Blocked Adaptive Cross Approximation (BACA)

KANIT Zaccarie [1]*

Résumé
Ce papier présente une variante de l'algorithme de calcul matricielle par ACA qui a pour objectif de réduire le temps de calcul et l'espace mémoire de la résolution des problèmes de nature électromagnétiques par la Méthode des Moments. L'algorithme proposé ici pour accélérer le calcul est l'approximation par bloc(BACA), une solution purement algébrique, qui remplace la solution par niveaux(MLACA) qui possède un manque de scalabilité. L'algorithme par bloc repose sur l'utilisation de la décomposition QR sur plus petites matrices pour améliorer la complexité. La convergence se fait donc plus rapidement et la qualité première de la ACA est maintenu; c'est à dire que l'algorithme étant purement algébrique est indépendant du noyau de Green tant que les matrices ont une déficience de rang. De plus,cet algorithme peut encore être optimisée grâce à la parallélisation (H-BACA) de l'algorithme par bloc. Celui-ci étant scalable en multi coeurs et donc diminuant le temps de compilation, en particulier pour les matrices sparses.

Mots clés
Algorithme de résolution rapide – (ML/B) ACA–Calcul Matriciel

[1] *Étudiant en cycle d'ingénieur généraliste, IMT Atlantique*
***mail**: zaccarie.kanit@imt-atlantique.net

Table des matières

Introduction

La résolution rapide d'algorithme est un sujet important ces dernières année. Des algorithmes ont alors été développés pour viser des résolutions en temps réel ; il est donc nécessaire de développer des solutions plus rapides et moins coûteuses en ressources.Particulièrement pour avoir des résultats instantanés dans le milieu de l'encéphalographie en résolvant de façon récursive l'équation intégral de l'électromagnétique [1]. Les algorithmes de décomposition et compression de matrices peuvent alors aider à résoudre des équations de façon numérique assez rapidement. Surtout que dans ce domaine pour avoir une bonne précision il est nécessaire de stocker les informations pour de nombreuses électrodes dans des matrices $m \times n$, où m et n représentent la taille du grillage utilisé pour modéliser l'espace de travail. Toutefois, malgré ce nombre potentiel d'informations, les matrices utilisées peuvent souffrir d'insuffisance de rang. Malgré cela, les algorithmes naïfs comme la SVD a un coup dû à sa complexité en $\mathcal{O}(N^3)$ [2]. C'est pourquoi des algorithmes plus efficaces ont été développés ; comme la ACA [3] qui a donnée naissance à de nombreux algorithmes de résolution algébrique qui permettent la résolution d'équation sans connaître le noyau de la fonction de Green au préalable (pour une grande partie des fonctions de Green) [2]. De nombreux algorithmes algébriques ont alors été développé pour améliorer cette efficacité, même si ces algorithmes se basent sur les propriétés des matrices [4] et qui ne sont pas toujours applicables aux matrices utilisés.

L'étude s'est principalement faite autour de la ACA qui possède malheureusement quelques défauts, comme le comportement asymptotique des la ACA tend vers $\mathcal{O}(N^3)$ [1] pour des matrices compressées, même si la complexité est en moyenne plus correcte. Le problème est donc de savoir comment améliorer les algorithmes de résolutions rapides basés sur des solutions algébriques.

1. Méthode

Ce travaille a été effectué de mars à juin 2022 dans le cadre d'un projet d'initiation à la recherche pour les élèves du cycle ingénieur de l'IMT Atlantique ; le projet CODEV. Il a été encadré par Merlini Adrien Maître de conférence à l'IMT Atlantique et chercheur dans le Département Micro-ondes.

1.1 Documentation

Les méthodes de cette étude ont été principalement théoriques. Un travail de recherche a été fait autour de la documentation et de l'état de l'art sur des articles des 20 dernières années, depuis le début de la ACA [3]. Pourtant le travail de documentation a commencé avec l'ouvrage de Mario Bebendorf "Hierarchical Matrices" [2], un ouvrage complet sur l'algèbre autour de la ACA et d'autres problématiques algébriques.

Les supports utilisés pour le langage C++ proviennent du Site OpenClassroom et des cours sur la programmation impérative [5] et objet [6] en C. Cette utilisation du langage C, vient de la volonté d'obtenir un algorithme rapide et donc d'utiliser un langage bas niveau ; comme c'est le cas pour le projet GitHub à partir duquel l'étude a été faite ERCEM321_ATREYU [7]. Ce projet comporte en ce qui concerne l'étude : des implémentassions de la ACA et de la MLACA.

1.2 Matériel

Pour ce projet, un serveur proposé par l'encadrant avec 64 coeurs et 756Go de mémoire RAM a été utilisé pour effectuer des simulations du projet ERCEM321_ATREYU [7].

Un autre serveur a temporairement été pretté par le ResEl avec 68 coeurs et 100Go de RAM.

Ces deux serveurs tournent sous Linux et utilisent gcc pour compiler le code en C++.

1.3 Procédure

L'objectif étant de trouver des algorithmes de résolution rapide, l'aboutissement est d'obtenir une complexité. Pour cela, une méthode théorique avec confirmation expérimentale a été utilisé pour trouver la complexité asymptotique de l'algorithme BACA [8] ; par l'obtention du temps de calcul en fonction de la taille des matrices en entrée. Il est nécessaire de confirmer par l'expérience la complexité car on a vu avec la MLACA les limites d'une étude théorique lors du transfert des données entre les coeurs [4] qui peut être un processus coûteux.

Le script n'a malheureusement pas pu être terminé à temps à cause de beaucoup de bugs entre le script de la ACA déjà existant dans le projet ERCEM321_ATREYU [7] et un script de décomposition QR [9].

Les simulations effectuées pour faire un point sur l'état du code n'ont pu avoir lieu qu'une fois. Alors qu'une moyenne sur plusieurs compilation aurait été plus pour la modélisation et aurait minimisé les erreurs écarts expérimentaux.

2. Motivations

Une présentation de l'état de l'art est ici faite pour mettre en avant le potentiel que peut apporter l'algorithme de résolution par BACA. Cela permettra de voir en quoi cet algorithme est une étape supplémentaire dans le développement d'algorithmes de résolution rapide et en quoi le BACA se différencie des autres algorithmes.

2.1 La ACA

2.1.1 Présentation de l'algorithme

La ACA est purement algébrique et se base sur une approximation à bas rang d. Cette approximation revient à avoir notre matrice M qui est supposée asymptotiquement continue, alors on peut l'exprimer sous forme compressée [2] :

$$M \approx UV \tag{1}$$

Avec M une matrice $m \times n$, U une matrice $m \times d$ et V une matrice $d \times n$, avec $d \ll m,n$ et d le rang effectif de la matrice, c'est à dire le bas rang approximé ou le nombre de degrés de liberté. Le fait qu'elle se base sur une approximation de la matrice et pas du noyau la rend a-priori applicable à presque tous les noyaux [2].

L'algorithme de la ACA a alors comme objecitf de partir de $R_0 := A$ notre matrice initiale, puis trouver un pivot non nul choisis au hasard (i_k, j_k) pour avoir notre matrice résiduelle R_k qui vérifie après une contraction pour la mise à l'échelle [2] :

$$R_{k+1} = R_k - R_k[i_k, j_k]^{-1} R_k[1:m, j_k] R_k[i_k, 1:n] \tag{2}$$

On obtient tout de même de meilleurs résultats avec j_{k_0} choisi de telle sorte à ce que le coefficient $R_k[i_k, j_{k_0}]$ soit maximal.

On remarque que à chaque itération de l'algorithme la matrice résiduelle devient voit ses coefficients s'annuler, donc on peut progressivement ne pas construire la matrice complète pour économiser de l'espace mémoire[2]. On propose donc l'algorithme de la ACA.

Algorithm 1: Adaptive Cross Approximation

Input: $M \in \mathbb{R}^{m \times n}, \varepsilon$ tolérance
Output: Approximation à bas rang d de $M \approx UV$

$U = 0, V = 0, \mu = 0, r_0 =$ 0, j_1 index aléatoire de colonne
for $k = 1$ *to min(m,n)* **do**
  $u_k = R_{k-1[:,j_k]}$
  $i_k = arg(max_{u \in u_k}|u|)$
  $u_k \longleftarrow u_k / u_k[i_k]$
  $v_k^t = R_{k-1}[i_k, :]$
  $l_{k+1} = arg(max_{v \in v_k}|v|)$
  $\nu^2 = ||u_k||_2^2 ||v_k||_2^2$
  $\mu^2 \longleftarrow \mu^2 + \nu^2 + 2(u_k^t U)(V v_k)$
  $U \longleftarrow [U, u_k], V \longleftarrow [V, v_k^t], r_k = r_{k-1} + 1$
  Return if $\nu < \varepsilon\mu$.

2.1.2 Analyse du coût

On voit que l'on arrête la boucle quand l'on atteint la condition de tolérance. On retrouve une complexité donc en $\mathcal{O}(n^3)$ dans les cas usuels et $\mathcal{O}((nr)^2)$ quand l'algorithme converge en $\mathcal{O}(r)$ itérations [2]. Cette complexité est confirmé sur le graphique suivant qui provient du travail de Kezhong Zhao sur la ACA :

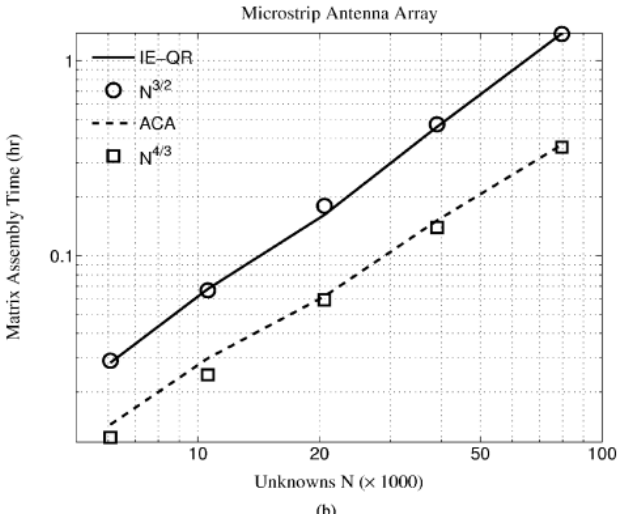

FIGURE 1. Comparisons of the complexity between ACA and single-level IE-QR algorithms for layered medium application (CPU time). All the computations were performed on a Pentium IV 1.8 GHz with 2 GB RAM PC [3]

Sur ce graphique on remarque que la ACA (Figure 1) suit une complexité de $\mathcal{O}(Nlog(N))$ à fréquence fixe. Ce résultat est meilleur que le pire cas théorique que l'on a présenté et peut donc convenir. Toutefois, on sait que le comportement asymptotique de la ACA n'est plus suffisant en complexité [1].

2.1.3 Erreurs

Ce qui fait aussi que de meilleurs algorithmes sont recherchés sont les erreurs lors de l'approximation par compression de la matrice M. Elle permet de gagner en espace de stockage au détriment de la précision. [3].

Sur ce graphique (Figure 2) qui représente la mémoire en fonction de la précision demandé, on voit bien que malgré la compression, le modèle de l'utilisation mémoire est exponentiel avec la monté en précision. L'algorithme est donc gourmand en erreur et fourni des erreurs de résolution. Il es donc intéressant de se tourner vers une autre solution d'algorithme de résolution rapide qui se base sur la ACA.

2.2 La MLACA

2.2.1 Présentation de l'algorithme

La MLACA est un autre algorithme algébrique qui sert à résoudre des équations comme des couplages magnétiques, ou des forces à l'intensité décroissante avec la distance [10]. L'objectif est d'approximer des parties du calcul selon la distance auxquels les éléments se trouvent les uns des autres.

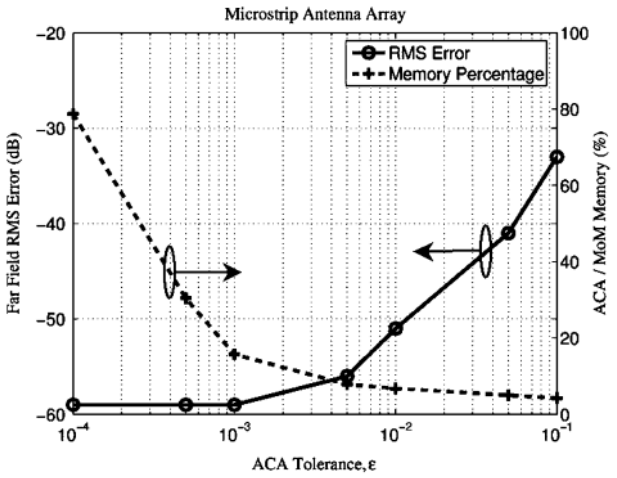

FIGURE 2. Far-field rms error and ratio between memory for ACA and memory for MoM as a function of ACA terminating tolerance ε. The study is performed with a 8×8 corporate-fed microstrip array [3]

On se figure une matrice L, l'ensemble des couples entre deux éléments (i, j), telle que :

$$L[i,j] = \frac{\mu}{4\pi a_i a_j} \int_{V_i} \int_{V_j} \frac{\hat{l}_i \hat{l}_j}{|r_i - r_j|} dV_i dV_j \qquad (3)$$

Ensuite, au lieu de calculer exactement la force qui s'applique, on définis un critère de champs lointain autour de notre observateur qui va servir de bordure à l'approximation des champs lointains. Comme on le voit représenté sur ce grraphique (Figure 3) tiré du travail de Ben A.P. Nel [11] :

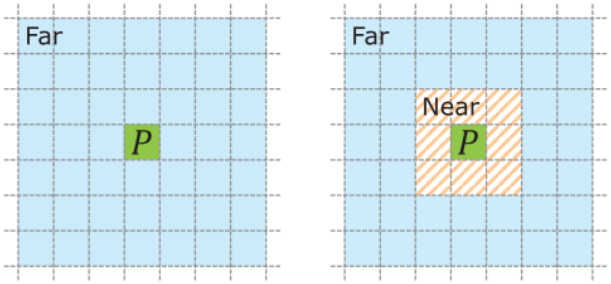

FIGURE 3. fig :Two near-interaction criteria, with respect to observer group P. Left : self interaction criterion. Right : nearest-neighbor criterion [10]

On comprend intuitivement que l'approximation d'un groupe d'éléments qui interagissent avec l'observateur comme un unique corps permet de diminuer la demande en calcul [10]. Cet algorithme par niveaux donne donc une compression de notre matrice M en divisant la fonction de base. On rappelle que cet algorithme est basé sur le fait que le degré de liberté reste asymptotiquement le même. Un niveau P, de la MLACA, divise les fonctions de base de la ACA ; un niveau se subdivise alors en 2^P [12] auxquels on applique l'algorithme du

papillon dans lequel les sous-groupes sont récursivement fusionnés et séparés avec une ACA appliquée à chaque niveau. Cet algorithme récursif permet de calculer des ensembles de matrices qui se subdivisent à chaque niveau. On voit bien sur la représentation (Figure 4) de la compression des matrices par multi niveaux que la compression de la matrice peut se calculer indépendemment en blocs eux même compressé et tiré du travail de Walton C. Gibson [12] :

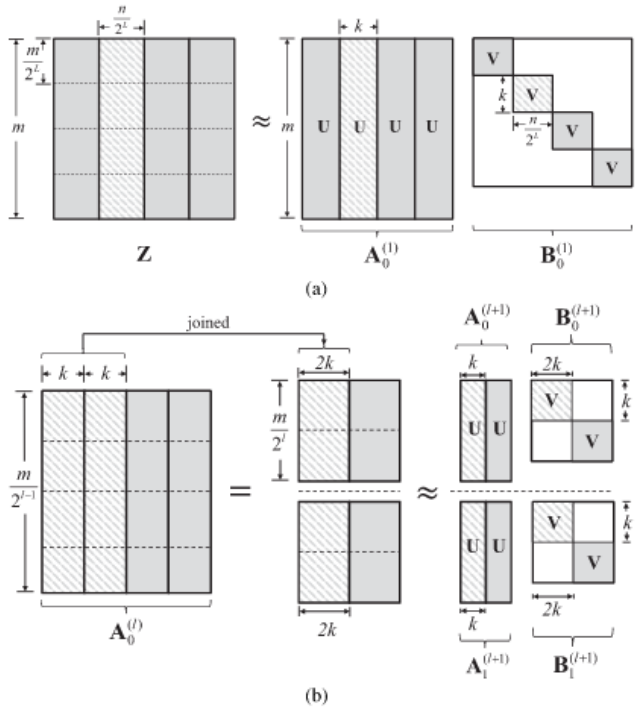

FIGURE 4. 2-Level U -type MLACA. (a) Operations on level l = 0. (b) Operations on level l + 1 [11]

C'est pourquoi cet algorithme est si facilement parallélisable. Par la MLACA on peut facilement approximer notre matrice par compression. Toutefois, il faut faire attention lors du découpage par bloc à ne pas calculer plusieurs fois les mêmes parties. De même, il y a un ordre dans lequel calculer ces sous-groupes de façon optimisée [1].

2.2.2 Analyse du coût

Tout d'abord, le coût de stockage de la MLACA se trouve presque intuitivement en sachant qu'à chaque niveau on divise par 2 notre matrice. On arrive rapidement à ue complexité :$\mathcal{O}(N+Nlog(N)) \approx \mathcal{O}(Nlog(N))$ [1].

D'autre part pour le coût de calcul, s'obtient en sachant que l'on effectue 2^P ACA (avec P le nombre de niveau de la MLACA) sur des matrices de rang k et de dimension $m \times n\,/\,(2^P)$. Il faut en plus à chaque étape effectuer une décomposition par ACA sur une matrice de même rang mais de dimension $(m/2^i) \times 2k$. L'ensemble de ces opérations donne un coût total de complexité $\mathcal{O}(N^2)$ [1].

La MLACA étant implémenté dans le projet GitHub ERCEM321_ATREYU [7], des tests ont été effectués pour voir son comportement asymptotique selon la taille de la matrice et la précision demandée.

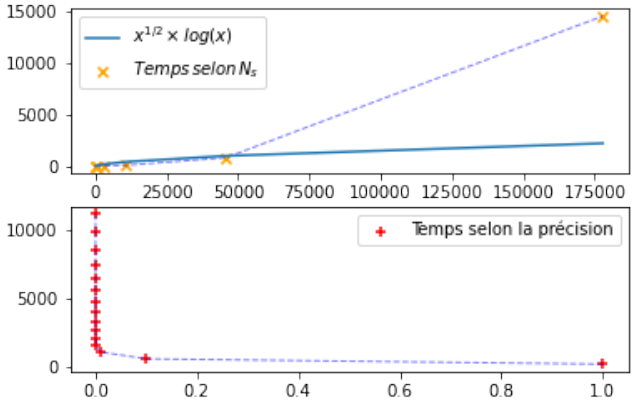

FIGURE 5. Benchmark effecuté depuis le projet GitHub [7]

Ce graphique (Figure 5) met en avant deux choses :

- L'implémentassions de la MLACA dans le projet GitHub possède une meilleure complexité que la MLACA classique car modèle qui lui correspond le plus est une complexité de $\mathcal{O}(N^{1/2} \times log(N))$ qui est loin de la complexité en $\mathcal{O}(N^2)$ dont on parlait précédemment. Mais malgré cette amélioration on observe une explosion du temps de calcul. Cette explosion s'explique par le temps que mettent les informations pour transiter de coeur à coeur [10]. Ce transfert de donnée fait perdre du temps à l'algorithme. De plus à cause de ce transit, tous les coeurs ne travaillent pas toujours, comme on le voit sur cette capture (Figure 6) lors de la compilation du code de MLACA.

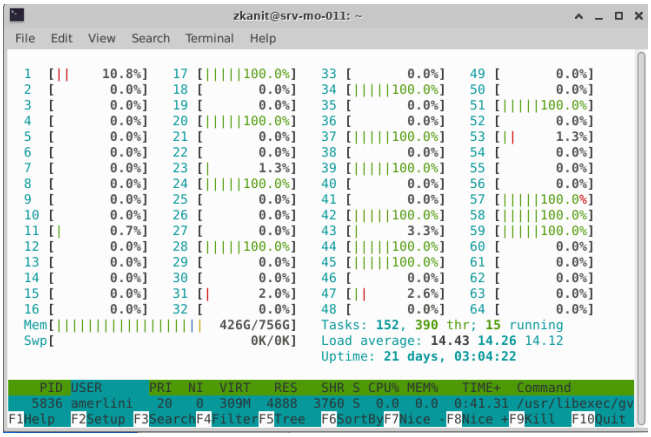

FIGURE 6. Activité des coeurs pendant la MLACA

- Le temps en fonction de la précision croit quant à lui de façon plus que exponentielle. La tolérance est donc un paramètre qui encore une fois décuple le temps de travail de l'algorithme quoi que l'on fasse. Malheureusement, ce temps de travail cumulé aux erreurs de compression comme on l'a vu sur la ACA nécessitent

la recherche de solution qui conviendrait mieux à la situation.

La MLACA est donc un algorithme très puissant, mais malheureusement, se sont les limites techniques du transfert de données qui l'empêchent de déployer tout son potentiel. Cherchons si l'approximation par bloc peut résoudre ces problème.

3. Approximation par blocs (BACA)

3.1 Algorithme

L'algorithme de BACA dérive directement de la ACA. Au lieu, de choisir une unique colonne et ligne pour compresser la matrice, on choisit maintenant un groupe fini de lignes et colonnes pour obtenir la relation (4) [8] :

$$M \approx UV = \sum_{k=1}^{n_j} U_k V_k \tag{4}$$

Avec $U_k \in \mathbb{R}^{m \times d_k}$ et $V_k \in \mathbb{R}^{d_k \times n}$.

Normalement, l'algorithme choisit des valeurs de pivot de façon aléatoire ou optimisée. Mais pour performer la BACA, les pivot (i_k, j_k) sont choisis grâce à une décomposition QR préalable[8] sur les transposées des $d_k < d$ colonnes. Nous allons présenter les 3 étapes de chaque étape de la BACA avant d'en proposer un algorithme :

1. Trouver le bloc de lignes I_k et le bloc de colonnes J_k par décomposition QR [9] en commencant par J_1 de façon aléatoire, puis en appliquant l'algorithme :

$$(Q_k^c, T_k^c, I_k) = \mathbf{QR}(R_{k-1}[:, J_k]^t, d), I_k \tag{5}$$

$$(Q_{k+1}^c, T_{k+1}^c, J_{k+1}) = \mathbf{QR}(R_{k-1}[I_k, :], d), J_k \tag{6}$$

L'algorithme choisit les blocs à partir des sous matrices R_k R_{k-1}. Pour se faire, l'implémentation de la décomposition **QR** sur Rosetta Code a été utilisé [9] et qui choisit le pivot en recherchant des le coefficient maximal dans la colone de la décompositon **QR**. A noter que pour avoir un résultat optimal il est préférable de prendre d lignes et colonnes [8]. De plus, il faut éviter d'utiliser sur plusieurs itérations les mêmes lignes ; c'est pourquoi la décomposition **QR** est faite sur une sous-matrice de $R_{k-1}[:, J_k]^t$ plutôt que directement sur $R_{k-1}[:, J_k]^t$. Bien sûr on fait de même avec $R_{k-1}[I_k, :]$. Pour comprendre cette étape, il est possible d'utiliser le schémas (Figure 7)
2. Ensuite, on forme les facteurs de bas rang produit de $U_k V_k$. Par exemple avec $C_k = R_{k-1}[:, J_k]$, $\mathbf{R} = R_{k-1}[I_k, :]$ et $W_k = R_{k-1}[I_k, J_k]$. On trouve la relation $R_{k-1} \approx C_k W_k^T \mathbb{R}_k = U_k V_k$. A noter que la matrice W_k est souvent en déficit de rang. Dans ce cas, la BACA devient beaucoup plus puissante que la ACA avec d_k pivots et peut être utilisée pour générer d colonnes J_{k+1} pour l'itération suivante.

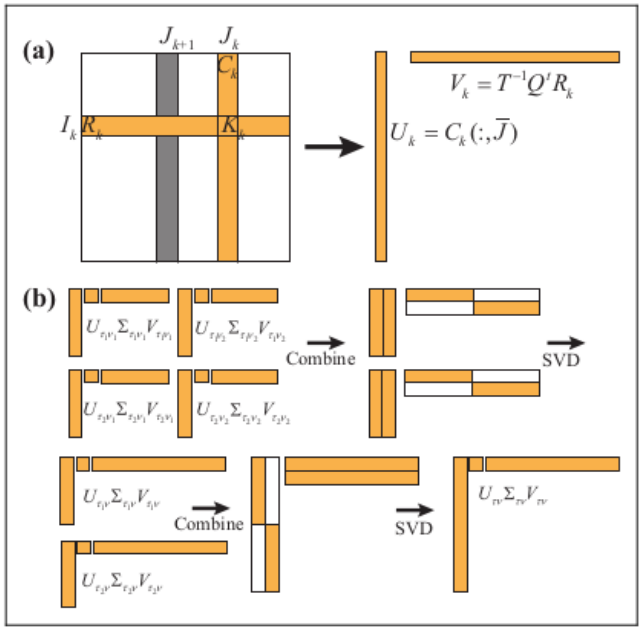

FIGURE 7. (a) Selection of $I_k/J_k and form the low-rank update U_k V_k$. (b) *Low-rank merge operation*

On met alors à jour les pivots (I_k, J_k) dans la relation (9) par le pivot $\bar{J}$ obtenu par décompoosition $\mathbb{QR}$.

$$(Q, T, \bar{J}) = \mathbf{QR}(W_k = R_k, \varepsilon) \tag{7}$$

$$U_k = C_k[:, \bar{J}], V_k = T^{-1} Q^t \mathbf{R_k} \tag{8}$$

$$I_k \longleftarrow I_k[1 : d_k], J_k \longleftarrow J_k[\bar{J}] \tag{9}$$

3. Ensuite il faut calculer $\nu = ||U_k V_k||_F$ et mettre à jour $\mu = ||UV||_F$ en supposant que la mise à jour de la valeur du bas rang de dimension $d \times d$ soit de complexité $\mathcal{O}(nd_k^2)$ par l'opération :

$$T_{U_k} = Chol(U_k^t U_k), T_{V_k} = Chol(V_k V_k^t) \tag{10}$$

$$\nu = ||T_{U_k} T_{V_k}^t||_F \tag{11}$$

Puis la mise à jour de la valeur de la norme de UV en $\mathcal{O}(nr_k d_k)$.

$$\mu^2 \longleftarrow \mu^2 + \nu^2 + 2 \sum_{i=1}^{r_{k-1}} \sum_{j=1}^{d_k} \tilde{V}[i, j] \tag{12}$$

On obtient donc le vecteur :

$$\tilde{V} = (V V_k^t) \circ (U^t U_k) \tag{13}$$

On peut noter qu'une valeur de d plus grande permet de donner une meilleure approximation à notre compression [8]. Pour le pseudcode de l'algorithme.

Le fait que BACA prennent des blocs entiers permet une convergence plus rapide et peut aussi être amélioré par le choix de bons critères d'arrêts [8]. Une fois encore la compression

Algorithm 2: Blocked Adaptive Cross Approximation

Input: $M \in \mathbb{R}^{m\times n}, d \in \mathbb{N}, \varepsilon$ tolérance
Output: Approximation à bas rang r de $M \approx UV$

$U = 0, V = 0, \mu = 0, r_0 =$ $0, J_1$ liste de d index aléatoire
for $k = 1$ *to min(m,n)* **do**
  $C_k = R_{k-1}[:, J_k]$ $(Q_k^c, T_k^c, I_k) = \mathbf{QR}(C_k^t, d), I_k$
  $\mathbf{R}_k = R_{k-1}[I_k, :]$
  $(Q_{k+1}^r, T_{k+1^r}, J_{k+1}) = \mathbf{QR}(R_k, d), J_{k+1}$
  $W_k = R_{k-1}[I_k, J_k]$
  $(U_k, V_k, d_k, \overline{J}) = LRID(C_k, W_k, R_k, \varepsilon)$
  $I_k \longleftarrow I_k[1 : d_k], J_k \longleftarrow J_k[\overline{J}]$ $r_k = r_{k-1} + d_k$
  $\nu = LRnorm(U_k, V_k)$
  $\mu \longleftarrow LRnormUp(U, V, \mu, U_k, V_k, \nu)$
  $U \longleftarrow (U, U_k), V \longleftarrow (V, V_k)$ Return if $\nu < \varepsilon\mu$
Def *LDIR(C,W,R,ε)* **:**
  $(Q, T, \overline{J}, r) = \mathbf{QR}(W, \varepsilon)$;
  $U = C[:, \overline{J}]$;
  $V = T^{-1}Q^t R$;
  Return $U, V, r, \overline{J}$
Def *LRnorm(U,V)* **:**
  $T_1 = Chol(U^t U)$;
  $T_2 = Chol(VV^t)$;
  Return $||T_1 T_2^t||_F$
Def *LRnormUp(U,V,ν,Ū,V̄,ν̄)* **:**
  $s = \nu^2 + \overline{\nu}^2 + 2\sum_{i=1}^{r_{k-1}} \sum_{j=1}^{d_k} \tilde{V}[i, j]$;
  $\tilde{V} = (VV_k^t) \circ (U^t U_k)$;
  Return $\sqrt{s}$

se fait sans a-priori sur le coeur, on peut donc appliquer cette méthode sur à peu près à toutes les fonctions de Green. Ce compactage en bloc permet à ce que certaines opérations deviennent "BLAS-3" par ce processus ; c'est à dire que les types ainsi que leur opérations élémentaires sur la carte mère sont adaptées à l'utilisation sur un seul processeur avec plusieurs coeurs. Cette situation convient donc parfaitement à l'usage que l'on veut faire de cet algorithme.

Le choix d'un d judicieux affectera la balance entre efficacité et véracité.

3.2 Parallélisation

Cet algorithme du fait de sa computation par blocs peut facilement se répartir sur différents coeurs afin de paralléliser le calcul [8]. Il faut alors hiérarchiser le grillage de surface, mais cette opération se fait sans grande difficulté avec les outils présents dans le projet GitHub [7]. Ensuite, pour mettre en oeuvre la parallélisation, il suffit de faire faire le BACA à un processeur pendant que deux autres gèrent deux calculs de sous-matrices avant de fusionner ces matrices de bas rang.

3.3 Analyses du coût

La nature à travailler par bloc de la BACA en fait un algorithme de choix en terme de complexité. Cela permet d'utiliser de moins nombreux pivots qui montre une convergence continue et presque assurée. De plus l'opération par bloc bénéficie de la haute performance des flip-flop en "BLAS-3". On peut donc penser que la complexité originellement en $\mathcal{O}(Nr^2)$ est peut être meilleure. Il faudrait prendre en compte le calcul des sous matrices qui aurait une complexité en $\mathcal{O}(\sqrt{n_b})$, mais à moins d'avoir un n_b qui dépende de la taille de la matrice, la complexité du BACA est la même que celle de la ACA puisqu'elle est asymptotiquement $\mathcal{O}(nr^2\sqrt{n_b})$; du moins en théorie [8].

En pratique, la "BLAS-3" permettrait une hausse des performances considérable [8].

Conclusion

Nous n'avons malheureusement pas pu voir avec notre propre algorithme avec des résultats. Cependant, nous pouvons voir que le développement de la BACA apporte une solution algébrique efficace aux problèmes de résolution d'équation. La BACA tiend sa force au fait d'utiliser le principe de la ACA sur des blocs, ce qui la rend d'autant plus robuste et efficace. Malheureusement, aggrandir la taille des blocs peut se trouver être un risque jusqu'à un certain point qu'il faudrait déterminer expérimentalement.

Il existe encore des façons d'améliorer la BACA ; la parallélisation requiert de faire attention à la distribution des données, car puisque BACA précède la MLACA, la réflexion sur le coût de communication est une réflexion à avoir en tête pour de futurs améliorations.

Remerciements

Je tiens à remercier M.Merlini qui a pris de son temps pour encadrer et expliquer ce projet ; les concepts théoriques et mathématiques qui soutiennent ces algorithmes. Ainsi que Clément Henry qui a aussi pris de son temps de travail pour m'expliquer le fonctionnement du projet et la connection au serveur.

Je remercie Benjamin Somers de m'avoir fourni un serveur puissant lorsque l'autre n'était pas accessible.

Références

[1] José M. Tamayo, Alexander Heldring, and Juan M. Rius. Multilevel Adaptive Cross Approximation (MLACA). *IEEE Transactions on Antennas and Propagation*, 59(12) :4600–4608, December 2011.

[2] Mario Bebendorf. *Hierarchical matrices : a means to efficiently solve elliptic boundary value problems ; with 53 tables*. Number 63 in Lecture notes in computational science and engineering. Springer, Berlin Heidelberg, 2008.

[3] K. Zhao, M.N. Vouvakis, and J.-F. Lee. The Adaptive Cross Approximation Algorithm for Accelerated Method of Moments Computations of EMC Problems. *IEEE Transactions on Electromagnetic Compatibility*, 47(4) :763–773, November 2005.

[4] Akihiro Ida, Takeshi Iwashita, Takeshi Mifune, and Yasuhito Takahashi. Parallel Hierarchical Matrices with Adaptive Cross Approximation on Symmetric Multiprocessing Clusters. *Journal of Information Processing*, 22(4) :642–650, 2014.

[5] Nebra Mathieu, Schaller Matthieu, and Gonnage Ranga. Apprenez à programmer en C++. `https://openclassrooms.com/fr/courses/1894236-apprenez-a-programmer-en-c`, 02 2022.

[6] Nebra Mathieu and Gonnage Ranga. Programmez en orientée objet avec C++. `https://openclassrooms.com/fr/courses/7137751-programmez-en-oriente-objet-avec-c`, 02 2022.

[7] Francesco P. Andriulli, Alexandre Dély, and Adrien Merlini. Ercem321_atreyu. `https://github.com/CERL-PoliTo/ERCEM321_Atreyu`, 2020.

[8] Yang Liu, Wissam Sid-Lakhdar, Elizaveta Rebrova, Pieter Ghysels, and Xiaoye Sherry Li. A parallel hierarchical blocked adaptive cross approximation algorithm. *The International Journal of High Performance Computing Applications*, 34(4) :394–408, July 2020.

[9] Mol Michael. QR decomposition - Rosetta Code. `https://rosettacode.org/wiki/QR_decomposition`, 03 2022.

[10] Ben A. P. Nel and Matthys M. Botha. An Efficient MLACA-SVD Solver for Superconducting Integrated Circuit Analysis. *IEEE Transactions on Applied Superconductivity*, 29(7) :1–10, October 2019.

[11] Ben A. P. Nel and Matthys M. Botha. MLACA With Modified Grouping Strategy for Efficient Superconducting Circuit Analysis. *IEEE Transactions on Applied Superconductivity*, 29(5) :1–5, August 2019.

[12] Walton C. Gibson. Efficient Solution of Electromagnetic Scattering Problems Using Multilevel Adaptive Cross Approximation and LU Factorization. *IEEE Transactions on Antennas and Propagation*, 68(5) :3815–3823, May 2020.