Propriétés algorithmiques des simulations de Smartgrid sur CPU-GPU

Béatrice Thomas

GDR Groupement de recherche
SEEDS Systèmes d'énergie électrique dans leurs dimensions sociétales
JCGE 24 Jeunes Chercheurs en Génie Électrique

Propriétés algorithmiques des simulations de Smartgrids sur CPU-GPU

Béatrice THOMAS *
* SATIE, CNRS, ENS Paris Saclay, ENS Rennes
Gif-sur-Yvette 91190 France
beatrice.thomas@ens-rennes.fr

RESUME - L'implémentation massive de production décentralisée renouvelable nécessaire à la transition énergétique nécessite de nouveaux modes de gestion pour le réseau. En effet le nombre d'agents intervenant augmentera de manière drastique et les difficultés de prévoir leur comportement imposera des calculs plus fréquents. La décentralisation du calcul permettra à la gestion de passer à l'échelle. Cependant les simulations de ce type de gestion seront confrontées à des temps de calculs prohibitifs lorsque les dimensions du problème augmenteront. Cet article présente une démarche d'Adéquation Algorithme Architecture pour la résolution du problème de la complexité calculatoire de marché Pair à Pair. Plusieurs implémentations de différents algorithmes ont été réalisées, analysées, optimisées et comparées. Les algorithmes de décentralisation d'Alternating Direction Method of Multipliers (ADMM) et de Proximal Atomic Coordination (PAC) ont été implémentés avec différents partitionnements sur une architecture dotée d'un processeur central et graphique (CPU-GPU). Dans le cas spécifique du marché Pair à Pair, l'utilisation d'une ADMM sous forme de consensus converge plus rapidement (en temps et nombre d'itérations) que le PAC. La parallélisation accélère les calculs, mais reste plus lent que l'utilisation de l'ADMM.

ABSTRACT - The growth of distributed energy resources implies a rise of constraints on the electricity market computation. Indeed, the increasing number of agents and their difficult forecasts may overload the system operator. The decentralization of computation may allow management to scale. Nevertheless, the simulation of this kind of management will face prohibitive computation time as the size increases. This article presents a hardware-software co-design approach to solve this issue on a single machine. Several implementations of different algorithms will be done, analyzed, optimized, and compared. The decentralization algorithms of the Alternating Direction Method of Multipliers (ADMM) and of Proximal Atomic Coordination (PAC) has been partitioned on Central and Graphic Processing Units (CPU-GPU). In the specific case of the Peer-to-Peer market, the use of an consensus ADMM converges faster (in time and number of iterations) than the PAC. Parallelization speeds up calculations, but remains slower than using ADMM.

MOTS-CLES - Adequation Algorithme Architecture; Marché Pair à pair; ADMM; PAC; GPU; OpenMP.

1. Introduction

Afin d'atteindre la neutralité carbone en 2050, le réseau électrique devra se constituer d'une majorité de moyen de production renouvelable [1]. Afin de garantir son bon fonctionnement et son équilibre malgré l'intermittence des énergies renouvelables, d'autres flexibilités notamment du côté consommateur (maison intelligente, voiture électrique, stockage, gestion des charges [2]) devront être utilisées. Ainsi le gestionnaire de réseau devra résoudre de nombreux défis. Il devra procéder à des calculs sur des cas plus larges à cause de la multiplication de petites sources de productions renouvelables. Les calculs seront plus fréquents à cause de la variabilité des dites productions et seront plus complexes, le gestionnaire ayant aussi des leviers d'actions du côté des consommateurs. Tous ces changements imposent une modification des mécanismes de gestion du réseau électrique. En effet, résoudre ce type de problème deviendra de plus en plus difficile avec l'augmentation de sa taille, jusqu'à devenir impossible à résoudre de manière centralisée [3].

Pour résoudre les limitations en terme de communication et de calcul, une approche décentralisée a été proposée dans la littérature [4]. Dans cette approche, la résolution nécessite plus de communications et de calculs mais ceux-ci sont réparties sur l'ensemble des agents. Ainsi l'utilisation d'un réseau de communication [5] permet de réduire la charge de calcul des agents pour permettre une résolution rapide. Parmi toutes les configurations possibles, le Marché Pair à Pair repose sur le fait qu'il n'y a plus aucun agent pour coordonner les échanges et les calculs. Le fait que chaque agent puisse être relié à n'importe quel autre agent permet au marché pair à pair d'être la représentation formelle de toutes les autres configurations [6] et d'inclure des mécanismes particuliers de type préférence hétérogène entre les agents.

Ainsi, une résolution en temps réel devient possible. Cependant avant d'être effectivement implémentés, de nombreux défis doivent être résolus pour le marché P2P [7]. Ainsi, de nombreuses simulations - pour le gestionnaire de réseau ou les chercheurs - sont nécessaires pour diverses raisons telles que l'identification de nouvelles règles du marché, ses performances, son dimensionnement. La détermination de la sûreté de la gestion du réseau doit être garantie même lors d'évènement extrême [8] ou lors du déploiement à grande échelle (avec de grands cas d'étude) [9] tout en conservant une cohérence temporelle de la simulation. Ainsi, la simulation devra simuler des cas de grandes dimensions sur une grande période de temps. Celles-ci étant réalisées avant toute implémentation, les simulations n'auront pas accès à la puissance de calcul distribuée des agents. Avec l'utilisation d'une unique unité de calcul les problèmes de complexité de calcul reviennent de manière d'autant plus exacerbée que le problème à simuler est décentralisé. Le simulateur ainsi

créé pourra permettre ensuite d'entraîner des Intelligences Artificielles; ou être mis à jour via des données en temps réel pour réaliser un jumeau numérique.

Ce verrou scientifique nécessite le suivi d'une démarche adéquation algorithme architecture afin de réduire les temps de calcul. La comparaison de différents algorithmes tel que *Alternating Direction Method of Multipliers* [10] (ADMM) et *Operator Splitting Quadratic Program* [11] (OSQP) dans [12] avec leur implémentation sur une structure hétérogène CPU-GPU [13] a montré qu'une grande accélération est possible. Cependant la grande accélération due au GPU est peut être surestimée [14]. En effet la version sur CPU n'était pas parallélisée. Ainsi cet article utilisera des méthodes de parallélisation logicielle de type OpenMP pour aussi paralléliser sur CPU. De plus des articles récents [15], [16] présentent le *Proximal Atomic Coordination* (PAC) un algorithme plus rapide à converger que l'ADMM. Ce nouvel algorithme doit donc être pris en considération dans l'étude pour aboutir à une Adéquation Algorithme Architecture. La contribution de ce papier consiste en l'implémentation et la comparaison des algorithmes de décentralisation **ADMM** et **PAC** sur une architecture hétérogène CPU-GPU.

La suite de cet article est organisée comme suit. Tout d'abord la section II présentera le problème de marché pair à pair ainsi que les différents algorithmes utilisés pour sa résolution. La partie suivante sera consacrée à l'étude de complexité et à la parallélisation en utilisant respectivement cuda sur GPU et OpenMP sur CPU. La dernière partie sera consacrée à la présentation et l'analyse des résultats obtenus. Le code ayant permis l'obtention de ces résultats est disponible en open source : https://gitlab.com/satie.sete/algorithm_p2p_jcge.

2. Modélisation

2.1 Marché pair à pair

On considère Ω l'ensemble des N agents du marché. Chaque agent a M_n voisins avec lesquels il peut échanger, tel que $M = \sum_n M_n$ est le nombre total d'échanges. Le marché que l'on considérera sera multi-énergies, c'est à dire que l'on suppose qu'en plus de la puissance active P, les agents s'échangent aussi une autre énergie noté Q qui peut être la puissance thermique par exemple. On supposera pour simplifier que P_n et Q_n sont indépendants. On négligera le réseau physique (pas de flux ou de pertes). Cependant des interactions indirectes avec le gestionnaire de réseau sont possibles. Les agents ont des bornes sur les échanges économiques ($t^{p,q}$) notées lb et ub dépendant de leurs types (consommateur, producteur ou prosumers) pour les deux types d'énergies. La puissance totale d'un agent (p_n, q_n) est égale à la somme de ses échanges. On considérera que tous les agents flexibles peuvent accéder à un moyen de stockage pour nous permettre de supposer une flexibilité continue des agents. Ainsi chaque agent a une fonction coût représentant ce qu'ils veulent consommer ou produire, P_0 ou Q_0 et qu'est ce que cela leur "coûterait" en perte de *welfare* (bonheur, bien-être) ou financière de s'éloigner de ce minimum. En se basant sur [17], [18], on supposera une fonction coût quadratique.

$$g_n(p_n, q_n) = \frac{1}{2}a_n^p p_n^2 + b_n^p p_n + \frac{1}{2}a_n^q q_n^2 + b_n^q q_n \tag{1}$$

Chaque agent possède aussi des limites sur les puissances totales qu'il peut échanger, notées respectivement $\underline{p}_n$, $\overline{p}_n$ et $\underline{q}_n$, $\overline{q}_n$ qui permettent aussi de représenter leur flexibilité. Comme le marché cherche à atteindre l'équilibre de puissance sans prendre en compte les pertes dans le réseau. Un agent a été ajouté pour échanger sur le marché pour "racheter les pertes" dans les lignes lorsqu'il y en a. Cet agent a une fonction coût nulle et $\underline{p}_n = \overline{p}_n = P_{pertes}$ (idem pour q_n). Le problème de marché pair à pair s'écrit ainsi :

$$\min_{T,P,Q} \quad \sum_{n\in\Omega}\Big(g_n(p_n,q_n) + \sum_{m\in\omega_n}\beta_{nm}t_{nm}^p\Big) \tag{2a}$$

$$\text{t.q.} \quad T = -{}^tT \qquad (\Lambda) \tag{2b}$$

$$(p_n, q_n) = \Big(\sum_{m\in\omega_n} t_{nm}^p, \sum_{m\in\omega_n} t_{nm}^q\Big) \qquad (\mu) \qquad n\in\Omega \tag{2c}$$

$$\underline{p_n} \le p_n \le \overline{p_n} \qquad n\in\Omega \tag{2d}$$

$$\underline{q_n} \le q_n \le \overline{q_n} \qquad n\in\Omega \tag{2e}$$

$$ub^{p,q} \le T_{nm}^{p,q} \le lb^{p,q} \qquad n\in\Omega \tag{2f}$$

L'objectif de cette résolution est de trouver les échanges t_{nm} (pour les deux types d'énergie) optimum entre les agents minimisant les fonctions objectifs $g_n(p_n, q_n)$. A ces fonctions coûts peuvent être ajoutées des préférences hétérogènes sur les échanges $\beta_{nm}t_{nm}^p$. Ce terme peut aussi représenter un terme exogène déterminé par les gestionnaires de réseau [17]. Les contraintes sont l'anti-symétrie des échanges (2b) car ce qui est acheté doit être vendu pour être à l'équilibre, et la puissance de chaque agent est la somme de ses échanges (2c).

On remarque que la résolution de la minimisation de (t^p, p) est indépendante et de la même forme que celle de (t^q, q). Ainsi pour résoudre le problème on considérera que le marché est composé de 2 marchés mono-énergie qui n'échangent pas entre eux avec $\beta = 0$ pour le marché de Q. Ainsi, pour la suite afin de simplifier les notations, on ne conservera que t^p noté t et p, mais les deux seront considérés.

2.2 Algorithmes

2.2.1 Alternating Direction Method of Multipliers : ADMM

L'utilisation de l'algorithme ADMM [10] sous sa forme de consensus permet de relaxer la contrainte d'anti-symétrie (2b) et de séparer le problème en différents problèmes itératifs k paramétrés par le facteur de pénalité ρ pour chaque agent n :

$$\begin{aligned} T_n^{k+1} = \underset{T_n}{\operatorname{argmin}} \quad & g_n(p_n) + \sum_{m\in\omega_n} \beta_{nm} t_{nm} \\ & + \sum_{m\in\omega_n} \left(\frac{\rho}{2} (t_{nm} - \frac{t_{nm}^k - t_{mn}^k}{2} + \frac{\lambda_{nm}^k}{\rho})^2 \right) \\ \text{t.q.} \quad & \text{(2c)} - \text{(2f)} \end{aligned} \tag{3}$$

T_n étant le vecteur contenant les offres d'échanges de l'agent n. A chaque pas, les multiplieurs de Lagrange sont mis à jour pour garantir le respect de la contrainte relaxée.

$$\lambda_{nm}^{k+1} = \lambda_{nm}^k + \frac{\rho}{2}(t_{nm}^{k+1} + t_{mn}^{k+1}) \tag{4}$$

La résolution de (3) peut se faire via l'utilisation d'un solveur tel que OSQP. Cependant dans cet article on appliquera une ADMM sous la forme de partage qui implique une autre résolution itérative appelée problème local qui lui sera résolu via une forme *closed*. Les détails de l'algorithme et de son implémentation peuvent être trouvés dans [12] ou [13]. L'algorithme en résultant est représenté dans Alg. 1.

input : Study case, ρ, ϵ and kmax
output : T, Λ, Residuals
while $err > \epsilon$ *and* $k < kmax$ **do**
  for *n=1,...N* **do** // agents
    $T_n^{k+1} \leftarrow$ (3);
  end
  Communication T ;
  for *n=1,...N* **do** // agents
    for *j=1,...m* **do** // peers
      $\Lambda_{nm} \leftarrow$ (4) ;
    end
  end
  $Res_R \leftarrow ||T^{k+1} +^t T^{k+1}||$;
  $Res_S \leftarrow ||T^{k+1} - T^k||$;
  $err \leftarrow max(Res_R, Res_S)$;
end

Algorithme 1 : Algorithme de l'ADMM

2.2.2 Proximal Atomic Coordination : PAC

L'algorithme de PAC est un autre moyen de décentraliser avec une vitesse de convergence (diminution de l'erreur en fonction de l'itération) plus élevée que l'ADMM. Cet algorithme permet aussi de mieux protéger la confidentialité (*privacy*) des agents intervenant dans le système. En effet le **PAC** ne transmet pas directement les optimums de chaque itération pour chaque agent (mais une version modifiée). Il transmet également une version modifiée des variables duales. Ici on parallélise encore sur chaque agent n qui est constitué :

- de ses propres variables (p_n et t_{nm} pour tous ses voisins m);
- des copies des variables dont il a besoin ($a_{mn} = t_{mn}$).

L'ensemble de ses variables et des copies des variables utiles est noté $x_n = (p_n, t_{nm}, a_{mn})$ pour l'agent n. Pour la suite, l'indice entre crochet indique à quel agent appartient la variable. Afin que les copies soient égales aux vraies valeurs, la matrice B est définie tel que $Bx = 0$ représente la contrainte $a_{mn}^{[n]} = t_{mn}^{[m]}$ pour tous les agents et pairs. La variable ν est la variable duale associée à cette contrainte. Le Lagrangien à optimiser pour chaque agent n est :

$$\begin{aligned} L_n &= f_n(x_n) + \mu_n^T G_n x_n + \nu^T B^n x_n \\ &= g(p_n) + \sum_m \beta_{nm} t_{nm} + \mu_1 (p_n - \sum_m t_{nm}) \\ &+ \sum_{p>1} \mu_p (t_{nm} + a_{mn}) + \sum_{m=1}^{M_n} \nu_m^n a_{mn} - \nu_n^m t_{nm} \end{aligned} \tag{5}$$

entrée : G, H, d, peers
sortie : x_{min}

while $err > \epsilon$ *and* $j < j_{max}$ **do**
  for *n=1,...N* **do** // all agents in parallel
    $x_n^{j+1} \leftarrow$ argmin (5) + (6);
    $\hat{x}_n^{j+1} \leftarrow x_n^{j+1} + \alpha_n^{j+1}(x_n^{j+1} - x_n^j)$;
    $\mu_n^{j+1} \leftarrow \hat{\mu}_n^j + \rho_n\gamma_n(G_n\hat{x}_n^{j+1})$;
    $\hat{\mu}_n^{j+1} \leftarrow \mu_n^{j+1} + \phi_n^{k+1}(\mu_n^{j+1} - \mu_n^j)$;
  end
  Communication $\hat{x}$;
  for *n=1,...N* **do** // all agents in parallel
    $\nu_n^{j+1} \leftarrow \hat{\nu}_n^j + \rho_n\gamma_n B_n\hat{X}^{j+1}$;
    $\hat{\nu}_n^{j+1} \leftarrow \nu_n^{j+1} + \theta_n^{j+1}(v_n^{j+1} - v_n^j)$;
  end
  Communication $\hat{\nu}$;
  for *n=1,...N* **do** // all agents in parallel
    Update α_n, ϕ_n, θ_n and γ_n ;
    Compute err
  end
end

Algorithme 2 : PAC

Ce Lagrangien peut être augmenté pour améliorer la convergence en y ajoutant les termes suivants :

$$\begin{aligned} L_\rho &= \frac{\rho_n\gamma_n}{2}||G_nx_n||_2^2 + \frac{\rho_n\gamma_n}{2}||B_nx_n||_2^2 + \frac{1}{2\rho_n}||x_n - x_n^k||_2^2 \\ &= \frac{\rho_n\gamma_n}{2}[(p_n - \sum_m t_{nm})^2 + \sum_{p>1}(t_{np} + a_{pn})^2)] \\ &+ \frac{\rho_n\gamma_n}{2}[\sum_{p=1}(a_{pn} - t_{pn}^{[p]})^2] + \frac{1}{2\rho_n}[\sum_p (x_n - x_n^k)^2] \end{aligned} \tag{6}$$

La somme des équations (5) et (6), que l'on appellera problème local à résoudre, est une optimisation quadratique avec pour seule contrainte des bornes sur les variables. Ainsi il suffit de trouver le minimum via la résolution d'un système linéaire et de projeter la solution dans ses bornes admissibles. Le système ne changeant pas au cours des itérations, la résolution se fera via l'inverse d'une matrice réalisée une seule fois à l'initialisation. L'algorithme est donc le suivant Alg. 2.

3. Adéquation Algorithme Architecture

Le principe de cette approche consiste à analyser les différents algorithmes afin de permettre leur implémentation en prenant en compte a priori les caractéristiques des architectures cibles. Cela passe par une modélisation et une étude qui va explorer l'espace des partitionnements et placements des calculs sur une architecture parallèle. La définition d'un modèle est basée sur l'utilisation d'un ensemble de vecteurs de tests logiciels et algorithmiques et une étude analytique des performances.

3.1 Complexité des blocs fonctionnels

La première étape de la démarche Adéquation Algorithme-Architecture est de séparer chaque algorithme en plusieurs blocs fonctionnels. Cette opération permet de remplir plusieurs objectifs tel que l'analyse des dépendances entre les différentes parties de l'algorithme en terme de séquentialité et de transfert de données ; la détermination de la complexité en série ou parallèle en calcul et en mémoire et la charge de calcul de chaque bloc par le calcul ou la mesure, pour en déduire l'intensité algorithmique; et enfin l'identification des goulots d'étranglement. On définira la complexité par le nombre d'opérations réalisées (qu'elles soient des sommes, multiplications ou divisions) en fonction des dimensions du problème. On ne s'intéressera qu'à la variation asymptotique de l'ordre de grandeur de cette valeur noté $O(...)$. Par exemple une complexité en $O(N^2)$ implique que pour un doublement du nombre d'agent le nombre de calcul est quadruplé. La notation en $O(...)$ nous permet de négliger les facteurs multiplicatifs et les termes de puissance plus faible, *i.e.* $2N^2 + 100N$ calculs devient $O(N^2)$.

En notant $M_{max} = \max_n M_n$ le nombre maximal de voisin d'un agent pouvant varier entre 1 et N, on peut évaluer la complexité théorique des différentes étapes de l'algorithme résumé dans le Tab. I. Dans le cas d'un marché pair à pair complètement connecté on a $M_n = O(N)$ et $M = O(N^2)$. **L'initialisation** consiste en la définition des différents vecteurs ou matrices pour les deux algorithmes. Pour le PAC, on y rajoute $2 * N$ inversions de matrices de taille $1 + 2 * M_n$ (une par agent et par puissance) pour résoudre le problème local. Ces inversions seront réalisées par la bibliothèque Eigen. **Le problème local** consiste en une résolution itérative pour l'ADMM. Celle-ci consiste en des opérations linéaires sur des vecteurs (additions et multiplications terme à terme) et une moyenne. On suppose que le

Tableau I: Blocs fonctionnels et complexité en série et en parallèle

Bloc Fonctionnel	Contenu	ADMM		PAC	
		CPU	GPU	CPU	GPU
FB0	initialisation	$O(M)$	$O(M)$	$O(N \cdot M_{max}^3 + M)$	$O(N \cdot M_{max}^3 + M)$
FB1	problème local	$O(M)$	$O(log(M_{max}))$	$O(N \cdot M_{max}^2)$	$O(log(M_{max}))$
FB2	variables duales	$O(M)$	$O(1)$	$O(M)$	$O(log(M_{max}))$
FB3	calcul des résidus	$O(M)$	$O(log(N))$	$O(M)$	$O(log(N))$
FB4	Récupération des résultats	$O(M)$	$O(M)$	$O(M)$	$O(M)$
FB5	Mise à jour	$O(M)$	$O(1)$	$O(M)$	$O(1)$
FB1&2&3	Ensemble calcul	$O(M)$	$O(log(N))$	$O(N \cdot M_{max}^2)$	$O(log(N))$
	Ensemble	$O(M)$	$O(M)$	$O(M + N \cdot M_{max}^3)$	$O(M + N \cdot M_{max}^3)$

nombre d'itération de ce bloc ne dépend pas du nombre d'agent. Pour PAC la résolution consiste en la multiplication de la matrice inversée et d'un vecteur pour chaque agent. **Le calcul des résidus** nécessite la recherche d'un maximum dans un vecteur pour les deux méthodes. L'ensemble des opérations restantes (**la mise à jour des variables duales**, **la récupération des résultats**, **mise à jour** entre deux pas temporels) correspondent pour les deux méthodes à des opérations linéaires sur des vecteurs (en effet les matrice G et B du PAC sont assez creuses pour permettre de réduire la complexité).

On peut remarquer dans le Tab. I que la complexité de l'ADMM est la complexité minimale atteignable (M étant le nombre de variables à trouver). Cependant cela se fait au prix de deux boucles imbriquées pouvant rendre assez long à calculer l'ensemble. Même si les étapes **FB 0 & 1** ont une plus grande complexité pour PAC, cela ne nous permet pas de conclure sur les performances des algorithmes. En effet l'étape **FB 0** n'est faite qu'une fois et l'étape **FB 1** n'est faite qu'une fois par itération pour PAC, alors qu'elle est faite jusqu'à plusieurs milliers de fois pour l'ADMM.

Dans le cas d'un marché faiblement connecté, on considère M_{max} indépendant du nombre d'agents. Ainsi la complexité du PAC et de l'ADMM deviennent identiques puisque $M = O(N)$, mais cela ne sera pas le cas ici.

3.2 Complexité de la version parallélisée

En utilisant le théorème de Brent [19] il est possible de calculer la complexité théorique de l'application en parallèle :

$$C_{para} = O(\frac{o}{p} + t) \tag{7}$$

avec o le nombre total d'opérations (donc la complexité en série, sauf en cas de redondance des calculs), p le nombre de processeurs sur lesquels on parallélise, et t le nombre d'étapes. Pour la suite on notera $T = N_b * N_{t/b}$ avec T le nombre total de *thread*, N_b le nombre de bloc, et $N_{t/b}$ le nombre de *thread* par bloc. On remarque pour la parallélisation sur CPU, on a p correspondant au nombre de coeur. Ce nombre étant assez faible, la parallélisation sur CPU ne permet pas de faire diminuer la complexité. Lors de la parallélisation sur GPU, $p = T$ vaut plusieurs milliers. Ce qui permet d'effectivement diminuer la complexité lorsque le nombre d'agents est de cet ordre de grandeur [1].

Les opérations linéaires sur des vecteurs, tel que les sommes, soustractions, multiplications et divisions termes à termes sont parallélisables efficacement sur GPU. En effet en ayant p égal au nombre de termes ces opérations de complexité linéaire en série deviennent en temps constant sur GPU. Les moyennes ou recherches de maximum d'un vecteur (aussi appelé réductions) sont plus difficilement parallélisables. En utilisant la mémoire partagée et la synchronisation dans un bloc, il est possible d'obtenir une complexité logarithmique pour ces opération. L'**ADMM** est constitué de ces deux types d'opération, ce qui nous permet d'obtenir la complexité en parallèle dans le Tab. I. Le lecteur intéressé trouvera plus de détails dans [13]- [12].

En plus des opérations précédentes, le **PAC** nécessite des inversions de matrices et une multiplication entre une matrice et un vecteur. L'inversion étant réalisée par Eigen, sa complexité ne change pas dans la version parallélisée.

[1]Ceci n'est vrai qu'en prenant en compte les particularités du GPU permettant la parallélisation, notamment en limitant les transferts mémoires CPU-GPU et en évitant les conflits d'accès mémoire (*race condition*, accès non coalescent ou *bank conflict*)

Pour paralléliser la multiplication matrice-vecteur, on peut créer un bloc de threads par ligne. Chaque bloc devra donc calculer une multiplication terme à terme de deux vecteurs, puis une somme sur tous les termes. La première étape est une opération linéaire et la deuxième est une réduction. En faisant ainsi la multiplication matrice-vecteur a une complexité logarithmique. La complexité du **PAC** sur GPU peut être vue dans Tab. I. Il est important de noter que cette manière de paralléliser réduit au maximum la complexité mais nécessite énormément de ressource du GPU.

Certaines opérations de mise en forme sont réalisées sur CPU, résultant en une complexité apparente identique pour les étapes **FB 0** et **FB 4**. Par exemple l'inversion des matrices restent réalisée par Eigen. Cependant ces opérations ne sont réalisées qu'une seule fois par résolution. Ainsi lorsque ces étapes deviendront négligeables devant le temps de résolution, l'utilisation du GPU devrait permettre de réduire la complexité de ces méthodes.

4. Résultats

Dans cette partie les différents algorithmes qui sont ADMM et PAC implémentés sur CPU en série, sur CPU parallélisés avec openMP ou sur GPU parallélisés avec Cuda seront évalués et comparés. Les simulations sont réalisés sur un GPU de Nvidia GeForce RTX 3060. Le CPU associé est un AMD RYZEN 5 5600H à 3,3GHz avec 6 coeurs et 12 processeurs logiques. La méthode **PAC** est évaluée avec les coefficients α_n, ϕ_n et θ_n nuls. Ce sont ces coefficients qui permettent la confidentialité, mais pour comparer les performances de convergence avec l'ADMM il est plus intéressant que les deux méthodes ne cherchent pas à cacher des informations.

4.1 Cas fixe

Dans cette partie les algorithmes sont évalués sur des cas fixes de la littérature. Ceci permettra la répétabilité du travail, les cas étant accessible en open Source. Ainsi les cas seront des cas Matpower [20] adaptés afin de permettre de faire un marché tel que celui dans [17] (cas 39 noeuds). Le fait d'utiliser des cas de la littérature permettra d'étudier les algorithmes sur des cas réalistes. La précision demandée est de 10^{-3} avec un nombre maximal d'itérations de 50 000. Pour les méthodes ADMM, le facteur de pénalité est fixé à 1. Pour PAC, les valeurs optimales sont déterminées via les valeurs propres des matrices des contraintes. Celles-ci ne dépendant que du nombre d'agents et de la proportion de consommateur, une heuristique a été utilisée pour trouver ces valeurs. Les tests ayant permis la détermination de cette heuristique sont disponibles sur le dépôt. Cependant l'optimalité n'est vraie que pour des fonctions coûts strictement convexes ce qui n'est pas le cas ici pour les échanges. Ainsi il n'y a aucune garantie d'optimalité des paramètres. Le **Cas 39 noeuds** ne considère que de la puissance active. Tous les autres cas calculent un marché de puissance active et de puissance réactive.

Tableau II: Temps (s) (nombre d'itérations) pour converger pour les différentes algorithmes, cas et architecture

Cas		ADMM			PAC		
Nom	taille	CPU	CPUMP	GPU	CPU	CPUMP	GPU
Matpower 10	24	$3e^{-3}$ (46)	$8e^{-3}$ (46)	0.9 (46)	0.1 (1057)	0.06 (1057)	0.3 (1057)
Cas 39 noeuds	31	0.07 (235)	0.04 (235)	1.3 (33)	5.3 (25579)	2.8 (25579)	9.0 (25237)
TestFeeder	114	0.05 (56)	0.03 (56)	0.7 (63)	12 (3371)	5 (3371)	1.6 (3371)
Matpower 85	122	0.03 (13)	0.01 (13)	1.3 (21)	4.1 (934)	2.0 (934)	0.5 (934)
Matpower 118	308	21 (711)	8.2 (711)	1.5 (2351)	$3.5e^3$ (50 000)	$1.2e^3$ (50 000)	75 (50 000)

Le cas **Matpower 118** a la particularité de normaliser avec $P_0 = 100MW$. Ceci provoque des fonctions coûts $g_n(p_n, q_n)$ avec des puissances plus faibles et coefficients plus élevés. Par exemple si $a_n = 0.1\$/MW^2$ cela donne $a_n = 10000\$/pu^2$. Pour ce cas, $\rho = 1000$ a été choisi pour l'ADMM afin de permettre la convergence. La méthode **PAC** ne converge pas dans le nombre permis d'itérations mais les résidus sont faibles ($5e^{-3}$). On peut voir sur ces cas l'augmentation du temps de calcul avec l'augmentation des dimensions. On remarque que la méthode **PAC** a besoin de bien plus d'itération et de temps pour converger que l'ADMM (même en changeant les paramètres de PAC). On remarque aussi qu'à part pour le cas le plus petit, l'utilisation de openMP permet de réduire les temps de calcul. Le GPU devient l'architecture la plus rapide pour la méthode PAC en étant même 40 (respectivement 16) fois plus rapide que le CPU (respectivement OpenMP) sur le dernier cas.

4.2 Etude de passage à l'échelle

Dans cette partie les différentes implémentations seront évaluées dans un ordre aléatoire sur 50 cas générés aléatoirement pour chaque nombre d'agent testé. Comme l'objectif est d'étudier le passage à l'échelle et non pas la convergence sur

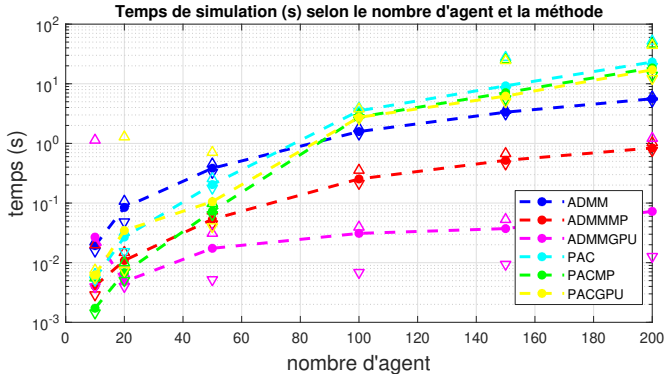

Figure 1: Évolution des temps de calcul avec le nombre d'agent

des cas potentiellement absurdes, le nombre maximal d'itérations est limité à 50. Pour rappel on considère un marché Pair à Pair avec deux types de puissances échangées. Ainsi un cas généré de 10 agents correspond (en dimension du problème) à un marché mono-énergie de 20 agents.

La Fig. 1 montre les temps moyens, maximum et minimum pour chaque algorithme selon le nombre d'agents. On peut y voir que l'algorithme **PAC**, et l'**ADMM** sur CPU ont des temps de calcul augmentant rapidement avec les dimensions du problème. On peut voir que pour un nombre fixé d'itérations **PAC** est plus rapide que l'**ADMM** à petite dimension. La parallélisation permet bien de réduire légèrement les temps de calcul. Cependant le faible nombre d'itérations ne permet pas ici de retrouver les fortes accélérations de la partie précédente.

Tableau III: Étude du passage à l'échelle des différentes implémentations

Méthode		ADMM			PAC		
	Architecture	CPU	CPUMP	GPU	CPU	CPUMP	GPU
nombre	temps (s) $N_1 = 20$	0.08	0.01	0.005	0.03	0.007	0.04
maximal	temps (s) $N_2 = 200$	5.6	0.85	0.08	25	18	17
d'itérations	ratio ($\frac{t_2}{t_1}$ par iter)	69	81	15	950	2800	490
à 50	complexité α	1.8	1.9	1.2	3.0	3.4	2.7
nombre	temps (s) $N_1 = 20$	0.26	0.05	0.06	0.17	0.05	0.09
maximal	temps (s) $N_2 = 200$	19	3.2	0.23	95	40	17
d'itérations	ratio ($\frac{t_2}{t_1}$ par iter)	61	51	2.8	560	870	190
à 500	complexité α	1.8	1.7	0.44	2.7	2.9	2.3

Le Tab. III regroupe l'évaluation de la complexité des différentes méthodes. Ici on considérera la complexité α tel que si la complexité est en $O(N^\alpha)$ la variation du temps entre 2 cas de taille N_1 et N_2 nous donne $\alpha = \frac{\log(t_2/t_1)}{\log(N_2/N_1)}$. Pour le calcul de la complexité, les temps utilisés sont ceux par itération. On remarque bien que l'utilisation du GPU permet bien de réduire la complexité apparente de l'ensemble de l'algorithme. Cependant le faible nombre d'itérations ne permet pas d'atteindre la complexité théorique obtenue sans considérer les étapes de gestion de données pré et post calcul. C'est pourquoi des mesures ont été réalisées à 20 et 200 agents avec 500 itérations. Les résultats ont été regroupés dans le tableau précédent. On peut y avoir que le fait d'augmenter le nombre d'itération permise diminue la complexité mesurée.

5. Conclusion

Cet article a permis d'étudier et de comparer différents algorithmes pour résoudre le problème d'optimisation du marché Pair à Pair. La comparaison de l'efficacité de la parallélisation sur CPU via Open-MP demandant peu d'efforts d'implémentation par rapport à celle sur GPU, plus chronophage, a été réalisée. Ainsi même si le **PAC** est sensé converger plus rapidement que l'ADMM, ce n'est pas le cas pour un marché Pair à pair. En effet **PAC** relaxe les contraintes d'antisymétrie et la relation échange-puissance là où l'ADMM sous forme de consensus puis de partage

permet de directement prendre en compte ces contraintes. Ainsi ces ADMM convergent bien plus rapidement qu'une ADMM dans le cas général. D'un autre côté on a pu montrer que l'utilisation d'openMP permettait de réduire les temps de calcul avec peu d'effort d'implémentation. Mais l'utilisation du GPU reste nécessaire lorsque les dimensions deviennent très grandes. Cependant ces résultats ne peuvent être généralisés sur tous les problèmes du génie électrique. Ainsi il pourrait être intéressant de les comparer sur des problème plus complexes comme l'AC-Optimal Power Flow ou un marché Endogène non linéarisé. Une étude des performances des algorithmes avec des fonctions coûts strictement convexes pour les échanges et les puissances permettrait de compléter cette étude.

References

[1] RTE. "Conditions and Requirements for the Technical Feasibility of a Power System with a High Share of Renewables in France Towards 2050"

[2] Saad Al-Sumaiti A, et al. "Smart Home Activities: A Literature Review, Electric Power Components and Systems", 42:3-4, 294-305,(2014) doi: https://doi.org/10.1080/15325008.2013.832439

[3] Dong A, et al. "Convergence analysis of an asynchronous peer-to-peer market with communication delays". Sustainable Energy, Grids and Networks, 26, 100475. 2021

[4] Sousa T, et al. "Peer-to-peer and community-based markets: A comprehensive review", Renewable and Sustainable Energy Reviews, Volume 104, 2019, Pages 367-378,ISSN 1364-0321 https://www.sciencedirect.com/science/article/pii/S1364032119300462

[5] Garau M, et al. "Co-simulation of smart distribution network fault management and reconfiguration with LTE communication". Energies, 2018, 11.

[6] Baroche T, et al. "Prosumer markets: A unified formulation". PowerTech (pp. 1-6). IEEE. 2019

[7] Tushar W, et al. "Peer-to-peer energy systems for connected communities: A review of recent advances and emerging challenges". Applied Energy. 2021

[8] Ryan H, et al. "Designing resilient decentralized energy systems: The importance of modeling extreme events and long-duration power outages." Iscience (2021): 103630.

[9] Tushar W, et al. "Peer-to-peer trading in electricity networks: An overview". IEEE Transactions on Smart Grid, 11(4), 3185-3200. 2020

[10] Boyd S, et al. "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers"

[11] Stellato B, et al. "OSQP: an operator splitting solver for quadratic programs". Mathematical Programming Computation https://doi.org/10.1007/s12532-020-00179-2

[12] Thomas B, et al. "Hardware–software codesign for peer-to-peer energy market resolution." Sustainable Energy, Grids and Networks 35 (2023).

[13] Thomas B, et al. "Optimization of a peer-to-peer electricity market resolution on GPU." 2022 IEEE International Conference on Electrical Sciences and Technologies in Maghreb (CISTEM). Vol. 4. IEEE, 2022.

[14] Lee V, et al. "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU." Proceedings of the 37th annual international symposium on Computer architecture. 2010.

[15] Romvary J J, et al. "A proximal atomic coordination algorithm for distributed optimization." IEEE Transactions on Automatic Control 67.2 (2021): 646-661.

[16] Ferro G, et al. "A distributed-optimization-based architecture for management of interconnected energy hubs." IEEE Transactions on Control of Network Systems 9.4 (2022): 1704-1716.

[17] Baroche T, et al. "Exogenous Cost Allocation in Peer-to-Peer Electricity Markets." IEEE Transactions on Power Systems, Institute of Electrical and Electronics Engineers, 2019, 34 (4), pp.2553 - 2564. ffhal-01964190f

[18] G Hug, et al. "Consensus + Innovations Approach for Distributed Multiagent Coordination in a Microgrid," in IEEE Transactions on Smart Grid, vol. 6, no. 4, pp. 1893-1903, July 2015, doi: https://doi.org/10.1109/TSG.2015.2409053.

[19] Brent R. P. 1974. The Parallel Evaluation of General Arithmetic Expressions. J. ACM 21, 2 (April 1974), 201–206.

[20] Zimmerman R D et al. "Matpower: Steady-State Operations, Planning and Analysis Tools for Power Systems Research and Education,". IEEE Transactions on Power Systems, vol. 26, no. 1, pp. 12–19, Feb. 2011.