E.N.S.H.G | PURDY | Gary | E.N.S.E.E.G |
| HAYASHI | Hirashi | E.N.S.I.E.G | | | |

## Professeurs Université des Sciences Sociales (Grenoble II)

| | | | | | |
|---|---|---|---|---|---|
| BOLLIET | Louis | | CHATELIN | Françoise | |

## Chercheurs du C.N.R.S

| | | | | | |
|---|---|---|---|---|---|
| FRUCHART | Robert | Directeur de recherche | GUELIN | Pierre | Maître de recherche |
| JORRAND | Philippe | Directeur de recherche | HOPFINGER | Emil | Maître de recherche |
| VACHAUD | Georges | Directeur de recherche | JOUD | Jean-Charles | Maître de recherche |
| | | | KAMARINOS | Georges | Maître de recherche |
| ALLIBERT | Michel | Maître de recherche | KLEITZ | Michel | Maître de recherche |
| ANSARA | Ibrahim | Maître de recherche | LANDAU | Ioan-Dore | Maître de recherche |
| ARMAND | Michel | Maître de recherche | LASJAUNIAS | Jean-Claude | Maître de recherche |
| BINDER | Gilbert | Maître de recherche | MERMET | Jean | Maître de recherche |
| BORNARD | Guy | Maître de recherche | MUNIER | Jacques | Maître de recherche |
| CARRE | René | Maître de recherche | PIAU | Monique | Maître de recherche |
| DAVID | René | Maître de recherche | PORTESEIL | Jean-Louis | Maître de recherche |
| DEPORTES | Jacques | Maître de recherche | THOLENCE | Jean-Louis | Maître de recherche |
| DRIOLE | Jean | Maître de recherche | VERDILLON | André | Maître de recherche |
| GIGNOUX | Damien | Maître de recherche | SUERY | Michel | Maître de recherche |
| GIVORD | Dominique | Maître de recherche | | | |

## Personnalités habilitées à diriger des travaux de recherche
(Décision du Conseil Scientifique)

### E.N.S.E.E.G.

| | | | | | |
|---|---|---|---|---|---|
| ALLIBERT | Colette | DIARD | Jean Paul | NGUYEN TRUONG | Bernadette |
| BERNARD | Claude | EUSTATHOPOULOS | Nicolas | RAVAINE | Denis |
| BONNET | Roland | FOSTER | Panayotis | SAINFORT | (CENG) |
| CAILLET | Marcel | GALERIE | Alain | SARRAZIN | Pierre |
| CHATILLON | Catherine | HAMMOU | Abdelkader | SIMON | Jean Paul |
| CHATILLON | Christian | MALMEJAC | Yves (CENG) | TOUZAIN | Philippe |
| COULON | Michel | MARTIN GARIN | Régina | URBAIN | Georges (Laboratoire des ultra-réfractaires ODEILLO). |

### E.N.S.É.R.G.

| | | | | | |
|---|---|---|---|---|---|
| BARIBAUD | Michel | CHEHIKIAN | Alain | HERAULT | Jeanny |
| BOREL | Joseph | DOLMAZON | Jean Marc | MONLLOR | Christian |
| CHOVET | Alain | | | | |

### E.N.S.I.E.G.

| | | | | | |
|---|---|---|---|---|---|
| BORNARD | Guy | KOFMAN | Walter | MAZUER | Jean |
| DESCHIZEAUX | Pierre | LEJEUNE | Gérard | PERARD | Jacques |
| GLANGEAUD | François | | | REINISCH | Raymond |

### E.N.S.H.G.

| | | | | | |
|---|---|---|---|---|---|
| ALEMANY | Antoine | MICHEL | Jean Marie | ROWE | Alain |
| BOIS | Daniel | OBLED | Charles | VAUCLIN | Michel |
| DARVE | Félix | | | WACK | Bernard |

### E.N.S.I.M.A.G.

| | | | | | |
|---|---|---|---|---|---|
| BERT | Didier | COURTOIS | Bernard | FONLUPT | Jean |
| CALMET | Jacques | DELLA DORA | Jean | SIFAKIS | Joseph |
| COURTIN | Jacques | | | | |

### U.E.R.M.C.P.P.

| | |
|---|---|
| CHARUEL | Robert |

### C.E.N.G.

| | | | | | |
|---|---|---|---|---|---|
| CADET | Jean | JOUVE | Hubert (LETI) | PERROUD | Paul |
| COEURE | Philippe (LETI) | NICOLAU | Yvan (LETI) | PEUZIN | Jean Claude (LETI) |
| DELHAYE | Jean Marc (STT) | NIFENECKER | Hervé | TAIEB | Maurice |
| DUPUY | Michel (LETI) | | | VINCENDON | Marc |

### Laboratoires extérieurs :

### C.N.E.T.

| | | | | | |
|---|---|---|---|---|---|
| DEMOULIN | Eric | GERBER | Roland | MERCKEL | Gérard |
| DEVINE | R.A.B. | | | PAULEAU | Yves |

### I.N.S.A. Lyon

| | |
|---|---|
| GAUBERT | C. |

**********

# ECOLE NATIONALE SUPERIEURE DES MINES DE SAINT-ETIENNE

Directeur : Monsieur M. MERMET
Directeur des Etudes et de la formation : Monsieur J. LEVASSEUR
Directeur des recherches : Monsieur J. LEVY
Secrétaire Général : Mademoiselle M. CLERGUE

## Professeurs de 1ère Catégorie

| | | |
|---|---|---|
| COINDE | Alexandre | Gestion |
| GOUX | Claude | Métallurgie |
| LEVY | Jacques | Métallurgie |
| LOWYS | Jean-Pierre | Physique |
| MATHON | Albert | Gestion |
| RIEU | Jean | Mécanique - Résistance des matériaux |
| SOUSTELLE | Michel | Chimie |
| FORMERY | Philippe | Mathématiques Appliquées |

## Professeurs de 2ème catégorie

| | | |
|---|---|---|
| HABIB | Michel | Informatique |
| PERRIN | Michel | Géologie |
| VERCHERY | Georges | Matériaux |
| TOUCHARD | Bernard | Physique Industrielle |

## Directeur de recherche

| | | |
|---|---|---|
| LESBATS | Pierre | Métallurgie |

## Maîtres de recherche

| | | |
|---|---|---|
| BISCONDI | Michel | Métallurgie |
| DAVOINE | Philippe | Géologie |
| FOURDEUX | Angeline | Métallurgie |
| KOBYLANSKI | André | Métallurgie |
| LALAUZE | René | Chimie |
| LANCELOT | Francis | Chimie |
| LE COZE | Jean | Métallurgie |
| THEVENOT | François | Chimie |
| TRAN MINH | Canh | Chimie |

## Personnalités habilitées à diriger des travaux de recherche

| | | |
|---|---|---|
| DRIVER | Julian | Métallurgie |
| GUILHOT | Bernard | Chimie |
| THOMAS | Gérard | Chimie |

## Professeur à l'UER de Sciences de Saint-Etienne

| | | |
|---|---|---|
| VERGNAUD | Jean-Maurice | Chimie des Matériaux & chimie industrielle |

**********

Ce travail a été réalisé au sein du Laboratoire d'Electrotechnique de Grenoble.

J'adresse mes sincères remerciements à :

Monsieur le Professeur J.C. SABONNADIERE qui m'a accueilli dans son équipe de recherche et m'a fait l'honneur de présider le Jury de cette thèse,

Monsieur AMIET, Ingénieur DRET,

Monsieur ROUCAIROL, Directeur du Laboratoire de Recherche en Informatique (Orsay - Paris XI)
pour l'intérêt qu'ils ont manifesté à ce travail en honorant de leur présence le Jury,

Madame M. BECKER, Chargée de Recherche au C.N.R.S., a proposé ce sujet intéressant. Elle a dirigé ce travail par ses conseils précieux, qu'elle soit assurée de ma profonde reconnaissance,

Je remercie Monsieur J.L. COULOMB, Maître-Assistant à l'E.N.S.I.E.G., qui a beaucoup collaboré à la réalisation de ce travail et a accepté de participer au Jury,

Madame E. CALLEGHER, Ingénieur C.N.R.S., m'a apporté une aide déterminante pour la réalisation des logiciels. Que son travail et sa gentillesse soient ici remerciés.

Monsieur FOURNIER, lorsqu'il était ingénieur au C.I.C.G., a également participé à la réalisation des logiciels, qu'il en soit remercié,

Je remercie Madame CHATELIN, Professeur d'Analyse Numérique, Ingénieur I.B.M., qui nous a beaucoup aidé par ses conseils sur les méthodes numériques,

Je remercie Monsieur François ROBERT, Professeur à l'U.S.M.G., pour d'intéressantes discussions sur les algorithmes parallèles,

Je tiens à exprimer ma gratitude à Monsieur Ph. MASSE, Maître-Assistant à l'ENSIEG pour le temps qu'il nous a consacré au début de ce travail,

Je remercie tous mes camarades du Laboratoire pour leur aide précieuse, en particulier Messieurs BLEUVIN, DU TERRAIL, FELIACHI, KOUYOUMDJIAN, MEUNIER, MOREL et SHEN.

Je tiens à remercier tout spécialement Madame BRICHET pour le soin avec lequel elle a assuré la frappe de ce mémoire et Monsieur FERRI qui en a mené à bien la reproduction.

Je remercie enfin le Ministère de l'Industrie et de la Recherche dont le soutien financier a rendu ce travail possible.

# AMELIORATION DES PERFORMANCES DES LOGICIELS D'ELEMENTS FINIS PAR UTILISATION DE METHODES D'AGREGATION ET D'ARCHITECTURES PARALLELES

# PREMIERE PARTIE

* * * * *

# CHAPITRE

# 1

## CHAPITRE I . GENERALITES SUR LES ORDINATEURS PARALLELES

INTRODUCTION

A . CLASSIFICATION DES ARCHITECTURES PARALLELES

I . ASPECTS DU PARALLELISME DANS LES ARCHITECTURES PARALLELES

1. Variétés de parallèlisme
2. Définitions

II . CLASSIFICATION DES STRUCTURES D'ORDINATEUR

B . REALISATION DE CES ARCHITECTURES

I . MULTIPROCESSEURS ET OPERATIONS SIMULTANEES

1. Définitions
2. Distribution des processeurs

II . STRUCTURE DES INTERCONNEXIONS POUR LES SYSTEMES PARALLELES

1. Transmission des données
2. Les interconnexions

C . MATERIELS ET LOGICIELS PARALLELES

I . QUELQUES EXEMPLES D'ORDINATEURS PARALLELES

II . LES LANGAGES DES SYSTEMES PARALLELES

# CHAPITRE I . GENERALITES SUR LES ARCHITECTURES PARALLELES

## INTRODUCTION

Les progrès considérables de la technologie des dernières années ont été le facteur principal de développement des différents concepts d'ordinateurs et de systèmes. C'est la technologie disponible à un instant donné qui limite le taux maximum avec lequel on peut traiter les données.
Initialement, les progrès de la puissance de calcul (environ multipliée par dix tous les cinq ans) étaient dûs à la technologie des circuits utilisés mais pas à l'organisation et au développement des logiciels systèmes.

La technologie de base a évolué, partant des relais, puis les transistors, les mémoires magnétiques et circuits intégrés pour arriver aux actuels circuits VLSI (Very Large Scale Integrated circuits). Parallèlement, des produits de la recherche tels les mémoires magnétiques à bulle ou l'effet Josephson commencèrent à être pris en considération comme des matériels tout à fait utilisables et commercialisables.

Grâce à ces éléments très puissants, le concepteur de nouveaux systèmes informatiques doit davantage porter sa réflexion sur l'analyse de système que sur la technologie ou les circuits. Malgré les vitesses de traitement que peut fournir une technologie, il y a et il y aura toujours un besoin d'augmentation de la puissance de calcul. Si le concepteur de nouvelles machines doit connaître les technologies de base, son but principal doit être orienté vers de nouveaux concepts évolutifs des systèmes.

## A .- CLASSIFICATION DES ARCHITECTURES PARALLELES

### I . ASPECTS DU PARALLELISME DANS LES ARCHITECTURES D'ORDINATEUR

#### 1. Variétés de parallèlisme

Les raisons de l'arrivée du parallèlisme dans les systèmes d'ordinateur sont diverses. La raison principale a toujours été le désir de diminuer le temps d'exécution d'un programme et d'augmenter l'éventail des calculs possibles. D'autres objectifs recherchés peuvent être la sécurité, la flexibilité, l'approche du temps réel.

Tout spécialement, les systèmes temps réels complexes (commes les problèmes de simulation ou le traitement d'images) nécessitent des ordinateurs de très hautes performances. Il faut donc rechercher la meilleure façon de réaliser des activités simultanées, c'est-à-dire parallèles.

Ces activités peuvent exister à différents niveaux :

- opérations multiples (activités parallèles sur les opérations)
- traitements multiples (activités parallèles sur les instructions)
- tâches ou processus multiples
- travaux multiples ("multiple jobs")

Un grand nombre de structures ont été proposées ou construites. Voici le nom de quelques unes :

- single-processor-system (système monoprocesseur)
- array-processor (processeur vectoriel et matrice de processeurs)
- data-flow-processor (processeur à flot de données)
- pipeline-processor (processeur pipeline)
- multi-processor-system (système multiprocesseur)
- multiple-special-purpose fonctional units (processeur spécialisé)
- computer network (système multi-ordinateur ou réseau d'ordinateurs).

Si l'on veut classer ces différents types de systèmes suivant leur vitesse, on s'aperçoit que les processeurs spécialisés sont de loin les plus rapides mais aussi les plus chers et les moins généraux. Ces systèmes sont construits pour une efficacité maximale dans leur champ d'application propre. Tous les autres types de machine décrits plus haut ont, par rapport au processeur spécialisé, une vitesse et une efficacité décroissantes mais plus de généralité.

L'ensemble des systèmes peut être séparé en deux classes principales :

- les systèmes monoprocesseurs,
- les systèmes multiprocesseurs.

## 2. Définitions

Même dans un système monoprocesseur les composants tels que l'unité centrale, les canaux d'Entrée/Sortie et les périphériques peuvent travailler simultanément.

L'Institut National Americain de Standardisation a défini un système multiprocesseur de la façon suivante : "un ordinateur employant deux ou plus unités de traitement avec contrôle intégré". Cependant, cette définition est apparue insuffisante.

Aujourd'hui, un système est considéré comme un système multiprocesseur s'il répond aux caractéristiques suivantes :

- le nombre de processeurs est plus grand que 1 : $1 < n$
- la mémoire principale est accessible à tous les processeurs
- le sous-système d'Entrée Sortie, avec ses canaux et périphériques, est accessible aux processeurs de contrôle
- le système tout entier est contrôlé par un "operating-system" intégré.

Un système informatique remplissant toutes ces conditions sauf la première est appelé un système monoprocesseur.

La structure multiprocesseur est symétrique si tous les processeurs sont identiques ; la structure est asymétrique si l'une au moins des unités possède une fonction particulière.

Les systèmes multiprocesseurs peuvent être répartis en deux classes :

- système multiprocesseur homogène
- système multiprocesseur inhomogène.

Un système est dit homogène si les différents processeurs, modules de mémoire et processeurs d'Entrée/Sortie sont structurés de la même façon ; dans le cas contraire le système est dit inhomogène.

Les systèmes multiprocesseurs peuvent être également classifiés suivant le degré d'interaction entre les modules de traitement. On distingue :

- les processeurs fortement couplés
- les processeurs faiblement couplés.

Dans le cas de processeurs fortement couplés (un grand nombre de processeurs partageant une mémoire commune via un bus multiplexé de grande vitesse), les processeurs opérent suivant un schéma très strict d'utilisation du bus, ce schéma est implanté au niveau matériel à l'interface bus/processeur.

Dans le cas de processeurs faiblement couplés, les communications et interactions se font sur la base d'échange d'informations. Il n'y a pas d'instruction propre aux interactions.

## II . CLASSIFICATION DES STRUCTURES D'ORDINATEUR

Il semble difficile de créer une classification pour le très large éventail d'architectures d'ordinateurs. Les types de classification sont très différents suivant les critères qu'ils utilisent. Néanmoins, une classification peut nous donner, sous forme d'un petit résumé, des informations intéressantes sur la structure générale et la puissance des systèmes pour une technologie donnée.

En 1972, Flynn classifia les structures d'ordinateur en quatre catégories en fonction du traitement des flots d'instructions et de données :

SISD = Single Instruction Single Data
MISD = Multiple Instruction Single Data
SIMD = Single Instruction Multiple Data
MIMD = Multiple Instruction Multiple Data

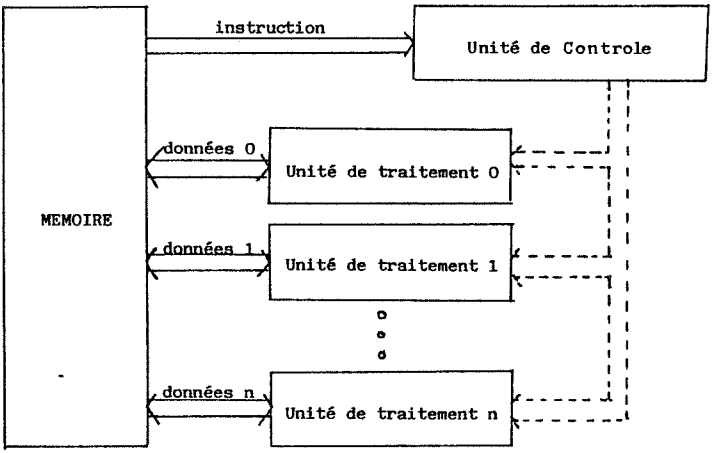
Un grand nombre d'ordinateurs entre dans cette classification, ce qui la rend très utile. Voici quelques détails complémentaires :

La structure SISD représente les habituels systèmes monoprocesseurs. Une unique donnée est traitée par une unique instruction.

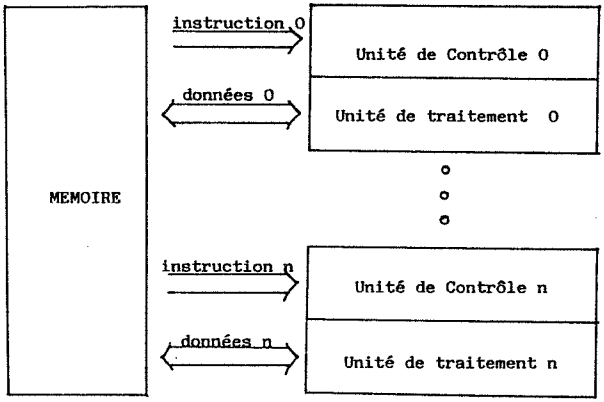
La structure MISD est un flot de plusieurs instructions qui traitent une même donnée.

La structure SIMD est le premier type ou apparaissent vraiment des activités parallèles. Une unité de contrôle décode une instruction. Cette instruction est soit exécutée dans l'unité de contrôle elle-même (saut, branchement conditionnel) soit adressée à d'autres unités. Les unités de traitement opèrent simultanément sur différentes données.

La structure MIMD : plusieurs processeurs travaillent sur plusieurs données, le plus souvent de façon asynchrone, et partageant une mémoire commune. Cette catégorie regroupe un grand nombre de systèmes, depuis le cas le plus simple où chaque processeur exécute son propre programme sur ses propres données jusqu'au cas de processeurs très fortement couplés partageant une mémoire commune.




Structure SIMD




Structure

FIGURE 1.

La classification de Flynn, bien que très utile, s'avère parfois insuffisante En effet, les frontières entre les diverses catégories ne sont pas toujours très nettes. Signalons par exemple, le cas du pipeline qui est intermédiaire entre les structures SIMD et SISD (modulo une période d'horloge).

## B . REALISATION DE CES ARCHITECTURES

### I . MULTIPROCESSEURS ET OPERATIONS SIMULTANEES

Si l'on passe en revue l'évolution des multiprocesseurs, on peut trouver une grande variété de systèmes à classer dans les catégories SIMD ou MIMD. On observe dans ces catégories les techniques du parallèlisme et du pipeline ou des combinaisons des deux.

Dans le cas du pipeline, la tâche à exécuter est partagée en petites parties qui sont affectées à des processeurs spécifiques.

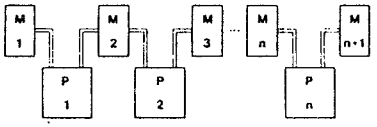



Figure 2 . Processeur pipeline

Le flot d'instruction doit être synchronisé. Le taux maximum d'entrée de nouvelles données dans le pipeline dépend uniquement du temps le plus long que l'on doit passer à l'un des étages mais pas du nombre d'étages.

Un système multiprocesseur qui correspond bien à la catégorie SIMD est le processeur vectoriel.

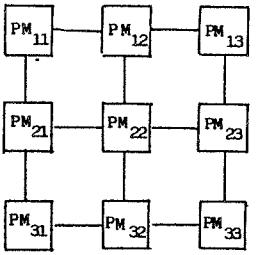



Figure 3 . Matrice de processeur

Une instruction opère sur plusieurs données simultanément. On trouve dans cette catégorie des systèmes tel ILLIAC IV et STARAN. Le grand avantage de ce type de multiprocesseur est le grand taux de traitement que l'on peut obtenir grâce aux opérations simultanées (parallèles). Pour permettre le parallèlisme, le problème qui doit être résolu sur les processeurs vectoriels possède les caractéristiques suivantes :

- les calculs doivent être décrits sous forme d'instructions relatives à des vecteurs, c'est-à-dire de nombreuses opérations identiques effectuées sur des données différentes.
- les opérandes manipulés simultanément doivent être exécutés simultanément.

Des exemples de calcul de ce type sont la résolution d'équations différentielles, les manipulations de matrice comme dans le cas de la prévision du temps, le contrôle du trafic aérien ou les simulations "on-line".

## II . SYSTEMES DISTRIBUES

Dans les systèmes répartis, les processeurs peuvent être localement ou géographiquement distribués (par exemple pour un réseau). Les communications entre processeurs se limitent généralement au passage de messsages par des ressources communes. Les systèmes distribués présentent quelques inconvénients :

- la charge du système est rarement connu de façon explicite, ce qui signifie une utilisation pas toujours efficace des processeurs.
- la défaillance d'un processeur peut dégrader sérieusement les performances du système.

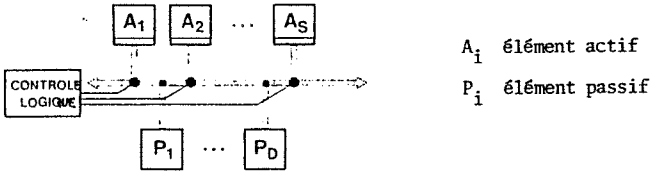



Figure 4 . Bus multiplexé.

Dans le cas de processeurs fortement couplés, les processeurs travaillent sous le strict contrôle d'un schéma d'utilisation du bus. La différence par rapport aux structures précédentes est qu'à certains moments, deux processeurs ou plus peuvent exécuter de façon autonome des instructions indépendantes.

Très souvent la structure de contrôle de processeurs fortement couplés est inhomogène, c'est-à-dire qu'elle est formée d'unités fonctionnelles spécialisées (des exemples typiques sont le CDC 6600 et IBM 360/91).

On trouve également de nombreux systèmes formés de combinaisons de processeurs fortement couplés et de processeurs faiblement coûplés. Ce qui signifie que la catégorie MIMD peut être étendue et inclure les réseaux locaux et distribués.

## III . STRUCTURES D'INTERCONNEXION POUR LES SYSTEMES PARALLELES

Les problèmes essentiels pour ces types d'architectures sont les conflits d'accès aux données.

### 1. Transmission des données

La transmission de données peut se faire en série ou en parallèle. Dans les systèmes d'ordinateurs parallèles, les distances sont en général petites mais on a besoin d'un canal de grande capacité, c'est pourquoi l'on préfère des bus parallèles.

A l'opposé du cas des processeurs fortement couplés, il n'y a pas d'interaction d'instructions entre des processeurs faiblement couplés, bien qu'ils puissent utiliser une mémoire commune pour passer des informations.

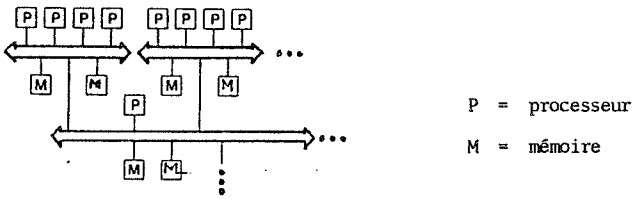



Figure 5 . Organisation hiérarchique d'un système d'ordinateur.

Si le système est composé de différents modules à fonctions variées, les connexions sont nécessaires pour donner à chaque processeur l'accès exclusif au bus pour l'exécution d'au moins une instruction complète. Le coût et la souplesse d'un système multiprocesseur sont influencés par :

- l'allocation et la synchronisation des processeurs et des tâches
- le contrôle des ressources du système
- les modules fonctionnels d'interconnexion
- les communications interprocesseurs.

## 2. Les interconnexions

Les types d'interconnexions entre processeurs, la mémoire partagée et les Entrées/Sorties partagées sont le bus commun et la mémoire multiport.

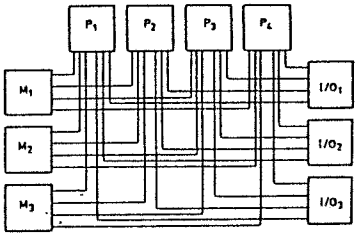



Figure 7 . Système avec des composants multiport.

Les systèmes multiport emploient de multiples bus spécialisés, un arbitrage logique doit être associé à chaque ensemble multiport. Le plus grand inconvénient de cette structure est le coût du cablage de toutes les connexions.

Dans une structure entièrement connectée, chaque noeud est connecté à tous les autres, la constitution du réseau est très coûteuse.

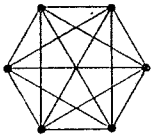


Figure 7 . Structure entièrement connectée.

S'il y a K noeuds, le nombre de connexions $Z_{total}$ est donné par :

$$Z_{total} = \frac{K \cdot (K-1)}{2}$$

Dans ce cas, la charge par connexion est minimale.

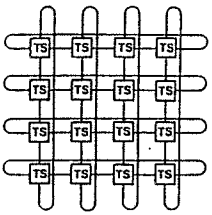



Figure 8 . Structure matricielle.

Pour la structure en matrice de la Figure 8, on obtient :

$$Z_{total} = 2\ K$$

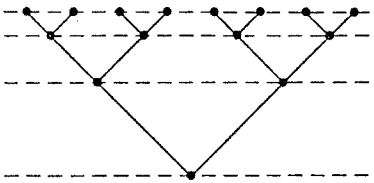


Figure 9 . Arbre binaire.

La Figure 9 montre une structure en arbre binaire où chaque processeur est connecté à deux enfants et un unique parent. Les structures en arbre offrent le nombre minimal de connexion par processeur. Le nombre total de connexions est :

$$Z_{total} = K - 1$$

Pour de nombreux problèmes la structure en arbre est efficace. Ceci n'est pas étonnant car l'on sait l'importance des structures en arbre en algorithmique. Dans ce cas la structure du matériel est un reflet des concepts algorithmiques.

Pour éviter les goulots d'étranglements dans les structures en arbre, on a défini des extensions de ces structures (Figure 10) (en ajoutant des connexions sur des noeuds de même niveau).

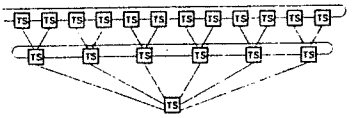



Figure 10 . Extension de la structure en arbre.

Dans ce cas de l'extension, on obtient :

$$Z_{total} = 2\ (K - 1)$$

Les structures en arbre offrent des vitesses d'exécution rapides et ce sont de bonnes solutions lorsqu'il y a à connecter un grand nombre de noeuds.

Dans le futur apparaîtront de nouveaux aspects du problème des mutliprocesseurs tout particulièrement dans le domaine des systèmes reconfigurables.

## C . MATERIELS ET LOGICIELS PARALLELES

### I . QUELQUES EXEMPLES D'ORDINATEURS PARALLELES

- le cas de Cray - 1 (Vectorisation des registres, segmentation des différentes opérations, chaînage des unités fonctionnelles).
- l'ordinateur CYBER 205 dont les caractéristiques sont :
  - traitement des instructions de vecteur à partir de la mémoire centrale (pas de registres de vecteurs)
  - pipeline non spécialisé ("general purpose pipeline")
- l'ordinateur AP - 120B FPS
  - 10 fois moins rapide que le Cray 1
  - 50 fois moins cher.
- réseau local à base de SM 90
- l'ordinateur MAP - 6400 CSPI.
- DAP : matrice de processus.

### II . LES LANGAGES DES SYSTEMES PARALLELES

Des notions limitées de parallèlisme ont été introduites dans des langages tel qu'ALGOL 68 ou PL 1. Il fallut néanmoins attendre les années soixante dix pour que les notions de parallèlisme soient totalement explicitées dans un langage : "Pascal concurrent". Enfin, et très récemment, ces notions furent précisées dans des langages tels que CSP ou ADA.

Des instructions spécifiques furent tout d'abord adjointes à des langages séquentiels pour rendre compte de l'utilisation de machines vectorielles. La description d'un parallèlisme plus complexe a conduit à des études remettant en question les structures de programmation usuelles. Peu de ces langages ont dépassé le stade de la proposition faute de structure de machine autorisant une implantation efficace.

Pour les langages parallèles, une notion est très importante et doit être précisée de façon correcte : il s'agit de la notion de processus. Cette notion représente pour les systèmes parallèles ce que représentent les notions de fonctions, de procédures, de modules pour la programmation séquentielle : c'est l'unité de programmation indépendante et c'est l'unité d'exécution avec son environnement propre.

En plus des actions exprimées par les instructions habituelles, le fonctionnement des processus fait aussi intervenir des actions de communication : un processus envoie de l'information en direction d'autres processus et il reçoit de l'information en provenance d'autres processus.

La notion de processus communicant est bien identifiée, mais il reste beaucoup à faire quant à la sémantique de ces objets, les éléments linguistiques pour les décrire restant généralement encore assez pauvres.

En effet, quand on utilise un processus, c'est-à-dire quand on le fait communiquer avec d'autres processus et quand on programme ce processus lui-même, il faut connaître la façon dont il sera perçu par ses utilisateurs : noms des moyens de communication ("canaux", "portes",...), type des informations qu'il faut envoyer et recevoir, séquencement possible dans l'usage des moyens de communication..., cela doit être exprimé indépendamment de tout choix de représentation relatif au fonctionnement et à l'organisation interne du processus.

Enfin, les travaux sur l'algèbre des processus doivent aboutir à envisager l'élaboration d'un langage rendant possible la construction hiérarchique de processus : un processus pouvant être obtenu par composition de processus plus élémentaires, les processus les plus primitifs étant ceux dont la construction est faite en termes d'instructions d'un langage algorithmique.

Pour décrire un système, deux niveaux d'expression devront être disponibles : programmer un nouveau processus en utilisant un mode algorithmique, construire un système en appliquant des opérateurs de composition à des processus choisis dans la base de données.

Un langage seul n'est pas suffisant. Un système composé de processus qui fonctionnent en parallèle et qui échangent des messages est un objet dont le comportement est beaucoup trop complexe pour que les techniques classiques du "debugging" permettent de parvenir à un degré de confiance satisfaisant sur le bon déroublement de ces opérations. Un langage seul et son compilateur ne peuvent donc suffire, il faut q'ils soient accompagnés de tout un environnement pour l'aide à la conception des systèmes parallèles.

# CHAPITRE

# 2

## CHAPITRE II . ALGORITHMES PARALLELES, DEFINITION ET STRUCTURE.

A . ELEMENTS FINIS ET PARALLELISME

Introduction

I . RAPPELS SUR LA METHODE DES ELEMENTS FINIS

1. Structure des systèmes physiques
   a) Définition
   b) Problèmes d'équilibre
   c) Problèmes de propagation
2. Formes intégrales
   a) Définition du résidu
   b) Méthode des résidus pondérés
   c) Intégrations par parties
3. Fonctionnelles
   a) Définitions
   b) Relation avec les formes intégrales
4. Formulation de la méthode
   a) Approximation des fonctions solution
   b) Discrétisation d'une forme intégrale
   c) Méthode de Galerkine
   d) Méthode de Ritz

II . EXEMPLE DE FORMULATION D'UN PROBLEME PAR ELEMENTS FINIS

III . LOGICIELS D'ELEMENTS FINIS ET UTILISATION DU PARALLELISME

1. Structure d'un logiciel de calcul de champ par éléments finis
   a) Définition du problème
   b) La résolution
   c) L'exploitation des résultats
2. Les problèmes de temps d'exécution
3. Introduction du parallèlisme

B . ALGORITHMES PARALLELES POUR LA CONSTITUTION DU SYSTEME LINEAIRE

I . METHODE DE CONSTRUCTION DU SYSTEME LINEAIRE

1. Formes intégrales élémentaires
2. Passage à l'élément de référence
3. Méthode d'intégration numérique
4. Assemblage des matrices.

II . PARALLELISATION DES ALGORITHMES PRECEDENTS

1. Analyse des algorithmes précédents
2. Evaluation des temps de calcul
3. Stratégies pour l'intégration parallèle
   a) Première stratégie
   b) Seconde stratégie
   c) Remarque
4. Remarque sur l'assemblage.

C . ALGORITHMES PARALLELES POUR LA RESOLUTION DU SYSTEME LINEAIRE

I . PROPRIETES DES MATRICES GLOBALES

1. Structure de bande
2. Symétrie
3. Méthodes de stockage
   a) Matrice pleine non symétrique
   b) Matrice pleine symétrique
   c) Matrice bande non symétrique
   d) Matrice bande symétrique
   e) Matrice symétrique creuse

II . METHODES NUMERIQUES DE RESOLUTION DES SYSTEMES LINEAIRES

1. Introduction
2. Méthodes directes
   a) Méthode d'élimination de Gauss
   b) Méthodes de décomposition
   c) Décomposition de Cholesky
3. Méthodes itératives
   a) Préconditionnement
   b) Minimisation d'une fonction
   c) Gradient Conjugué avec Préconditionnement Incomplet de Cholesky (ICCG)

III . PARALLELISATION D'UNE METHODE DIRECTE

1. Algorithmes de résolution avec q processus
2. Détail des calculs
3. Algorithme pour un processus
4. Remarques
5. Conditions de validité de la méthode

IV . PARALLELISATION D'UNE METHODE ITERATIVE

1. Modification du préconditionnement
2. Algorithme parallèle
3. Remarque
4. Convergence de l'algorithme.

# CHAPITRE II . ALGORITHMES PARALLELES : DEFINITION ET STRUCTURE

## A . ELEMENTS FINIS ET PARALLELISME

### Introduction

La plupart des méthodes informatiques consistent à remplacer dans des équations analytiques des fonctions quelconques par des séries de fonctions, le plus souvent continues par morceau. Le problème est ainsi ramené à un système d'équations algébriques, facilement traité par ordinateur.
La méthode des éléments finis suit ce principe. Elle permet de résoudre les problèmes d'équations aux dérivées partielles par discrétisation du domaine d'étude et par projection de la solution sur une famille de fonctions a priori connues.
Les équations de base sur lesquelles nous avons fait porter notre travail, sont les équations de Maxwell. On peut alors généralement définir une fonctionnelle qui correspond à une formulation énergétique sur le domaine. La solution du problème sera la fonction qui rend optimale la valeur de la fonctionnelle.
Cette méthode conduit à des logiciels structurés où nous étudierons l'introduction d'algorithmes parallèles.

### I . RAPPELS SUR LA METHODE DES ELEMENTS FINIS

#### 1. Structure des systèmes physiques

a) Définitions :

Un système physique est représenté par des variables dépendant des coordonnées d'espace (x, y, z) et du temps t.
Le système est statique si les variables ne dépendent pas du temps.
Certaines variables du système sont connues a priori : elles sont fixées par les conditions aux limites, les propriétés physiques,... Les autres variables notées u, sont inconnues : potentiel vecteur, température, vitesse...
Le nombre de degrés de liberté d'un système est le nombre de paramètres nécessaires pour définir u.
Un système est discret s'il possède un nombre de degrés de liberté fini, sinon il est dit continu.

b) Problème d'équilibre :

C'est le cas des systèmes statiques.
Pour un système discret on obtient :

$$[K]\ \{U\} = \{F\}$$

[K] = matrice caractérisant le système
{U} = variables inconnues
{F} = sollicitations connues.

Pour un système continu :

$$L(u) + f_v = o \text{ sur un domaine V}$$
$$C(u) = f_s \text{ sur la frontière S de V}$$

L et C : opérateurs différentiels caractérisant le système.
u = fonctions inconnues
$f_v$ et $f_s$ : fonctions connues représentant les sollicitations.

c) Problèmes de propagation :

Il s'agit de calculer u(x, y, z, t) pour $t > t_o$.

Pour un système discret :

$$[M]\ \frac{d^2}{dt^2}\{U\} + [C]\ \frac{t}{dt}\{U\} + [K]\ \{U\} = \{F(t)\} \quad \forall\ t > t_o$$

et à $t = t_o$ $\{U\} = \{U_o\}$ et $\frac{d}{dt}\{U\} = \{U_o\}$

[M] est la matrice de masse et [C] la matrice d'amortissement :

Pour le système continu :

$$m\frac{\partial^2 u}{\partial t^2} + c\frac{\partial u}{\partial t} + L(u) + f_v = o \text{ sur V et } C(u) = f_s \text{ sur S}$$

et $u = u_o$ , $\frac{\partial u}{\partial t} = u_o$ pour $t = t_o$

Si [K], [M], [C] et [F] sont indépendants de u, le système est linéaire (resp. si $f_v$, $f_s$, m, c sont indépendants de u et L et C linéaires en u).

Un système différentiel linéaire est dit symétrique ou auto-adjoint si :

$$\int_V u\, L(v)\, dV = \int_V v\, L(u)\, dV \quad \forall u, v$$

où u et v sont des fonctions dérivales sur V et telles que C(u) = C(v) = 0.

Il est positif si :

$$\int_V u\, L(u)\, dV \geq 0 \qquad \forall u$$

Il est défini positif si :

$$\forall u \neq 0 \qquad \int_V u\, L(u)\, dV > 0$$

## 2. Formes intégrales

### a) Définition du résidu :

Considérons un système physique continu représenté par un système d'équations aux dérivées partielles d'ordre m :

$$L(u) + f_V = 0 \quad \text{sur le domaine V} \qquad (1)$$
$$C(u) = f_S \quad \text{sur la frontière.}$$

On appelle résidu la quantité R(u) définie par :

$$R(u) = L(u) + f_V$$

qui s'annule lorsque u est solution du système.

### b) Méthode des résidus pondérés :

La méthode des résidus pondérés consiste à rechercher les fonctions u qui annulent la forme intégrale :

$$W(u) = \int_V \beta\, R(u)\, dV = \int \beta\, (L(u) + f_V)\, dV \qquad (2)$$

Toute solution u qui vérifie (1) annule la forme intégrale (2). Par contre, la fonction u annulant (2) dépend du choix des fonctions $\beta$.
Si le nombre de fonctions $\beta$ est fini, la solution qui annule (2) est une solution approximative de (1).

### c) Intégration par parties :

L'intégration par parties permet de transformer une forme intégrale du type (2) en une forme intégrale dite faible.
Les formes intégrales dites faibles représentent les avantages suivants :

- les conditions de dérivabilité de u sont moins fortes car on diminue l'ordre des dérivées de u dans (2).
- certaines conditions aux limites sont directement prises en compte.

Par contre, l'intégration par parties fait intervenir des dérivées d'ordre supérieur ainsi que des conditions aux limites supplémentaires pour les fonctions $\beta$.

## 3. Fonctionnelles :

### a) Définitions :

Une fonctionnelle $\Pi$ est une fonction d'un ensemble de fonctions et de leurs dérivées :

$$\Pi = \Pi(u, u'_x, \ldots)$$

la variation de $\Pi$ est donnée par :

$$\delta\Pi = \frac{\partial \Pi}{\partial u}\,\delta u + \frac{\partial \Pi}{\partial (u'_x)}\,\delta(u'_x) + \ldots$$

### b) Relation avec les formes intégrales :

Pour certains problèmes, il est possible de construire une fonctionnelle $\Pi$ telle que :

$$\delta\Pi \equiv W = 0$$

où W est une forme intégrale particulière dite de Galerkine obtenue en choisissant $\beta = \delta u$ :

$$W = \int_V \delta u\,(L(u) + f_v)\,dV = 0$$

Ainsi, la méthode des résidus pondérés peut être équivalente à rendre stationnaire une fonctionnelle (dans le cas où celle-ci existe). Ceci permet d'obtenir une formulation intégrale directement à partir de la fonctionnelle, ce qui peut être utile lorsque celle-ci est plus simple que les équations aux dérivées partielles.

## 4. Formulation de la méthode

### a) Approximation des fonctions solutions :

Les fonctions u sont approchées par éléments finis, qui est un cas particulier d'approximation par sous domaine, dont voici les caractéristiques :

- on identifie un ensemble de sous-domaines $V^e$, appelés éléments finis, du domaine V
- pour chaque sous domaine $V^e$ on définit une fonction approchée $u^e(x, y, z)$ ne faisant intervenir que les variables nodales attachées au sous domaine $V^e$
- les fonctions $u^e(x, y, z)$ sont continues sur la frontière des sous domaines.

L'approximation par éléments finis se fait en deux temps :
- définir la géométrie de tous les éléments,
- construire les fonctions d'interpolation $\alpha_i(x, y, z)$ correspondant à chaque élément.

On a alors pour chaque sous domaine $V^e$ :

$$u^e(x,y,z) = \Sigma\ \alpha_i(x,y,z)\ u_i = < \alpha_1\ \alpha_2\ \ldots\ \alpha_i\ \ldots > \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_i \\ \vdots \end{Bmatrix}$$

où $u_i$ est la valeur de u aux noeuds d'interpolation (qui sont les points où la fonction approchée et la fonction exacte coïncident).

On cherche u annulant l'intégrale $W = \int_V \beta \,.\, R(u)\ dV = \int_V \beta(L(u) + f_v)\ dV$ pour toute fonction $\beta$.

Pour trouver une fonction approchée u, il faut discrétiser en deux étapes :
- choisir une approximation par éléments finis de u :

$$u = u(u_1, u_2, \ldots, u_i, \ldots u_n)$$

d'où $W = \int_V \beta \,.\, (L(u(u_1, u_2, \ldots, u_n)) + f_v)\ dV = 0$

- choisir une famille de fonctions $\beta$.

c) Méthode de Galerkine :

On choisit pour cette méthode :

$$\beta = < \beta_1, \beta_2, \ldots, \beta_n >$$

d'où $W = \int_V \beta\ (L(u) + f_v)\ dV - 0$

$$W = \int_V \begin{Bmatrix} \beta_1 \\ \beta_2 \\ \vdots \\ \beta_n \end{Bmatrix} [L[<\alpha_1,\alpha_2,\ldots,\alpha_n> \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix}] + f_v\ ]\ dV = 0$$

W devant s'annuler pour tout $u_i$, la relation précédente est équivalente à un système de n équations algébriques :

$$W_1 = \int_V \beta_1\ [\ <L(\alpha_1), L(\alpha_2), \ldots L(\alpha_n)> \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix} + f_v\ ]\ dV = 0$$

$$\vdots$$

$$W_n = \int_V \beta_n\ [\ < L(\alpha_1), \ldots L(\alpha_n)\ > \begin{Bmatrix} u_1 \\ \vdots \\ u_n \end{Bmatrix} + f_v\ ]\ dV = 0$$

d) Méthode de Ritz :

Pour cette méthode, on discrétise une fonctionnelle $\Pi$ en utilisant une approximation de u par éléments finis, puis on écrit les conditions de stationnarité de $\Pi$ par rapport aux paramètres de l'approximation par éléments finis.

$$\Pi(u) = \Pi\,[u(u_1, u_2, \ldots u_n)]$$

$$\delta\Pi(u_1, u_2, \ldots, u_n) = \frac{\partial \Pi}{\partial u_1}\delta u_1 + \frac{\partial \Pi}{\partial u_2} + \ldots + \frac{\partial \Pi}{\partial u_n}\delta u_n = 0$$

Ce qui donne n équations :

$$\frac{\partial \Pi}{\partial u_1} = 0 \quad ; \quad \frac{\partial \Pi}{\partial u_2} = 0 \quad ; \ldots ; \quad \frac{\partial \Pi}{\partial u_n} = 0.$$

Si la fonctionnelle existe, $\delta\Pi$ est identique à la forme intégrale W de type Galerkine. La solution obtenue par la méthode de Ritz est alors identique à celle obtenue par la méthode de Galerkine.

## II . EXEMPLE DE FORMULATION D'UN PROBLEME PAR ELEMENTS FINIS

Pour notre exemple, nous traiterons un problème d'électrostatique.
Les équations de Maxwell deviennent :

$$\vec{\text{rot}}\ \vec{E} = \vec{0}$$

$$\text{div}\ \vec{D} = \rho$$

où $\vec{E}$ est le champ électrique (V/m), $\vec{D}$ le vecteur induction électrique ($C/m^2$) et $\rho$ la densité de charge électrique ($C/m^3$). On a de plus, la relation supplémentaire :

$$\vec{D} = \varepsilon_o\, \varepsilon_r\, \vec{E}$$

$\varepsilon_o = \frac{10^{-9}}{36\,\Pi}$ (F/m) permittivité du vide

$\varepsilon_r$ permittivité du milieu avec :

| | |
|---|---|
| isotrope | $\varepsilon_r$ scalaire |
| anisotrope | $\varepsilon_r$ tenseur |
| linéaire | $\varepsilon_r(x,y,z)$ |
| non linéaire | $\varepsilon_r(x,y,z,E)$ |

On définit le potentiel électrique u(x,y,z) tel que $\vec{E} = -\ \vec{\text{grad}}\ u$
automatiquement on obtient :

$$\vec{\text{rot}}\ \vec{E} = -\ \vec{\text{rot}}\ \vec{\text{grad}}\ u = \vec{0}$$

d'où l'équation de V :

$$\text{div}\ (\varepsilon_r\ \vec{\text{grad}}\ u) = -\ \rho/\varepsilon_o \text{ sur le domaine V.}$$

On doit ajouter des conditions aux limites :

$$u = u' \text{ sur } (\Gamma_1)$$

ou bien

$$\frac{\partial u}{\partial n} = u'' \text{ sur } (\Gamma_2)$$

Considérons la fonctionnelle de coénergie associée à cette équation, avec la condition aux limites $u = u'$ sur $\Gamma_1$ :

$$\Pi(u(x,y,z)) = \int_V \left(\int_0^E D \cdot dE - \rho u\right) dV + \int_{\Gamma_1} D'_n \cdot u \cdot d\Gamma$$

La fonction $u(x,y,z)$ qui vérifie $u = u'$ sur $\Gamma_1$ et qui rend extrêmale la fonctionnelle $\Pi$ est la solution du problème différentiel.

Considérons maintenant une famille de fonctions d'approximation :

$$u(x,y,z) = u_o(x,y,z) + \sum_{i=1}^{N} \alpha_i(x,y,z)\, u_i \qquad \text{(sur V)}$$

avec :

$u_o(x,y,z) = u'(x,y,z)$ connu sur $\Gamma_1$
$\alpha_i(x,y,z) = 0$ sur $\Gamma_1$
$u_i$ = paramètres à déterminer.

On voit que si les fonctions d'approximation sont fixées, $\Pi$ n'est plus fonction que des valeurs des coefficients $u_i$. On cherche ces coefficients pour rendre $\Pi$ extrêmale, c'est-à-dire :

$$\frac{\partial \Pi}{\partial u_i} = 0 \quad (i = 1 \text{ à } N)$$

$$\frac{\partial}{\partial u_i} \int_0^E \vec{D} \cdot d\vec{E} = \vec{D} \cdot \frac{\partial \vec{E}}{\partial u_i} = \frac{\partial \vec{E}}{\partial u_i} \cdot \vec{D}$$

d'où : $$\frac{\partial \Pi}{\partial u_i} = \int_V \frac{\partial \vec{E}}{\partial u_i} \cdot dD\, dV + \int_\Gamma \frac{\partial u}{\partial u_i} D'_n \, d\Gamma - \int_V \rho \frac{\partial u}{\partial u_i} d V$$

or : $$u(x,y,z) = u_o(x,y,z) + \sum_{j=1}^{N} \alpha_j(x,y,z) \cdot u_j$$

$$\vec{E}(x,y,z) = - \vec{\text{grad}}\, u_o - \sum_{j=1}^{N} \vec{\text{grad}}\, \alpha_j \cdot V_j$$

$$\vec{D}(x,y,z) = - \varepsilon_o \varepsilon_r \left[\vec{\text{grad}}\, u_o + \sum_{j=1}^{N} \vec{\text{grad}}\, \alpha_j \cdot u_j\right]$$

$$\frac{\partial u}{\partial u_i} = \alpha_i$$

$$\frac{\partial E}{\partial u_i} = - \vec{\text{grad}}\, \alpha_i$$

Ce qui donne la formule :

$$\frac{\partial \Pi}{\partial u_i} = \int_V \vec{grad}\ \alpha_i \cdot \varepsilon_o\ \varepsilon_r\ [\vec{grad}\ u_o + \sum_{j=1}^{N} \vec{grad}\ \alpha_j \cdot u_j]\ dV$$

$$+ \int_\Gamma \alpha_i \cdot D'_n\ d\Gamma - \int_V \rho\ \alpha_i\ dV = 0 \quad \text{pour } i = 1 \text{ à } N$$

Si nous supposons avoir un problème linéaire, donc $\varepsilon_r$ est indépendant de E et $D'_n$ indépendant de u, nous obtenons un système linéaire de N équations à N inconnues :

$$[K]\ \{U\} = \{F\}$$

avec $K_{ij} = \int_V \vec{grad}\ \alpha_i \cdot \varepsilon_o\ \varepsilon_r \cdot \vec{grad}\ \alpha_j\ dV$

$$F_i = - \int_V \vec{grad}\ \alpha_i \cdot \varepsilon_o\ \varepsilon_r \cdot \vec{grad}\ u_o\ dV - \int_\Gamma \alpha_i\ D'_n\ d\Gamma + \int_V \alpha_i\ \rho\ dV$$

Cette méthode permet donc de passer d'équations analytiques à des équations algébriques que l'on peut résoudre sur ordinateur.

Il reste à définir les fonctions d'approximation. Sur le domaine V de contour $\Gamma$ on définit un certain nombre de points appelés noeuds. Le domaine V est divisé en sous domaines, le plus souvent triangulaires ou rectangulaires $V^k$ :

$$V = V^1 \cup V^2 \cup \ldots \cup \ldots$$
$$V^k \cap V^L = \emptyset \quad \text{si} \quad k \neq L$$

Généralement, les noeuds occupent les sommets et quelques points intermédiaires dans le domaine.

Un même sous domaine ne doit pas contenir des régions aux propriétés physiques différentes.

On choisit alors les fonctions d'approximation de telle sorte que :

sur un élément $u(x,y,z) = \sum_{\substack{i \\ \text{noeuds de} \\ \text{l'élément}}} \alpha_i^k\ (x,y,z)\ u_i \quad (V^k)$

globalement : $u(x,y,z) = \sum_{\substack{i \\ \text{noeuds du} \\ \text{domaine}}} \alpha_i\ (x,y,z)\ u_i \quad (V)$

On a alors :

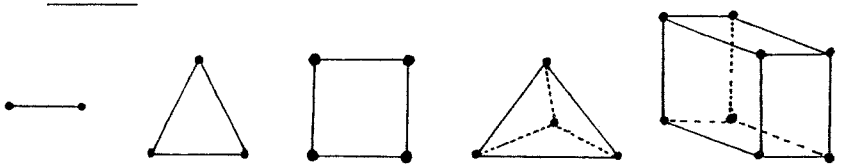
au noeud i : $\alpha_i\ (x_i, y_i, z_i) = 1$

$\alpha_j\ (x_i, y_i, z_i) = 0 \quad$ si $j \neq i$

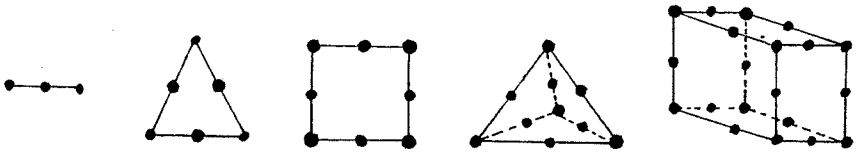
partout : $\Sigma\ \alpha_i\ (x, y, z) = 1$

Les fonctions d'approximation sont le plus souvent des polynômes dont le degré varie avec l'ordre de l'élément fini que l'on utilise, c'est-à-dire le nombre de noeuds de cet élément :

1er ordre



2ème ordre



## III . LOGICIEL D'ELEMENTS FINIS ET UTILISATION DU PARALLELISME

### 1. Structure d'un logiciel de calcul de champs par éléments finis

a) Définition du problème :

Définir la géométrie du problème, les propriétés physiques des différents domaines ainsi que les conditions aux limites. Cette partie contient également ce que l'on nomme le maillage, c'est-à-dire la définition des éléments finis à partir du domaine initial et des noeuds que l'on y a placés.

b) La résolution :

Cette partie se compose de trois étapes :

- l'intégration : les différentes intégrales des formules précédentes sont calculées en effectuant la somme des intégrales, pour chaque élément :

$$\int_V (\ldots)\, dV = \sum_k \int_{Vk} (\ldots)\, dV$$

Les intégrales $\int_{Vk} (\ldots)\, dV$ sont calculées sur chaque élément par des méthodes classiques telles que la méthode de Gauss.

- l'assemblage : lors de la phase d'assemblage on effectue l'addition des résultats précédents pour tous les éléments pour construire le système linéaire final .
- la résolution du système linéaire.

c) L'exploitation des résultats :

La résolution nous a fourni la valeur du potentiel en chaque noeud du domaine donc sur tout le domaine puisque l'on connaît les fonctions d'interpolation.
A partir de ces résultats on est capable d'effectuer un certain nombre de calculs : calcul de champ en tout point, calcul de force ou de couple, tracé d'équipotentielles.

## 2. Les problèmes de temps d'exécution

La taille des problèmes traités va en grandissant au fil des années. Elle augmente encore plus lorsque l'on veut étudier des problèmes non linéaires ou des phénomènes tels que l'hystérésis, les courants induits, l'évaluation dans le temps et surtout pour le passage de l'étude 2D (en 2 dimensions) à l'étude 3D (en 3 dimensions).
Il en résulte que les temps de calcul qui commencent déjà à devenir prohibitifs ne font que croître et ceci dans des proportions très importantes.

Il faut donc développer rapidement des structures tant au niveau matériel qu'au niveau logiciel qui permettront d'accélérer les vitesses de traitement de ces problèmes.
Le parallèlisme est une bonne réponse pour ces problèmes, nous allons montrer comment l'utiliser dans un logiciel de calcul de champs par éléments finis (FLUX 2D).

## 3. Introduction du parallèlisme

Si nous reprenons les diverses étapes d'un logiciel de calcul par éléments finis tel que nous les avons définies au paragraphe (III.1) , nous nous apercevons des faits suivants :

- La partie (a) relative à la définition du problème est très intéractive et demande de la réflexion de la part de l'utilisateur alors que la machine a peu de calcul à effectuer.

Il n'y a pas lieu d'utiliser le parallèlisme. [Nous devons néanmoins remarquer que ceci ne s'applique pas aux logiciels disposant d'un mailleur automatique, auquel cas les besoins de calculs peuvent devenir plus importants, mais nous n'avons pas étudié ce problème].

- la partie (b) concernant la résolution demande, au contraire de la précédente, un nombre volumineux de calculs alors que l'utilisateur lui n'a presque rien à faire si ce n'est qu'attendre son résultat. Le parallélisme interviendra prioritairement à ce niveau.
- la partie (c) pourrait également trouver de l'intérêt au parallèlisme.

Nous présenterons des algorithmes parallèles principalement pour la partie (b), tant au niveau de l'intégration que de la résolution.

## B . ALGORITHMES PARALLELES POUR LA CONSTITUTION DU SYSTEME LINEAIRE

### I . METHODE DE CONSTRUCTION DU SYSTEME LINEAIRE

#### 1. Formes intégrales élémentaires

Nous reprenons les formes intégrales de type Galerkine, et nous choisissons les fonctions de pondérations

$$\beta = \delta u = <\alpha_1, \alpha_2, \ldots, \alpha_n> \begin{Bmatrix} \delta u_1 \\ \delta u_2 \\ \vdots \\ \delta u_n \end{Bmatrix}$$

$$W = \int_V \delta u \, (L(u) + f_v) \, dV = 0$$

On peut calculer l'intégrale sur le domaine V par une somme sur chaque élément $V^e$ :

$$W = \sum_{e=1}^{n_e} W^e = \sum_{e=1}^{n_e} \int_{Ve} \delta u^e \, (L(u^e) + f_v) \, dV = 0$$

où $W^e$ est appelé forme intégrale élémentaire et où $u^e$ est la fonction d'approximation sur l'élément $W^e$ :

$$u^e = < \alpha_1, \alpha_2, \ldots \alpha_n > \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix}$$

Comme $<\alpha_1, \alpha_2, \ldots, \alpha_n>$ est nul à l'extérieur de $V^e$ et comme $<u_1, u_2, \ldots, u_n>$ ne fait intervenir que les variables nodales de l'élément $V^e$, pour calculer $W^e$ on n'a n'a besoin que de variables liées à l'élément e :

$$W^e = \int_{Ve} \delta u^e \, (L(u^e) + f_v) \, dV$$

$$W^e = <\delta u_1, \delta u_2, \ldots, \delta u_n> \left( \int_{Ve} <\alpha_1, \alpha_2, \ldots, \alpha_n> L(<\alpha_1, \alpha_2, \ldots, \alpha_n>) dV \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix} \right.$$

$$\left. + \int_{Ve} <\alpha_1, \alpha_2, \ldots, \alpha_n> f_v \, dV \right)$$

On peut écrire cette relation sous la forme matricielle suivante :

$$W^e = <\delta u_1, \delta u_2, \ldots, \delta u_n> \left( [k] \begin{Bmatrix} u_1 \\ u_2 \\ \vdots \\ u_n \end{Bmatrix} - \{f\} \right)$$

où $[k]$ est la matrice élémentaire

$\{f\}$ le vecteur élémentaire des sollicitations

$\{u_n\}$ le vecteur élémentaire des variables nodales

$\{\delta u_n\}$ le vecteur élémentaire des variations des variables nodales

d'où :

$$W = \sum_g W^e = \sum_e <\delta u_n> ([k] \{u_n\} - \{f\}) = 0$$

qui s'écrit sous forme matricielle :

$$W = <\delta U_n> ([K] \{U_n\} - \{F\}) = 0$$

Ceci doit être vrai pour tout $<\delta U_n>$, d'où le système :

$$[K] \{U_n\} = \{F\}$$

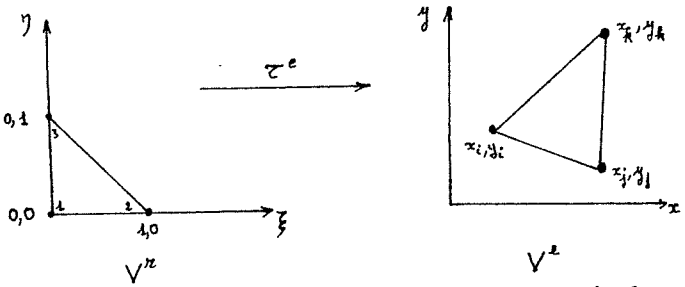
$[K]$ = matrice globale

$\{F\}$ = vecteur global des sollicitations

$\{U_n\}$ = vecteur des variables nodales.

## 2. Passage à l'élément de référence

De manière à simplifier la définition des éléments, on introduit la notion d'élément de référence $V^r$ qui est un élément de forme simple repéré dans un espace de référence qui peut être transformé en chaque élément réel par une transformation géométrique $\tau$.




La transformation $\tau$ doit être bijective et faire correspondre les noeuds et les éléments de frontières de $V^r$ et $V^e$ un à un. On appellera [J] la matrice jacobienne de la transformation $\tau$, par exemple en deux dimensions :

$$(\xi, \eta) \xrightarrow{\tau} (x, y)$$

$$[J] = \begin{vmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{vmatrix}$$

Les fonctions d'approximation seront également calculées au niveau des éléments de référence. Pour un triangle du 1er ordre, on aura :

$$\alpha_1 = 1 - \xi - \eta \quad ; \quad \alpha_2 = \xi \quad ; \quad \alpha_3 = \eta$$

et les différentes intégrales seront calculées par :

$$\int_{V^e} \ldots dV = \int_{V^r} \ldots \det [J] \, d\xi \, d\eta$$

## 3. Méthode d'intégration numérique

Pour chaque élément, on doit calculer les matrices [k] et {f} qui sont définies par des intégrales sur l'élément ou sur l'élément de référence. Ces intégrales ne sont pas calculées analytiquement. On les obtient par intégration numérique.

La méthode d'intégration de Gauss consiste à écrire, pour une fonction $g(x,y,z)$ quelconque :

$$\int_{V^e} g(x,y,z)\, dx\, dy\, z = \sum_{i=1}^{r} w_i \, g(x_i, y_i, z_i) \qquad (3)$$

où $(x_i, y_i, z_i)$ sont les coordonnées de r points d'intégration et $w_i$ les coefficients de pondération (ou poids) correspondants.

Dans la méthode de Gauss, les r coefficients et les r points d'intégration sont déterminés de manière à intégrer exactement des polynômes d'ordre $m \leq 2r - 1$.

Lorsque l'on fait une intégrale à plus d'une dimension, au lieu d'utiliser la formule directe (3), on peut utiliser la méthode produit :

$$\int_{x_1}^{x_2}\int_{y_1}^{y_2}\int_{z_1}^{z_2} g(x,y,z)\,dx\,dy\,dz = \sum_{i=1}^{r_1}\sum_{j=2}^{r_2}\sum_{k=3}^{r_3} w_i\,w_j\,w_k\,g(x_i,y_i,z_i)$$

où les $w_{i,j,k}$ et $x_i, y_j, z_k$ sont les poids et les abscisses d'intégration à une dimension.

## 4. Assemblage des matrices

L'assemblage est l'opération qui consiste à construire la matrice globale [K] et le vecteur global des sollicitations {F} à partir des matrices élémentaires [k] et {f}.

$$W = \Sigma\, W^e = \Sigma\, \langle \delta u_n \rangle\, ([k]\,\{u_n\} - \{f\}) = \langle \delta U_n \rangle\, ([K]\,\{U_n\} - \{F\})$$

on a alors :

$$W^e = \langle \delta U_n \rangle\, ([K^e]\,\{U_n\} - \{F^e\})$$

où $[K^e]$ est la matrice construite par expansion de la matrice [k] grâce à des insertions de lignes et de colonnes de zéros. [k] a pour dimension le nombre de degrés de liberté de l'élément, $[K^e]$ a pour dimension le nombre de degrés de liberté total (idem pour {f} et $[F^e]$).

L'assemblage comporte deux étapes :

- construction de la matrice étendue $[K^e]$ et du vecteur étendu $\{F^e\}$ de chaque élément.
- addition des matrices et vecteurs étendus.

Ces deux étapes sont en pratique effectuées simultanément pour éviter de construire explicitement $[K^e]$ et $\{F^e\}$. Les résultats sont directement stockés dans la matrice [K] et le vecteur {F}.

## II . PARALLELISATION DES ALGORITHMES DE CONSTRUCTION DU SYSTEME LINEAIRE

### 1. Analyse des algorithmes précédents

On a défini au Chapitre précédent, deux types de calculs dont l'organisation est très différente. D'une part l'intégration et d'autre part l'assemblage. Pour chacun de ces deux types, on distinguera deux niveaux :

A l'intégration :

(a) l'ensemble des calculs à l'intérieur d'un élément pour calculer [k] et {f},

(b) l'ensemble des calculs sur tous les éléments du domaine à l'assemblage,

(c) calcul des matrices étendues $[K^e]$ et $\{F^e\}$,

(d) somme des matrices étendues pour former [K] et {F}.

Les parties (b) et (c) sont facilement parallèlisables car elles sont composées de tâches élémentaires indépendantes les unes des autres tant au niveau des calculs qu'à celui de la localisation des données.

Les parties (a) et (d) par contre font appel à des séries de calculs qui nécessitent des données communes et pour lesquelles on est pratiquement obligé de mener les opérations de façon séquentielle.

Ceci nous amène aux réflexions suivantes :

- pour l'intégration, la tâche élémentaire sera le calcul de la matrice k et du vecteur f pour un élément, on ne doit pas séparer cette tâche. L'ensemble des tâches peuvent être conduites en parallèle.
- pour l'assemblage, on a déjà remarqué que les parties (c) et (d) ne sont en fait pas séparées dans les algorithmes. Il ne sera donc pas très intéressant de paralléliser cette partie.

### 2. Evaluation des temps de calcul

Une autre donnée milite pour ne rendre parallèle que l'intégration et non l'assemblage. Il s'agit de la comparaison des temps d'exécution de ces deux étapes.

Nous avons réalisé un ensemble de mesures de temps de calcul pour la phase d'intégration et la phase d'assemblage, pour le logiciel FLUX 2D, logiciel de calcul de champs électriques et magnétiques, développé au Laboratoire d'Electrotechnique de Grenoble et implanté sur le système Multics (HB 68).

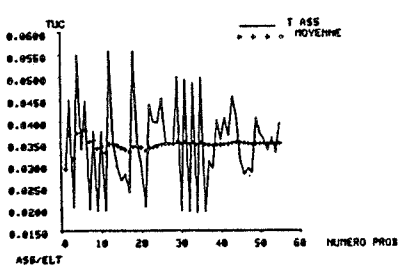



Figure 1.

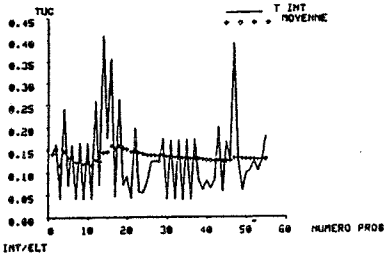



Figure 2.

Les Figures 1 et 2 donnent les temps de calcul ramenés à un élément respectivement pour l'intégration et pour l'assemblage pour 60 problèmes différents.

La Figure 3 donne l'évolution du temps de calcul pour l'intégration, l'assemblage et la résolution (méthode de Cholesky, voir Chapitre suivant) en fonction du nombre d'éléments.

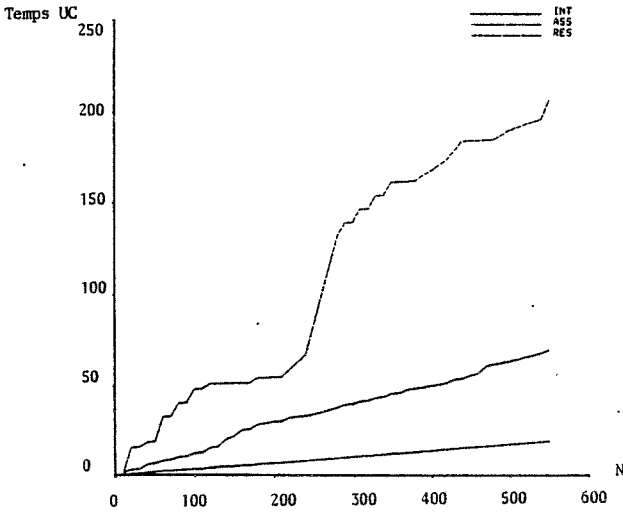



Figure 3.

On peut estimer alors que le temps de calcul par élément est :

- 0,150 pour l'intégration,
- 0,035 pour l'assemblage.

L'intégration consomme cinq fois plus de temps que l'assemblage. Il sera donc très intéressant de paralléliser l'intégration.

On notera que le temps de 0,15 seconde pour l'intégration d'un élément est une valeur moyenne. Les temps de calcul sont en fait assez dispersés par rapport à cette moyenne, cette dispersion provient de la variété des calculs à effectuer dûe aux différents types d'éléments (triangle, quadrilatère, premier ou second ordre,...) et aux différents types d'intégrales à calculer (région linéaire, non linéaire, isotrope, anisotrope, propriétés physiques diverses,...).

## 3. Stratégies pour l'intégration parallèle

Nous avons envisagé deux stratégies pour la réalisation du parallèlisme au niveau de l'intégration, voici leurs descriptions :

### a) Première stratégie :

Avec cette première stratégie, les tâches élémentaires d'intégration par élément sont réparties au fur et à mesure de l'exécution d'ensemble. Un processus parallèle choisit parmi les éléments non encore calculés un élément à traiter, il signale aux autres que cet élément est réservé puis il effectue son calcul et inscrit le résultat.

Trois zones communes sont nécessaires :

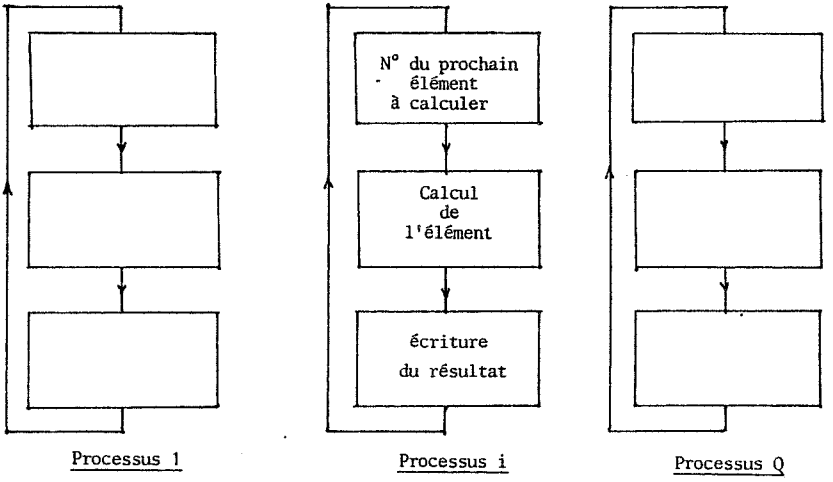
- les données relatives à chaque élément (en lecture)
- une table des éléments à calculer (en lecture écriture)
- un fichier de stockage des résultats (en écriture).

Les zones communes accédées en lecture ne posent aucun problème, par contre les zones accédées en écriture ne peuvent l'être que par un processus à la fois.

Tableau des éléments à calculer

| N° de l'élément | 1 | 2 | 3 | | N-1 | N |
|---|---|---|---|---|---|---|
| Indicateur | 1 | 1 | 0 | | 0 | 0 |

Indicateur { = 0 élément non calculé
= 1 élément calculé

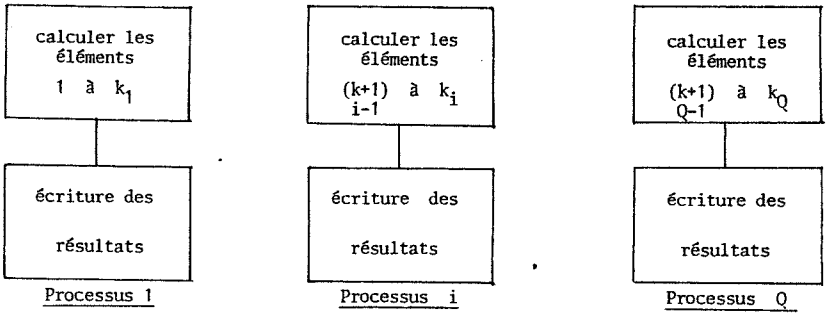



Cette stratégie présente les caractéristiques suivantes :

- c'est un ensemble de processus asynchrones, par cette méthode la répartition du calcul sur les éléments entre les divers processus est optimisée ;
- par contre, de nombreux conflits d'accès peuvent se produire au niveau des zones communes accédées en écriture. Ces conflits peuvent ralentir de façon sensible l'exécution des algorithmes.

b) Seconde stratégie :

On attribue à chaque processus parallèle un certain nombre d'éléments à intégrer. La répartition est faite a priori, et sans présager de la répartition des besoins de calcul liés à chaque élément.




Caractéristiques

- pas de problèmes de conflits d'accès,
- la répartition des éléments entre processus est empirique et non optimisée.

c) Remarque :

Si l'on traite un problème comportant un très grand nombre d'éléments, la répartition des tâches entre chaque processus doit devenir équivalente pour les deux stratégies, donc les temps de calcul de chaque processus tendent vers des valeurs identiques. Par contre les temps de réponse peuvent être très différents en fonction des conflits d'accès.
Les résultats et l'analyse des performances de ces méthodes seront donnés au Chapitre IV du rapport.

4. Remarque sur l'assemblage

On peut mettre en évidence une possibilité assez simple de parallèlisme au niveau de l'assemblage. En effet, on peut considérer qu'il existe entre l'intégration et l'assemblage un mécanisme producteur/consommateur. Le mécanisme producteur (intégration des éléments) est parallèlisé, le mécanisme consommateur (assemblage) reste séquentiel. L'interface entre les deux types de processus peut être réalisé par des tampons travaillant en bascule.

Cependant l'apport de cette méthode est très limité car la vitesse du processus de consommation est trop faible par rapport au processus de production.
C'est pourquoi dans nos réalisations nous n'avons parallélisé que l'intégration, l'assemblage n'étant effectué séquentiellement qu'une fois l'intégration terminée.

## C . ALGORITHMES PARALLELES POUR LA RESOLUTION DU SYSTEME LINEAIRE

### I . PROPRIETES DES MATRICES GLOBALES

1. Structure de bande

La matrice globale [K] est construite par addition des matrices élémentaires étendues $[K^e]$ qui comportent un grand nombre de zéros

$$[K] = \sum_{\text{éléments}} [K^e]$$

Un terme $K_{ij}$ est non nul que s'il existe un élément qui fait intervenir simultanément les variables $u_i$ et $u_j$.
La matrice [K] est une matrice creuse, et topologiquement symétrique (si $K_{ij}$ est non nul alors $K_{ji}$ est non nul).

Dans chaque ligne de [K], il existe un petit nombre d'éléments $K_{ij}$ non nuls.
De plus, si la numérotation des noeuds est bien faite, la largeur de bande $w_i$ de la ligne i reste très inférieure à la dimension N du système :

$$w_i = J_i - i \quad \text{où} \quad J_i = \max j \text{ tels que } K_{ij} \neq 0$$

$$w_i \ll N$$

de même $w = \max (w_i) \ll N$.

## 2. Symétrie

Dans de nombreux problèmes, l'opérateur L est auto-adjoint. Les matrices [k] sont symétriques, la matrice [K] l'est donc aussi :

$$K_{ij} = K_{ji}$$

Cette propriété permet des économies importantes de stockage de la matrice. Nous avons développé essentiellement des algorithmes de résolution de systèmes symétriques, mais qui pourraient être étendus à des systèmes non symétriques.

## 3. Méthodes de stockage

### a) Matrice pleine non symétrique :

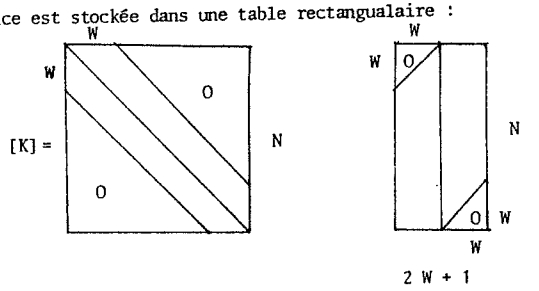
Une matrice pleine non symétrique de dimensions (N x N) occupe $N^2$ membres réels en mémoire d'ordinateur.

### b) Matrice pleine symétrique :

On ne stocke que le triangle supérieur, ligne à ligne (ou colonne à colonne). Il faudra stocker $\frac{N(N+1)}{2}$ nombres réels.

### c) Matrice bande non symétrique :

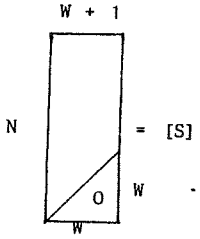
La matrice est stockée dans une table rectangulaire :




On stocke N(2W + 1) nombres réels dont W(W + 1) valeurs nulles inutiles.

d) Matrice bande symétrique :

Le triangle supérieur de la matrice est stockée dans une table rectangulaire :




On a alors :

$$K_{ij} = S_{IJ} \text{ avec } I = i, \quad J = y - i + 1, \quad j \geq i$$

Il faut stocker $N(W+1)$ nombres réels dont $\frac{W(W+1)}{2}$ valeurs nulles inutiles.

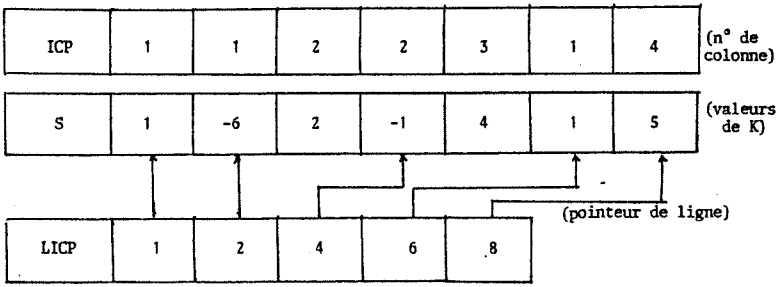
Ce stockage a été utilisé pour la résolution par la méthode de Cholesky qui sera décrite plus loin.

e) Matrice symétrique creuse :

Nous utilisons également un autre mode de stockage, notamment pour les algorithmes de résolution par la méthode ICCG (voir Chapitre suivant).
Si la matrice étudiée est très creuse, il faut éviter de stocker les valeurs nulles. On ne stocke alors que les termes non nuls et deux ensembles de pointeurs qui permettent de repérer ces termes.

Les termes non nuls de [K] sont stockés dans un vecteur S, les indices colonnes sont stockés dans un vecteur ICP. On stocke alors dans un vecteur LICP les pointeurs sur S des premiers termes de chaque ligne de [K] jusqu'à la diagonale car on ne stocke que le triangle inférieur de [K]

$$[K] = \begin{bmatrix} 1 & -6 & 0 & 1 \\ -6 & 2 & -1 & 0 \\ 0 & -1 & 4 & 0 \\ 1 & 0 & 0 & 5 \end{bmatrix}$$




Si M est le nombre moyen de termes non nul par ligne de K , alors il faut stocker (M/2+1)N nombres réels.

Pour que la méthode des éléments finis ne produise pas d'erreur trop importante, on limite souvent l'angle minimum à l'intérieur d'un élément. Une valeur usuelle de limitation est que dans chaque élément, aucun angle ne soit inférieur à 30°. Ce qui implique qu'un noeud est relié au plus à 12 autres noeuds. Donc la valeur maximale de M est 12.

Avec le stockage ainsi défini, on stocke au plus 7 N nombres réels.

## 3. Introduction des conditions aux limites

Un certain nombre de degrés de liberté $U_i$ peuvent avoir une valeur imposée $\bar{U}_i$ due aux conditions aux limites. On peut introduire ces conditions de plusieurs manières.

a) Méthode du terme unité sur la diagonale :

Elle consiste à modifier pour chaque relation $U_i = \bar{U}_i$ le vecteur {F} et la matrice [K]

$$F_j = F_j - K_{ji}\,\bar{U}_i \qquad \forall j \neq i$$
$$F_i = \bar{U}_i$$
$$K_{ij} = K_{ji} = 0 \qquad \forall j \neq i$$
$$K_{ii} = 1$$

$$\begin{bmatrix} K_{11} & \cdots & K_{1,i-1} & 0 & K_{1,i+1} & \cdots & K_{1n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ K_{i-1,1} & & K_{i-1,i-1} & 0 & K_{i-1,i+1} & \cdots & K_{i-1,n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ 0 & & 0 & 1 & 0 & \cdots & 0 \\ K_{i+1,1} & & K_{i+1,i-1} & 0 & K_{i+1,i+1} & \cdots & K_{i+1,n} \\ \vdots & & \vdots & \vdots & \vdots & & \vdots \\ K_{n,1} & & K_{n,i-1} & 0 & K_{n,i+1} & & K_{n,n} \end{bmatrix} \begin{bmatrix} U_1 \\ U_{i-1} \\ U_i \\ U_{i+1} \\ U_n \end{bmatrix} = \begin{bmatrix} F_1 - K_{1i}\,\bar{U}_i \\ F_{i-1} - K_{i-1,i}\,\bar{U}_i \\ \bar{U}_i \\ F_{i+1} - K_{i+1,i}\,\bar{U}_i \\ F_n - K_{n,i}\,\bar{U}_i \end{bmatrix}$$

b) Méthode de suppression des équations :

Il s'agit de restructurer la matrice [K] de manière à supprimer les équations correspondant aux degrés de liberté imposés $U_i$. Cette méthode à l'avantage de réduire le nombre d'inconnues du système. Comme la restructuration de la matrice [K] est couteuse, il est préférable de ne pas assembler les équations inutiles.

La restructuration de [K] et de {F} correspondant à $U_i = \bar{U}_i$ conduit à la même équation que la méthode (a) dans laquelle la ligne i et la colonne i sont supprimées.

## II . METHODES NUMERIQUES DE RESOLUTION DES SYSTEMES LINEAIRES

### 1. Introduction

La précision et le champ d'application des méthodes par éléments finis sont limités par la dimension des systèmes d'équations que l'on peut résoudre. Actuellement on résout de façon courante des systèmes de quelques milliers d'équations alors que des systèmes de quelques dizaines de milliers d'équations sont encore exceptionnels.

On classe traditionnellement les méthodes de résolution en deux catégories :

- les méthodes directes qui conduisent à la solution en un nombre d'opérations connu a priori,
- les méthodes itératives qui conduisent à la solution par une succession d'améliorations d'une solution approchée, le nombre d'itérations étant très dépendant de la structure de [K] et inconnu a priori.

Des méthodes semi-itératives se sont également développées. Elles consistent à accélérer une méthode directe en utilisant un processus itératif.

Nous présenterons quelques méthodes parmi les plus employées, en vue notamment d'utiliser deux de ces algorithmes (décomposition de Cholesky et ICCG) pour la définition d'algorithmes parallèles.

### 2. Méthodes directes

Ces méthodes sont des méthodes très classiques, connues et employées depuis très longtemps.

a) Méthode d'élimination de Gauss :

Cette méthode se compose de deux étapes :

* par des combinaisons linéaires d'équations on transforme le système $[K]\{U_n\} = \{F\}$ en un système triangulaire :

$$\begin{bmatrix} \diagdown & S \\ 0 & \diagdown \end{bmatrix} \quad \{U_n\} = \{F'\}$$

* résolution du système triangulaire, en partant de la dernière inconnue jusqu'à la première.

L'algorithme d'élimination est le suivant :

$S = 1, \ldots n-1$

$i = S + 1, \ldots n$

$c = K_{is}\ K_{ss}^{-1}$

$F_i = F_i - c\ F_s$

$y = S + 1, \ldots n$

$K_{ij} = K_{ij} - c\ K_{sj}$

la matrice triangulaire [S] et le nouveau second membre {F'} sont directement stockés dans [K] et {F}

La résolution se fait alors par :

$U_n = K_{nn}^{-1}\ F_n$

$i = n - 1, \ldots 1$

$U_i = K_{ii}^{-1}\ (F_i - \sum_{j=i+1}^{n} K_{ij}\ U_j)$

Cette méthode est très sensible aux valeurs $K_{ii}$ appelées pivots.

De plus elle génère un grand nombre d'éléments non nuls, elle est donc à éviter pour la résolution des systèmes "creux".

Le nombre d'opérations nécessaire est :

$\frac{1}{2}\ n(n+1)$ divisions

$\frac{1}{6}\ (n-1)\ n\ (2n+5)$ multiplications et autant de soustractions.

soit au total, en négligeant les soustractions d'une durée négligeable par rapport aux divisions et aux multiplications :

$\frac{1}{3}\ n\ (n^2-1) + n^2$ opérations.

Pour n grand, on a donc sensiblement $n^3/3$ opérations.

b) Méthodes de décomposition :

En fait, la méthode de Gauss décompose [K] sous la forme :

$$[K] = [L]\,[S]$$

où L est une matrice triangulaire inférieure (à termes diagonaux unités) et S une matrice triangulaire supérieure.

Il existe différentes formes de décomposition de [K] :

* forme de Doolittle

$$[K] = [L]\,[S]$$

* forme L D U

$$[K] = [L]\,[D]\,[U]$$

[D] matrice diagonale

[L] matrice triangulaire inférieure à termes diagonaux unités

[U] matrice triangulaire supérieure à termes diagonaux unités

* forme de Crout

$$[K] = [L]\,[D]\,[L]^T$$

ceci à condition que la matrice [K] soit symétrique.

* forme de Cholesky

$$[K] = [L]\,[L]^T$$

[K] est symétrique définie positive. [L] est une matrice triangulaire inférieure (les termes diagonaux sont positifs).

Nous nous intéresserons plus particulièrement à cette dernière méthode qui est une des méthodes de résolution utilisées dans le logiciel (FLUX 2D) sur lequel nous travaillons.

c) Décomposition de Cholesky :

[K] est symétrique définie positive et l'on cherche [L] triangulaire inférieure telle que $[K] = [L]\,[L]^T$, on alors :

$$K_{ij} = \sum_{k=1}^{n} L_{ik} L_{jk} \quad \text{où } i \leq j$$

d'où $K_{11} = L_{11}^2 \rightarrow L_{11} = \pm \sqrt{K_{11}} \neq 0$

On choisit la détermination positive :

$$K_{ij} = L_{11} L_{1j} \rightarrow L_{j1} = \frac{K_{j1}}{L_{11}} \quad (j = 2,\ldots n)$$

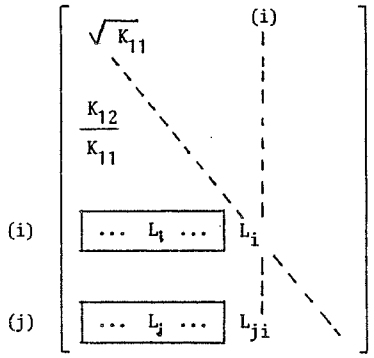
la $i^{ème}$ colonne sera déterminée par :

$$K_{ii} = \sum_{k=1}^{k=i-1} L_{ik}^2 + L_{ii}^2$$

d'où $$L_{ii} = K_{ii} - \sum_{k=1}^{k=i-1} L_{ik}^2$$

puis $$K_{ij} = \sum_{k=1}^{k=i-1} L_{ik} L_{ik} + L_{ii} + L_{ji}$$

d'où $$L_{ji} = \frac{1}{L_{ii}} [K_{ij} - \sum_{k=1}^{k=i-1} L_{ik} L_{jk}]$$




On remplace $K_{ii}$ par la racine carrée de $K_{ii}$ - Σ (carrés des éléments de $[L_i]$) :

$$L_{ii}^2 = K_{ii} - [L_i]\,[L]^T$$

Ensuite on détermine $L_{ji}$ par : $L_{ji} = [K_{ji} - [L_i]\,[L_j]^T] / L_{ii}$ $\quad (j > i)$

La seule condition pour appliquer cette méthode est que [K] soit symétrique définie positive.

Pour une dimension n du système assez grande, le nombre d'opérations à effectuer est de l'ordre de $n^3/6$ pour une matrice pleine. Si la matrice a une structure bande le largeur de bande w, le nombre d'opérations devient de l'ordre de $n w^2/6$.

## 3. Méthodes itératives

### a) Préconditionnement :

Les méthodes itératives sont très diverses, néanmoins on peut en dégager quelques principes fondamentaux.

Soit à résoudre le système [K] {U} = {F}, on notera désormais K u = f.
La matrice K est une matrice de grande dimension et contenant de nombreux termes nuls, on ne sait généralement pas calculer $K^{-1}$ de façon simple. Il s'agit alors de définir une matrice $M^{-1}$ qui soit une approximation de l'inverse de K.

On obtient alors le système préconditionné suivant :

$$M^{-1} K u = M^{-1} f$$

où M est la matrice de conditionnement telle que l'on puisse facilement calculer $M^{-1}$.
On peut alors définir la relation de récurrence :

$$u^{(n+1)} = r^{(n)} + \tau M^{-1} (f - K u^{(n)})$$

où  est un paramètre réel.

Si l'on écrit K = I - L - U où L et U sont des matrices triangulaires supérieures et inférieures, on peut constuire des méthodes qui choisissent

$$M = (I - \omega L) (I - \omega U)$$

Selon la valeur de $\tau$ et de $\omega$ on obtient les méthodes suivantes :

| $\omega$ | $\tau$ | M | Méthode Itérative |
|---|---|---|---|
| 0 | 1 | I | Jacobi |
| 0 | $\tau_o$ | I | Déplacements simultanés |
| $\omega$ | $\omega(2-\omega)$ | $(I-\omega L)(I-\omega U)$ | S.S.O.R. |
| $\omega$ | $\tau$ | $(I-\omega L)(I-\omega U)$ | Déplacements simultanés préconditionnés |

Avec d'autres formulations, on peut développer de nombreuses autres méthodes telles Gauss-Seidel, Gauss-Seidel Relaxé, Approximations Successives, Gradients Conjugués, Projections,...

b) Minimisation d'une fonction :

Pour des systèmes symétriques définis positifs, on peut montrer que résoudre

$$K\,u = f$$

est équivalent à minimiser la fonction $F(v) = \frac{1}{2}\, v^T K\, v - v^T f$.

On construit alors une suite :

$$u^{(i+1)} = u^{(i)} + \mu v$$

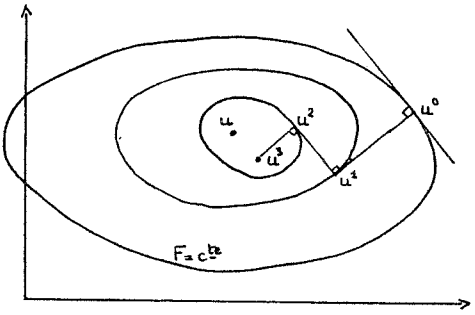
où $\mu$ est un réel et $v$ un vecteur qui sont choisis de manière à se rapprocher de la solution. On peut distinguer trois stratégies :

* la méthode des plus fortes pentes

On choisit $\mu$ et $v$ de manière à donner à partir de $u^{(i)}$ la plus forte décroissance à $F(u^{(i)})$. On peut montrer qu'il faut choisir :

$$v = r^{(i)} = f - K\, u^{(i)} \qquad \text{(résidu)}$$

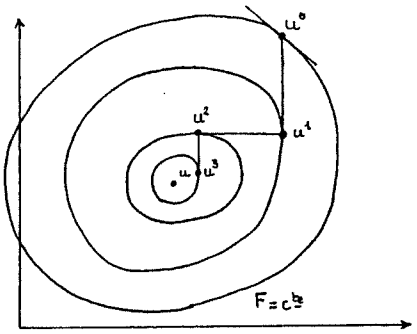
$$\mu = \frac{r^{T(i)}\, r^{(i)}}{r^{T(i)}\, K\, r^{(i)}}$$




* la méthode de relaxation

On choisit pour V un des vecteurs unitaires des axes de coordonnées $e^{(k)}$ en donnant à k successivement toutes les valeurs possibles. On a alors :

$$\mu = \frac{e^{T(k)}\, r^{(i)}}{e^{T(k)}\, K\, e^{(k)}}$$

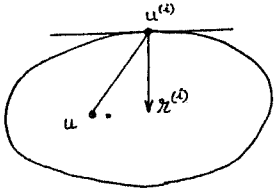



* Méthode des directions conjuguées

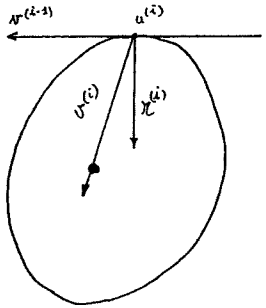
L'équation F(u) = constante représente une famille d'ellipsoïdes de même centre $u=K^{-1}f$. Soit un vecteur z tangeant en $u^{(i)}$ à un des ellipsoïdes, il est orthogonal au vecteur résidu $r^{(i)}$, d'où :

$$0 = z^T r^{(i)} = z^T (K u^{(i)} - f) = z^T K (u^{(i)} - K^{-1} f) = z^T K (u^{(i)} - u)$$

Le vecteur z est tangeant à l'ellipsoïde et $(u^{(i)} - u)$ passe par le centre, ce sont des directions conjuguées par rapport à l'ellipsoïde.




On construit alors l'intersection entre l'ellipsoïde précédent avec le plan contenant $r^{(i)}$ et $v^{(i-1)}$, ce qui donne une ellipse. Le vecteur $v^{(i)}$ est alors un vecteur issu de $u^{(i)}$ et dirigé vers le centre de cette éllipse :



On a alors la relation de conjugaison $v^{T(i)}\ K\ v^{(i-1)} = 0$.
Les itérations deviennent alors :

$$r^{(i)} = f - K\,u^{(i)}$$

$$\lambda_i = \frac{r^{T(i)}\ r^{(i)}}{r^{T(i-1)}\ r^{(i-1)}}$$

$$v^{(i)} = r^{(i)} + \lambda_i\ v^{(i-1)}$$

$$\mu_i = \frac{v^{T(i)}\ r^{(i)}}{v^{T(i)}\ K\ v^{(i)}}$$

$$u^{(i+1)} = u^{(i)} + \mu_i\ v^{(i)}$$

c) Gradient Conjugué avec préconditionnement incomplet de Cholesky (ICCG)

En appliquant des préconditionnements convenables à des méthodes itératives, on obtient très souvent de bonnes méthodes à convergence rapide. Nous étudierons particulièrement la méthode du Gradient Conjugué avec un préconditionnement incomplet de Cholesky (ICCG).

Pour résoudre $K\,u = f$ , on applique la méthode du Gradient conjugué au système :

$$(L^{-1}\ K\ (L^T)^{-1})\ L^T\ u = L^{-1}\ f$$

où $M = L\,L^T$ est inversible.

La matrice $L^{-1}\ K\ (L^T)^{-1}$ est symétrique définie positive.

L'algorithme du Gradient Conjugué s'écrit :

Gradient Conjugué :

$u^\circ = \ldots$ ;

$\tilde{f} = L^{-1}\ f$

$r^\circ = L^{-1}\ K\ (L^T)^{-1}\ u^\circ - \tilde{f}$

$d^\circ = -\ r^\circ$ ; $\alpha^\circ = \langle r^\circ, r^\circ \rangle$ ;

$n = 0$ ;

Boucle :

$n = n + 1$ ;

$\lambda^n = \alpha^{n-1}$ ; $\langle d^{n-1}, L^{-1} K (L^T)^{-1} d^{n-1} \rangle$

$u^n = u^{n-1} + \lambda^n d^{n-1}$

$r^n = L^{-1} K (L^T)^{-1} u^n - \tilde{f}$

$\beta^n = \langle r^n, r^n \rangle$ ;

Si $\beta^n \leq \varepsilon$ (précision) aller à Fin ;

$\gamma^n = \beta^n / \alpha^{n-1}$ ; $\alpha^n = \beta^n$ ;

$d^n = - r^n + \gamma^n d^{n-1}$ ;

aller à Boucle ;

Fin solution = $(L^{-1})^T u^n$

On peut mettre l'algorithme sous une forme mieux adaptée au calcul :

Gradient Conjugué-préconditionné :

vecteurs : u, s, r, d

matrice creuse : K

matrice creuse triangulaire : L

$u = u_0$ ; $s = K u - f$ ; $r = (L^T)^{-1} L^{-1} s$ ;

$d = - r$ ; $\alpha = \langle s, r \rangle$ ;

Boucle :

$r = K d$ ;

$\lambda = \alpha / \langle d, r \rangle$

$u = u + \lambda d$ ; $s = s + \lambda r$ ;

$r = (L^T)^{-1} L^{-1} s$ ;

$\beta = \langle r, s \rangle$ ;

Si $\beta \leq \varepsilon$ aller à Fin ;

$\gamma = \beta / \alpha$ ; $\alpha = \beta$ ;

$d = - r + \gamma d$ ;

aller à Boucle ;

Fin : solution = u

Pour éviter de faire apparaître des éléments non nuls lors de la décomposition de $M = L L^T$, on choisit de forcer la structure de L à être identique à celle de K. C'est-à-dire, pour calculer les éléments de L on applique la décomposition de Cholesky en ne calculant que les éléments de L se trouvant à des places telles que l'élément correspondant de K était non nul.

Pour que le Gradient Conjugué puisse être appliqué, il faut que les termes diagonaux de L soient strictement positifs ($\forall_i$, $L_{ii} > 0$). On peut montrer que si la matrice K est une M-Matrice ($M_{ij} \leq 0$ si $i \neq j$), alors la décomposition incomplète de Cholesky donne toujours $L_{ii} > 0$.

## III . PARALLELISATION D'UNE METHODE DIRECTE

Nous avons choisi de définir des algorithmes parallèles pour la méthode de Cholesky qui est une des méthodes déjà implantées sur le logiciel FLUX 2D et qui pourra nous fournir de bons éléments de référence.

### 1. Algorithmes de résolution avec q processus

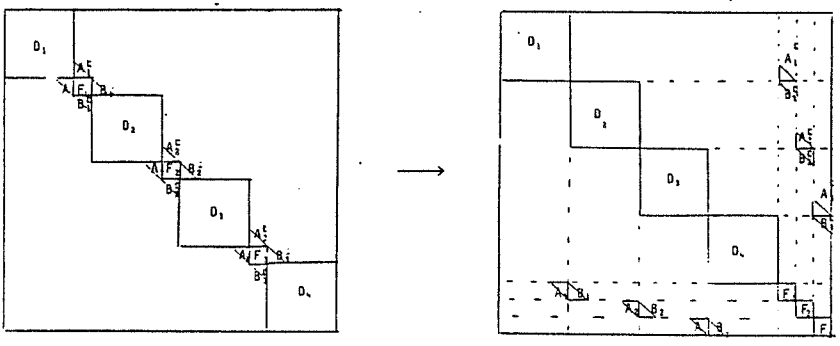
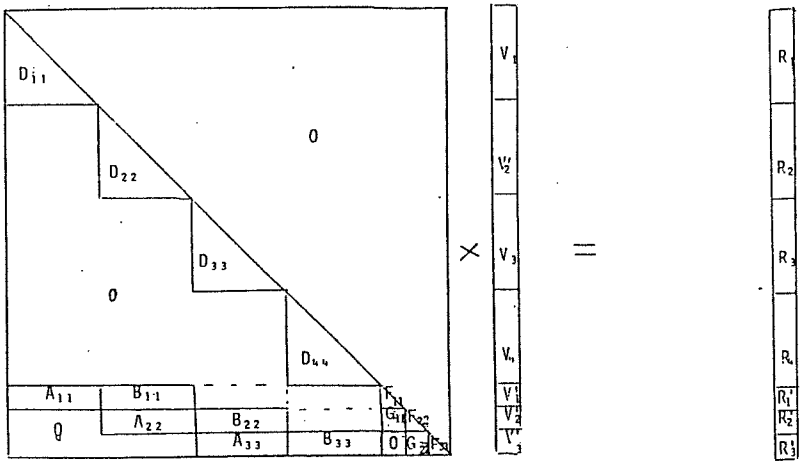


Figure 1

On cherche à résoudre le système $S_o X_o = R_o$ où $S_o$ est une matrice bande symétrique définie positive. Pour permettre le parallèlisme, nous effectuons un changement de base qui fera apparaître des blocs indépendants, la matrice $S_o$ donne la matrice S. Sur la Figure 1 nous donnons un exemple avec q = 4 blocs.

Par l'algorithme de Cholesky nous décomposons S : ($S = L L^T$) où L est une matrice triangulaire inférieure qui a le même profil que S. La résolution se fait alors en deux étapes : calcul de V tel que $L V = R$ puis calcul de X tel que $L^T X = V$. (voir Figure 2.).

L V = R




$L^t$ X = V

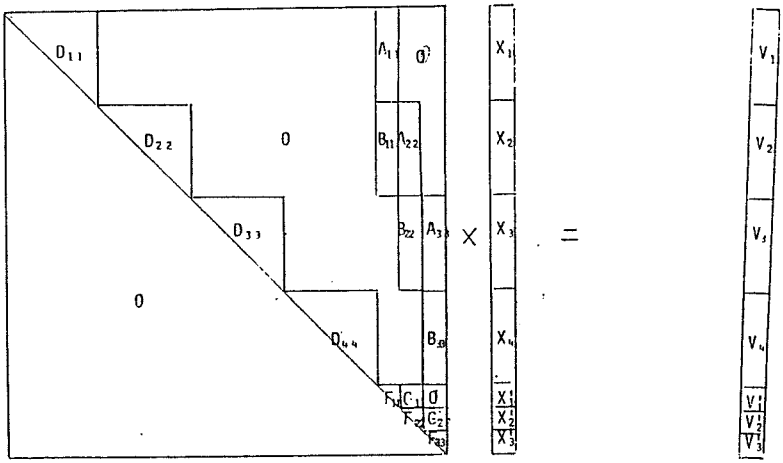



Figure 2

## 2. Détails des calculs

Donnons les calculs à effectuer et leur répartition entre les q processus.

1) décomposition de Choleski:

$$D_i = D_{ii}.D_{ii}^t \qquad (i=1,q)$$

$$A_i = A_{ii}.D_{ii}^t \qquad (i=1,q-1)$$

$$B_i = B_{ii}.D_{i+1,i+1}^t \qquad (i=1,q-1)$$

$$F_1 = A_{11}A_{11}^t + B_{11}B_{11}^t + F_{11}F_{11}^t$$

$$0 = A_{i+1,i+1}B_{ii}^t + F_{ii}^tG_{ii} \qquad (i=1,q-2)$$

$$F_i = A_{ii}A_{ii}^t + B_{ii}B_{ii}^t + G_{i-1,i-1}G_{i-1,i-1}^t + F_{ii}F_{ii}^t \qquad (i=2,q-1)$$

Il faudra effectuer:

. Bloc 1 : Choleski sur $D_1$ suivi de $A_1^t$

. Bloc $i$ : Choleski sur $D_i$ suivi de $A_i^t$ . suivi de $B_{i-1}^t$

. Bloc $q$ : Choleski sur $D_q$ suivi de $B_{q-1}^t$

.Puis Choleski sur $E_1 = F_1 - A_{11}A_{11}^t - B_{11}B_{11}^t$ et sur

$$E_i = F_i - A_{ii}A_{ii}^t - B_{ii}B_{ii}^t - G_{i-1,i-1}G_{i-1,i-1}^t \qquad (i=2,q-1)$$

2)Résolution:

. Calcul de $V_i$ tel que $D_{ii}.V_i = R_i \qquad (i=1,q)$

.Calcul de $V'_1$ tel que $A_{11}V_1 + B_{11}V_2 + F_{11}V'_1 = R'_1$ et de $V'_i$ tel que: $A_{ii}V_i + B_{ii}V_{i+1} + G_{i-1,i-1}V'_{i-1} + F_{ii}V'_i = R'_i$ $(i=2,q-$

.Calcul de $X'_{q-1}$ tel que $F_{q-1,q-1}X'_{q-1} = V'_{q-1}$

.Calcul de $X'_i$ tel que $F_{ii}X'_i + G_{ii}X'_{i+1} = V'_i \qquad (i=q-2,1)$

. Puis calcul de :

$X_1$ tel que $D_{11}.X_1 + A_{11}.X'_1 = V'_1$

$X_i$ tel que $D_{ii}.X_i + B_{i-1,i-1}.X'_{i-1} + A_{ii}.X'_i = V_i \qquad (i=2,q-1)$

$X_q$ tel que $D_{qq}.X_q + B_{q-1,q-1}.X'_{q-1} = V_q$

### 3. Algorithmes pour un processus

Détaillons par exemple ce que fait le processus $i$

① Lecture de $D_i$ , $A_i^t$ , $B_{i-1}^t$
Choleski sur $D_i + A_i^t + B_{i-1}^t$ donne $D_{ii}$, $A_{ii}$, $B_{i-1,i-1}$
Envoi d'un signal vers $i-1$ "$B_{i-1,i-1}^t$ calculé"

② Calcul de $V_i$
Envoi d'un signal vers $i-1$ "$V_i$ calculé"

③ Calcul de $R'_i = A_{ii}V_i$
Lecture de $F_i$
Calcul de $E_i = F_i - A_{ii}A_{ii}^t$

Attente du calcul de $B_{ii}$ fait par $i+1$

④ Calcul de $E_i = E_i - B_{ii}B_{ii}^t$

Attente du calcul de $F_{i-1,i-1}$ fait par $i-1$

⑤ Calcul de $G_{i-1,i-1}$
Calcul de $E_i = E_i - G_{i-1,i-1}G_{i-1,i-1}^t$
Choleski sur $E_i$ donne $F_{ii}$
Envoi d'un signal vers $i+1$ "$F_{ii}$ calculé"

Attente du calcul de $V_{i+1}$ fait par $i+1$

Attente du calcul de $V'_{i-1}$ fait par $i-1$

⑥ Calcul de $V'_i$
Envoi d'un signal vers $i+1$ "$V'_i$ calculé"

Attente du calcul de $X'_{i+1}$ fait par $i+1$

⑦ Calcul de $X'_i$
Envoi d'un signal vers $i-1$ "$X'_i$ calculé"

Envoi d'un signal vers $i+1$ "$X'_i$ calculé"

⑧ Calcul de $V_i - A_{ii}^t.X'_i$

Attente du calcul de $X'_{i-1}$ fait par $i-1$

⑨ Calcul de $X_i$

Envoi d'un signal à 1 "Fin d'éxécution".

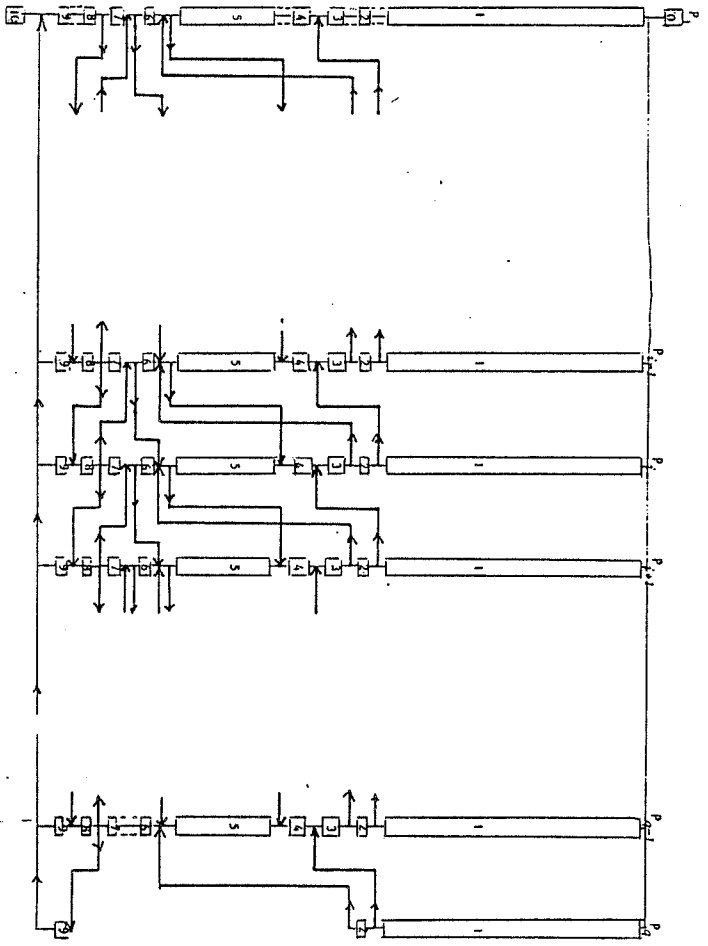


Figure 3. : Schéma des synchronisations.

4. Remarques

Ces calculs s'organisent suivant l'organigramme de la Figure 3.
Les blocs de la colonne i représentent les calculs faits par le processus i.
Les flèches partant de la colonne i indiquent les messages envoyés par le processus i. Les flèches arrivant dans la colonne i indiquent les points où le processus i attend un message envoyé par un autre processus.

Le processus 1 est analogue au processus i mais il faut naturellement supprimer les communications avec le processus de numéro inférieur. Par contre le processus q est différent, ceci provient du fait qu'il n'y a que q-1 blocs $F_i$. De ce fait, le cas où il n'y a que 2 processus est plus simple que les autres : il y a beaucoup moins de communications.

5. Conditions de validité de la méthode

On peut facilement montrer que si le système initial est symétrique défini positif alors la nouvelle matrice utilisée pour l'algorithme parallèle est également symétrique définie positive. En effet on passe de l'un à l'autre par un simple changement de base (en fait une permutation des vecteurs de base) ce qui n'affecte pas le caractère symétrique définie positive de la matrice.

Donc si l'on peut appliquer la décomposition de Cholesky au système linéaire initial on peut aussi lui appliquer l'algorithme parallèle. Il n'y a pas de conditions supplémentaires pour assurer la validité de l'algorithme.

## IV . PARALLELISATION D'UNE METHODE ITERATIVE

Nous donnerons la parallèlisation de la résolution du système linéaire par la méthode de Gradient Conjugué avec Préconditionnement Incomplet de Cholesky (ICCG)

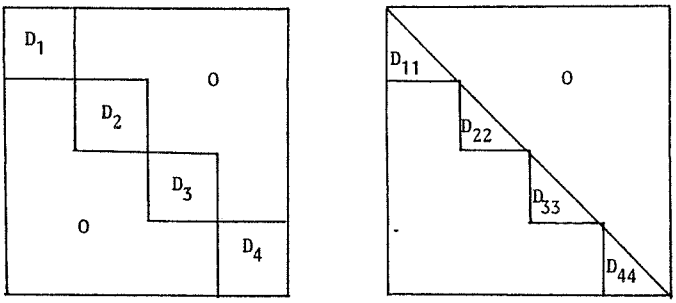
1. Modification du préconditionnement

Pour résoudre le système

$$A\,x = b$$

on utilise une matrice de préconditionnement $M = L\,L^T$, telle que L soit obtenue par décomposition de Cholesky de A mais en forçant à zéro les termes $L_{ij}$ tels que $A_{ij}$ est nul.

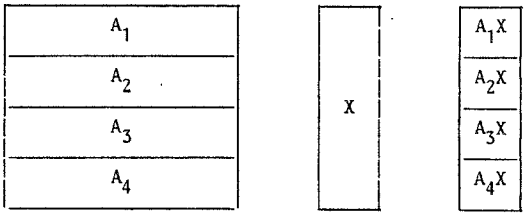
Pour l'algorithme parallèle, nous forçons à zéro des termes supplémentaires dans L. Nous définissons des blocs diagonaux $D_i$ dans la matrice M. Tout terme en dehors de ces blocs est forcé à zéro. Sur la Figure 1 nous donnons un exemple pour q = 4 processus donc 4 blocs $D_i$.




Matrice M  Matrice L

Figure 1

La décomposition incomplète peut alors être effectuée indépendamment sur chacun des blocs $D_i$.
Dans l'algorithme d'ICCG, on trouve un ensemble de produits scalaires qui sont faciles à paralléliser et des produits matrice A par vecteur. Pour paralléliser ce dernier type de produit nous adoptons la notation de la Figure 2.




A  X  V = AX

Figure 2.

## 2. Algorithme parallèle

En utilisant les notations précédentes, on peut reprendre l'algorithme d'ICCG qui devient pour le processus i :

(1)

Lecture de $A_i$ et $D_i$

Décomposition incomplète de $D_i$ donne $D_{ii}$

Envoyer un signal vers le processus 1 : "décomposition achevée"

Attendre du processus 1 le signal "toutes les décompositions achevées" sinon arrêter

$u_i = u_i^o$ (initialisation)

$S_i = b_i - A_i\, u_o$

(2) $r^i = (D_{ii}^T)^{-1}\, D_{ii}^{-1}\, S_i$

$d_i = r_i$

$\alpha_i = \langle s_i, r_i \rangle$

Envoyer au processus 1 le signal "$\alpha_i$ calculé"

Le processus 1 calcule $\alpha = \Sigma\, \alpha_i$

Attendre du processus 1 le signal "$\alpha$ calculé"

Boucle

(3)

$r_i = A_i\, d$

$\delta_i = \langle d_i, r_i \rangle$

Envoyer au processus 1 le signal "$\alpha_i$ calculé"

Le processus 1 calcule $\delta = \Sigma\, \delta_i$ et $\lambda = \alpha/\delta$

Attendre du processus 1 le signal "$\lambda$ calculé"

(4)

$u_i = u_i + \lambda\, d_i$

$s_i = s_i - \lambda\, r_i$

$\varepsilon_i = \| s_i \|^2$

Envoyer au processus 1 le signal " $\varepsilon_i$ calculé"

Le processus 1 calcule $\varepsilon = \sqrt{\Sigma\, \varepsilon_i}$ , si $\varepsilon <$ précision demandée alors ARRET

Attendre du processus 1 le signal "précision non atteinte"

(5) $r_i = (D_{ii}^T)^{-1} D_{ii}^{-1} s_i$

$\beta_i = \langle s_i, r_i \rangle$

Envoyer au processus 1 le signal "$\beta_i$ calculé"

Le processus 1 calcule $\beta = \Sigma \beta_i$ ainsi que $\gamma = \beta / \alpha$ et il fait $\alpha = \beta$

Attendre du processus 1 le signal "$\beta$, $\gamma$ et $\alpha$ calculés"

(6) $d_i = r_i + \gamma d_i$

Envoyer au processus 1 le signal "$d_i$ calculé"

Attendre du processus 1 le signal "d calculé"

Aller à Boucle.

Cet algorithme conduit au schéma de représentation de la Figure 3.

## 3. Remarques

Tous les processus effectuent exactement le même travail sauf le processus 1 qui contrôle les autres et effectue quelques calculs supplémentaires. Ces calculs supplémentaires sont néanmoins très limités et ne disymétrisent pas l'ensemble des algorithmes parallèles.

## 4. Convergence de l'algorithme

Pour assurer le succès de la méthode il faut que la décomposition incomplète de Cholesky de la matrice M soit telle que tous les termes diagonaux de L soient strictement positifs.

On peut montrer facilement que forcer des éléments hors de la diagonale ($L_{ij}$, $i \neq j$) à zéro aura tendance à augmenter la valeur des termes diagonaux ($L_{ii}$) lors de la décomposition. Donc si la matrice du système initiale donnait un préconditionnement acceptable, la nouvelle matrice convient aussi.

Donc les algorithmes parallèles ne nécessitent pas de conditions supplémentaires pour assurer leur convergence, par contre le nombre d'itérations peut varier suivant la méthode employée.

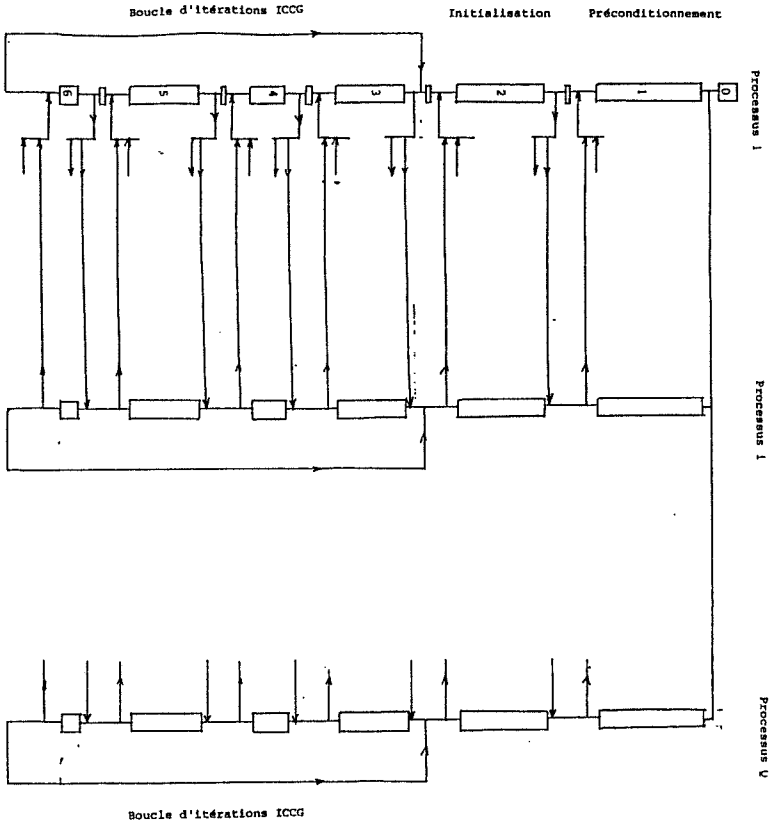



Figure 3 . Schéma de répartition.

# CHAPITRE

# 3

## CHAPITRE III . METHODES D'EVALUATION DES PERFORMANCES D'ALGORITHMES PARALLELES PAR SIMULATION

INTRODUCTION

A . DEFINITION DES PARAMETRES D'EVALUATION DES PERFORMANCES

I . REPRESENTATION DES ALGORITHMES DES PERFORMANCES

1. Structure d'un programme parallèle
2. Représentation

II . PARAMETRES DU PARALLELISME

1. Mises en évidence de temps d'attente
2. Paramètres
3. Quelques relations
4. Signification des paramètres

B . REALISATION ET SIMULATION SUR LE SYSTEME MULTICS (HB-68)

INTRODUCTION

I . FONCTIONNEMENT DU PARALLELISME SUR LE SYSTEME MULTICS

1. Caractéristiques générales
2. Langage de programmation
3. Lancement des programmes

II . STRUCTURE DE CONTROLE

1. Représentation de la structure de contrôle
2. Réalisation de la structure de contrôle sous le système Multics

III . REMARQUE GENERALE

C . MESURES COMPARATIVES - ETUDE DES TEMPS DE CALCUL

I . METHODE DE MESURE DES TEMPS DE CALCUL

1. Définition des paramètres d'état d'un algorithme
2. Mesure des temps de calcul

II . APPROXIMATION DES TEMPS DE CALCUL

1. Interpolation polynomiale des temps de calcul
2. Détermination des coefficients de l'interpolation

III . PERFORMANCES EN TEMPS DE CALCUL DE L'ENSEMBLE DU PROGRAMME PARALLELE

D . ETUDE DES TEMPS DE REPONSE - PRINCIPE DE MODELISATION

I . ETUDE DES TEMPS DE REPONSE

1. Définitions
2. Position du problème
3. Nature des temps de réponse

II . CALCUL DES TEMPS DE REPONSE

1. Principe général
2. Utilisation de logiciels de résolution de systèmes de files d'attente.
3. Un exemple de modélisation.

# CHAPITRE III . METHODES D'EVALUATION DES PERFORMANCES D'ALGORITHMES PARALLELES PAR SIMULATION

## INTRODUCTION

Nous avons développé dans le Chapitre précédent, des méthodes autorisant le parallèlisme. Il nous faut alors connaître quelle sera l'efficacité de ces méthodes pour le but que nous recherchons et qui est d'obtenir des diminutions dans les temps d'exécution de logiciels d'éléments finis. Nous nous heurtons à ce niveau à deux difficultés majeures. D'une part, dans le cas de ces méthodes, quelques mesures isolées ne peuvent suffire à déterminer les performances de façon nette et précise : il faudra mener plusieurs séries de mesures et développer des outils mathématiques et informatiques pour les exploiter. D'autre part, nous ne disposons au sein de notre Laboratoire, d'aucun équipement informatique doté d'une architecture parallèle. C'est pourquoi nous proposons une simulation sur un système fonctionnant en temps partagé.

Nous nous intéresserons donc dans ce Chapitre aux trois points suivants : définir des paramètres représentatifs des performances d'algorithmes parallèles, simuler une architecture parallèle, prévoir l'interprétation des résultats de mesures. Pour ce dernier point, nous ne présenterons pas de résultats de mesures, mais nous indiquerons comment les interpréter. Les résultats seront donnés au Chapitre IV.

## A . DEFINITION DES PARAMETRES D'EVALUATION DES PERFORMANCES

### I . REPRESENTATION DES ALGORITHMES PARALLELES

#### 1. Structure d'un programme parallèle

Ce que nous appelons un programme parallèle est en fait un ensemble de programmes coopérant pour obtenir un résultat final commun. Nous nommerons processus l'unité relative au système informatique, qui permet d'exécuter chacun de ces programmes. De plus il est nécessaire de disposer de fonctions permettant l'échange d'informations entre processus. Ces fonctions seront développées dans un autre paragraphe, seule leur existence nous intéresse ici.

Nous définirons alors une tâche comme l'ensemble des instructions exécutables par un processus entre deux communications successives avec d'autres processus. Chaque processus est généralement composé de plusieurs tâches.

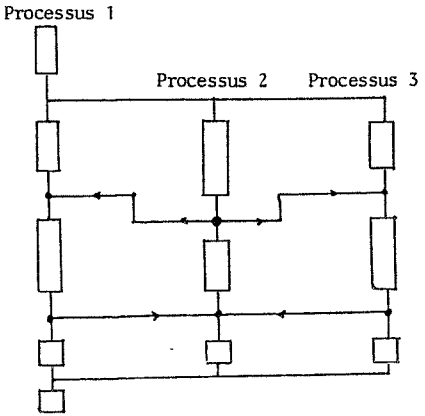
## 2. Représentation.

Nous adopterons une représentation sous forme de graphe avec les conventions suivantes. Chaque tâche est représentée par un rectangle vertical dont la longueur est proportionnelle à l'espérance du temps requis pour l'exécution de cette tâche, Cependant, les temps de calcul étant des variables aléatoires très peu dispersées, nous les supposerons constants pour la suite de l'exposé.

L'ensemble des tâches appartenant à un même processus sont rangées en colonnes dans l'ordre où elles doivent être exécutées. Elles sont reliées entre elles par un simple trait non fléché, étant donné que l'on admet que l'ordre naturel de parcours du graphe est du haut vers le bas.

Les communications entre processus sont représentées par des traits reliant le processus émetteur au processus receveur. Dans les algorithmes que nous avons décrits auparavant, les communications nécessaires entre processus se résument toujours à signaler qu'un ensemble de données a bien été calculé, mais il n'y a pas à proprement parlé transfert de blocs de données. Il s'agit seulement de donner des droits d'accès. Nous pourrons donc considérer que le temps d'exécution de ces communications est nul. Par contre, tant qu'une tâche n'a pas reçu toutes les communications qu'elle attend, elle ne peut commencer.

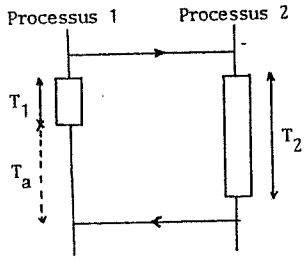
Voici un exemple de représentation d'algorithme parallèle :



## II . PARAMETRES DU PARALLELISME

### 1. Mise en évidence de temps d'attente

Une vision simpliste ou idéale, selon le choix, ferait dire que des programmes s'exécutant sur Q processus parallèles permettent de diviser les temps d'exécution par Q. Ceci ne serait possible que si les processus parallèles étaient composés de tâches identiques, nécessitant peu de synchronisation et que celles ci soient "bien placées", ce qui est loin d'être le cas général. Souvent apparaissent des distorsions dans la répartition des temps de calcul entre les différentes tâches, ce qui est générateur de temps d'attente comme le montre la Figure suivante.




Le temps d'attente généré par ce cas de figure est $T_a = \text{Max}\ (0,\ T_2 - T_1)$. Ce sont ces temps d'attente qui font perdre de l'efficacité aux programmes parallèles. C'est pour évaluer leur influence et connaître les réelles performances des algorithmes parallèles que l'on a introduit des paramètres particuliers.

### 2. Paramètres

Ces paramètres permettent de comparer les temps d'exécution de deux algorithmes. Dans le cas d'algorithmes parallèles, le programme de référence est en général le programme monoprocesseur d'où est issue la méthode parallèle. Nous noterons $T_{mono}$ le temps associé à ce programme. Nous noterons également $T_i$ le temps d'excécution du processus i du programme parallèle, c'est-à-dire la somme des temps d'exécution de chacune des tâches composant ce processus augmentée des temps d'attente dues aux synchronisations.

$t_{ij}$ = temps associé à la tâche j du processus i

$d_{ik}$ = temps d'attente du processus i dûe à la $k^{ième}$ synchronisation

$$T_i = \sum_j t_{ij} + \sum_k d_{ik}$$

On posera en plus : $T_{multi} = \max(T_i)$

$T_{multi}$ sera le temps d'exécution du programme parallèle.

On définit alors :

- le gain G par :

$$G = \frac{T_{mono} - T_{multi}}{T_{mono}} = 1 - \frac{T_{multi}}{T_{mono}}$$

ou bien $G = \min(G_i)$

avec $$G_i = \frac{T_{mono} - T_i}{T_{mono}}$$

- l'accélération A par :

$$A = \frac{T_{mono}}{\max(T_i)} = \frac{T_{mono}}{T_{multi}}$$

- l'efficacité :

$$E = \frac{T_{mono}}{\sum_{i=1}^{i=Q} T_i}$$

3. Quelques relations

On peut facilement obtenir les relations suivantes, qui nous serons utiles pour la suite.

$$G = 1 - \frac{1}{A} = \frac{A-1}{A}$$

$$A = \frac{1}{1-G}$$

$$E = \frac{T_{mono}}{\sum_{i=1}^{Q} T_i} \geq \frac{T_{mono}}{Q \, Max(T_i)} = \frac{T_{mono}}{Q \, T_{multi}} = \frac{A}{Q}$$

G est généralement exprimé en pourcentage.

On sait de plus, que si l'on a Q processus en parallèle, l'accélération maximale sera Q :

$$A \leq Q$$

De même, l'efficacité est toujours inférieure ou égale à 1, d'où

$$\frac{A}{Q} \leq E \leq 1$$

4. Signification des paramètres

Le gain et l'accélération sont deux notions proches l'une de l'autre. On peut d'ailleurs trouver une relation simple entre elles. L'indication principale qu'elles fournissent porte sur la rapidité relative des programmes mono et multi-processeurs. Si notre seul but est de diminuer des temps d'exécution de programmes, nous devons chercher un gain ou une accélération maximale.

L'efficacité est une notion duale des précédentes. Elle indique si chaque processus est utilisé au mieux, c'est-à-dire si chacun ne passe pas trop de temps à attendre les informations de ses voisins. La multiplication du nombre de processus parallèles entraîne généralement la diminution de l'efficacité de l'algorithme alors qu'elle tend souvent à accroître le gain ou l'accélération qu'on peut en obtenir.

De bons programmes parallèles doivent permettrent de trouver un compromis entre une accélération maximale recherchée et une efficacité pas trop dégradée.

## B . REALISATION ET SIMULATION SUR LE SYSTEM MULTICS (HB.68)

### INTRODUCTION

Ne disposant pas d'une architecture parallèle, nous avons simulé les algorithmes parallèles sur un système fonctionnant en temps partagé : Multics.
L'ordinateur HB.68 qui supporte ce système est doté de trois unités centrales. Cependant on ne peut pas attribuer l'exécution d'un programme à l'une particulière des unités centrales. Pour chaque utilisateur, tout se passe comme s'il n'y avait qu'une seule unité centrale approximativement trois fois plus rapide. On ne peut donc pas utiliser les possibilités de parallèlisme de ce matériel informatique de façon directe. Par contre le système Multics (logiciel gérant l'ordinateur) permet de simuler des exécutions parallèles en utilisant la multiprogrammation, c'est ce que nous allons décrire.

Note importante :

Dans le paragraphe suivant, nous introduisons la notion de "process". Nous avons voulu garder la terminologie du système Multics, mais il s'agit en fait de ce qu'ailleurs nous appelons "processus".

## I . FONCTIONNEMENT DU PARALLELISME SUR LE SYSTEME MULTICS

### 1. Caractéristiques générales

On dispose, sur cet ordinateur, de trois unités centrales travaillant simultanément pour 145 utilisateurs au maximum.
Le système Multics est essentiellement interactif, mais permet cependant à un utilisateur donné (même identité de facturation) l'existence simultanée de plusieurs environnements de travail indépendants appelés "process". Ces process simultanés travaillent soit en interactif, soit en "absentee" c'est-à-dire hors du contrôle de l'utilisateur.

Pour réaliser, sur les process travaillant en absentee les fonctions de contrôle, l'utilisateur peut les faire exécuter de plusieurs façons, en les préparant à l'avance soit par le biais de fichiers de commandes, soit par l'intermédiaire d'un programme, ce qui est notre cas.

### 2. Langage de programmation

Les programmes réalisant le traitement parallèle sont écrits en PL/I qui est le langage système de Multics, car ils utilisent les fonctions systèmes suivantes :

- lancement d'un nouveau process à partir d'un programme,
- attente par un process d'informations venant de l'autre,
- réveil d'un process.

Cependant, les algorithmes de calcul sont tous programmés en FORTRAN et sont appelés à partir de programmes PL/I.

### 3. Lancement de programmes

Nous noterons $P_i$ le programme exécuté par le process i.
Le programme $P_1$ est lancé en interactif par l'utilisateur, mais il pourrait être activé également en absentee. Il s'exécute dans un process qui est son environnement propre, là où sont créées en particulier ses variables locales. C'est lui qui va mettre en oeuvre la commande de lancement de nouveaux process dans lesquels s'éxécuteront les programmes $P_i (i \geq 2)$.

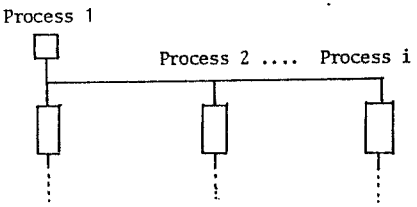
On peut remarquer tout de suite qu'un certain nombre de données (matrices, vecteurs, valeurs nodales, propriétés physiques, ...) doivent être accessibles par plusieurs programmes donc plusieurs process. Ceci forme l'ensemble des variables et données communes ou partagées. Tous ces tableaux sont stockés dans des segments (fichiers Multics) qui peuvent être gérés en FORTRAN comme des tableaux ordinaires.

## II . STRUCTURE DE CONTROLE

### 1. Représentation de la structure de contrôle

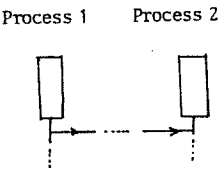
Les algorithmes parallèles se déroulent par exécution de sous-programmes appelés dans les différents programmes $P_i$. A différentes étapes du travail, $P_i$ doit savoir si un autre programme a terminé un algorithme donné, ou doit faire savoir qu'il a lui-même terminé un calcul, (donc que des données sont prêtes à être utilisées par un autre programme). Nous donnerons la représentation suivante des diverses tâches de contrôle, cette représentation est utilisée dans les graphes définis au paragraphe (A).

Génération de process




Le process 1, en début d'exécution, crée tous les autres process qui dès lors exécutent leur programme propre. Quand le process 1 a exécuté le sous programme de lancement des autres process, il poursuit son propre cours. La durée d'exécution de l'ordre de lancement est considérée comme nulle.

Envoi d'un message




Le process 1 envoie un message au process 2, puis poursuit l'exécution de son programme.
La durée de lancement d'un message est considérée comme nulle.

Réception d'un message

En reprenant le schéma précédent, le process 2 reçoit le message du process 1. Tant que le message n'a pas été reçu, l'exécution du programme du process 2 est interrompue. Ceci est souvent générateur de temps d'attente. Le temps de réception lui-même est considéré comme nul alors que l'attente peut être très importante et doit être prise en compte.

2. Réalisation de la structure de contrôle sous le système Multics

Pour réaliser cette structure, on dispose sous Multics d'un procédé de communication inter-process. Ce procédé est constitué de tables gérées par le système et de routines utilisables par programme permettant d'accéder à ces tables.
Les trois routines que nous avons principalement utilisées sont décrites ci-dessous.

a) ipc - S create - event - channel :

Cette routine permet à un process de créer dans les tables ad'hoc une zone dénommée "event - wait - channel", munie d'une identification propre. C'est dans cette zone qu'il recevra un signal venant d'un autre process. Un process, qui attend des signaux de plusieurs autres process, peut créer autant d'event - wait - channel que nécessaire.

b) hcs - S wake up :

Cette routine réalise le réveil d'un process. Les paramètres de cette routine sont : l'identification du process que l'on veut réveiller, et un message (numérique) que l'on veut transmettre.
C'est ce message qui permet de faire savoir si un algorithme s'est terminé normalement ou non, et qui permet au programme réveillé d'enchaîner ses algorithmes ou au contraire de se terminer en erreur.

c) ipc - S block :

Cette routine réalise le blocage (attente) d'un process. Les paramètres de cette routine sont : l'identification du ou des event - wait - channel sur lesquels un process attend d'être réveillé et l'identification d'une zone où il trouvera en particulier l'indication du process qui l'aura réveillé (dans le cas où il attend un réveil de plusieurs process) et le message de celui-ci.
Si un réveil survient avant que le process concerné soit bloqué, ce dernier sera immédiatement réveillé dès qu'il se mettra en blocage.

On constate que pour ces deux dernières routines, chaque process doit connaître les identificateurs des process avec lesquels il communique, et ceux de leurs event - wait - channel.

## III . REMARQUE GENERALE

Il faut bien noter que l'utilisation de ce système permet de simuler le parallèlisme sur un ensemble fonctionnant en temps partagé. Cette simulation aura le double avantage de nous permettre de valider nos programmes parallèles et d'étudier leur comportement en effectuant des séries de mesures.

# C . MESURES COMPARATIVES - ETUDE DES TEMPS DE CALCUL

## I . METHODE DE MESURE DES TEMPS DE CALCUL

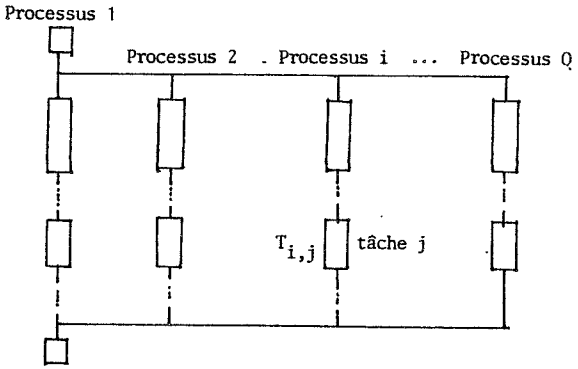
### 1. Définition des paramètres d'état d'un algorithme

Nous appelons paramètres d'état l'ensemble des grandeurs dont dépend le temps de calcul nécessaire à l'exécution d'un programme. La connaissance de tous les paramètres d'état permet de prédire le temps de calcul d'un programme.

Ces paramètres sont des caractéristiques des programmes. Par exemple si un programme effectue l'addition de deux vecteurs, le paramètre d'état sera la longueur du vecteur. Dans le cas de résolution de système linéaire, la dimension du système est le paramètre principal.

2. Mesure des temps de calcul

On considère un algorithme parallèle se déroulant sur Q processus.




On posera $T_{i,j}$ le temps de calcul nécessaire à l'exécution de la $j^{ème}$ tâche du processeur i. Pour chacun des $T_{i,j}$, c'est-à-dire pour toutes les tâches de tous les processus, on effectue des mesures de temps. Le nombre de mesures effectuées à partir d'exemples différents, dépend du nombre de paramètres d'état de l'algorithme ainsi que du choix des fonctions d'interpolation qui constituent l'approximation entre $T_{i,j}$ et les paramètres d'état.

## II . APPROXIMATION DES TEMPS DE CALCUL

### 1. Interpolation polynomiale des temps de calcul

Supposons que pour un algorithme, il existe p paramètres d'état $x_1$, $x_2$,...$x_p$. Le temps de calcul de cet algorithme est alors :

$$T = f(x_1, x_2, \ldots, x_p)$$

où f est une fonction a priori inconnue.

Nous choisissons de recherche une approximation polynomiale de f. L'approximation n'est pas quelconque. En particulier, on peut trouver l'ordre des différents monômes du polynôme d'approximation par des considérations mathématiques sur le nombre d'opérations à effectuer.

Ainsi, pour l'addition des deux vecteurs de longueur N, on obtient :

$$T = a_1 N + a_2$$

Pour la multiplication d'une matrice par un vecteur

$$T = a_1 N^2 + a_2 N + a_3$$

De façon générale :

$$T = \sum_{i_1,i_2,\ldots,i_p} a_{i_1,i_2,\ldots,i_p} \; x_1^{i_1} \; x_2^{i_2} \ldots x_p^{i_p} \qquad \text{(formule II.1)}$$

Les mesures définies au paragraphe précédent ont pour but de déterminer les coefficients $a_{i_1,i_2,\ldots i_p}$ pour toutes les tâches de chaque processeur.

## 2. Détermination des coefficients de l'interpolation

On utilise la mathode des moindres carrés pour déterminer les coefficients recherchés.

On fait n mesures pour des valeurs diverses de $x_1$, $x_2$,... $x_p$.
Les résultats des mesures sont stockés dans des vecteurs :

$$X_1 \begin{Bmatrix} x_{1,1} \\ x_{1,2} \\ \vdots \\ x_{1,n} \end{Bmatrix}, \; X_2 \begin{Bmatrix} x_{2,1} \\ x_{2,2} \\ \vdots \\ x_{2,n} \end{Bmatrix} \ldots X_p \begin{Bmatrix} x_{p,1} \\ x_{p,2} \\ \vdots \\ x_{p,n} \end{Bmatrix}, \; D \begin{Bmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{Bmatrix}$$

où $x_{i,j}$ est la valeur du $i^{ème}$ paramètre d'état à la $j^{ème}$ mesure et $d_j$ est la mesure du $j^{ème}$ temps de calcul.

Dans la formule II.1 nous supposerons que l'ensemble des combinaisons possibles de $i_1,i_2,\ldots i_p$ est égal à m, on pourra alors noter :

$$T = \sum_{j=1}^{j=m} a_j \; x_1^{I_1(j)} \; x_2^{I_2(j)} \ldots x_p^{I_p(j)}$$

avec $I_k(j) = i_k$

Nous posons alors $g_j = x_1^{I_1(j)} \; x_2^{I_2(j)} \ldots x_p^{I_p(j)}$

d'où $T = \sum_{j=1}^{j=m} a_j \; g_j$

de même nous poserons :

$$g_{i,j} = x_{1,i}^{I_1(j)} \; x_{2,i}^{I_2(j)} \ldots x_{p,i}^{I_p(j)}$$

Si l'on applique la méthode des moindres carrés, on aura :

$$d_i = \sum_{j=1}^{j=m} a_j \, g_{ij} \qquad (i \quad \{1,\ldots n\})$$

Ce qui s'écrit sous forme matricielle G A = D, où G est la matrice des $g_{ij}$ et A le vecteur des coefficients $a_j$ recherché.

A est un vecteur de dimension m, D de dimension n et G une matrice n x m à coefficients réels.

En général D n'appartient pas à l'image de G notée R(G) c'est-à-dire que D n'est pas une combinaison linéaire des colonnes de G. Alors plutôt que d'essayer de résoudre exactement m équations et pas du tout les autres, on cherche une solution vérifiant approximativement chaque équation, ce que l'on formule de la façon suivante :

Trouver $A \in R^m$ réalisant le minimum de $\|GA - D\|$

Le problème de moindres carrés a une interprétation géométrique claire : la fonctionnelle que l'on veut minimiser $\|GA - D\|$ représente la distance de D au point GA qui appartient au sous espace R(G). Dans ces conditions, il est clair que trouver la solution $\overline{A}$ revient à chercher le point $\overline{D} = G\overline{A} \in R(G)$ qui est le plus proche de D. L'intuition géométrique nous apprend (et l'on démontre) que D est la projection orthogonale de D sur R(G). Ainsi le vecteur $G\overline{A} - D$ doit être orthogonal à R(G), c'est-à-dire à tous les vecteurs de la forme G W, soit

$$(G\,W,\ G\overline{A} - D) = 0 \quad \forall\, W$$
$$(W,\ G^T\,(G\overline{A} - D)) = 0$$

donc la solution vérifie

$$G^T G \overline{A} = G^T D$$

Pour déterminer les coefficients de l'approximation, on effectue donc les opérations suivantes :

pour chaque processus
  pour chaque tâche de ce processus
    - faire des mesures donnant les vecteurs $X_1, X_2, \ldots X_p$ et D
    - construire la matrice symétrique $G^T G$ et le second membre $G^T D$
    - résoudre le système $G^T G A = G^T D$.

## III . PERFORMANCES EN TEMPS DE CALCUL DE L'ENSEMBLE DU PROGRAMME PARALLELE

Dans les définitions des paramètres mesurant les performances d'algorithmes parallèles, au paragraphe A, nous avons fait appel à la grandeur notée $T_{multi} = \max(T_i)$ où $T_i$ est le temps d'exécution associé à un processus. Il nous faut déterminer comment calculer les valeurs de $T_i$.

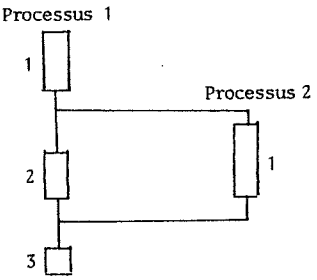
Dans un algorithme qui ne comporterait aucun point de blocage, donc aucune attente on aurait :

$$T_i = \sum_j T_{ij}$$

où $T_{i,j}$ est le temps de la $j^{ème}$ étape du processus i, calculé par l'approximation définie au paragraphe précédent.

Mais nous avons mis en évidence au paragraphe A.II.1 l'existence de points de synchronisation entraînant des temps d'attente que nous noterons $T_k^{(a)}$.

Exemple :




A l'issue de la deuxième étape, le processus 1 doit attendre la fin de l'étape 1 du processus 2, ce qui entraîne un temps d'attente :

$$T_1^{(a)} = \text{Max}\ (0,\ T_{1,2} - T_{2,1})$$

et l'on aura :

$$T_1 = T_{1,1} + T_{1,2} + T_{1,3} + T_1^{(a)}$$

De façon générale, nous écrivons :

$$T_i = \sum_j T_{ij} + \sum_k T_k^{(a)}$$

Ce sont ces temps de calcul modifiés des temps d'attente que nous utiliserons pour définir les performances de nos programmes parallèles.

# D . ETUDE DES TEMPS DE REPONSE - PRINCIPE DE MODELISATION

## I . ETUDE DES TEMPS DE REPONSE

### 1. Définitions

Nous donnerons ici deux définitions :

temps de calcul :
c'est le temps cumulé consommé par un processus lorsqu'il utilise une unité centrale de l'ordinateur lors de l'exécution d'un programme.

temps de réponse :
c'est le temps qui s'écoule entre l'instant où un utilisateur lance un programme et le moment où il reçoit la réponse qu'il attend.

### 2. Position du problème

On sait bien que l'unité centrale ne consacre pas tout son temps à du calcul. Elle possède un grand nombre de fonctions de contrôle du système, de contrôle des périphériques et de gestion interne. Donc, de façon générale, même dans un système monoprogrammé, le temps de réponse est supérieur au temps de calcul. Ceci est encore plus évident sur un système en temps partagé, où l'unité centrale est occupée tout à tour par différentes tâches lancées par plusieurs utilisateurs.

Nous savons mesurer les performances en temps de calcul d'algorithmes parallèles. Mais ce qui nous intéresse davantage est de connaître les performances en temps de réponse. Nous chercherons donc à établir des relations, si possible simples, entre les temps de calcul et les temps de réponse.

### 3. Nature des temps de réponse

Un programme étant déterminé et ses paramètres d'état fixés, on sait définir de façon déterministe la valeur, ou du moins une bonne approximation, du temps de calcul $T^{(C)}$ de ce programme.

Par contre, si on lance le même programme avec les mêmes paramètres à divers instants, on s'aperçoit que le temps de réponse $T^{(R)}$ varie.
En effet, des paramètres extérieurs au programme peuvent avoir une grande influence sur le temps de réponse. On peut citer, à titre d'exemple, la charge du système, c'est-à-dire le nombre d'interactions de l'ensemble des utilisateurs à un instant donné ; on peut également citer le problème des défauts de page.

Lorsqu'un programme arrive sur une ligne de code ou sur une donnée non présente en mémoire centrale, il y a lancement d'une lecture sur le disque de la page manquante, ce temps de lecture est variable.

Le temps de réponse d'un programme est une variable aléatoire. Ce que nous étudierons alors est la variation de la moyenne de cette variable aléatoire en fonction du temps de calcul et d'autres paramètres, notamment la charge du système. Nous essaierons de trouver une relation entre la moyenne du temps de réponse et le temps de calcul à charge constante.

Il est à noter que désormais ce que nous appellerons temps de réponse sera la moyenne de la variable aléatoire qu'est le temps de réponse.

## II . CALCUL DES TEMPS DE REPONSE

### 1. Principe général

Pour le calcul du temps de réponse on doit définir un modèle probabiliste du système informatique où sont implantés les algorithmes sous forme de réseau de files d'attente.

Un modèle à files d'attente simple est caractérisé par un ensemble de clients (processus) et de serveurs (unités centrales,...). Les serveurs sont soit en attente de travail, soit en cours de traitement d'un client. Un client ne peut être traité que par un serveur à la fois et un serveur ne peut traiter qu'un seul client à la fois.

On doit alors définir les grandeurs suivantes :

- le processus des arrivées, en particulier la loi de l'intervalle de temps séparant les instants de deux arrivées successives.
- la loi du service. Un client arrivant effectue une demande de service suivant une distribution qu'il faut connaître ; cette demande est exécutée par un serveur avec une certaine vitesse de traitement.
- le nombre de serveurs.
- la discipline dans la file, c'est-à-dire quand un serveur est disponible, quel est le client qu'il choisit.
- la population totale des clients pouvant accéder à la file est-elle finie ou non.
- la capacité de la file est-elle finie ou infinie.

Pour caractériser une file d'attente, on utilise souvent la notation suivante :

| A | / | B | / | m | / | F | / | K | / | N |
|---|---|---|---|---|---|---|---|---|---|---|
| loi des arrivées | | loi des services | | nombre de serveurs | | discipline dans la file | | nombre de places dans la file | | nombre total de clients |

Les lettres A et B qui caractérisent les lois d'arrivées et de service peuvent être par exemple :

M si la loi est exponentielle
G si la loi est quelconque
D si la durée est constante
H si la loi est hyper-exponentielle
E si la loi est une loi d'Erlang.

La discipline de la file représentée par F peut être par exemple :

FCFS : first come first served, encore improprement appelé FIFO first in first out : premier arrivé, premier servi.

LCFS : last come first served, dernier arrivé premier servi.

quantum : le service dure au maximum une durée $t_q$, ensuite il est interrompu et le client suivant, s'il y en a en attente, est servi. Le client dont le service a été interrompu est remis en dernier dans la file d'attente.

Processor sharing (PS) : c'est la limite du précédent lorsque $t_q$ tend vers 0.

Random : les clients sont choisis au hasard.

## 2. Utilisation de logiciel de résolution de systèmes de files d'attentes

Il s'avère souvent, face à la complexité des réseaux de file d'attente, très difficile de trouver des solutions analytiques. C'est pourquoi des logiciels de résolution ou de simulsation de ces systèmes ont été développés. Nous nous sommes servis du logiciel QNAP2.

QNAP2 peut être défini comme un système de description et de résolution des modèles de réseaux de file d'attente. Un langage particulier permet l'étude des modèles et le contrôle de leur résolution à l'aide de modules regroupant la plupart des techniques actuellement connues.

Le langage de QNAP2 permet à l'utilisateur de décrire les composants suivants :

- la configuration du réseau :

un réseau de files d'attentes est composé d'un ensemble de stations (avec un ou plusieurs serveurs par file) dans lesquelles circulent des clients suivant certaines règles. Il peut y avoir différentes classes de clients.

- le travail effectué par chaque station :

qui peut être décrit par sa loi de distribution ou par des algorithmes plus complexes.

- le contrôle de la résolution :

où l'on spécifie les initialisations, puis l'activation des séquences de contrôle et enfin l'édition des résultats.

Le langage de QNAP2 comprend deux niveaux :

- un langage de contrôle,
- un langage algorithme (dérivé de SIMULA et de PASCAL) qui complète les commandes du langage de contrôle.

Un programme QNAP2 est une séquence ordonnée de commandes permettant l'initialisation, la description du modèle, le contrôle de l'analyse et la solution.

Les outils de résolution que l'on trouve dans QNAP2 sont de trois types :

- simulation d'événements discrets,
- résolution analytique exacte,
- méthodes analytiques approchées.

Ces trois catégories fournissent les critères standards des performances (facteur d'utilisation, longueur moyenne des files, temps de réponse moyen de chaque file,...) qui caractérisent l'état stationnaire du réseau que l'on étudie.

QNAP2 a été défini initialement comme un outil d'analyse d'architectures complexes d'ordinateurs. Son champ d'application s'est élargi, et il peut être utilisé dans l'étude des systèmes de communication, de transport, les problèmes de gestions de stocks,...

## 3. Un exemple de modélisation

Dans le Chapitre V, nous développerons plus largement une modélisation. Nous donnerons ici un résumé rapide d'un travail de modélisation du système Multics dont le but était de trouver une relation entre le temps de calcul et le temps de réponse d'algorithmes parallèles.

Pour obtenir la loi du temps de réponse en fonction du temps de calcul, nous avons fait deux sortes de tests :

- des mesures directes utilisant des programmes de résolution de système linéaire dans le cas biprocessus, pour différentes tailles de matrices ;

- une modélisation de Multics à l'aide du logiciel de modélisation QNAP2, en utilisant pour déterminer les paramètres du modèle un outil de mesure qui existe sur Multics : METERING.

Nous avons choisi de caractériser la charge par le produit du nombre d'interactions créées par unité de temps, par le temps UC moyen demandé par une interaction.

On peut lire sur la Figure (A) les résultats obtenus par quelques mesures directes. Les mesures correspondent à trois valeurs distinctes de la charge. Pour chacune d'elle on peut lire 6 points correspondant à 6 tailles de matrice et repérés par le même signe. Il apparaît déjà qu'à charge donnée, le temps de réponse est à peu près proportionnel au temps de calcul.

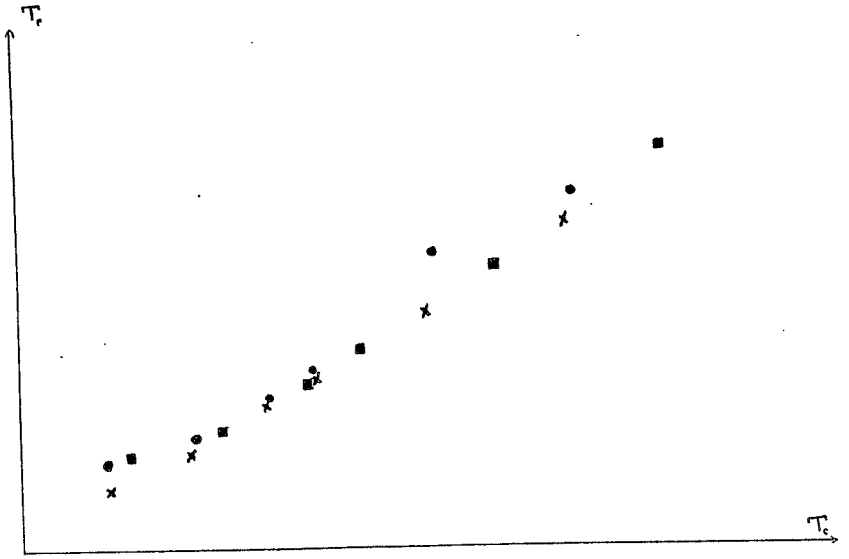



Figure A . Mesures directes.

La modélisation de Multics a donné le même résultat. Le modèle QNAP2 utilisé est représenté sur la Figure (B) avec les notations suivantes :

UC : file de ceux qui reçoivent le temps de calcul

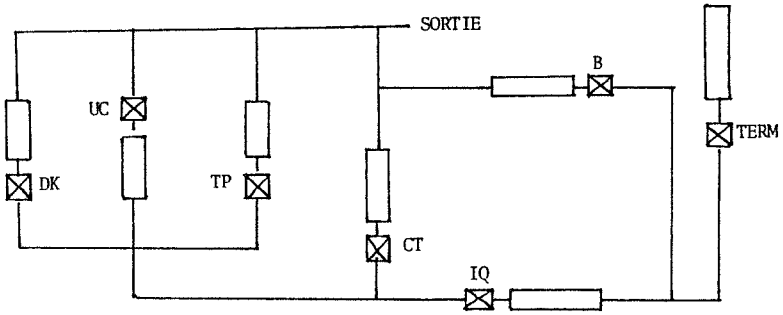
TERM : génération des interactions

B : blocage dû à une communication entre processus parallèles (cette file concerne les programmes particuliers que nous étudions)

IQ : file des interactifs qui sont prioritaires pour avoir accès à l'unité centrale.
Après leur création, les interactions sont placées dans la file des interactifs, après un premier quantum elles sont placées, si elles ne sont pas terminées, dans des files correspondant à leur classe de travail CT.

DK : attente de l'un des disques en cas de défaut de page

TP : attente de dévérouillage de la table des pages.




CONCLUSION

Cet exemple est intéressant car il amène à un résultat que nous retrouverons au Chapitre V : la moyenne du temps de réponse est proportionnelle au temps de calcul. Ce qui nous intéressera par la suite sera de modéliser des architectures parallèles monoprogrammées, c'est-à-dire où l'on génére Q processus lorsqu'il y a Q unités centrales. Nous ne désirons pas nous intéresser pour l'instant, au cas de systèmes en temps partagé.
Ceci aura pour conséquence que la variable aléatoire représentant le temps de réponse sera peu dispersée. A chaque exécution, la probabilité sera très forte que le temps de réponse de la mesure soit proche de la moyenne attendue.
Donc les performances mesurées à l'aide des temps de calcul restent assez bonnes pour les temps de réponse.

# CHAPITRE

# 4

## CHAPITRE IV . PERFORMANCES DES ALGORITHMES PARALLELES

INTRODUCTION

A . PERFORMANCES DES METHODES D'INTEGRATION

I . SIMPLIFICATION DES MESURES DANS LE CAS DE L'INTEGRATION

II . RESULTATS COMPARES DES DEUX STRATEGIES

1. Mesures
2. Commentaires

III . EVOLUTION DES PERFORMANCES EN FONCTION DU NOMBRE DE PROCESSUS

B . RESULTATS POUR L'ALGORITHME DE CHOLESKY

I . AMELIORATION DU PROGRAMME PARALLELE

1. Le programme de référence
2. Amélioration du programme multiprocesseur
3. Mesure de la valeur optimale de $N_1/N_2$ dans le cas biprocesseur

II . RESULTATS POUR LE CAS BIPROCESSEUR

1. Estimation des temps
2. Estimation approximative du gain dans le cas biprocesseur
3. Etude à la limite pour des grandes valeurs de N
4. Etude pour $\alpha$ = constante

III . RESULTATS POUR q PROCESSUS PARALLELES

1. Utilisation de la structure de graphe
2. Validation des mesures pour le cas biprocesseur
3. Variation des performances en fonction de q pour N grand
4. Variation des performances en fonction de N
5. Performances optimales

C . PERFORMANCES POUR ICCG PARALLELE

I . EVALUATION DES TEMPS DE CALCUL

1. Paramètres d'état
   a) Elimination du paramètre $\tau$
   b) Elimination du paramètre I
2. Interpolation des temps de calcul

II . RESULTATS DE PERFORMANCES

1. Etude en fonction de N
2. Performances en fonction de Q

D . CONCLUSION

# CHAPITRE IV . PERFORMANCES DES ALGORITHMES PARALLELES

## INTRODUCTION

Ce Chapitre est consacré à la présentation des résultats numériques donnant les performances des algorithmes parallèles définis au Chapitre II et à l'aide des méthodes d'évaluation données au Chapitre III.

Des résultats intermédiaires ont permis d'optimiser les algorithmes présentés ou de faire des choix lorsque nous avons proposé plusieurs stratégies de parallélisation.

Nous essaierons enfin d'analyser les résultats présentés pour en dégager l'intérêt que peut représenter le parallèlisme. Ces résultats seront utilisés au Chapitre V lors de l'étude d'une architecture parallèle particulière.

## A . PERFORMANCES DES METHODES D'INTEGRATION

### I . SIMPLIFICATION DES MESURES DANS LE CAS DE L'INTEGRATION

Comme nous l'avons montré précédemment, l'intégration consiste en la répétition de calculs au niveau de chaque élément, les calculs étant indépendants d'un élément à l'autre. Donc le nombre $N_e$ d'éléments à calculer est un paramètre déterminant pour caractériser un système.

On doit ensuite s'interroger sur la nature des calculs sur chacun des éléments. Ce temps de calcul dépend principalement de quatre caractéristiques :

1/ La forme de l'élément (triangle, quadilatère, ... en 2 Dimensions)
2/ L'ordre de l'interpolation sur l'élément (on utilise communément des interpolations du premier et du second ordre)
3/ La précision des calculs d'intégrales (en particulier le choix du nombre de points d'intégration).
4/ Les propriétés physiques spécifiques à chaque élément (notamment la différence entre des régions linéaires et des régions non linéaires).

Ces différentes caractéristiques sont difficilement quantifiables ce qui pose un problème pour l'évaluation des temps de calcul.

On peut en fait se passer de mesurer ces paramètres à condition d'émettre l'hypothèse suivante :

> lors de l'exécution des programmes parallèles on affecte à chaque processus un lot d'éléments tels que leur répartition (en fonction des caractéristiques précédentes) est la même que la répartition pour le problème total.

Si cette hypothèse est vérifiée, la nature des éléments à intégrer n'a plus d'influence sur l'évaluation des performances des algorithmes parallèles.

Nous avons proposé au Chapitre II, deux stratégies pour effectuer l'intégration en parallèle. Pour ces deux stratégies, l'hypothèse précédente sera statistiquement vérifiée, à condition que le nombre $N_e$ d'éléments à intégrer soit grand. Or ce doit être le cas, car l'utilisation du parallèlisme est utile pour l'étude de grands problèmes.

Nous considérons donc que le temps d'intégration d'un élément peut être désormais assimilé à une constante pour ce qui est de mesurer les performances des algorithmes parallèles. Les temps de calcul des différents processus seront donc mesurés par des relations linéaires à $N_e$. Cette remarque nous permettra de tirer des considérations très générales à partir de l'étude et de mesures sur quelques exemples.

## II . RESULTATS COMPARES DES DEUX STRATEGIES

### 1. Mesures

Nous choisissons un exemple test dont la description est donnée à la Figure 2.1. Le problème est découpé en 320 éléments (Figure 2.2).
Le programme normal demande le temps de calcul :

$$T_{mono} = 80,1 \text{ secondes}$$

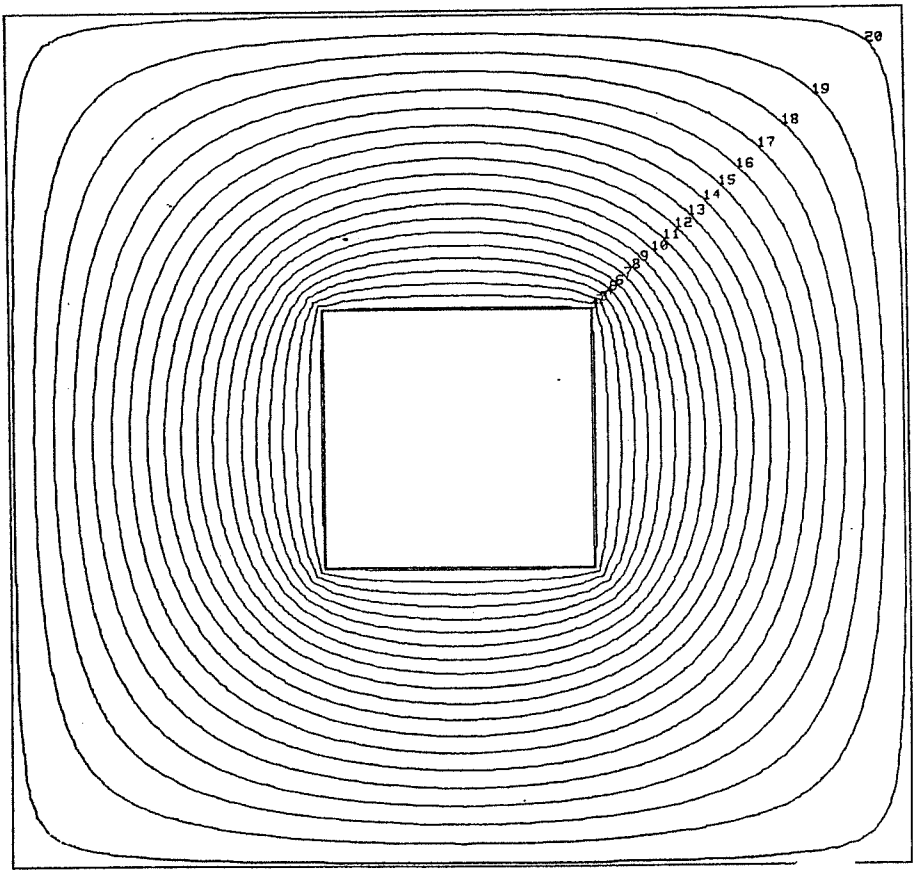



Figure 2.1 . Exemple test.

Condensateur carré

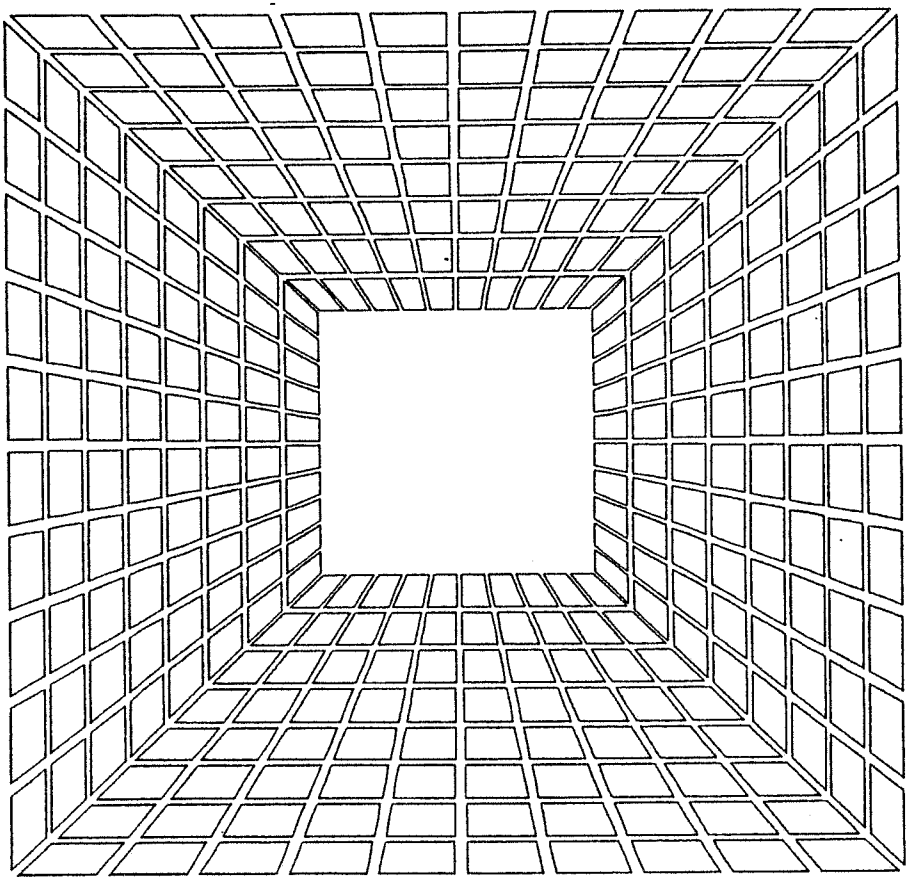


Figure 2.2 . Découpage en éléments finis
de l'exemple test.

Nous avons étudié les deux stratégies dans le cas où l'on exécutait les programmes parallèles avec 4 processus. Les résultats sont donnés aux Tableaux 2.3 et 2.4.

| | Nombre d'éléments calculés | Temps de calcul |
|---|---|---|
| Processus 1 | 69 | 18,58 |
| Processus 2 | 76 | 20,59 |
| Processus 3 | 97 | 24,92 |
| Processus 4 | 78 | 20,26 |

Tableau 2.3 . 1ère stratégie (Répartition aléatoire des éléments)

| | Nombre d'éléments calculés | Temps de calcul |
|---|---|---|
| Processus 1 | 80 | 22,41 |
| Processus 2 | 80 | 23,02 |
| Processus 3 | 80 | 22,23 |
| Processus 4 | 80 | 22,45 |

Tableau 2.4 . 2ème stratégie (Répartion déterministe des éléments)

La première méthode conduit aux performances suivantes :

$$\text{Gain} = 1 - \frac{24,92}{80,1} = 69\ \%$$

$$\text{Accélération} = \frac{80,1}{24,92} = 3,2$$

$$\text{Efficacité} = \frac{80,1}{(18,58 + 20,59 + 24,92 + 20,26)} = 0,94$$

La deuxième méthode :

Gain = 71 %

Accélération = 3,5

Efficacité = 0,88

2. Commentaires

Si l'on utilise les résultats précédents, qui sont mesurés en temps de calcul, on peut conclure que les deux méthodes sont équivalentes, puisqu'elles donnent pratiquement les mêmes performances.

Pour cette étude de l'intégration, il ne faut pas oublier que ce sont les performances en temps de réponse qui sont intéressantes pour notre étude.

Or les deux stratégies présentent une différence fondamentale. Les temps mesurés dans les tableaux précédents doivent être augmentés des temps d'attente générés par les synchronisations et accès aux données communes. Pour la seconde stratégie, les processus étant indépendants les uns des autres, les temps d'attente sont nuls, on peut garder les performances inchangées. Par contre, pour la première stratégie, à chaque fois qu'un processus a fini les calculs pour un élément il doit rechercher l'élément suivant et il risque de se produire des conflits entre processus à ce moment, ce qui génére des temps d'attente.

Les performances en temps de réponse de la première stratégie seront dégradées par rapport à la seconde. De plus, la première stratégie conviendrait mieux à un ensemble de processeurs fortement couplés. Comme nous nous orientons plutôt vers des applications à un réseau de processeurs faiblement couplés, nous avons retenu la seconde stratégie qui affecte à chaque processus un lot d'éléments connus a priori.

## III . EVOLUTION DES PERFORMANCES EN FONCTION DU NOMBRE DE PROCESSUS

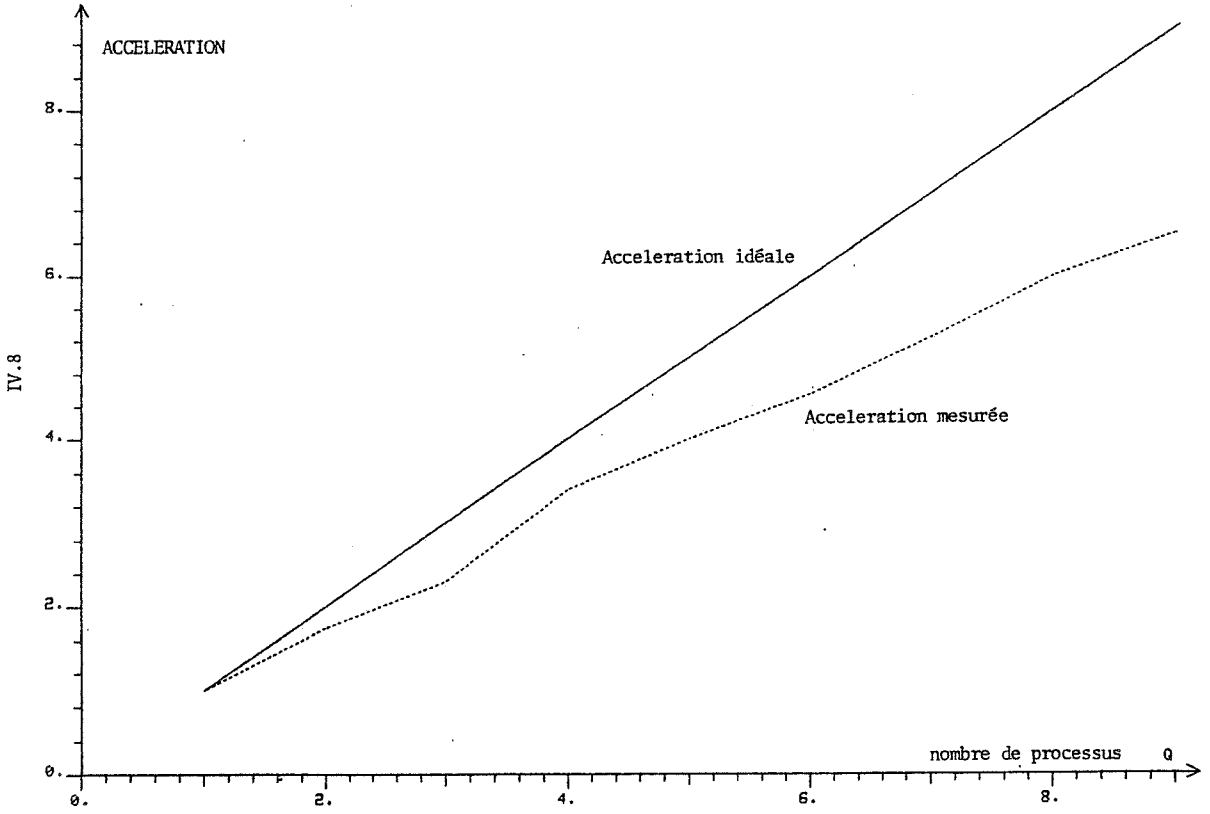
Nous étudions désormais les performances pour la seconde stratégie.
Pour quelques exemples de problèmes d'éléments finis, nous avons mené des intégrations avec un nombre variable de processus parallèles.
Les résultats obtenus sont donnés dans le Tableau 2.5.

La courbe 2.6 donne l'accélération que l'on peut obtenir pour l'intégration en fonction du nombre Q de processus parallèles. On peut voir que cette courbe est très proche de la courbe idéale : accélération = nombre de processus.

Ceci montre que l'intégration est bien adaptée au calcul parallèle.

| Nombre d'éléments | Nombre de processus | temps de calcul | Gain | Accélération |
|---|---|---|---|---|
| 144 | 1 | 10,17 | 0 % | 1 |
| | 2 | 5,53 | 46 % | 1,84 |
| | 4 | 2,97 | 71 % | 3,42 |
| | 8 | 1,70 | 83 % | 5,98 |
| | 16 | 1,01 | 90 % | 10,07 |
| 186 | 1 | 16,58 | 0 % | 1 |
| | 2 | 8,52 | 49 % | 1,95 |
| | 4 | 4,48 | 73 % | 3,70 |
| | 8 | 2,46 | 85 % | 6,74 |
| 200 | 1 | 20,76 | 0 | 1 |
| | 2 | 11,86 | 43 % | 1,75 |
| | 3 | 9,03 | 57 % | 2,3 |
| | 4 | 6,11 | 70 % | 3,4 |
| | 5 | 5,20 | 75 % | 4,0 |
| | 6 | 4,56 | 78 % | 4,55 |
| | 7 | 3,96 | 80 % | 5,25 |
| | 8 | 3,46 | 83 % | 6,0 |
| | 9 | 3,20 | 85 % | 6,5 |
| | 10 | 2,97 | 86 % | 7,0 |

Tableau 2.5




IV.8

Courbe 2.6

## B . RESULTATS POUR L'ALGORITHME DE CHOLESKY

Remarque préliminaire

Les notations de ce paragraphe sont celles utilisées au Chapitre II - § C.III.

### I . AMELIORATION DU PROGRAMME PARALLELE

#### 1. Le programme de référence

Le programme monoprocesseur qui sert de référence pour l'évaluation du gain est le programme implanté dans le logiciel FLUX et qui effectue d'une part la décomposition de Cholesky sur la matrice bande et d'autre part la résolution du système. Ce programme est considéré comme assez performant et constitue donc une bonne référence.

#### 2. Amélioration du programme multi-processeur

Plusieurs modifications du programme multi-processeur ont permis de diminuer son temps de réponse et donc d'en améliorer les performances.

L'amélioration principale du temps de réponse s'est faite au niveau de la répartition de la charge entre les différents processeurs. En effet, si l'on compte le nombre d'opérations effectuées par chaque partie du programme, on s'aperçoit que :

a) le temps consacré à la décomposition du système est bien supérieur au temps nécessaire à la résolution proprement dite,

b) si l'on choisit des dimensions égales pour $D_1$, $D_2$, $D_3$ ..., les temps de calculs pour les programmes $P_2$, $P_3$... sont les mêmes, par contre le temps de calcul de $P_1$ est tout à fait différent.
La décomposition du bloc $D_1A_1$ est beaucoup plus rapide que la décomposition des blocs $D_iA_iB_{i-1}$ pour $i \geq 2$. Ceci vient du fait que dans la décomposition de $D_1A_1$ on effectue un algorithme classique sur une matrice rectangulaire mais qui a gardé sa forme bande, alors que dans le second cas de la matrice rectangulaire n'est plus du tout bande, et au niveau nombre de calculs, tout se passe comme si l'on faisait deux Cholesky au lieu d'un seul. D'où l'idée d'avoir des blocs $D_1$ et $D_i$ de dimensions différentes : respectivement $N_1$ et $N_2$.

Calculons alors quel est le meilleur rapport $N_1/N_2$, soit m le temps d'une multiplication et s le temps d'une soustraction et soit W la demie largeur de bande. La décomposition de $D_1A_1$ nécessite :

$$t_1 = N_1 \left(\frac{W^2}{2} m + W s\right)$$

le calcul pour $D_2A_2B_1$ nécessite :

$$t_2 = 2N_2 \left(\frac{W^2}{2} m + W s\right)$$

Pour avoir $t_1 = t_2$ il faut donc que $N_1 = 2N_2$ c'est-à-dire.compte-tenu de :

$$N_1 + (q-1)\, N_2 = N - (q-1)W$$

$$N_2 = \frac{1}{q+1} (N - (q-1)W)$$

$$N_1 = \frac{2}{q+1} (N - (q-1)W)$$

En fait ces valeurs ne sont qu'approximatives, car d'une part, toutes les opérations ne sont pas ainsi décomptées, et d'autre part, en équilibrant Cholesly, on a déséquilibré la partie résolution de systèmes. Il s'agit donc de trouver expérimentalement les valeurs optimales de $N_1$ et $N_2$.

## 3. Mesure de la valeur optimale de $N_1/N_2$ dans le cas biprocesseur

Nous déterminerons cette valeur pour quelques problèmes, en donnant à $N_1$ et $N_2$ des valeurs simples.

Nous appellerons $T_o$ le temps du programme monoprocesseur, $T_1$ et $T_2$ ceux des programmes biprocesseurs, l'indice "c" pour le temps UC et l'indice "r" pour le temps de réponse. Les résultats correspondant à N = 227 et W = 21 sont donnés dans le Tableau 1. Ceux qui correspondent à N = 345 et W = 126 sont donnés dans le Tableau 2.

D'après les indications de ces tableaux, nous avons choisi :

$$N_1 = \frac{7}{12} (N - W) \quad ; \quad N_2 = \frac{5}{12} (N - W)$$

Ces valeurs sont empiriques, et en fait, pour avoir les meilleures performances il faudrait déterminer $N_1$ et $N_2$ de façon dynamique pour chaque valeur de N et W, mais cela nécessiterait un trop grand nombre d'essais alors que nous nous sommes aperçus que les valeurs ci-dessus citées sont assez bonnes.

| $N_1$ | $N_2$ | $T_{or}$ | $T_{oc}$ | $T_{1r}$ | $T_{1c}$ | $T_{2r}$ | $T_{2c}$ |
|---|---|---|---|---|---|---|---|
| $\frac{2}{3}$ | $\frac{1}{3}$ | 5,20 | 3,50 | 4,60 | 2,13 | 4,60 | 2,70 |
| $\frac{1}{2}$ | $\frac{1}{2}$ | - | - | 6,40 | 1,67 | 6,40 | 4,06 |
| $\frac{22}{23}$ | $\frac{1}{23}$ | - | - | 5,10 | 3,50 | 5,10 | 0,50 |
| $\frac{3}{4}$ | $\frac{1}{4}$ | - | - | 4,40 | 2,42 | 4,30 | 2,03 |
| $\frac{7}{12}$ | $\frac{5}{12}$ | - | - | 4,60 | 2,30 | 4,55 | 2,31 |

Tableau 1

| $N_1$ | $N_2$ | $T_{or}$ | $T_{oc}$ | $T_{1r}$ | $T_{1c}$ | $T_{2r}$ | $T_{2c}$ |
|---|---|---|---|---|---|---|---|
| $\frac{2}{3}$ | $\frac{1}{3}$ | 96,45 | 84,81 | 88,30 | 55,33 | 88,00 | 36,76 |
| $\frac{1}{2}$ | $\frac{1}{2}$ | - | - | 112,61 | 45,42 | 112,00 | 66,91 |
| $\frac{7}{24}$ | $\frac{17}{24}$ | - | - | 191,60 | 32,30 | 191,40 | 117,74 |
| $\frac{17}{24}$ | $\frac{7}{24}$ | - | - | 85,50 | 58,80 | 85,50 | 30,80 |
| $\frac{7}{12}$ | $\frac{5}{12}$ | - | - | 71,10 | 50,80 | 71,10 | 50,50 |

Tableau 2

## II . RESULTATS POUR LE CAS BIPROCESSUS

### 1. Estimation des temps

Nous voulons estimer les temps de calculs des programmes multiprocesseurs afin de les comparer aux temps correspondants pour les programmes monoprocesseurs.

Nous avons choisi d'estimer les temps de calcul $T_{ij}$ de chaque bloc j (voir schéma du Chapitre II § C.III), pour des calculs effectués par le processus i, par des polynômes de degré 3 par rapport à l'ensemble des variables $N_i$ et W, $N_i$ étant la taille du bloc $D_i$, dont la formule générale est donnée par :

$$T_{ij} = a_{1,i,j}\, N_i W^2 + a_{2,i,j}\, W^3 + a_{3,i,j}\, N_i^2\, W + a_{4,i,j}\, N_i^3 + a_{5,i,j}\, N_i^2$$

$$+ a_{6,i,j}\, N_i\, W + a_{7,i,j}\, W^2 + a_{8,i,j}\, N_i + a_{9,i,j}\, W + a_{10,i,j}$$

Pour déterminer les coefficients $a_{k,i,j}$ correspondant à chacun des temps $T_{i,j}$ nous avons effectué pour chacun de ces temps 50 mesures et appliqué une méthode des moindres carrés.

Comme nous l'avons mentionné plus haut, il y a une assez grande différence entre le cas de 2 process et le cas de plus de 2. Les synchronisations sont beaucoup moins nombreuses dans le cas de 2 process. (Il n'y a pas de bloc $G_i$, il n'y a pas de flèches qui remontent dans la Figure 3).

## 2. Estimation approximative du gain dans le cas biprocesseur

Dans un premier temps nous n'avons pas mesuré les temps de calcul dans chaque bloc, mais les temps totaux de chacun des process. L'estimation du gain que nous obtenons de ce fait, est évidemment approximative, mais nous verrons plus loin que les résultats sont en fait assez bons. Ceci provient du fait que lorsque les dimensions varient c'est soit l'un, soit l'autre des process qui a le plus de travail et c'est toujours le même, qui attend l'autre. C'est donc bien à peu près toujours le temps total de l'un ou l'autre qui intervient.

La méthode indiquée au § 1. a été appliquée pour estimer les temps de calcul pour $P_1$, $P_2$ et $P_{mono}$. Les résultats apparaîssent dans le Tableau 3.
Ces résultats nous ont amené à estimer que les $a_{3,j}$ et $a_{4,j}$ sont nuls. Des raisonnements sur le nombre d'opérations effectuées confirment cette propriété.
Nous estimons finalement les temps de calcul par les polynômes en N et $\alpha = \frac{W}{N}$ :

$$T_j = b_{1,j}N^3\alpha^2 + b_{2,j}N^3\alpha^3 + b_{3,j}N^2 + b_{4,j}N^2.\alpha + b_{6,j}N + b_{7,j}N + b_{8,j}$$

Les résultats apparaissent dans le Tableau 4.

| | $T_{mono}$ | $T_1$ | $T_2$ |
|---|---|---|---|
| $a_1$ | $0{,}198 \cdot 10^{-4}$ | $0{,}950 \cdot 10^{-5}$ | $0{,}444 \cdot 10^{-5}$ |
| $a_2$ | $-0{,}132 \cdot 10^{-4}$ | $-0{,}443 \cdot 10^{-5}$ | $-0{,}224 \cdot 10^{-4}$ |
| $a_3$ | $0{,}168 \cdot 10^{-6}$ | $0{,}113 \cdot 10^{-6}$ | $0{,}515 \cdot 10^{-5}$ |
| $a_4$ | $0{,}174 \cdot 10^{-7}$ | $-0{,}937 \cdot 10^{-8}$ | $-0{,}408 \cdot 10^{-6}$ |
| $a_5$ | $-0{,}130 \cdot 10^{-4}$ | $0{,}125 \cdot 10^{-4}$ | $0{,}199 \cdot 10^{-3}$ |
| $a_6$ | $-0{,}102 \cdot 10^{-4}$ | $0{,}649 \cdot 10^{-4}$ | $-0{,}103 \cdot 10^{-2}$ |
| $a_7$ | $-0{,}105 \cdot 10^{-3}$ | $0{,}148 \cdot 10^{-3}$ | $0{,}260 \cdot 10^{-2}$ |
| $a_8$ | $0{,}702 \cdot 10^{-2}$ | $-0{,}123 \cdot 10^{-2}$ | $-0{,}210 \cdot 10^{-1}$ |
| $a_9$ | $0{,}151 \cdot 10^{-1}$ | $-0{,}495 \cdot 10^{-2}$ | $-0{,}299 \cdot 10^{-1}$ |
| $a_{10}$ | -0,50 | 0,308 | 1.38 |

Tableau 3.

| | $T_{mono}$ | $T_1$ | $T_2$ |
|---|---|---|---|
| $10^5 . b_1$ | 2,06 | 0,995 | 2,52 |
| $10^5 . b_2$ | -1,46 | -0,512 | -5,38 |
| $10^5 . b_3$ | 1,13 | 0,899 | 6,23 |
| $10^5 . b_4$ | -1,43 | 7,88 | -41 |
| $10^5 . b_5$ | -4,48 | 17,1 | 370 |
| $10^5 . b_6$ | -158 | -88,4 | -1200 |
| $10^5 . b_7$ | 1480 | -842 | -18300 |
| $10^5 . b_8$ | 8110 | 36200 | 468000 |

Tableau 4.

3. Etude à la limite des grandes valeurs de N

Nous étudierons le cas où N→∞ , les gains deviennent égaux aux rapports des termes en $N^3$ et ne sont plus fonction que de α.

$$G_{1\infty} = \frac{1,065 - 0,948\alpha}{2,06 - 1,46\alpha} \qquad G_{2\infty} = \frac{3,92\alpha - 0,46}{2,06 - 1,46\alpha}$$

La Figure 4 montre que pour les très grandes matrices (N→∞), le programme biprocesseur est plus performant que le programme monoprocesseur dès que la largeur de bande n'est pas trop petite face aux dimensions du système (c'est-à-dire $W \geq \frac{N}{10}$ environ).

Pour $W \leq \frac{N}{10}$ il faudrait un algorithme qui décompose la matrice totale en un plus grand nombre de sous blocs.

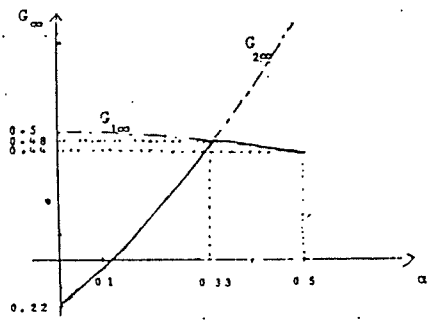



Figure 4.

Gain limite pour N grand.

Cas biprocesseur.

4. Etude pour $\alpha$ = constante

En gardant $\alpha$ constant, on fait varier N pour étudier G. Les résultats sont donnés sur la Figure 5.

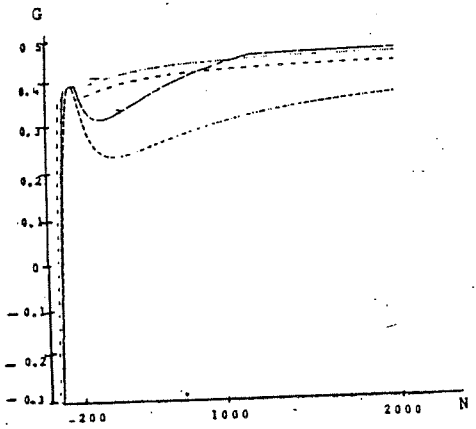



Figure 5.

$\alpha$ = 0.3
$\alpha$ = 0.35
$\alpha$ = 0.4
$\alpha$ = 0.5

Cette courbe nous montre que :

- le parallèlisme biprocesseur est performant pour des systèmes de taille supérieure à 100,
- pour tirer le meilleur parti de l'algorithme, il faut que $\alpha$ soit supérieur à 0.3,
- il existe une valeur optimale de $\alpha$, qui est comprise entre 0.3 et 0.4 dans l'étude à la limite on trouvait 0.313,
- pour des valeurs de $\alpha$ plus petites, c'est-à-dire rappelons le pour des largeurs de bande plus petites par rapport à la taille de la matrice il faut couper en un nombre plus grand de blocs et utiliser plus de processeurs.

## III . RESULTATS POUR q PROCESSUS PARALLELES

### 1. Utilisation de la structure de graphe

Pour calculer les performances de nos algorithmes, nous utilisons les mesures effectuées et les interprétons comme il est dit dans le Chapitre III.
A cette fin, nous avons exploité la structure de graphe (donnée au Chapitre II, paragraphe C.III) de cet algorithme et qui permet de tenir compte des temps d'attente qui peuvent apparaître lors des synchronisations.

### 2. Validation des mesures pour les cas biprocessus

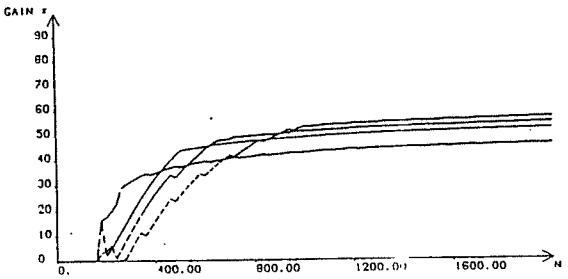



Figure 6. Variations du gain, cas biprocessus
$\alpha$ = 0.3, 0.35, 0.4 et 0.5

pour q = 2 :
Pour le gain limite pour de très grandes valeurs de N, on trouve exactement la même courbe que celle de la Figure 4. On voit sur la Figure 6 que les variations du gain en fonction de N sont très analogues à celles qu'on avait obtenues précédemment et qui étaient présentées sur la Figure 5.

Ce résultat montre que les phénomènes d'attente sont tout à fait marginaux dans le bas biprocessus, ce qui rend l'algorithme très efficace.

### 3. Variation des performances en fonction de q pour N grand

On suppose la dimension N du système très grande et l'on étudie les variations des performances en fonction de q et du paramètre $\alpha = W/N$. Les résultats sont données aux Figures 7 et 8 pour q variant de 3 à 9.

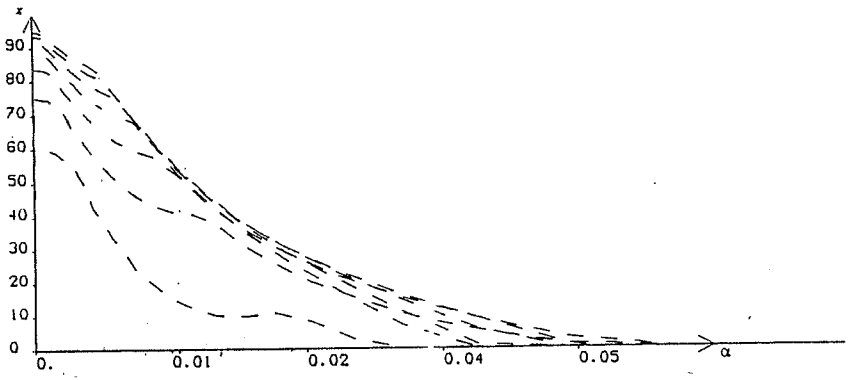



Figure 7. Gains limites pour N infini
q = 3 ... 9

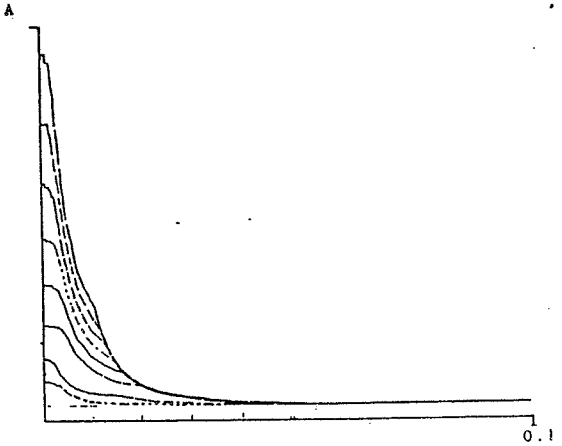



Figure 8. Accélération limites pour N infini.

Valeur optimale du nombre de processeurs

Il apparaît sur la Figure 8 donnant l'accélération en fonction de α pour diverses valeurs du nombre de processeurs, qu'il y a une accélération optimale correspondant à un nombre optimal de processeurs qui sont des fonctions linéaires de $\alpha^{-1/3}$

$$A_{opt} = a + b\ \alpha^{-1/3}$$

$$q_{opt} = c + d\ \alpha^{-1/3}$$

a, b, c, d, sont des constantes.

Donnons quelques intervalles de variation de $\alpha$ et la valeur optimale de q.

$q_{opt} = 4$ pour $0.04 < \alpha < 0.05$

$q_{opt} = 5$ pour $0.02 < \alpha < 0.04$

$q_{opt} = 6$ pour $0.016 < \alpha < 0.02$

$q_{opt} = 7$ pour $0.015 < \alpha < 0.016$

$q_{opt} = 8$ pour $0.013 < \alpha < 0.015$

$q_{opt} = 9$ pour $0.012 < \alpha < 0.013$

$q_{opt} = 10$ pour $0.010 < \alpha < 0.012$

Ces figures montrent que les performances des algorithmes parallèles sont très sensibles à la valeur du paramètre $\alpha$. Deux facteurs permettent d'atténuer cette sensibilité. D'une part, on connait des algorithmes de renumérotation des variables nodales d'un problème d'éléments finis qui permettent de réduire la largeur de bande W donc $\alpha$. D'autre part, la valeur de W ne croit que très peu (si la numérotation des noeuds est bien faite) lorsque N augmente. Donc pour de grands systèmes, on aura des valeurs de $\alpha$ très petites qui permettront d'obtenir de bonnes performances des algorithmes parallèles.

## 4. Variation des performances en fonction de N

On fixe la valeur de q à 5 processus et l'on étudie les performances des algorithmes parallèles en fonction de N pour diverses valeurs de $\alpha$. Les résultats sont produits sur la Figure 9.

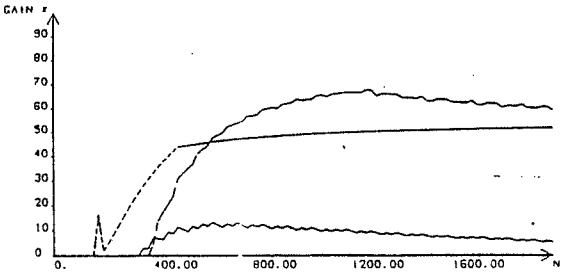



Figure 9. Variations du gain pour q = 5.

5. Performances optimales

Nous portons sur la Figure 10, une courbe donnant la variation en fonction du nombre de processus, de l'accélération dans le cas où l'on suppose N grand et $\alpha$ petit pour avoir la valeur optimale.

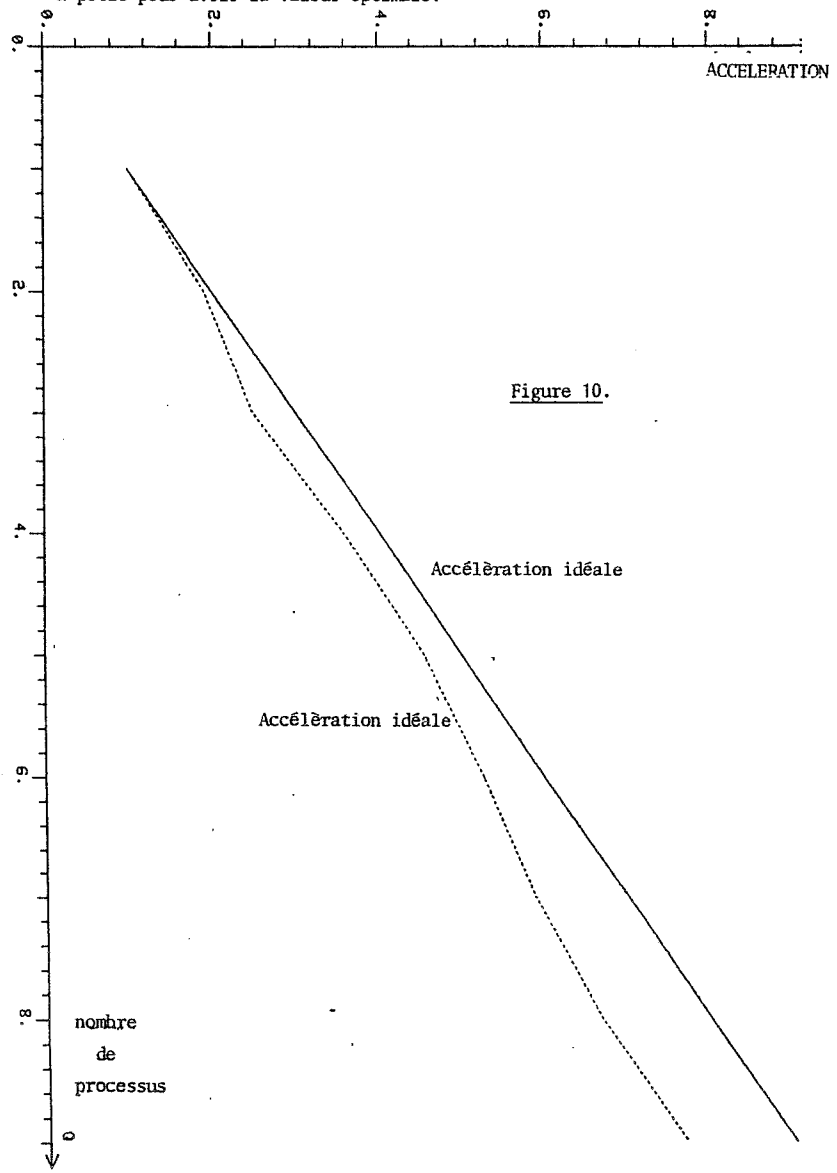



Figure 10.

## C . PERFORMANCES POUR 'ICCG' PARALLELE

### I . EVALUATION DES TEMPS DE CALCUL

#### 1. Paramètres d'état

Le temps de calcul d'une résolution par ICCG d'un système linéaire dépend actuellement de trois paramètres :

- la dimension N du système,
- le nombre total $\tau$ d'éléments non nuls de la matrice du système,
- le nombre I d'itérations nécessaires à la résolution.

$T = T(N, \tau, I)$

Nous allons montrer que l'on peut en fait éliminer les paramètres $\tau$ et I.

#### a) Elimination du paramètre $\tau$

Le paramètre $\tau$ représente le nombre total d'éléments non nuls dans la matrice représentant le système. On définit alors la variable $m = \tau/N$ qui représente le nombre moyen d'éléments non nuls par ligne du système. Cette variable représente également le nombre maximal de relations que peut avoir un noeud avec ses voisins.

En effet, si le système à résoudre est :

$$A\,x = b$$

on sait que $A_{ij} \neq 0$ si les noeuds i et j sont reliés par un segment (i.e. il existe un élément au quel ils appartiennent simultanément).

Nous avons déjà montré au Chapitre II, que le nombre m est limité à 7 pour des problèmes en deux dimensions si pour assurer une bonne précision de la méthode on admet que des éléments ayant des angles supérieurs à 30°

$$m \leq 7 \qquad \text{donc} \qquad \tau \leq 7\,N \qquad \text{(matrice symétrique)}$$

En fait, pour la plupart des problèmes d'éléments finis on peut considérer que m est une constante de l'ordre de 7 à 10. On peut choisir $m \# 8$.
Ceci est d'autant mieux vérifié que N est grand.
On peut considérer pour les mesures qui nous intéressent que $\tau$ est proportionnel à N, ce qui nous permet d'éliminer $\tau$ de la liste des paramètres d'état.

b) Elimination du paramètre I

Le préconditionnement retenu pour l'algorithme d'ICCG parallèle est légèrement différent de celui de l'algorithme classique, puisque des termes supplémentaires sont forcés à zéro lors de la décomposition incomplète. Cette modification du préconditionnement peut entraîner des nombres d'itérations différents entre l'algorithme parallèle et l'algorithme classique.

Mais l'on sait que le nombre de processus parallèles utilisés reste très inférieur à la dimension N du système, donc la différence de préconditionnement restera assez marginale.

En conséquence, la variation du nombre d'itérations entre les deux méthodes sera limitée. Nous avons effectué quelques mesures préliminaires pour un ensemble de résolutions de systèmes linéaires. Sur la Figure 1 nous avons tracé la variation du rapport :

$$i = \frac{I \text{ (algorithme parallèle)}}{I \text{ (algorithme classique)}}$$

en fonction de la taille N du système.

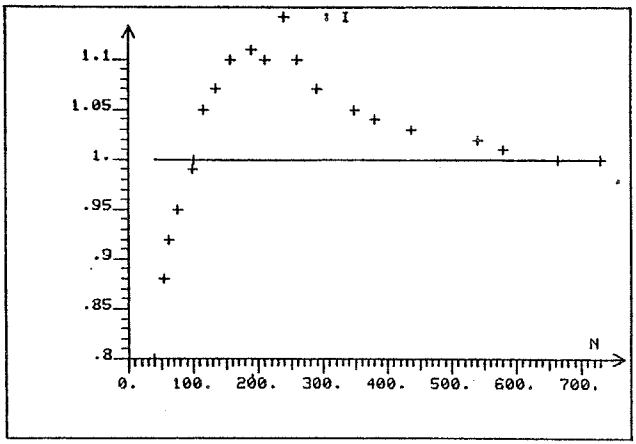



Figure 1.

Cette courbe montre que pour les petites valeurs de N, le programme parallèle demande moins d'itérations, mais que si N > 100, le programme parallèle exécute plus d'itérations. Néanmoins ce nombre d'itérations supplémentaires reste limité (< 10 %) et devient quasiment nul lorsque N est grand.

Ceci nous permet de ne pas tenir compte de la variation du nombre d'itérations entre les programmes parallèles et séquentiels.

### 2. Interpolation des temps de calcul

Le seul paramètre qui détermine le temps de calcul est alors N.
Les temps de calcul pour chaque étape j du processus i sont alors évalués par :

$$T_{i,j} = a_{1,i,j} N^{3/2} + a_{2,i,j} N^{5/4} + a_{3,i,j} N + a_{4,i,j}$$

Les coefficients $a_{k,i,j}$ sont calculés par une méhode de moindres carrés.

Ces mesures n'ont qu'à être effectuées pour le premier processus, puisque l'on trouve dans les autres processus des tâches identiques.

## II . RESULTATS DE PERFORMANCES

### 1. Etude en fonction de N

On a représenté sur la Figure 2 la variation de l'accélération pour le programme parallèle en fonction de N.
Cette courbe montre que ces algorithmes atteignent très rapidement l'accélération maximale. Les performances de ICCG parallèle sont pratiquement indépendantes de la valeur de N.

### 2. Performances en fonction de Q

Sur la Figure 3 est tracée la courbe donnant l'accélération A en fonction du nombre de processus parallèles Q. On s'apperçoit que dans le cas de l'algorithme d'ICCG la courbe est très proche de la courbe idéale A = Q.

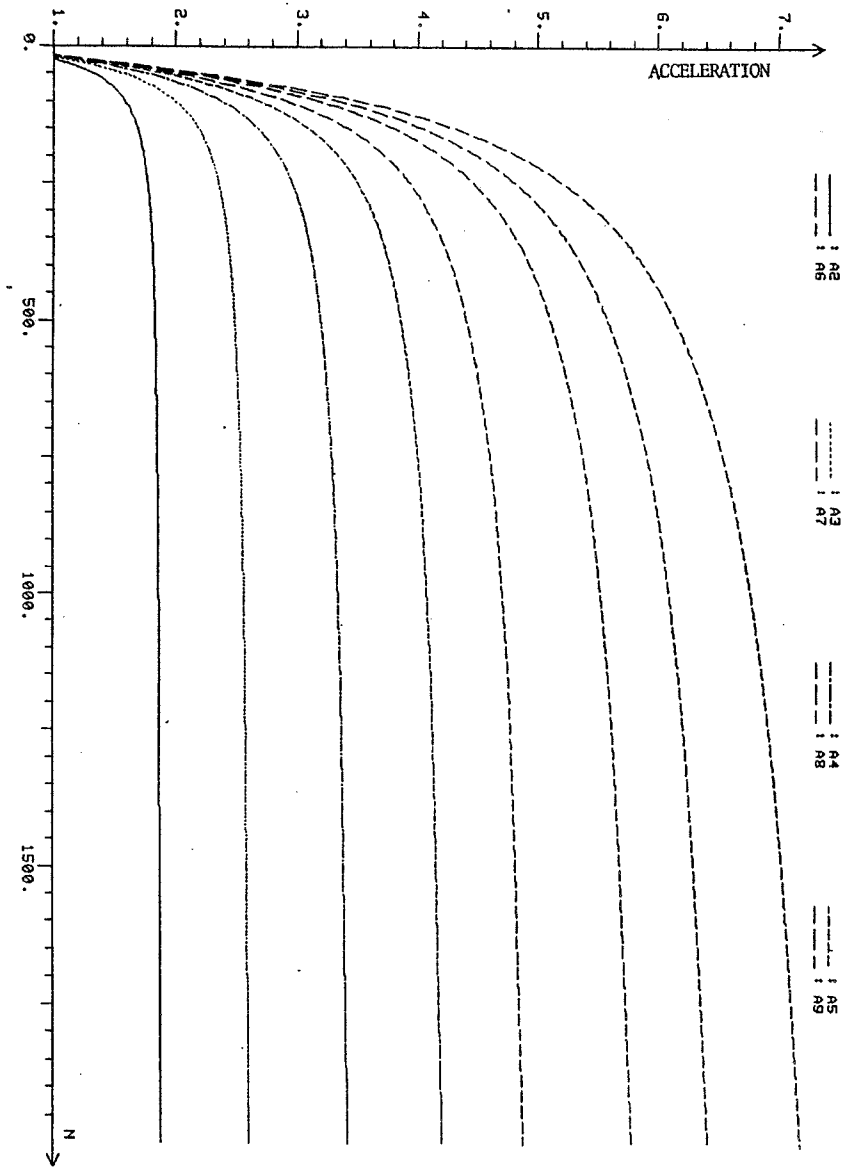



Figure 2. ICCG - Variation de l'accélération en fonction de N.

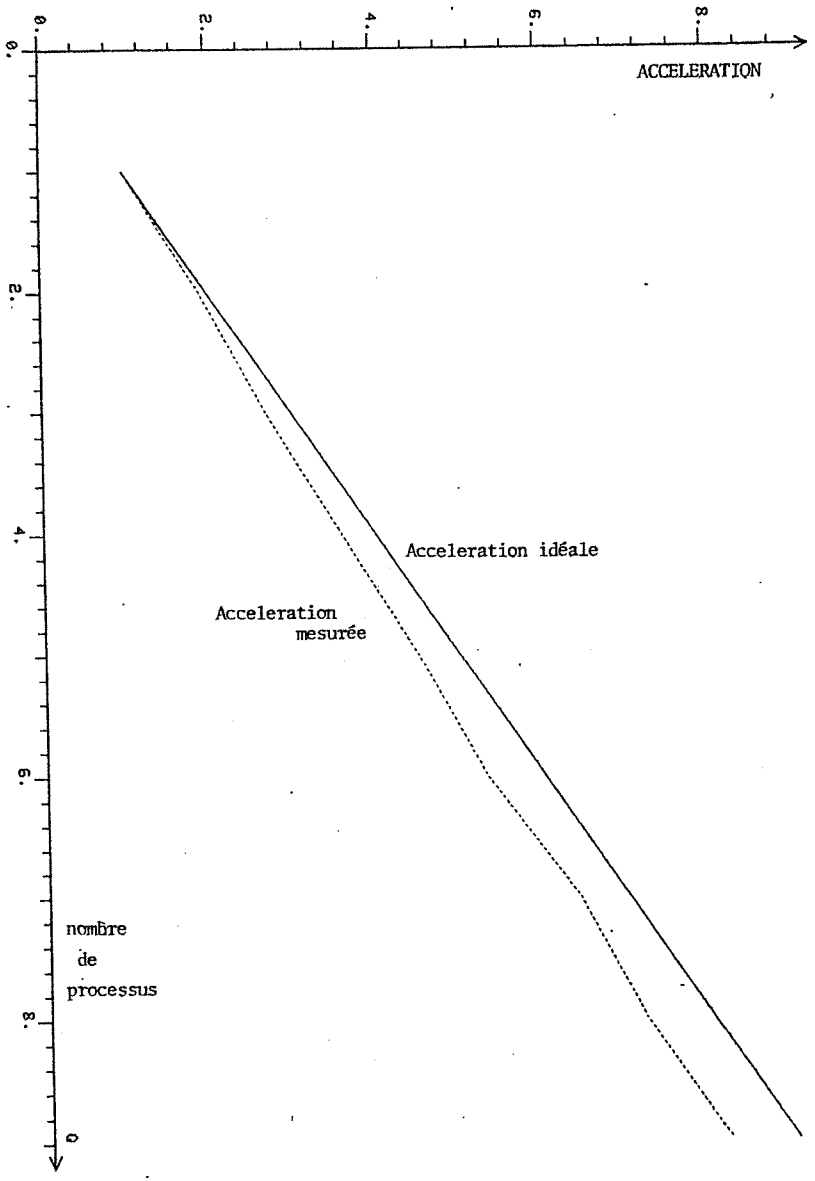

0.
2.
4.
6.
8.
ACCELERATION
0.
2.
4.
6.
8.
nombre
de
processus
Acceleration idéale
Acceleration
mesurée

Figure 3.

## D . CONCLUSION

Nous rappelons sur la Figure suivante les valeurs des performances constatées pour les diverses étapes de la construction et de la résolution d'un problème d'éléments finis.

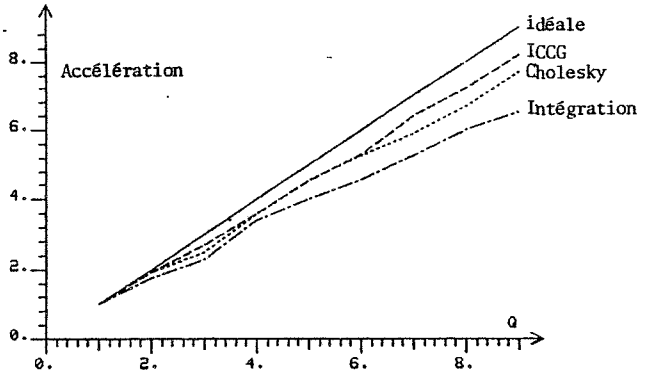



Figure 1.

La première conclusion qui apparaît sur ce schéma est que les algorithmes que nous avons définis se prêtent très bien au parallèlisme et qu'ils permettent d'espérer réduire fortement les temps d'exécution.

La seconde remarque est que pour l'intégration et la résolution par la méthode d'ICCG, ces performances optimales sont atteintes pour n'importe quel problème, alors que pour la résolution par la méthode de Cholesky il existe une contrainte assez forte (il faut que le paramètre $\alpha$ soit petit).

Pour les logiciels sur lesquels nous travaillons (FLUX 2D et DIDACT-FLUX), la méthode retenue pour la résolution est dans la plupart des cas ICCG. Ceci ne constitue pas un handicap pour notre étude puisque nos propres conclusions tendent à encourager l'usage d'ICCG.

Lors de la partie d'exploitation des résultats, les types de calcul sont tout à fait semblables (au niveau de la structure) à ceux réalisés pour l'intégration puisqu'il s'agit de calculs au niveau de chaque élément.

Nous n'avons pas approfondi l'étude pour cette partie, néanmoins, il semble que le travail réalisé pour l'intégration en parallèle puisse s'appliquer directement à cette partie. Les résultats devraient être équivalents à ceux obtenus pour l'intégration.

Notre logiciel parallèle pour la résolution des problèmes par éléments finis assure la création automatique des processus, leur utilisation pour l'intégration parallèle puis pour la résolution parallèle par ICCG.

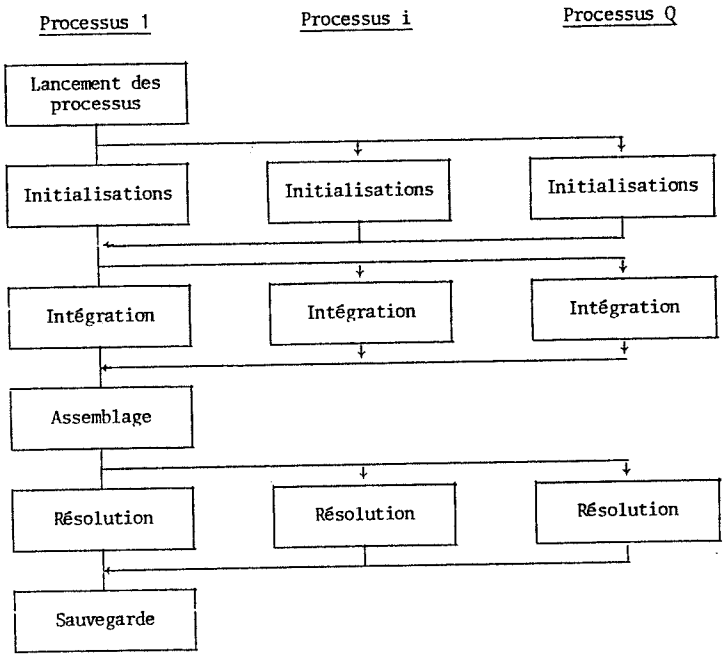



Figure 2.

Si l'on se rappelle que l'assemblage est une phase assez brève (au moins 5 fois plus rapide que l'intégration), on voit alors que les performances globales du logiciel parallèle seront très proches de celles obtenues pour l'intégration et la résolution.
L'ensemble du logiciel parallèle peut donc apporter des améliorations très sensibles pour le problème des temps d'exécution.

# CHAPITRE

# 5

## CHAPITRE V . ARCHITECTURE PARALLELE EN RESEAU

INTRODUCTION

A . LE RESEAU, UNE BONNE ARCHITECTURE POUR LES PROBLEMES D'ELEMENTS FINIS

B . CAS DU RESEAU APOLLO, DEFINITIONS TECHNIQUES

- I . GESTION DU RESEAU
- II . STRUCTURE MATERIELLE DES STATIONS
- III . SYSTEME DE GESTION DES STATIONS
- IV . CARACTERISTIQUES TECHNIQUES

C . MODELISATION DU RESEAU

- I . INTRODUCTION
- II . MODELE D'UNE STATION ET DU RESEAU
- III . CONDITIONS D'ETUDE DU RESEAU
- IV . FONCTIONNEMENT D'UNE STATION NON CONNECTEE AU RESEAU
  - Introduction
  1. Modèle général
     - a) Définition du modèle
     - b) Résolution du système
  2. Cas où il n'y a qu'un processus sur la station
     - a) Calcul du débit
     - b) Calcul du temps de réponse
     - c) Calcul de $\mu_1$, $\mu_2$ et $p_2$
  3. Cas d'un processus parmi k dans la station
  4. Cas de Q processus en parallèle parmi k
  5. Performances
     - a) Rappel
     - b) Première comparaison
     - c) Deuxième comparaison

V . COMPORTEMENT DES ALGORITHMES PARALLELES SUR LE RESEAU

1. Modèle
   a) Remarque préliminaire et conditions d'études
   b) Modélisation
   c) Résolution du modèle
   d) Performances d'algorithmes parallèles
2. Mesure des paramètres du modèle
3. Méthodes de calcul de $R_D$ et $R_{UC}$
   a) Méthode analytique
   b) Simulation

D . APPLICATION DES MODELES A DES ALGORITHMES PARALLELES

1. Cas d'une station isolée
2. Cas d'un réseau
   a) Validation des modèles
   b) Conséquences pour les algorithmes parallèles
   c) Mode d'utilisation recommandé du réseau.

# CHAPITRE V . ARCHITECTURE PARALLELE EN RESEAU

## INTRODUCTION

L'exploitation des besoins en télécommunication a permis un développement considérable de grands réseaux d'ordinateurs. Profitant des retombées scientifiques et techniques de ce développement, sont apparus des systèmes aux dimensions plus modestes, organisés en réseaux locaux. Ces systèmes relient entre eux, un petit nombre de postes de travail autonomes. Généralement, le but recherché à l'origine est la mise en commun de ressources physiques (des périphériques comme des imprimantes, des postes graphiques, ...) ou l'échange facile de logiciels. Mais un tel réseau peut également permettre l'utilisation de ressources en capacité de calcul lorsque, depuis un noeud du réseau, on est capable de router des tâches (de calcul) vers d'autres noeuds.

C'est pourquoi nous nous sommes intéressés, dans le cadre de notre étude, à ce type d'architecture qui présente d'importantes qualités telles la souplesse d'utilisation, des possibilités de diversification. Nous montrerons ici qu'un réseau local peut être bien adapté pour réduire les temps d'exécution de programmes d'éléments finis. En particulier, nous décrirons le cas d'un réseau de type "Apollo" dont nous analyserons les performances. Nous déterminerons enfin les conditions optimales d'implantation d'algorithmes parallèles sur ce réseau.

Le matériel Apollo est en cours de développement dans notre Laboratoire et dans la région Grenobloise et doit donner rapidement naissance à un réseau local.

## A . LE RESEAU, UNE BONNE ARCHITECTURE POUR LES PROBLEMES D'ELEMENTS FINIS.

Un système multiprocesseur tel qu'on l'entend classiquement, est constitué par un ensemble de modules (unités de calcul, processus) fortement connectés entre eux, et permettant des échanges au niveau d'informations élémentaires.
Au contraire, une architecture en réseau est constituée d'un ensemble d'unités de traitement faiblement couplées entre elles, Pour utiliser des programmes parallèles sur un réseau, il faudra donc un ensemble de programmes faiblement couplés entre eux, c'est-à-dire où le nombre de synchronisations, d'attentes ou d'échanges de données reste limité par rapport à la durée d'exécution.

Or, il se trouve que les algorithmes parallèles que nous avons proposés dans les Chapitres précédents, répondent assez bien à ces critères. Ceci ressort particulièrement si l'on regarde la représentation sous forme de graphes de ces algorithmes. On distingue un certain nombre de tâches, dont nos mesures ont montré qu'elles représentent chacune un grand nombre de calculs élémentaires. Ces tâches sont reliées entre elles par un petit nombre de synchronisations.

De façon plus générale, nous savons que tout logiciel d'éléments finis est composé de deux types de travaux, d'une part des travaux très interactifs ne nécessitant que peu de besoins de calcul, d'autre part quelques séquences de travaux demandant une forte puissance de calcul. On voit par là, qu'une structure en réseau, du fait de sa souplesse d'utilisation, convient très bien à ces problèmes. En effet, un utilisateur de tels logiciels peut se contenter de la puissance de calcul d'un seul noeud du réseau lors de la phase interactive, puis utiliser toute la puissance du réseau, mais de façon momentanée, lors des phases de calcul.

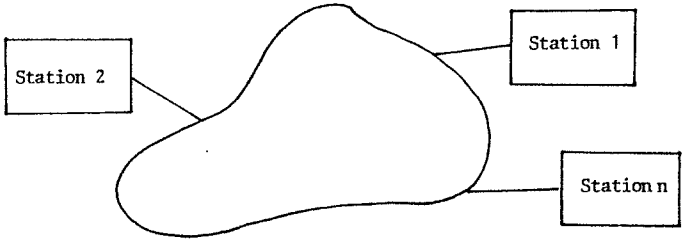
Un réseau est donc doublement efficace pour des logiciels d'éléments finis car il permet le parallèlisme pour des algorithmes variés et ceci à un moindre coût.

## B . CAS DU RESEAU APOLLO, DEFINITIONS TECHNIQUES

Le poste de travail Apollo est constitué d'un ordinateur que l'on peut classer parmi les super-mini 32 bits. Il constitue un noeud du réseau appelé DOMAIN. Le système DOMAIN est un réseau local à haute performance, reliant un ensemble d'ordinateurs, constituant un environnement distribué. Les noeuds du réseau sont reliés par un câble coaxial à large bande, et sont organisés avec une topologie en anneau. Ce système permet aux ressources en matériel et en logiciel d'être partagées entre tous les points du réseau.

### I . GESTION DU RESEAU

Chaque station a, successivement, droit d'émettre sur le réseau. Lorsqu'une station a fini d'émettre, elle passe la main à la suivante. Cette règle permet d'éviter les collisions de paquets d'émissions, sur le réseau. On appelle souvent ce type de réseau, un réseau "à jeton", en ce sens que seule la station qui dispose du jeton (façon imagée de représenter le droit d'accès au réseau) peut émettre.



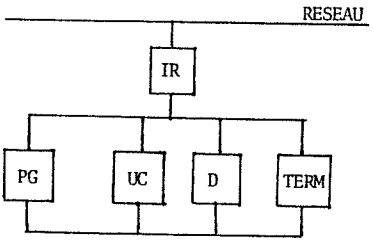

Cette stratégie est bien adaptée pour un réseau où le débit maximal est très élevé mais le taux d'utilisation relativement faible.

## II . STRUCTURE MATERIELLE DES STATIONS

Le réseau accepte des stations composées de postes de travail différents. Néanmoins, dans un souci de simplification, nous supposerons pour la suite que toutes les stations sont d'un seul type que nous allons décrire maintenant.

Une station est composée de l'ensemble suivant :

- Une Unité Centrale (notée UC)
- Un disque dur } (D)
- Un processeur chargé des échanges avec le disque }
- Un terminal graphique } (PG)
- Un processeur graphique }
- Une console alphanumérique (TERM)
- L'interface d'échange avec le réseau (IR)



## III . SYSTEME DE GESTION DES STATIONS

Nous rappellerons tout d'abord deux définitions de base que nous utiliserons dans ce Chapitre.
On appelle processeur toute unité physique capable de traiter des informations, par exemple une Unité Centrale ou bien un processeur Graphique. C'est une notion associée au matériel.
On appelle processus une entité du système de gestion d'une machine informatique. Cette entité est chargée d'effectuer des tâches de calcul, d'entrée-sortie. C'est une notion associée au logiciel.

Ainsi, le système gérant un processeur peut permettre l'existence de plusieurs processus, travaillant en temps partagé (Time Sharing).
Pour le réseau que nous utiliserons, chaque station ne dispose que d'un seul processeur de calcul, l'unité centrale. Par contre, le système implanté sur le matériel Apollo permet de générer jusqu'à 15 processus par station.

## IV . CARACTERISTIQUES TECHNIQUES

Nous donnerons ici les différentes grandeurs fournies par le constructeur et qui seront par la suite :

- unité centrale : . processeur 32 bits V L S I
  . 15 processus permis simultanément
  . 16 M byte d'espace d'adresse (1 page = 1024 byte)
- mémoire centrale : de 0,5 à 1,5 M byte. Pour l'étude qui nous intéresse, nous avons choisi 1 M byte.
- interface avec le réseau : 2 ports d'Entrée-Sortie RS 232-C, la vitesse maximale est de 19,2 K baud
- réseau : . vitesse de transmission 12 M bit/s
  . topologie en anneau, type "à jeton"
  . jusqu'à 1 km entre deux noeuds du réseau.
- disque : . Winchester : 3,4 M byte
  . vitesse de transfert 1 M byte/s
  . temps de latence moyen 8 ms
  . temps de recherche moyen 42 ms
- terminal graphique : . résolution 1024 x 800 pixels
  . transfert câblé 12 M bit/s

## C . MODELISATION DU RESEAU

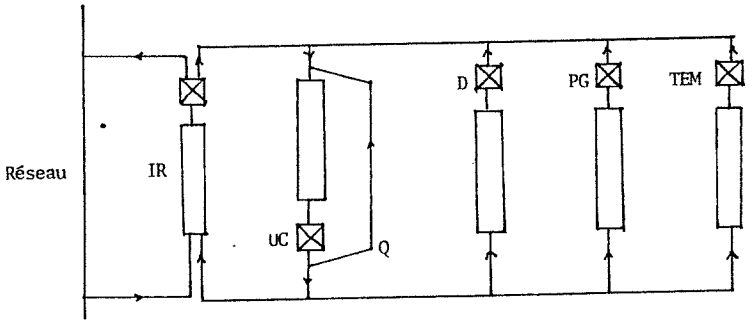
### I . INTRODUCTION

Lorsque l'on construit le modèle d'un système, on est souvent amené à des choix permettant de faire des hypothèses simplificatrices. Ces choix sont fortement dépendant des informations que l'on désire obtenir du modèle.

L'objectif de ce travail est de répondre aux deux questions suivantes. Tout d'abord déterminer quel est le mode de fonctionnement optimal du réseau. Puis d'en déduire les performances des algorithmes parallèles sur ce réseau.

### II . MODELE D'UNE STATION ET DU RESEAU

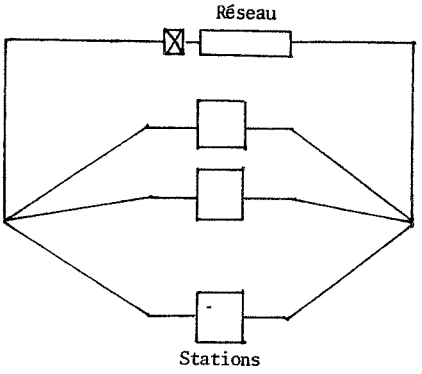
Le modèle d'étude d'une station est constitué d'un réseau de files d'attentes, dans lesquelles les "clients" sont les processus générés par le système.
On obtient le schéma suivant :




UC : Unité Centrale D : Disque PG : processeur graphique
TERM : Terminal IR : Interface Réseau.

Le passage dans la branche notée Q correspond à la fin de question. Lorsqu'un processus accède à l'unité centrale, il ne peut la garder indéfiniment. Au bout d'un temps appelé quantum, il est replacé en fin de file d'attente de l'unité centrale, de manière à ne pas bloquer les autres processus.

On peut également représenter l'ensemble du réseau Apollo par un système de files d'attentes.




## III . CONDITIONS D'ETUDE DU RESEAU

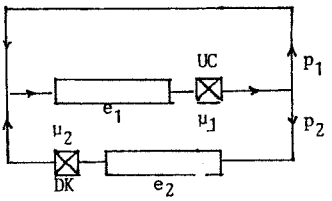
Le réseau, tel qu'il est décrit au Chapitre précédent, est très complexe. Un certain nombre de remarques vont nous permettre de le simplifier.

Nous ne nous intéressons pas au fonctionnement des fonctions graphiques pour cette étude, ni aux interactions du terminal. Nous ne tiendrons donc pas compte des files TERM et PG.

La gestion du réseau est faite au niveau global, ce qui permet de supprimer les files notées IR sur chaque station pour les remplacer par une file unique représentant le réseau.

Nous supposerons enfin que le réseau a atteint son état stationnaire. Les tâches associées à chaque processus sont considérées comme infiniment longues et nous nous intéresserons en particulier au débit en sortie de l'unité centrale.

Les hypothèses précédentes permettent de construire le schéma simplifié suivant :



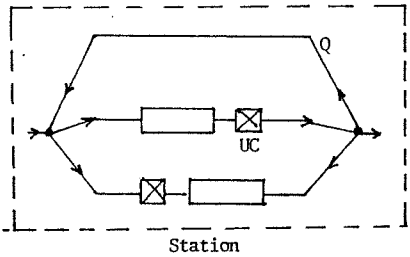

## IV . FONCTIONNEMENT D'UNE STATION NON CONNECTEE AU RESEAU

### Introduction

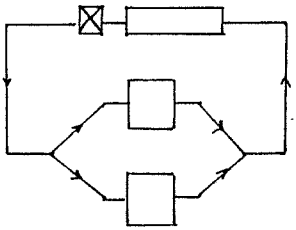
Le but que nous poursuivons est de connaître le fonctionnement du réseau. Il est néanmoins très utile d'étudier une station isolée. En effet, à ce jour nous ne disposons que d'un seul poste de travail, son étude permettra de mesurer des paramètres utiles pour la modélisation du réseau. Cette étude préliminaire est donc importante pour la validation des résultats que nous désirons obtenir.

### 1. Modèle général

#### a) Définition du modèle




Station



Réseau

Nous notons : $e_i$ le nombre moyen de passages dans la file i
$\mu_i$ le taux de service de la file i
$U_i$ le taux d'occupation de la file i
$n_i$ le nombre de processus dans la file i

On posera $k = \sum_{i=1}^{i=2} n_i$ nombre total de processus dans le système.

Un état du système est caractérisé par une valeur de $n_1$ et une valeur de $n_2$.

La probabilité de se trouver dans cet état est donnée par le théorème de Gordon et Newell :

$$P(n_1, n_2) = C \left(\frac{e_1}{\mu_1}\right)^{n_1} \left(\frac{e_2}{\mu_2}\right)^{n_2}$$

Cette formule est valable si les conditions suivantes sont vérifiées :

- pour chaque file, le service est exponentiel de taux $\mu_i$
- les règles de priorités sont First In First Out (premier entré, premier servi)
- les probabilités $p_1$ (fin de quantum) et $p_2$ (accès au disque) sont choisies constantes.

Remarque : $p_1$ et $p_2$ dépendent en fait du taux instantané de défauts de pages dont la loi est connue de façon imprécise. Il ne parait alors pas justifié d'utiliser des méthodes d'analyse plus fines (décomposabilité).

C est une constante de normalisation, que l'on calcule en écrivant :

$$\sum_{n_1+n_2=k} P(n_1, n_2) = 1$$

d'où : $$C = \frac{1}{\sum_{n_1+n_2=k} \prod_{i=1}^{i=2} \left(\frac{e_i}{\mu_i}\right)^{n_i}}$$

Le taux d'occupation de l'unité centrale est $U_1 = 1 - P(0,k)$ et le débit en sortie de cette file est : $d_o = U_1 \mu_1$.

b) Résolution du système

On peut écrire $e_2 = p_2 e_1$ d'où :

$$P(n_1,n_2) = C e_1^k \left(\frac{1}{\mu_1}\right)^{n_1} \left(\frac{p_2}{\mu_2}\right)^{k-n_1} = C \left(\frac{e_1 p_2}{\mu_2}\right)^{k} \left(\frac{\mu_2}{\mu_1 p_2}\right)^{n_1}$$

d'où :

$$C \left(\frac{e_1 p_2}{\mu_2}\right)^{k} = \frac{1}{\sum_{n_1=o}^{n_1=k} \left(\frac{\mu_2}{\mu_1 p_2}\right)^{n_1}} = \frac{1 - \frac{\mu_2}{\mu_1 p_2}}{1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{k+1}}$$

On cherche à calculer P(0,k) :

$$P(n_1,n_2) = C \left(\frac{e_1 p_2}{\mu_2}\right)^k \left(\frac{\mu_2}{\mu_1 p_2}\right)^{n_1}$$

d'où $P(0,k) = C \left(\frac{e_1 p_2}{\mu_2}\right)^k$

finalement :

$$\boxed{P(0,k) = \frac{1 - \frac{\mu_2}{\mu_1 p_2}}{1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{k+1}}}$$

et $d_o = U_1 \mu_1 = (1 - P(0,k))\ \mu_1$

2. Cas où il n'y a qu'un processus sur la station

a) Calcul du débit

Si $k = 1$ $(n_1 + n_2 = 1)$ alors $P(0,k) = \frac{1}{1 + \frac{\mu_2}{\mu_1 p_2}}$

d'où $U_1 = 1 - P(0,k) = \frac{\frac{\mu_2}{\mu_1 p_2}}{1 + \frac{\mu_2}{\mu_1 p_2}} = \frac{\mu_2}{\mu_1 p_2 + \mu_2}$

finalement :

$$\boxed{d_o = \mu_1 U_1 = \frac{\mu_1 \mu_2}{\mu_1 p_2 + \mu_2}}$$

b) Calcul du temps de réponse

Posons L la longueur moyenne de la file de l'unité centrale :

$$L = \sum_{i=o}^{i=k} i \times P(i,k-i)$$

Le temps de réponse d'un processus pour un passage dans la file est :

$$T^{(R)}_{1\ \text{passage}} = \frac{L}{d_o} \qquad \text{[formule de Little]}$$

Comme l'on s'intéresse au cas où il n'y a qu'un seul processus, $L = U_1$ ; donc

$$T^{(R)}_{1 \text{ passage}} = \frac{L}{d_o} = \frac{U_1}{d_o} = \frac{1}{\mu_1}$$

Ce qu'il nous faut calculer est le temps de réponse associé à $e_1$ passages dans la file 1 et $e_2$ passages dans la file 2, avec $e_2 = p_2\ e_1$ :

$$T^{(R)} = \frac{e_1}{\mu_1} + \frac{e_2}{\mu_2} = e_1 \left(\frac{1}{\mu_1} + \frac{p_2}{\mu_2}\right)$$

Or $\mu_1$ représente le temps de service de l'unité centrale pour 1 passage, donc le temps de calcul nécessaire à un processus qui demande $e_1$ passages dans l'unité centrale est :

$$T^{(C)} = \frac{e_1}{\mu_1}$$

d'où

$$\boxed{T^{(R)} = \mu_1\ T^{(C)} \left(\frac{1}{\mu_1} + \frac{p_2}{\mu_2}\right) = T^{(C)} \left(1 + \frac{p_2\ \mu_1}{\mu_2}\right)}$$

c) Calcul de $\mu_1$, $\mu_2$ et $p_2$

Nous posons $\theta$ = temps moyen entre 2 défauts de page (accès au disque)

$q$ = question, temps maximum d'utilisation de l'unité centrale au delà duquel un processus est remis en fin de file d'attente ($q = 0{,}5$ s).

Pendant un temps T, il y a $\frac{T}{q}$ fin de quantum et $\frac{T}{\theta}$ défaut de page.

Si I est le temps moyen entre deux interruptions, il y aura pendant le temps T, un nombre d'interruptions égal à $\frac{T}{I}$ .

On a alors :

$$\boxed{\mu_1 = \frac{1}{I} = \frac{1}{q} + \frac{1}{\theta}}$$

$\mu_2$ = inverse de (temps pour copier une page + temps de latence)

d'où

$$\boxed{\mu_2 = \frac{1}{50 \times 10^{-3} + \frac{1024 \text{ (byte/page)}}{10^6 \text{ (byte/seconde)}}} \simeq \frac{1}{51 \times 10^{-3}} = 19{,}6}$$

On aura également :

$$\boxed{p_2 = \frac{T/\theta}{T/I} = \frac{I}{\theta} = \frac{1}{\mu_1 \theta}} \qquad \text{et } p_1 = 1 - p_2$$

Pour connaître toutes les caractéristiques du modèle il faut avoir les valeurs de q et $\theta$. Le quantum q est fourni par le constructeur. Quant au paramètre $\theta$, il doit être mesuré. On admet généralement une loi empirique reliant $\theta$ à la taille M de la mémoire centrale.

$$\theta = K \left(\frac{M}{k}\right)^{\alpha}$$

où K et $\alpha$ sont des constantes, et k est le nombre de processus qui se partagent la mémoire centrale.

3. Cas d'un processus (programme monoprocesseur) parmi k dans la station

Soient $R_1$ et $R_2$ les temps de réponse des files 1 et 2, c'est-à-dire le temps moyen nécessaire pour un processus pour traverser ces files. Pour un processus, le temps de réponse est :

$$T^{(R)} = e_1 R_1 + e_2 R_2 = e_1 R_1 + e_1 p_2 R_2$$

et ceci pour un temps de calcul :

$$T^{(C)} = \frac{e_1}{\mu_1}$$

Si $L_1$ et $L_2$ sont les longueurs moyennes des files 1 et 2, on a :

$$R_1 = \frac{L_1}{d_o} \quad \text{et} \quad R_2 = \frac{L_2}{p_2 d_o} \quad \text{d'où} \quad R_1 + p_2 R_2 = \frac{L_1 + L_2}{d_o} = \frac{k}{d_o}$$

d'où $T^{(R)} = e_1 \frac{k}{d_o} = k \frac{\mu_1}{d_o} T^{(C)}$

or $d_o = \mu_1 U_1$, on a la relation :

$$\boxed{T^{(R)} = k \frac{T^{(C)}}{U_1}}$$

avec : $$U_1 = 1 - P(0,k) = 1 - \frac{1 - \frac{\mu_2}{\mu_1 p_2}}{1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{k+1}} \quad \text{(cf. §.1.b)}$$

d'où $$U_1 = \left(\frac{\mu_2}{\mu_1 p_2}\right) \left[ \frac{1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{k}}{1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{k+1}} \right]$$

4. Cas de Q processus en parallèle parmi k

Dans la station, il y a Q processus exécutant des programmes parallèles et $k_1 = k - Q$ autres processus.

Soit e' le nombre de passages dans la file 1 de tous les processus parallèles, le temps de calcul pour 1 de ces processus est :

$$T^{(C)}_{multi} = \frac{e'}{Q\,\mu_1}$$

et le temps de réponse associé : $$T^{(R)}_{multi} = \frac{e'}{\mu_1 Q} \cdot \frac{k_1 + Q}{U_1} = T^{(C)}_{multi} \cdot \frac{k_1 + Q}{U_1}$$

formulé où $U_1$ prend sa valeur pour $k = k_1 + Q$ processus :

$$U_1 = \left(\frac{\mu_2}{\mu_1 p_2}\right) \left( \frac{1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{k_1+q}}{1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{k_1+q+1}} \right)$$

5. Performances

a) Rappel

Nous donnons ici un bref rappel de définitions qui ont été développées ailleurs.

le gain $$G = \frac{T_{mono} - T_{multi}}{T_{mono}}$$

l'accélération $$a = \frac{T_{mono}}{T_{multi}}$$

l'efficacité $$e = \frac{T_{mono}}{\sum_{i=1}^{i=Q} T_{multi}}$$

b) Première comparaison

Nous comparerons les temps de réponse associés d'une part au cas d'un processus unique sur une station et d'autre part Q processus occupant entièrement la station. Avec les résultats des paragraphes précédents, nous obtenons les résultats suivants :

$$T^{(R)}_{mono} = T^{(C)}_{mono} \left(1 + \frac{p_2 \mu_1}{\mu_2}\right)$$

$$T^{(R)}_{multi} = T^{(C)}_{multi} \cdot \frac{Q \left(1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{Q+1}\right)}{\left(\frac{\mu_2}{\mu_1 p_2}\right) \left(1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{Q}\right)}$$

Si l'on appelle $a_R$ l'accélération en temps de réponse et $a_C$ l'accélération en temps de calcul, on obtient :

$$a_R = \frac{T^{(R)}_{mono}}{T^{(R)}_{multi}} = \frac{T^{(C)}_{mono}}{T^{(C)}_{multi}} \cdot \frac{\left(1 + p_2 \frac{\mu_1}{\mu_2}\right) \left(\frac{\mu_2}{\mu_1 p_2}\right) \cdot \left(1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{Q}\right)}{Q \left(1 - \left(\frac{\mu_2}{\mu_1 p_2}\right)^{Q+1}\right)}$$

d'où

$$\boxed{a_R = a_C \cdot \frac{1}{Q} \cdot \frac{(1 + x)(1 - x^{Q})}{(1 - x^{Q+1})} \quad \text{avec} \quad x = \frac{\mu_2}{\mu_1 p_2}}$$

c) Deuxième comparaison

On choisit de calculer d'une part le temps de réponse associé à un processus exécutant un programme sur une station parmi $(k_1 + 1)$ processus, et d'autre part, le temps de réponse associé à Q processus parallèles sur une station en comportant $(k_1 + Q)$. En posant toujours $x = \frac{\mu_2}{\mu_1 p_2}$, on a :

$$T^{(R)}_{mono} = T^{(C)}_{mono} \cdot \frac{(k_1 + 1)(1 - x^{k_1 + 2})}{x (1 - x^{k_1 + 1})}$$

$$T^{(R)}_{multi} = T^{(C)}_{multi} \cdot \frac{(k_1 + Q)(1 - x^{k_1 + Q + 1})}{x (1 - x^{k_1 + Q})}$$

ce qui donne finalement :

$$a_R = a_C \cdot \left(\frac{k_1 + 1}{k_1 + Q}\right) \cdot \left(\frac{1 - x^{k_1+2}}{1 - x^{k_1+Q+1}}\right) \cdot \left(\frac{1 - x^{k_1+Q}}{1 - x^{k_1+1}}\right).$$

## V . COMPORTEMENT DES ALGORITHMES PARALLELES SUR LE RESEAU

### 1. Modèle

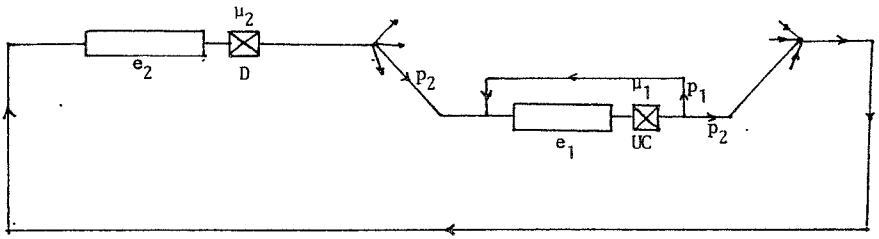
#### a) Remarque préliminaire et conditions d'études

Des caractéristiques techniques citées au paragraphe B.IV, il ressort que l'on peut considérer que le débit du réseau est suffisament grand pour que l'on puisse négliger son influence dans la modélisation qui va suivre.

Nous supposerons un réseau sur lequel sont connectées $N_S$ stations comportant chacune une unité centrale mais pas de disque. En plus est connecté sur le réseau un unique disque dur. Le problème majeur est alors celui des défauts de page. Chaque fois qu'un processus a besoin de données ou de code informatique ne se trouvant pas en mémoire centrale de la station où il travaille, il doit faire un accès au disque via le réseau.

#### b) Modélisation

Les hypothèses précédentes permettent de construire le schéma suivant :



On suppose que la probabilité $p_e$ d'accéder à une station en sortie du disque est la même pour toutes les stations :

$p_e = \frac{1}{N_S}$ . Les résultats des chapitres précédents nous donnent $\mu_2 = 19{,}6$ ;

$\mu_1 = \frac{1}{q} + \frac{1}{\theta}$ et $p_2 = \frac{1}{\mu_1 \theta}$ .

c) Résolution du modèle

On note toujours $T^{(C)}$ le temps de calcul et $T^{(R)}$ le temps de réponse demandés par un processus. Nous voulons obtenir la loi donnant $T^{(R)} / T^{(C)}$ en fonction du nombre de processus dans le système.

Soit $e_1$ le nombre de passages dans une file d'une unité centrale, quand on est passé $e_2$ fois sur le disque, on a alors

$$e_1 p_1 + \frac{e_2}{N_S} = e_1 \quad \text{donc} \quad e_1 = \frac{e_2}{p_2 N_S} .$$

On peut choisir $e_2 = 1$ d'où $e_1 = \frac{1}{p_2 N_S}$ .

Soit K le nombre de passages dans une unité centrale donnée pendant une exécution de durée $T^{(C)}$ :

$$T^{(C)} = K \frac{N_S}{\mu_1} \quad \text{et} \quad K = H e_1$$

où H est une constante.

Soit K' le nombre de passages sur le disque pour la même exécution de durée $T^{(C)}$ :

$$K' = H e_2$$

On obtient pour $e_2 = 1$ : $K' = H$ et $T^{(C)} = \frac{H e_1 N_S}{\mu_1} = \frac{H}{\mu_1 p_2}$

d'où : $K' = H = T^{(C)} \mu_1 p_2$

Or, si l'on pose $R_D$ = temps de réponse de la file du disque
$R_{UC}$ = temps de réponse de la file de l'unité centrale

On a : $T^{(R)} = K' R_D + K N_S R_{UC}$

d'où finalement :

$$\boxed{T^{(R)} = T^{(C)} \mu_1 [p_2 R_D + R_{UC}]}$$

$R_D$ et $R_{UC}$ dépendent du nombre de processus et du nombre de stations, ils ne peuvent être obtenus avec une formule simple. Pour obtenir leurs valeurs, on peut utiliser une méthode analytique (appelée Mean Value Analysis) ou bien une simulation. Le calcul de $R_D$ et $R_{UC}$ sera développé plus loin dans l'exposé.

Remarque :
en utilisant la formule de Little on trouve $T^{(R)} = T^{(C)} . k/U_1$
où k est le nombre total de processus sur l'ensemble du réseau.

d) Performances d'algorithmes parallèles

On pose $f = T^{(R)} / T^{(C)} = \mu_1 (p_2 R_D + R_{UC})$

Pour un programme monoprocesseur $f_{mono} = T^{(R)}_{mono} / T^{(C)}_{mono}$ et pour des programmes parallèles $f_{multi} = T^{(R)}_{multi} / T^{(C)}_{multi}$.

d'où
L'accélération en temps de calcul est $a_C = T^{(C)}_{mono} / T^{(C)}_{multi}$

d'où l'accélération en temps de réponse :

$$a_R = T^{(R)}_{mono} / T^{(R)}_{multi} = \frac{f_{mono}\ T^{(C)}_{mono}}{f_{multi}\ T^{(C)}_{multi}}$$

Ce qui nous donne la relation :

$$\boxed{a_R = a_C \frac{f_{mono}}{f_{multi}}}$$

La valeur de $a_C$ est obtenue par les mesures de temps de calcul de programmes parallèles. Les paramètres f seront donnés par application des formules données aux chapitres précédents. La relation ci-dessus permet alors de prévoir les performances en temps de réponse des programmes parallèles.

On remarquera que $f = f(k, N_S)$ où k est le nombre total de processus sur le réseau et $N_S$ le nombre de stations.

## 2. Mesure des paramètres du modèle

Nous savons que $\mu_2 = 19,6$ le constructeur donne un quantum $q = 0,5$ seconde. Il reste à déterminer $\theta$ qui nous permettra de calculer

$$\mu_1 = \frac{1}{q} + \frac{1}{\theta} \quad \text{et} \quad p_2 = \frac{1}{\mu_1 \theta}.$$

Pour cela il faut mesurer les paramètres de la loi donnant $\theta$ en fonction de la taille de la mémoire centrale :

$$\theta = K \left(\frac{M}{K}\right)^{\alpha} .$$

Comme il n'était pas possible de mener des mesures sur du matériel Apollo au niveau des défauts de page, les mesures ont été effectuées sur HB 68 (Système Multics). Sur ce système la mémoire centrale est de 20 M byte. Si l'on ramène ces mesures à la taille méméoire de 1 M byte d'une station Apollo, on obtient la formule suivante :

$$\boxed{\theta = 0,964 \cdot \frac{1}{k^{1,61}}}$$

Cette formule sera validée plus loin par des mesures de temps de calcul et de temps de réponse sur une station Apollo et par application des formules précédentes.

## 3. Méthodes de calcul de $R_D$ et $R_{UC}$

Pour pouvoir utiliser nos modèles, il faut encore définir des méthodes de calcul des paramètres $R_D$ et $R_{UC}$. Nous disposons de deux méthodes pour cela. D'une part une méthode analytique qui fournit un algorithme itératif résolvant les systèmes de files d'attentes tels que ceux que nous avons décrits. La seconde méthode est l'utilisation d'un logiciel, appelé QNAP2, qui permet de simuler des systèmes de files d'attentes. Nous avons mis en oeuvre ces deux méthodes de manière à avoir une confirmation des résultats.

a) Méthode analytique

L'algorithme utilisé est l'algorithme de Reiser pour les reseaux fermés ou Mean Value Analysis.

$\forall i$ , $L_i(o) = 0$

Faire de N = 1 à K

Calculer $\forall i$ , $R_i(N) = \frac{1}{\mu_i} (1 + L_i(N-1))$

Calculer $\forall i$ , $\lambda_i(N) = \frac{N}{\sum_i e_i R_i(N)}$

Calculer $\forall i$ , $L_i(N) = \lambda_i(N) \; e_i \; R_i(N)$

Dans notre cas particulier, nous nous heurtons à un problème supplémentaire. En effet, la valeur de $\mu_1$ dépend $\theta$ et ce paramètre $\theta$ dépend de la longueur de la file 1, c'est-à-dire $L_1$. Donc pour appliquer cet algorithme il faudrait connaître $L_1$ a priori, ce qui est contradictoire. On peut surmonter cette difficulté en itérant la méthode précédente à partir d'une valeur de $\theta$ donnée a priori, qui fournit une première valeur de $L_1$ qui permet à son tour de calcul un nouveau $\theta$, etc...
En théorie, on devrait itérer jusqu'à ce que l'on trouve successivement deux fois la même valeur de $\theta$. En pratique, on peut montrer qu'il suffit de quelques itérations.

$L_1 = L_1(0)$ donné a priori

Faire de M = 1 à 5

Calculer $\theta = 0{,}964 \cdot \frac{1}{L_1^{1{,}61}}$

Appliquer l'algorithme de Reiser

Trouver une nouvelle valeur de $L_1$

Le programme correspondant est donné à l'Annexe V.A.
Celui-ci permet d'obtenir le rapport $f = T^{(R)} / T^{(C)}$ en fonction du nombre de stations sur le réseau et en fonction du nombre total de processus sur le réseau. Nous fournissons à l'Annexe V.B le Tableau des valeurs de f pour une station isolée. L'Annexe V.C présente des courbes donnant f en fonction du nombre de processus pour un nombre de stations égal successivement à 1, 2, 3 et 10.

b) Simulation

Le logiciel QNAP2 permet de simuler le réseau de files d'attentes qui nous intéresse. Les caractéristiques de chaque station du système pouvent être décrites dans un langage propre au logiciel, il en est de même pour indiquer les relations entre les différentes files. Le programme est donné à l'Annexe V.D. Ce programme a été écrit dans le cas où l'on place deux stations sur le réseau. Le but de ce programme est de valider les résultats obtenus par la méthode précédente.

Les résultats des deux méthodes sont comparés à l'Annexe V.E.
Cette comparaison montre une très bonne concordance entre les deux méthodes.

## D . APPLICATION DES MODELES A DES ALGORITHMES PARALLELES

### 1. Cas du réseau isolé

Rappelons la formule du paragraphe C.IV.5.b :

$$a_R = a_C \times \frac{1}{Q} \frac{(1+x)(1-x^Q)}{(1-x^{Q+1})} \quad \text{avec} \quad x = \frac{\mu_2}{\mu_1 p_2}$$

Cette formule permet d'obtenir l'accélération obtenue sur une station isolée en utilisant Q processus parallèle au lieu d'un seul processus, lorsqu'au niveau des temps de calcul l'accélération est $a_C$.

$$x = \frac{\mu_2}{\mu_1 p_2} \quad \text{avec} \quad \mu_2 = 19{,}6 \quad \text{et} \quad p_2 = \frac{1}{\mu_1 \theta}, \quad \theta = 0{,}964 \text{ pour 1 processus}$$

donc $x = \mu_2 \theta = 18.$

Dans la formule précédente, on a la fonction suivante :

$$g(x) = \frac{(1+x)(1-x^Q)}{1-x^{Q+1}} = 1 + \frac{x - x^Q}{1+x^{Q+1}}$$

quand $x > 1$ alors $g(x) < 1$ et $\lim_{x \to \infty} g(x) = 1$

donc $a_R < \frac{a_C}{Q}$

Or l'on sait que Q processus parallèles peuvent généralement apporter au mieux une accélération égale à Q.

D'où $\boxed{a_R < 1}$

Ceci signifie que si l'on ne dispose que d'une seule station, il ne sert à rien d'utiliser des algorithmes parallèles, ce résultat était bien sûr attendu.

2. Cas du réseau

a) Validation des modèles

Nous avons validé les résultats obtenus grâce à l'Algorithme de Reiser en effectuant des mesures de temps de calcul et des temps de réponse de certains programmes pour un nombre variables de provessus sur une seule station. Nous avons regroupé ces résultats ci-dessous, les mesures correspondent à des moyennes obtenues sur plusieurs mesures.

On s'aperçoit qu'il y a une très bonne concordance entre les valeurs mesurées et les valeurs fournies par l'algorithme de Reiser.

| Nombre de processus | $T^{(C)}_{mesuré}$ | $T^{(R)}_{mesuré}$ | $f_{mesuré} = T^{(R)}/T^{(C)}$ | $f_{calculé}$ (algorithme de Reiser) |
|---|---|---|---|---|
| 1 | 65,18 | 68,47 | 1,0505 | 1,0524 |
| 2 | 65,40 | 136,54 | 2,0877 | 2,0481 |
| 3 | 35,92 | 114,39 | 3,1848 | 3,0442 |
| 4 | 29,64 | 133,01 | 4,48 | 4,0610 |
| 6 | 29,75 | 198,03 | 6,65 | 6,0477 |
| 12 | 10,25 | 164,93 | 16,11 | 12,003 |
| 13 | 13,192 | 506,11 | 38,6 | 43,66 |

b) Conséquence pour les algorithmes parallèles

On a vu que si l'on pose $f = T^{(R)}/T^{(C)}$ alors, pour des programmes parallèles

$$a_R = a_C \frac{f_{mono}}{f_{multi}}$$

Donc, pour qu'un algorithme parallèle soit le plus efficace, il faudra que le rapport $f_{mono}/f_{multi}$ soit le plus grand possible.

Il y a cependant une seconde condition. Il ne faut pas que $f_{multi}$ ou $f_{mono}$ soit très grand. En effet, si f est grand alors le temps de réponse $T^{(R)} = f.T^{(C)}$ est trop grand lui-même, ce que nous de désirons pas.

La première conséquence est, et c'est bien évident, pour obtenir un f petit donc des temps de réponse également petits, il faut faire fonctionner le système à faible charge. On voit d'ailleurs sur les courbes de l'Annexe V.C que le facteur f croît très rapidement en fonction du nombre de processus.

D'autre part, si l'on se place dans les conditions optimales, c'est-à-dire 1 processus par station, alors on observe d'une part que $f_{mono} \simeq 1$ (cas d'un processus sur une station), et que pour les programmes parallèles on a Q processus en parallèle sur Q stations, l'algorithme de Reiser donne $f_{multi} \simeq 1$ à 2 (voir Annexe V.F).

Dans ce cas optimal, on obtient que l'accélération en temps de réponse peut être équivalente à l'accélération en temps de calcul (voir Annexe V.G).

c) Mode d'utilisation recommandé du réseau

De ce qui précède, on peut déduire quelques recommandations pour l'emploi du réseau. Celui-ci ne sera efficace que si le nombre total de processus sur le réseau est de l'ordre du nombre de stations.

On peut alors envisager la situation suivante : un ensemble de stations fonctionnant chacune en monoposte, mais reliées au réseau. Au moment où l'une des stations a besoin d'une forte puissance de calcul, en utilisant des algorithmes parallèles, elle génère plusieurs processus, qui auront une brève durée de vie, mais à qui l'on confie une très forte priorité. Ainsi, pour quelques instants le réseau met toute ou partie de sa puissance à la disposition d'un utilisateur, puis il revient à sa situation antérieure.

Ceci représente l'utilisation la plus efficace du réseau, et pour une faible gêne des différents utilisateurs.

## ANNEXE AU CHAPITRE V

### ANNEXE V.A : Programme de l'algorithme de Reiser

```
      SUBROUTINE TEREPO(NS,KO,NL,AL,R,E,MU,TC)
      REAL AL(NL),R(NL),MU(NL)
      MU2=19.6
      Q=0.5
      DOFOR ITER=1,5
      IF (ITER.EQ.1) THEN
         U = FLOAT(KO)/FLOAT(NS+1)
      ELSE
         U = AL(1)
      ENDIF
      TETA=(U)**(-1.61)*0.964
      MU1=1./Q+1./TETA
      P2 = 1./(MU1*TETA)
      E1=1/(P2*NS)
      DOFOR I=1,NS
         MU(I)=MU1
         E(I)=E1
         AL(I)=0.
      ENDDO
      E(NS+1)=1
      MU(NS+1)=MU2
      AL(NS+1)=0.
      DOFOR J=1,KO
         DOFOR I=1,NS+1
            R(I)=(1.+AL(I))/MU(I)
         ENDDO
         S=0.
         DOFOR I=1,NS+1
              S=S+E(I)*R(I)
         ENDDO
         D=FLOAT(J)/S
         DOFOR I=1,NS+1
              AL(I)=D*E(I)*R(I)
         ENDDO
      ENDDO
      ENDDO
      TC = MU(1)*(R(NS+1)*P2+R(1))
      RETURN
      END
```

Paramètres : NS = nombre de stations
KO = nombre total de processus
NL = nombre de files d'attentes
AL = table des longueurs des files
R = table des temps de réponse moyens des files
E = table des taux de passage
MU = table des taux de service
TC = rapport temps de réponse sur temps de calcul = $\mu_1(R_D \cdot p_2)$

ANNEXE V.B

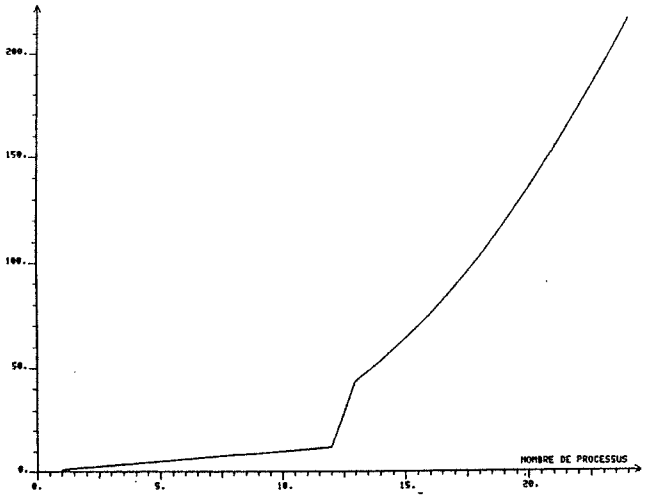
Cas d'une station isolée, évolution de $f = T^{(R)}/T^{(C)}$ en fonction de k, nombre total de processus :

| k | $f(k, N_S = 1)$ |
|---|---|
| 1 | 1,05237 |
| 2 | 2,0408 |
| 3 | 3,04423 |
| 4 | 4,06097 |
| 5 | 5,07669 |
| 6 | 6,04768 |
| 7 | 7,00754 |
| 8 | 8,00471 |
| 9 | 9,00245 |
| 10 | 10,0024 |
| 11 | 11,0028 |
| 12 | 12,0028 |
| 13 | 43,6623 |
| 14 | 53,1046 |
| 15 | 63,6184 |

ANNEXE V.C

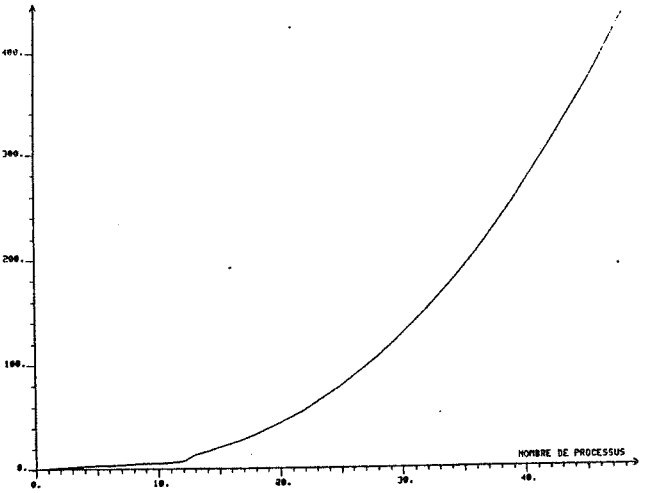
Courbes donnant f en fonction du nombre de processus et du nombre de stations sur le réseau.

$f(k,N_s=1)$



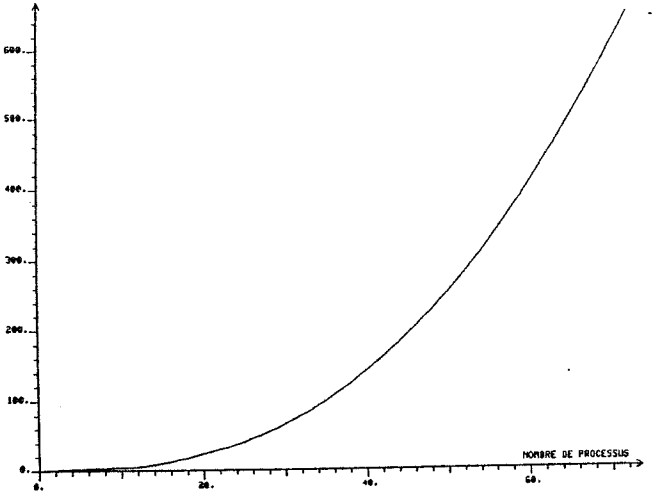

1 station

$f(k,N_s=2)$



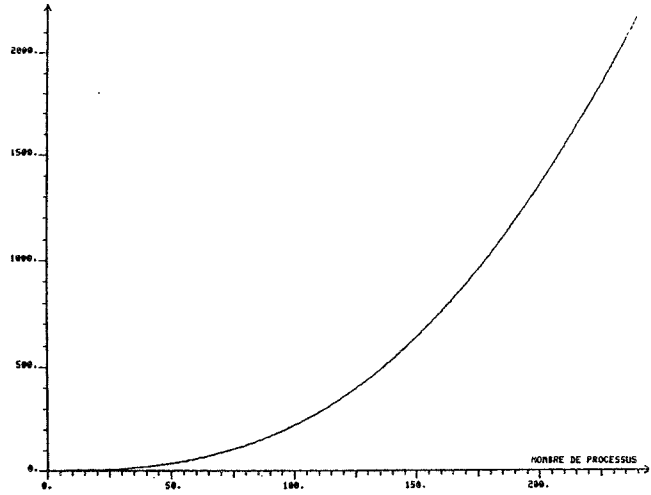

2 stations

$f(k,N_s=3)$




3 stations

$f(k,N_s=10)$




10 stations

ANNEXE V.D : Programme QNAP2 de modélisation du réseau.

```
/DECLARE/QUEUE CPU1,CPU2,DISK;
         REAL P1,P2,TETA,MU1,MU2,U,L,TC,GRAK;
         INTEGER N,I;
/STATION/NAME=CPU1;INIT=U;TRANSIT=DISK,P2,CPU1,P1;SERVICE=EXP(1/MU1);
/STATION/NAME=CPU2;INIT=U;TRANSIT=DISK,P2,CPU2,P1;SERVICE=EXP(1/MU1);
/STATION/NAME=DISK;INIT=GRAK;SERVICE=EXP(1/MU2);TRANSIT=CPU1,0.5,CPU2,0.5;
/CONTROL/TMAX=5000;
          OPTION=NRESULT;
/EXERC/BEGIN PRINT(" NB DE CLIENTS ?");
       N:=GET(INTEGER);
       I:=1;
       WHILE I<=101 DO BEGIN
        IF I=1 THEN BEGIN
          U:=REALINT(N)/3.;
          END
        ELSE
                    U:=L;
        GRAK:=REALINT(N)-2.*U;
       TETA:=0.964/(U**1.61);
       MU1:=1./TETA + 2.;
       MU2:=19.6;
       P2:=1./(TETA*MU1);
       P1:=1.-P2;
       SOLVE;
       L:=MCUSTNB(CPU1);
       I:=I+1;
      & PRINT("MU1",MU1,"MU2",MU2,"DISK",MRESPONSE(DISK),"CPU",
                                                    MRESPONSE(CPU1));
       & PRINT("P2",P2);
           TC:=MU1*(MRESPONSE(DISK)*P2+MRESPONSE(CPU1));
       PRINT("TC",TC);
       END;
       END;
/END/
```

ANNEXE V.E : Tableau comparatif de la simulation QNAP2 et l'algorithme de Reiser.

On compare les résultats entre QNAP et l'algorithme de Reiser dans le cas de deux stations connectées sur un réseau :

| Nombres de processus | f(Algorithme de Reiser) | f(QNAP2) | Δf/f en % |
|---|---|---|---|
| 2 | 1,531 | 1,513 | 1,2 % |
| 3 | 2,046 | 2,046 | 0 % |
| 4 | 2,560 | 2,533 | 1,1 % |
| 5 | 3,080 | 2,675 | 13 % |
| 6 | 3,603 | 3,558 | 1,3 % |
| 7 | 4,138 | 4,083 | 1,3 % |
| 8 | 4,660 | 4,784 | 2,7 % |
| 9 | 5,145 | 5,060 | 1,7 % |
| 10 | 5,589 | 5,612 | 0,4 % |
| 20 | 44,16 | 38,63 | 12 % |
| 30 | 150,7 | 116,4 | 22 % |

ANNEXE V.F

Variation de $f = T^{(R)}/T^{(C)}$ sur un réseau comportant $N_S$ stations et un nombre total de processus de $N_S$ également (1 processus par station) :

| $N_S$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| $f(k = N_S, N_S)$ | 1,0524 | 1,5316 | 1,700 | 1,784 | 1,8348 |
| $N_S$ | 6 | 7 | 8 | 9 | 10 |
| $f(k = N_S, N_S)$ | 1,869 | 1,894 | 1,913 | 1,928 | 1,938 |

ANNEXE V.G

Nous donnerons ici la variation de l'accélération en temps de réponse $a_R$ dans un cas représentatif. Il s'agit d'un programme parallèle idéal tel que l'accélération en temps de calcul $a_C = Q$ quand on utilise Q processus parallèles. On étudie alors $a_R$ en fonction de Q et de NPS nombre de processus par station du réseau (ce nombre représente la charge des stations du réseau.

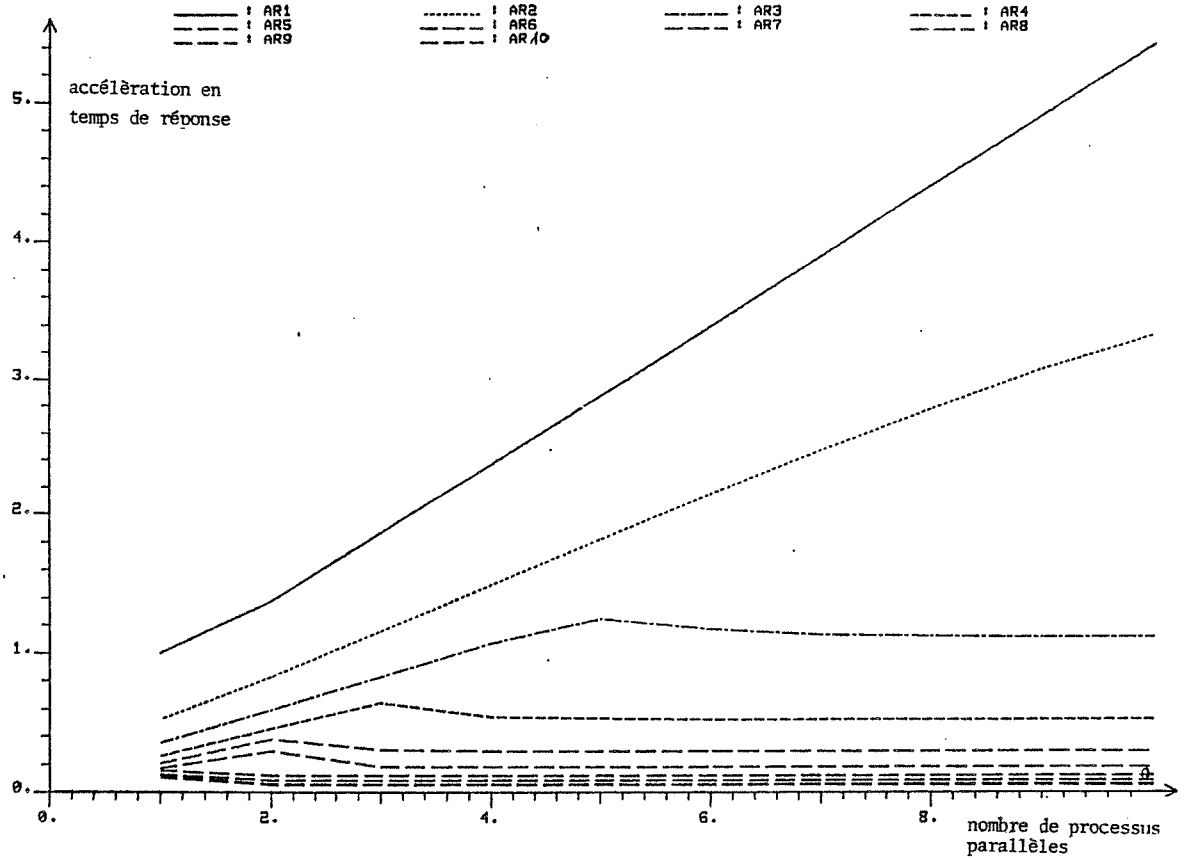
La courbe 1 donne l'évolution de $a_R$ en fonction de Q lorsque l'on fait varier NPS de 1 à 10 processus par station (multiprogrammation sur chaque station). Le programme de référence monoprocesseur s'exécute sur une station avec un processus unique sur cette station (monoprogrammation de la station de référence).

Cette courbe indique que les meilleures performances sont obtenues pour les charges minimales pour chaque station.

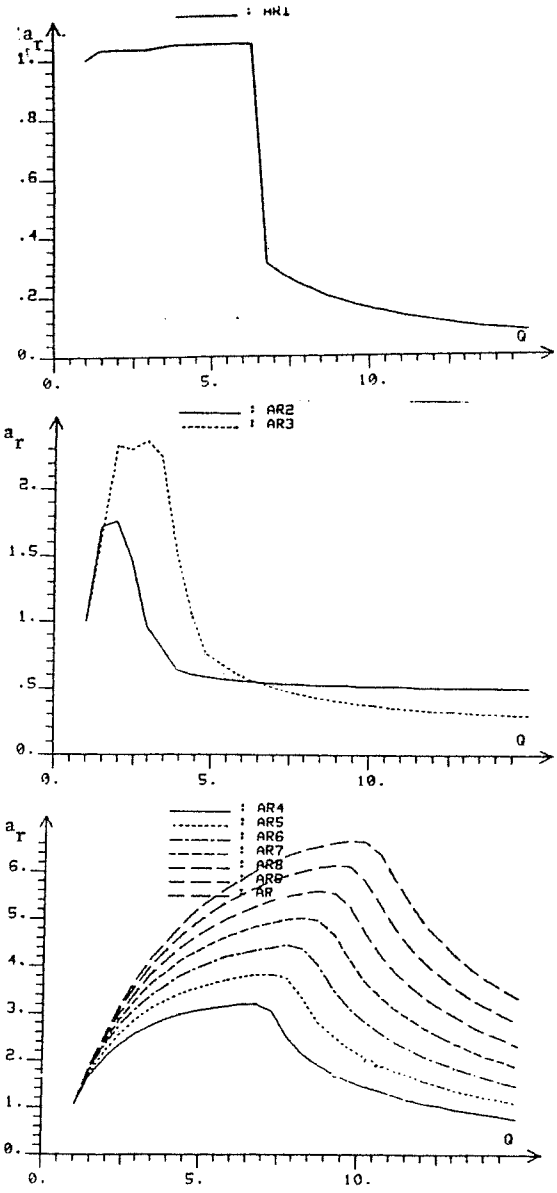
La courbe 2 représente l'évolution de $a_R$ en fonction du nombre de processus parallèles Q pour un nombre de stations $N_S$ allant de 1 à 10. Cette courbe confirme les précédentes en montrant que $a_R$ est maximum lorsque $Q = N_S$.

Courbe 1 : $$a_R = Q \frac{f_{mono}(k = 1,\ N_S = 1)}{f_{multi}(k = nQ,\ N_S = Q)} \qquad n = 1, 2, \ldots, 10$$

Courbe 2 : $$a_R = Q \frac{f_{mono}(k = 1,\ N_S = 1)}{f_{multi}(k = Q,\ N_S = n)} \qquad n = 1, 2, \ldots, 10 .$$




Courbe 1 . Chaque station est multiprogrammée / référence monoprogrammée.




Courbes 2 . Accélération ($a_R$) sur un réseau comportant Q processus au total, tous utilisés pour le parallèlisme. Chaque courbe correspond à une valeur de $N_S$ (nombre de stations).

# BIBLIOGRAPHIE

# BIBLIOGRAPHIE

## PREMIERE PARTIE

### CHAPITRE I . ARCHITECTURES PARALLELES


[1] THEIS D.

"Applications for Array Processors"
Computer Vol.16, Number 6, 1983

[2] GINSBERG M.

"Some Observations on Supercomputer Computional Environnements"
IMACS, Transaction on Scientific Computation Volume II, 1983

[3] AMELING W.

"Parallelism in Computer Architecture"
IMACS Transaction on Scientific Computation - Volume II, 1983

[4] ENSLOW P.

"Multiprocessors and other parallel systems"
Computer Architecture - Springer, Berlin, 1976

[5] SIEGEL H.

"A model of SIMD Machines and a comparison of various interconnection networks"
IEEE Trans. Comp. Vol.C-28, N° 12, 1979

[6] FLYNN M.

"Some computer organisation and their effectivness"
IEEE Transactions on Computers, Vol.C-21, 1972

[7] MAZARE G.

"Structures multimicroprocesseurs, problèmes de parallèlisme, définition et évaluation d'un système particulier"
Thèse de Docteur d'Etat es-Sciences, Grenoble, 1978

[8] BOSSAVIT A.

"L'ordinateur vectoriel et les méthodes numériques du calcul des structures"
Service Informatique et mathématiques Appliquées EdF, 1982

[9] HOCKNEY R.

"Characterization of Parallel Computers"
IMACS, Transaction on Scientific Computation, Volume II, 1983

[10] HOCKNE R.W. and JESSHOPE C.R.
"Parallel Computers", Adam Hilger Ltd, 1981

[11] Highly Parallel Computing
Numéro spécial de Computer, Janvier 1982.

## CHAPITRE II . ELEMENTS FINIS


[1] SABONNADIERE J.C., MEUNIER G., MOREL B.

"FLUX, a general interactive finites elements package for 2D electromagnetic fields"
IEEE Trans. on Mag., Vol. MAG-18, N° 2, 1982

[2] DHATT G, TOUZOT G.

"Une présentation de la méthode des éléments finis"
Presses de l'Université Laval - Quebec, 1981

[3] COULOMB J.L.

"Analyse tridimensionnelle des champs électriques et magnétiques par la méthode des éléments finis"
Thèse de Docteur d'Etat, Grenoble 1981

[4] GOLUB G-H, MEURANT G.

"Résolution numérique des grands systèmes linéaires"
Ecole d'Eté d'Analyse Numérique, 1983

[5] MEIJERINK J-A., A.van der VORST

"An Iterative solution method for linear systems of which the coefficient matrix is a symmetric M-Matrix"
Mathematics of computation, vol.31, n° 137, 1977

[6] KERSHAW D.S.

"The incomplete Cholesky Conjugate Gradient Method for the iterative solution of systems of linear equations"
Journal of Computational Physics, n° 26, 1978

[7] DURAND E.

"Solution numérique des équations algébriques"
Tomme II, 1972

[8] RODRIGUE G.

"The incomplete Cholesky conjugate gradient method for the Star - Lawrence Livermore Laboratory" 1977

[9] KUTHILL E.

"Several strategies for reducing the band width of matrices"
Proc. Conf. at IBM, 1979

[10] ZINKIEWICZ O.C.

"The finite element method in engineering", 1971

[11] KOUYOUMDJIAN A.

"Contribution à l'étude et à la résolution des grands systèmes linéaires creux"
Stage de DEA, Grenoble, 1982

[12] COULOMB J.L., MASSE Ph., ANCELLE B., MEUNIER G., SABONNADIERE J.C.

"Computer methods for electrical and magnetic devices designed by field analysis"
IEEE Transactions on magnetics, vol.MAG-15, N° 6, 1979

CHAPITRE III et CHAPITRE IV . ALGORITHMES PARALLELES


[1] SAMEH A.H, CHEN S.C., KUCK D.J.

"Parallel Poisson and biharmonic solvers"
Springer Verlag, Computing 17, 1976

[2] SAAD Y., SAMEH A., SAYLOR P.

"Parallel interative methods for elliptic difference equations"
Yale University, 1981

[3] SAMEH A.

"Solving the linear least squares problem on a linear array of processors"
Purdue workshop on algorithmically specialized computer organizations, 1982

[4] KUCK D.J.

"High speed computer and algorithm organization"
Academic Press, 1977

[5] LICHNEWSKY A.

"Sur la résolution de systèmes linéaires issus de la méthode des éléments finis par une machine multiprocesseurs"
INRIA, 1982

[6] JORDAN T.L.

"A performance evaluation of linear algebra software in parallel architectures"
Los Alamos Scientific laboratory, 1978

[7] PLATEAU B., STAPHYLOPATIS A.

"Modelling of the parallel resolution of a numerical problem on a locally distributed computing system"
Laboratoire de Recherche en Informatique, Université de Paris-Sud, 1982

[8] RODRIGUE G., WOLITZER D.

"Preconditionning by incomplete block cyclic reduction"
Lawrence Livermore Laboratory, 1982

[9] CHATELIN F., MIRANKER W.

"Aggregation/Disaggregation for eigenvalue problems"
IBM Research Division, 1982

[10] CHATELIN F.

"Acceleration by aggregation of successive approximation methods"
IBM Research Division, 1980

[11] BOERI F., AUGUIN M., JALBY M.

"Etude de la construction d'une matrix de rigidité sur une architecture multiprocesseur"
EdF, Bulletin de la Direction des Etudes et Recherches, série C, Mathématiques, Informatique, 1983

[12] BERGER P., BROUAYE P.

"Méthodes d'assemblage et de factorisation parallèle pour approximations par éléments finis sur multiprocesseurs MIMD à mémoires locales"
EdF, Bulletin de la Direction des Etudes et Recherches, série C, 1983

[13] PION G., BECKER M., CALLEGHER E., FOURNIER R.

"Is parallelism useful in order to improve response time of a finite element method ?"
Proc. of COMPUMAG, Gênes, 1983

[14] PION G.

"Amélioration des performances du logiciel FLUX à l'aide du parallèlisme"
Stage de DEA - Grenoble, 1982

[15] PION G.

"Définition d'algorithmes parallèles pour la méthode des éléments finis et évaluation de leurs performances"
Journée Jeunes Réalisateurs de l'IEEE, Grenoble 1983

[16] ARNOLD C.P., PARR M.I., DEWE M.B.

"An efficient parallel algorithm for the solution of large sparse matrix equations"
IEEE trans. on Computer, vol.C-32, N° 3, 1983

CHAPITRE V . EVALUATION DES PERFORMANCES

[[1] BASKETT F., CHANDY K.M., MUNTZ R.R., PALAGIOS F.G.

Open, Closed and Mixed Networks of queues with different classes of customers"
JACM 22, 1975

[2] REISER M., LAVENBERG S.S.

"Mean Value Analysis of Closed multichain queuein networks"
JACM 27, 1980

[3] COURTOIS P.J.

"Decomposability" Academic Press, 1977

[4] BADEL M., CHANDESRIS D., GUILLEMAUD J.J., POTIER D., SAINTOYANT P.Y., VEDAN M.

"QNAP 2, Reference Manual" CII-HB, 1981

[5] VERAN M., BECKER M.

"Evaluation des performances des systèmes informatiques"
Cours de DEA, Grenoble 1982

[6] Special Issue on Performance Evaluation of Multiple Processor Systems

IEEE Trans. on Computers, Vol.C-32, N° 1, January 1983

[7] MARSAN M.A., BALBO G. and CONTE G.

"Comparative Performance Analysis of Single Bus Multiprocessor Architectures"
IEEE Trans. on Computers, Vol.C-31, N° 12, December 1982.

[8] VANTILBORG H.

"The error of Aggregation in Decomposable Systems"
Rapport interne du MBLE Research Laboratory - Bruxelles, Avril 1982

[9] LABETOULLE J

"Etude analytique du réseau DANUBE"
Rapport INRIA - Projet Pilote KAYAK - Rel.2.518, Mars 1980

[10] FAYOLLE G., GELENBE E. and LABETOULLE J.

"Stability and Optimal Control of the Packet Switching Broodcast Channel"
JACM vol.24, N° 3, July 1977 pp.375-386

[11] BECKER M.

"Validité des simulations de files d'attente"
Thèse de Doctorat d'Etat, Université de Paris VI, Juin 1976.

# DEUXIEME PARTIE

* * * * *

## METHODES MULTIGRILLES :

### DEVELOPPEMENT A DES PROBLÈMES D'ÉLÉMENTS FINIS ET EXTENSION AU MAILLAGE ADAPTATIF.

# INTRODUCTION

Dans la première partie de ce rapport, nous avons étudié des algorithmes nouveaux pour la résolution des systèmes linéaires qui permettent l'utilisation de calculateurs parallèles.

Pour établir ces algorithmes, nous avons défini des méthodes que l'on peut considérer comme des méthodes d'agrégation. En effet, il s'agissait dans un premier temps de créer des relations privilégiées entre certaines inconnues pour mener des calculs sur ce groupe indépendamment (et en parallèle) avec les autres groupes d'inconnues. Suivant alors une étape de désagrégation permettant de corriger les distorsions introduites par les relations privilégiées créées à l'étape précédente.

Ce principe d'agrégation / désagrégation a été appliqué à des méthodes plus ou moins classiques (Cholesky, ICCG) qui n'étaient pas conçues initialement pour en tirer parti.

Il nous est alors apparu intéressant d'étudier des méthodes utilisant explicitement le principe d'agrégation et si possible tenant compte de l'origine "éléments finis" des systèmes que nous cherchons à résoudre.

Dans cette optique, nous avons noté le développement grandissant accordé aux méthodes multigrilles. Ces méthodes correspondent très bien aux objectifs de notre recherche car elles présentent quelques caractéristiques importantes telles que des diminutions de temps de calcul (en accélérant la vitesse de convergence des méthodes itératives et en réduisant le coût de calcul de chaque itération), des possibilités de parallèlisme, une relation étroite avec l'origine du système (éléments finis ou différences finies) et des extensions possibles très diverses (maillage adaptatif, calcul localisé,...).

# CHAPITRE

# 6

# CHAPITRE IV . GENERALITES SUR LES METHODES MULTIGRILLES

## REMARQUES PRELIMINAIRES

Les méthodes multigrilles ont été essentiellement introduites pour la résolution de problèmes de différences finies pour lesquels la notion de grille est transparente. Pour les généralités et la présentation de la méthode nous conserverons cette approche issue des différences finies. Ultérieurement nous étudierons son extension au cas des éléments finis ou à des outils plus généraux de résolution.

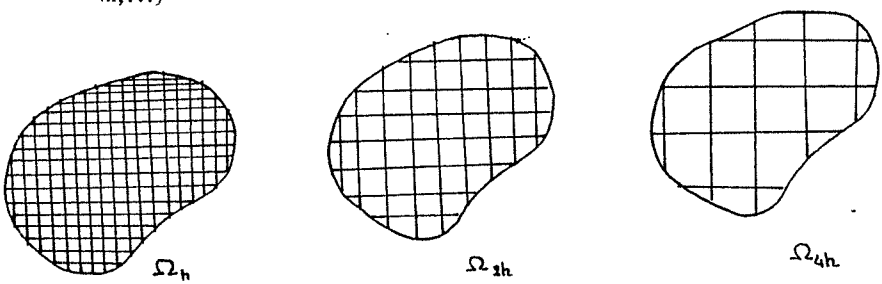
## A . PRINCIPES ET OBJECTIFS DE LA METHODE

### I . IDEE GENERALE

Nous cherchons à résoudre le système

$$L_h u_h = f_h \quad (\Omega_h)$$

et l'on suppose disposer simultanément d'un ensemble d'approximations discrètes sur un ensemble de grilles (caractérisées par la taille des mailles : h, 2h, 4h,...)




Des approximations avec des erreurs "lissées" (nous verrons plus loin le sens exact de ce terme) peuvent être obtenues efficacement en utilisant des méthodes de relaxation. Du fait du lissage de l'erreur, la correction de ces approximations peut être calculée sur des grilles "grossières" (c'est-à-dire pour lesquelles h est grand.) Cette idée de base peut être employée récursivement avec des grilles de plus en plus grossières.

Le but est donc d'obtenir la précision la meilleure sur la grille la plus fine en travaillant principalement sur les grilles les plus grossières.
Ceci conduit à des méthodes "(asymptotiquement) optimale ", c'est-à-dire pour lesquelles le travail nécessaire pour obtenir une précision désirée est proportionnel au nombre d'inconnues.
On peut alors distinguer trois éléments :

(1) lissage de l'erreur par des méthodes de relaxation
(2) calcul de la correction sur grilles grossières, et application récursive
(3) éventuellement une combinaison avec des itérations "emboitées" (c'est-à-dire où l'on utilise la solution sur la grille grossière comme approximation initiale sur la grille fine).

## II . CONDITIONS D'UTILISATION DE METHODES MULTIGRILLES

Le point de départ des méthodes multigrilles (ou plus généralement des Techniques Adaptatives à Niveau Multiple), mais aussi leur but ultime ont été énoncés par Achi Brandt par la règle suivante :

La somme de travail en calcul doit être proportionnelle au nombre de changements physiques du système calculé.

C'est-à-dire, si une méthode nécessite de nombreux calculs pour obtenir des changements physiques faibles, alors il doit exister une méthode meilleure pour parvenir au même but. De telles constatations se produisent lorsque des systèmes itératifs convergent lentement, ou plus généralement, lorsque l'utilisation de grilles trop fines conduisent à des problèmes où l'échelle de temps ou d'espace de la discrétisation sont beaucoup plus petits que l'évolution des phénomènes physiques.

Pour de tels problèmes, les techniques multigrilles peuvent être utiles. Le problème peut provenir par exemple de l'existence de plusieurs composantes de la solution à des échelles différentes, en conflit les unes avec les autres. En employant plusieurs échelles de discrétisation, des techniques à plusieurs niveaux peuvent éviter ces conflits et accélérer les convergences.

Le développement principal des techniques à plusieurs niveaux s'est essentiellement limité à leur utilisation comme outil de résolution rapide pour les problèmes de conditions aux limites (état stationnaire ou schéma implicite pour les évolutions dans le temps).

En plus de cette utilisation, les techniques multigrilles peuvent être très utiles à d'autres applications. Elles peuvent fournir des outils de maillage adaptatif performant pour les problèmes (conditions aux limites ou évolution dans le temps) pour lesquels différentes échelles de discrétisation sont nécessaires suivant les parties du domaine d'étude. Elles peuvent permettre de résoudre le conflit entre précision et stabilité dans le cas de problèmes non elliptiques.

Les méthodes multigrilles peuvent être étendues à des problèmes non issus de différences finies pour être utilisées comme la plupart des outils classiques de résolution. La notion de grille n'a plus alors de support physique, elle représente le graphe de la matrice du système à résoudre.

Des calculs parallèles peuvent être réalisés. Si l'on dispose d'un réseau de processeurs fortement couplés, on pourra effectuer certaines opérations associées à chaque noeud simultanément. Si l'on ne dispose que de quelques processeurs faiblement couplés, on pourra alors diviser le domaine d'étude en plusieurs sous-domaines où seront menés séparément les calculs.

Dans les chapitres suivants, nous développerons de façon plus détaillée ces principes généraux. On peut néanmoins s'apercevoir après cette brève description de l'intérêt que peuvent représenter les méthodes multigrilles.

Avant de décrire l'aspect mathématique de ces méthodes, nous en donnerons un rapide historique.

## B . HISTORIQUE DE LA METHODE

L'idée du multigrille et les algorithmes de base ont été développés dans les années 1960 à 1970 par Fedorenko et Bakhvalov. Fedorenko étudia les problèmes de convergence pour l'équation de Poisson restreinte au carré unité.
Bekhvalov s'interessa au cas plus général des problèmes elliptiques du second ordre avec coefficients variables, toujours dans un domaine restreint au carré unité. Il fut également le premier à signaler la possibilité de combiner des méthodes multigrilles avec des itérations emboitées. Bien que les études de Fedorenko et Bakhvalov aient montré l'optimalité de l'approche multigrille, leur efficacité ne fut reconnue qu'après les travaux d'Achi Brandt.

C'est en 1970, à la suite d'études sur le maillage adaptatif et ses rapports avec la définition d'outils de résolution rapides pour les systèmes linéaires, que Brandt montra toute l'efficacité des méthodes multigrilles. Il définit alors des méthodes multigrilles pour des problèmes non linéaires (FAS - Full Approximation Storage) et des techniques adaptatives (MLAT - Multi Level Adaptative Techniques), il étudia les problèmes soulevés par des domaines plus complexes que le carré unité, par des raffinements locaux des maillages, par l'application des itérations emboitées (FMG - Full Multi Grid). Il définit enfin un outil d'étude théorique et d'optimisation des méthodes : l'analyse de Fourier locale ("Local Fourier Analysis").

En 1975, Nicolaïdes présenta un ensemble d'idées de base pour l'utilisation de méthodes multigrilles à des problèmes d'éléments finis.

Pendant les années 1975-1976, Hackbusch développe des éléments fondamentaux de la méthode.
Il définit un autre outil d'étude et d'optimisation : l'analyse du problème modèle ("Model Problem Analysis"), il étudia le cas de domaines non rectangulaires et des problèmes non linéaires. Il peut ainsi bâtir une théorie générale de la convergence des méthodes multigrilles.

Depuis 1977, ces méthodes ont commencé à être mieux acceptées et donc plus étudiées. Des recherches pratiques sont menées de façon intensive pour l'application aux éléments finis, notamment en dynamique des fluides qui semble pour l'instant être le domaine d'application le plus propice aux méthodes multigrilles. Des extensions de la méthode sont menées pour la résolution de systèmes linéaires généraux sans support physique.

En novembre 1981, une conférence sur les méthodes multigrilles fut organisée à Cologne-Porz par Wolfgang Hackbush et Ubrich Trottenberg. Cette conférence permit de réunir les principaux spécialistes de ces méthodes pour faire le point de l'état d'avancement de leurs recherches. A cette occasion, la grande diversité des possibilités d'application de ces méthodes fut mise en évidence, mais de nombreux intervenants purent faire remarquer que le sujet était loin d'être entièrement traité, notamment au niveau des tests sur les méthodes pour des problèmes plus complexes et plus généraux. On notera à ce propos, les travaux de Wesseling pour la réalisation de bibliothèques de programmes multigrilles (pour l'instant restreints aux problèmes issus de différences finies).

# CHAPITRE

# 7

## CHAPITRE VII . PRESENTATION DETAILLEE DES METHODES MULTIGRILLES

A . ELEMENTS THEORIQUES

I . METHODE A DEUX GRILLES

1. Approximation d'une solution par itération sur le résidu
2. Correction par grille grossière
3. Comparaison entre relaxation et correction
4. Structure d'une méthode à deux grilles

II . METHODE MULTIGRILLE

1. Principe et représentation
2. Définition récursive d'un cycle multigrille complet
3. Efficacité de la méthode

III . L'ALGORITHME FULL MULTI GRID (FMG)

B . METHODE MULTIGRILLE POUR DES PROBLEMES NON LINEAIRES

I . APPLICATION DIRECTE

II . L'ALGORITHME FULL APPROXIMATION SCHEME (FAS)

III . DUALITE DE LA METHODE FAS

C . ANALYSE DES METHODES MULTIGRILLES

I . COMPOSANTS D'UNE METHODE MULTIGRILLE

II . ANALYSE DU COMPORTEMENT D'UNE METHODE.

# CHAPITRE VII . PRESENTATION DETAILLEE DES METHODES MULTIGRILLES

## A . ELEMENTS THEORIQUES

### I . METHODE A DEUX GRILLES

#### 1. Approximation d'une solution par itération sur le résidu

Soit un problème linéaire elliptique sur une domaine $D_h$ découpé en éléments finis caractérisés par une dimension h, et représenté par :

$$L_h u_h = f_h \qquad (D_h) \tag{1.1}$$

où $L_h$ est la matrice d'un opérateur linéaire, $f_h$ représente les contraintes (sources et conditions aux limites) et $u_h$ les valeurs nodales aux $N_h$ noeuds du domaine (encore appelé grille).

A la $j^{ème}$ itération du processus de résolution, $u_h^j$ est la solution approchée et $v_h^j$ l'erreur : $v_h^j = u_h - u_h^j$ (1.2)

et $d_h^j$ le résidu : $d_h^j = f_h - L_h u_h^j$ (1.3)

l'équation relative au résidu est :

$$L_h v_h^j = d_h^j \tag{1.4}$$

Pour le processus itératif, on remplace $L_h$ par une matrice "plus simple" $\hat{L}_h$ telle que $\hat{L}_h^{-1}$ existe et soit "facilement" calculée. La solution $v_h^j$ de :

$$\hat{L}_h \, \hat{v}_h^j = d_h^j \tag{1.5}$$

donne une nouvelle approximation :

$$u_h^{j+1} = u_h^j + \hat{v}_h^j \tag{1.6}$$

L'opérateur itératif est $I_h - \hat{L}_h^{-1} L_h$, où $I_h$ est la matrice identité.

On a alors : $\hat{v}_h^{j+1} = (I_h - \hat{L}_h^{-1} L_h) \hat{v}_h^j$ (1.7)

et $d_h^{j+1} = L_h(I_h - \hat{L}_h^{-1} L_h) L_h^{-1} = (I_h - L_h \hat{L}_h^{-1}) d_h^j$ (1.8)

On sait alors que les propriétés de convergence asymptotique du processus itératif précédent sont caractérisées par le rayon spectral (ou facteur de convergence asymptotique) :

$$\rho(I_h - \hat{L}_h^{-1} L_h) = \max \{|\lambda| : \lambda \text{ valeur propre de } I_h - \hat{L}_h^{-1} L_h\}$$

Si une norme a été définie, alors $\| I_h - L_h^{-1} L_h \|$ représente le facteur de réduction de l'erreur et $\| I_h - L_h L_h^{-1} \|$ le facteur de réduction du résidu par itération.

## 2. Correction par "grille grossière"

Supposons que l'on puisse définir un maillage "grossier" extrait du maillage précédent, comportant moins d'éléments et de noeuds. Ce maillage sera caractérisé par un paramètre $H > h$.

On peut choisir, dans la méthode précédente, la matrice $L_H$ représentant le système pour le maillage grossier, comme matrice $\hat{L}_h$. L'équation (1.5) devient

$$L_H \hat{v}_H^j = d_H^j \tag{2.1}$$

où $v_h^j$ et $d_h^j$ sont des vecteurs relatifs au maillage grossier. Nous définissons alors deux opérations de transfert $I_h^H$ et $I_H^h$ :

la restriction : $d_h^j = I_h^H d_h^j$ (2.2)

l'interpolation : $\hat{v}_h^j = I_H^h v_H^j$ (2.3)

La relation par grille grossière se traduit par le schéma de la Figure 2.1 :

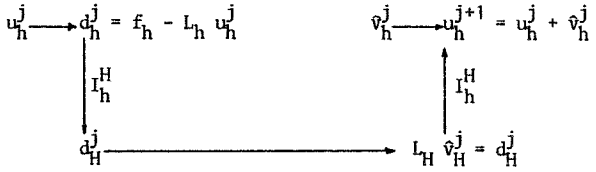



Figure 2.1

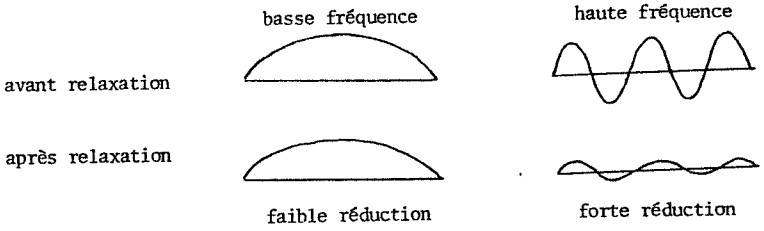
## 3. Comparaison entre relaxation et correction

Que l'on choisisse des déterminations particulières pour $\hat{L}_h$ (pour les méthodes de Jacobi, Gauss - Seidel,...) ou que l'on choisisse une correction $L_H$ par grille grossière, la convergence est généralement peu satisfaisante. Mais on peut montrer que ces deux approches ont des propriétés différentes qui peuvent être exploitées pour donner des combinaisons très efficaces.

### a) Méthodes de Relaxation

La convergence des méthodes de relaxation appliquées aux équations elliptiques est connue pour devenir mauvaise lorsque $h \to 0$. Pour la méthode de Jacobi ou de Gauss Seidel on obtient un comportement en $1-0(h^2)$, et pour SOR on obtient $1-0(h)$.
On peut analyser les propriétés de réduction de l'erreur en décomposant l'erreur $v_h^j$ en série de Fourier. On distingue les composantes lissées (basses fréquences) et les composantes non lissées (haute fréquence). Des études ont montré que les composantes lissées de l'erreur étaient responsables de la lenteur de la convergence des méthodes de relaxation.

Par contre, des méthodes de relaxation convenables sont très efficaces pour lisser l'erreur, c'est-à-dire pour réduire les composantes de haute fréquence de l'erreur. Les propriétés de lissage d'une méthode sont représentées par un facteur de lissage qui représente le facteur (le plus grand, donc le pire) par lequel les composantes de hautes fréquences sont réduites (il est équivalent au rayon spectral mais limité aux hautes fréquences).

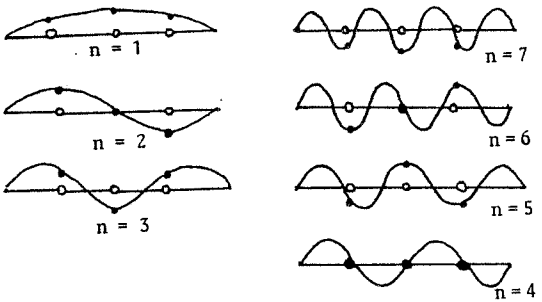


b) Comportement de la correction par grille grossière

La méthode de correction par grille grossière n'est pas en elle même convergente :

$$\rho(I_h - I_H^h L_H^{-1} I_h^H L_h) \geq 1$$

L'équation du résidu de la grille grossière (2.1) n'est pas en général une bonne bonne approximation de l'équation originale (1.4). En particulier, les composantes de $v_h^j$ qui ne peuvent pas être représentées sur la grille grossière $D_H$ (qui sont, pour ainsi dire, "invisibles" sur $D_H$) ne peuvent pas être réduites sur cette grille. La figure suivante illustre ce fait.




Composantes visibles sur $D_H$ Composantes invisibles sur $D_H$

Figure 3.1 : $\sin(n\pi x)$ : basses (n-1,2,3) et hautes (n=4,5,6,7) composantes de la fréquence pour h=1/8 et H = 1/4

De cela, nous pouvons conclure que l'approximation sur la grille grossière n'est valable que si l'erreur originale $v_h^j$ est lissée, c'est-à-dire que les composantes de hautes fréquences ont une amplitude faible par rapport aux composantes de basses fréquences. Cette condition est justement obtenue par la méthode de relaxation sur le maillage fin.

## 4. Structure d'une méthode à deux grilles

Supposons que nous utilisons une méthode itérative de résolution de système linéaire notée R (par exemple Jacobi, Gradient-Conjugué, Newton,...).

Nous noterons : $\bar{w}_h = R^{\nu}(w_h, L_h, f_h)$ (4.1)

où $\bar{w}_h$ est le résultat de $\nu$ itérations pour résoudre le système (1.1) en utilisant la méthode R avec $\bar{w}_h$ comme valeur initiale.

En conséquence du paragraphe précédent, il apparaît judicieux de combiner la méthode de relaxation et la correction par grille grossière. Nous obtenons alors la méthode itérative à deux grilles (h, H). Chaque itération de cette méthode est composée de parties de lissage et de parties de correction. La méthode est alors la suivante :

a) Lissage (1ère partie)

Calculer $\bar{u}_h^j$ par application de $\nu_1$ intérations de la méthode de relaxation R

$$\bar{u}_h^j = R^{\nu_1}(u_h^j, L_h, f_h)$$

b) Correction par grille grossière

- Calculer le résidu $\bar{d}_h^j = f_h - L_h \bar{u}_h^j$
- Restriction du résidu $\bar{d}_h^j = I_H^h \bar{d}_H^j$
- Résoudre $L_H \hat{v}_H^j = \bar{d}_H^j$
- Interpolation $\hat{v}_h^j = I_H^h \hat{v}_H^j$
- Calculer l'approximation corrigée $\bar{u}_h^j + \hat{v}_h^j$

c) Lissage (2ème partie)

- Calculer $u_h^{j+1}$ en application $\nu_2$ itérations de R à $\bar{u}_h^j + \hat{v}_h^j$

$$u_h^{j+1} = R^{\nu_2}(\bar{u}_h^j + \hat{v}_h^j, L_h, f_h)$$

Ceci peut être illustré par la Figure 4.1

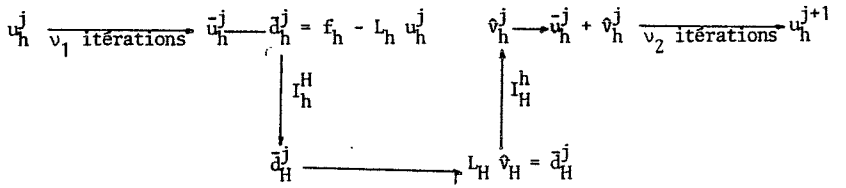



Figure 4.1

L'opérateur de la méthode à deux grilles est alors :

$$M_h^H = S_h^{\nu 2} K_h^H S_h^{\nu 1} \quad \text{avec} \quad K_h^H = I_h - I_H^h L_H^{-1} I_h^H L_h \tag{4.2}$$

Ici, $S_h$ représente l'opérateur itératif correspondant à la méthode de relaxation R que nous utilisons.

Un certain nombre de choix doivent être faits :

- la méthode de relaxation $S_h$
- les nombres d'itération $\nu_1$ et $\nu_2$
- la grille grossière $\Omega_H$
- l'opérateur de restriction $I_h^H$
- l'opérateur sur la grille grossière $L_H$
- l'opérateur d'interpolation $I_H^h$

Le choix de ces composantes a une influence importante sur l'efficacité de la méthode. De plus, la construction d'un algorithme optimal ne semble suivre aucune règle précise.

## II . METHODE MULTI GRILLE

### 1. Principe et représentation

L'idée de la méthode Multi Grille provient de l'observation que pour la méthode à deux grilles il n'est pas nécessaire de résoudre exactement sur la grille grossière l'équation du résidu :

$$L_H \, \varphi_H^j = d_H^j$$

Sans perte importante de convergence, on peut remplacer $\varphi_H^j$ par une approximation appropriée. En particulier on peut appliquer à nouveau une méthode à deux grilles à cette équation en définissant une grille encore plus grossière que $D_H$. Si le facteur de convergence de cette méthode est suffisamment petit, seul un petit nombre $\gamma$ d'itérations sera nécessaire.

Nous représenterons la structure d'une itération (cycle) d'une méthode multi-grille à l'aide de schémas identiques à ceux de la figure (5.1).

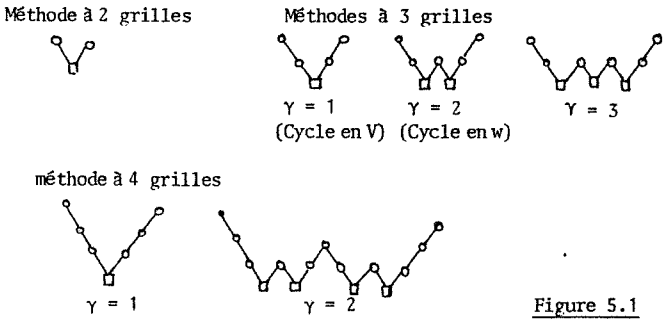



Figure 5.1

## 2. Définition récursive d'un cycle Multi Grille Complet

Notons $G(D_h)$ l'espace des fonctions sur la grille $D_h$. Pour décrire les méthodes multigrilles nous utiliserons un ensemble de grilles notées $D_{h_l}$ de plus en plus fines ($l = 0,1,2,\ldots$), et pour plus de facilité, nous remplaçons $h_l$ par $l$.

Nous aurons les opérations suivantes :

$$L_l : G(D_l) \to G(D_l) \qquad S_l : G(D_l) \to G(D_l)$$

$$I_l^{l-1} : G(D_l) \to G(D_{l-1}) \qquad I_{l-1}^{l} : G(D_{l-1}) \to G(D_l)$$

l'équation pour chaque grille est $L_l\, u_l = f_l \ (D_l)$ (6.1)

$S_l$ est l'opérateur itératif correspondant à la méthode de relaxation R choisie.

On peut alors définir récursivement la méthode multigrille :

Si $l = 1$ : méthode à deux grilles avec $D_0$ et $D_1$ au lieu de $D_H$ et $D_h$.
Si $l > 1$ :

a) Lissage (1ère partie)

- Calculer $\bar{u}_l^j$ par $\nu_1$ étapes de lissage de $u_l^j$ : $\bar{u}_l^j = R^{\nu_1}(u_l^j, L_l, f_l)$

b) Correction par grille grossière

- Calculer le résidu $\bar{d}_l^j = f_l - L_l\, \bar{u}_l^j$
- Restriction du résidu $\bar{d}_{l-1}^j = I_l^{l-1}\, \bar{d}_l^j$
- Calculer une solution approchée $\hat{v}_{l-1}^j$ de l'équation du résidu sur $D_{l-1}$

$$L_{l-1}\, \hat{v}_{l-1}^j = \bar{d}_{l-1}^j \qquad (6.2)$$

en effectuant $\gamma$ itérations de la méthode à l grilles (en utilisant $D_{l-1}$, $D_{l-2},\ldots, D_0$) appliquée à (6.2) :

- Interpolation de la correction : $\hat{v}_l^j = I_{l-1}^{l}\, \hat{v}_{l-1}^j$
- Calculer l'approximation corrigée de la solution sur $D_l$ : $\bar{u}_l^j + \hat{v}_l^j$

c) Lissage (2ème partie)

Calculer $u_l^{j+1}$ par $\nu_2$ étapes de relaxation sur $\bar{u}_l^j + \hat{v}_l^j$

$$u_l^{j+1} = R^{\nu_2}(\bar{u}_l^j + \hat{v}_l^j\ , L_l, f_l)$$

L'opérateur $M_1$ représentant le cycle complet est alors donné par la relation récurrente suivante :

$$M_\ell = S_\ell^{\nu_2} (I_\ell - I_0^1 L_0^{-1} I_\ell^0 L_\ell) S_\ell^{\nu_1}$$

$$M_{k+1} = S_{k+1}^{\nu_2} (I_{k+1} - I_k^{k+1} (I_k - M_k^\gamma) L_k^{-1} I_{k+1}^k L_{k+1}) S_{k+1}^{\nu_1} \quad (k = 1,\ldots,1-1)$$

On peut montrer (cf. Hackbush, Stüben) que si une méthode à deux grilles est convergente, alors la méthode multigrille l'est au moins aussi pour $\gamma = 2$ (cycle en W). On peut également assurer la convergence d'un cycle en V ($\gamma = 1$).

## 3. Efficacité de la méthode

De la définition récursive d'un cycle multigrille, il apparaît que la quantité de calculs $W_1$ par cycle sur $D_1$ est donnée par :

$$W_1 = W_1^0 + W_0, \quad W_{k+1} = W_{k+1}^k + \gamma_k W_k \quad (k = 1,\ldots 1-1)$$

où $W_{k+1}^k$ représente les calculs du cycle à deux grilles $(h_{k+1}, h_k)$ sauf la partie de résolution de l'équation au résidu sur $D_k$. Si $\gamma$ est indépendant de k, on obtient :

$$W_1 = \sum_{k=1}^{1} \gamma^{1-k} W_k^{k-1} + \gamma^{1-1} W_0 \qquad (7.1)$$

Si l'on suppose qu'à chaque grille, le nombre de noeuds $N_k$ est tel que $N_k = 4 N_{k=1}$

on aura $W_k^{k-1} \leq C N_k$ (7.2)

où C est une constante majorant la quantité de calcul sur une grille (relaxation, correction,...)

De (7.1) on obtient alors la quantité totale de calcul $W_1$ pour un cycle complet :

$$W_1 \leq \begin{cases} \frac{4}{3} C N_1 & (\gamma = 1) \\ 2 C N_1 & (\gamma = 2) \\ 4 C N_1 & (\gamma = 3) \\ 0(N_1 \log N_1) & (\gamma = 4) \end{cases}$$

Ces relations montrent (pour un choix particulier de grilles) la grande efficacité des méthodes multigrilles pour $\gamma \leq 3$, et tout particulièrement des méthodes en V ($\gamma = 1$).

Dans le cas d'un choix plus général des grilles :

$$N_k = \tau\, N_{k-1} \quad \text{avec} \quad \tau > 1$$

on obtient :

$$W_1 \leq \begin{cases} \frac{\tau}{\tau-\gamma}\, C\, N_1 & (\gamma < \tau) \\ 0\ (N_1 \log N_1) & (\gamma = \tau) \end{cases}$$

## III . L'ALGORITHME "FULL MULTIGRID" (FMD)

L'algorithme Multigrille Complet ou Full Multigrid, que l'on doit à Brandt, présente une approche différente, et qui peut être intéressante pour l'étude du maillage adaptatif.

Le point de départ de cet algorithme est l'utilisation d'itérations emboitées : on choisit une valeur initiale sur la grille la plus grossière $D_0$ que l'on étend par interpolation à la grille $D_1$. On applique alors la méthode à deux grilles sur $(D_0, D_1)$ pour trouver une meilleure approximation sur $D_1$. Puis l'on recommence l'ensemble de ce processus pour les grilles $D_2$, $D_3$,... $D_1$.

Ce principe est illustré par la Figure (8.1)

| Niveau | Taille de la grille |
|---|---|
| 0 | $k_0$ |
| 1 | $h_1 = \frac{h_0}{2}$ |
| 2 | $h_2 = \frac{h_1}{4}$ |
| . . . | |
| $\ell$-1 | $h_{\ell-1} = 2\, h_\ell$ |
| | $h_\ell$ |

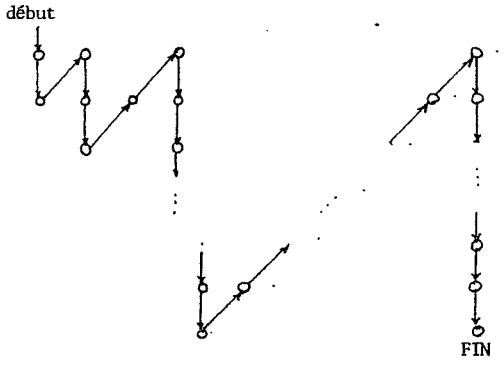
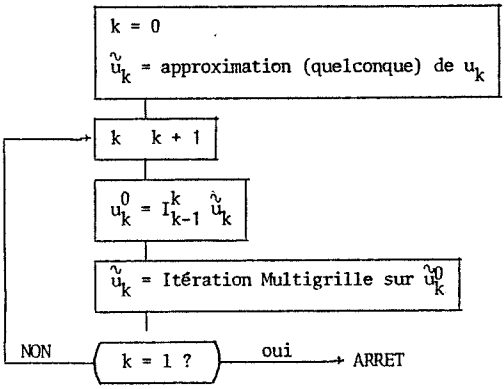



FIGURE 8.1 . Alogrithme FMG

Cette figure montre bien que l'on commence des calculs sur des maillages grossiers pour aller vers des maillages plus en plus fins. Si l'on ajoute à cette méthode un critère permettant de définir dynamiquement les grilles fines, on obtient un processus de raffinement du maillage. Nous étudierons ce problème plus loin.

De façon plus formelle, on représente l'algorithme FMG de la manière suivante :




## B . METHODE MULTIGRILLE POUR DES PROBLEMES NON LINEAIRES

### I . APPLICATION DIRECTE

Considérons maintenant l'équation :

$$N_h u_h = f_h \quad (D_h) \tag{8.1}$$

où $N_h$ : $G(D_h) \to G(D_h)$ est un opérateur non linéaire. Pour résoudre le système (8.1), on utilise une méthode itérative de linéarisation :

$$N_h u_h^j + L_h^j v_h^j = f_h \quad ; \quad u_h^{j+1} = u_h^j + v_h^j \quad (j = 0,1,2,\ldots) \tag{8.2}$$

où $L_h^j$ est une approximation de $N_h'(u_h^j)$. En particulier, pour la méthode de Newton, on choisit $L_h^j = N_h'(u_h^j)$. On peut alors utiliser une méthode multigrille (linéaire) au système :

$$L_h^j v_h^j = d_h^j = f_h - N_h u_h^j \tag{8.3}$$

Une première façon de combiner la méthode de Newton avec une méthode multigrille (linéaire) appliquée à (8.3), est d'adapter le nombre d'itérations à chaque pas de la méthode de Newton. Il faut pour cela définir une technique de contrôle de la convergence de la méthode de Newton.

La seconde façon de procéder consiste à figer le nombre d'itérations multigrilles à chaque pas de Newton. La méthode de Newton prend alors une convergence linéaire. Cette méthode nécessitera un plus grand nombre d'étapes de linéarisation ; par contre aucune technique de contrôle n'est nécessaire.

## II . L'ALGORITHME "FULL APPROXIMATION SCHEME" (FAS)

Comme pour le cas linéaire, la méthode multigrille non linéaire appelée FAS peut être définie récursivement sur la base d'une méthode FAS à deux grilles. Nous décrirons donc une itération de la méthode à deux grilles (h, H) qui permet de calculer $u_h^{j+1}$ à partir de $u_h^j$.

Ceci est représenté sur la Figure (9.1)

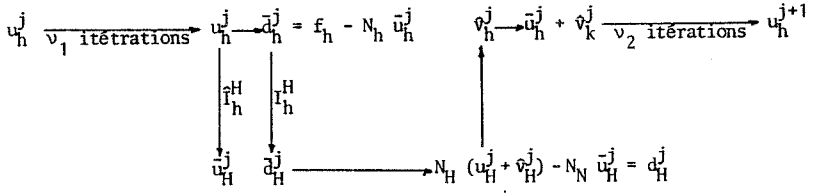



Figure 2.1 : Méthode FAS à deux grilles

Comme dans le cas linéaire, on effectue $\nu_1$ et $\nu_2$ itérations de lissage. Par contre on doit transférer à la grille non seulement le résidu $\bar{d}_h^j$ (grâce à l'opérateur $I_h^H$) mais aussi $\bar{u}_h^j$ lui-même (par un opérateur $\hat{I}_h^H$ qui peut être différent de $I_h^H$). Ceci est nécessaire, car le résidu sur la grille $D_h$ est donné par :

$$N_h (\bar{u}_h^j + v_h^j) - N_h \bar{u}_h^j = \bar{d}_h^j$$

dont l'approximation sur $D_H$ est :

$$N_H (\bar{u}_H^j + \hat{v}_H^j) - N_H \bar{u}_H^j = \bar{d}_H^j$$

ou encore $N_H W_H^j = d_H^j + L_H \bar{u}_H^j$ ; $\hat{v}_H^j = W_H^j + \bar{u}_H^j$

Avec la méthode FAS, sur la grille grossière on ne fait pas de résolution pour trouver directement $\hat{v}_H^j$, mais plutôt pour trouver l'approximation complète $W_H^j$. Bien sûr, pour le transfert à la grille fine $D_h$, il faut définir $\hat{v}_H^j$ car seule la correction (et le résidu) sont lissées par les méthodes de relaxation.

Avec les mêmes notations que pour le cas linéaire, l'algorithme multigrille FAS est alors (pour $l > 1$)

a) Lissage première partie

- Calculer $\bar{u}_l^j$ par $\nu_2$ itérations sur $u_l^j$ :

$$\bar{u}_l^j = R^{\nu_2}(u_l^j, N_l, f_l)$$

b) Correction par grille grossière

- Calculer le résidu $\quad \bar{d}_l^j = f_l - N_l \bar{u}_l^j$
- Restriction du résidu $\quad \bar{d}_{l-1}^j = I_l^{l-1} \bar{d}_l^j$
- Restriction de $\bar{u}_l^j \quad \bar{u}_{l-1}^j = \hat{I}_l^{l-1} \bar{u}_l^j$
- Calculer une solution approchée $\tilde{W}_{l-1}^j$ de :

$$N_{l-1} W_{l-1}^j = \bar{d}_{l-1}^j + N_{l-1} \bar{u}_{l-1}^j \qquad (9.2)$$

en appliquant $\gamma$ itérations de la méthode FAS (en utilisant les grilles $D_0$, $D_1$,... $D_{l-1}$), puis calculer la correction :

$$\tilde{v}_{l-1}^j = \tilde{W}_{l-1}^j - \tilde{u}_{l-1}^j$$

- Interpoler la correction : $\tilde{v}_l^j = I_{l-1}^l \tilde{v}_{l-1}^j$
- Calculer l'approximation corrigée sur $D_l$ : $\bar{u}_l^j + \tilde{v}_l^j$

c) Lissage deuxième partie

- Calculer $u_l^{j+1}$ par $\nu_2$ itérations sur $\bar{u}_l^j + \tilde{v}_l^j$

$$u_l^{j+1} = R^{\nu_2}(\bar{u}_l^j + \tilde{v}_l^j, N_l f_l)$$

## III . DUALITE DE LA METHODE FAS

L'équation (9.2) du résidu sur la grille grossière peut s'écrire :

$$N_H w_H^j = I_h^H f_h + \tau_h^H [\bar{u}_h^j] \tag{10.1}$$

où $$\tau_h^H [Z_h] = N_H \hat{I}_h^H Z_h - I_h^H N_h Z_h \tag{10.2}$$

on a alors : $$N_H (I_h^H u_h) = I_h^H f_h + \tau_h^H [u_h] \tag{10.3}$$

$\tau_h^H [u_h]$ est appelé l'erreur de troncation relative à (h, H). Par rapport aux grilles $D_h$ et $D_H$, $\tau_h^H$ joue le même rôle que l'erreur de troncation (ou erreur de discrétisation) :

$$\tau_h [u] = N_h I^h u - I^h N_u$$

de la solution continue u ($I^h$ représente l'injection du domaine continu D dans le domaine discrétisé $D_h$). On peut interpréter $\tau_h^H [u_h]$ comme la quantité à ajouter au second membre $I_h^H f_h$ pour obtenir la solution de la grille fine $u_h$ en résolvant l'équation sur la grille grossière. Ce paramètre est très important et peut donner lieu à de nombreux développements :

- maillage adaptatif : nous avons vu que $\tau_h^H$ était une approximation de l'erreur de troncation locale, plus précisément on a : $\tau_h^H \approx \tau^H - \tau^h$. On peut donc utiliser $\tau_h^H$ pour estimer $\tau^h$ et ainsi obtenir un critère naturel de raffinement du maillage.

- technique dite "frozen-τ", spécialement appliquée aux problèmes paraboliques pour lesquels on utilise à chaque pas les résultats du pas précédent pour toutes les grilles.

- τ-extrapolation : permet pour certains problèmes, si l'on connait $\tau_h^H$ d'augmenter la précision sur la grille $D_h$.

- calcul locaux : nous développerons ce point par la suite.

Cet ensemble de techniques exprime la dualité de la méthode FAS. En effet suivant la technique utilisée, la grille de base peut être la grille grossière $D_H$ ou la grille fine $D_h$ et l'on peut développer de nombreuses méthodes qui ont pour but soit de diminuer la quantité de calculs en partant d'une grille fine, soit d'augmenter la précision en partant d'une grille grossière.

## C . ANALYSE DES METHODES MULTIGRILLES

Nous avons décrit les méthodes multigrilles avec une approche assez générale. Pour passer aux applications, un point très important doit être résolu : comment choisir les différents composants d'une méthode. Nous donnerons tout d'abord quelques uns des composants les plus fréquemment utilisés puis nous présenterons des méthodes d'analyse des propriétés de ces composants.

### I . COMPOSANTS D'UNE METHODE MULTIGRILLE

#### 1. La méthode de relaxation

C'est la méthode de lissage, représentée par $S_h$ ou $N_h$. On choisit traditionnellement parmi les méthodes suivantes :

- Jacobi-point
- Gauss-Seidel
- Jacobi par blocs
- relaxation par ligne
- gradients conjugués
- décomposition LU incomplète.

Pour les système non-linéaires, on retient le plus souvent la méthode de Newton (ou l'une de ses variantes).

On a démontré, pour l'ensemble de ces méthodes, de bonnes propriétés de lissage.

Les nombres $\nu_1$ et $\nu_2$ d'itération de ces méthodes ne peuvent être déterminé à l'avance. On sait néanmoins qu'ils ne doivent pas être trop grand, (on choisit généralement $\nu_1 + \nu_2 \leq 4$).

#### 2. Taille H de la grille grossière

Il s'agit de savoir connaissant h, comment définir H.

- réduction standard : $H = 2h$
- semi-réduction : en deux dimensions : réduction standart dans une direction et pas de réduction dans l'autre direction
- réduction "Rouge/Noir" (Red-Black) : seulement pour les maillages carrés, on a alors $H = \sqrt{2}\, h$

- quadruplement de h : $H = 4\,h$
- réduction quelconque : $H = n\,h$ $(n > 1)$

3. Opérateurs de transfert entre grilles $I_h^H$ et $I_H^h$

a) Restriction $I_h^H$

- injection : c'est une simple restriction que l'on définit par

$$d_H(P) = d_h(P) \quad \text{pour tout point } P \in D_H \cap D_h$$

- demi-pondération du résidu (Half Residual Weighting)

$$d_H(P) = \frac{1}{8}\begin{bmatrix} & 1 & \\ 1 & 4 & 1 \\ & 1 & \end{bmatrix}_h d_h(P)$$

- pondération complète du résidu (Full Residual Weighting)

$$d_H(P) = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_h d_h(P)$$

b) Interpolation $I_H^h$

- interpolation bilinéaire : on choisit $I_H^h = I_h^{H*}$ (opérateur adjoint de $I_h^H$)
- interpolation quadratique ou cubique
- interpolation utilisant l'équation générale du système.

4. Opérateur sur la grille grossière $L_H$

On distingue deux possibilités :

- $L_H$ correspond à la discrétisation sur la grille grossière $D_H$ de l'équation générale $L\,u = f$ (D)
- Méthode de Galerkin : $L_H = I_h^H\, L_h\, I_H^h$

## II . ANALYSE DU COMPORTEMENT D'UNE METHODE

On ne connait pas de méthode générale pour optimiser les choix précédents. Néanmoins, deux méthodes ont été définies pour connaître l'efficacité d'un ensemble de choix de composants lorsqu'il est appliqué à un problème simple. Ces méthodes permettent de comparer par exemple plusieurs relaxations possibles. Les conclusions qu'on en retire ne sont qu'indicatives, et doivent être validées par la suite, pour les applications particulières où elles seront réellement employées.

### 1. Analyse du problème modèle (Model Problem Analysis)

Le problème modèle correspond à la résolution de l'équation de Poisson

$$\Delta u = f = \text{sur } D, \quad u = g \quad \text{sur } \Gamma$$

sur un domaine carré unité : $D = (0,1)^2$, avec une discrétrisation régulière par différences finies.

Après élimination des conditions aux limites, on obtient :

$$L_h u_h \qquad \text{avec} \quad L_h = \frac{1}{h^2} \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}$$

Pour la méthode multigrille, on choisit :

- réduction standard pour les grilles grossières : h, 2h, 4h,...
- opérateur sur les grilles grossières : $L_H = \frac{1}{H^2} \begin{bmatrix} & -1 & \\ -1 & 4 & -1 \\ & -1 & \end{bmatrix}_H$
- lissage par relaxation rouge noir : $L_h$
- restriction : pondération complète $I_h^{2h} = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_h$
- interpolation bilinéaire $I_{2h}^{h} = \frac{1}{4} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}_{2h}$

Cette méthode particulière permet de tirer des conclusions pour des cas plus généraux en permettant, grâce à sa simplicité, de connaître précisément son comportement [voir Stüben, Hackbush, Trottenberg].

Par exemple, on peut montrer que le facteur de convergence est :

$$\rho(\nu) = \begin{cases} 1/4 & \nu = 1 \\ \frac{1}{2\nu}\left(\frac{\nu}{\nu+1}\right)^{\nu+1} & \nu \geq 2 \end{cases}$$

avec $\nu = \nu_1 + \nu_2$

En particulier $\rho(2) = 0{,}074$, $\rho(3) = 0{,}052$.

Ceci montre qu'il faut choisir des petites valeurs de $\nu$. On considère souvent que la valeur optimale est $\nu = 2$.

On définit également le facteur de lissage $\mu$ de la même façon que $\rho$ mais en ne tenant compte que de la réduction de l'erreur sur les composantes de hautes fréquences. On obtient :

$$\mu(\nu) = \begin{cases} 1/4 & (\nu \leq 2) \\ \left(\frac{2\nu-1}{2\nu}\right)^2 \sqrt[\nu]{\frac{1}{2(2\nu-1)}} & (\nu \geq 3) \end{cases}$$

Cette formule montre que pour de petites valeurs de $\nu$, les propriétés de lissage sont également très bonnes.

L'analyse du problème modèle peut s'étendre à des problèmes tels que l'anisotropie, des problèmes de perturbations singulières,... ce qui la rend utile pour obtenir des premières indications. Néanmoins elle reste limitée à cause de la simplicité du domaine d'étude.

## 2. Analyse de Fourier locale

L'analyse du problème modèle apporte des résultats sur les propriétés de convergence de certaines méthodes multigrilles, mais ne peut être appliquée qu'à un petit nombre de problèmes. Une extension a été construite : l'analyse de Fourier locale.

Cette analyse est basée sur l'étude des fonctions propres associées à un domaine, pour trouver les valeurs de $\rho$ et $\mu$. Nous ne parlerons pas davantage de cette méthode, car comme la précédente elle est essentiellement utile pour les problèmes de différences finies alors que nous nous intéressons aux problèmes d'éléments finis.

# D . APPLICATION DES METHODES MULTIGRILLES AUX ELEMENTS FINIS

## I . POSITIONNEMENT DU PROBLEME

### 1. Objectifs de cette réalisation

Nous avons utilisé un logiciel (DIDACT-FLUX) permettant de résoudre des problèmes magnétostatiques à deux dimensions par la méthode des éléments finis.

Nous avons choisi d'utiliser des méthodes multigrilles pour des problèmes non linéaires. En effet, pour les problèmes linéaires, nous avons décrit quelques méthodes efficaces utilisant le parallèlisme. De plus, le comportement des multigrilles appliquées aux problèmes non linéaire nous a permis d'obtenir des résultats intéressants qui nous amènerons à étudier des extensions au maillage adaptatif et au calcul local (se reporter au Chapitre III).

### 2. Caractéristique du problème éléments finis

Le logiciel permet de définir interactivement une géométrie quelconque, avec une ou plusieurs régions. Il autorise la définition de matériaux linéaires ($\mu$ constant) ou saturable (avec une courbe B(H)). Les sources sont représentées par des régions avec des courants constants. La variable traitée est le potentiel vecteur A qui vérifie l'équation $\vec{rot}(\nu\ \vec{rot}\ \vec{A}) = \vec{J}$. Les conditions aux limites peuvent être de type Dirichlet (A = constante), Neuman ($\frac{\partial A}{\partial n} = 0$), de translation ou cycliques ($A(x_o, y_o) = A\ (x_o + \Delta x_o, y_o + \Delta y_o)$).

Le domaine est maillé automatiquement par des triangles d'ordre 1 ou 2.

On utilise la méthode de Newton pour résoudre le système non linéaire. A chaque pas, on résout le système linéaire définit par ICCG.

## II . PREMIERE APPLICATION : UNE METHODE MULTIGRILLE "ALGEBRIQUE"

### 1. Définition

La géométrie des problèmes traités étant quelconque et le maillage très irrégulier, il apparaît naturel de choisir une méthode multigrille algébrique. Il s'agit d'une méthode pour laquelle on ne tient pas compte de la géométrie sous jacente au problème, mais seulement des équations du système à résoudre.

La notion de grille correspond au graphe de la matrice du système. Ces méthodes constituent des outils de résolution très généraux.

Nous avons choisi une méthode à deux grilles :

$$u_h^j \xrightarrow{\nu_1 \text{ itérations}} \bar{u}_h^j \quad d_h^j = f_h - N_h u_h^j \qquad \hat{v}_h^j \quad \bar{u}_h^j + \hat{v}_h^j \xrightarrow{\nu_2 \text{ itérations}} u_h^{j+1}$$

$$I_h^H \qquad\qquad I_H^h$$

$$\bar{u}_H^j \quad d_h^j \xrightarrow{\nu_3 \text{ itérations}} N_H (\bar{u}_H^j + \hat{v}_H^j) \quad N_H \bar{u}_h^j = d_H^j$$

La structure de donnée du logiciel (DIDACT-FLUX) sur lequel nous avons implanté cette méthode nous a obligé à choisir $I_h^H$ et $I_H^h$ de façon simple de manière à limiter les recherches de valeurs de pondérations de variables voisines. C'est pourquoi, nous avons fait les choix suivants :

- On passe du maillage fin au maillage grossier par suppression d'un noeud sur deux (si la numérotation des noeuds est correcte, ceci assure un recouvrement acceptable du domaine d'étude pour le maillage grossier).

- La restriction $I_h^H$ est une injection. Comme l'on garde un noeud sur deux, l'injection correspond à conserver les valeurs nodales des noeuds conservés par le changement de grille.

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_{2k-1} \\ u_{2k} \\ u_{2k+1} \\ \vdots \\ u_{n-1} \\ u_n \end{bmatrix} \xrightarrow{I_h^H} u_H \qquad \begin{bmatrix} u_1 \\ u_3 \\ \vdots \\ u_{2k-1} \\ u_{2k+1} \\ \vdots \\ u_n \end{bmatrix}$$

- L'interpolation $I_H^h$ est également une injection :

$$v_H \begin{bmatrix} v_1 \\ v_3 \\ \vdots \\ v_{2k-1} \\ v_{2k+1} \\ \vdots \\ v_n \end{bmatrix} \xrightarrow{I_H^h} v_h \begin{bmatrix} v_1 \\ 0 \\ v_3 \\ 0 \\ \vdots \\ v_{2k-1} \\ 0 \\ v_{2k+1} \\ 0 \\ \vdots \\ v_n \end{bmatrix}$$

L'opérateur $N_H$ pour la grille grossière est celui issu de la méthode de Galerkine qui est naturelle à la méthode des éléments finis :

$$N_H = I_h^H \; N_h \; I_H^h$$

$$N_h = \begin{bmatrix} n_{i-1,j-1} & n_{i-1,j} & n_{i-1,j+1} \\ n_{i,j-1} & n_{i,j} & n_{i,j+1} \\ n_{i+1,j-1} & n_{i+1,j} & n_{i+1,j+1} \end{bmatrix} \qquad N_H = \begin{bmatrix} n_{i-2,j-2} & 0 & n_{i-2,j} & 0 & n_{i-1,j+2} \\ n_{i,j-2} & 0 & n_{ij} & 0 & n_{i,j+2} \\ n_{i+2,j-2} & 0 & n_{i+2,j} & 0 & n_{i+1,j+2} \end{bmatrix}$$

Les paramètres $\nu_1$, $\nu_2$, $\nu_3$ ont été déterminés par des essais pour assurer la convergence la plus rapide de la méthode. On a trouvé :

$$\nu_1 = 1 \quad ; \quad \nu_2 = 0 \quad ; \quad \nu_3 = 1$$

## 2. Résultats

Nous donnerons les résultats de cette méthode pour un exemple test. Cet exemple constitue un problème de magnétostatique typique des problèmes traités par le logiciel que nous utilisons. La géométrie et les propriétés physiques sont définies à la Figure 1. Le maillage généré est montré à la Figure 2.

La Figure 3 montre les vitesses de convergence d'une part et de la méthode de Newton et d'autre part la méthode multigrille.
Notons que tous les calculs sont effectués en double précision.

Sur cet exemple, on notera que la méthode multigrille converge plus rapidement que la méthode de Newton, et de plus pour le multigrille, il y a la moitié des itérations effectuées sur des systèmes de dimension moitié (grille grossière).

On peut montrer que le nombre d'itérations multigrille ne dépasse pas le nombre d'itérations de Newton. Ceci assure donc un gain de 25 % avec cette nouvelle méthode (on gagne au moins la moitié des calculs sur la moitié des itérations). Sur l'exemple test du fait de l'accélération de convergence, le gain est de 37 %.

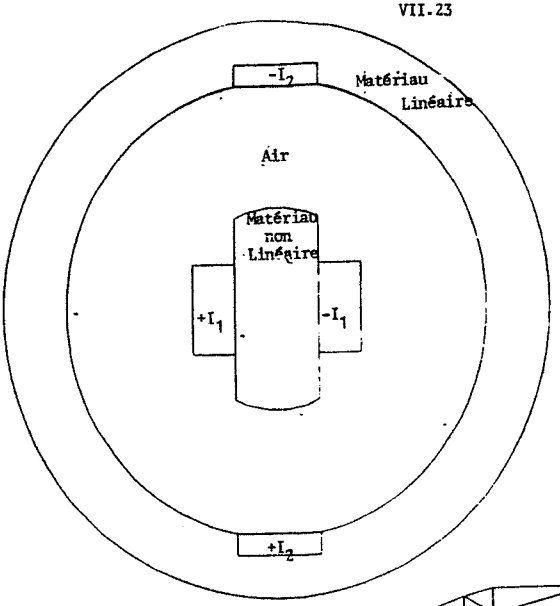



Figure 1

Problème test.

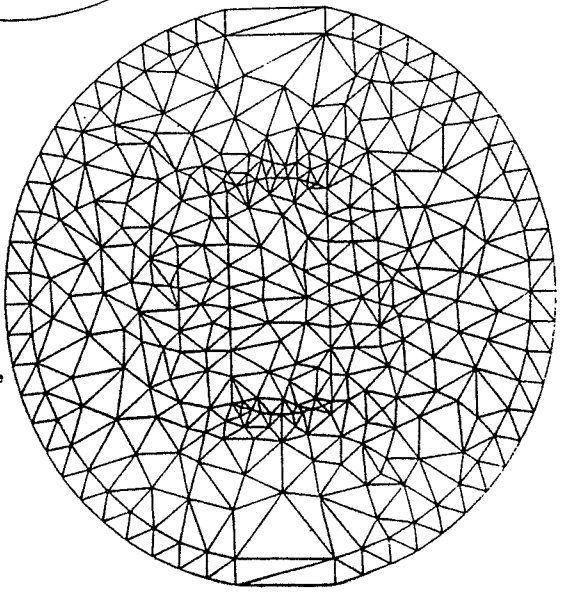


Figure 2

Découpage du Problème

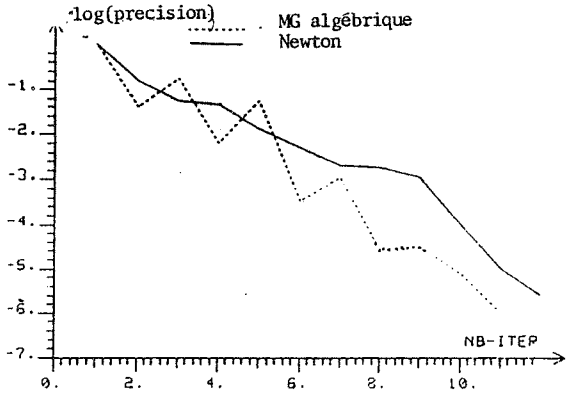



Figure 3

Convergence de la méthode pour le problème test.

## II . ANALYSE DU RESIDU ET EXTENSION DE LA METHODE

On peut utiliser la méthode précédente de façon récursive et travailler sur plus de deux niveaux. Plutôt que d'avoir à passer par l'intermédiaire de plusieurs niveaux, nous avons pensé passer directement du niveau fin au niveau plus grossier en modifiant l'interpolation. Le maillage grossier peut n'avoir alors qu'un petit nombre de noeuds (un dixième du maillage fin).

Pour que la méthode reste applicable, il faut que le maillage grossier reste assez représentatif du problème à traiter. C'est pourquoi nous avons choisi de garder comme noeuds du maillage grossier les noeuds où l'erreur sur la valeur nodale est la plus importante.
On peut démontrer dans des cas simples, et nous admettrons pour des cas plus complexes, que la valeur de l'erreur d'une itération donnée est proportionnelle à la valeur du résidu à cette itération. Rappelons qu'à ce niveau de l'exposé, l'erreur est définie par $u_h - u_h^j$ (erreur par rapport à la solution discrète) et non par $u - u_h^j$ (erreur par rapport à la solution continue exacte).

L'algorithme que nous avons implanté teste donc le résidu en chaque noeud, et retient pour définir le maillage grossier les noeuds où le résidu est le plus grand. A l'aide de cette définition du maillage grossier, on utilise la méthode multigrille précédente.

Les zones locales sont visualisées aux Figures 4, 5, 6.
La Figure 7 montre la convergence de cette méthode pour l'exemple test.
Une itération sur deux a un coût en calcul quasiment nul (au moins dix fois inférieur aux itérations normales), ces itérations sur le maillage grossier ne sont donc pas prises en compte sur ce schéma. Tout revient donc avec cette méthode à diviser le nombre d'itérations par deux, donc d'avoir un gain de 50 % par rapport à la méthode de Newton.

Nous avons décrit sur les Figures 8 et 9 un second exemple. Pour celui-ci, la méthode de Newton classique nécessite 12 itérations, alors qu'avec la méthode multigrille, on obtient la convergence en 7 itérations, soit un gain de l'ordre de 40 %.

Nous avons effectué de nombreux essais pour des problèmes très divers, et dans la plupart des cas nous avons trouvé des gains de 40 % à 50 % en utilisant cette méthode multigrille.

On peut en tirer deux conclusions.
D'une part, la méthode est efficace, et l'on devrait pouvoir améliorer les gains en utilisant une méthode à plus de deux grilles.
D'autre part, la méthode est très générale car elle n'utilise pas le support éléments finis d'où est issu le système. Elle pourra donc être employée pour des résolutions de systèmes non linéaires quelconques.

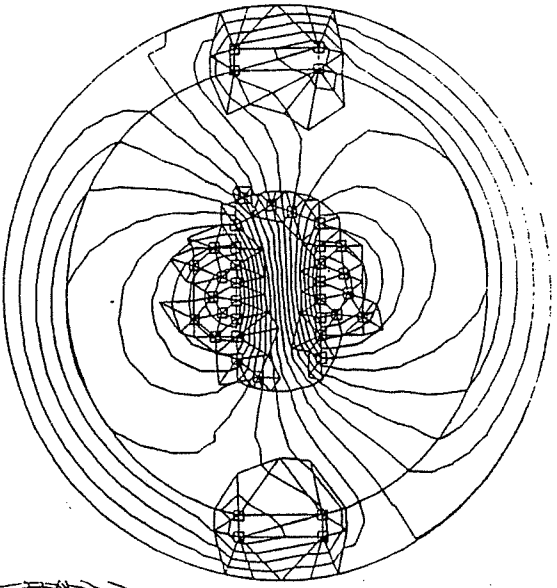


Figure 4

2ème méthode.
Localisation du
Résidu Maximum
(1ère itération)

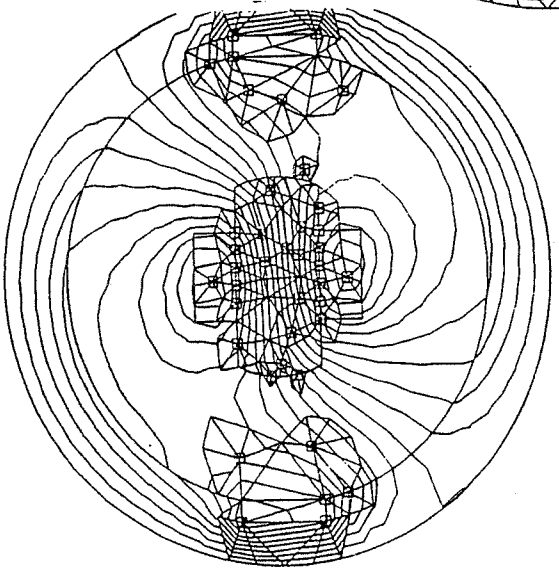
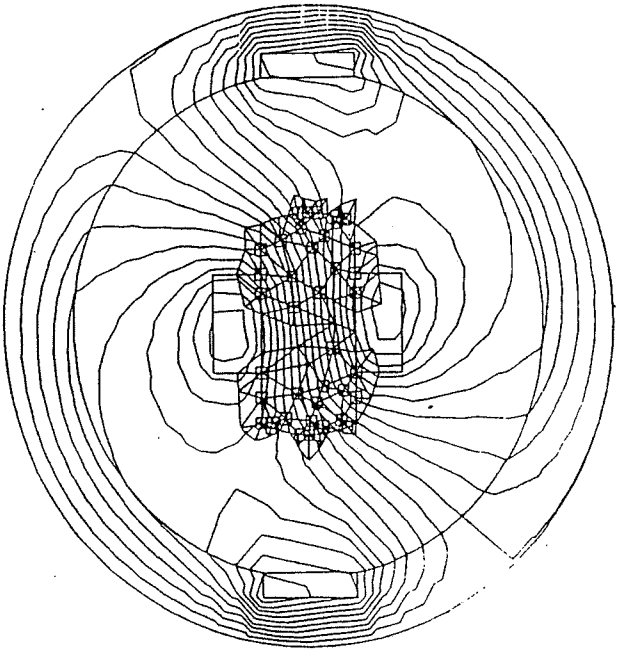


Figure 5

(4ème itération)

Figure 6

(6ème itération)



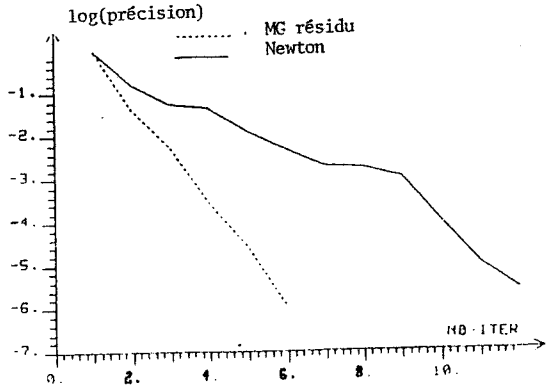



Figure 7

Convergence de la 2ème méthode.

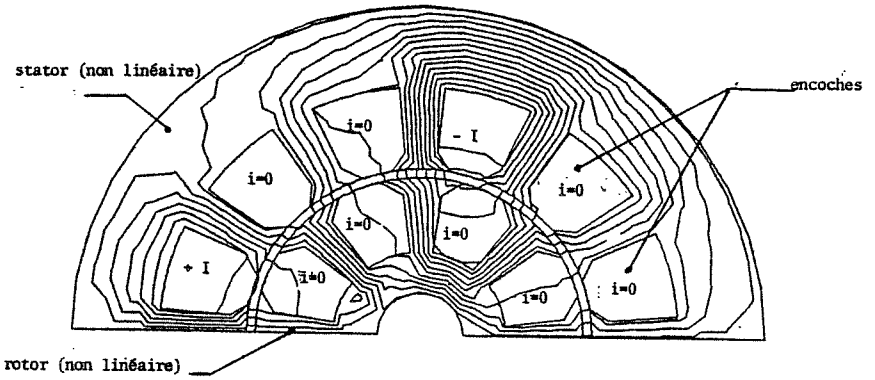



Figure 8. Exemple d'application.

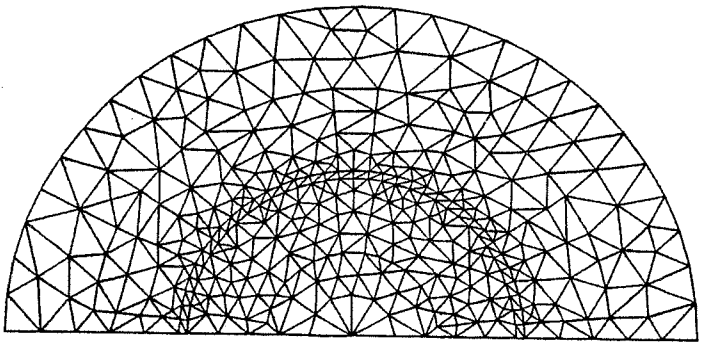


Figure 9. Maillage de l'exemple.

# CHAPITRE

# 8

## CHAPITRE VIII . EXTENSION DE LA METHODE

INTRODUCTION

I . EXISTENCE DE MAILLAGES OPTIMAUX POUR LES ELEMENTS FINIS

II . QUELQUES EXEMPLES D'APPLICATION

1. Raffinement du maillage
2. Calcul local

III . APPROCHE DU MAILLAGE ADAPTATIF A PARTIR DES MULTIGRILLES

IV . REALISATION D'UNE METHODE COMBINANT CES TECHNIQUES

# CHAPITRE VIII . EXTENSION DE LA METHODE

## INTRODUCTION

L'étude des méthodes multigrilles amène à se poser le problème de la qualité des maillages utilisés. Comme nous l'avons remarqué à plusieurs reprises, plutôt que de partir du maillage le plus fin, les méthodes multigrilles doivent permettre de progresser des maillages les plus grossiers vers des maillages plus fins. Il est alors intéressant de savoir comment définir un maillage optimal. Il faut donc d'abord montrer l'existence de maillages optimaux. Les techniques de maillage adaptatif permettent alors de définir des raffinements de maillage tendant vers la valeur optimale. Nous donnerons quelques exemples de ces techniques. Nous montrerons les rapports entre les maillages adaptatifs et certaines méthodes multigrilles.

Ces études nous amènerons également à considérer les possibilités de calculs locaux et nous montrerons pour finir une application que nous avons réalisée.

## I . EXISTENCE DE MAILLAGES OPTIMAUX POUR LES ELEMENTS FINIS

La méthode des éléments finis fournit pour chaque problème une solution u approchée de la solution exacte $u_{ex}$. L'erreur d'approximation en tout point M d'un élément $V^e$ est définie par :

$$e(M) = u(M) - u_{ex}(M) \tag{1.1}$$

Sur l'élément $V^e$, on peut caractériser l'erreur par :

$$|e| = \text{Maximum sur } V^e \text{ de } |e(M)|$$

L'erreur sur chacune des dérivées d'ordre S est donnée par :

$$e_s(M) = D^s(e(M)) = \frac{\partial^s e(M)}{\partial x^a \partial y^b \partial z^c} \quad ; \quad a+b+c = s \tag{1.2}$$

La norme correspondante est : $|e|_s = \text{Maximum sur } V^e \text{ de } |D^s(e(M))|$

On utilise également la semi-norme des moindres carrés :

$$\|e\|_s = \left( \sum_{a+b+c=s} \int_{V^e} (D^s(e(M)))^2 \, d V^e \right)^{1/2} \tag{1.3}$$

Strang donne les expressions suivantes :

$$|e|_s \leq c\, h^{n-s}\, |u_{ex}(M)|_n \qquad (1.4)$$

$$\|e\|_s^2 \leq C\, h^{2(n-s)}\, |u_{ex}(M)|_n^2 \qquad (1.5)$$

où
- c et C dépendent du type d'élément et de l'approximation utilisés,
- la base des fonctions d'approximation est complète jusqu'à l'ordre n-1,
- h est lié à la taille de l'élément (par exemple le rayon du cercle circonscrit à un élément triangulaire),
- $|u_{ex}(M)|_n$ est la norme de $u_{ex}(M)$ avec s = n.

Pour améliorer la précision de l'approximation, on peut :

- soit diminuer h, donc la dimension de chaque élément,
- soit augmenter n, c'est-à-dire des fonctions d'approximation d'ordre plus élevé.

De nombreuses études ont été menées pour étudier plus finement le comportement de l'erreur en fonction des maillages. Certaines amènent à des relations du même type que celles données ci-dessus (voir Lin Qun).
D'autres permettent d'établir des relations entre la norme de l'erreur et la norme du résidu (voir Babuska, Rheinboldt, Paula de Oliveira, Carey, Humphrey).

Par exemple Babuska donne la relation :

$$\|e\|_{1,p} \leq C \left( \sum_{\text{éléments}} \left( \left| \int_{V^e} r(M)\, d\, V^e \right| \right)^p h^p \right)^{1/p} \qquad (1.6)$$

où r est le résidu $r(M) = L\,u(M) - f(M)$

en utilisant les normes des espaces de Sobolev : $\|u\|_{k,p} = \left| \sum_{0 \leq \alpha \leq k} \left\| \frac{d^\alpha u}{dx^\alpha} \right\|^p \right|^{1/p}$

La plupart des formules sont de la forme :

$$\|e\| \leq C\, h^p\, \|R\| \qquad (1.7)$$

ce qui montre qu'une amélioration du maillage se fera par réduction de la taille des éléments, mais aussi que l'on doit augmenter la précision sur les éléments où le résidu est grand.
Ceci est le point de départ des techniques de maillage adaptatif qui améliorent la précision d'un système par raffinement du maillage à l'aide de critères essentiellement basés sur l'analyse de la répartition du résidu.

## II . QUELQUES EXEMPLES D'APPLICATION

### 1. Raffinement du maillage

Les exemples d'étude des maillages adaptatifs avec des applications concrètes sont encore peu nombreux. Nous citerons néanmoins les travaux de Randolph E. Bank qui a proposé une méthode pour la résolution de l'équation :

$$- \nabla . (a \nabla u) + b . \nabla u + c u = f \text{ pour } D \quad R^2$$

et $$\frac{\partial u}{\partial n} = 0 \text{ sur la frontière de D.}$$

La formulation consiste à trouver la solution u telle que

$$\forall v, \quad a(u, v) = (f, v)$$

avec $$a(u, v) = \int_D (a \nabla u . \nabla v + b . \nabla u v + c u v) \, d V$$

et $$(f, v) = \int_D f v \, d V$$

La technique de raffinement du maillage est la suivante. Si l'on connait la solution $u_1$ pour un maillage $D_1$ dont la taille moyenne des éléments est $h_1$, on peut montrer que l'erreur commise est pour chaque élément $V^e$

$$\|e\|_{V^e} = \|u_1 - u\|_{V^e} = C_1 h_1^2 \int_{V^e} R^2 \, dV + C_2 h_1 \int_{\partial V^e} J^2 \, ds$$

où
- $C_1$ et $C_2$ sont des constantes qui ne dépendent que de a et de la forme géométrique des éléments (mais pas de leur taille),
- R est le résidu,
- J représente les discontinuités de la dérivée normale de $u_1$ sur l'élément $V^e$.

On calcule alors l'erreur maximum $e_{max}$ et l'élément correspondant $V^e_{max}$.
L'élément $V^e_{max}$ est alors raffiné en sous-élément, et l'on calcule pour l'ensemble de ces sous-éléments l'erreur maximum locale : $\bar{e}_{max}$.

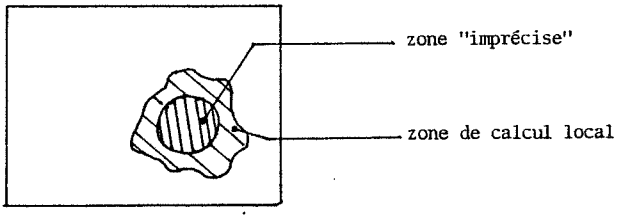
Le maillage raffiné $D_{1+1}$ est obtenu en maillant plus finement tous les éléments de $D_1$ dont l'erreur est comprise entre $\bar{e}_{max}$ et $e_{max}$.
L'auteur montre sur quelques exemples, que la méthode permet d'obtenir de très bonnes précisions avec une rapidité de convergence importante.

En dehors d'études basées principalement sur la répartition du résidu, on peut noter d'autres techniques développées pour des applications spécialisées (par exemple en électromagnétisme) et basées sur des considérations énergétiques. Nous noterons en particulier, les travaux récents de Z.J. Czendés. Celui-ci définit l'erreur dû à son maillage en calculant pour chaque élément deux bornes de l'énergie, ces deux valeurs étant basées sur des formulations duales du problème.
De façon générale, le problème prédominant est la définition de l'erreur associée à un maillage. Le maillage adaptatif sert à augmenter la précision d'une solution, mais on peut également l'utiliser au cours du processus de résolution pour accélérer la convergence (ou minimiser les calculs).

## 2. Calcul local

On peut faire apparaître une seconde notion lorsque l'on étudie un processus en évolution, par exemple une évolution temporelle mais le raisonnement peut être étendu à toute résolution de type itératif. Il peut être intéressant de définir à chaque étape un raffinement du maillage, celui-ci pouvant varier d'une étape à l'autre car l'erreur n'est pas toujours localisée dans les mêmes régions.

On peut noter à ce propos, les travaux de l'équipe de G.H. Rodrigue qui a étudié des méthodes adaptatives pour des équations aux dérivées partielles dépendant du temps. Dans ces études, on montre qu'à chaque pas de résolution, les régions de mauvaise précision sont localisées. Pour éviter de recommencer un calcul sur l'ensemble du domaine, on définit une zone locale de calcul en prenant la zone de mauvaise précision entourée d'une zone où la précision est meilleure. Sur la frontière de cet ensemble local, on fixe des conditions de type Dirichlet en gardant les valeurs actuelles de la solution en chaque noeud. On résoud le problème local puis on réinjecte la solution dans le problème global.




A chaque itération la zone locale peut être différente de la précédente.
Cette méthode peut être utilisée pour tout processus itératif.

## III . APPROCHE DU MAILLAGE ADAPTATIF A PARTIR DES MULTIGRILLES

Comme nous l'avons déjà remarqué, la précision obtenue à partir d'un maillage dépend de la taille h des éléments, de p l'ordre de fonctions d'approximations et est également liée à la répartition du résidu.

Le problème est de minimiser une certaine erreur e, en utilisant pour cela une quantité de calculs représentée par W. L'optimisation ne doit pas être couteuse en calcul pour ne pas aller à l'encontre du but recherché.

On peut, pour les méthodes multigrilles, définir un estimateur E de l'erreur par :

$$E = \int_D G(M)\ \tau_h(M)\ d\,V \tag{3.1}$$

où $\tau_h(M)$ est l'erreur locale de troncation (définie au Chapitre II, (B-III), $G(M) \geq 0$ est une fonction de pondération. G(M) est en principe donnée par l'utilisateur.
En pratique, G(M) peut être utile comme outil de contrôle, seul l'ordre de grandeur est important et il peut être choisi de façon simple. Par exemple, si l'on désire calculer les dérivées d'ordre 1 d'une solution à partir d'une frontière donnée, on peut choisir $G(M) = d_M^{m-1-1}$, où $d_M$ représente la distance du point M à la frontière et m l'ordre de l'équation différentielle (Dans des cas simples, on se restreint souvent à G(M) = 1).

La quantité globale de calculs W est alors :

$$W = \int_D \frac{w(p(M)}{h(M)^d}\ d\,V \tag{3.2}$$

où p(M) représente l'ordre des fonctions d'approximation au point M, d est la dimension de l'espace (et $h^{-d}$ la densité de noeuds du maillage), et w(p) la quantité de calculs par noeuds du domaine.

Si l'on traite h(M) comme une fonction continue, l'équation d'Euler de minimisation de E s'écrit :

$$G\,\frac{\partial \tau}{\partial h} - \lambda\ d\ w(p)\ h^{-d-1} = 0 \tag{3.3}$$

où $\lambda$ est une constante (multiplicateur de Lagrange) représentant le taux marginal de variation de la précision par rapport à la quantité de calculs :

$$\lambda = -\ dE\,/\,dW \tag{3.4}$$

Lorsque $\lambda$ est fixé, l'équation (3.3) définit en tout point $M \in D$ la valeur optimale de h(M).

Pour l'algorithme FAS, nous savons obtenir la quantité $\tau_h^H$ et l'on a $\tau^h \simeq \tau_h^H$.
La quantité $-\Delta E(M) = G(M)\ \tau_h^H(M)$ peut être utilisée comme incrément de E lors du passage du maillage de taille H au maillage de taille h au voisinage du point M. La quantité de calculs nécessaire à ce travail est : $\Delta W = w(p)\ h^{-d}(1 - 2^d)$.

Le taux local de variation de précision est $Q = -\Delta E/\Delta w \simeq -G(M)\ \Delta R(M)/W$
où $\Delta R$ est la variation du résidu correspondant à $\Delta E$.

Si Q est plus grand que $\lambda$, on peut en conclure que la transition de H à h est profitable. On peut continuer à mailler plus finement tant que $Q > \lambda$.
On peut remarquer que l'on peut conduire le même raisonnement pour optimiser l'ordre des fonctions d'approximation p(x) |l'opérateur $\tau^h$ étant remplacé par $\sigma^h = L_h^{(0)}\ u_h - L_h^{(1)}\ u_h$ où $L^{(0)}$ est l'opérateur de degré le plus bas et $L^{(1)}$ l'opérateur de haut degré|.

Le paramètre $\lambda$ est fixé a priori, pour augmenter les raffinements de maillage il faut faire diminuer à chaque étape la valeur de $\lambda$.

Le raffinement peut se poursuivre indéfiniment. On peut l'arrêter lorsque l'on des trois paramètres E, W ou $\lambda$ a atteint une valeur prédéterminée.

## IV . REALISATION D'UNE METHODE COMBINANT CES TECHNIQUES

Pour la résolution de problèmes électromagnétiques non linéaires, nous avons développé une méthode s'inspirant de la technique de calcul local.

Il s'agit d'une méthode multigrille à deux niveaux (bien qu'elle puisse être étendue à plus de deux niveaux) de type FAS (voir Chapitre II). Mais on ne dispose pas à proprement parler de deux grilles. Le maillage original correspond à la grille fine. La grille grossière est simplement une zone locale de la grille fine construite en considérant les éléments où la valeur du résidu est la plus grande, augmentée d'une zone frontalière (comme définie au § II.2). Nous avons réservé la possibilité d'ou bien conserver le maillage original pour cette zone locale ou bien de raffiner ce maillage.

La méthode consiste alors à effectuer $\nu_1$ itérations de Newton sur la grille fine, puis de définir la zone locale. On résoud alors un nouveau problème d'éléments finis non linéaires pour cette zone. Puis les résultats sont interpolés sur la grille fine.

Il faut bien noter que contrairement aux méthodes du Chapitre II où l'opérateur sur la grille grossière est défini par la méthode de Galerkine, ici l'opérateur est directement issu d'un problème d'éléments finis local (défini par les éléments peu précis, et les noeuds de la frontière de cette zone étant pris comme des conditions de Dirichlet).

Des essais ont montré qu'on obtient une bonne méthode, notamment lorsque $\nu_1 = 1$ (un calcul local par itération de Newton) et qu'il suffit de rassembler dans la zone locale un petit nombre d'éléments (en général un dixième de l'ensemble).

Nous avons signalé que nous avions réservé une possibilité de maillage adaptatif (raffinement de la zone locale), mais nous n'avons pas pour l'instant effectué des tests pour connaître l'efficacité de cette méthode.
Par contre, nous avons testé la méthode pour l'exploitation du calcul local (sans raffinement du maillage). Nous présentons dans les figures suivantes (4.1 à 4.10) trois exemples assez représentatifs des résultats obtenus. On obtient une méthode dont les gains sont encore plus importants que ceux du Chapitre II, car le calcul local permet d'accélérer considérablement la convergence du calcul du système non-linéaire. Ainsi, on obtient la convergence en 10 itérations au lieu de 23 pour l'exemple 1, en 7 au lieu de 16 pour l'exemple 2 et en 9 au lieu de 26 pour l'exemple 3.

Ces quelques résultats, ainsi que d'autres non cités ici, illustrent l'efficacité de cette méthode. Par contre, cette méthode est étroitement liée à l'origine du système à résoudre et ne pourrait pas être généralisée à d'autres systèmes comme les méthodes du Chapitre II. On retrouve ici le problème de l'antagonisme entre l'efficacité des méthodes et leur généralité. Néanmoins pour les cas qui nous intéressent particulièrement, les gains apportés sont non négligeables et pourront certainement être très utiles à la résolution de problèmes très complexes posés par les développements constants des méthodes liées à la Conception Assistée par Ordinateur.

Exemple 1.

Description

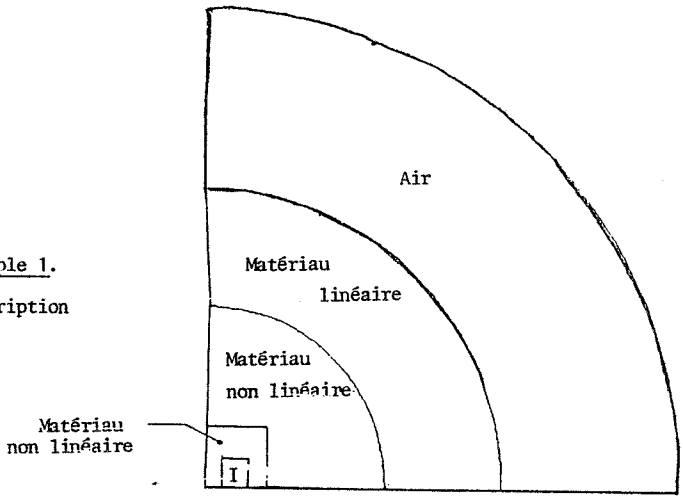



Figure 4.1

Exemple 1.

Maillage

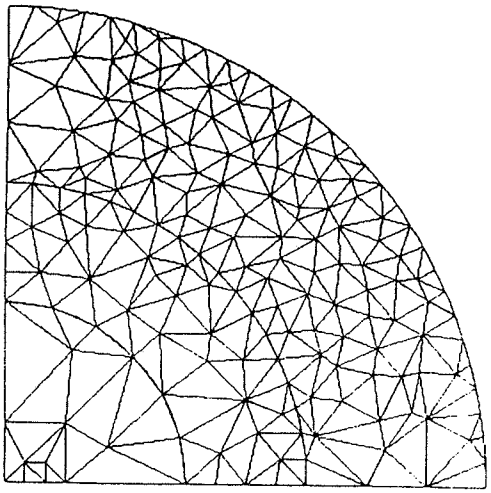


Figure 4.2

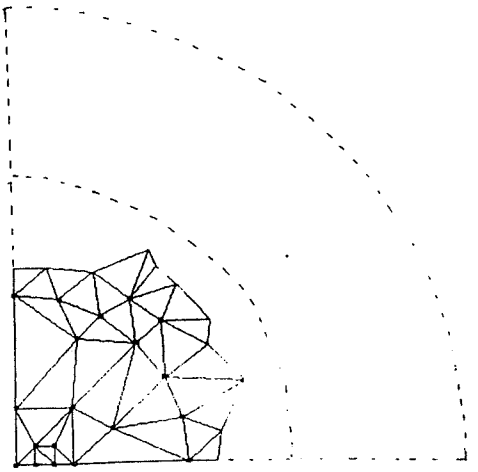


Figure 4.3

Zone locale de calcul pour l'exemple 1.

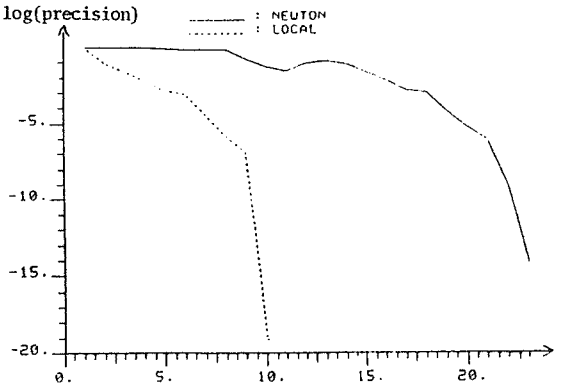



Figure 4.4

Convergence de la méthode de calcul local par rapport à la méthode de Newton pour l'exemple 1.

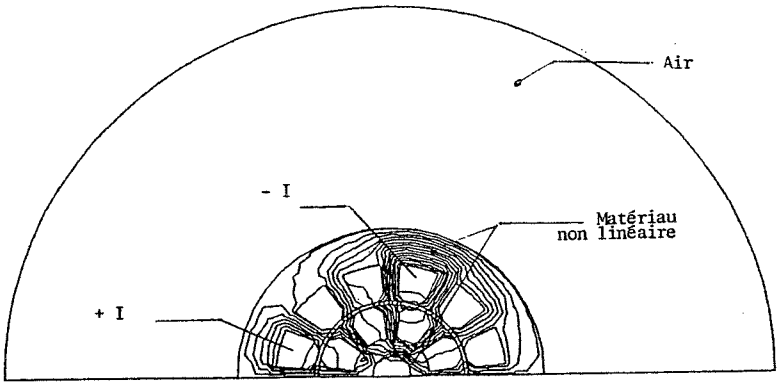



Figure 4.5 . Exemple 2 - Description

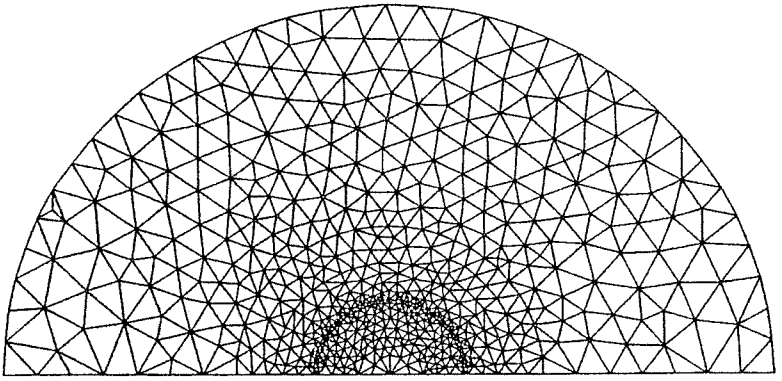


Figure 4.6 . Exemple 2 - Maillage

Figure 4.7 . Tableau comparatif de la convergence de la méthode de Newton et de la méthode par calcul local pour l'exemple 2.

| Itération | Précision - méthode de Newton | Précision - méthode par calcul local |
|---|---|---|
| 1 | 1. | 1. |
| 2 | $0,86\times10^{-1}$ | 0,11 |
| 3 | $0,41\times10^{-1}$ | $0,19\times10^{-1}$ |
| 4 | $0,44\times10^{-1}$ | $0,20\times10^{-1}$ |
| 5 | $0,34\times10^{-1}$ | $0,41\times10^{-2}$ |
| 6 | $0,86\times10^{-2}$ | $0,16\times10^{-2}$ |
| 7 | $0,21\times10^{-1}$ | $0,10\times10^{-4}$ |
| 8 | $0,18\times10^{-1}$ | |
| 9 | $0,15\times10^{-1}$ | |
| 10 | $0,11\times10^{-1}$ | |
| 11 | $0,99\times10^{-2}$ | |
| 12 | $0,82\times10^{-2}$ | |
| 13 | $0,76\times10^{-2}$ | |
| 14 | $0,73\times10^{-3}$ | |
| 15 | $0,75\times10^{-4}$ | |
| 16 | $0,39\times10^{-6}$ | |

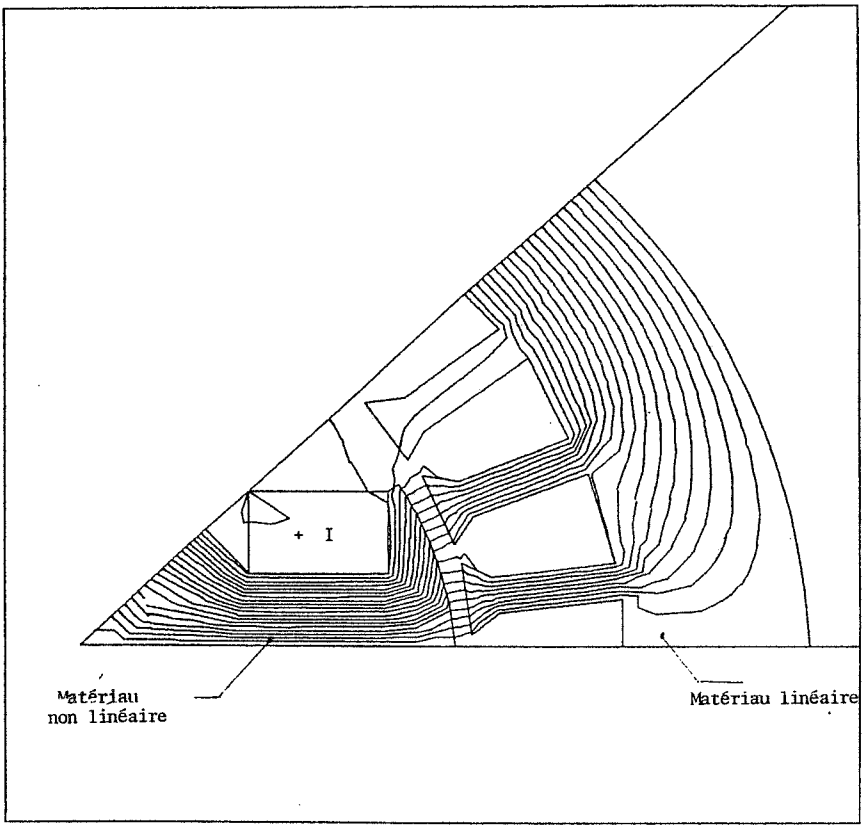



Figure 4.8 . Exemple 3

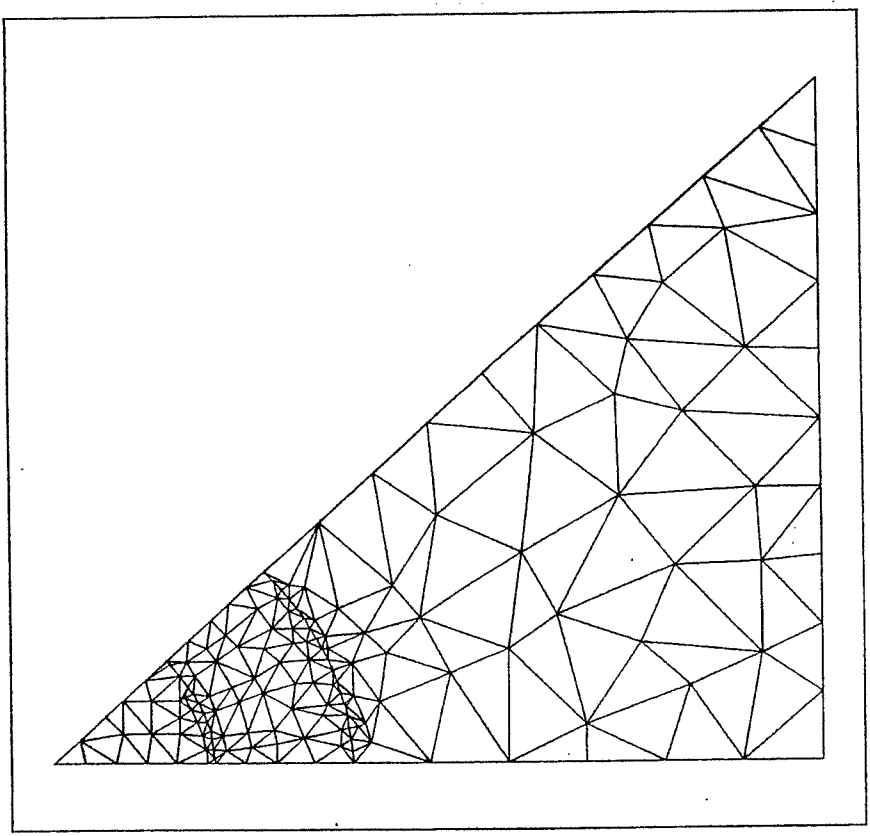


Figure 4.9 . Exemple 3 - Maillage

Figure 4.10 . Convergence pour l'exemple 3.

| Itération | Méthode de Newton | Newton avec calcul local |
|---|---|---|
| 1 | 1. | 1. |
| 2 | 0,48 | 0,14 |
| 3 | 0,39 | $0{,}26\times10^{-1}$ |
| 4 | 0,36 | $0{,}21\times10^{-1}$ |
| 5 | $0{,}4\times10^{-1}$ | $0{,}58\times10^{-2}$ |
| 6 | $0{,}24\times10^{-1}$ | $0{,}62\times10^{-2}$ |
| 7 | $0{,}17\times10^{-1}$ | $0{,}93\times10^{-3}$ |
| 8 | $0{,}32\times10^{-2}$ | $0{,}1\times10^{-4}$ |
| 9 | $0{,}46\times10^{-2}$ | $0{,}19\times10^{-9}$ |
| 10 | $0{,}24\times10^{-2}$ | |
| 11 | $0{,}17\times10^{-2}$ | |
| 12 | $0{,}15\times10^{-2}$ | |
| 13 | $0{,}13\times10^{-2}$ | |
| 14 | $0{,}89\times10^{-3}$ | |
| 15 | $0{,}91\times10^{-3}$ | |
| 16 | $0{,}83\times10^{-3}$ | |
| 17 | $0{,}51\times10^{-3}$ | |
| 18 | $0{,}23\times10^{-2}$ | |
| 19 | $0{,}28\times10^{-2}$ | |
| 20 | $0{,}3\times10^{-3}$ | |
| 21 | $0{,}44\times10^{-3}$ | |
| 22 | $0{,}41\times10^{-3}$ | |
| 23 | $0{,}17\times10^{-3}$ | |
| 24 | $0{,}13\times10^{-3}$ | |
| 25 | $0{,}50\times10^{-4}$ | |
| 26 | $0{,}61\times10^{-5}$ | |

# B I B L I O G R A P H I E

## DEUXIEME PARTIE

### CHAPITRE VI - CHAPITRE VII - CHAPITRE VIII . METHODES DES MULTIGRILLES


[1] BRANDT A.

"Multilevel adaptative technique for fast numerical solution to boundary value problems"
Third International Conference Numerical Methods in Fluid Mechanics, Paris, 1972

[2] BRANDT A.

"Multilevel adaptative finite elements method. Variational problems"
Special Topics of Applied Mathematics. North Holland Publishing Company, Amsterdam, 1979

[3] BRANDT A.

"Guide to Multigrid Development"
Lecture Notes in Mathematics Springer-Verlag - Köln Porz, 1981

[4] BRANDT A.

"Multigrid solvers on parallel computers"
Elliptic Problem Solvers - Academic Press, 1981

[5] HACKBUSCH W.

"Survey of convergence proofs for Muligrid iterations"
Special Topics of Applied Mathematics - North-Holland Publishing Company, Amsterdam, 1980

[6] HACKBUSCH W.

"On the convergence of multigrid iterations"
Beiträge Numer, Math. 1981

[7] HACKBUSCH W.

"Multigrid Convergence Theory"
Lecture Notes in Mathematics. Springer-Verlag - Köln-Porz, 1981

[8] STUBEN K., TROTTENBERG U.

"On the construction of fast solvers for elliptic equations"
Computational Fluid Dynamics Lecture, series 1982-04, Rhode-Saint-Genese 1982

[9] STUBEN K., TROTTENBERG U.

"Multigrid methods, Fundamental Algorithms, Model Problem Analysis and Applications"
Lecture Notes in Mathematics. Springer-Verlag - Köln Porz, 1981

[10] AXELSON O.

"On Multigrid Methods of the Two Level Type"
Lecture Notes in Mathematics. Springer-Verlag - Köln Porz, 1981

[11] WESSELING P.

"Numerical Solution of the stationnary Navier-Stokes equations by means of a multiple grid method and Newton iteration"
Report NA-18. Delft University of Technology 1981

[12] WESSELING P.

"A robust and efficient multigrid method"
Lectures Notes in Mathematics. Springer-Verlag - Köln-Porz, 1981

[13] MEIS Th., LEHMANN H., MICHAEL H.

"Application of the Multigrid Method to a Non linear Indefinite Problem"
Lecture Notes in Mathematics, Springer-Verlag - Köln-Porz, 1981

[14] BANK R.E.

"A Multilevel iterative Method for Nonlinear Elliptic Equations"
Elliptic Problem Solvers - Academic Press, 1981

[15] RODRIGUE G. HEDSTROM G.

"Adaptative-grid methods for time dependent partial differential equations"
Lawrence Livemore National Laboratory, 1982

[16] RODRIGUE G., HEDSTROM G.

"Adaptative Mesh Refinement for 1 - Dimensionnal Gas Dynamics"
10th IMACS World Congress on System Simulation and Scientific Computation, 1982

[17] CAREY G., HUMPHREY D.

"Mesh Refinement and iterative solution methods for finite element computations"
International Journal for Numerical Methods in Engineering, vol.17, pp.1717-1734, 1981

[18] OLIVIERA P.

"Existence de maillages optimaux dans les méthodes d'éléments finis"
RAIRO, Analyse Numérique - vol.14, n° 3, 1980

[19] BABUSKA I., RHEINBOLDT W.

"A posteriori error for the finite element method"
International Journal for Numerical Methods in Engineering, vol.12, pp.1597-1615, 1978

[20] Lin QUN

"Iterative refinement of Finite Element approximations for elliptic problems"
RAIRO, Analyse Numérique, vol.16, N° 1, 1982

[21] CZENDES Z.J., SHENTON D., SHAHNASSER H.

"Adaptative finite element mesh generation using the Delaunay algorithm"
Proc. of IEEE, COMPUMAG - Gênes 1983

# AUTORISATION de SOUTENANCE

VU les dispositions de l'article 3 de l'arrêté du 16 avril 1974,

VU les rapports de présentation de
. MME BECKER, Chargée de recherche et de M. IVANES, Professeur
. M. AMIET, Ingénieur DRET

**Monsieur PION Gilbert**

est autorisé à présenter une thèse en soutenance pour l'obtention du diplôme de DOCTEUR-INGENIEUR, spécialité "Génie électrique".

Fait à Grenoble, le 21 août 1984

Le Président de l'I.N.P.-G

D. BLOCH
Président
de l'Institut National Polytechnique
de Grenoble

*P.O. le Vice-Président,*

## *RESUME*

*Des méthodes d'agrégation sont appliquées à des logiciels d'éléments finis dans deux directions. Dans une première partie, on étudie les possibilités de parallélisme. Après avoir donné une description des types d'architecture parallèles actuellement disponibles, on définit plusieurs algorithmes permettant le parallélisme. On indique une méthode d'analyse des performances de ces algorithmes utilisant une simulation d'une architecture sur laquelle ils seront implantés. Les résultats sont développés dans le cas d'un réseau local. La seconde partie est consacrée à des méthodes adaptatives à plusieurs niveaux de discrétisation de type multigrilles. Pour la résolution des systèmes linéaires, on obtient d'importantes accélérations de convergence et l'on peut définir des outils pour le maillage adaptatif.*