Vers une factorisation symbolique hiérarchique de rang faible pour des matrices creuses

Emmanuel Agullo, Aurélien Falco, Luc Giraud, Guillaume Sylvand

Compas'2017 : Parallélisme / Architecture / Système
Sophia Antipolis, France, du 27 au 30 juin 2017

Vers une factorisation symbolique hiérarchique de rang faible pour des matrices creuses

* Emmanuel AGULLO, * Aurélien FALCO, * Luc GIRAUD, † Guillaume SYLVAND

* Inria Bordeaux Sud Ouest, 200 avenue de la Vieille Tour, 33400 TALENCE - France
† Airbus Group Innovations (AGI)

Résumé
Les algorithmes hiérarchiques basés sur des techniques de compression de rang faible ont révolutionné les méthodes de résolution de systèmes linéaires denses à l'aube du XXIème siècle en réduisant considérablement les coûts de calcul. Toutefois, leur application au traitement de systèmes linéaires creux (comportant de nombreux éléments nuls), qui sont le type de problèmes apparaissant le plus souvent au cœur des simulations numériques, reste aujourd'hui un challenge majeur auquel s'attellent à la fois la communauté des matrices hiérarchiques et celle des matrices creuses. À cet effet, une première classe d'approche a été développée par la communauté des matrices hiérarchiques pour exploiter la structure creuse des matrices. Si le point fort de ces méthodes est que l'algorithme résultant reste hiérarchique, celles-ci ne parviennent pas à exploiter certains zéros comme le font naturellement les méthodes classiques de factorisation de systèmes linéaires creux. À l'opposé, du fait qu'une factorisation creuse peut être vue comme une séquence de plus petites opérations denses, la communauté des matrices creuses a exploré cette propriété pour introduire des techniques hiérarchiques au sein de ces opérations élémentaires. Cependant, l'algorithme résultant perd la propriété fondamentale des algorithmes hiérarchiques, dans la mesure où la hiérarchie de compression est seulement locale. Dans cet article, nous introduisons un nouvel algorithme, effectuant une factorisation symbolique creuse hiérarchique qui permet d'exploiter efficacement l'ensemble des zéros de la matrice creuse et de ses facteurs tout en préservant une structure hiérarchique globale. Nous montrons expérimentalement que cette nouvelle approche permet d'obtenir à la fois un nombre réduit d'opérations (du fait de son caractère hiérarchique) et un nombre d'éléments non nuls aussi réduit qu'une méthode creuse (grâce au recours à une factorisation symbolique).

Mots-clés : $\mathcal{H}$-Matrice, factorisation symbolique, dissection emboîtée, matrice creuse, algèbre linéaire, compression.

1. Contexte

1.1. Matrices hiérarchiques

Les matrices hiérarchiques ($\mathcal{H}$-Matrices), introduites en 1999 par Hackbusch [11], ont permis de réduire la complexité d'une factorisation LU (décomposition d'une matrice A en des matrices triangulaires inférieure L et supérieure U) d'une matrice dense d'ordre N de $\mathcal{O}(N^3)$ à $\mathcal{O}(N \log^*(N))$ opérations à virgules flottante (flop) pour certaines classes de problèmes. Si une description exhaustive de l'algorithmique des $\mathcal{H}$-Matrices et de leur cadre d'application est

hors du propos du présent article, nous en présentons ici les ingrédients algorithmiques principaux et nous renvoyons à [11, 3, 12] pour plus de détails.
La première étape de construction d'une $\mathcal{H}$-Matrice consiste à trier récursivement les inconnues (lignes / colonnes de la matrice). Ce tri récursif peut être représenté par un arbre, dit « arbre de groupe », comme représenté en correspondance des lignes (gauche) et des colonnes (dessus) de Fig. 1. Le squelette de la $\mathcal{H}$-Matrice correspond simplement à la mise en relation de cet arbre avec lui-même comme représenté sur Fig. 1. Par exemple le bloc bleu (en haut à droite) correspond à la mise en relation des groupes bleus au sein des arbres ligne (gauche) et colonne (dessus). Lors de la construction de la $\mathcal{H}$-Matrice, pour chaque interaction de nœuds de l'arbre, on peut choisir de compresser le bloc d'interactions (blocs verts), de stocker ce bloc tel quel de manière dense (blocs rouges), ou bien de récurser (descendre d'un niveau au sein de l'arbre de groupe, à la fois sur les colonnes et les lignes). Différents algorithmes permettent de trier efficacement les inconnus et de déterminer quels blocs pourront être compressés et nous renvoyons le lecteur curieux à [11, 3, 12]. Le tri le plus communément utilisé donne lieu à des arbres binaires (cf. Fig. 4a) et la (classe de) méthode correspondante est appelée bissection (que nous noterons « Bisection » dans le reste de l'article, suivant la terminologie anglophone).

1.2. Matrices creuses et méthodes directes

Une matrice creuse A est une matrice comportant un nombre suffisamment élevé de zéros pour qu'une gestion particulière de la structure irrégulière qui en résulte permette d'améliorer la consommation mémoire et/ou le temps de calcul. La factorisation d'une telle matrice A induit un phénomène de « remplissage » qui fait que le nombre de non zéros dans les facteurs L et U est beaucoup plus élevé que le nombre de non zéros de la matrice originale A [4]. Le remplissage pouvant être prohibitif dans certaines situations, d'autres classes d'algorithmes telles que les méthodes itératives [20] peuvent être employées pour l'éviter, et la factorisation LU d'une matrice creuse est appelée, par opposition, une « méthode directe » [4, 17].
Le nombre de zéros des facteurs dépend de l'ordre d'élimination des variables. La minimisation de la taille des facteurs (ainsi que du nombre d'opérations induit) étant NP-complet, des heuristiques sont employées. Elles opèrent sur le graphe associé à la matrice. Les nœuds de ce graphe correspondent aux inconnues (lignes / colonnes de la matrice) et les arêtes aux valeurs non nulles de la matrice. Parmi l'ensemble des méthodes employées classiquement (nous renvoyons à nouveau à [4]), l'une d'elle consiste à déterminer un ensemble de nœuds séparant le graphe en deux parties indépendantes. Cet ensemble de nœuds est appelé un « séparateur ». Par exemple, si on représente (grossièrement) le graphe d'une matrice associé à une discrétisation par une méthode de différence finie ou d'élément fini d'un maillage 2D par un carré comme Fig. 3b, une coupe médiane verticale (représentée par le rectangle rouge S_0) sépare le graphe en deux sous parties intérieures I_1 (gauche) et I_2 (droite). L'application récursive de cet algorithme s'appelle une dissection emboîtée [5] (notée ND dans la suite pour « Nested Dissection »). En numérotant à chaque étape les intérieurs avant les séparateurs, alors on assure que les intérieurs n'interagissent pas, formant ainsi de gros blocs de zéros ($A_{I_1 I_2} = 0$) représentés en blanc dans Fig. 2 comme dans l'ensemble des représentations matricielles du présent article. Suite à cette numérotation, une factorisation symbolique est appliquée pour déterminer, non seulement ces gros blocs de zéros entre intérieurs, mais également les (plus petits) blocs de zéros entre intérieurs et séparateurs (comme le montre la structure du bloc A_{S_0,I_1} dans Fig. 2).

1.3. Algorithmes hiérarchiques de compression de rang faible étendus aux matrices creuses

1.3.1. Bissection (Bisection)

Une façon d'étendre immédiatement les algorithmes hiérarchiques de compression de rang faible aux matrices creuses consiste à appliquer la méthode Bisection (voir section 1.1) telle quelle. Si on considère à nouveau le graphe associé à une matrice issu d'une discrétisation 2D, alors l'application de l'algorithme Bisection sur un tel graphe peut être représenté par Fig. 3a. L'ensemble des inconnues (I_0) est ici scindé en deux parties I_1 et I_2, sans séparateur. Ainsi, A_{I_1,I_2} n'est (en général) pas nul et le taux de remplissage conséquent est très élevé (Fig. 5a).

1.3.2. Dissection emboîtée (ND : Nested Dissection)

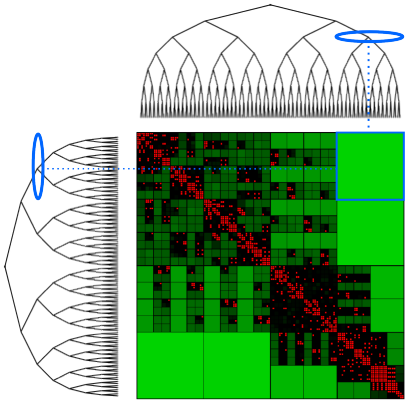


FIGURE 1 – Interaction entre deux arbres de groupes faisant apparaître une $\mathcal{H}$-Matrice dense.

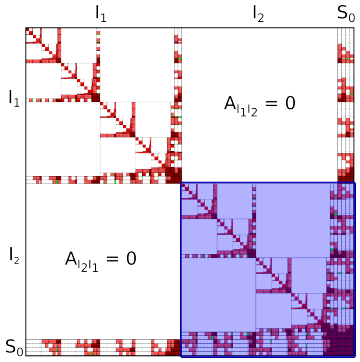
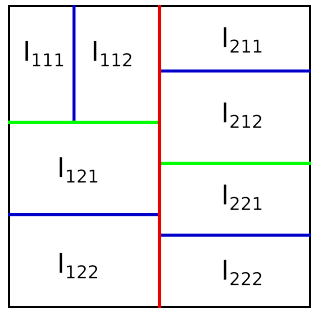


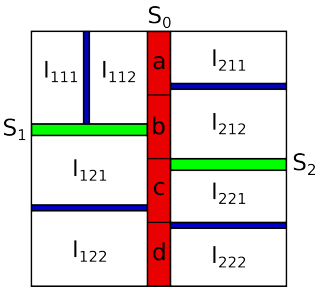
FIGURE 2 – Dissection emboîtée (ND) appliquée à une matrice creuse. Nous nous focaliserons sur la partie bleue dans la suite de ce document.

Alternativement, la dissection emboîtée peut être appliquée (Fig. 3b) pour former un arbre de groupe ternaire (Fig. 4b) et obtenir une $\mathcal{H}$-Matrice exhibant de gros blocs de zéros entre les intérieurs séparés à chaque niveau (Fig. 2 et zoom Fig. 4b). La communauté $\mathcal{H}$-Matrice a utilisé cette approche sous la dénomination DD (Domain Decomposition) ou ND (Nested Dissection). Nous l'appellerons ND dans le reste de l'article conformément à la section 1.2. Nous nous appuierons plus particulièrement sur l'étude de Hackbusch de 2015 [13, § 5.8] qui recense les divers travaux effectués dans ce sens [16, 14, 8, 9, 15]. Pour conserver un caractère hiérarchique, ces méthodes appliquent ensuite Bisection sur le séparateur (« Matrix blocks with interface indices are non-zero and are handled as standard $\mathcal{H}$-Matrices, i.e., based on standard geometrical or algebraic clustering algorithms. » [15, § 3.3]). Par exemple, S_0 est scindé en $\{a, b\}$ et $\{c, d\}$, puis $\{a, b\}$ en $\{a\}$ et $\{b\}$ comme on peut le voir sur le graphe associé à la matrice (Fig. 3b) ainsi que sur l'arbre de groupe (Fig. 4b).

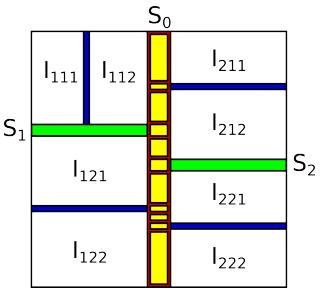
La limite de cette approche est qu'un découpage physique un peu irrégulier peut engendrer un chevauchement entre des blocs du séparateur et des blocs externes au séparateur. On cherche à minimiser le nombre de groupes extérieurs qui interagissent avec chacun des blocs du séparateur (par exemple sur la Fig. 3b, b n'interagit pas avec I_{211}). Mais ici on peut voir que le bloc a, issu de la bissection du séparateur, ne colle pas tout à fait à ses voisins I_{211} et I_{112} : il est aussi connexe au bloc I_{212} alors qu'ils n'ont au final que peu d'interactions. Cela favorise le stockage de zéros non nécessaires.



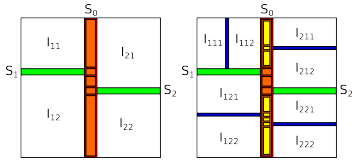
(a) Bisection.



(b) ND.

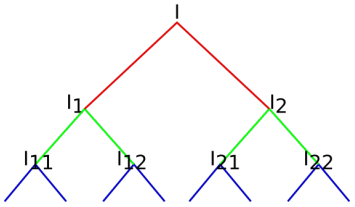


(c) FSF.

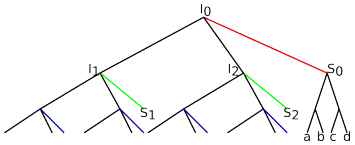


(d) HSF.

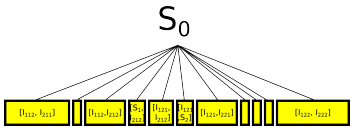
FIGURE 3 – Découpage du graphe associé à la matrice.



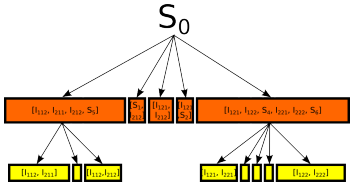
(a) Bisection.



(b) ND.

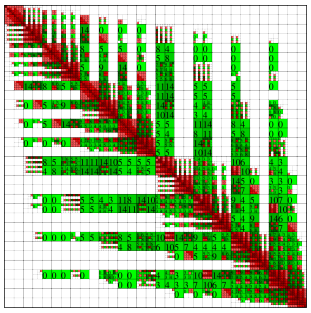


(c) FSF (zoom sur S_0).

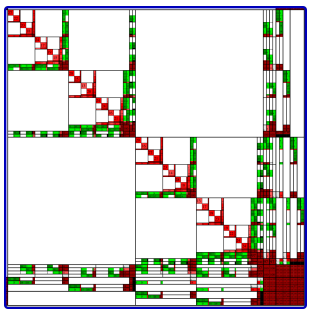


(d) HSF (zoom sur S_0).

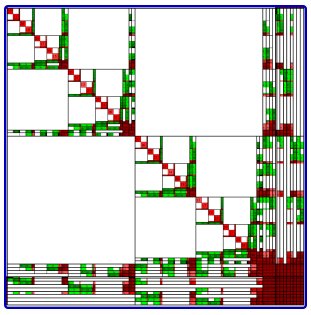
FIGURE 4 – Arbre de groupes.



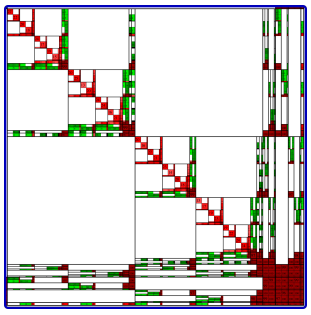
(a) Bisection.



(b) ND.



(c) FSF.



(d) HSF.

FIGURE 5 – H-matrices (après factorisation).

1.4. Méthodes directes creuses étendues par des techniques de compression hiérarchique de rang faible

De même que les efforts menés par la communauté hiérarchique pour adapter les $\mathcal{H}$-Matrices aux problèmes creux, des travaux sont en cours au sein de la communauté des solveurs creux afin d'y intégrer des techniques de compression et une notion de hiérarchie. Ainsi, des $\mathcal{H}$-Matrices (variante HSS) ont été intégrées au sein de méthodes multifrontales [21] et ont récemment donné lieu au logiciel STRUMPACK [6]. De même, les équipes MUMPS [1, 2] et PaStiX [19] ont intégré des variantes « Block Low Rank » (BLR).

1.4.1. Factorisation symbolique à plat (FSF : Flat Symbolic Factorization)

La finesse qu'arrivent à capturer naturellement ces travaux issus de la communauté des solveurs creux (et que ne capturent pas les travaux [8, 9, 15] issus de la communauté $\mathcal{H}$-Matrice) est l'exploitation de la structure non zéro dans les couplages intérieurs-séparateurs. N'étant pas contraints par la recherche d'une hiérarchie dans le traitement des séparateurs, ils s'appuient sur une factorisation symbolique.

Afin de limiter au maximum les chevauchements expliqués auparavant (le groupe α connecté à I_{212} sur la Fig. 3b), le séparateur est découpé selon les interactions de ses inconnues : les inconnues ayant les mêmes interactions (Fig. 3c) sont rassemblées dans un unique enfant de l'arbre de groupes (Fig. 4c). Étant donné que le séparateur n'est pas découpé hiérarchiquement (ses enfants seront déjà les parties les plus fines possibles), nous appelons ici, cette factorisation symbolique, classique dans un contexte creux : « factorisation symbolique à plat » (FSF : Flat Symbolic Factorization). Elle s'apparente à ce qu'effectue un solveur creux utilisant des techniques de compression par blocs.

Fig. 5c montre une partie de la $\mathcal{H}$-Matrice lorsque traitée avec la méthode FSF. Le problème de cette méthode est que le séparateur est découpé en un trop grand nombre de parties différentes, elles-mêmes trop petites, et toutes au même niveau. La notion de hiérarchie est perdue.

2. Factorisation symbolique hiérarchique (HSF : Hierarchical Symbolic Factorization)

Pour bénéficier des propriétés hiérarchiques de ND tout en n'introduisant pas de zéros supplémentaires, nous avons introduit un nouvel algorithme : une factorisation symbolique hiérarchique (HSF : Hierarchical Symbolic Factorization). Pour cela, nous découpons le séparateur selon les interactions de ses inconnues avec les nœuds d'un même niveau de la dissection emboîtée (Fig. 3d, Fig. 4d). Nous pouvons voir sur ces figures les deux premières étapes de cet algorithme, qui atteint alors dans cet exemple le même niveau de granularité (les blocs jaunes) que pour la FSF.

Cette méthode permet de bénéficier à la fois de la hiérarchie d'une $\mathcal{H}$-Matrice au sein du séparateur *et* d'avoir un découpage cohérent basé sur les interactions des inconnues, contrairement à FSF qui se contentait d'avoir un bon découpage mais perdait toute hiérarchie dans le séparateur. De plus, de nombreux zéros sont exhibés très rapidement dans la hiérarchie et nous pouvons donc éviter de nombreux niveaux inutiles.

3. Étude expérimentale préliminaire

Les expériences ont été menées pour l'instant en séquentiel sur des cubes maillés comportant de quelques dizaines de milliers à un million d'inconnues. Chaque inconnue interagit avec ses 6 voisins latéraux. L'ensemble des algorithmes présentés (Bisection, ND, FSF, HSF) ont été implémentés dans le code H-mat d'Airbus (qui ne permettait avant cette étude que de traiter des

matrices denses [18]). Quelques précisions sont de mise : ND détermine (par une factorisation symbolique post-découpage, ne remettant donc pas en question le découpage de ND) quels blocs seront remplis lors de la décomposition LU. Cette méthode peut être rapprochée de la structure creuse développée par Ibragimov et al. dans [14]. Notons aussi qu'avec un découpage physique parfait, où le découpage récursif des deux sous-arbres serait symétrique (possible lors de la découpe d'un cube, mais en réalité très difficile à obtenir sur un cas industriel), ND pourrait être équivalent à HSF.
Les résultats confirment que découper le séparateur selon une factorisation symbolique permet de réduire le nombre de valeurs numériques stockées (en réduisant le nombre de zéros stockés) *et* le nombre d'opérations effectuées. Fig. 6 nous permet de constater que HSF stocke moins de valeurs numériques que FSF et ND (FSF étant pénalisé par une agrégation plus approximative introduisant des zéros, car non hiérarchique). En terme d'opérations à virgule flottante (flop), les méthodes utilisant une factorisation symbolique sont aussi plus intéressantes (Fig. 7), HSF prévalant ici aussi sur FSF.

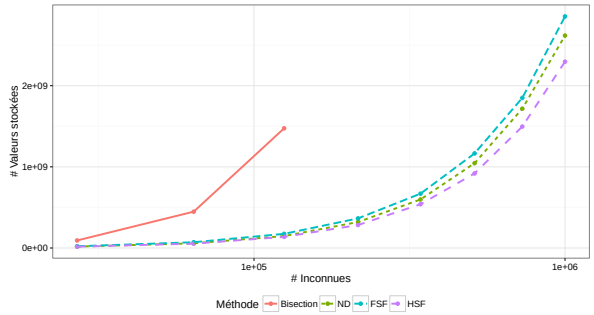


FIGURE 6 – Nombre de valeurs numériques stockées (après décomposition LU).

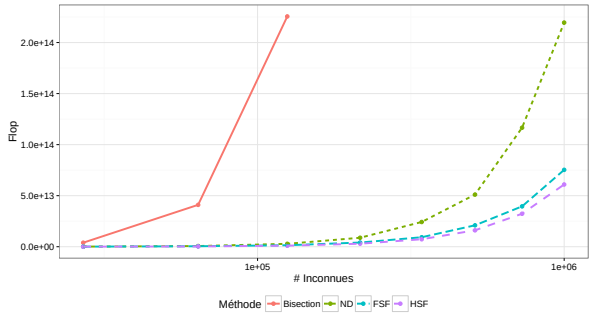


FIGURE 7 – Nombre d'opérations à virgule flottante (flop) nécessaires pour effectuer une décomposition LU.

4. Conclusion & perspectives

Nous avons pu confirmer les résultats précédents [8, 9] concernant le gain apporté par la dissection emboîtée au sein des $\mathcal{H}$-Matrices, ainsi que montrer qu'il était possible de réduire encore la mémoire et le nombre d'opérations grâce à une factorisation symbolique hiérarchique. Pour plus de précisions sur le gain dû à la dissection emboîtée au sein des $\mathcal{H}$-Matrices, avec différentes tolérances de précision pour la compression, nous renvoyons le lecteur vers [8, 10]. Une comparaison de celles-ci avec divers solveurs creux est faite en [7].
Pour aller plus loin, nous envisageons de permettre aux $\mathcal{H}$-Matrices de choisir un grain fin sur les blocs extra-diagonaux, et plus gros sur les supernœuds, chose qui n'est pas encore possible à l'heure actuelle. Ce travail est en cours de développement car il permettrait à la fois une bonne gestion des parties denses de la matrice et un stockage optimal pour les parties creuses. Nous avons implémenté l'ensemble des algorithmes présentés dans cet article au-dessus de StarPU (conformément à [18]) et travaillons actuellement sur une étude de leur performance en parallèle.

Bibliographie

1. Amestoy (P.), Ashcraft (C.), Boiteau (O.), Buttari (A.), L'Excellent (J.-Y.) et Weisbecker (C.). – Improving multifrontal methods by means of block low-rank representations. *SIAM Journal on Scientific Computing*, vol. 37, n3, 2015, pp. A1451–A1474.
2. Amestoy (P.), Buttari (A.), L'Excellent (J.-Y.) et Mary (T.). – *On the Complexity of the Block Low-Rank Multifrontal Factorization*. – Rapport technique, IRIT, May 2016.
3. Bebendorf (M.). – *Hierarchical matrices*. – Springer, 2008.
4. Davis (T. A.). – *Direct methods for sparse linear systems*. – Siam, 2006volume 2.
5. George (A.). – Nested dissection of a regular finite element mesh. *SIAM Journal on Numerical Analysis*, vol. 10, n2, 1973, pp. 345–363.
6. Ghysels (P.), Li (X. S.), Rouet (F.-H.), Williams (S.) et Napov (A.). – An efficient multicore implementation of a novel HSS-structured multifrontal solver using randomized sampling. *SIAM Journal on Scientific Computing*, vol. 38, n5, 2016, pp. S358–S384.
7. Grasedyck (L.), Hackbusch (W.) et Kriemann (R.). – Performance of h-lu preconditioning for sparse matrices. *Computational Methods in Applied Mathematics Comput. Methods Appl. Math.*, vol. 8, n4, 2008, pp. 336–349.
8. Grasedyck (L.), Kriemann (R.) et Le Borne (S.). – Parallel black box $\mathcal{H}$-LU preconditioning for elliptic boundary value problems. *Computing and Visualization in Science*, vol. 11, 2008, pp. 273–291. – 10.1007/s00791-008-0098-9.
9. Grasedyck (L.), Kriemann (R.) et Le Borne (S.). – Domain decomposition based $\mathcal{H}$-LU preconditioning. *Numerische Mathematik*, vol. 112, 2009, p. 565–600.
10. Grasedyck (L.), Kriemann (R.) et Le Borne (S.). – Domain decomposition based $\mathcal{H}$-LU preconditioning. *Numerische Mathematik*, vol. 112, 2009, p. 565–600.
11. Hackbusch (W.). – A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part I : Introduction to $\mathcal{H}$-matrices. *Computing*, vol. 62, 1999, pp. 89–108. – 10.1007/s006070050015.
12. Hackbusch (W.). – *Hierarchical matrices : Algorithms and analysis*. – Springer, 2015volume 49.
13. Hackbusch (W.). – Survey on the technique of hierarchical matrices. *Vietnam Journal of Mathematics*, vol. 44, n1, 2016, pp. 71–101.
14. Ibragimov (I.), Rjasanow (S.) et Straube (K.). – Hierarchical cholesky decomposition of sparse matrices arising from curl–curl-equation. *Journal of Numerical Mathematics jnma*, vol. 15, n1, 2007, pp. 31–57.
15. Kriemann (R.). – $\mathcal{H}$-LU factorization on many-core systems. *Computing and Visualization in Science*, vol. 16, n3, 2013, pp. 105–117.
16. Le Borne (S.), Grasedyck (L.) et Kriemann (R.). – Domain-decomposition Based $\mathcal{H}$-LU Preconditioners. *In : Domain decomposition methods in science and engineering XVI*, pp. 667–674. – Springer, 2007.
17. L'Excellent (J.-Y.). – *Multifrontal Methods : Parallelism, Memory Usage and Numerical Aspects*. – Habilitation à diriger des recherches, Ecole normale supérieure de lyon - ENS LYON, septembre 2012.
18. Lizé (B.). – *Résolution Directe Rapide pour les Éléments Finis de Frontière en Électromagnétisme et Acoustique : $\mathcal{H}$-Matrices. Parallélisme et Applications Industrielles.* – Thèse de PhD, Université Paris 13, 2014.
19. Pichon (G.), Darve (E.), Faverge (M.), Ramet (P.) et Roman (J.). – Sparse Supernodal Solver Using Block Low-Rank Compression. – In *PDSEC'2017 workshop of IPDPS*, Orlando, United States, May 2017.
20. Saad (Y.) et Van Der Vorst (H. A.). – Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, vol. 123, n1, 2000, pp. 1–33.

21. Xia (J.), Chandrasekaran (S.), Gu (M.) et Li (X. S.). – Superfast multifrontal method for large structured linear systems of equations. *SIAM Journal on Matrix Analysis and Applications*, vol. 31, n3, 2009, pp. 1382–1411.