HAL
open science

Conception d'un solveur linéaire creux parallèle hybride direct-itératif

Jérémie Gaidamour

To cite this version:

Jérémie Gaidamour. Conception d'un solveur linéaire creux parallèle hybride direct-itératif. Modélisation et simulation. Université Sciences et Technologies - Bordeaux I, 2009. Français. NNT : . tel-00456605

HAL Id: tel-00456605

https://theses.hal.science/tel-00456605v1

Submitted on 15 Feb 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 3904

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Jérémie Gaidamour**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Conception d'un solveur linéaire creux parallèle hybride direct-itératif

---

Soutenue le : 8 décembre 2009

Après avis des rapporteurs :

Serge Gratton	Professeur, ENSEEIHT
Yousef Saad	Professeur, Université du Minnesota

Devant la commission d'examen composée de :

Oliver Coulaud	Directeur de recherche, INRIA	Président du Jury
Luc Giraud	Directeur de recherche, INRIA	Examinateur
Serge Gratton	Professeur, ENSEEIHT	Rapporteur
Pascal Hénon	Chargé de recherche, INRIA	Directeur de Thèse
Jean-Jacques Pesqué	Chercheur, CEA	Examinateur
Jean Roman	Professeur, ENSEIRB	Directeur de Thèse

– 2009 –

Remerciements

Cher lecteur, c'est par cette page que j'achève la rédaction de mon manuscrit. Je suis profondément heureux d'écrire enfin ces remerciements et de pouvoir ainsi exprimer ma sincère reconnaissance à tous ceux qui m'ont accompagné pendant ces trois dernières années pour mener à bien ce travail.

Je voudrais commencer par remercier Serge Gratton pour avoir accepté de rapporter sur mon travail et pour les remarques qu'il a formulées sur mon manuscrit. Je remercie aussi particulièrement Yousef Saad pour avoir suivi avec intérêt mon travail tout au long de ma thèse et pour m'avoir accueilli plusieurs mois à l'université du Minnesota. Ces échanges ont considérablement contribué à ouvrir des perspectives sur mes travaux de recherche et je suis honoré qu'il ait accepté d'être mon second rapporteur.

Je remercie également tous les membres de mon jury de thèse. En premier lieu, je remercie Oliver Coulaud, pour avoir accepté d'être mon directeur de jury ; Luc Giraud pour l'intérêt qu'il a porté à mes travaux et Jean-Jacques Pesqué à travers qui je remercie également l'ensemble des personnes du CEA/CESTA avec lesquelles j'ai eu l'occasion de travailler. Je remercie aussi tout naturellement Jean Roman, mon directeur de thèse, notamment pour m'avoir encouragé à prolonger mes études par une thèse, avoir facilité et guidé mes premiers pas dans le monde de la recherche.

Je ne sais comment exprimer toute ma gratitude envers Pascal Hénon qui m'a encadré tout au long de cette thèse. J'ai initialement travaillé sous sa direction en projet à l'ENSEIRB puis en Master Recherche et je fus très heureux qu'il accepte de continuer à m'encadrer pour ma thèse. Tout au long de ce parcours, sa disponibilité a été sans faille. Il a su m'expliquer, me ré-expliquer et m'expliquer encore avec patience tout ce qui a pu longtemps m'échapper. Son encadrement a permis un déroulement idéal de ma thèse. Il a su à la fois maintenir le cap et me laisser une totale liberté de manœuvre dans la réalisation des objectifs. Je ne sais pas si c'est normal, mais il a rendu ces trois années de travail plus qu'agréable. Merci pour tout Pascal.

Impossible de ne pas remercier ici l'ensemble des membres des équipes ~~ScAlApplix~~, Bacchus et HiePACS. Je remercie tout d'abord l'ancienne génération, c'est-à-dire les MdC et les thésards « super-sérieux » pour avoir aussi bien accueilli les petits nouveaux. Grâce à vous, j'ai beaucoup appris sur les techniques d'organisation « juste-à-temps » ou « zéro-délai ». J'ai une pensée particulière pour ceux de ma génération et spécialement pour les ENSEIRBiens qui ont suivi la même voie que moi. Les moulte déménagements m'ont permis de connaître rapidement et d'apprécier toute la nouvelle génération de thésards. Je les félicite pour avoir plutôt bien toléré ma méchanceté. Cela va me manquer de ne plus pouvoir vous embêter ;-).

Sur un ton plus léger, je remercie tout ceux qui ont participé à des missions mémorables, notamment en Suisse, au Brésil et en Chine (trop dure la vie de thésard). Merci à ceux qui ont accepté d'être photographié ou filmé pour la postérité, pas toujours en possession de tous leurs moyens... D'un point de vue culinaire, je remercie les organisateurs de l'épiphanie journalière, des barbecues mais aussi le restaurant du CNRS et les « BDI » qui m'y ont accompagné. Je me dois de remercier ici l'incendie des locaux de l'INRIA qui m'a évité d'avoir à trier mes documents à la fin de ma thèse...

Au-delà des relations de travail et des amis, il y a toute ma famille dont l'influence a été déterminante dans la réalisation de ce projet. Je souhaite donc remercier mes parents pour le soutien qu'ils m'ont apporté pendant toutes ces années d'études. Je vous suis très reconnaissant de m'avoir toujours encouragé dans la voie que j'avais choisie et de m'avoir toujours offert les meilleures conditions d'études. Merci aussi d'avoir réalisé un rêve d'enfant en me permettant de devenir pilote pendant mes années de thèse.

Ces remerciements ne pourraient être complets sans un grand merci à Laure qui m'a accompagné pendant toutes ces années. Merci pour le bonheur que tu m'offres chaque jour.

Résumé

Cette thèse présente une méthode de résolution parallèle de systèmes linéaires creux qui combine efficacement les techniques de résolutions directes et itératives en utilisant une approche de type complément de Schur. Nous construisons une décomposition de domaine. L'intérieur des sous-domaines est éliminé de manière directe pour se ramener à un problème sur l'interface. Ce problème est résolu grâce à une méthode itérative préconditionnée par une factorisation incomplète. Un réordonnancement de l'interface permet la construction d'un préconditionneur global du complément de Schur. Des algorithmes minimisant le pic mémoire de la construction du préconditionneur sont proposés. Nous exploitons un schéma d'équilibrage de charge utilisant une répartition de multiples sous-domaines sur les processeurs. Les méthodes sont implémentées dans le solveur Hips et des résultats expérimentaux parallèles sont présentés sur de grands cas tests industriels.

Mots-clés: calcul haute performance, parallélisme, algèbre linéaire creuse, solveur parallèle de systèmes linéaires creux, méthode hybride directe-itérative, factorisation incomplète, complément de Schur, décomposition de domaine.

Abstract

This thesis presents a parallel resolution method for sparse linear systems which combines effectively techniques of direct and iterative solvers using a Schur complement approach. A domain decomposition is built ; the interiors of the subdomains are eliminated by a direct method in order to use an iterative method only on the interface unknowns. The system on the interface (Schur complement) is solved thanks to an iterative method preconditioned by a global incomplete factorization. A special ordering on the Schur complement allows to build a scalable preconditioner. Algorithms minimizing the memory peak that appears during the construction of the preconditioner are presented. The memory is balanced thanks to a multiple domains per processors parallelization scheme. The methods are implemented in the Hips solver and parallel experimental results are presented on large industrial test cases.

Keywords: high-performance computing, parallelism, sparse linear algebra, parallel solver for sparse linear systems, direct-iterative hybrid method, incomplete factorization, Schur complement, domain decomposition.

Table des matières

Introduction générale **1**

1 État de l'art **3**

1.1 Introduction 4

1.2 Méthodes de résolution directe 6

1.2.1 Renumérotation des inconnues 7

1.2.2 Factorisation symbolique 9

1.2.3 Factorisation numérique 11

1.3 Méthodes de résolution itérative 15

1.3.1 Méthode par projection dans un espace de Krylov 16

1.3.2 Technique de préconditionnement 17

1.4 Préconditionnement par factorisation incomplète 19

1.4.1 Factorisations ILU(k) 19

1.4.2 Factorisations ILUT(τ) 21

1.5 Méthodes de décomposition de domaine 23

1.5.1 Techniques de décomposition de domaine 24

1.5.2 Méthodes de Schwarz 24

1.5.3 Technique du complément de Schur 27

1.5.4 Utilisation du complément de Schur dans une technique de décomposition de domaine 27

1.5.5 Préconditionneurs utilisant des techniques de décomposition de domaine . 28

2 Méthode de résolution hybride fondée sur une décomposition de domaine **31**

2.1 Construction d'une factorisation incomplète 32

2.1.1 Motivations de l'approche 32

2.1.2 Rénumérotation et partitionnement des inconnues de l'interface 33

2.1.3 Ordonnancement global de l'élimination du complément de Schur 37

2.1.4 Construction d'une décomposition de domaine 38

2.1.5 Factorisation symbolique 40

2.2 Construction d'une méthode hybride directe-itérative 40
2.2.1 Utilisation du complément de Schur 40
2.2.2 Variantes algorithmiques de résolution 42
2.3 Expérimentations 46
2.3.1 Présentation des cas tests 46
2.3.2 Etude des partitions obtenues sur des matrices irrégulières 47
2.3.3 Etude asymptotique de la convergence de la méthode 48
2.3.4 Etude du remplissage du préconditionneur 49
2.3.5 Etude des variantes algorithmiques 52

3 Algorithmique 55
3.1 Construction à faible coût mémoire 59
3.1.1 Factorisation incomplète à partir du complément de Schur exact 61
3.1.2 Approche fondée sur le calcul approché du complément de Schur 65
3.2 Parallélisation et équilibrage de charge 71

4 Etude expérimentale et validation 77
4.1 Etude séquentielle 80
4.2 Etude parallèle en temps et en mémoire 87
4.3 Passage à l'échelle sur de grands cas tests industriels 91

Conclusion et perspectives 93

Annexes 95

A Résultats expérimentaux complémentaires 95

B Bibliographie 105

C Liste des publications 111

Liste des Algorithmes

1 Factorisation $L.U$ scalaire (version KIJ, sur place) 6
2 Factorisation symbolique 9
3 Factorisation symbolique par blocs 11
4 Factorisation numérique $L.U$ par blocs denses (version « Right-Looking ») 12
5 Factorisation $L.U$ par blocs pour les matrices creuses (version « Left-Looking ») . 14
6 Factorisation ILU avec remplissage prédéfini 19
7 Factorisation symbolique ILU(K) 21
8 Version IKJ de l'algorithme de factorisation LU 22
9 Factorisation ILU(τ, p) 23
10 Méthode de Schwarz multiplicative appliquée à deux sous-domaines 26
11 Méthode de Schwarz multiplicative appliquée à p domaines 26
12 Méthode de Schwarz multiplicative avec coloriage 26
13 Méthode de Schwarz additive 26
14 Algorithme de factorisation ILUM. 29
15 Algorithme de constuction du séparateur dans l'arbre d'élimination par blocs d'une renumérotation de type dissection emboîtée. 40
16 Factorisation incomplète de A selon la structure bloc induite par la partition ($\mathscr{R}_S$ ou $\mathscr{R}_C$) 58
17 Étapes du calcul du préconditionneur 59
18 Factorisation LU selon la structure bloc induite par la partition, sans stockage de la matrice de couplage (Algo. I) 61
19 Factorisation L.U selon la structure bloc induite par la partition, avec seuillage numérique (Algo. II) 68
20 Factorisation incomplète parallèle du complément de Schur selon $\mathscr{R}_C$ ou $\mathscr{R}_S$ . . . 74

Liste des tableaux

2.1 Propriétés générales des cas tests 47
2.2 Partitionnement de la matrice SHIPSEC5 (2D) 48
2.3 Partitionnement de la matrice HALTERE (3D) 48
2.4 Détails du remplissage du préconditionneur 51
2.5 Tableau récapitulatif du stockage mémoire nécessaire dans les quatre variantes du préconditionneur 52

3.1 Propriétés générales des cas tests 63
3.2 Comparaison entre l'algorithme 18 et l'algorithme 19. 70
3.3 Etude expérimentale de l'équilibrage de charge 75

4.1 Propriétés générales des cas tests 79
4.2 AMANDE - 3034 domaines - $\mathscr{R}_S+\tau = 0$ 91
4.3 Propriétés du cas test 10MILLIONS 91
4.4 10MILLIONS - 2193 domaines - $\mathscr{R}_S+\tau = 0$ 91

Table des figures

1.1 Etapes de résolution d'un solveur direct. 7
1.2 Exemple de matrice creuse avant et après application d'une permutation, notée P. 8
1.3 Arbre d'élimination correspondant à la dissection emboîtée d'un maillage physique 9
1.4 Structure de données par blocs, graphe des échanges et arbre d'élimination par blocs associés. 10
1.5 Notations de la factorisation numérique $L.U$ par blocs denses (version « Right-Looking ») 12
1.6 Illustration des factorisations « Right-Looking » et « Left-Looking ». 13
1.7 Factorisation numérique $L.U$ par blocs denses (version « Left-Looking ») 14
1.8 Factorisation ILU(k) et niveaux de remplissage 21
1.9 Décomposition d'un domaine en deux sous-domaines 25
1.10 ARMS : un solveur algébrique récursif utilisant une approche multi-niveaux. . . . 30

2.1 Notion de remplissage compatible 33
2.2 Grille 8×8 décomposée en neuf sous-domaines et structure hiérarchique associée. 34
2.3 Matrice associée à la grille de la figure 2.2 après renumérotation. 35
2.4 Décomposition hiérarchique de l'interface 35
2.5 Structure initiale de la matrice A. Correspond au remplissage autorisé dans la stratégie $\mathscr{R}_C$. 36
2.6 Structure de la matrice A avant la factorisation de S. Correspond au remplissage autorisé dans la stratégie $\mathscr{R}_S$. 37
2.7 Graphe d'adjacence des connecteurs après l'élimination de l'intérieur des sous-domaines. 38
2.8 Ordonnancement global de l'élimination des connecteurs pour une stratégie $\mathscr{R}_S$. . 39
2.9 Découpage du système et application de la méthode du complément de Schur à partir d'une numérotation du direct. 39
2.10 Illustration de la factorisation symbolique 41
2.11 Structure bloc du préconditionneur 41
2.12 Etude asymptotique de la convergence (remplissage $\mathscr{R}_C$ et $\mathscr{R}_S$) 49
2.13 Coût relatif d'une itération pour M_0, M_1 et M_2 54

3.1 Notations de la structure bloc induite par la partition (1/2) 58
3.2 Notations de la structure bloc induite par la partition (2/2) 58
3.3 Algorithme de construction du préconditionneur. 59
3.4 Factorisation symbolique par bloc de la matrice symétrique BCSSTK14 60
3.5 Illustration de l'algorithme 18 62
3.6 Etude du pic mémoire de l'algorithme 18 64

3.7 Illustration de l'algorithme 19 67
3.8 Etude du pic mémoire de l'algorithme 19 70
3.9 Décomposition de domaines et matrices locales ($\mathscr{R}_C$) 71
3.10 Répartition des sous-domaines entre les différents processeurs 72
3.11 Répartition des sous-domaines entre les processeurs, dans le cas d'une grille régulière 2D 73
3.12 Notations de l'algorithme 20 74

4.1 Le logo de HIPS 78
4.2 Illustration des cas tests du CEA/CESTA 79
4.3 Matrice AUDI : historique de convergence 82
4.4 Matrice AUDI : influence du nombre de sous-domaines 85
4.5 Matrice AUDI : influence du nombre de sous-domaines sur la mémoire 86
4.6 Scalabilité en temps 88
4.7 Scalabilité mémoire 89
4.8 Scalabilité du pic mémoire 90

1 Historique de convergence à taille de sous-domaines fixée (en nombre d'itérations) - $N_{cible} = 2000$ 96
2 Historique de convergence à taille de sous-domaines fixée (en temps) - $N_{cible} = 2000$ 97
3 Nombre d'itérations en fonction de la taille des sous-domaines (10^{-7}) 98
4 Temps séquentiel pour le calcul du préconditionneur, en fonction de la taille des sous-domaines (10^{-7}) 99
5 Temps séquentiel de résolution, en fonction de la taille des sous-domaines (10^{-7}) 100
6 Temps séquentiel total en fonction de la taille des sous-domaines (10^{-7}) 101
7 Ratio mémoire en fonction de la taille des sous-domaines (10^{-7}) 102
8 Pic mémoire en fonction de la taille des sous-domaines (10^{-7}) 103

Introduction générale

La résolution de grands systèmes linéaires creux est une brique fondamentale de nombreuses simulations numériques, en sciences ou en ingénierie. Ces grands systèmes numériques proviennent la plupart du temps de la discrétisation et de la linéarisation d'équations aux dérivées partielles de type elliptiques ou paraboliques ; mais on les retrouve aussi dans de nombreuses autres applications. On peut citer à titre d'exemple la conception de circuits électroniques, la simulation de réseaux d'énergie, de processus chimiques industriels, de modèles économiques ou de systèmes à file d'attente. Les systèmes à résoudre peuvent atteindre des tailles de plusieurs dizaines voire de plusieurs centaines de millions d'inconnues, particulièrement lorsque l'on considère des applications irrégulières 3D en vraie grandeur. La résolution des systèmes est souvent la partie des simulations numériques la plus consommatrice aussi bien en temps CPU qu'en espace mémoire. Les coûts opératoires et mémoires induits sont tels que le parallélisme est alors une technique incontournable pour résoudre ces très grands systèmes.

Les études sur les méthodes directes de résolution ont permis des progrès considérables en efficacité. En effet, les techniques de partitionnement et de calcul par blocs denses permettent d'exploiter particulièrement bien les calculateurs hautes performances actuels. Malgré ces avancées, l'utilisation des méthodes directes reste limitée par leurs énormes besoins en mémoire et les méthodes itératives sont généralement utilisées pour surmonter ces problèmes de passage à l'échelle. Certaines méthodes itératives sont développées pour des familles de problèmes. D'autres visent à la même généricité que les méthodes directes, mais elles sont beaucoup moins robustes : une convergence rapide vers la solution peut être difficile, voire impossible à obtenir en un temps raisonnable pour des problèmes trop complexes.

L'objectif de cette thèse est de concevoir un solveur linéaire creux capable de résoudre des cas tests difficiles et/ou avec une grande précision à moindre coût mémoire afin de réduire l'écart entre ces deux grandes familles de méthodes. Nous cherchons à développer une méthode de résolution algébrique n'utilisant comme information que la matrice du système linéaire à résoudre. La méthode doit être facilement parallélisable, afin de permettre l'exploitation des architectures des machines de calculs scientifiques actuelles et un bon passage à l'échelle. Pour couvrir un large spectre de problèmes, on cherche aussi à obtenir une méthode souple, dont on peut régler la robustesse et le coût en fonction de la difficulté du problème.

Pour atteindre ces objectifs, l'approche envisagée est de construire un solveur hybride direct/itératif couplant une résolution directe à une méthode itérative par une technique de décomposition de domaines. Le domaine de calcul initial est subdivisé en sous-domaines ; l'intérieur des sous-domaines est éliminé par une méthode directe, ce qui permet de se concentrer sur la résolution du problème restreint aux interfaces par n'importe quelle méthode itérative. Pour résoudre ce problème réduit, nous nous intéressons plus spécifiquement aux méthodes itératives de type Krylov (ex : GMRES) préconditionnées par une factorisation incomplète.

D'autres travaux [42] s'intéressent à ce type d'approche hybride, mais utilisent une méthode de Schwarz additive sur l'interface entre les domaines. Le préconditionnement proposé ici est plus

sophistiqué. On utilise sur l'interface la renumérotation et la partition du graphe d'adjacence proposée dans [47] pour construire un préconditionneur global sur l'interface conservant un stockage local des facteurs, grâce à une renumérotation compatible entre les sous-domaines. Cette renumérotation originale des inconnues de l'interface permet de construire une factorisation incomplète par blocs dont la structure est connue à l'avance par un critère combinatoire. Cette renumérotation exhibe un parallélisme naturel exploitable par le solveur.

La partition initiale en sous-domaines est obtenue, quant à elle, de manière algébrique à partir d'une renumérotation provenant des méthodes directes. La taille des sous-domaines permet de contrôler à la fois le temps et l'espace mémoire nécessaires à la résolution. Le découpage en sous-domaines est utilisé dans le schéma de parallélisation du solveur.

Le chapitre 1 dresse un état de l'art des méthodes de résolution. Cet état de l'art se focalise naturellement sur les techniques de résolutions algébriques qui exploitent des factorisations et aborde les principales techniques qui seront réutilisées dans notre approche. Nous y présentons les méthodes directes et les méthodes de Krylov préconditionnées par des factorisations incomplètes. Une dernière partie de ce chapitre est consacrée aux techniques de décomposition de domaines.

Le chapitre 2 décrit la méthode de résolution développée dans le cadre de cette thèse. Il détaille la factorisation incomplète qui est employée. Celle-ci est définie par les renumérotations qui sont utilisées à l'intérieur des sous-domaines et sur l'interface. Nous verrons aussi comment coupler efficacement résolutions directes et itératives par une approche de type complément de Schur. Une étude expérimentale permet d'introduire les travaux du chapitre suivant.

Le chapitre 3 présente l'ensemble de l'algorithmique mise en œuvre dans la méthode de résolution. Une première partie présente deux variantes algorithmiques du préconditionneur permettant le calcul de la factorisation incomplète avec un faible coût mémoire. Une seconde partie est consacrée à la parallélisation de la méthode. On y introduit aussi un schéma d'équilibrage de charge fondé sur l'attribution de plusieurs sous-domaines par processeurs et permettant d'obtenir une bonne scalabilité, en temps et en mémoire.

Dans **le chapitre 4** nous comparons les performances de notre solveur avec celles d'un préconditionneur Schwarz Additif. Nous finissons sur des résultats parallèles afin d'illustrer les capacités de passage à l'échelle de notre méthode sur de grands cas tests industriels difficiles.

Enfin, nous conclurons en dressant les perspectives de ce travail.

Chapitre 1

État de l'art

Sommaire

1.1 Introduction **4**
1.2 Méthodes de résolution directe **6**
1.2.1 Renumérotation des inconnues 7
1.2.2 Factorisation symbolique 9
1.2.3 Factorisation numérique 11
1.3 Méthodes de résolution itérative **15**
1.3.1 Méthode par projection dans un espace de Krylov 16
1.3.2 Technique de préconditionnement 17
1.4 Préconditionnement par factorisation incomplète **19**
1.4.1 Factorisations ILU(k) 19
1.4.2 Factorisations ILUT(τ) 21
1.5 Méthodes de décomposition de domaine **23**
1.5.1 Techniques de décomposition de domaine 24
1.5.2 Méthodes de Schwarz 24
1.5.3 Technique du complément de Schur 27
1.5.4 Utilisation du complément de Schur dans une technique de décomposition de domaine 27
1.5.5 Préconditionneurs utilisant des techniques de décomposition de domaine 28

1.1 Introduction

On s'intéresse à la résolution de grands systèmes linéaires creux de la forme :

$$A.x = b \tag{1.1}$$

où $A = (a_{ij})$ est une matrice carrée inversible d'ordre n, x est le vecteur solution recherché et b est le vecteur second membre du système.

On dit qu'une matrice est **creuse** lorsqu'elle comporte une forte proportion de coefficients nuls. Concrètement, les matrices creuses correspondent à des systèmes d'équations dans lesquels chaque équation fait intervenir un très faible nombre d'inconnues. Lorsqu'une matrice A est creuse, son inverse A^{-1} est en général beaucoup plus **dense** : A^{-1} contient beaucoup de termes non-nuls. Par contre, les matrices triangulaires L et U issues de la factorisation $A = L.U$ de A peuvent être raisonnablement creuses lorsqu'on utilise des techniques adaptées. D'un point de vue général, on cherche donc à utiliser des algorithmes et des structures de données qui prennent en compte la structure creuse des matrices, dans le but de réduire les coûts en mémoire et en calcul qui seraient induits par un calcul en dense, si on ne tenait pas compte de cette particularité. Les opérations matricielles classiques peuvent être effectuées de manière économique sans stocker les termes nuls des matrices et sans réaliser de calculs inutiles entre termes nuls.

Pour résoudre le système (1.1), on peut distinguer deux grandes classes de méthodes :

- **Les méthodes directes** sont basées sur une élimination de Gauss en creux, c'est à dire des factorisations de A (factorisation de A en matrices $L.U$ facilement inversible par exemple).
- **Les méthodes itératives** consistent à raffiner une solution initiale grossière pour obtenir une solution approchée d'une précision suffisante.

Ces deux classes de méthodes ont toujours été en compétition.

Avec les nombreux solveurs directs creux parallèles développés ces dernières années (voir par exemple [2, 45, 57]), il est maintenant possible de résoudre **efficacement** des systèmes linéaires de très grandes tailles, en particulier lorsque le problème initial sous-jacent est 2D. Les solveurs directs actuels exploitent notamment très bien les performances des processeurs modernes, en exhibant une structure dense par blocs des matrices creuses. Les ressources nécessaires en temps et en mémoire sont facilement prédictibles [27, 34], ce qui permet une bonne distribution des calculs sur les architectures parallèles. De plus, l'approche est **générique** et les solveurs directs peuvent être utilisés comme des *boîtes noires* dans les codes de simulation. Les solutions obtenues par ces solveurs sont très précises (écart à la solution de l'ordre de la précision machine) et la méthode est peu sensible à la difficulté numérique des problèmes traités (la résolution est assurée). On dit alors que la méthode est **robuste**. Les méthodes directes sont ainsi souvent utilisées dans les codes industriels où la fiabilité est primordiale ou pour des problèmes difficiles. Ainsi, ils sont utilisés notamment lorsque les équations ne proviennent pas d'équations aux dérivées partielles, pour les problèmes de circuits ou les modélisations de l'industrie chimique.

Si l'efficacité des méthodes directes est maintenant difficile à surpasser sur des problèmes 2D, les inconnues des problèmes 3D sont plus couplées et les facteurs sont plus denses qu'en 2D. Le passage à l'échelle sur de grands problèmes est rendu difficile par les coûts en nombre d'opérations de la factorisation et le coût mémoire du stockage des facteurs (L et U). Les méthodes itératives sont alors souvent utilisées, en particulier pour les problèmes provenant de la discrétisation d'équations aux dérivées partielles en 3D. Les simulations multi-physiques 3D induisent des systèmes linéaires de centaines de millions voire de milliards d'équations et d'autant d'inconnues. Même sans considérer ces systèmes, il est maintenant fréquent de résoudre des systèmes de plusieurs millions d'inconnues et l'on cherche donc à utiliser sur de tels systèmes des méthodes itératives pour réduire les coûts de résolution. En effet, les méthodes itératives

nécessitent souvent moins d'opérations et de mémoire que les méthodes directes, en particulier quand la solution n'a pas besoin d'être très précise.

Les méthodes itératives couvrent un large spectre de techniques, allant des méthodes stationnaires (comme la méthode de Jacobi, de Gauss-Seidel ou les méthodes SOR) aux méthodes multi-niveaux en passant par les méthodes de Krylov [79]. De nombreuses techniques itératives sont conçues pour résoudre un problème spécifique. Certaines, comme les méthodes multi-grilles, sont ainsi optimales pour une classe donnée de problèmes : sous certaines conditions, il est possible d'obtenir une complexité linéaire en mémoire et en nombre d'opérations. Mais ce type d'approche n'est pas toujours possible. Pour diverses raisons, il peut être difficile d'avoir une parfaite connaissance du problème et d'obtenir ou d'exploiter des informations pertinentes pour un solveur. Souvent, ce type d'approche ad hoc n'est pas utilisée en raison de son manque de généricité et de l'investissement important qu'il nécessite : la méthode est en général très sensible aux détails d'un problème et de petits changements peuvent compromettre fortement l'efficacité du solveur.

Toutes ces raisons expliquent l'intérêt pour les méthodes purement algébriques qui exploitent uniquement l'information contenue dans la matrice A. Bien que non optimales, ces méthodes résolvent avec une efficacité raisonnable un large panel de problèmes. Elles sont particulièrement adaptées aux problèmes irréguliers, comme ceux provenant de la discrétisation sur des maillages non structurés ou d'applications qui ne sont pas issues de maillage. Elles sont aussi plus faciles à développer, à utiliser et à adapter suite à des changements de simulation. En outre, il est toujours possible de paramétrer ou spécialiser un code générique pour exploiter les particularités d'un problème. On a alors affaire à des solveurs *boîte grise*, plus souples que les solveurs ad hoc et aux performances très satisfaisantes. Ce type de solveur peut aussi être réutilisé comme brique de base dans la conception de solveurs ad hoc (voir par exemple l'approche de [89]).

Les méthodes itératives n'ont par contre pas la robustesse des méthodes directes : pour un certain nombre d'applications, elles ne convergent pas ou difficilement ou en un temps prohibitif. Une technique très répandue pour améliorer la convergence des méthodes itératives consiste à préconditionner le système initial (1.1) [6]. Il s'agit de le transformer en un système équivalent dont les propriétés sont plus favorables à la convergence. On peut préconditionner le système en utilisant une factorisation incomplète de A. Il s'agit de calculer une approximation des facteurs L et U en limitant les calculs et la mémoire allouée au stockage de ces facteurs. Les frontières entre les deux classes de méthodes sont ainsi de plus en plus floues. Les méthodes itératives sont devenues plus robustes mais il reste difficile de trouver un bon compromis entre robustesse et efficacité.

Dans cet état de l'art, nous nous concentrons sur les méthodes de résolution qui utilisent uniquement les informations provenant de la matrice du système à résoudre et qui sont les plus universelles possibles. Une première partie de ce chapitre est consacrée aux méthodes directes et aux techniques qui les rendent efficaces. La deuxième partie est consacrée aux méthodes de résolution itératives. Elle présente plus spécifiquement les méthodes de Krylov et des techniques de préconditionnement algébrique. La troisième partie s'intéresse plus spécifiquement aux préconditionneurs de type factorisation incomplète. Enfin, une dernière partie présente des techniques de décomposition de domaines qui permettent à la fois d'introduire du parallélisme dans les préconditionneurs de type factorisation incomplète et le couplage de différentes méthodes de résolution.

1.2 Méthodes de résolution directe

Introduction

Cette section est consacrée aux méthodes de résolution directe. On y présente les modèles de graphes permettant l'analyse du processus de factorisation ainsi que les étapes nécessaires à la construction d'une factorisation directe performante. On pourra se reporter à [31, 26, 25, 71, 61, 34, 43, 23] et aux références incluses pour une vision plus complète des méthodes directes.

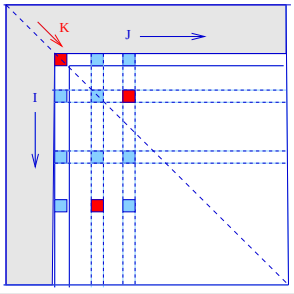
Les méthodes directes de résolution consistent à calculer la solution x du système (1.1) par la résolution successive de systèmes triangulaires ou diagonaux obtenus par factorisation de la matrice A. Il existe différents types de factorisations, au premier rang desquelles on peut citer :

- La factorisation « L.U » : une matrice A régulière peut être décomposée à une permutation près sous la forme $A = L.U$ où L est une matrice triangulaire inférieure à diagonale unité et U une matrice triangulaire supérieure ;
- La factorisation de Cholesky : une matrice A symétrique définie positive peut être décomposée sous la forme $A = L.L^T$ où L est une matrice triangulaire inférieure ;
- la factorisation de Crout : une matrice A symétrique définie positive peut être décomposée sous la forme $A = L.D.L^T$ où L est une matrice triangulaire inférieure à diagonale unité et D une matrice diagonale. Il est à noter qu'il est possible d'utiliser la factorisation $L.D.L^T$ pour des matrices symétriques non définies positives. Dans ce cas la matrice D est diagonale par blocs.

Après factorisation, la solution du système initial peut être obtenue par la résolution successive de sytèmes triangulaires. Par exemple, dans le cas d'une factorisation « LU », on résout :

$$\begin{cases} L.y &= b \\ U.x &= y \end{cases} \tag{1.2}$$

Algorithme 1 : Factorisation $L.U$ scalaire (version kij, sur place)



Entrées : Matrice $A = (a_{ij})$ régulière de dimension n
Sorties : Matrices L et U telles que $L.U = A$

3 **pour** $k = 1$ *à* n **faire**
  pour $i = k + 1$ *à* $n - 1$ **faire**
    $a_{ik} \leftarrow a_{ik}/a_{kk}$
    pour $j = k + 1$ *à* n **faire**
7 $a_{ij} \leftarrow a_{ij} - a_{ik} * a_{kj}$

L'algorithme 1 présente l'algorithme de factorisation $A = L.U$. La matrice A est factorisée sur place en remplaçant ses termes par ceux des deux matrices triangulaires L et U. Lorsque la matrice est creuse, seuls les éléments non nuls sont stockés et les opérations sur les éléments nuls sont supprimées. Mais, lors de l'affectation de la ligne 7 de l'algorithme, si a_{ij} était nul avant l'affectation, un nouvel élément non nul est introduit dans la matrice factorisée $L.U$. Au cours du processus de factorisation, de nouveaux termes non nuls sont donc créés et les facteurs sont plus remplis que la matrice initiale. Il s'agit du phénomène de **remplissage** [34]. De nombreuses techniques se sont développées pour permettre une factorisation efficace de matrices creuses. Il est ainsi possible de limiter ce remplissage et de prévoir la structure des facteurs. On utilise généralement pour cela deux étapes de prétraitements :

- **La renumérotation** : Cette étape consiste à trouver une permutation P telle que le remplissage de la factorisation de PAP^T soit minimal. Dans un contexte parallèle, on

cherche aussi à créer le plus de tâches indépendantes possibles dans la factorisation.

– **La factorisation symbolique** : Le but de cette étape est de calculer la structure de remplissage des facteurs avant l'étape de factorisation numérique. Elle permet d'éviter l'utilisation de structures de données dynamiques. Ces structures sont peu performantes car elles nécessitent des réallocations mémoires et empêchent un stockage contigu des données permettant une bonne utilisation des caches. Dans un contexte parallèle, la connaissance de la structure des facteurs permet aussi de prévoir un équilibrage de charge.

La figure 1.1 présente les étapes d'une résolution directe d'un solveur linéraire creux.

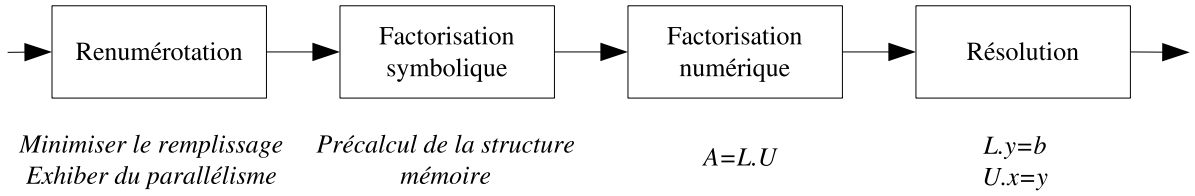


FIG. 1.1 – Etapes de résolution d'un solveur direct.

1.2.1 Renumérotation des inconnues

Le remplissage des facteurs est directement lié à l'ordre d'élimination des inconnues. Si l'on considère par exemple la matrice creuse A de la figure 1.2, sa factorisation induit des facteurs complètement denses, alors qu'il suffit de permuter la première et la dernière inconnue dans x pour que la factorisation de la matrice permutée PAP^T ne produise aucun remplissage. On dit que la matrice a été renumérotée. En réduisant le remplissage, on réduit aussi le nombre d'opérations à effectuer lors de la factorisation et des résolutions triangulaires.

La théorie liée aux factorisations directes a surtout été développée dans le cadre des matrices symétriques ou à structure symétrique. L'outil central d'analyse du remplissage est le modèle de **graphe d'adjacence** associé à la matrice à factoriser [34, 27]. Dans le cas des matrices non-symétriques, on peut se ramener à un cadre similaire en considérant le graphe de $A + A^t$ qui est symétrique.

Définition 1 (Graphe d'adjacence) On représente la structure d'une matrice A symétrique ou à structure symétrique par le graphe $G = (V, E)$ à n sommets ayant une arête $(i, j) \in E$ *ssi* $a_{ij} \neq 0$.

Définition 2 (Graphe d'élimination) On appelle graphe d'élimination le graphe d'adjacence $G^*(V, E^*)$ associé à la matrice factorisée $L + U$.

Dans l'algorithme 1, ligne 7, si au cours de la boucle sur k, a_{ik} et a_{kj} sont non nuls, le terme a_{ij} peut devenir non nul alors qu'il était initialement nul (pour i et j supérieurs à k). La factorisation de la $k$ième ligne et $k$ième colonne induit ainsi la création d'une arête pour chaque couple (i, j) de sommets adjacents au sommet k et de numéros supérieurs à k (si cette arête n'existe pas déjà). On peut en déduire le théorème suivant :

$$A = \begin{pmatrix} \bullet & \bullet & \bullet & \bullet & \bullet \\ \bullet & \bullet & & & \\ \bullet & & \bullet & & \\ \bullet & & & \bullet & \\ \bullet & & & & \bullet \end{pmatrix} \qquad PAP^T = \begin{pmatrix} \bullet & & & & \bullet \\ & \bullet & & & \bullet \\ & & \bullet & & \bullet \\ & & & \bullet & \bullet \\ \bullet & \bullet & \bullet & \bullet & \bullet \end{pmatrix}$$

FIG. 1.2 – Exemple de matrice creuse avant et après application d'une permutation, notée P.

Théorème 1 (Théorème de caractérisation du remplissage)

$\forall (i,j) \in \{1,\ldots,n\}$,

$$(i,j) \in E^* \Leftrightarrow \begin{cases} (i,j) \in E \\ \quad \text{ou si} \\ \text{il existe un chemin } (i,\ldots,k_1,k_2,\ldots,k_p,j) \text{ dans G} \\ \text{tq } \forall k_r,\ r \in \{1,\ldots,p\}, k_r < min(i,j) \end{cases}$$

L'un des points critiques dans la conception d'un solveur creux performant concerne donc le choix d'une renumérotation des inconnues qui minimise le nombre de chemins d'élimination [1, 34, 67, 68]. Le calcul d'une renumérotation des inconnues d'un système linéaire creux qui minimise le remplissage de la matrice lors de la factorisation est NP-complet [91] mais différentes heuristiques efficaces ont été proposées. Une première méthode consiste à minimiser le remplissage introduit entre deux itérations de la boucle externe de l'algorithme (la boucle d'indice k ligne 3 de l'algorithme 1). Une solution approchée de ce problème local de minimisation est la méthode de degré minimum [1, 35] qui consiste à permuter à chaque itération le nœud k avec celui de degré minimum d'indice supérieur. La permutation peut ainsi être construite à partir du modèle de graphe par un algorithme glouton. Dans un contexte parallèle, la renumérotation doit aussi permettre de maximiser l'indépendance dans les calculs de factorisation. Pour représenter les dépendances entre les itérations de la boucle externe, on utilise la notion d'**arbre d'élimination** [61].

Définition 3 (Arbre d'élimination) Il s'agit d'un arbre recouvrant du graphe G^* dont les arêtes sont définies par la relation de parenté suivante :

$$parent(j) = \min\{i \text{ tq } i > j \text{ et } (i,j) \in E^*\}$$

Il existe ainsi une arête entre i et j *ssi* la ligne du premier terme non nul dans la colonne j de la matrice factorisée est i. L'arbre d'élimination exhibe directement le parallélisme induit par le creux de la matrice : celui-ci est d'autant plus important que l'arbre d'élimination est large et de faible hauteur. Les renumérotations qui tendent à minimiser le remplissage et maximiser le parallélisme sont basées sur les techniques de « dissections emboîtées » [33, 59] qui partitionnent récursivement le graphe G en utilisant la notion de **séparateur**.

Définition 4 (Séparateur d'un graphe) Un séparateur sommet d'un graphe G est un ensemble de nœuds C tel que si on enlève de G le sous-ensemble de nœuds C et les arêtes qui y sont reliées, alors on obtient deux sous-graphes G_1 et G_2 non connectés.

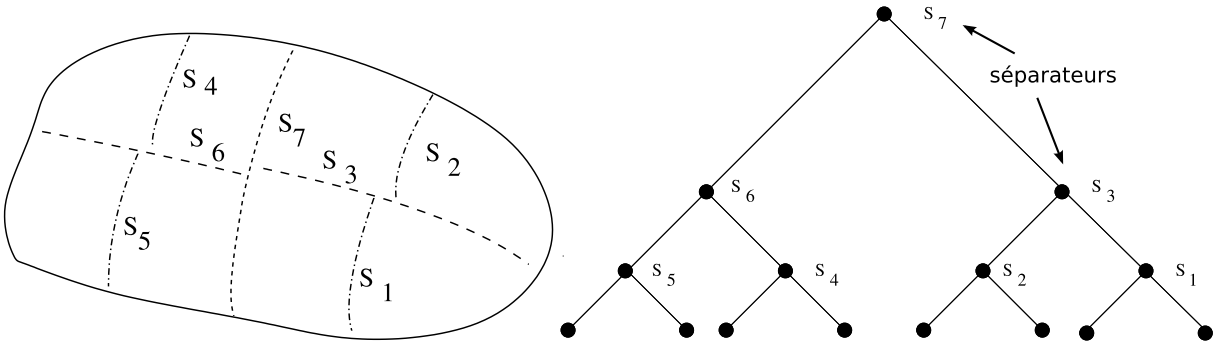


FIG. 1.3 – Arbre d'élimination correspondant à la dissection emboîtée d'un maillage physique

L'algorithme consiste à chercher un séparateur du graphe d'adjacence, à numéroter les sommets du séparateur avec les plus grands numéros disponibles afin qu'il ne puisse pas y avoir création d'arêtes de remplissage entre les sous-graphes séparés. On applique ensuite récursivement l'algorithme aux deux sous-graphes séparés pour obtenir la renumérotation finale. La figure 1.3 présente l'arbre d'élimination obtenu avec un tel algorithme. La qualité de la renumérotation est directement liée à la taille des séparateurs qui doit être faible et à l'équilibrage des tailles des sous-graphes obtenus par dissection.

Les logiciels de partitionnement et de renumérotation de graphes tels que METIS [53] ou SCOTCH [68] utilisent des techniques de type dissection emboîtées, couplées à des heuristiques locales (méthode de degré minimum) ainsi qu'à des algorithmes de contractions multi-niveaux [52].

1.2.2 Factorisation symbolique

Après avoir trouvé une renumérotation adaptée, les solveurs directs effectuent une étape de factorisation symbolique [74] pour déterminer la structure des facteurs, c'est-à-dire la position dans la matrice factorisée des éléments non nuls. Cette étape peut s'exprimer élégamment en terme d'arbre d'élimination. On note $[k]$ la structure creuse de la colonne i sous la forme d'une liste d'indices lignes en ordre croissant correspondant aux termes non-nuls. L'algorithme 2 présente la factorisation symbolique. La fonction `fusion[i][j]` fusionne les listes de termes non nuls. En partant du bas de l'arbre d'élimination, l'algorithme consiste à reporter le remplissage induit par la factorisation d'une colonne uniquement sur la première colonne impactée k' (le père de k dans l'arbre d'élimination). Les autres colonnes seront elles-mêmes impactées par k' en cours d'algorithme. L'algorithme ne nécessite donc que deux boucles, contre trois pour l'élimination de Gauss.

ALGORITHME 2 : Factorisation symbolique

Construire $[k]$ pour chaque colonne k de la matrice initiale
pour $k = 1$ *à* $n - 1$ **faire**
  $[\text{parent}(k)] = \texttt{fusion}([k], [\text{parent}(k)])$

Les solveurs directs tirent parti de l'efficacité de bibliothèque de calcul dense en exhibant une structure par blocs denses des facteurs de la matrice creuse (voir section 1.2.3). Cette structure provient directement des techniques de dissections emboîtées. En fait, les colonnes de L (et les lignes de U) ayant une structure creuse similaire peuvent être regroupées ensemble. Ces groupes de bloc-colonnes, appelés **super-nœuds**, permettent de construire une structure bloc-colonne

de la matrice. Ils correspondent pour la plupart aux séparateurs de la dissection emboîtée. Les super-nœuds peuvent être trouvés directement à partir de A, sans connaître la structure creuse finale des facteurs [62]. On peut alors utiliser la partition P induite par les super-nœuds pour calculer la structure bloc de la matrice factorisée en utilisant une factorisation symbolique par blocs. Cette factorisation exploite le fait que, lorsque P est obtenue par une dissection emboîtée, on peut travailler directement sur le graphe quotient de G induit par P. Dans ce graphe, un sommet représente une partie de P et les arêtes correspondent aux dépendances entre chaque partie dans la factorisation. Lorsque P est obtenue par une dissection emboîtée, en notant $Q(G, P)$ le graphe quotient de G induit par P, on a :

$$Q(G, P)^* = Q(G^*, P)$$

Ainsi, les opérations de passage au graphe quotient et d'élimination commutent et l'on peut définir un arbre d'élimination par blocs associé à $Q(G, P)^*$.

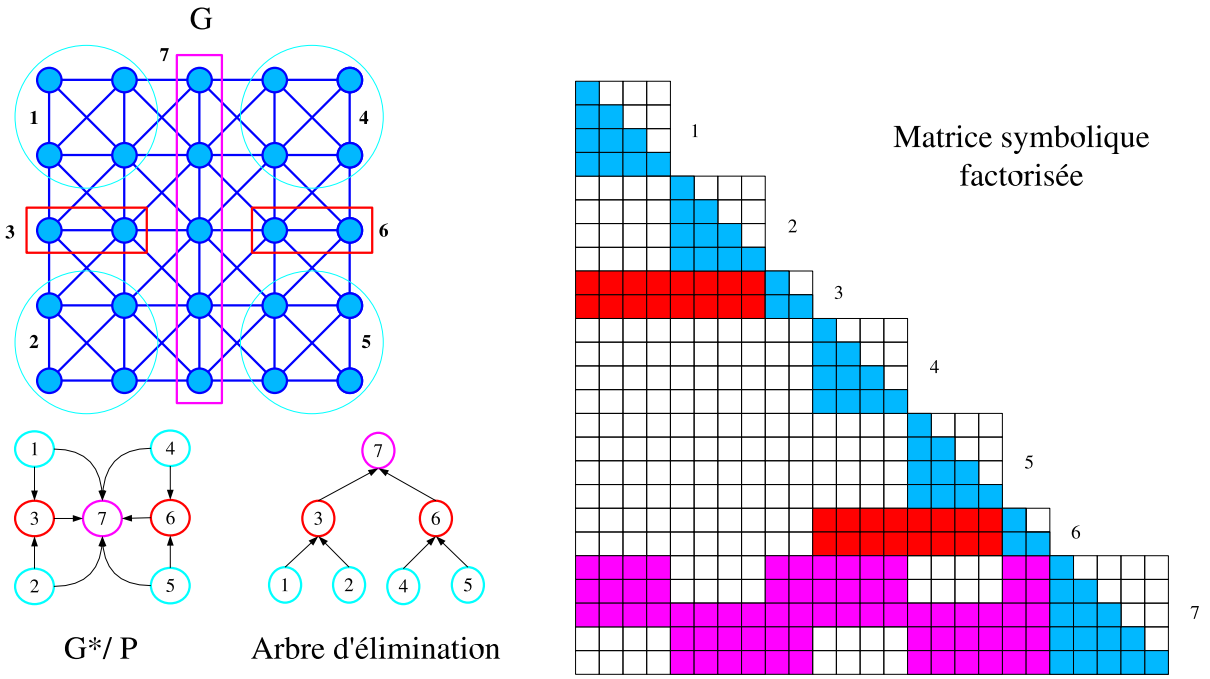


FIG. 1.4 – Structure de données par blocs, graphe des échanges et arbre d'élimination par blocs associés. Dans G, le séparateur du premier niveau de récursion de la dissection emboîtée est représenté en violet. Les séparateurs du second niveau sont en rouge. Dans la matrice factorisée, les mêmes couleurs sont reprises pour identifier les blocs de couplage associés à ces séparateurs.

La figure 1.4 présente une grille 5×5 renumérotée par dissection emboîtée, le graphe d'élimination quotienté, l'arbre d'élimination par blocs correspondant à la renumérotation et le remplissage de la matrice factorisée associée. La structure de la matrice est formée par des bloc-colonnes, contenant elle-même des blocs denses. Un bloc dense ne contient aucune ligne identiquement nulle [62].

Les blocs extra-diagonaux sont identifiables par une paire d'entiers (α, β) correspondant aux numéros de la première et dernière ligne du bloc. On note $[k]$ la structure creuse du bloc du super-nœud i sous la forme d'une liste d'intervalles (α, β) en ordre croissant et N le nombre de

super-nœuds. On peut ainsi construire un algorithme de factorisation par blocs en utilisant une fonction de fusion d'intervalles (`Bfusion`). L'algorithme 3 présente la factorisation par blocs. La complexité en temps et en espace de cet algorithme croît comme le nombre total de blocs extra-diagonaux dans la structure ainsi construite [16].

ALGORITHME 3 : Factorisation symbolique par blocs

Construire $[k]$ pour chaque super-nœud k de la matrice initiale
pour $k = 1$ *à* $N - 1$ **faire**
  $[\text{parent}(k)] =$ `Bfusion`$([k], [\text{parent}(k)])$

1.2.3 Factorisation numérique

L'étape de factorisation numérique est bien sûr la plus coûteuse. Sur un système à n inconnues renumérotées par dissection emboîtée et issu de la discrétisation d'un problème 3D, le stockage mémoire est en $O(n^{4/3})$ et le nombre d'opérations nécessaires est en $O(n^2)$ (voir [60] pour plus de détails).

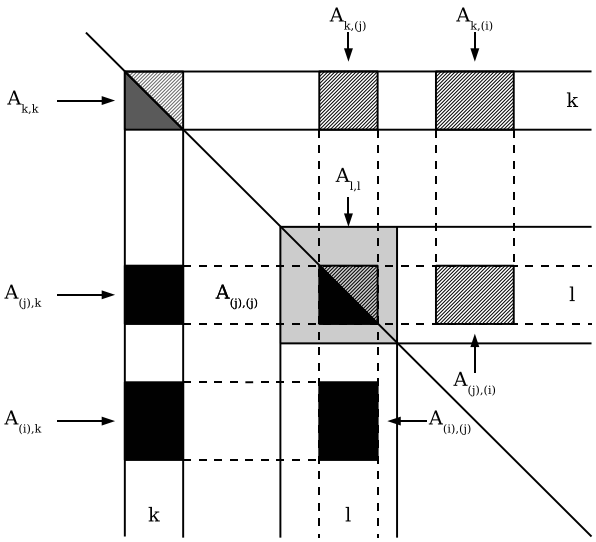
La technique principale pour rendre efficace la factorisation numérique est de réorganiser les calculs de l'algorithme 1 en fonction des super-nœuds de manière à remplacer les opérations scalaires par des opérations réalisables sur des blocs denses. Il est alors possible d'utiliser des routines de la bibliothèque BLAS (basic linear algebra subroutines, [55, 24, 22]) de niveau 3 entre ces blocs denses. Ces routines réalisent des calculs vectoriels et matriciels optimisés sur des structures denses en tirant parti des effets de caches pour se rapprocher de la puissance crête des processeurs. Le niveau 3 correspond à des opérations entre matrices, celles pour lesquelles le cache peut être le plus exploité. L'algorithme de factorisation par blocs denses présenté algorithme 4 exploite les informations de la factorisation symbolique. L'algorithme est illustré figure 1.5. Comme pour l'algorithme scalaire, on peut distinguer deux méta-opérations :

- Lignes 2 à 5 : factorisation du bloc-colonne k (factorisation du bloc diagonal et multiplication des blocs extradiagonaux par l'inverse du bloc diagonal par la résolution d'un système triangulaire),
- Lignes 6 à 8 : ajout des contributions de la factorisation du bloc-colonne k sur les colonnes suivantes.

L'opération sur les blocs extra-diagonaux des lignes 3 à 5 peut être réalisée de manière **compacte**, c'est-à-dire en remplaçant la boucle par un seul appel BLAS, lorsque les blocs des bloc-colonnes sont stockés dans une structure de données contiguës. De même, on peut réaliser l'opération $L_{(i),k}.U_{k,(j)}$ de la ligne 8 de manière compacte pour tous les blocs du bloc-colonne $(A_{i\in[k]})$ en utilisant un buffer temporaire.

On peut distinguer deux variantes de l'algorithme de factorisation. La version présentée jusqu'à présent (algorithmes 1 et 4) est la version dite « Right-Looking ». Elle consiste à calculer définitivement la factorisation du bloc-colonne courant k et à répercuter immédiatement ses contributions sur les bloc-colonnes suivants. Une autre approche, appelée « Left-Looking » consiste à retarder l'apport des contributions de la colonne factorisée. Les contributions apportées à une colonne $k' > k$ ne sont calculées que lors de la factorisation de la colonne impactée k'. La variante « Left-Looking » de l'algorithme 4 est présentée par l'algorithme 5. Les solveurs directs utilisent ces deux variantes pour envoyer au plus tôt des communications (« Right-Looking ») ou au contraire pour agréger les contributions avant de les communiquer à un autre processeur (« Left-Looking »). Les deux algorithmes sont illustrés figure 1.6.

Dans la factorisation directe d'une matrice creuse, on exploite essentiellement trois niveaux de parallélisme :



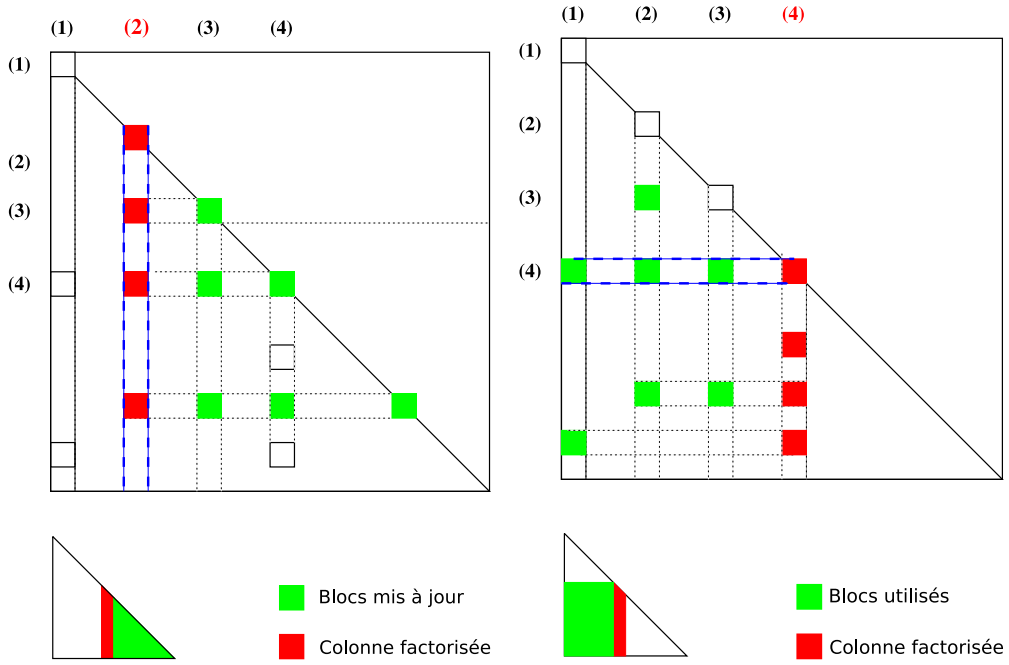
Notations :
Pour chaque bloc-colonne k, $1 \leq k \leq N$,
- $A_{k,k}$ est le bloc diagonal dense,
- $[k]$ est la liste des blocs extra-diagonaux du bloc-colonne k,
- $A_{(j),k}$ est le $j^{ième}$ bloc extra-diagonal dense, (j) étant un multi-indice (α, β) décrivant un intervalle de lignes.

De plus, $A_{(i),(j)}$ est le bloc rectangulaire dense correspondant aux lignes du multi-indice (i) et aux colonnes du multi-indice (j). $A_{(j),(j)}$ correspond à la même notion mais pour un sous-bloc du bloc diagonal $A_{l,l}$ faisant face au bloc extra-diagonal $A_{(j),k}$.

FIG. 1.5 – Notations de la factorisation numérique $L.U$ par blocs denses (version « Right-Looking »)

ALGORITHME 4 : Factorisation numérique $L.U$ par blocs denses (version « Right-Looking »)

pour $k = 1$ *à* N **faire**
  Factoriser $A_{k,k}$ en $L_{k,k}.U_{k,k}$
  pour $(i) \in [k]$ **faire**
    $L_{(i),k} \leftarrow A_{(i),k}.U_{k,k}^{-1}$
    $U_{k,(i)} \leftarrow L_{k,k}^{-1}.A_{k,(i)}$
  pour $(j) \in [k]$ **faire**
    pour $(i) \in [k]$ **faire**
      $A_{(i),(j)} \leftarrow A_{(i),(j)} - L_{(i),k}.U_{k,(j)}$



(a) « Right-Looking » (b) « Left-Looking »

FIG. 1.6 – Illustration des factorisations « Right-Looking » et « Left-Looking ».

1. le parallélisme à gros grain, induit par l'indépendance des calculs entre les sous-arbres d'un même nœud de l'arbre d'élimination (parallélisme induit par le creux),
2. le parallélisme à grain moyen, dû à la possibilité de raffiner la partition des inconnues en découpant un nœud et en le distribuant sur plusieurs processeurs (phase de repartitionnement),
3. le micro-parallélisme obtenu en tirant parti du parallélisme interne d'un processeur lors du calcul sur les blocs denses (utilisation optimale du pipeline du processeur). Ce dernier niveau de parallélisme est assuré par l'utilisation des primitives BLAS de niveau 3.

La parallélisation des méthodes directes consiste à répartir les nœuds de l'arbre d'élimination sur les processeurs disponibles. Une bonne distribution doit être équilibrée et doit optimiser la localité des données pour éviter les contentions réseaux. Pour tirer parti du premier niveau de parallélisme, les super-nœuds indépendants du bas de l'arbre d'élimination sont distribués entre les processeurs. On exploite le deuxième niveau de parallélisme en réalisant une phase de repartitionnement sur les plus gros super-nœuds. Ceux-ci sont situés dans le haut de l'arbre d'élimination et représentent la plus grande masse de calcul. Ils sont découpés et leur calcul est réparti sur plusieurs processeurs. Les sous-blocs peuvent être distribués par exemple de manière cyclique pour utiliser le parallélisme des calculs denses. On construit ainsi de manière statique une bonne régulation des calculs [28, 44, 32, 73, 75]. Les premiers super-nœuds de la matrice sont en général trop petits pour tirer une bonne efficacité des BLAS 3 et un algorithme d'amalgamation de bloc-colonne [46] très peu coûteux peut être utilisé pour former des super-nœuds plus gros. L'algorithme d'amalgamation consiste par exemple à autoriser 10% de remplissage supplémentaire (des vrais zéros) pour optimiser la taille des appels BLAS et accélérer ainsi le calcul de la factorisation. Une présentation plus complète de la parallélisation des méthodes directes dépasserait le cadre de cet état de l'art. Aussi, on pourra se reporter aux références [3, 65, 38, 39, 2] pour plus de détails.

Notations : Pour chaque bloc-colonne k, $1 \leq k \leq N$,
- $BCol(k)$ est l'ensemble des indices $i > k$ de bloc-colonnes en face des blocs extra-diagonaux du bloc-colonne k dans la matrice L ;
- $BLigne(k)$ est l'ensemble des indices $j < k$ de bloc-colonnes tels que $k \in BCol(j)$.

Ainsi, dans l'exemple de la figure 1.4, on a $N = 7$ et pour $k = 3$, $BCol(3) = \{7\}$ et $BRow(3) = \{1, 2\}$.

FIG. 1.7 – Factorisation numérique $L.U$ par blocs denses (version « Left-Looking »)

ALGORITHME 5 : Factorisation $L.U$ par blocs pour les matrices creuses (version « Left-Looking »)

pour $k = 1$ *à* N **faire**
  pour $(j) \in BLigne(k)$ **faire**
    pour $(i) \in BCol(j), i \geq k$ **faire**
      $A_{(i),k} \leftarrow A_{(i),k} - L_{(i),(j)}.U_{(j),k}$
      $A_{k,(i)} \leftarrow A_{k,(i)} - L_{k,(j)}.U_{(j),(i)}$
  Factoriser $A_{k,k}$ en $L_{k,k}.U_{k,k}$
  pour $(i) \in BCol(k)$ **faire**
    $L_{(i),k} \leftarrow A_{(i),k}.U_{k,k}^{-1}$
    $U_{k,(i)} \leftarrow L_{k,k}^{-1}.A_{k,(i)}$

1.3 Méthodes de résolution itérative

Introduction

Une méthode itérative consiste à résoudre un problème en utilisant une valeur initiale x^0 que l'on raffine pour se rapprocher graduellement de la solution. On construit donc une suite de vecteurs $x^1, x^2, \dots x^k$ qui converge vers le vecteur solution $x = (x_1, x_2, \dots, x_n)$. Les premières traces de méthodes itératives pour la résolution de systèmes linéaires apparaissent avec les travaux de Gauss, Jacobi, Seidel, et Nekrasov au XIXième siècle (voir [49] pour plus de détails). Les années 50 à 70 sont dominées par l'usage des méthodes itératives stationnaires. Ces méthodes consistent à modifier les composantes de l'approximation de la solution une par une (ou quelques-unes à la fois) dans un certain ordre jusqu'à atteindre une solution à la précision satisfaisante. On peut citer les quatre principales méthodes itératives stationnaires :

- La méthode de Jacobi consiste à calculer à l'étape k une nouvelle approximation de chaque composante $(x_1, x_2, \dots, x_n)$ du vecteur solution x de manière indépendante. On extrait de la $i$ème équation du système une nouvelle approximation de x_i en utilisant l'approximation précédente x^{k-1} pour les autres termes de x présents dans l'équation. On réalise donc une résolution par inconnues à chaque itération à partir de la solution de l'itération précédente. Cette méthode est facile à implémenter mais sa convergence est lente.
- La méthode de Gauss-Seidel est très similaire à la méthode de Gauss. Elle permet juste de prendre en compte les nouvelles approximations des composantes dès qu'elles sont calculées.
- La méthode de relaxation (SOR pour Successive over-relaxation) est une amélioration de la méthode précédente qui introduit un paramètre de relaxation pour améliorer la vitesse de convergence de la méthode.

En dépit de leur élégance théorique, ces méthodes itératives souffrent de sérieuses limitations comme leur manque de généricité. Leur convergence dépend de paramètres a priori difficilement estimables comme par exemple du spectre de la matrice. Sur de nombreux problèmes concrets, ces méthodes divergent ou convergent difficilement. Elles sont encore utilisées comme lisseurs de méthodes multi-grilles ou comme préconditionneurs de méthodes de Krylov.

En 1952, Lanczos [54], Hestenes et Stiefel [48] ont chacun découvert la méthode du gradient conjugué pour la résolution de systèmes avec A symétrique définie positive. Il s'agit d'une technique par projection qui consiste à chercher une approximation de la solution x dans un sous-espace de l'espace solution. Le gradient conjugué a d'abord été introduit comme une méthode directe garantissant la convergence en au plus n étapes et c'est à partir du milieu des années 70 que l'usage des méthodes de type Krylov s'est popularisé, avec la publication d'un papier de Reid [70] qui montre que pour de grandes matrices raisonnablement bien conditionnées, celle-ci permet d'atteindre une bonne précision en beaucoup moins d'étapes. Ce type de méthode a ensuite été étendu aux cas non symétriques avec le développement de méthodes comme le GMRES par Saad and Schultz [80] dans les années 80.

Les méthodes multi-grilles [10, 13, 85] ont été utilisées à partir des années 70, notamment par Brandt [12] et Hackbusch [41]. Elles se fondent sur l'utilisation d'un ensemble de grilles de différentes résolutions. L'idée est d'accélérer la convergence d'une méthode itérative en corrigeant la solution calculée sur un maillage fin par celle obtenue rapidement sur un maillage plus grossier. Des opérateurs de restriction et projection permettent de passer d'un maillage à l'autre. Ce type de méthodes a d'abord été proposé pour des problèmes particuliers comme des équations aux dérivées partielles elliptiques du second ordre pour des grilles régulières. Bien que ces méthodes ne soient pas complètement généralistes, elles ont maintenant été étendues à de nombreuses

classes de problèmes pour lesquelles elles convergent avec une complexité en $O(n)$. Les grilles des méthodes multi-grilles algébriques (AMG) ne sont construites qu'en utilisant les coefficients de la matrice du système et sont donc de véritables boîtes noires pour certaines classes de problèmes. On consultera par exemple les travaux de Dendy [21] et de Ruge et Stünben [76]. Un certain nombre d'idées de ces méthodes ont été reprises pour construire des variantes multi-niveaux de factorisations incomplètes. Ces factorisations sont présentées à la section 1.5.5. Elles visent un compromis entre l'optimalité des méthodes multi-grilles ad hoc et la généricité des factorisations.

Il est difficile de dépeindre un tableau complet de toutes les méthodes itératives de résolution, surtout si on y ajoute les diverses possibilités de préconditionnement qui permettent d'en améliorer la convergence. Nous nous intéressons ici aux méthodes par projections qui consistent à rechercher la solution dans un sous-espace de Krylov. Le principe général du préconditionnement est aussi présenté. Dans la section 1.4, on s'intéressera plus spécifiquement au préconditionnement par factorisation incomplète.

1.3.1 Méthode par projection dans un espace de Krylov

Soit le système $A.x = b$ à n inconnues où A est carrée et inversible. Les méthodes itératives par projection consistent à chercher une approximation de la solution x dans un sous-espace de $\mathbb{R}^n$. On appelle ce sous-espace l'espace de recherche. Soit K_m un espace de recherche de dimension m. Il faut en général imposer m contraintes pour obtenir une approximation de x dans K_m. Ces contraintes peuvent être exprimées sous la forme de m conditions d'orthogonalisation indépendantes : on impose au vecteur résidu $r = b - A.x$ d'être orthogonal à m vecteurs linéairement indépendants (condition de Petrov-Galerkin). Ces vecteurs définissent un autre sous-espace, noté L_m et appelé sous-espace des contraintes.

Soit x_0 une solution initiale et $r_0 = b - A.x_0$ le résidu initial correspondant. Les méthodes de Krylov sont des méthodes de projections pour lesquelles les sous-espaces de recherche K_m sont des sous-espaces de Krylov de la forme :

$$K_m(A, r_0) = Vec(r_0, A.r_0, \ldots, A^{m-1}.r0)$$

Théorème 2 La solution du système linéaire $A.x = b$ appartient à l'espace affine $x_0 + K_m(A, r_0)$ où m est la dimension maximale pour x_0 donné des espaces de Krylov (dont la famille est nécessairement bornée).

Les méthodes de Krylov consistent donc à calculer successivement l'approximation de la solution dans $x_0 + K_1(A, r_0)$, $x_0 + K_2(A, r_0)$, ... en utilisant à chaque fois la dernière solution approchée et ce, jusqu'à obtenir une solution de la précision souhaitée. Les approximations obtenues par une méthode de Krylov sont de la forme $x_m = x_0 + q_{m-1}.(A).r_0$ où q_{m-1} est un polynôme de degré $m - 1$.

On peut construire différentes méthodes de Krylov en fonction des contraintes et des conditions d'optimalités que l'on associe à la solution recherchée. Une première classe de méthode consiste à définir $L_m = AK_m(A, r_0)$. Les méthodes du gradient conjugué et du GMRES appartiennent à cette classe de méthodes. La méthode du gradient conjugé consiste à rechercher la solution de systèmes symétriques définis positifs dans un espace de Krylov tel que le résidu associé soit orthogonal à l'espace. La méthode du GMRES (Generalized Minimal Residual Method) est une méthode de Krylov qui fonctionne quant à elle pour des matrices quelconques (non symétriques complexes par exemple). Elle est fondée sur la recherche du vecteur qui minimise la norme euclidienne du résidu. Il est aussi possible de définir $L_m = K_m(A, r_0)$ (méthodes FOM et

ORTHORES) ou $L_m = K_m(A^t, r_0)$ (méthode Bi-CG). Pour plus de détails, on pourra consulter les références [36, 79].

Dans le cas général, les méthodes de Krylov nécessitent le calcul d'une base orthonormale de K_m (GMRES, FOM, ...). Lorsque m devient grand, une itération nécessite beaucoup de calculs pour construire un nouveau vecteur de la base de projection. De plus, le stockage mémoire des vecteurs de la base peut devenir prohibitif. Il est possible de resteindre le nombre de vecteurs de la base de projection en réinitialisant le processus itératif avec la dernière approximation de x que l'on a calculée. On n'assure alors que la diminution du résidu entre chaque réinitialisation et l'utilisation d'une base trop petite peut être très pénalisante pour la convergence. Certaines méthodes comme le gradient conjugué ne nécessitent pas de stockage de la base de projection.

1.3.2 Technique de préconditionnement

Principe général

Sur un problème donné, la convergence d'une méthode de Krylov est difficilement prévisible. Dans le cas du Gradient Conjugué, elle dépend du nombre de conditionnement $K_p(A) = ||A||_p||A^{-1}||_p$. En effet :

$\frac{||x-\tilde{x}||}{||x||} \leq K(A)\frac{||r||}{||b||}$ où $r = b - A\tilde{x}$, x est la solution exacte et $\tilde{x}$ une solution approchée.

Lorsque la résolution de $A.x = b$ prend trop d'itérations, c'est généralement que la matrice A est mal conditionnée (c'est-à-dire que $K(A)$ est grand). On remplace alors la résolution du système $A.x = b$ par la résolution d'un système préconditionné admettant la même solution.

Etant donné une matrice M inversible d'ordre n, le système

$$M^{-1}Ax = M^{-1}b \tag{1.3}$$

admet la même solution que le système initial. La matrice M est appelée **matrice de préconditionnement** associée au système préconditionné (1.3). Dans ce qui précède, le préconditionnement est effectué à gauche ($M_G^{-1}A.x = b$), mais il peut aussi être réalisé à droite ($A.M_D^{-1}.y = b$ avec $x = M_D^{-1}y$) ou des deux côtés ($M_G^{-1}A.M_D^{-1}.x = M_G^{-1}.b$). Le résidu d'un système préconditionné à droite est identique au résidu du système initial. Dans la suite de ce paragraphe, nous utiliserons les notations d'un préconditionnement à gauche mais le discours s'applique aussi aux autres préconditionneurs.

Reste à choisir M, de manière à accélérer la convergence. Le but est de trouver une matrice M^{-1} telle que $M^{-1}.A$ soit mieux conditionnée que A. Il n'existe pas de solution universelle mais quelques idées directices peuvent guider l'élaboration d'un préconditionneur. Notons par exemple que si $M = A$, la résolution est immédiate car le système devient :

$$A^{-1}Ax = x = A^{-1}b, \tag{1.4}$$

mais le calcul du préconditionneur est dans ce cas aussi difficile que le problème initial. Pour construire un préconditionneur, on peut donc chercher à construire une matrice M proche de A mais plus facile à inverser. Pour les problèmes symétriques définis positifs, le taux de convergence du gradient conjugué dépend de la distribution des valeurs propres de A [79] et le préconditionneur doit améliorer les propriétés spectrales de la matrice (le système préconditionné doit avoir un nombre de préconditionnement plus faible et les valeurs propres doivent être regroupées autour de 1). Pour les matrices non-symétriques, la relation entre convergence et valeurs propres est

plus complexe [37] mais on peut retenir qu'un spectre regroupé induit souvent une convergence rapide, surtout lorsque la matrice préconditionnée devient presque normale.

L'utilisation d'un préconditionneur dans une méthode de Krylov se fait de la manière suivante : on considère le système $A.x = b$ préconditionné par la matrice M :

$$M^{-1}.A.x = M^{-1}.b \tag{1.5}$$

Le résidu initial préconditionné est alors défini par

$$r_0 = M^{-1}(b - Ax_0) \tag{1.6}$$

et la recherche de la solution s'effectue dans l'espace de Krylov

$$K_m = Vec(r_0, M^{-1}.A.r_0, (M^{-1}.A)^2.r_0, \ldots, (M^{-1}.A)^{m-1}.r_0) \tag{1.7}$$

en recherchant la solution dans $u_0 + K_m$ qui minimise le résidu préconditionné.

Dans la pratique, on ne calcule jamais la matrice inverse M^{-1} car les méthodes de Krylov préconditionnées n'ont pas besoin explicitement de cette matrice. Au cours du processus itératif, seul le produit `matrice` $\times$ `vecteur` avec M^{-1} est utilisé. Ainsi, par exemple, pour calculer le $k$ième résidu $r_k = M^{-1}.A.r_{k-1}$, on pourra résoudre $Mr_k = Ar_{k-1}$. Il est donc fondamental que ce produit `matrice` $\times$ `vecteur` ne soit pas trop coûteux à effectuer. Lorsqu'on utilise un préconditionnement par factorisation incomplète, cette opération peut être effectuée par des résolutions triangulaires. Afin d'obtenir un préconditionnement performant, on cherche de plus à ce que le stockage de M ne soit pas rédhibitoire.

D'un point de vue général, un bon préconditionneur doit donc à la fois :
- améliorer la convergence du système préconditionné en diminuant le nombre d'itérations,
- être peu coûteux à calculer et à utiliser : le coût d'initialisation du préconditionneur et le calcul d'une itération doit rester rapide.

Un bon compromis entre la robustesse du préconditionneur et son coût doit donc être trouvé. Ce compromis dépend notamment de la difficulté du problème à résoudre et de la précision souhaitée.

Techniques de préconditionnement

Comme les préconditionneurs jouent un rôle très important dans la convergence des méthodes itératives, les recherches concernant le préconditionnement sont très actives. De nombreuses stratégies de préconditionnement différentes peuvent être appliquées à un système. Un premier exemple de préconditionneur très simple peut être obtenu en utilisant par exemple $M = diag(A)$. Celui-ci est bien sûr très peu robuste. On peut distinguer principalement les classes de préconditionneurs algébriques suivantes :
- Une première classe de préconditionneurs consiste à utiliser une méthode itérative stationnaire, par exemple la méthode SSOR (pour « symmetric SOR »).
- Une seconde classe de méthodes utilise des factorisations incomplètes. Elle est présentée en détails à la section 1.4.
- Une troisième classe se fonde sur le calcul d'inverses approchés creux de A. L'étude de ces préconditionneurs dépasse le cadre de cet état de l'art. L'avantage de ce type de préconditionneurs par rapport aux factorisations incomplètes est d'être plus stable numériquement et plus facile à paralléliser. On pourra se reporter à [8] ou [9] pour plus de détails.

1.4 Préconditionnement par factorisation incomplète

Introduction

Les méthodes de factorisations incomplètes ont été introduites vers 1960 séparément par Buleev [14] et Varga [87]. L'existence théorique des factorisations incomplètes peut être vérifiée pour certaines classes de matrices [64, 63, 72, 88] mais pour des matrices quelconques, les factorisations incomplètes peuvent échouer. Le principe d'une factorisation incomplète (ILU) est de limiter en cours de factorisation le remplissage des facteurs en ignorant une partie des coefficients des facteurs. Une telle factorisation peut être utilisée comme un préconditionneur d'une méthode itérative, sous la forme $M = \widetilde{L}\widetilde{U}$ où $\widetilde{L}$ et $\widetilde{U}$ sont les facteurs approchés d'une factorisation de A. Le but est d'obtenir une bonne approximation de A, au moindre coût en temps et en mémoire.

La robustesse du préconditionneur va dépendre du remplissage admis dans les facteurs et des critères que l'on se fixe pour admettre ou ignorer le remplissage créé par l'élimination de Gauss. Les algorithmes de factorisations incomplètes utilisent en général deux sortes de critères : la position des termes dans la matrice et les valeurs numériques des coefficients. On peut bien sûr combiner ces deux critères. La suite de cette section présente les méthodes de factorisations incomplètes les plus populaires [79] :

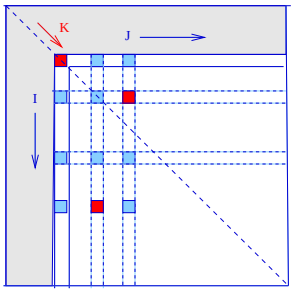
- les factorisations ILU(k) qui utilisent un critère combinatoire pour limiter le remplissage.
- les factorisations ILUT (pour « Incomplete LU Threshold ») qui contrôlent le remplissage en fonction des valeurs numériques des coefficients.

On peut aussi construire des méthodes de préconditionnement multi-niveaux en utilisant une approche récursive de la factorisation ILU. Nous nous intéresserons à ces méthodes dans la section 1.5.5.

1.4.1 Factorisations ILU(k)

Une première méthode de factorisation incomplète consiste à déterminer à l'avance le remplissage autorisé dans la structure des facteurs $\widetilde{L}$ et $\widetilde{U}$. Pour la suite, on note $\mathscr{R}$ l'ensemble des coefficients pour lesquels on autorise du remplissage. La figure 6 présente l'algorithme de factorisation ILU à remplissage prédéfini. Comme pour une factorisation directe, les structures de données peuvent être allouées de manière statique, ce qui permet une implémentation efficace de la factorisation.

ALGORITHME 6 : Factorisation ILU avec remplissage prédéfini



pour $k = 1$ *à* $n-1$ **faire**
  pour $i = k+1$ *à* n *tq* $(i,k) \in \mathscr{R}$ **faire**
    $a_{ik} := a_{ik}/a_{kk}$
    pour $j = k+1$ *à* n *tq* $(i,j) \in \mathscr{R}$ **faire**
      $a_{ij} := a_{ij} - a_{ik} * a_{kj}$

Généralement, on construit $\mathscr{R}$ en se basant uniquement sur le graphe d'adjacence $G(V,E)$ de A (les valeurs numériques ne sont pas prises en compte). Une règle simple pour limiter le remplissage consiste à n'autoriser du remplissage que là où il y en a déjà dans la matrice initiale. Ce type de factorisation incomplète, appelée ILU(0), est très simple à implémenter et très peu coûteuse à calculer. Elle est suffisamment efficace pour un certain nombre de problèmes. Elle

a été initialement proposée pour des problèmes résultant de la discrétisation d'équations aux dérivées partielles elliptiques scalaires d'ordre peu élevé qui produisent des M-matrices et pour des matrices symétriques à diagonale dominante par Meijerink et Van der Vorst [64] en 1977. Par contre, pour des problèmes plus difficiles, ces préconditionneurs sont des approximations trop grossières de A.

En général, on détermine alors un schéma de remplissage à partir du graphe d'adjacence $G(V, E)$ de la matrice creuse. Les méthodes ILU(k) admettent ainsi du remplissage en se basant sur la notion de *niveaux de remplissage*, initialement proposée par Gustafsson [40]. On associe à chaque coefficient de la matrice factorisée $L+U$ un niveau de remplissage $\text{lev}(a_{ij})$ en fonction de la manière dont ce coefficient apparaît pendant la factorisation. On initialise les niveaux de la manière suivante :

- si $a_{ij} \neq 0$ dans A, alors $\text{lev}(a_{ij}) = 0$,
- si $a_{ij} = 0$ dans A, alors $\text{lev}(a_{ij}) = \infty$.

Le niveau de remplissage d'un coefficient a_{ij} est ensuite défini à partir des niveaux de remplissage des termes a_{ik} et a_{kj} qui le modifient dans le processus de factorisation :
$\text{lev}(a_{ij}) = \min(\text{lev}(a_{ij}), \text{lev}(a_{ik}) + \text{lev}(a_{kj})+1)$. Les niveaux de remplissage traduisent ainsi une notion de distance entre les coefficients initiaux de A et les coefficients de remplissage : le niveau de remplissage d'un coefficient est plus faible lorsqu'il provient d'un calcul impliquant des coefficients apparus tôt dans le processus de factorisation. Intuitivement, les termes ayant un niveau de remplissage élevé ont moins d'impact sur d'autres coefficients et sur la qualité de la factorisation. Ainsi, pour les matrices à diagonale dominante, plus le niveau de remplissage est élevé, plus les éléments ont de faibles valeurs. Pour limiter le remplissage, on supprime alors les coefficients en fonction de leur niveau de remplissage. Une factorisation ILU(k) ne retient que les coefficients de niveau inférieur ou égal à k. Pour $k = 0$, on retrouve le préconditionneur ILU(0) évoqué précédemment.

Comme pour une factorisation complète, le remplissage d'une factorisation ILU(k) d'une matrice creuse A symétrique ou à structure symétrique peut être caractérisé en utilisant la notion de chemin dans son graphe d'adjacence (dans le cas des matrices non-symétriques, on considère le graphe d'adjacence symétrique associé à $A + A^t$).

Théorème 3 (Caractérisation du remplissage d'une factorisation ILU(k)) Un terme nul a_{ij} de A deviendra non nul dans la factorisation ILU(k) de A *ssi* il existe un **chemin de longueur au plus** $k+1$ allant du sommet i au sommet j en ne passant que par des sommets de plus petit numéro que i et j dans le graphe $G(V, E)$ d'adjacence de A.

La figure 1.8 illustre cette notion sur une matrice ayant une chaîne pour graphe d'adjacence. Plus le chemin d'élimination entre 2 inconnues x_i et x_j est long, moins le couplage entre ces inconnues est important pour la factorisation et plus le terme a_{ij} peut être ignoré.

Le remplissage des factorisations ILU(k) peut être déterminé à l'avance par une factorisation symbolique. Malheureusement, le calcul d'une factorisation symbolique pour ILU(k) a un coût comparable à celui de la factorisation numérique car on a besoin de connaître la provenance du remplissage. La factorisation symbolique est donc bien plus coûteuse que celle utilisée pour une méthode directe. L'algorithme classique est donné algorithme 7. Hysom et Pothen en ont proposé en 2002 une version parallèle fondée sur la recherche de tous les chemins d'élimination de longueurs $k+1$ dans $G(A)$ et permettant de calculer indépendamment la structure de chaque colonne [51].

Dans le cas d'une discrétisation de type éléments finis où un nœud du maillage représente plusieurs inconnues, le graphe de A peut être quotienté naturellement en numérotant consécutivement les variables d'un même nœud. La matrice a alors une structure par blocs et l'on peut

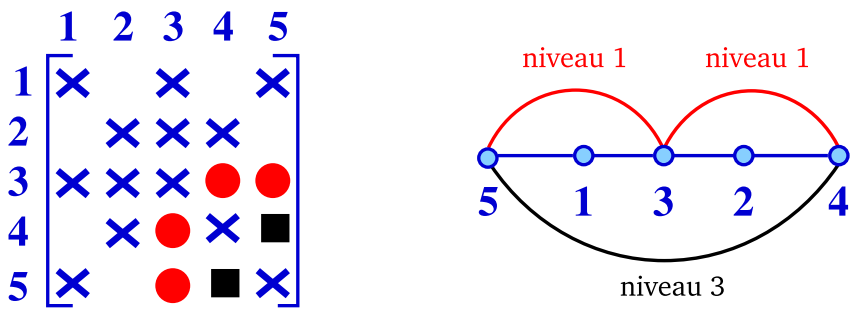


FIG. 1.8 – Factorisation ILU(k) et niveaux de remplissage

réaliser une factorisation par blocs exploitant les routines BLAS de niveau 3. La factorisation symbolique par blocs a alors une complexité fonction du nombre de nœuds du maillage (et non du nombre d'inconnues). Lorsque la structure bloc n'est pas naturelle, elle peut être imposée par une numérotation. On peut aussi utiliser un algorithme d'amalgamation pour admettre des termes de remplissage supplémentaires et améliorer la structure bloc. Des solveurs ILU(k) par blocs denses ont ainsi pu être développés en adaptant des solveurs directs [46].

ALGORITHME 7 : Factorisation symbolique ILU(K)

```
/* Initialisation                                                    */
pour tous les termes de A faire
    si a_ij ≠ 0 alors lev(a_ij) ← 0
    si a_ij = 0 alors lev(a_ij) ← ∞
/* Factorisation symbolique                                          */
pour k = 1 à n − 1 faire
    pour i = k + 1 à n faire
        si lev(a_ik) ≤ p alors
            pour j = k + 1 à n faire
                lev(a_ij) = min(lev(a_ij), lev(a_ik) + lev(a_kj)+1)
                si lev(a_ij) ≤ K alors
                    on ajoute (i, j) dans la structure des non-zéros de L et U
```

Dans de nombreux cas, le préconditionnement ILU(1) est suffisant alors que ILU(0) ne fonctionne pas. Le coût de factorisation symbolique reste faible pour des niveaux k inférieurs à trois et il est souvent peu efficace d'utiliser des ILU(k) à des niveaux de remplissage plus élevés.

1.4.2 Factorisations ILUT(τ)

Les approches basées sur les niveaux de remplissage peuvent ne pas être assez robustes pour certaines classes de problèmes. L'approche ne tient en effet pas compte des valeurs des coefficients et peut en rejeter d'importants. De plus, lorsque les matrices ne sont pas à diagonale dominante, les factorisations ILU(k) contiennent un nombre élevé de petits coefficients qui ont peu d'influence sur la qualité du préconditionneur mais coûtent cher à calculer et stocker. Une autre approche consiste alors à accepter ou rejeter des termes en fonction de leur taille. Ainsi,

on est sûr que seuls les termes contribuant significativement à la qualité du préconditionneur sont conservés.

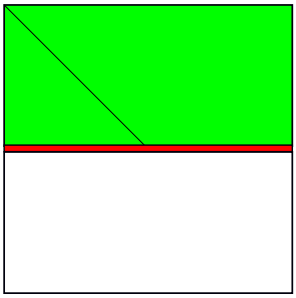
Les factorisations ILUT (pour « Incomplete LU Threshold ») contrôlent le remplissage en fonction des valeurs numériques des coefficients : on ne garde au cours de la factorisation que les termes vérifiant un critère numérique. Généralement, on n'accepte un terme de remplissage que si sa norme est supérieure à une valeur notée τ et appelée valeur de **seuillage numérique**. On utilise en général un critère de seuillage relatif. Par exemple, lors de l'élimination de la colonne i, un nouveau terme de remplissage peut n'être accepté que si la valeur de ce coefficient est supérieur à $\tau||a_{i*}||_2$ où a_{i*} est la $i$ème ligne de A.

L'inconvénient des méthodes ILUT est que le remplissage est déterminé au cours de la factorisation et la structure de la matrice doit donc être mise à jour de manière dynamique. Par contre, cela permet d'employer facilement une stratégie de pivotage. Une factorisation symbolique est ici impossible. Il est aussi difficile de choisir une bonne valeur pour τ. Souvent, on utilise des matrices témoins représentatives de l'application visée pour trouver τ de manière empirique. Généralement, on prend $\tau \in [10^{-4}, 10^{-2}]$ mais les solveurs fondés sur des factorisations ILUT sont difficiles à utiliser comme des boîtes noires.

L'inconvénient de la factorisation ILUT est qu'il est impossible de connaître à l'avance un majorant de la mémoire nécessaire au stockage des facteurs $\widetilde{L}$ et $\widetilde{U}$. Une variante notée ILU(τ,p) utilise un critère d'élimination supplémentaire (noté p) pour limiter le nombre de termes de remplissage autorisés dans chaque ligne des facteurs [77]. L'élimination de Gauss est réalisée par lignes (ou colonnes) pour l'application de ce critère. L'algorithme de factorisation LU par ligne utilisant une boucle IKJ est présenté figure 8. Lors de la factorisation ILU(τ,p) de la $i$ème ligne, on élimine tous les termes inférieurs à $\tau||a_{i*}||_2$ et, dans les termes restants, on ne retient que les p plus grands termes. L'algorithme complet est présenté figure 9. Une variante de cette approche consiste à toujours conserver les termes initiaux de A et à autoriser p termes de remplissage additionnels. Cette approche est efficace lorsque l'on traite des problèmes irréguliers où les termes de A ne sont pas uniformément répartis entre les lignes.

L'avantage de cette approche est qu'elle permet de borner la mémoire nécessaire au stockage des facteurs. La factorisation ILU(τ, p) est aussi facilement implémentable. Par contre, l'emploi du critère p ajoute un paramètre au solveur et ce critère local a des effets assez imprévisibles pour la convergence de la méthode itérative : le remplissage d'une ligne pouvant être plus important à certains endroits de la matrice, de part la numérotation (généralement à la fin de la matrice).

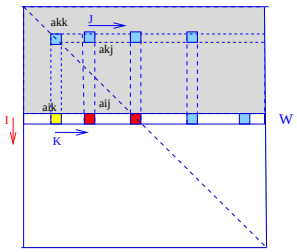
ALGORITHME 8 : Version IKJ de l'algorithme de factorisation LU



Entrées : Matrice $A = (a_{ij})$ régulière de dimension n
Sorties : Matrices L et U telles que $L.U = A$
pour $i = 2$ *à* n **faire**
  pour $k = 1$ *à* $i - 1$ **faire**
    $a_{ik} \leftarrow a_{ik}/a_{kk}$
    pour $j = k + 1$ *à* n **faire**
      $a_{ij} \leftarrow a_{ij} - a_{ik} * a_{kj}$

La boucle j peut s'écrire de manière compacte sous la forme : $a_{i*} \leftarrow a_{i*} - a_{ik} * a_{k*}$

ALGORITHME 9 : Factorisation ILU(τ, p)



pour $i = 1$ *à* n **faire**
  $w \leftarrow a_{i*}$
  pour $k = 1$ *à* $i - 1$ *et si* $w_k \neq 0$ **faire**
    $w_k = w_k / a_{kk}$
    Appliquer seuillage sur w_k (R_1)
    si $w_k \neq 0$ **alors**
      $w \leftarrow w - w_k * a_{k*}$
  Appliquer seuillage sur w (R_2)
  $l_{ij} \leftarrow w_j$ pour $j = 1, ..., i - 1$
  $u_{ij} \leftarrow w_j$ pour $j = i, ..., n$

- R_1 : a_{ik} est remplacé par 0 si $a_{ik} < \tau.||a_{i*}||_2$.
- R_2 : on applique R_1 à tous les éléments de a_{i*}. On ne garde que les p plus grands.

Conclusion

Malgré un coût théorique faible, les algorithmes de factorisations incomplètes ne sont en pratique pas toujours très efficaces d'un point de vue informatique : souvent, ces méthodes s'appuient sur une implémentation scalaire et elles sont plus difficilement réalisables par blocs et en parallèle. On les utilise souvent en tant que préconditionneurs locaux (donc séquentiels) dans des méthodes de type décomposition de domaines (voir section 1.5).

Pour améliorer la qualité des factorisations, on peut les coupler avec des méthodes de prétraitements numériques. En général, la renumérotation utilisée dans la construction de préconditionneurs par factorisation incomplète a un impact sur la convergence des méthodes de Krylov. Une renumérotation peut être utilisée pour réduire le remplissage, améliorer la stabilité numérique ou introduire du parallélisme. Elle peut aussi servir à introduire des blocs pour réaliser des factorisations incomplètes par blocs [4, 15, 18, 19]. Des contraintes de remplissage supplémentaire peuvent être ajoutées pour faciliter le parallélisme.

1.5 Méthodes de décomposition de domaine

Introduction

Les préconditionneurs ILU sont en général difficiles à paralléliser. Un cas particulier notable concerne les factorisations ILU(0) qui peuvent l'être en utilisant un coloriage du graphe d'adjacence [81]. Un coloriage consiste à assigner une couleur à chaque sommet de telle façon que deux sommets adjacents aient toujours des couleurs différentes. Le nombre minimal de couleurs nécessaires au coloriage, appelé nombre chromatique, reflète le parallélisme potentiel de la méthode. Le problème du coloriage est NP-difficile mais des heuristiques sont suffisantes. Des versions distribuées de factorisation ILU(k) utilisant ce principe ont tout de même été proposées [50]. On peut distinguer trois grandes familles d'approches pour construire des préconditionneurs parallèles :

- La première consiste à paralléliser le calcul matriciel. On peut classer dans cette famille les ILU parallèles et les méthodes fondées sur le calcul d'inverses approchées [7].
- Une deuxième approche utilise des techniques de décompositions de domaine. C'est le cas des méthodes de Schwarz [90, 11] ou des méthodes utilisant la technique du complément de Schur.

– Enfin, les méthodes multi-niveaux sont basées sur un parallélisme de type multi-grilles [86]
On s'intéresse ici à la seconde approche, qui consiste à diviser un domaine de calcul en sous-domaines pour introduire du parallélisme dans les préconditionneurs ILU.

1.5.1 Techniques de décomposition de domaine

Les méthodes de décomposition de domaine (ou de sous-structuration) sont un ensemble de techniques pour résoudre des équations aux dérivées partielles (PDEs) en s'appuyant sur le principe *diviser pour régner*. Il s'agit de décomposer le domaine de calcul en sous-domaines pour faciliter la résolution du problème global. La décomposition peut être guidée par différentes contraintes et objectifs. Elle peut initialement tenir compte notamment :

– du couplage de modèles physiques différents (en séparant les régions régies par des équations différentes),
– du maillage, s'il est obtenu par la réunion de maillages réalisés indépendamment (avec des logiciels différents par exemple).

Pour les problèmes linéaires, une décomposition de domaine peut être utilisée comme technique de préconditionnement pour une méthode de Krylov. On peut utiliser la résolution des systèmes linéaires associés aux sous-domaines (plus petits) pour construire un préconditionneur pour la résolution du problème initial. Dans un contexte parallèle, on peut aussi se servir d'une décomposition pour répartir les données sur une machine à mémoire distribuée en assignant une partie du problème à chaque processeur. Il est alors intéressant de réaliser un équilibrage de charge, pour répartir à la fois les besoins en mémoire et en calcul. Il faut bien sûr choisir un bon partitionnement du graphe, respectant une localité des données et de leurs interactions, mais aussi la manière de traiter les *interfaces* entre les sous-domaines (interaction entre les sous-domaines).

Considérons le domaine de calcul $\Omega = \Omega_1 \cup \Omega_2$ de la figure 1.9. On note $\delta\Omega$ la frontière de Ω. On considère que les domaines Ω, Ω_1 et Ω_2 ne contiennent pas leurs frontières et on note $\bar{\Omega} = \Omega \cup \delta\Omega$ la réunion de l'intérieur d'un domaine et de sa frontière. Les frontières artificielles Γ_i sont définies comme les parties de Ω_i qui sont intérieures à Ω. Ainsi, $\delta\Omega_i \backslash \Gamma_i$ correspond à la frontière du sous-domaine ($\delta\Omega_i$) qui n'est pas une frontière intérieure (Γ_i).

Rappelons que les systèmes à résoudre sont issus de problèmes physiques. Souvent, un problème a été discrétisé sur un maillage et les inconnues du système à résoudre sont des nœuds de ce maillage. Les termes de la matrice représentent les liens entre ces nœuds. La représentation en graphe d'un problème est alors naturelle. La méthode de décomposition par domaines partitionne ce maillage en sous-domaines. En fonction du problème considéré, ce partitionnement peut être réalisé sur les éléments, les sommets ou les arêtes. Lorsque $\Omega_1 \cap \Omega_2 \neq \varnothing$, on dit que la décomposition est sans recouvrement.

1.5.2 Méthodes de Schwarz

Les méthodes de décomposition de domaine avec recouvrements des sous-domaines sont souvent appelées méthodes de Schwarz en référence au mathématicien suisse qui utilisa cette idée en 1870 pour prouver l'existence d'une solution à un problème elliptique sur un domaine à la géométrie complexe, formée par la superposition de domaines plus simples où les solutions sont connues [83]. Cette technique, appelée méthode de Schwarz alternée, a connu un regain d'intérêt avec l'avènement du calcul parallèle.

L'idée générale des méthodes de Schwarz est de diviser un problème initial en sous-problèmes et de résoudre chacun des sous-problèmes en les considérant indépendants les uns des autres.

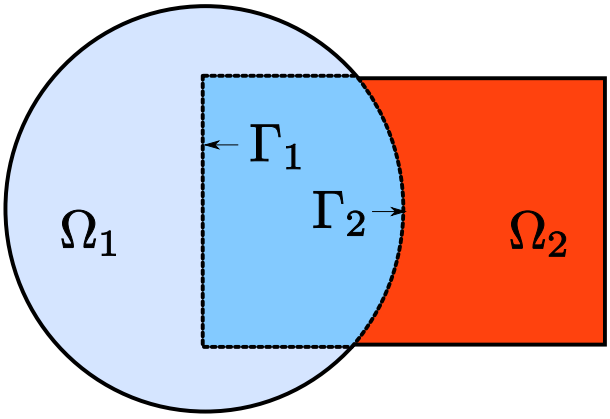


FIG. 1.9 – Décomposition d'un domaine en deux sous-domaines

La superposition de la solution des sous-problèmes donne une approximation de la solution du problème global. Plus formellement, on utilise les résolutions locales pour former un préconditionneur pour la résolution du problème initial. La méthode itérative va permettre de « récupérer » la précision que l'on a perdue en déconnectant les sous-domaines. On appelle solveur local la méthode utilisée pour résoudre les sous-systèmes indépendants. Ce solveur local peut être une méthode directe ou une méthode itérative quelconque.

On considère ici qu'il est possible d'utiliser des grilles dont les maillages correspondent dans la zone de recouvrement. Dans cette section, on présente deux variantes des méthodes de Schwarz qui diffèrent par la manière dont les solutions locales sont exploitées lors de l'application du préconditionneur : la méthode additive permet d'effectuer les calculs des contributions de chaque sous-domaine simultanément alors que la méthode multiplicative présente des dépendances dans le calcul entre sous-domaines. La variante multiplicative est équivalente à un Gauss-Seidel par blocs alors que la variante additive est similaire à un Jacobi par blocs. Plus de détails sont donnés dans le livre [84].

Méthode de Schwarz multiplicative

Considérons que l'on a discrétisé un problème sur l'ensemble de la grille sous la forme du système linéaire $Ax = b$ où x est le vecteur dont les coefficents représentent les valeurs discrétisées de la solution aux sommets du maillage et b est le second membre du système. On note A_{Ω_i} la matrice locale correspondant au couplage entre les nœuds intérieurs au sous-domaine Ω_i. Les méthodes de Schwarz consistent à superposer la solution de domaines plus simples où les solutions sont connues.

Par formalisme, on définit R_i comme la matrice rectangulaire de l'opérateur de restriction dont l'application retourne le vecteur des coefficients de l'intérieur du domaine Ω_i ($x_{\Omega_i} = R_i x$). R_i a donc pour valeur de coefficients un ou zéro. Par exemple, en renumérotant les inconnues tel que $x = (x_{\Omega_1 \setminus \bar{\Omega}_2}, x_{\Gamma_2}, x_{\Omega_1 \cap \Omega_2}, x_{\Gamma_1}, x_{\Omega_2 \setminus \bar{\Omega}_1})^t$ et en notant I la matrice identité, on a : $R_1 = (I\, 0)$ et $R_2 = (0\, I)$. L'algorithme multiplicatif de Schwarz pour deux domaines est donné algorithme 10. Il est présenté sous la forme de l'opération matrice-vecteur $x \leftarrow M.y$ correspondant à l'application

du préconditionneur dans une méthode de Krylov.

Algorithme 10 : Méthode de Schwarz multiplicative appliquée à deux sous-domaines	
$x \leftarrow M.y \Leftrightarrow$	$x_{\Omega_1} \leftarrow A_{\Omega_1}^{-1}\, y_{\Omega_1}\ ;\ x_{\Omega_2} \leftarrow 0$ $x_{\Omega_2} \leftarrow A_{\Omega_2}^{-1}\, R_2\,(y - Ax)$

On définit B_i par $B_i = R_i^t.A_{\Omega_i}^{-1}.R_i$. L'application de la matrice B_i restreint le résidu au sous-domaine i, résout le problème sur ce sous-domaine pour générer une correction de l'erreur et étend cette correction au domaine entier. Avec ce formalisme, on peut alors écrire simplement l'algorithme 10 pour p sous-domaines (algorithme 11). En pratique, les matrices R_i et B_i ne sont jamais formées de manière explicite.

Algorithme 11 : Méthode de Schwarz multiplicative appliquée à p domaines	
$x \leftarrow M.y \Leftrightarrow$	$x \leftarrow B_1 y$ **pour** $i = 2$ *à* p **faire** $\quad x \leftarrow x + B_i(y - Ax)$

Cette version de la méthode de Schwarz est séquentielle, à cause des dépendances directes entre les boucles de calculs successives. En pratique, sur une décomposition à multiple sous-domaines, il existe des sous-domaines qui ne se recouvrent pas. On peut alors définir un coloriage des sous-domaines en associant à chaque sous-domaine Ω_i une couleur c_i telle que si $\Omega_i \cup \Omega_i \neq \varnothing$, alors $c_i \neq c_j$. On obtient ainsi une partition des sous-domaines et ceux de la même couleur, indépendants entre eux, peuvent être traités en parallèle. Le parallélisme potentiel est d'autant plus important que le nombre q de couleurs obtenues est faible. L'algorithme modifié est présenté algorithme 12.

Algorithme 12 : Méthode de Schwarz multiplicative avec coloriage	
$x \leftarrow M.y \Leftrightarrow$	$x \leftarrow \sum_{j \in c_1} B_j y$ **pour** $i = 2$ *à* q **faire** $\quad x \leftarrow x + \sum_{j \in c_i} B_j(y - Ax)$

Méthode de Schwarz additive

La variante additive de la méthode de Schwarz consiste à effectuer le calcul sur chaque sous-domaine en ignorant les dépendances de calculs entre les sous-domaines, pour obtenir un parallélisme plus important. Le préconditonneur devient simplement $B = \sum B_i$ et on obtient alors l'algorithme présenté figure 13. En pratique, la méthode de Schwarz additive est souvent utilisée pour introduire du parallélisme dans les méthodes de résolution en raison de sa facilité de mise en œuvre. Lorsque le solveur local est un solveur direct, la méthode de Schwarz peut être vue comme une première manière de construire un solveur hybride direct-itératif.

Comme la méthode de Jacobi par blocs, la méthode de Schwarz additive est une méthode très parallèle mais en contrepartie, il est souvent nécessaire d'effectuer un grand nombre d'itérations pour converger. Il est parfois possible d'utiliser une grille grossière pour limiter l'impact de l'augmentation du nombre de sous-domaines sur le nombre d'itérations comme dans une méthode multi-niveaux (voir [84]). La grille grossière sert alors à récupérer l'information que l'on perd en interdisant du remplissage entre les sous-domaines.

Algorithme 13 : Méthode de Schwarz additive	
$x \leftarrow M.y \Leftrightarrow$	$v \leftarrow \sum_j B_j u$

1.5.3 Technique du complément de Schur

Soit la partition des inconnues du système en deux sous-ensembles P_B et P_C. Le système $A.x = b$ peut s'écrire :

$$A \begin{pmatrix} x_B \\ x_C \end{pmatrix} = \begin{pmatrix} b_B \\ b_C \end{pmatrix} \text{ où } A = \begin{pmatrix} B & F \\ E & C \end{pmatrix} \tag{1.8}$$

Considérons maintenant l'élimination de Gauss par blocs avec les notations de l'équation (1.8). On peut exprimer x_B à partir de la première équation, sous la forme

$$x_B = B^{-1}.(b_B - F.x_C) \tag{1.9}$$

En substituant x_B dans la seconde équation, on obtient :

$$(C - E.B^{-1}.F)x_C = b_C - E.B^{-1}.b_B \tag{1.10}$$

La matrice

$$S = C - E.B^{-1}.F \tag{1.11}$$

est appelée matrice du complément de Schur. Si on résout le système linéaire associé (1.10), on obtient x_C et on peut alors obtenir x_B par l'équation (1.9). Une méthode de résolution en quatre étapes peut être déduite de cette approche :

1. Calculer $b_C - E.B^{-1}b_B$ en utilisant l'équation (1.10),
2. Former la matrice du complément de Schur grâce à l'équation (1.11),
3. Résoudre le système réduit associé défini par l'équation (1.10),
4. Obtenir les inconnues intérieures aux sous-domaines par l'équation (1.9).

Cette approche de la résolution du système a de nombreuses applications. Elle peut notamment être utilisée pour :

- construire des méthodes multi-niveaux,
- réduire la dimension du problème à résoudre (pour itérer sur un système plus petit),
- isoler une partie de la matrice,
- construire des factorisations out-of-core.

La technique du complément de Schur peut aussi être utilisée pour coupler différentes techniques de résolution. Si l'on note $\widetilde{B}^{-1}$ et $\widetilde{S}^{-1}$ des inverses (exacts ou approchés) pour les matrices B et S, alors on peut définir un préconditionneur M sous la forme :

$$x \leftarrow M.y \Leftrightarrow \begin{cases} x_C \leftarrow \widetilde{S}^{-1}(y_C - E.\widetilde{B}^{-1}y_B) \\ x_B \leftarrow \widetilde{B}^{-1}(y_B - F.x_C) \end{cases} \tag{1.12}$$

Le choix de $\widetilde{S}^{-1}$ et de $\widetilde{B}^{-1}$ permet de décliner plusieurs variantes. Par exemple, on peut utiliser $\widetilde{S}^{-1} = U_S^{-1}.L_S^{-1}$ et $\widetilde{B}^{-1} = U^{-1}.L^{-1}$ mais on peut aussi utiliser une méthode itérative pour résoudre $S.x = b_B - E.x_B$ en la préconditionnant à l'aide de L_S et U_S. Cette remarque peut s'appliquer également au sous-système induit par B.

1.5.4 Utilisation du complément de Schur dans une technique de décomposition de domaine

Considérons une partition du domaine de calcul en p sous-domaines. Nous pouvons renuméroter les nœuds sous-domaines par sous-domaines, en finissant la numérotation par les nœuds

de l'interface. Avec les notations précédentes, le vecteur solution x renuméroté est alors de la forme :

$$x = (x_{\bar{\Omega}_1 \backslash \Gamma_i}, \ldots, x_{\bar{\Omega}_p \backslash \Gamma_p}, \cup x_{\Gamma_i})^t$$

Le système linéaire associé a alors la structure par blocs suivante :

$$A = \begin{pmatrix} B_1 & & & & F_1 \\ & B_2 & & & F_2 \\ & & \ddots & & \vdots \\ & & & B_p & F_p \\ E_1 & E_2 & \cdots & E_p & C \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_p \\ y \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \\ g \end{pmatrix} \tag{1.13}$$

Par la suite, on notera ce système de manière plus compacte sous la forme de l'équation (1.8). Ainsi, x_B représente les inconnues de l'intérieur des sous-domaines, x_C les inconnues de l'interface entre les sous-domaines. Les matrices E et F représentent le couplage entre les nœuds des sous-domaines et de l'interface. B est la réunion de p systèmes linéaires indépendants pouvant être résolus en parallèle.

1.5.5 Préconditionneurs utilisant des techniques de décomposition de domaine

Préconditionneur Schwarz additif du complément de Schur

Le choix de $\widetilde{S}^{-1}$ et de $\widetilde{B}^{-1}$ dans l'équation (1.12) permet de coupler des méthodes de résolutions. Dans la thèse [42], une méthode parallèle hybride directe-itérative est ainsi construite à partir d'une décomposition de domaine. L'intérieur des sous-domaines est éliminé par une méthode directe ($\widetilde{B}^{-1}.x = (L_B.U_B)^{-1}.x$) et, pour résoudre le système associé à S, une méthode itérative préconditionnée par un Schwarz additif est utilisée.

Le complément de Schur S peut être formé à partir de la factorisation exacte de B. Il peut s'écrire sous la forme de p matrices locales S_i correspondant au couplage entre les nœuds de l'interface Γ_i du domaine i. Ces matrices peuvent être calculées de manière indépendante et en parallèle pour chaque sous-domaine (l'opération (1.11) est en effet une opération locale aux sous-domaines : $S_i = C_i - E_i B_i^{-1} F_i$).

Les matrices S_i forment alors une décomposition de la matrice S, avec recouvrement : les nœuds de l'interface sont partagés entre plusieurs sous-domaines. Rappelons que la méthode additive de Schwarz consiste à ignorer les dépendances de calculs entre les sous-domaines. Chaque matrice S_i peut être factorisée de manière indépendante et la somme des résolutions peut être utilisée comme préconditionneur pour le système associé à S.

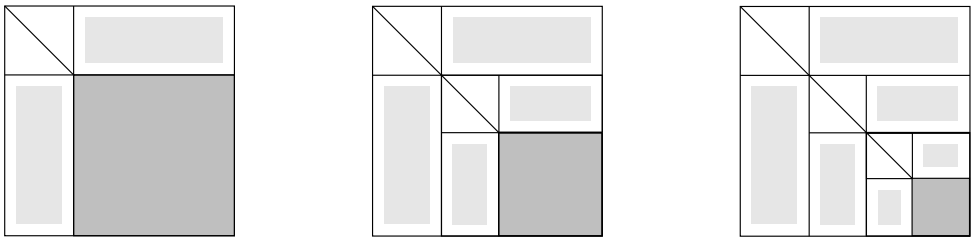
Factorisations ILU multi-niveaux

La technique du complément de Schur permet de construire des factorisations incomplètes fondées sur un schéma de remplissage permettant d'avoir le parallélisme d'une méthode de décomposition de domaine. Les méthodes multi-niveaux consistent à partitionner les inconnues en ensembles indépendants [69, 78, 56]. On appelle ensemble indépendant d'un graphe un sous-ensemble I de sommets deux à deux indépendants (non adjacents). On dit que I est de taille maximum lorsqu'il n'existe pas d'ensembles indépendants plus larges.

Soit I un ensemble indépendant du graphe d'adjacence d'une matrice A. Si on numérote en premier les inconnues associées aux sommets de I, on obtient une matrice de la forme :

$A = \begin{pmatrix} B & F \\ E & C \end{pmatrix}$ où B est une matrice diagonale. Lorsqu'on élimine les inconnues associées à la matrice B, on obtient un système réduit et creux de la forme : $S = C - E.B^{-1}.F$. L'idée des méthodes multi-niveaux est d'éliminer successivement les inconnues associées à des ensembles indépendants d'inconnues. L'algorithme de résolution récursif ILUM est présenté figure 14. Un critère de seuillage numérique est employé pour limiter le remplissage et le dernier niveau de récursion est résolu par n'importe quelle méthode itérative. La méthode peut être vue comme une simplification d'une technique de coloriage d'un graphe telle qu'utilisée notamment pour la parallélisation des méthodes ILU(0) discutée page 23.

ALGORITHME 14 : Algorithme de factorisation ILUM.



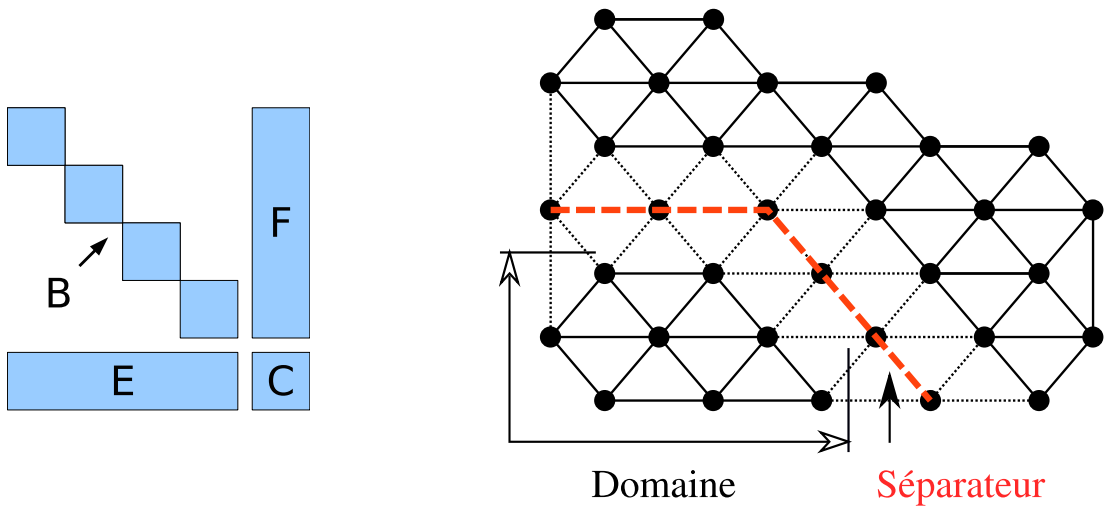
1. Construire un ensemble I de sommets indépendants,
2. Eliminer les inconnues de I et construire la matrice S associée au système réduit,
3. Appliquer un critère de seuillage numérique sur S,
4. Appliquer récursivement l'algorithme à S ou, au dernier niveau, calculer une factorisation incomplète de S

Pour améliorer la robustesse des méthodes ILUM, une approche proposée dans [58, 82] généralise l'idée d'ensembles indépendants pour construire une matrice B diagonale par blocs (équation 1.13 et figure 1.10(a)). L'approche consiste à rechercher des ensembles indépendants de sommets fortement connectés entre eux (par exemple des super-nœuds). La technique est implémentée dans le solveur ARMS (Algebraic Recursive Multilevel Solver). ARMS construit ces groupes de sommets de manière algébrique à partir du graphe d'adjacence. L'implémentation proposée pour construire la décomposition de domaine utilise une renumérotation du graphe de type Reverse Cuthill-McKee [20]. Cette renumérotation réduit la taille de la bande dans le profil de la matrice. Les sous-domaines sont construits successivement en parcourant le graphe suivant l'ordre de cette numérotation. Des séparateurs sommets sont définis lors de ce parcours pour séparer les nœuds en groupes d'inconnues indépendants. La technique est illustrée figure 1.10(b).

Deux techniques peuvent être utilisées pour paralléliser ce type de factorisation :

- elles peuvent être utilisées comme préconditionneur du solveur itératif local dans une méthode de Schwarz,
- on peut exploiter le parallélisme du premier niveau de récursion et résoudre le premier système réduit par une méthode itérative parallèle, comme un Jacobi par blocs ou un ILU(0) parallèle.

Ces techniques sont implémentées dans la version parallèle de ARMS (pARMS, [58]).



(a) Structure de la matrice

(b) Construction des domaines par un parcours en largeur du graphe d'adjacence

FIG. 1.10 – ARMS : un solveur algébrique récursif utilisant une approche multi-niveaux.

Chapitre 2

Méthode de résolution hybride fondée sur une décomposition de domaine

Sommaire

2.1 Construction d'une factorisation incomplète 32
2.1.1 Motivations de l'approche 32
2.1.2 Rénumérotation et partitionnement des inconnues de l'interface 33
2.1.3 Ordonnancement global de l'élimination du complément de Schur 37
2.1.4 Construction d'une décomposition de domaine 38
2.1.5 Factorisation symbolique 40
2.2 Construction d'une méthode hybride directe-itérative 40
2.2.1 Utilisation du complément de Schur 40
2.2.2 Variantes algorithmiques de résolution 42
2.3 Expérimentations 46
2.3.1 Présentation des cas tests 46
2.3.2 Etude des partitions obtenues sur des matrices irrégulières 47
2.3.3 Etude asymptotique de la convergence de la méthode 48
2.3.4 Etude du remplissage du préconditionneur 49
2.3.5 Etude des variantes algorithmiques 52

Introduction

Ce chapitre décrit la méthode de résolution développée dans le cadre de cette thèse. Rappelons que nous cherchons à concevoir une méthode algébrique robuste, flexible et facilement parallélisable. L'idée générale est de combiner l'efficacité des méthodes directes à une méthode itérative passant mieux à l'échelle sur des problèmes 3D. Nous cherchons de plus à exploiter le parallélisme des méthodes de décomposition en domaines.

Une méthode hybride directe-itérative peut être construite à partir d'une décomposition de domaine, en éliminant de manière directe l'intérieur des sous-domaines et en utilisant une méthode de type Schwarz aux interfaces (voir section 1.5.5, page 28). Dans ce cas, on s'intéresse à la résolution de problèmes locaux indépendants dont la somme est utilisée comme préconditionneur. Ici, nous cherchons à construire un préconditionneur plus robuste (c'est à dire qui se dégrade moins avec l'augmentation du nombre de sous-domaines), sous la forme d'une factorisation incomplète globale qui prenne en compte dans la résolution les contributions entre les sous-domaines (qui sont les chemins d'élimination de la factorisation). Il a été présenté dans le chapitre précédent le principe des méthodes de factorisation multi-niveaux (voir ILUM et pARMS, page 28) qui partitionnent récursivement un domaine de calcul en créant des niveaux d'inconnues. Notre préconditionneur est aussi fondé sur ce principe. Dans notre cas, les niveaux d'inconnues sont issus de travaux présentés dans [47].

Dans un premier temps, nous donnerons donc les principes clefs de la factorisation incomplète décrite dans cet article. Dans un second temps, nous verrons les implications en termes de remplissage de l'utilisation d'une factorisation exacte dans le premier niveau de récursion de la méthode multi-niveaux. Nous présenterons une manière naturelle d'exploiter dans notre approche les techniques de renumérotation et de partitionnement des méthodes directes pour construire de manière algébrique la décomposition de domaine initiale.

La seconde partie de ce chapitre est consacrée au couplage entre résolution directe et itérative. Nous verrons comment tirer parti de la technique du complément de Schur pour utiliser la méthode itérative uniquement sur les inconnues des niveaux supérieurs. Nous étudierons différentes variantes algorithmiques qui peuvent être employées pour résoudre le complément de Schur et qui sont numériquement équivalentes.

Enfin, la troisième partie présente de premiers résultats expérimentaux. Nous vérifierons expérimentalement l'impact de ces variantes en termes d'occupation mémoire et de nombre d'opérations sur des cas tests académiques et des cas tests irréguliers issus de problèmes industriels.

2.1 Construction d'une factorisation incomplète à partir d'une décomposition en domaines

2.1.1 Motivations de l'approche

Le but recherché est d'obtenir une factorisation exhibant un très fort potentiel de parallélisation en réutilisant les idées des méthodes de décomposition de domaine. On peut voir notre approche comme un pont entre des méthodes de résolution classiques en décomposition de domaine telles que les méthodes de Schwarz et les méthodes de préconditionnement utilisant des factorisations incomplètes.

Dans ce qui suit, nous allons décrire les principes d'une factorisation incomplète dont le schéma de remplissage est dit compatible avec celui d'une décomposition en domaines du graphe d'adjacence de la matrice du système à résoudre. Le schéma de remplissage d'une factorisation

incomplète est **compatible** lorsque les termes de remplissage ne sont autorisés qu'entre des inconnues d'un même sous-domaine. Les termes de remplissage se trouvent alors uniquement dans les sous-matrices correspondant aux inconnues assignées à un domaine (on parlera aussi de matrice locale à un domaine). Le but est de construire une factorisation incomplète n'admettant donc qu'un remplissage local, mais utilisant une renumérotation qui conserve au mieux les chemins d'élimination importants. Soit I_{Ω_p} l'ensemble des inconnues appartenant au sous-domaine Ω_p. On peut définir formellement l'ensemble des arêtes autorisées dans le graphe d'élimination associé à une factorisation incomplète de A compatible avec la décomposition de Ω en m domaines $\Omega_{p,\,1\leq p\leq m}$ comme étant l'ensemble des arêtes (i,j) telles que :

$$(i,j) \in G^*(A) \Leftarrow \exists p \text{ tq } (i,j) \in I_{\Omega_p} \times I_{\Omega_p}$$

La figure 2.1 illustre la notion de remplissage *compatible*. Le terme de remplissage (i,j) est autorisé alors que (i,j') est interdit.

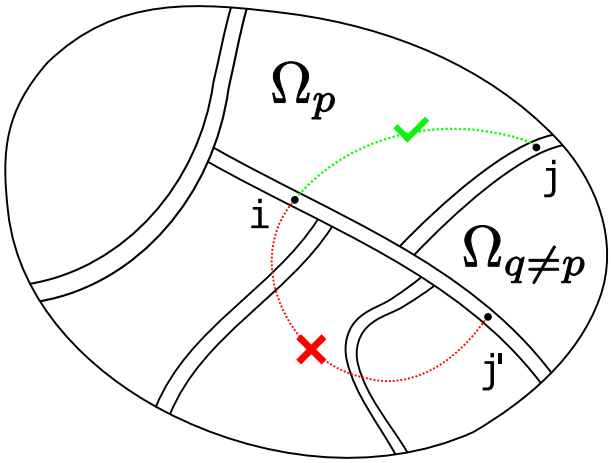


FIG. 2.1 – Notion de remplissage compatible

2.1.2 Rénumérotation et partitionnement des inconnues de l'interface

Cette section présente la manière dont les nœuds de l'interface entre les domaines sont partitionnés. Ce partitionnement, nommé **décomposition hiérarchique de l'interface** dans [47], nous permet de construire une numérotation *globale* des inconnues de l'interface. Ce partitionnement va nous permettre de définir les règles de remplissage *compatibles*. Cette numérotation permet aussi de construire une factorisation incomplète ayant un bon degré de parallélisme et robuste numériquement.

La décomposition hiérarchique regroupe les inconnues de l'interface en sous-graphes appelés **connecteurs**, eux-mêmes organisés en niveaux. Afin d'illustrer la démarche, intéressons-nous à la décomposition hiérarchique que l'on peut obtenir dans le cas d'une grille régulière d'une manière géométrique. La figure 2.2 présente l'exemple d'une grille 8×8 décomposée en neuf sous-domaines avec une zone commune de recouvrement entre les domaines couplés. Les nœuds des sous-domaines peuvent être de trois types : les nœuds intérieurs (niveau 0 de la hiérarchie), les nœuds de l'interface communs à deux domaines (niveau 1) ou communs à quatre domaines (niveau 2). Ces niveaux respectent la définition suivante :

Définition 5 (Décomposition hiérarchique) Les niveaux hiérarchiques sont définis comme un ensemble de sous-graphes (appelés connecteurs) répondant aux propriétés suivantes :

1. Les connecteurs sont disjoints et forment une partition du graphe.
2. Deux connecteurs d'un même niveau ne sont pas couplés.
3. Chaque connecteur d'un niveau l donné est un séparateur pour au moins deux connecteurs de niveau $l-1$.

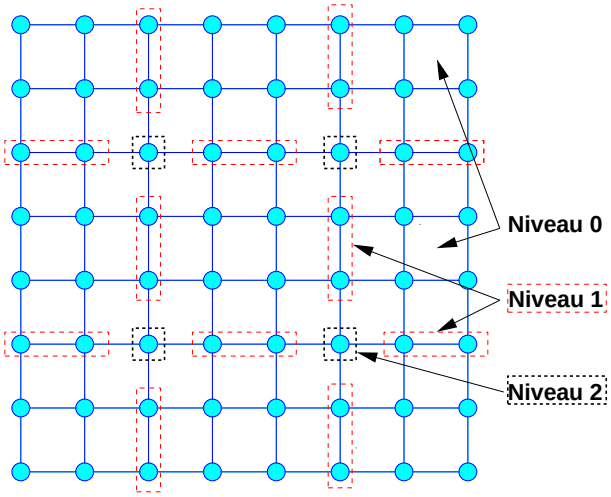


FIG. 2.2 – Grille 8×8 décomposée en neuf sous-domaines et structure hiérarchique associée.

Les propriétés de la définition 5 peuvent être exploitées dans un contexte parallèle : la propriété (2) garantit une indépendance des calculs entre les connecteurs d'un même niveau et la propriété (3) donne l'ordre des calculs à effectuer. On peut ainsi renuméroter les inconnues de l'interface en commençant par celle d'un même connecteur du niveau 1, en continuant avec les autres connecteurs du même niveau avant de répéter l'opération sur les niveaux supérieurs. On obtient alors une matrice dont le motif de remplissage est présenté figure 2.3.

L'article [47] propose un algorithme pour construire une décomposition hiérarchique respectant les propriétés de la définition 5 dans le cas de graphes irréguliers et ce, d'une manière purement algébrique. Sur ce type de graphe, on peut aussi obtenir une décomposition hiérarchique en regroupant les nœuds de l'interface en niveaux, en fonction du nombre de domaines auxquels ils appartiennent. Mais pour assurer le rôle de séparateur des niveaux (propriété (3) de la définition 5), des nœuds doivent alors être déplacés dans des connecteurs de niveaux plus élevés. Ce phénomène est illustré figure 2.4(a). Sur ce graphe, des connecteurs du niveau 2 ont été élargis pour assurer la séparation entre des connecteurs de niveau 1. Trouver le nombre minimal de nœuds devant être déplacés est un problème NP-Complet [47] et l'algorithme utilise donc pour cela une heuristique.

Deux schémas de remplissage fondés sur la décomposition hiérarchique du graphe d'adjacence

Avec les notations de la section 1.5.3, on a :

$$A = \begin{pmatrix} B & F \\ E & C \end{pmatrix}$$

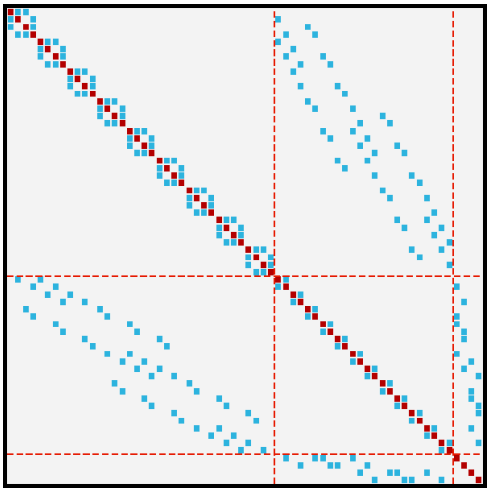
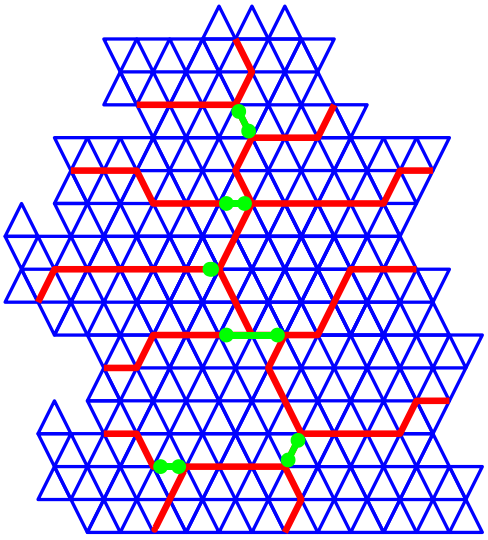
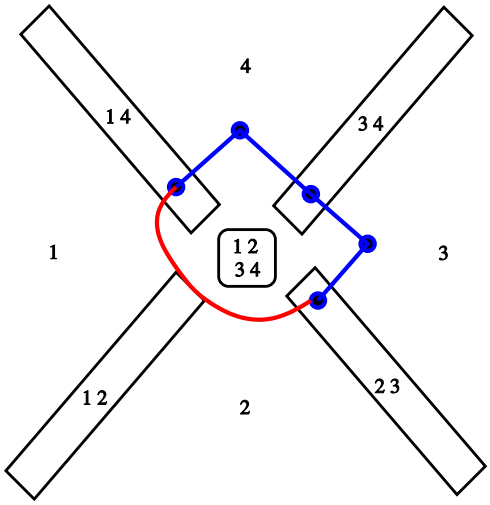


FIG. 2.3 – Matrice associée à la grille de la figure 2.2 après renumérotation.



(a) Cas des graphes irréguliers



(b) Exemple d'interface entre quatre sous-domaines et chemins d'élimination

FIG. 2.4 – Décomposition hiérarchique de l'interface

avec B la matrice de l'intérieur des sous-domaines et C la matrice de l'interface.

On cherche à construire une factorisation incomplète de la matrice S du complément de Schur qui utilise un critère combinatoire pour limiter le remplissage des facteurs L_S et U_S. L'utilisation d'un critère combinatoire permet de prévoir le motif du remplissage des facteurs de manière statique. Cette information peut aussi être utile pour réaliser un équilibrage de charge entre processeurs. Une stratégie de remplissage doit interdire certains chemins d'éliminations qui apparaîtraient dans une factorisation directe de S.

La décomposition hiérarchique de l'interface induit une structure par bloc de C qui est présentée sur une décomposition de domaine simple en figure 2.5. Les termes extra-diagonaux de la matrice C correspondent aux couplages entre les connecteurs de l'interface et il n'y a par

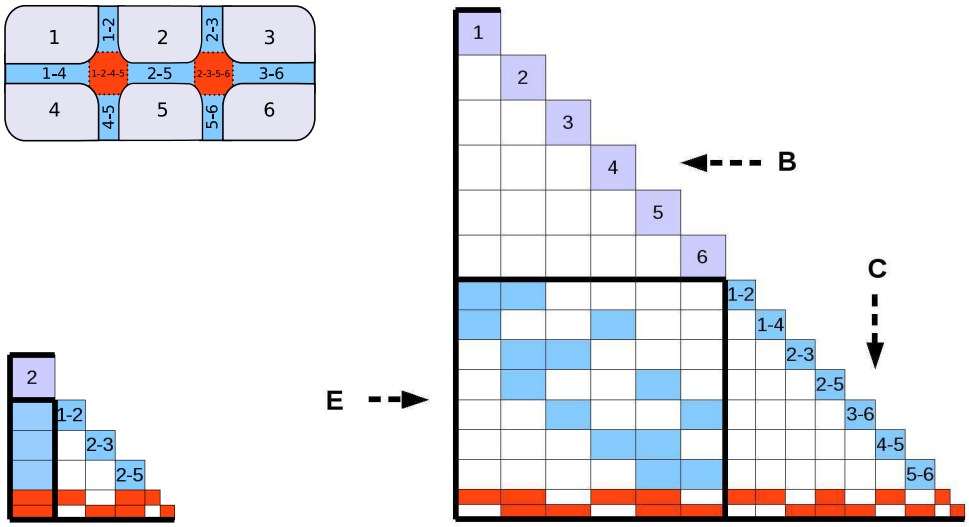


FIG. 2.5 – Structure initiale de la matrice A. Correspond au remplissage autorisé dans la stratégie $\mathscr{R}_C$. Le premier et le second niveau de la structure hiérarchique sont respectivement représentés en bleu et en rouge.

construction aucune arête entre des connecteurs d'un même niveau. Une première règle pour limiter le remplissage des facteurs L_S et U_S consiste à n'autoriser du remplissage que dans la structure par blocs de la matrice C obtenue par la partition des inconnues en connecteurs. On notera par la suite ce motif de remplissage $\mathscr{R}_C$. La structure peut être exploitée pour réaliser une élimination parallèle niveau par niveau.

Si l'on considère le graphe de A quotienté par rapport à la partition hiérarchique, alors ce motif de remplissage équivaut à n'admettre aucun terme de remplissage en dehors de blocs correspondant aux arêtes de ce graphe quotient.

Une seconde règle de remplissage, moins restrictive, peut être obtenue en ne limitant le remplissage des facteurs L_S et U_S qu'au remplissage de S. Cela revient, comme nous l'avons vu dans le paragraphe 2.1.1, à autoriser tout remplissage qui est dans les sous-matrices correspondant aux sous-domaines. On note $\mathscr{R}_S$ ce motif de remplissage. La factorisation de l'intérieur d'un sous-domaine ne crée des termes de remplissage que sur l'interface commune avec les domaines voisins car la numérotation de l'interface joue le rôle d'un séparateur entre les domaines. Le remplissage de S est illustré figure 2.6. Lorsque les propriétés de la décomposition hiérarchique sont respectées (définition 5), le remplissage hors des matrices locales correspond à des chemins d'élimination de longueur au moins 4. Cette propriété est illustrée figure 2.4(b). Ainsi, le remplissage autorisé dans une factorisation ILU(k) pour $k \geq 3$ est inclus dans les matrices locales aux sous-domaines et le schéma de remplissage d'une factorisation ILU(k) pour $k \geq 3$ est inclus dans $\mathscr{R}_S$. Cette propriété est démontrée dans [47].

Les deux motifs de remplissage $\mathscr{R}_C$ et $\mathscr{R}_S$ peuvent s'exprimer comme des règles de remplissage entre connecteurs :

$\mathscr{R}_C$: On n'autorise aucun remplissage entre deux connecteurs qui appartiennent à la zone de recouvrement d'un même sous-domaine.

$\mathscr{R}_S$: On n'autorise du remplissage qu'entre des connecteurs adjacents au même sous-domaine.

Dans les deux cas, le remplissage autorisé dans les facteurs est local aux sous-domaines. Il n'y

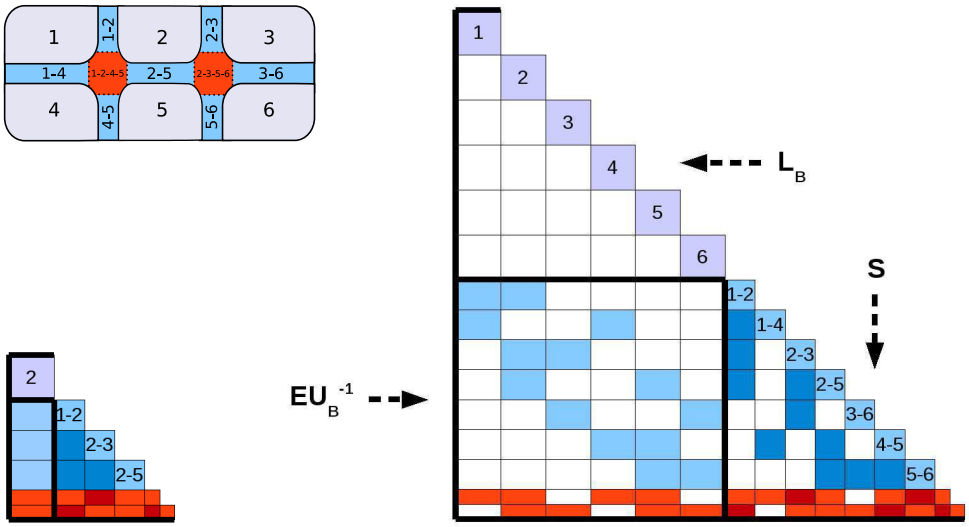


FIG. 2.6 – Structure de la matrice A avant la factorisation de S. Correspond au remplissage autorisé dans la stratégie $\mathscr{R}_S$. Le remplissage des blocs bleu foncé (niveau 1) et rouge foncé (niveau 2) n'est autorisé que dans cette stratégie.

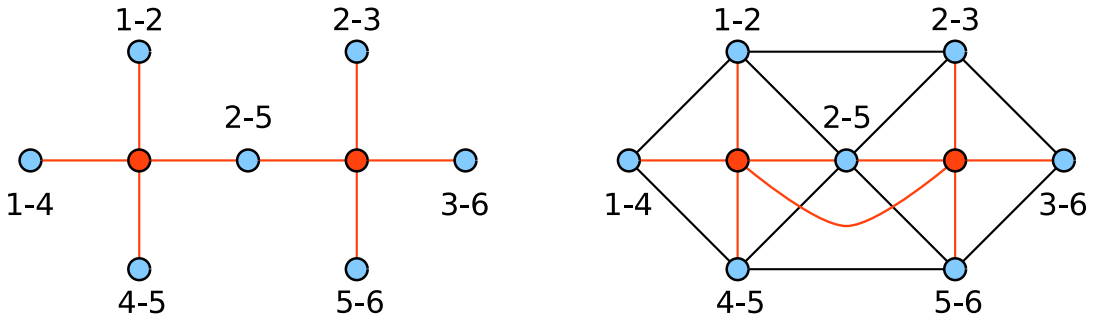
a donc pas de problème de stockage de remplissage non local des facteurs et le schéma de communication de la factorisation ne fait intervenir que le voisinage des sous-domaines.

2.1.3 Ordonnancement global de l'élimination du complément de Schur

Le remplissage $\mathscr{R}_C$ des facteurs L_S et U_S offre un fort potentiel de parallélisation. Les connecteurs d'un même niveau ne sont pas couplés et ils peuvent donc être éliminés simultanément. La renumérotation et le remplissage n'imposent qu'une élimination niveau par niveau des connecteurs. Le motif de remplissage $\mathscr{R}_S$ est moins restrictif et permet de calculer une meilleure approximation des facteurs L_S et U_S. On autorise des termes de remplissage entre les connecteurs d'un même niveau et une dépendance est introduite dans l'élimination de connecteurs d'un même niveau. En effet, le graphe d'élimination quotient montre que par exemple, le calcul du connecteur X dépend de celui de Y, alors que dans le cas de $\mathscr{R}_C$, ils sont indépendants. Un ordonnancement global de l'élimination des connecteurs d'un même niveau est alors nécessaire.

Les connecteurs qui ne sont pas adjacents à un même sous-domaine peuvent être éliminés en parallèle car la règle $\mathscr{R}_S$ n'autorise aucun remplissage entre eux (par définition, les nœuds de tels connecteurs n'appartiennent pas à un sous-domaine commun). On représente les dépendances entre les connecteurs d'un même niveau par un graphe de dépendance (figure 2.7). Les nœuds de ce graphe sont les connecteurs et il y a une arête entre deux connecteurs que s'ils sont adjacents à un même sous-domaine. Un ordonnancement global de l'élimination des connecteurs d'un niveau peut être obtenu par des recherches successives d'ensembles indépendants de taille maximum dans ce graphe. Un exemple d'ordre d'élimination des connecteurs d'une grille 2D est présenté figure 2.8.

Le partitionnement de l'interface ne modifie pas la renumérotation des nœuds internes aux sous-domaines et laisse libre de choisir la renumérotation des nœuds à l'intérieur de chaque connecteur. La section suivante présente la renumérotation de l'intérieur des sous-domaines que nous utilisons.



(a) $\mathscr{R}_C$ (b) $\mathscr{R}_S$

FIG. 2.7 – Graphe d'adjacence des connecteurs après l'élimination de l'intérieur des sous-domaines. Seuls les niveaux 1 et 2 sont représentés.

2.1.4 Construction d'une décomposition de domaine

La renumérotation joue un rôle très important dans le calcul d'une factorisation incomplète. Elle a une grande influence sur la qualité du préconditionneur. Les renumérotations des méthodes directes minimisent le remplissage, permettent la construction d'une élimination par blocs et favorisent le parallélisme. Ces remarques motivent la construction d'une décomposition de domaine qui utilise un tel ordonnancement global des inconnues.

Nous cherchons ainsi à construire une décomposition de domaine de manière algébrique, en utilisant comme point de départ la rénumérotation d'une méthode directe obtenue grâce à un partitionneur de graphe. Le haut de l'arbre d'élimination par blocs ainsi obtenu est proche de l'arbre des séparateurs induit par une dissection emboîtée. On utilise alors cet arbre pour construire la décomposition de domaine. On va partitionner les super-nœuds du système en deux sous-ensembles de nœuds P_B et P_C, en créant une séparation dans l'arbre d'élimination par blocs (voir figure 2.9). Cette séparation est telle que pour tout super-nœud X de P_B, les fils de X sont aussi dans P_B. La renumérotation induite par P_B et P_C de la matrice A, sous la forme $\begin{pmatrix} B & F \\ E & C \end{pmatrix}$, respecte alors les dépendances de calculs exhibées par l'arbre d'élimination par blocs. Les sous-arbres enracinés au niveau du séparateur constituent l'intérieur des sous-domaines (c'est l'ensemble P_B). Ils sont indépendants entre eux. Les super-nœuds du haut de l'arbre correspondent à l'interface entre les domaines (c'est l'ensemble P_C).

Le choix de la séparation dans l'arbre d'élimination par blocs définit la proportion respective de résolution directe (sur B) et de résolution incomplète (sur C) utilisée. La définition du séparateur permet donc d'adapter le préconditionneur à la difficulté du problème. Un préconditionneur plus robuste est obtenu en augmentant la taille des domaines et donc la taille de B.

L'algorithme construisant le séparateur est décrit par l'algorithme 15. L'algorithme prend en paramètre une taille de domaine cible N_{cible} (en nombre d'inconnues) et construit un séparateur de sorte que la taille moyenne des sous-domaines obtenus soit proche de N_{cible}. On note Sep l'ensemble des nœuds de l'arbre formant le séparateur, et `nSousArbre`(v) la taille d'un sous-arbre enraciné au nœud v. Sep est construit de manière récursive. Le séparateur initial est constitué par la racine r de l'arbre ou, lorsque le système contient des sous-systèmes non connectés, par les racines de chaque arbre d'élimination. Le séparateur est raffiné récursivement en remplaçant chaque nœud de Sep par ses fils lorsque cela produit des sous-arbres dont la taille est plus proche de N_{cible} que l'arbre initial. L'algorithme consiste donc à partir du haut de l'arbre d'élimination

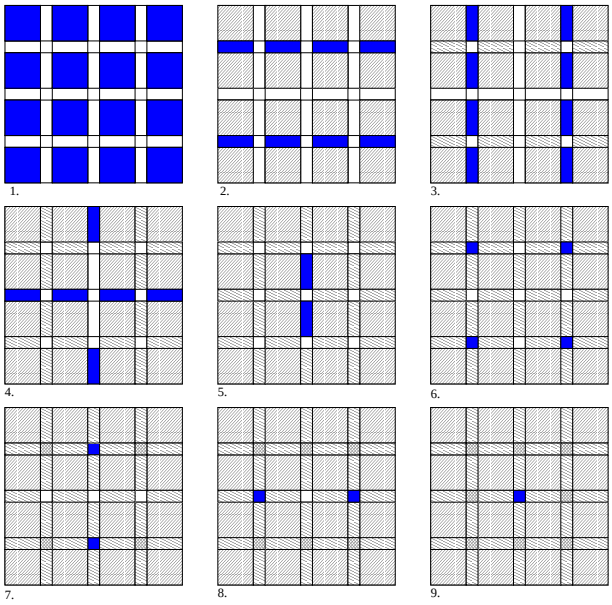


FIG. 2.8 – Ordonnancement global de l'élimination des connecteurs pour une stratégie $\mathscr{R}_S$. Les connecteurs qui sont éliminés en parallèle à chaque étape sont représentés en bleu. Après élimination, ils sont marqués comme ayant été calculés (hachures) et, à l'étape suivante, on recherche un ensemble indépendant de connecteurs de taille maximum dans les connecteurs restant (ils sont représentés en blanc).

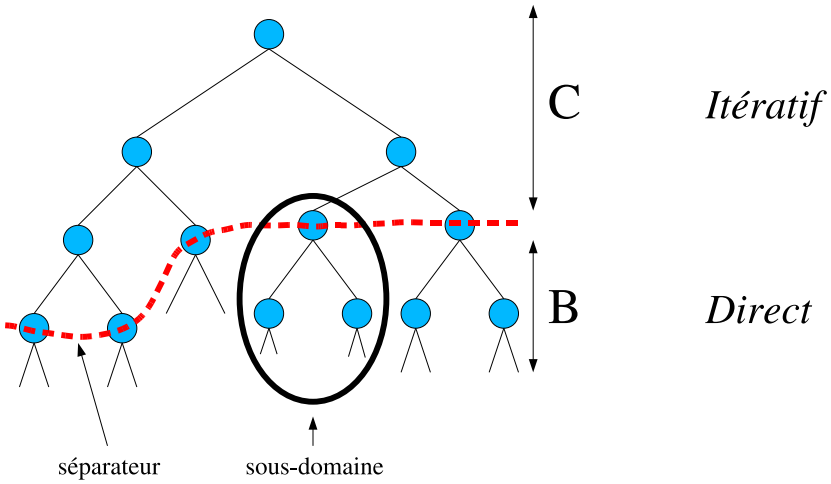


FIG. 2.9 – Découpage du système et application de la méthode du complément de Schur à partir d'une numérotation du direct.

par blocs et à descendre le séparateur dans l'arbre. Le critère utilisé pour optimisation de la taille des sous-domaines est local et l'algorithme peut donc être implémenté en parallèle.

L'algorithme ne construit que l'intérieur des sous-domaines. Pour obtenir un sous-domaine complet (avec ses nœuds de l'interface), on lui applique un algorithme qui lui ajoute les nœuds de recouvrement. Le recouvrement entre les domaines est de faible largeur (en général un nœud)

ALGORITHME 15 : Algorithme de constuction du séparateur dans l'arbre d'élimination par blocs d'une renumérotation de type dissection emboîtée.

$Sep \leftarrow$ `RaffinerSeparateur`(r) où la fonction récursive `RaffinerSeparateur` est définie par :

`RaffinerSeparateur`$(s) =$
$N_s =$ `nSousArbre`(s)
$N_{fils} = \left(\sum_{f_i,\text{fils de } s} \texttt{nSousArbre}(f_i)\right) / (\text{nombre de fils de } s)$
si $|N_{fils} - N_{cible}| > |N_s - N_{cible}|$ **alors**
  retourner s
sinon
  retourner $\bigcup_{f_i}$ `RaffinerSeparateur`(f_i)
fin

car il s'agit d'une union de séparateurs provenant d'une dissection emboîtée. L'interface obtenue par cette méthode est donc de faible dimension.

2.1.5 Factorisation symbolique

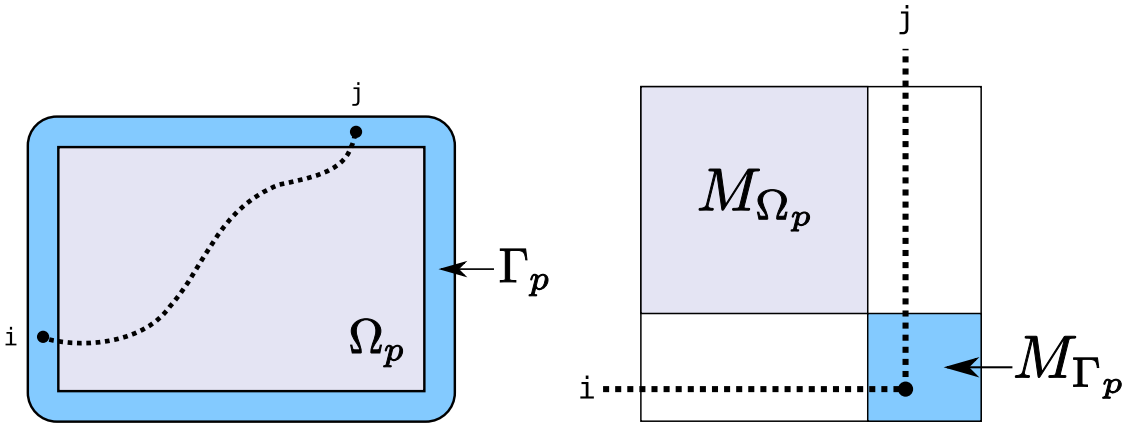
La renumérotation utilisée et les critères statiques de remplissage permettent de prévoir la structure des facteurs incomplets. A l'intérieur des sous-domaines, une renumérotation du direct est utilisée et la structure bloc des matrices B et E peut donc être obtenue par une factorisation symbolique par blocs peu coûteuse (algorithme 3 du chapitre 1).

Le théorème de caractérisation du remplissage 1 de la page 8 permet de démontrer que la structure bloc des facteurs L_S et U_S obtenue par la partition des inconnues de l'interface en connecteurs est en fait une structre bloc **dense**. En effet, les noeuds intérieurs ont dans la renumérotation de plus petits numéros et si on considère deux sommets i et j de l'interface Γ_p du domaine Ω_p, alors il existe un chemin entre i et j ne passant que par les noeuds intérieurs de Ω_p (lorsque Ω_p est constitué d'une seule composante connexe). Cette propriété est illustrée figure 2.10. Or, comme nous l'avons vu dans la section précédente, la partition bloc de la matrice S globale issue de la décomposition hiérarchique correspond, dans chaque matrice locale d'un sous-domaine Γ_p à la partition bloc de la matrice M_{Γ_p}. La structure bloc de la matrice S est donc aussi faite de blocs denses. Ainsi, en nous basant uniquement sur la décomposition hiérarchique du graphe de A et sur sa renumérotation dans l'intérieur des sous-domaines, il est possible de calculer à l'avance la structure en blocs denses des facteurs de A. La structure par bloc-colonnes et par blocs denses des facteurs de A est présentée figure 2.11.

2.2 Construction d'une méthode hybride directe-itérative

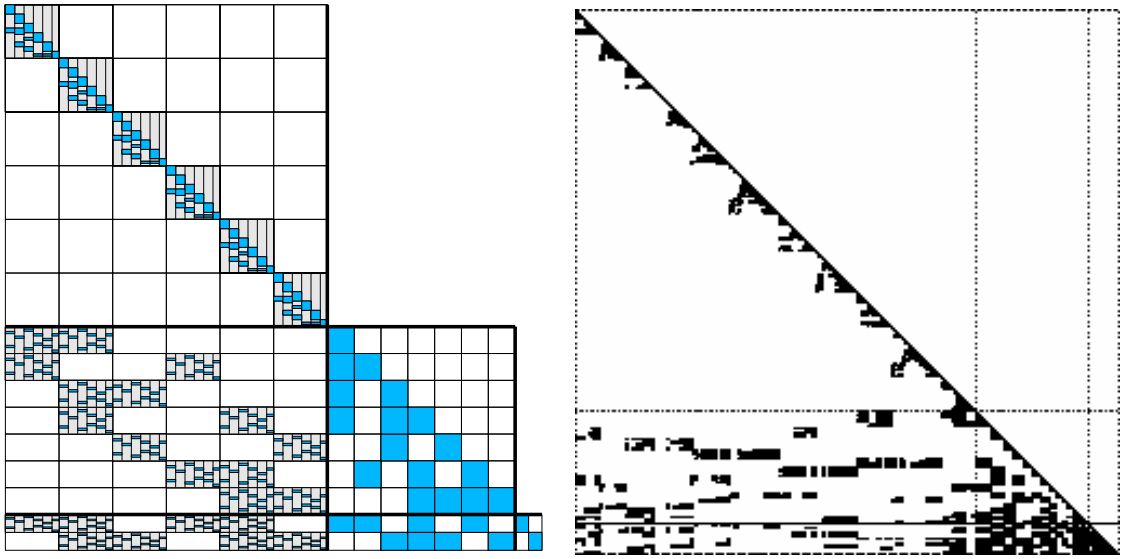
2.2.1 Utilisation du complément de Schur

Notre méthode de résolution hybride combine une méthode de résolution directe avec une méthode itérative par une approche de type complément de Schur. On considère une décomposition de domaine sans recouvrement. Celle-ci est obtenue à partir d'un partitionnement arête du graphe d'adjacence de la matrice (la frontière entre deux parties est constituée de sommets). On élimine de manière directe les nœuds associés à l'intérieur des sous-domaines pour ramener



(a) Domaine Ω_p (b) Matrice locale au domaine Ω_p

FIG. 2.10 – Illustration de la factorisation symbolique



(a) Motif des blocs d'une décomposition en 3 niveaux (b) Remplissage $\mathscr{R}_S$ de la matrice symétrique BCSSTK14 renumérotée par la décomposition hiérarchique de l'interface.

FIG. 2.11 – Structure bloc du préconditionneur

la résolution du système global à la résolution d'un système réduit aux nœuds de l'interface. D'une manière générale, en distinguant deux sous-ensembles d'inconnues P_B et P_C, le système $A.x = b$ peut se décomposer de la manière suivante :

$$A.\begin{pmatrix} x_B \\ x_C \end{pmatrix} = \begin{pmatrix} b_B \\ b_C \end{pmatrix} \tag{2.1}$$

On peut écrire A sous la forme d'une matrice par bloc :

$$A = \begin{pmatrix} B & F \\ E & C \end{pmatrix} \tag{2.2}$$

Si l'on considère une factorisation incomplète de la matrice A, la décomposition précédente s'écrit :

$$A = \begin{pmatrix} L_B & \\ EU_B{}^{-1} & S \end{pmatrix} \times \begin{pmatrix} U_B & L_B{}^{-1}F \\ & I \end{pmatrix}, \tag{2.3}$$

où $L_B.U_B$ est la factorisation de la matrice B et où S est la matrice du complément de Schur, définie par la relation :

$$S = C - E.U_B^{-1}.L_B^{-1}.F \tag{2.4}$$

Pour la suite, nous noterons $G = L_B^{-1}.F$ et $W = E.U_B^{-1}$.

Dans notre approche, les nœuds intérieurs sont éliminés en premier et correspondent donc à l'ensemble P_B et à la matrice B. La matrice C correspond aux nœuds interfaces entre les sous-domaines. La décomposition en facteurs de l'équation (2.3) permet de décrire la factorisation incomplète de A que l'on cherche à obtenir : on souhaite calculer une factorisation exacte de $L_B.U_B$ de B et une factorisation incomplète de S.

2.2.2 Variantes algorithmiques de résolution

Partant de la factorisation (2.3), la résolution du système peut se décomposer en trois systèmes :

$$\begin{cases} B.x'_B = b_B \\ S.x_C = b_C - E.x'_B \\ B.x_B = b_B - F.x_C \end{cases} \tag{2.5}$$

Si l'on note $\widetilde{B}^{-1}$ et $\widetilde{S}^{-1}$ des inverses (exacts ou approchés) pour les matrices B et S, alors on peut définir un préconditionneur M sous la forme :

$$x = M.y \Leftrightarrow \begin{cases} x_C = \widetilde{S}^{-1}(y_C - E.\widetilde{B}^{-1}y_B) \\ x_B = \widetilde{B}^{-1}(y_B - F.x_C) \end{cases} \tag{2.6}$$

L'expression (2.6) est une écriture très générique d'un préconditionneur utilisant la technique du complément de Schur. Le choix de $\widetilde{S}^{-1}$ et de $\widetilde{B}^{-1}$ permet d'en décliner plusieurs variantes. Dans notre cas, B^{-1} peut être considéré comme exacte ; nous nous intéressons donc à plusieurs méthodes pour calculer S^{-1}. Le sous-système $S.x_C = b_B - E.x_B$ peut par exemple être résolu en utilisant une factorisation de la matrice S ou en utilisant une méthode itérative.

Cette section présente différentes manières d'organiser le calcul du produit matrice-vecteur $x = M.y$ utilisé dans une méthode de Krylov préconditionnée par M. Les quatre variantes algorithmiques de M étudiées sont notées M_{norec}, M_0, M_1 et M_2. Une étude expérimentale de ces préconditionneurs est présentée plus tard, section 2.3.5, page 52.

Dans les méthodes M_{norec} et M_0, on préconditionne une résolution itérative du système global par la factorisation incomplète par bloc de A de l'équation (2.3). Les factorisations de B et de S interviennent directement dans l'écriture de cette factorisation. Grâce à la résolution directe du système associé à B, on peut aussi itérer uniquement sur le système du complément de Schur, comme dans les méthodes M_1 et M_2. De plus, aucune itération externe sur le système global n'est alors nécessaire. Ces méthodes sont numériquement strictement équivalentes lorsque B^{-1} est exacte.

Méthode sans récursion

Le préconditionneur M_{norec} correspond à l'utilisation « classique » d'une factorisation incomplète comme préconditionneur d'une méthode de Krylov : le produit $x = M.y$ est calculé par une descente-remontée sur les facteurs incomplets de la matrice A, obtenus par le schéma de remplissage $\mathscr{R}_C$ ou $\mathscr{R}_S$. En utilisant la décomposition (2.1) de A, le préconditionneur de l'équation (2.6) peut s'écrire sous la forme :

$$x = M_{norec}.y \Leftrightarrow \begin{cases} x_C = U_S^{-1} L_S^{-1} (y_C - W.L_B^{-1} y_B) \\ x_B = U_B^{-1} (L_B^{-1} y_B - G.x_C) \end{cases} \tag{2.7}$$

Dans notre approche hybride, $L_B.U_B$ est la factorisation exacte de B et $L_S.U_S$ est une factorisation incomplète de S.

La méthode itérative est alors appliquée sur le système initial A, dans l'espace de résolution complet. Si on utilise un GMRES, chaque itération stockera donc un vecteur supplémentaire de dimension n (où n est la dimension de A). Généralement, il sera alors nécessaire d'utiliser un paramètre de réinitialisation de la base assez « petit » pour limiter la mémoire nécessaire au stockage de ces vecteurs.

On s'intéresse au coût théorique d'une itération et donc au nombre d'opérations correspondant à l'application de l'opérateur $M^{-1}A$ à un vecteur. On note nnz(Mat) le nombre de non-zéros d'une matrice Mat. Le nombre d'opérations correspondant à un produit `matrice-vecteur` et à une descente-remontée est alors :

$$\text{nnz}(A) + \text{nnz}(L_B) + \text{nnz}(U_B) + \text{nnz}(G) + \text{nnz}(W) + \text{nnz}(L_S) + \text{nnz}(U_S)$$

Pour appliquer le préconditionneur M_{norec}, les facteurs de A doivent être intégralement conservés en mémoire. Le nombre de non-zéros à stocker en plus de A dans ce cas est :

$$\text{nnz}(M_{norec}) = \text{nnz}(L_B) + \text{nnz}(U_B) + \text{nnz}(G) + \text{nnz}(W) + \text{nnz}(L_S) + \text{nnz}(U_S)$$

Méthode 0 : récursion sans itération interne

Les matrices G et W stockent les opérateurs $L_B^{-1}F$ et $E.U_B^{-1}$ sous forme matricielle. Si on utilise directement ces opérateurs dans l'égalité (2.7), on obtient une variante du préconditionneur précédent que l'on note M_0 :

$$x = M_0.y \Leftrightarrow \begin{cases} x_C = U_S^{-1} L_S^{-1} (y_C - E.U_B^{-1}.L_B^{-1} y_B) \\ x_B = U_B^{-1}.L_B^{-1} (y_B - F.x_C) \end{cases} \tag{2.8}$$

Lorsque B est factorisée de manière exacte, M_0 est numériquement équivalent à M_{norec}. L'intérêt de cette variante est de ne pas stocker G et W. Le nombre d'opérations correspondant à un produit matrice-vecteur et un appel à M_0 est alors :

$$\text{nnz}(A) + 2 \times (\text{nnz}(L_B) + \text{nnz}(U_B) + \text{nnz}(E) + \text{nnz}(F) + \text{nnz}(L_S) + \text{nnz}(U_S)$$

Généralement, ce préconditionnement est donc plus coûteux que M_{norec} lorsque la dimension de B est grande. Dans notre cas, B est grande lorsque le nombre de sous-domaines est faible. Lorsque le nombre de sous-domaines augmente, le remplissage dans les couplages G et W augmente par rapport au remplissage dans L_B et U_B (l'intérieur des sous-domaines) et donc, M_0 pourra coûter moins cher en nombre d'opérations.

Le nombre de non zéros à stocker en plus de A pour utiliser le préconditionneur $M0$ est :

$$\text{nnz}(L_B) + \text{nnz}(U_B) + \text{nnz}(L_S) + \text{nnz}(U_S)$$

Méthode 1 : résolution itérative sur le complément de Schur (avec stockage du complément de Schur)

Dans les méthodes M_{norec} et M_0, S^{-1} est approchée par $U_S^{-1}.L_s^{-1}$ mais il est aussi possible d'approcher S^{-1} par une méthode de Krylov préconditionnée par $U_S^{-1}.L_S^{-1}$. Les méthodes M_1 et M_2 utilisent ce principe.

Lorsque B est factorisée de manière exacte, alors le complément de Schur S est aussi connu de manière exacte par l'égalité (2.4). Donc pour résoudre le système global $A.x = b$, on a juste besoin de connaître la solution du système $S.x_C = b_S$ où $b_S = b_C - E.x_B$ (cf. système (2.5).

Une première solution consiste donc à calculer explicitement le complément de Schur de manière exacte et à le stocker en mémoire pour résoudre de manière itérative le complément de Schur :

$$S.x_C = b_S \tag{2.9}$$

Les étapes de la résolution sont donc les suivantes :

1. $x_B = U_B^{-1}.L_B^{-1}b_B$,
2. Résoudre le système : $S.x_C = b_C - E.x_B$ par une méthode itérative préconditionnée, en utilisant L_S et U_S,
3. $x_B = U_B^{-1}.L_B^{-1}(b_B - F.x_C)$.

Il est important de noter que le grand avantage de cette solution par rapport aux préconditionneurs M_{norec} et M_0 est que les itérations se font sur le complément de Schur. Le système est plus petit, donc **les itérations plus rapides**. On peut aussi utiliser un paramètre de réinitialisation de la base dans un GMRES beaucoup plus grand que lorsqu'on itère sur le système initial. La proposition 1 assure que la résolution par une méthode de Krylov du système (2.9) préconditionné en utilisant $M_S = U_S^{-1}L_S^{-1}$ convergera de la même manière que le système global préconditionné par M_{norec} ou M_0.

Proposition 1 Soit L_A et U_A les préconditionneurs à gauche et à droite du système A tels que le système B soit préconditionné de manière exacte et S de manière approchée ($S \approx L_S.U_S$). On a L_A et U_A de la forme :

$$L_A = \begin{pmatrix} I & 0 \\ E.B^{-1} & L_S \end{pmatrix}, U_A = \begin{pmatrix} B & F \\ 0 & U_S \end{pmatrix} \tag{2.10}$$

La séquence des solutions approchées successives obtenue par une méthode de Krylov en utilisant ce préconditionnement appliqué à A est la même que celle qui serait obtenue en appliquant cette même méthode au système réduit $S = b_C - E.B^{-1}.b_B$ (en utilisant la même solution approchée de départ, réduite au sous-système S).

Démonstration 1 Cette proposition est démontrée dans [79], page 481.

■

La proposition 1 permet d'inférer que le comportement du préconditionneur M_1 est identique à celui des variantes précédentes :

Proposition 2 M_1 est numériquement équivalent aux variantes M_{norec} et M_0.

Proposition 3 Le comportement numérique a tolérance tol_S avec laquelle doit être résolu le système réduit pour que la solution du système global vérifie le critère d'arrêt $\frac{\|r\|}{\|b\|} < tol$ vérifie l'inégalité :

$$tol_S < tol.\frac{\|b\|}{\|b_S\|}$$

Lorsque l'on itère sur le système réduit (2.9), il faut déterminer un critère d'arrêt qui corresponde à celui qui est désiré sur le système global. Un critère d'arrêt pour la méthode itérative fréquemment employé est la norme relative du résidu par rapport au second membre. Une tolérance tol est fixée par l'utilisateur et l'on arrête la résolution du système global lorsque $\frac{\|r\|}{\|b\|} < tol$ où $r = b - A.x$. La proposition suivante détermine un critère d'arrêt sur le système réduit :

Proposition 4 La tolérance tol_S avec laquelle doit être résolu le système réduit pour que la solution du système global vérifie le critère d'arrêt $\frac{\|r\|}{\|b\|} < tol$ vérifie l'inégalité :

$$tol_S < tol.\frac{\|b\|}{\|b_S\|}$$

Démonstration 2 On note r_B et r_C les restrictions de r aux sous-systèmes B et C. On a :

$$\begin{cases} r_C = b_C - E.x_B + C.x_C \\ r_B = b_B - B.x_B - F.x_C \end{cases}$$

x_B est obtenue en utilisant une factorisation exacte de B par la relation :

$$x_B = U_B^{-1}.L_B^{-1}(b_B - F.x_C)$$

En supposant que la précision de la méthode directe permet de négliger l'erreur sur le résidu, on a $\|r_B\| = 0$ et donc $\|r\| = \|r_C\|$. De plus, en notant $r_S = b_S - S.x_C$ le résidu associé au système du complément de Schur, on a :

$$\begin{aligned} r_C &= b_C - E.x_B - C.x_C \\ &= b_C - E.(B^{-1}.(b_B - F.x_C)) - C.x_C \\ &= b_C - E.B^{-1}.b_B + (E.B^{-1}.F - C).x_C \\ &= b_S - S.x_C = r_S \end{aligned}$$

Rappelons que tol_S est défini par la relation $\dfrac{\|r_S\|}{\|b_S\|} < tol_S$.

Donc $\dfrac{\|r\|}{\|b\|} = \dfrac{\|r_S\|}{\|b\|} = \dfrac{\|r_S\|}{\|b_S\|} \times \dfrac{\|b_S\|}{\|b\|} \leq tol_S \dfrac{\|b_S\|}{\|b\|}$ et si $tol_S < tol \dfrac{\|b\|}{\|b_S\|}$ alors $\dfrac{\|r\|}{\|b\|} \leq tol$.

■

La méthode M_1 est pénalisante en terme de stockage mémoire car elle oblige à stocker le complément de Schur exact, c'est-à-dire une matrice contenant autant de zéros que dans L_S et U_S en utilisant le motif de remplissage $\mathscr{R}_S$. Le nombre d'opérations correspondant à un produit matrice-vecteur $S.x$ et à un appel au préconditionneur M_1 est :

$$\text{nnz}(S) + \text{nnz}(L_S) + \text{nnz}(U_S)$$

Le nombre de non-zéros à stocker en plus de A pour M_1 est :

$$\text{nnz}(L_B) + \text{nnz}(U_B) + \text{nnz}(S) + \text{nnz}(L_S) + \text{nnz}(U_S)$$

Méthode 2 : résolution itérative sur le complément de Schur (sans stockage du complément de Schur)

Dans la méthode M_1, le complément de Schur S est conservé sous forme matricielle afin de résoudre le système réduit sur l'interface à l'aide d'une méthode de Krylov préconditionnée. La méthode M_2 consiste à ne pas utiliser directement l'opérateur linéaire S sous forme matricielle mais à utiliser à la place l'égalité $S.x = (C - E.B^{-1}.F).x$.

L'intérêt de cette méthode est qu'elle permet **d'éviter le stockage** en mémoire de S. De plus, on peut éviter le calcul explicite du complément de Schur exact pour pouvoir appliquer la méthode itérative sur le système réduit correspondant. En effet, les facteurs L_S et U_S du préconditionneur peuvent par exemple être construits en limitant le calcul exact des termes de S au schéma de remplissage $\mathscr{R}_C$ des facteurs, ou a partir d'une approximation $\widetilde{S}$ de S moins coûteuse en calcul et en mémoire (voir l'approche présentée dans le chapitre suivant, page 65). Ce qui dans ces deux cas ne nécessite pas le calcul exact de S.

Cette méthode est équivalente numériquement à la méthode M_1 et reprend les mêmes étapes de résolution. Seule la manière d'effectuer les produits $S.x$ diffère. Le nombre d'opérations correspondant à un produit $S.x = (C - E.U_B^{-1}.L_B^{-1}.F).x$ et un appel au préconditionneur est alors :

$$\text{nnz}(C) + \text{nnz}(E) + \text{nnz}(F) + \text{nnz}(L_B) + \text{nnz}(U_B) + \text{nnz}(L_S) + \text{nnz}(U_S)$$

Le nombre de non-zéros à stocker en plus de A pour M_1 est :

$$\text{nnz}(L_B) + \text{nnz}(U_B) + \text{nnz}(S) + \text{nnz}(L_S) + \text{nnz}(U_S)$$

2.3 Expérimentations

Nous présentons dans cette partie des résultats expérimentaux sur des cas tests réguliers et irréguliers dans le but de valider notre approche et de justifier les études algorithmiques qui sont réalisées dans le chapitre suivant. Dans une première partie, l'algorithme de décomposition de l'interface est illustré sur deux matrices irrégulières. Nous nous intéressons ensuite à la convergence de la méthode de résolution dans le but de souligner sa résistance à l'augmentation du nombre de sous-domaines. Une étude du remplissage du préconditionneur souligne l'intérêt d'éviter le stockage mémoire de la matrice du complément de Schur. Enfin, une étude des coûts calculatoires des variantes algorithmiques du préconditionneur permet de dégager la variante la mieux adaptée aux problèmes issus de la discrétisation de maillage 3D.

2.3.1 Présentation des cas tests

On considère le problème de Poisson avec des conditions aux limites de type Dirichlet. On utilise un schéma de discrétisation de type différence finie sur un schéma centré utilisant 5 points en 2D et 7 points en 3D. Le domaine physique est un carré (grille 2D) ou un cube (grille 3D). La décomposition de domaine est régulière et la dimension de la grille est choisie afin d'obtenir une décomposition de domaine avec recouvrement parfaitement équilibrée. La taille des domaines est fixée et on double le nombre de domaines dans chaque dimension, en ajoutant un recouvrement d'une largeur d'un nœud entre les sous-domaines.

Nous avons aussi choisi deux problèmes irréguliers pour illustrer cette partie :

- Le cas test Shipsec5 est une matrice symétrique réelle issue d'un problème 2D de mécanique des structures de la collection PARASOL [1].

[1] http://www.parallab.uib.no/projects/parasol/data

– Le cas test HALTERE est une matrice symétrique complexe issue d'un problème 3D d'électromagnétisme de la collection de matrices de GRID-TLSE [2].

Le tableau 2.1 donne les caractéristiques générales de ces matrices. Les deux dernières colonnes donnent respectivement le remplissage des facteurs et le nombre d'opérations nécessaires à une factorisation directe de la matrice renumérotée par la bibliothèque de partitionnement SCOTCH. Les remplissages des facteurs sont donnés sous forme de ratio par rapport au nombre initial de termes dans A.

TAB. 2.1 – Propriétés générales des cas tests

Matrices	Nombre d'inconnues	Nombre de termes	Factorisation directe	
			nnz(L)	Nb d'op.
SHIPSEC5	179 860	4 598 604	$10,9$	$7,2\,.\,10^{10}$
HALTERE	1 288 825	10 476 775	$38,7$	$7,5\,.\,10^{11}$

2.3.2 Etude des partitions obtenues sur des matrices irrégulières

On s'intéresse tout d'abord à la décomposition hiérarchique obtenue sur les deux matrices irrégulières. Les matrices sont initialement renumérotées en utilisant la bibliothèque de partitionnement de graphe SCOTCH. On applique ensuite l'algorithme décrit dans la section 2.1.4 pour obtenir une décomposition de domaine plus ou moins fine. Enfin, on applique à l'interface l'algorithme de décomposition hiérarchique algébrique décrit à la section 2.1.2 et dans [47].

Les tableaux 2.2 et 2.3 présentent des statistiques du partitionnement obtenu sur l'interface, en fonction de la taille des domaines de la décomposition. La première colonne correspond au paramètre N_{cible} de l'algorithme 15 (page 40) et la deuxième colonne correspond au nombre de domaines obtenus par l'application de l'algorithme. La dernière colonne donne la taille de l'interface relativement à la dimension de la matrice initiale.

On peut voir que le nombre de niveaux est relativement élevé lorsque les tailles de domaines sont faibles. A titre de comparaison, l'interface de grilles régulières 2D et 3D peut être respectivement décomposée en 2 et 3 niveaux de manière géométrique, quelle que soit la taille des grilles considérées. Le nombre de niveaux et de connecteurs de la décomposition hiérarchique donne une indication de la complexité des interfaces et de la structure des matrices.

Lorsque la taille des domaines est trop faible, l'élimination en parallèle des nœuds de l'interface est plus difficile car chaque niveau induit une séquentialité (dépendance des calculs entre les niveaux). De plus, des problèmes numériques peuvent alors apparaître car cela signifie que pendant la factorisation, on ignore plus de chemins d'élimination. Lorsqu'il y a trop de connecteurs, la taille des blocs denses composant la structure du complément de Schur diminue et l'on perd l'efficacité des calculs BLAS.

Sur ces deux matrices, l'expérience montre qu'un paramètre minimal raisonnable pour N_{cible} est environ 1000 nœuds. Il permet d'obtenir un très grand nombre de domaines en regard de la taille des problèmes considérés.

Lorsqu'on utilise les variantes M_1 et M_2 du préconditionneur, on itère uniquement sur le système réduit aux interfaces. La taille du système est donnée par la dernière colonne des tableaux. Sa faible taille permet de considérer des paramètres de réinitialisation de la base de Krylov d'un

[2] http://tlse.enseeiht.fr

TAB. 2.2 – Partitionnement de la matrice SHIPSEC5 (2D)

Taille domaines	Nombre de domaines	Nombre de niveaux	Nombre de connecteurs	dim(S)/n (%)
125	857	6	4264	27,82
250	457	5	2821	23,58
500	221	6	1703	19,19
1000	131	5	914	15,46
2000	72	5	505	12,24
4000	36	6	232	8,56
8000	20	4	121	5,97
10000	18	4	110	5,67
20000	7	4	32	3,41

TAB. 2.3 – Partitionnement de la matrice HALTERE (3D)

Taille domaines	Nombre de domaines	Nombre de niveaux	Nombre de connecteurs	dim(S)/n (%)
125	5788	9	79124	27,30
250	3557	8	50566	22,96
500	1882	8	24863	18,22
1000	1020	6	9824	14,34
2000	551	5	4214	10,84
4000	286	5	2030	7,97
8000	151	5	989	5,80
10000	120	4	772	5,20
20000	57	4	352	3,62

GMRES beaucoup plus élevés que dans les méthodes M_{norec} et M_1. Par exemple, sur ces deux matrices, pour $M_{cible} = 1000$, on peut pour un coût en mémoire équivalent utiliser un paramètre de réinitialisation de la base sept fois plus important que si l'on itérait sur le système global.

2.3.3 Etude asymptotique de la convergence de la méthode

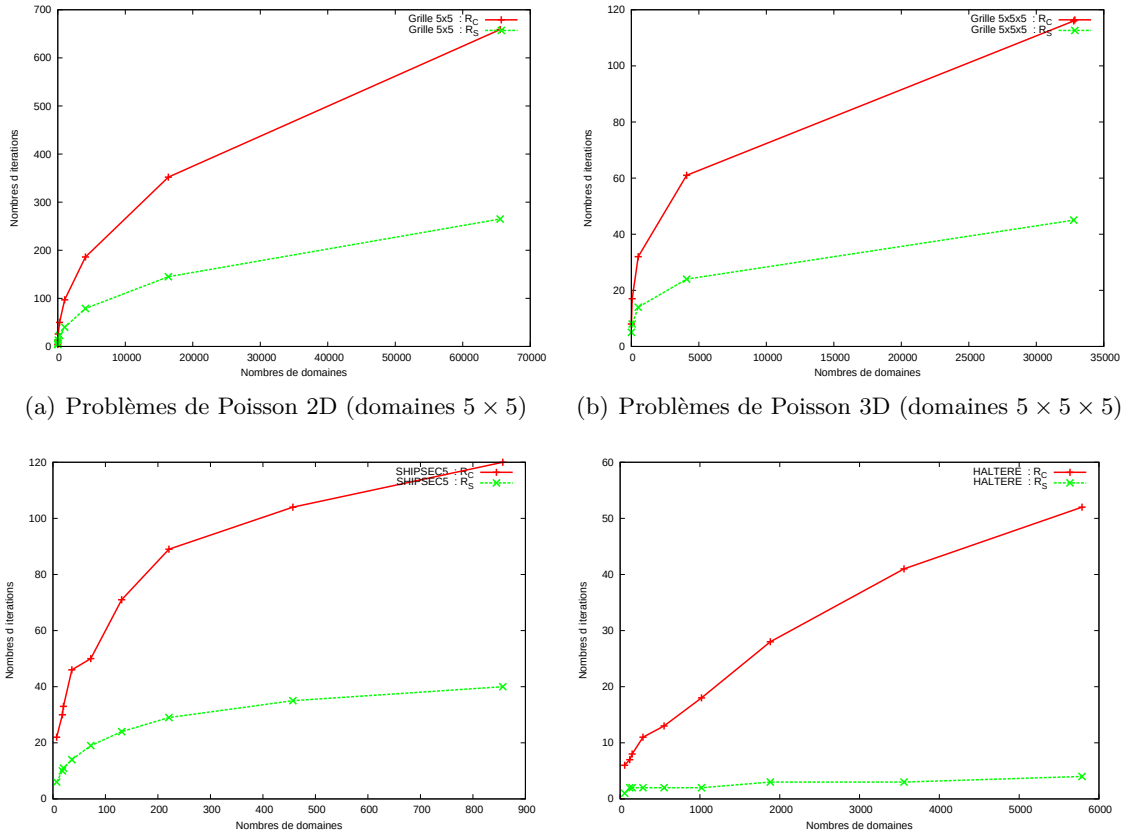
On étudie l'influence de l'augmentation du nombre de sous-domaines sur la convergence de notre méthode hybride. Les résultats de convergence sont indépendants de la variante utilisée pour le préconditionnement. Dans ce qui suit, le membre de droite de l'équation est généré pour la recherche de la solution $x = \mathbb{1}$. La précision est mesurée par la norme résiduelle relative $||b - A.x||/||b||$ et les résultats de convergence sont présentés pour une précision de 10^{-7}. La méthode itérative utilisée est un GMRES. Aucun paramètre de réinitialisation de la base de Krylov n'est utilisé afin de simplifier l'analyse de la convergence.

Les courbes de la figure 2.12 présentent le comportement asymptotique de la convergence. Sur chaque graphique, on représente les courbes de convergences obtenues sur un même problème pour les stratégies de remplissage $\mathscr{R}_C$ et $\mathscr{R}_S$. Les courbes 2.12(a) et 2.12(b) concernent des

problèmes réguliers de Poisson 2D et 3D. La machine utilisée dispose de 32 Go de mémoire. La taille des domaines est fixe (5×5 nœuds en 2D et $5 \times 5 \times 5$ nœuds en 3D) et on double le nombre de domaines dans chaque dimension, jusqu'à atteindre respectivement des grilles de 256×256 domaines 2D et $32 \times 32 \times 32$ domaines 3D. Pour ces grilles, les systèmes correspondants sont respectivement de dimension 2 356 225 en 2D et 6 967 871 en 3D.

Asymptotiquement, les résultats obtenus sont assez similaires à ceux d'une méthode de Schwarz additive ou multiplicative sur des problèmes de Poisson [84]. Le préconditionneur résiste bien à l'augmentation de la taille des problèmes. Comme attendu, le motif de remplissage $\mathscr{R}_S$ est plus robuste que la stratégie $\mathscr{R}_C$.

Une étude analogue a été réalisée pour les matrices SHIPSEC5 et HALTERE (figure 2.12(c) et figure 2.12(d)). Ici, c'est la taille du problème qui est fixe et on raffine la décomposition de domaines en utilisant un paramètre M_{cible} de plus en plus petit. Le comportement asymptotique est là aussi très satisfaisant. Le nombre d'itérations augmente faiblement, notamment dans le cas d'un remplissage $\mathscr{R}_S$ des facteurs.



(a) Problèmes de Poisson 2D (domaines 5×5)

(b) Problèmes de Poisson 3D (domaines $5 \times 5 \times 5$)

(c) Matrice SHIPSEC5 (2D)

(d) Matrice HALTERE (3D)

FIG. 2.12 – Etude asymptotique de la convergence (remplissage $\mathscr{R}_C$ et $\mathscr{R}_S$)

2.3.4 Etude du remplissage du préconditionneur

Les tableaux de la page 51 donnent le remplissage du préconditionneur pour les quatre cas tests et pour les motifs de remplissage $\mathscr{R}_C$ et $\mathscr{R}_S$. Le remplissage de chaque partie du préconditionneur est détaillé et exprimé sous forme de ratio par rapport au nombre de termes de la

matrice initiale. Le coût global du préconditionneur diminue lorsque le nombre de sous-domaines augmente car la part de factorisation incomplète grossit. Lorsque le nombre de domaines n'est pas faible par rapport à la taille du problème, la plus grande partie du remplissage se trouve dans $W = E.U^{-1}$ et S. Le coût de stockage de S est particulièrement important dans le cas des grilles 3D.

Le tableau 2.5 récapitule les parties de la matrice qu'il faut conserver pour chaque variante du préconditionneur. Les stratégies M_0 et M_2 sont particulièrement intéressantes, car les matrices S et W ne sont pas utilisées pour la phase de résolution. Ce sont alors des matrices temporaires que l'on peut supprimer au cours du préconditionnement. Le chapitre 3 présente des algorithmes évitant le stockage simultané de ces deux matrices lors du calcul du préconditionneur.

Le motif de remplissage $\mathscr{R}_C$ permet de creuser efficacement les facteurs du complément de Schur mais ce remplissage peut s'avérer trop restrictif pour des cas tests difficiles. Nous verrons dans le chapitre 3 que l'on peut alors limiter efficacement le remplissage de la stratégie $\mathscr{R}_S$ en ajoutant à la factorisation incomplète un critère de seuillage numérique.

TAB. 2.4 – Détails du remplissage du préconditionneur

(a) Problèmes de Poisson 2D (domaines 5×5)

Nombre de domaines	nnz(A)	nnz(L) (rapport)	nnz(W) (rapport)	nnz(S) (rapport)	nnz(L_S) (rapport) $\mathscr{R}_C$	$\mathscr{R}_S$
1	65	1,86				
2×2	341	1,34	1,08	0,53	0,24	0,53
4×4	1541	1,18	1,44	1,37	0,36	1,37
8×8	6533	1,12	1,60	1,81	0,41	1,81
16×16	26885	1,08	1,67	2,04	0,44	2,04
32×32	109061	1,07	1,70	2,15	0,46	2,15
64×64	439301	1,06	1,72	2,21	0,47	2,21
128×128	1763333	1,06	1,73	2,24	0,47	2,24
256×256	7065605	1,06	1,73	2,25	0,47	2,25

(b) Problèmes de Poisson 3D (domaines $5 \times 5 \times 5$)

Nombre de domaines	nnz(A)	nnz(L) (rapport)	nnz(W) (rapport)	nnz(S) (rapport)	nnz(L_S) (rapport) $\mathscr{R}_C$	$\mathscr{R}_S$
1	425	4,59				
$2 \times 2 \times 2$	4961	3,10	4,65	5,16	1,48	5,16
$4 \times 4 \times 4$	47081	2,63	5,95	13,46	2,37	13,46
$8 \times 8 \times 8$	408665	2,44	6,43	18,28	2,83	18,28
$16 \times 16 \times 16$	3402425	2,35	6,63	20,83	3,06	20,83
$32 \times 32 \times 32$	27762041	2,31	6,72	22,14	3,18	22,14

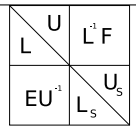
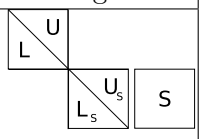
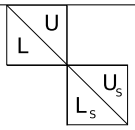
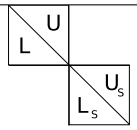
(c) Matrice SHIPSEC5 (2D) – nnz(A) = 179 860

Nombre de domaines	nnz(L) (rapport)	nnz(W) (rapport)	nnz(S) (rapport)	nnz(L_S) (rapport) $\mathscr{R}_C$	$\mathscr{R}_S$
7	12,85	6,68	4,15	0,92	4,15
18	9,49	6,52	5,19	0,79	5,19
20	9,26	6,47	5,31	0,83	5,31
36	7,13	5,92	6,26	0,92	6,26
72	5,09	5,05	7,32	1,04	7,32
131	4,00	4,36	7,08	0,99	7,08
221	2,94	3,74	8,13	0,96	8,13
457	2,03	3,27	7,79	0,95	7,79
857	1,53	2,94	7,04	0,88	7,04

(d) Matrice HALTERE (3D) – nnz(A) = 1 288 825

Nombre de domaines	nnz(L) (rapport)	nnz(W) (rapport)	nnz(S) (rapport)	nnz(L_S) (rapport) $\mathscr{R}_C$	$\mathscr{R}_S$
57	15,73	8,87	6,99	0,88	6,99
120	12,64	8,40	6,87	0,89	6,87
151	11,76	8,24	6,86	0,89	6,86
286	9,32	7,65	7,08	0,92	7,08
551	7,16	6,93	6,98	0,97	6,98
1020	5,43	6,12	7,07	1,00	7,07
1882	4,17	5,35	7,12	0,98	7,12
3557	3,07	4,73	6,93	1,00	6,93
5788	2,31	4,34	6,64	1,05	6,64

TAB. 2.5 – Tableau récapitulatif du stockage mémoire nécessaire dans les quatre variantes du préconditionneur

	Itérations	Stockage		Itérations	Stockage
M_{norec}	sur A		M_1	sur S	
M_0	sur A		M_2	sur S	

2.3.5 Etude des variantes algorithmiques

Les courbes de la page 54 présentent un comparatif théorique du coût d'utilisation des différents préconditionneurs. Pour chaque variante M_{norec}, M_0, M_1, M_2, on considère le nombre d'opérations nécessaires au calcul d'une itération. Il est calculé à partir du remplissage de la factorisation incomplète et les formules de coût de la section 2.2.2. La méthode M_{norec} sert de référence. Les courbes indiquent le coût relatif des autres variantes, en fonction du nombre de domaines. Chaque courbe représente donc le nombre d'itérations que l'on peut effectuer dans chacune des méthodes pour le même nombre d'opérations que prend une itération de la méthode M_{norec}.

Les cas tests réguliers permettent de mettre en évidence la tendance qui se dégage lorsque le nombre de domaines est important. Itérer sur S est moins coûteux qu'itérer globalement. Le préconditionneur M_2 est le moins coûteux, tant en 2D qu'en 3D car le remplissage de S est plus important que celui des matrices L_B, U_B, E, F utilisées par la stratégie M_2 pour le calcul de $S.x = (C - E.U_B^{-1}.L_B^{-1}.F).x$.

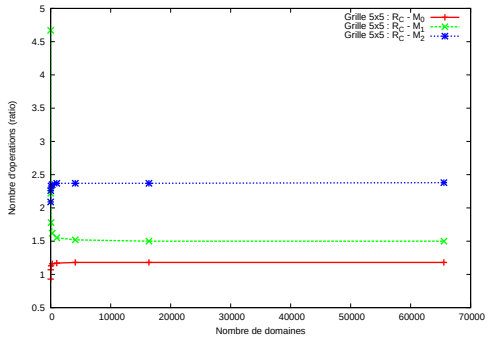
Les cas tests irréguliers présentent la même tendance asymptotique. On peut de plus vérifier à l'inverse que lorsque le nombre de sous-domaines est plus faible, il est plus intéressant de stocker le complément de Schur et d'utiliser la méthode M_1 (la matrice B est grande et S est alors petite).

Conclusion

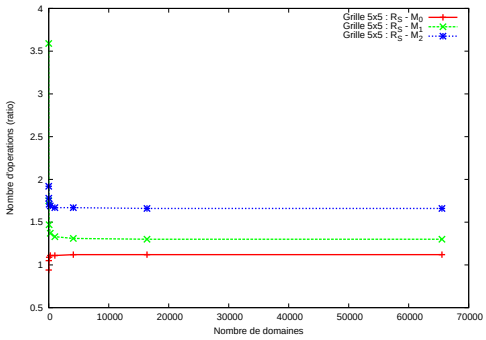
Nous nous sommes intéressés dans ce chapitre à la construction d'un préconditionneur pour une méthode itérative de type Krylov. Il utilise une décomposition de domaine algébrique qui provient d'une renumérotation du direct qui minimise le remplissage. On se ramène à la résolution d'un problème aux interfaces par une technique de type complément de Schur. L'interface peut alors être partitionnée pour obtenir une factorisation incomplète par blocs denses du complément de Schur fortement parallèle. Ce préconditionneur est hybride dans le sens où il combine une factorisation directe et une factorisation incomplète.

Les expérimentations ont permis de démontrer la robustesse de l'approche sur des cas tests réguliers et irréguliers. La méthode de préconditionnement résiste bien à l'augmentation du nombre de sous-domaines. L'usage de petits domaines diminue le coût du préconditionneur et augmente le parallélisme de la factorisation. L'étude du remplissage permet de conclure qu'il est plus intéressant, à la fois pour des critères mémoires et de temps séquentiels, d'utiliser la variante

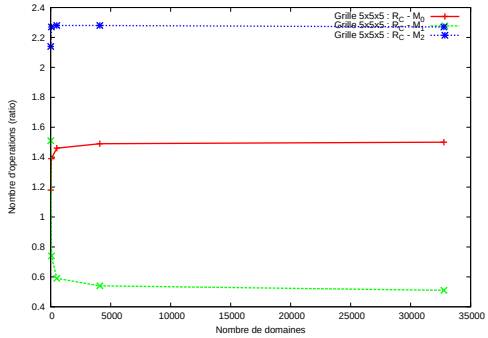
M_2 du préconditionneur pour la résolution de cas 3D. Cette variante consiste à n'itérer que sur le système réduit et à utiliser un produit de Schur $S.x$ implicite dans la résolution itérative.



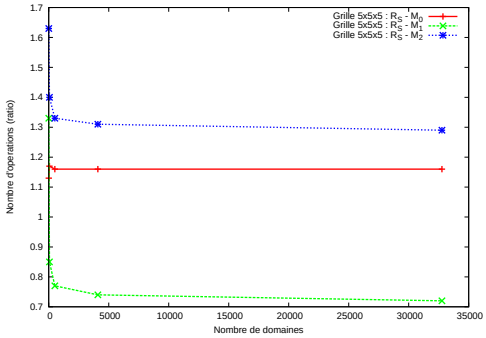
(a) Problèmes de Poisson 2D – Remplissage $\mathscr{R}_C$



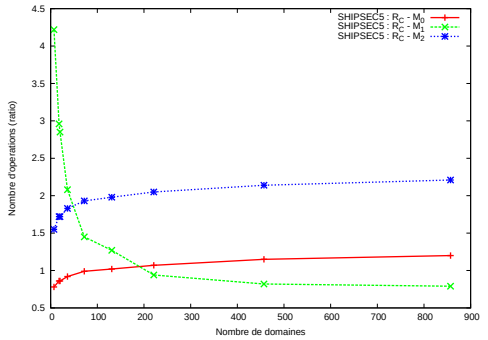
(b) Problèmes de Poisson 2D – Remplissage $\mathscr{R}_S$



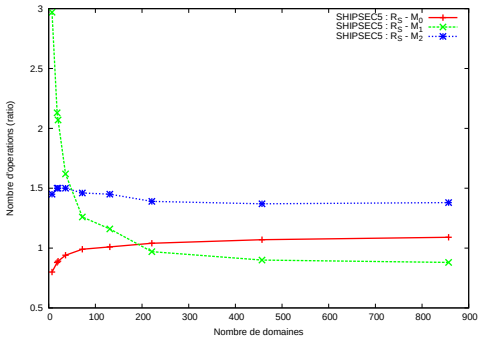
(c) Problèmes de Poisson 3D – Remplissage $\mathscr{R}_C$



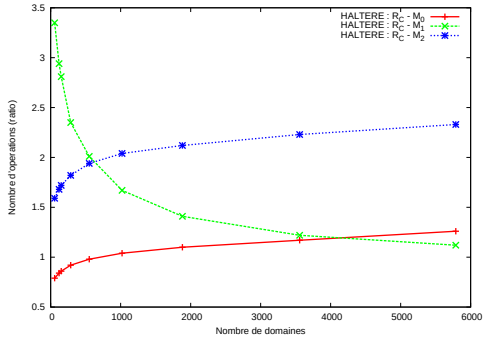
(d) Problèmes de Poisson 3D – Remplissage $\mathscr{R}_S$



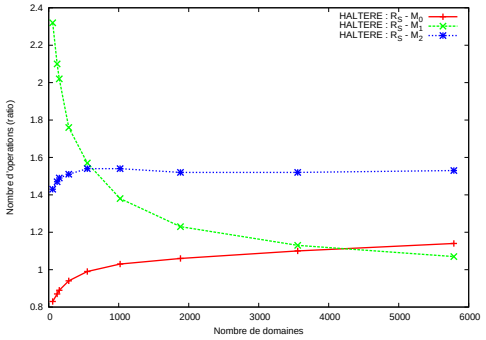
(e) Shipsec5 – Remplissage $\mathscr{R}_C$



(f) Shipsec5 – Remplissage $\mathscr{R}_S$



(g) Haltere – Remplissage $\mathscr{R}_C$



(h) Haltere – Remplissage $\mathscr{R}_S$

Fig. 2.13 – Coût relatif d'une itération pour M_0, M_1 et M_2

Chapitre 3

Algorithmique

Sommaire

3.1 Construction à faible coût mémoire **59**
3.1.1 Factorisation incomplète à partir du complément de Schur exact 61
3.1.2 Approche fondée sur le calcul approché du complément de Schur 65
3.2 Parallélisation et équilibrage de charge **71**

Introduction

Ce chapitre présente les algorithmes utilisés pour construire un préconditionneur à moindre **coût mémoire** et en **parallèle**, en s'appuyant sur l'approche générale décrite dans le chapitre 2. Nous avons construit un préconditionneur pour le système réduit du complément de Schur, sous la forme d'une factorisation incomplète L_S, U_S. Cette factorisation a une structure par blocs denses induite par la numérotation et la décomposition de l'interface entre les sous-domaines.

Dans une première partie, on s'attache à réduire le coût mémoire inhérent au calcul de ce préconditionneur. Deux approches sont étudiées. Dans la première approche, on considère l'algorithme de factorisation incomplète utilisant uniquement le critère de remplissage $\mathscr{R}_C$ ou $\mathscr{R}_S$ défini sur la partition de l'interface. Il s'agit dans ce cas d'un algorithme par blocs denses. Dans la seconde approche, on utilise en plus un critère de seuillage numérique pour limiter la consommation mémoire. Il s'agit alors de coupler efficacement une factorisation ILUT restreinte au schéma de remplissage $\mathscr{R}_C$ ou $\mathscr{R}_S$ avec une factorisation directe supernodale.

La seconde partie est consacrée à la parallélisation du calcul du préconditionneur. Le schéma d'équilibrage de charge est fondé sur l'attribution de plusieurs sous-domaines par processeur.

Dans tout ce qui suit, on ne considère que le préconditionneur M_2 présenté dans le chapitre 2 car il s'agit de la variante la plus efficace pour les cas issus de la discrétisation de problèmes en 3D (voir section 2.3). Pour rappel, cette variante consiste à n'itérer que sur le système réduit et à utiliser un produit de Schur $S.x$ implicite dans la résolution itérative.

Présentation de l'algorithme de calcul du préconditionneur

Considérons la factorisation incomplète de A qui utilise la partition et la renumérotation décrite au chapitre 2. Le remplissage admis est défini par un schéma de remplissage ($\mathscr{R}_C$ ou $\mathscr{R}_S$). On note $\widetilde{L_S}.\widetilde{U_S}$ la factorisation incomplète de S obtenue en n'autorisant aucun remplissage en dehors de $\mathscr{R}_C$ ou $\mathscr{R}_S$. Avec les notations du chapitre précédent, la factorisation incomplète globale du système $A.x = b$ peut s'écrire sous la forme :

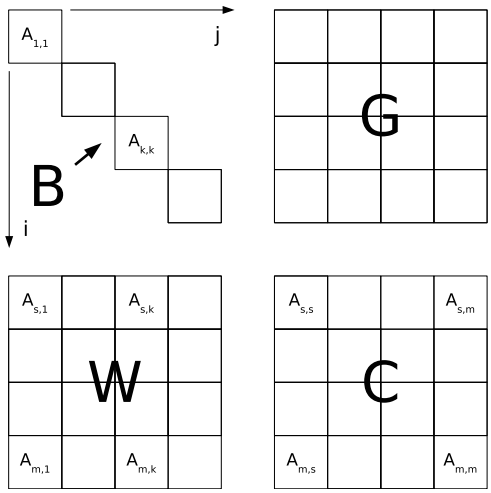
$$A = \begin{pmatrix} L_B & 0 \\ EU_B^{-1} & \widetilde{L_S} \end{pmatrix} \times \begin{pmatrix} U_B & L_B^{-1}F \\ 0 & \widetilde{U_S} \end{pmatrix} = \begin{pmatrix} L_B & 0 \\ W & \widetilde{L_S} \end{pmatrix} \times \begin{pmatrix} U_B & G \\ 0 & \widetilde{U_S} \end{pmatrix} \tag{3.1}$$

L'algorithme pour obtenir la factorisation incomplète peut donc être vu comme l'adaptation de l'algorithme de factorisation directe au motif de remplissage $\mathscr{R}_C$ ou $\mathscr{R}_S$ (algorithme 4 du chapitre 1). Cet algorithme peut aussi s'écrire selon la structure bloc induite par la partition en sous-domaines et en connecteurs. Dans cette formulation, les indices de boucles correspondent donc à des morceaux de cette partition. Nous noterons $\overline{\mathscr{R}_C}$ et $\overline{\mathscr{R}_S}$ les schémas de remplissage par blocs : si $(i,j) \in \overline{\mathscr{R}}$ alors on admet du remplissage dans le bloc d'indice (i,j) ($\overline{\mathscr{R}}$ est le passage au quotient de $\mathscr{R}$ par rapport à la partition). Les notations liées à cette structure sont présentées figure 3.1 et illustrées figure 3.2. L'algorithme 16 correspond à la factorisation « Right-Looking » d'une matrice non-symétrique A. La matrice S peut être obtenue par extraction de la sous-matrice correspondante dans A, avant factorisation de la colonne s. Il est important de noter que dans l'algorithme, le calcul n'est effectué que pour les termes appartenant au motif de remplissage de $\widetilde{L_S}$ et $\widetilde{U_S}$, c'est-à-dire selon $\mathscr{R}_C$ ou $\mathscr{R}_S$ (ligne 6). Rappelons aussi que les facteurs L et U de A sont des matrices creuses par blocs denses. Comme pour un solveur direct supernodal, la factorisation peut donc tirer parti de l'efficacité des opérations BLAS 3 `matrice` × `matrice` (cf. chapitre 1, page 12). Les opérations matricielles entre blocs $A_{i,j}$ de l'algorithme sont donc des opérations complexes, réalisées entre des matrices creuses par blocs denses. De

plus, les blocs $A_{i,k}$ d'une même colonne k partagent la même structure interne en colonnes supernodales et les blocs d'une colonne supernodale peuvent être stockés de manière contiguë afin de compacter les appels BLAS sur l'ensemble d'une colonne de L (et ligne de U).

Notations :
- On note m le nombre de connecteurs.
- La partition en domaines et connecteurs induit une structure par blocs de la matrice : $A = (A_{i,j})_{1 \leq i \leq m, 1 \leq j \leq m}$.
- Dans B, les matrices $A_{i,i}$ sont creuses. Elles ont une structure interne en bloc-colonnes où chaque colonne est constituée de blocs denses.
- On note $\mathscr{R}$ le schéma de remplissage par blocs de $\widetilde{L_S}$, $\widetilde{U_S}$ ($\overline{\mathscr{R}} = \overline{\mathscr{R}_C}$ ou $\overline{\mathscr{R}_S}$).
- Dans S, si $(i,j) \in \overline{\mathscr{R}}$, alors $A_{i,j}$ est dense.
- On note s le premier indice du complément de Schur.
- On note c_k la liste des indices lignes des blocs extra-diagonaux du bloc-colonne k. (Note : ces indices sont nécessairement supérieurs ou égaux à s).

FIG. 3.1 – Notations de la structure bloc induite par la partition (1/2)



$B = (A_{i,j})_{1 \leq i \leq s-1,\, 1 \leq j \leq s-1}$

$W = (L_{i,j})_{s \leq i \leq m,\, 1 \leq j \leq m}$

$C = (A_{i,j})_{s \leq i \leq m,\, s \leq j \leq m}$

FIG. 3.2 – Notations de la structure bloc induite par la partition (2/2)

ALGORITHME 16 : Factorisation incomplète de A selon la structure bloc induite par la partition ($\mathscr{R}_S$ ou $\mathscr{R}_C$)

pour $k = 1$ *à* m **faire**
  `/* Factorisation du bloc diagonal */`
  Factoriser $A_{k,k}$ en $L_{k,k}.U_{k,k}$
  `/* Calcul d'un bloc-colonne et d'un bloc-ligne */`
  pour $i \in c_k$ **faire**
    $L_{i,k} \leftarrow A_{i,k}.U_{k,k}^{-1}$
    $U_{k,i} \leftarrow L_{k,k}^{-1}.A_{k,i}$
  `/* Calcul des contributions de la colonne */`
  `/* (mise à jour des colonnes` $j > k$`) */`
6 **pour** $i \in c_k$, $j \in c_k$ *tq* $(i,j) \in \overline{\mathscr{R}}$ **faire**
    $A_{i,j} \leftarrow A_{i,j} - L_{i,k}.U_{k,j}$

3.1 Construction à faible coût mémoire d'une factorisation incomplète du complément de Schur

Dans cette partie, on cherche à construire le préconditionneur en minimisant la mémoire nécessaire à la fois pour son calcul et son stockage. Le calcul du préconditionneur peut s'effectuer en trois étapes distinctes :

ALGORITHME 17 : Étapes du calcul du préconditionneur

---

1. Factorisation exacte de $B = L_B.U_B$
2. Calcul de $W \leftarrow EU_B^{-1}$, $G \leftarrow L_B^{-1}F$ et du complément de Schur $S \leftarrow A_C - W.G$,
3. Factorisation incomplète de $S = L_S.U_S$.

Cet algorithme correspond simplement à l'écriture de la factorisation par bloc de la matrice A, le calcul du complément de Schur correspondant au report des contributions de la factorisation du premier bloc-colonne $\binom{B}{E}$ et bloc-ligne (BF) dans C.

Dans le cas du préconditionneur M_2, on rappelle que :

1. La résolution itérative a uniquement besoin du vecteur résultat de l'opération $S.x$ (produit de Schur). Ce vecteur peut être calculé sans utiliser la matrice S par la formule $(A_C - E.U_B^{-1}.L_B^{-1}F).x$ qui ne nécessite que la factorisation directe de l'intérieur des sous-domaines ainsi que les sous-parties E et F de la matrice A.
2. Les matrices W, G et S ne sont pas utilisées pour la résolution du système d'équations (2.5) page 42. Ce sont donc des matrices temporaires que l'on peut supprimer au cours du préconditionnement.

Ainsi, le préconditionneur M_2 ne nécessite ni de conserver la matrice S, ni les matrices de couplage (W, G). La factorisation de S est réalisée sur place et les matrices (W, G) peuvent être éliminées au cours du calcul du préconditionneur, dès que leurs contributions ont été reportées dans S. Le préconditionneur est alors formé par les matrices L_B, U_B, L_S et U_S. La mémoire nécessaire au stockage final du préconditionneur, en nombre de non-zéros, est donc :

$$nnz(P) = nnz(L_B) + nnz(U_B) + nnz(L_S) + nnz(U_S)$$

Les étapes du calcul sont illustrées figure 3.3.

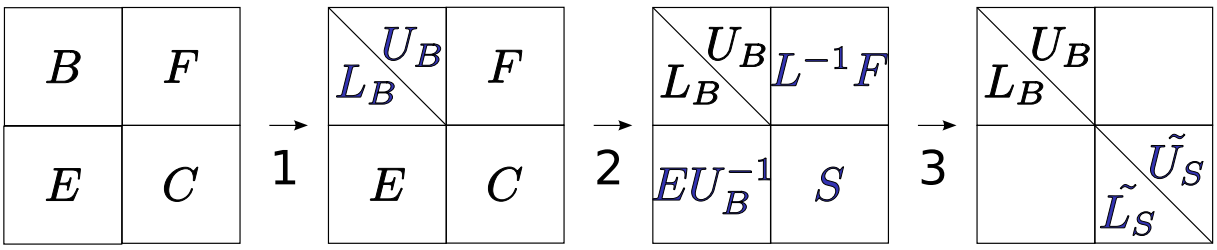


FIG. 3.3 – Algorithme de construction du préconditionneur.

Si on utilise l'algorithme 17, il faut temporairement stocker simultanément les matrices W, G et S (étape 2 de l'algorithme). A cette étape, on atteint alors un pic mémoire : il faut plus de mémoire pendant le calcul du préconditionneur que pour le stockage final de celui-ci. La figure 3.4 illustre le schéma de remplissage du facteur global de A obtenu sur une matrice réordonnée. La

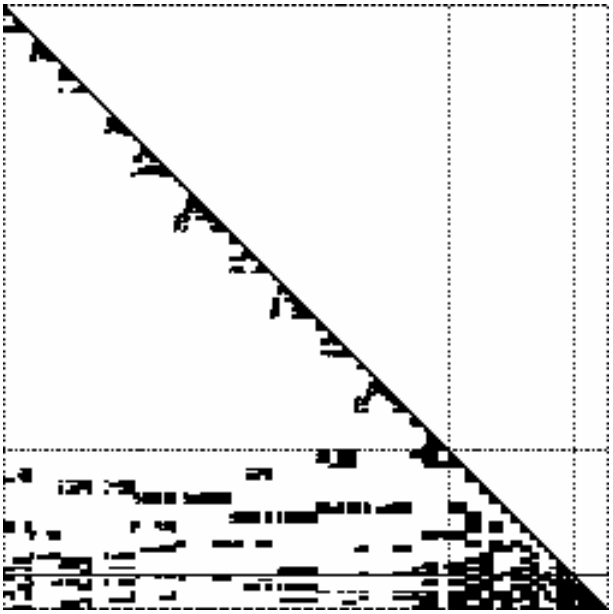


FIG. 3.4 – Factorisation symbolique par bloc de la matrice symétrique BCSSTK14

grande partie en haut à gauche de la matrice correspond au schéma de remplissage de L_B et la partie en bas à droite correspond à S. On peut constater que la plus grande partie du remplissage apparaît dans W et S. C'est aussi ce qui ressort de l'étude du remplissage des facteurs, chapitre 2, page 49. C'est pourquoi il est intéressant d'optimiser l'étape 2 de l'algorithme 17. Étudions ce surcoût mémoire temporaire :

- Dans le cas d'un motif de remplissage $\mathscr{R}_S$, la mémoire nécessaire pour les facteurs L_S et U_S est par définition identique à la mémoire nécessaire au stockage de S. A l'étape 2 de l'algorithme 17, on doit donc stocker :

$$\begin{aligned} nnz(P_{pic}) &= nnz(L_B) + nnz(U_B) + nnz(W) + nnz(G) + nnz(S) \\ &= nnz(P) + nnz(W) + nnz(G) \end{aligned}$$

- Dans le cas d'un motif de remplissage $\mathscr{R}_C$, il n'est pas nécessaire de calculer complètement la matrice S. En effet, dans le cas du préconditionneur M_2, celle-ci ne sert qu'à initier la factorisation incomplète du complément de Schur. On n'a donc besoin que des termes de S inclus dans $\mathscr{R}_C$.

Dans tous les cas, le surcoût mémoire temporaire correspond donc au stockage des matrices W et G.

La première partie de cette section présente un algorithme permettant de réduire le pic mémoire de la phase de construction du préconditionneur par un couplage fin des calculs de (W, G) et de S, sans modifier le préconditionneur. Pour réduire globalement la mémoire nécessaire à la fois au calcul et au stockage du préconditionneur, une autre possibilité est d'utiliser un seuillage numérique, à la fois pendant le calcul de (W, G) et celui de S et de ses facteurs. C'est la méthode présentée dans un second temps. La factorisation incomplète de S correspond alors à celle d'un complément de Schur approché.

3.1.1 Factorisation incomplète à partir du complément de Schur exact

Pour réduire le pic mémoire, on affine l'écriture de l'algorithme 17 en s'intéressant à la structure bloc-colonne de la matrice A. A chaque sous-domaine correspond un bloc-colonne de W (et respectivement un bloc-ligne de G). A chaque fois qu'un tel bloc-colonne est calculé, sa contribution au calcul de S peut être effectuée immédiatement et ce bloc-colonne peut donc être libéré. Lors de l'élimination d'un bloc-colonne, nous n'avons pas besoin de stocker simultanément plusieurs bloc-colonnes.

Le pic mémoire peut ainsi être réduit en n'allouant au même instant qu'un seul bloc-colonne de W (et qu'un bloc-ligne de G). Le surcoût mémoire lors du calcul du préconditionneur correspond alors au stockage du bloc-colonne le plus coûteux. L'algorithme est illustré figure 3.5. Il peut s'écrire comme une factorisation « Right-Looking » de A (voir algorithme 18). En pratique, on utilise un unique buffer mémoire de la taille maximale d'un bloc-colonne et L_S, U_S sont pré-alloués complètement. Cela est possible car les tailles de toutes les structures mémoires sont connues à l'avance par la factorisation symbolique.

ALGORITHME 18 : Factorisation LU selon la structure bloc induite par la partition, sans stockage de la matrice de couplage (ALGO. I)

Allouer $(A_{i,i})_{1\leq i<s}$ et $(A_{i,j})_{s\leq i\leq m,\, s\leq j\leq m}$

pour $k = 1$ *à* m **faire**

```
/* Factorisation du bloc diagonal                                  */
```

Factoriser $A_{k,k}$ en $L_{k,k}.U_{k,k}$

```
/* Allocation W / G                                                */
```

si $k < s$ **alors**

  Allouer le bloc-colonne $W_k = (L_{i,k})_{i\in c_k}$

  Allouer le bloc-ligne $G_k = (U_{k,j})_{i\in c_k}$

```
/* Calcul d'un bloc-colonne et d'un bloc-ligne                     */
```

pour $i \in c_k$ **faire**

  $L_{i,k} \leftarrow A_{i,k}.U_{k,k}^{-1}$

  $U_{k,i} \leftarrow L_{k,k}^{-1}.A_{k,i}$

```
/* Calcul des contributions de la colonne                          */
/* (mise à jour des colonnes j > k)                                */
```

pour $i \in c_k,\ j \in c_k\ tq\ (i,j) \in \mathscr{R}$ **faire**

  $A_{i,j} \leftarrow A_{i,j} - L_{i,k}.U_{k,j}$

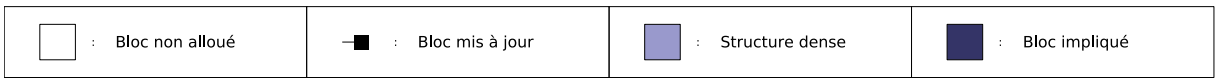
```
/* Libération W / G                                                */
```

si $k < s$ **alors**

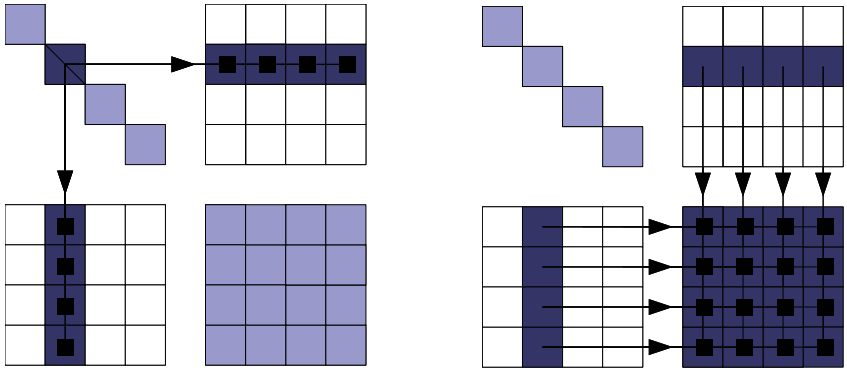
  Libérer le bloc-colonne $W_k = (L_{i,k})_{i\in c_k}$

  Libérer le bloc-ligne $G_k = (U_{k,j})_{i\in c_k}$

Légende :

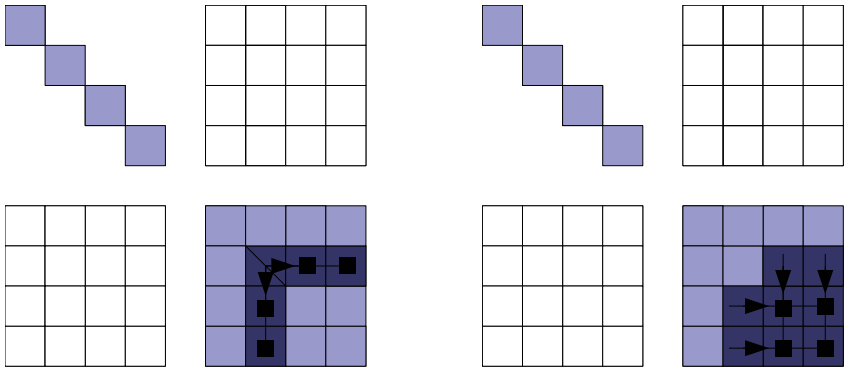


Dans l'algorithme 18, pour $k < s$ (calcul de L_B, U_B, W, G et S) :



(a) Factorisation d'un bloc-colonne (b) Calcul des contributions

Dans l'algorithme 18, pour $k \geq s$ (calcul de L_S et U_S) :



(c) Factorisation d'un bloc-colonne de S (d) Calcul des contributions

Note : pour plus de visibilité, tous les blocs $A_{i,j}$ de W, G et S ont été représentés comme participant aux calculs. En réalité, de nombreux blocs $A_{i,j}$ sont vides (voir figure 2.11, page 41).

FIG. 3.5 – Illustration de l'algorithme 18

Résultats expérimentaux

On étudie le gain mémoire apporté par l'algorithme 18 sur deux cas tests irréguliers :
- Le cas test HALTERE est une matrice symétrique complexe issue d'un problème 3D d'électromagnétisme de la collection de matrices de GRID-TLSE [3].
- Le cas test AUDI est une matrice symétrique réelle issue d'un problème 3D de mécanique des structures de la collection PARASOL [4].

Le tableau 3.1 donne les caractéristiques générales de ces matrices.

TAB. 3.1 – Propriétés générales des cas tests

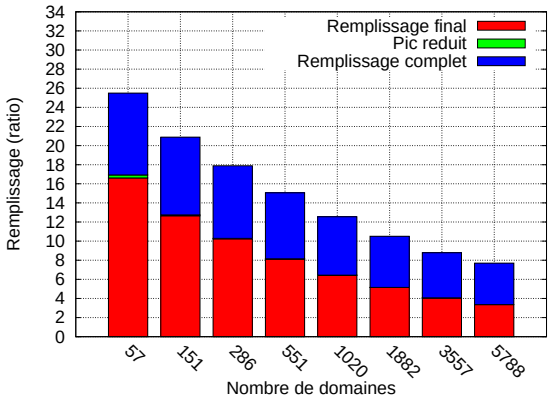
Matrices	Nombre d'inconnues	Nombre de termes	Factorisation directe	
			nnz(L)	Nb d'op.
HALTERE	1 288 825	10 476 775	$38,7$	$7,5\,.\,10^{11}$
AUDI	943 695	39 297 771	$28,1$	$5,3\,.\,10^{12}$

Les histogrammes de la page 64 présentent, pour chaque cas test, la mémoire nécessaire :
- au calcul du préconditionneur, dans le cas où W, G et S sont stockées simultanément (« Remplissage complet »),
- au calcul du préconditionneur, en utilisant l'algorithme 18 (« Pic réduit »),
- au stockage final du préconditionneur (« Remplissage final »).

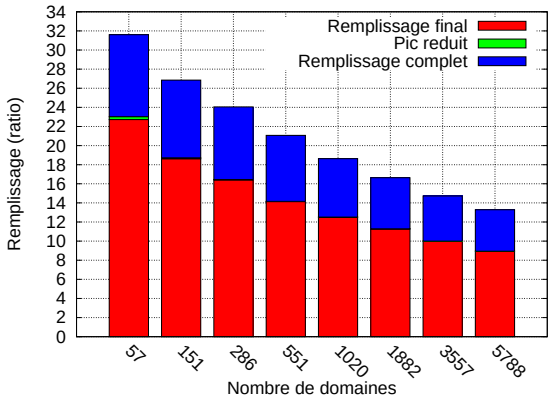
Les coûts mémoires sont exprimés en terme de non-zéros et sous forme de ratio par rapport au nombre de non-zéros de la matrice initiale. Les résultats sont présentés dans le cas d'un motif de remplissage $\mathscr{R}_C$ et $\mathscr{R}_S$. Le stockage complet de W est très coûteux mais l'algorithme 18 fait presque entièrement disparaître le pic mémoire. L'algorithme est d'autant plus efficace que le nombre de domaines est élevé car cela augmente le nombre de bloc-colonnes de W et en diminue la taille. Il serait possible de réduire encore davantage le pic mémoire en augmentant la granularité de l'algorithme. Nous pourrions en effet allouer les bloc-colonnes supernodaux les uns après les autres, mais cela nécessiterait un plus grand morcellement de la mémoire (pour pouvoir allouer et désallouer séparément les bloc-colonnes supernodaux).

L'écart mémoire entre un remplissage $\mathscr{R}_C$ et $\mathscr{R}_S$ est très important et le second algorithme de cette section permet notamment de trouver un compromis entre ces deux approches.

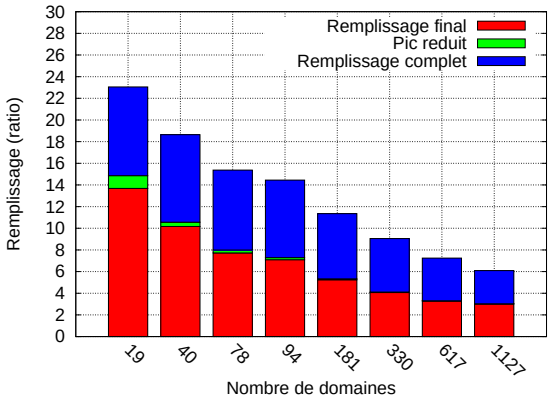
[3] http://tlse.enseeiht.fr
[4] http://www.parallab.uib.no/projects/parasol/data



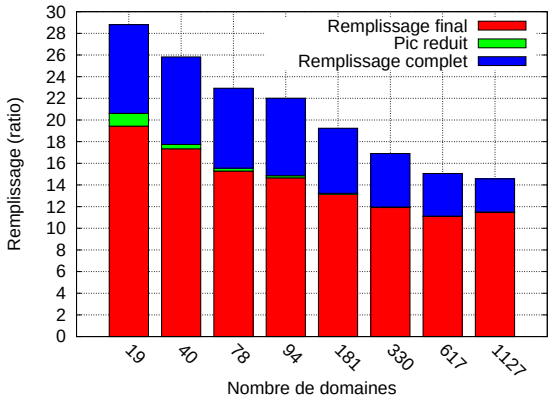
(a) Matrice HALTERE - $\mathscr{R}_C$



(b) Matrice HALTERE - $\mathscr{R}_S$



(c) Matrice AUDI - $\mathscr{R}_C$



(d) Matrice AUDI - $\mathscr{R}_S$

FIG. 3.6 – Etude du pic mémoire de l'algorithme 18

3.1.2 Approche fondée sur le calcul approché du complément de Schur

Grâce à l'étude précédente, nous avons constaté que $\mathscr{R}_S$ est beaucoup plus robuste que la variante $\mathscr{R}_C$ en nombre d'itérations. Mais la mémoire et le coût de calcul de cette méthode sont beaucoup plus importants. Dans [30], nous proposons donc d'ajouter une règle de seuillage numérique en plus des règles de remplissage imposées par $\mathscr{R}_S$. Le seuillage numérique correspond à l'utilisation d'une factorisation ILUT (voir chapitre 1, page 22) à l'intérieur des schémas de remplissage $\mathscr{R}_C$ ou $\mathscr{R}_S$. Cette approche peut permettre notamment de traiter plus rapidement et à moindre coût mémoire des cas tests plutôt faciles. En contrepartie, on effectue des calculs scalaires et on accepte de perdre l'efficacité des calculs par blocs utilisant les bibliothèques BLAS. Dans l'algorithme précédent, la factorisation du complément de Schur est initiée à partir du complément de Schur de manière exacte (selon le motif de remplissage $\mathscr{R}_S$ ou $\mathscr{R}_C$). Si les facteurs $\widetilde{L_S}$ et $\widetilde{U_S}$ sont creusés par une factorisation ILUT, le stockage final diminue mais le pic mémoire reste inchangé. Pour le diminuer, on n'utilise alors qu'un calcul approché du complément de Schur pour débuter la factorisation incomplète de cette matrice. Cela est généralement plus économe en mémoire mais le fait d'utiliser une approximation du complément de Schur diminue évidemment la qualité de la factorisation incomplète utilisée en tant que préconditionneur.

On note $\widehat{\widetilde{S}}$ la matrice du complément de Schur limité à $\mathscr{R}_C$ ou $\mathscr{R}_S$ et seuillée numériquement. Le calcul de $\widehat{\widetilde{S}}$ s'effectue à partir de la matrice initiale C et des contributions de W et G. La structure de $\widehat{\widetilde{S}}$ étant scalaire, il est plus efficace d'utiliser un algorithme « Left-looking » pour mettre à jour $\widehat{\widetilde{S}}$. En effet, la structure d'une matrice scalaire est allouée dynamiquement (lorsque des termes sont ajoutés) et l'approche « Left-looking » permet de retarder et grouper l'apport de contributions pour limiter les ré-allocations de la structure. Pour utiliser un tel algorithme, W et G doivent être entièrement conservés avec la factorisation de S. Comme ces matrices sont coûteuses à stocker, nous leur appliquons aussi un seuillage numérique. Lorsque le $k$ième bloc-colonne de S est factorisé, toutes les contributions de la $k$ième ligne de W et de la $k$ième colonne de S ont été reportées et ces structures peuvent être libérées.

L'algorithme s'écrit comme une factorisation globale ILUT, dans laquelle on a en plus limité le remplissage par un critère statique ($\mathscr{R}_C$ ou $\mathscr{R}_S$) :

$$A = \begin{pmatrix} L_B & 0 \\ \widehat{W} & \widehat{\widetilde{L_S}} \end{pmatrix} \times \begin{pmatrix} U_B & \widehat{G} \\ 0 & \widehat{\widetilde{U_S}} \end{pmatrix} \tag{3.2}$$

où :

- $\widehat{G}$ et $\widehat{W}$ sont les matrices G et W seuillées numériquement,
- $\widehat{\widetilde{L_S}}$ et $\widehat{\widetilde{U_S}}$ sont les facteurs de la factorisation incomplète ILUT du complément de Schur à l'intérieur de $\mathscr{R}_C$ ou $\mathscr{R}_S$.

Pour ne pas perdre en efficacité sur le calcul de $W \leftarrow E.U^{-1}$ (et $G \leftarrow L^{-1}.F$), celui-ci est toujours réalisé par blocs denses et W (respectivement G) est creusée au fur et à mesure de son calcul.

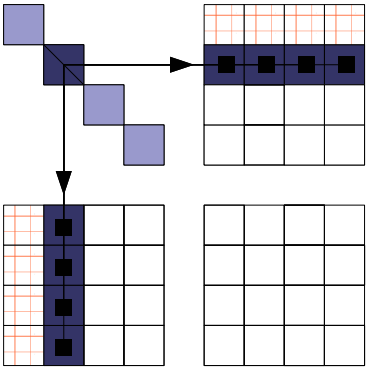
Les étapes de l'algorithme 17 deviennent alors (voir la figure 3.7 et l'algorithme 19) :

1. Factorisation exacte de $A_B = L_B.U_B$ par un algorithme supernodal et calcul d'une approximation de W et de G : dès qu'un bloc-colonne de W (bloc-ligne de G) est calculé (par un algorithme supernodal), elle est immédiatement creusée (par un seuillage numérique τ).
2. Calcul de $\widehat{\widetilde{S}}$ par un algorithme colonne guidé par la partition en sous-domaines (algorithme « Left Looking »). Un bloc-colonne de $\widehat{\widetilde{S}}$ est alloué et calculé à chaque étape de l'algorithme

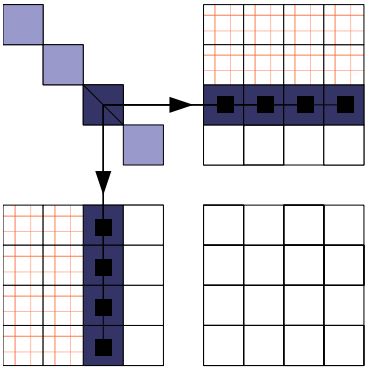
et un bloc-ligne de $\widehat{W}$ (bloc-colonne de $\widehat{G}$) est alors libéré. On factorise un bloc-colonne de S dès qu'il est calculé pour limiter le remplissage. La factorisation ILUT(τ) de $\widehat{\widetilde{S}}$ est donc effectuée au fur et à mesure du calcul de $\widehat{\widetilde{S}}$.

Légende :

Pour $k < s$ (calcul de L_B, U_B, $\widehat{W}$, $\widehat{G}$) :

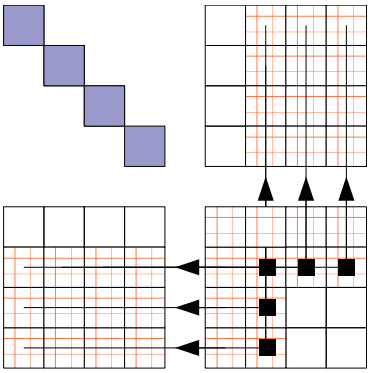


(a) Boucle $k = 2$

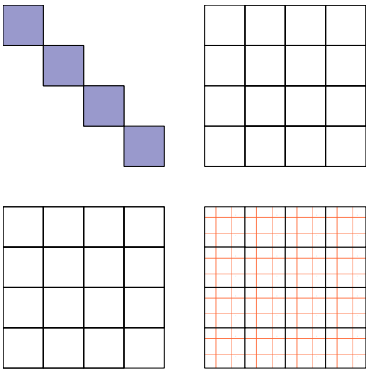


(b) Boucle $k = 3$

Pour $k \geq s$ (calcul de $\widehat{\widetilde{L_S}}$ et $\widehat{\widetilde{U_S}}$) :



(c) Factorisation d'un bloc-colonne de S



(d) Préconditionneur final

FIG. 3.7 – Illustration de l'algorithme 19

ALGORITHME 19 : Factorisation L.U selon la structure bloc induite par la partition, avec seuillage numérique (ALGO. II)

Notations :
- On note $\widehat{M} \leftarrow_\tau M$ la transformation d'une matrice à structure bloc dense en matrice scalaire, par l'application d'un seuillage numérique.
- On note l_k la liste des indices colonnes $i < k$ des blocs présents dans le bloc-ligne k.

```
/* Etape 1 : Factorisation de B, E et F                                  */
pour k = 1 à s − 1 faire
    /* Factorisation du bloc diagonal de B                               */
    Factoriser A_{k,k} en L_{k,k}.U_{k,k}

    /* Allocation des structures denses                                  */
    Allouer le bloc-colonne W_k = (L_{i,k})_{i∈l_k}
    Allouer le bloc-ligne G_k = (U_{k,j})_{i∈l_k}

    /* Calcul d'un bloc-colonne de Ŵ et d'un bloc-ligne de Ĝ             */
    pour i ∈ l_k faire
        L_{i,k} ← A_{i,k}.U^{-1}_{k,k}
        U_{k,i} ← L^{-1}_{k,k}.A_{k,i}
    pour i ∈ l_k faire
        L̂_{i,k} ←_τ L_{i,k} ; Libérer L_{i,k}
        Û_{k,i} ←_τ U_{k,i} ; Libérer U_{k,i}

/* Etape 2 : Factorisation de S : Algorithme Left-Looking                */
pour k = s à m faire
    /* Calcul des contributions apportées à la colonne k                 */
    pour i ∈ c_k faire
        pour j ∈ l_i ∩ l_k faire
            Â_{i,k} ← Â_{i,k} − L̂_{i,j}.Û_{j,k}

    /* Factorisation du bloc-colonne k                                   */
    Factoriser Â_{k,k} en L̂_{k,k}.Û_{k,k}
    pour i ∈ c_k faire
        L̂_{i,k} ← Â_{i,k}.Û^{-1}_{k,k}
        Û_{k,i} ← L̂^{-1}_{k,k}.Â_{k,i}
```

Résultats expérimentaux

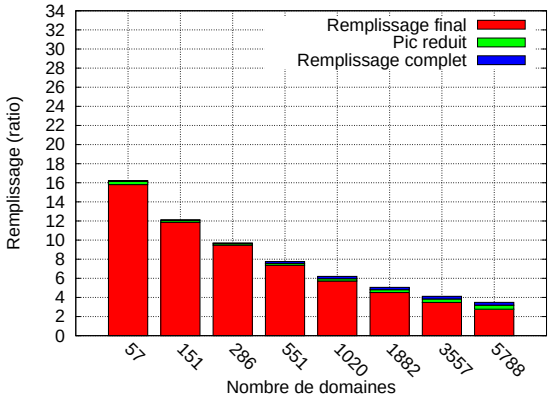
Nous étudions le gain mémoire apporté par l'algorithme 19 sur les cas tests HALTERE et AUDI. Le critère numérique est fixé à $\tau = 10^{-2}$ pour le cas test HALTERE et à $\tau = 10^{-4}$ pour le cas test AUDI. Dans le chapitre suivant, on étudiera plus précisément l'influence du choix de τ. Les histogrammes de la figure 3.8 de la page 70 présentent, pour chacun des cas tests, la mémoire nécessaire :

- au calcul du préconditionneur dans le cas où $\widehat{W}$, $\widehat{G}$, $\widehat{\widehat{L_S}}$ et $\widehat{\widehat{U_S}}$ sont stockées simultanément,
- au calcul du préconditionneur, en utilisant l'algorithme 19,
- au stockage final de ce préconditionneur.

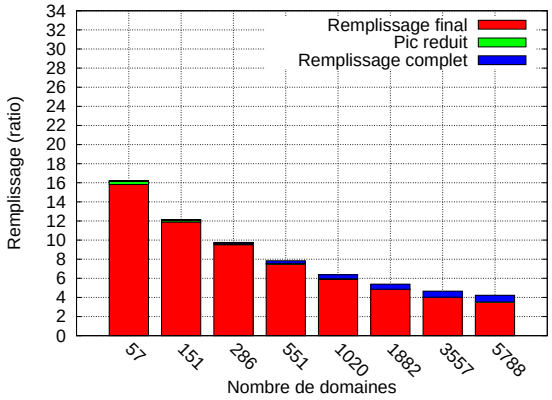
Les résultats sont présentés pour $\mathscr{R}_C$ et $\mathscr{R}_S$. Pour la matrice HALTERE, le stockage complet et le stockage final sont très proches ; cela indique que les matrices seuillées $\widehat{W}$ et $\widehat{G}$ sont très creuses et quasiment négligeables par rapport au reste de la factorisation. Pour l'AUDI, cette différence est plus prononcée. On peut aussi voir sur ce cas test que le pic mémoire atteint au cours de l'algorithme 19 est moins important par rapport au stockage final pour $\mathscr{R}_S$ que pour $\mathscr{R}_C$. Cela est dû au fait que dans le cas de $\mathscr{R}_S$, les facteurs $\widehat{\widehat{L_S}}$ et $\widehat{\widehat{U_S}}$ sont moins creux et le coût de stockage de $\widehat{W}$ et $\widehat{G}$ est recouvert par le coût de $\widehat{\widehat{L_S}}$ et $\widehat{\widehat{U_S}}$.

L'écart entre les remplissages finaux des deux motifs a été très fortement réduit par rapport aux résultats de l'algorithme 18. Cela veut dire que le nombre de termes importants (supérieurs à τ) hors du motif de remplissage $\mathscr{R}_C$ est assez faible. Cela montre que les critères combinatoires utilisés pour définir $\mathscr{R}_C$ sont pertinents. Par contre, avec un seuillage numérique, comme le pic mémoire dépend très peu du motif de remplissage choisi, il est plus intéressant d'utiliser un remplissage $\mathscr{R}_S$, afin de capturer des termes importants hors du remplissage $\mathscr{R}_C$ et afin d'augmenter la robustesse du préconditionneur à moindre coût. On s'intéresse donc au préconditionneur $\mathscr{R}_S + \tau > 0$.

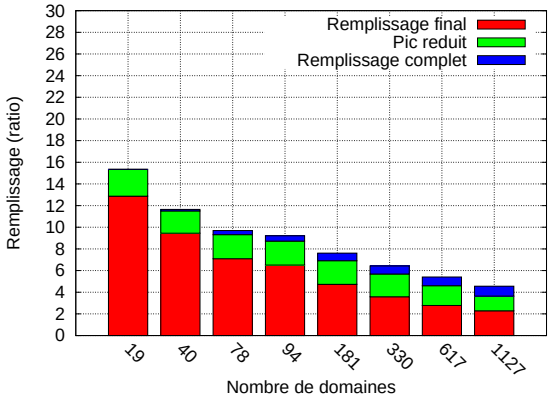
Le remplissage du préconditionneur $\mathscr{R}_S + \tau$ pour les valeurs de τ choisies est très proche de celui du préconditionneur $\mathscr{R}_C$ sans seuillage numérique. Les tableaux 3.2 de la page 70 permettent de comparer la convergence de ces deux préconditionneurs. Les prises de temps, exprimées en secondes, ont été réalisées sur des processeurs Opteron cadencés à 2,6 GHz. La version $\mathscr{R}_S + \tau > 0$ est plus robuste : elle nécessite moins d'itérations car elle capture des termes importants que l'autre méthode ignore. Le seuillage numérique permettra de traiter de très grands cas tests, dont la difficulté nécessite l'utilisation d'un motif de remplissage $\mathscr{R}_S$ mais dont la taille empêche l'exploitation de ce motif de remplissage sans seuillage numérique. Nous avons moins de calculs à réaliser mais en contrepartie, ces calculs sont scalaires et sont donc effectués moins efficacement qu'avec l'algorithme 18.



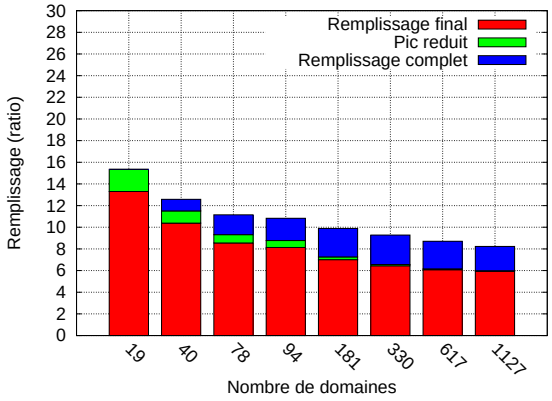
(a) Matrice HALTERE - $\mathscr{R}_C + \tau = 10^{-2}$



(b) Matrice HALTERE - $\mathscr{R}_S + \tau = 10^{-2}$



(c) Matrice AUDI - $\mathscr{R}_C + \tau = 10^{-4}$



(d) Matrice AUDI - $\mathscr{R}_S + \tau = 10^{-4}$

FIG. 3.8 – Etude du pic mémoire de l'algorithme 19

TAB. 3.2 – Comparaison entre l'algorithme 18 et l'algorithme 19.

(a) Matrice HALTERE

	$\mathscr{R}_C + \tau = 0$		$\mathscr{R}_S + \tau = 10^{-2}$	
# dom.	Temps	It.	Temps	It.
57	251,64	6	213,69	8
151	163,53	8	142,44	9
286	130,26	11	114,24	10
551	107,73	13	94,60	11
1020	95,28	18	104,18	12
1882	107,19	28	170,33	13
3557	147,90	41	227,73	15
5788	207,08	52	345,14	16

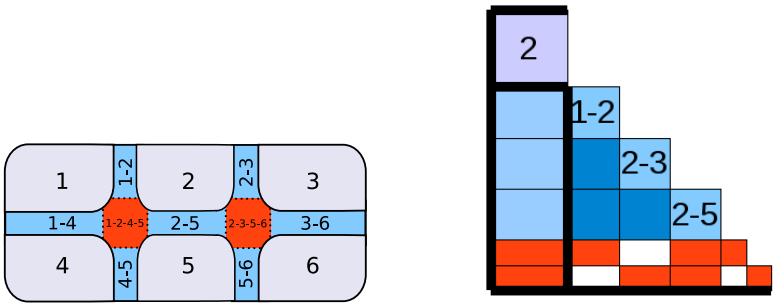
(b) Matrice AUDI

	$\mathscr{R}_C + \tau = 0$		$\mathscr{R}_S + \tau = 10^{-4}$	
# dom.	Temps	It.	Temps	It.
19	1048,09	67	926,01	35
40	702,20	85	885,85	41
78	550,35	110	947,91	51
94	428,77	96	909,67	54
181	354,76	112	1069,47	62
330	327,70	132	1294,94	72
617	379,31	160	1905,50	76
1127	554,66	184	3163,19	78

3.2 Parallélisation et équilibrage de charge

Dans cette partie, nous nous intéressons à la parallélisation des algorithmes 18 et 19. La factorisation globale de la matrice A a été construite spécifiquement pour tirer parti du parallélisme des méthodes de type décomposition de domaine. Ainsi, le calcul de la factorisation directe de B est local à chaque sous-domaine. Un algorithme supernodal séquentiel de factorisation directe est donc suffisant. De même, le complément de Schur, exact ou approché, se calcule localement pour chacun des sous-domaines. Ces étapes de calcul ne requièrent donc aucune communication. Seul le calcul parallèle de la factorisation $\widetilde{L_S}, \widetilde{U_S}$ en blocs denses ou en ILUT nécessite la mise en place d'un algorithme de factorisation parallèle.

La renumérotation de la factorisation incomplète et les schémas de remplissage ($\mathscr{R}_C$ et $\mathscr{R}_S$) ont été choisis spécifiquement pour limiter le remplissage des facteurs $\widetilde{L_S}, \widetilde{U_S}$ aux matrices locales aux sous-domaines. Les nœuds de l'interface sont dupliqués entre les matrices locales concernées (recouvrement des domaines). Les matrices locales ont une structure bloc induite par la décomposition hiérarchique en connecteurs. La figure 3.9 décrit la structure d'une telle matrice. Les blocs diagonaux correspondent aux inconnues des connecteurs qui sont dans le sous-domaine et les blocs extra-diagonaux correspondent au couplage entre les connecteurs. Ce couplage n'intervient qu'entre des sous-domaines adjacents, grâce aux schémas de remplissage choisis. La factorisation parallèle ne nécessite donc que des communications entre sous-domaines voisins qui partagent une interface.



(a) Décomposition de domaines

(b) Matrice locale au sous-domaine 2

FIG. 3.9 – Décomposition de domaines et matrices locales ($\mathscr{R}_C$)

La factorisation parallèle nécessite un ordre global d'élimination des connecteurs. L'élimination des connecteurs doit en effet être réalisée niveau par niveau et un ordonnancement de l'élimination des connecteurs à l'intérieur d'un niveau est nécessaire dans le cas de $\mathscr{R}_S$. Cela a été décrit précédemment dans le chapitre 2, page 37.

Dans une méthode de type décomposition de domaine, on associe classiquement à chaque sous-domaine un processeur différent. Mais il est intéressant de choisir le nombre de sous-domaines indépendamment du nombre de processeurs afin de toujours utiliser un nombre de sous-domaines correspondant au « meilleur » compromis entre la mémoire nécessaire à la résolution et la vitesse de convergence. Dans notre cas, la robustesse du préconditionneur nous permet d'utiliser de petits sous-domaines (la taille d'un sous-domaine est alors de quelques centaines à quelques milliers de nœuds) pour diminuer les besoins mémoire en dégradant peu la convergence. Nous associons donc en général plusieurs sous-domaines à chaque processeur.

La distribution des sous-domaines est effectuée en appliquant un algorithme de partitionnement au graphe quotient G induit par la décomposition de l'interface en connecteurs (voir fi-

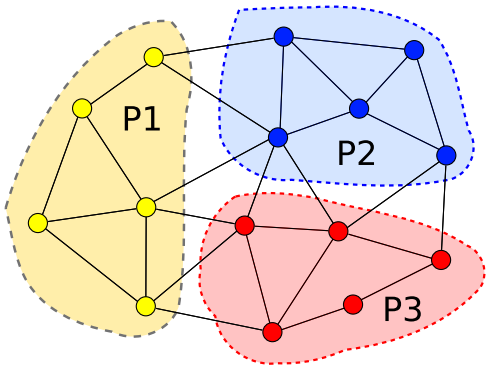


FIG. 3.10 – Répartition des sous-domaines entre les différents processeurs

gure 3.10). Le graphe $Q(V, E)$ est défini par les propositions suivantes :

- Les sous-domaines sont les sommets de V. Ils sont pondérés par le nombre de non-zéros induit par la factorisation directe du sous-domaine (ce nombre est obtenu par la factorisation symbolique).
- Il existe une arête (i, j) dans E lorsqu'il y a un connecteur entre les sous-domaines i et j. Les arêtes sont pondérées par le nombre de nœuds du connecteur correspondant.

On obtient une partition de Q sur P processeurs grâce à un partitionneur de graphe. La répartition équilibre alors la plus grande partie du remplissage entre les processeurs (remplissage créé par la factorisation directe de l'intérieur des sous-domaines) et minimise la coupe de la partition entre les processeurs (limite les communications). Il est à noter que l'équilibrage ainsi effectué est axé sur la mémoire qui est le critère limitant et non sur la charge de calcul pendant la construction du préconditionneur. L'équilibrage de charge pendant la phase de résolution est lui assez bien réalisé par cette méthode car il est linéaire par rapport au nombre de non-zéros dans les facteurs. Lorsqu'on utilise un seuillage numérique, on ne peut pas connaître à l'avance le nombre de non-zéros des facteurs $\widehat{\widetilde{L}}_S$ et $\widehat{\widetilde{U}}_S$. Il est alors plus difficile d'obtenir un bon équilibrage de charge, mais nous verrons dans le chapitre 4 que cet équilibrage statique est suffisant.

Les processeurs ayant des sous-domaines connexes partagent des connecteurs de l'interface. Chaque bloc de la matrice correspond à un ensemble de chemins d'élimination entre des inconnues de deux connecteurs. Un bloc est donc partagé par tous les processeurs possédant ces connecteurs. Afin de ne pas dupliquer le stockage et les calculs pouvant être réalisés par plusieurs processeurs sur les blocs partagés, on élit pour chaque bloc un processeur maître parmi l'ensemble des processeurs possibles. Le choix de ce processeur maître permet d'équilibrer la charge des calculs sur le complément de Schur. Il est réalisé par un algorithme glouton : les blocs de la matrice sont parcourus successivement et on élit, pour chaque bloc, le processeur candidat le moins chargé en mémoire au moment du parcours.

Une répartition idéale des sous-domaines dans le cas d'une grille 2D est représentée figure 3.11. A l'intérieur d'un processeur, l'élimination des sous-domaines est réalisée de manière séquentielle. La factorisation parallèle ne requièrt que des communications entre des sous-domaines voisins placés sur des processeurs différents.

L'algorithme 20 décrit la factorisation parallèle du complément de Schur. L'algorithme est écrit sous la forme d'une factorisation « Left-Looking » qui débute à la première colonne du complément de Schur. Sa description correspond donc exactement à la factorisation ILUT du complément de Schur de l'algorithme 19 (figure 3.7(c)). La factorisation par blocs denses du

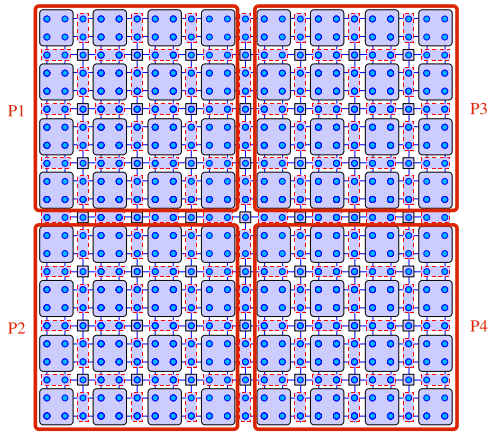


FIG. 3.11 – Répartition des sous-domaines entre les processeurs, dans le cas d'une grille régulière 2D

complément de Schur de l'algorithme 18 peut aussi s'écrire sous cette forme. Dans cette variante de l'algorithme, les contributions de W et G sont calculées avant le début de la factorisation de S. La boucle i des lignes 2-3 n'est alors réalisée que pour les bloc-colonnes de $\widetilde{L_S}$ (et bloc-lignes de $\widetilde{U_S}$). Il en va de même pour la boucle des contributions aux blocs extra-diagonaux des lignes 11-14 et la désallocation ligne 15.

Résultats expérimentaux

Dans cette partie, nous étudions expérimentalement la qualité de l'équilibrage de charge. Nous nous intéressons à la mémoire nécessaire au stockage des matrices locales du préconditionneur. Soit n le nombres de processeurs, P_i le préconditionneur local du processeur i et P_{global} le préconditionneur global. On note nnz(Mat) le nombre de non-zéros d'une matrice Mat. On a :

$$nnz(P_{global}) = \sum nnz(P_i) \tag{3.3}$$

Idéalement, les processeurs se partagent équitablement le coût mémoire du stockage du préconditionneur global. Dans ce cas,

$$\forall i, nnz(P_i) = \frac{nnz(P_{global})}{n} \tag{3.4}$$

Pour mesurer la qualité de l'équilibrage mémoire, on compare la mémoire nécessaire au processeur le plus chargé à l'équilibrage idéal de l'équation (3.4), sous la forme du ratio :

$$\alpha = \max_i \, nnz(P_i) \, / \Big(\frac{nnz(P_{global})}{n} \Big) \tag{3.5}$$

Lorsque $\alpha = 1$, l'équilibrage mémoire est idéal.

Les tableaux 3.3 présentent ce ratio α pour les cas tests HALTERE et AUDI. Nous utilisons la bibliothèque de partitionnement SCOTCH pour réaliser la répartition des sous-domaines sur les processeurs. Nous faisons varier le nombre de domaines et le nombre de processeurs. Les résultats sont présentés pour l'algorithme par blocs denses et pour la version ILUT. L'équilibrage de charge obtenu par la méthode décrite précédemment est dans les deux cas très bon. On peut le vérifier par exemple sur le cas test HALTERE. A nombre de sous-domaines fixé, lorsqu'on diminue le

Notations :
- N est le nombre de bloc-colonne de la matrice locale (cf. figure 3.9).
- N_1 est le numéro de la première colonne du complément de Schur local.
- p est le numéro du processeur pour lequel l'algorithme est décrit.
- $C = (A_{i,j})_{s \leq i \leq m,\, s \leq j \leq m}$
- $ProcSet(i,j)$ est l'ensemble des processeurs partageant le bloc $A_{i,j}$. Les ensembles $ProcSet(i,j)$ découlent directement du partitionnement du graphe Q.
- $\texttt{leader}(E)$ est une fonction qui élit un processeur dans un ensemble E de processeurs candidats.

FIG. 3.12 – Notations de l'algorithme 20

ALGORITHME 20 : Factorisation incomplète parallèle du complément de Schur selon $\mathscr{R}_C$ ou $\mathscr{R}_S$

pour $k = N_1$ *à* N **faire**
  pour $i = 1$ *à* $k-1$, *tq* $(k,i) \in \mathscr{R}$ *et* $p = \texttt{leader}(ProcSet(k,i))$ **faire**
    $C_{k,k} \leftarrow C_{k,k} - A_{k,i}.A_{i,k}$
  si $p = \texttt{leader}(ProcSet(k,k))$ **alors**
    Recevoir et **Ajouter** à $A_{k,k}$ tous les $C_{k,k}$ des processeurs $\in ProcSet(k,k)$
    Factoriser $A_{k,k}$ en $L_k.U_k$
    Envoyer $\{L_k, U_k\}$ aux processeurs $\in ProcSet(k,k) \setminus \{p\}$
  sinon
    Recevoir L_k, U_k de $\texttt{leader}(ProcSet(k,k))$
  pour $i = k+1$ *à* N, *tq* $(i,k) \in \mathscr{R}$ *et* $p \neq \texttt{leader}(ProcSet(i,k))$ **faire**
    pour $j = 1$ *à* $k-1$, *tq* $(i,j) \in \mathscr{R}$ *et* $(j,k) \in \mathscr{R}$ **faire**
      si $p = \texttt{leader}(ProcSet(i,j) \cap ProcSet(j,k))$ **alors**
        $C_{i,k} \leftarrow C_{i,k} - L_{i,j}.U_{j,k}$
        $C_{k,i} \leftarrow C_{k,i} - L_{k,j}.U_{j,i}$
    Désallouer $A_{i,[1,N_1-1]}$
    si $p \neq \texttt{leader}(ProcSet(i,k))$ **alors**
      Envoyer $\{C_{i,k}, C_{k,i}\}$ à $\texttt{leader}(ProcSet(i,k))$
      Recevoir $L_{i,k}$ et $U_{k,i}$ de $\texttt{leader}(ProcSet(i,k))$
    sinon
      Recevoir et **Ajouter**
        à $A_{i,k}$ et $A_{k,i}$ tous les $\{C_{i,k}, C_{k,i}\}$ de $\texttt{leader}(ProcSet(i,k))$
      $L_{i,k} \leftarrow A_{i,k}.U_k^{-1}$
      $U_{k,i} \leftarrow L_k^{-1}.A_{k,i}$
      Envoyer $L_{i,k}$ et $U_{k,i}$ aux processeurs $\in ProcSet(i,k)$

nombre de processeurs, la qualité de l'équilibrage augmente. De même, à nombre de processeurs fixé, lorsqu'on augmente le nombre de sous-domaines, l'équilibrage s'améliore. Ce comportement était attendu.

Sur l'AUDI, la structure de l'arbre d'élimination est complexe et la décomposition de domaine obtenue par l'algorithme 15 de la section 2.1.4 est assez déséquilibrée. Par exemple, pour 181 domaines, la taille moyenne des domaines est de 6385 nœuds mais la taille des sous-domaines va de 3720 nœuds pour le plus petit à 11508 nœuds pour le plus grand. Le déséquilibre provient ici de la décomposition de domaine initiale.

TAB. 3.3 – Etude expérimentale de l'équilibrage de charge

(a) Matrice HALTERE - $\mathscr{R}_S$

# dom.	# proc. 16	32	64	128
288	1,060	1,105	1,257	1,534
549	1,041	1,074	1,185	1,290
1014	1,038	1,068	1,125	1,212
1882	1,042	1,059	1,096	1,136
3579	1,041	1,054	1,074	1,148

(b) Matrice HALTERE - $\mathscr{R}_S + \tau = 10^{-2}$

# dom.	# proc. 16	32	64	128
288	1,033	1,058	1,099	1,298
549	1,024	1,040	1,078	1,171
1014	1,023	1,041	1,052	1,115
1882	1,031	1,051	1,051	1,079
3579	1,022	1,040	1,051	1,085

(c) Matrice AUDI - $\mathscr{R}_S$

# dom.	# proc. 16	32	64	128
181	1,352	1,528	1,760	2,918
229	1,249	1,335	1,844	2,440
330	1,268	1,428	1,522	1,724
617	1,256	1,324	1,519	1,673
1127	1,309	1,395	1,479	1,802

(d) Matrice AUDI - $\mathscr{R}_S + \tau = 10^{-4}$

# dom.	# proc. 16	32	64	128
181	1,104	1,111	1,242	2,171
229	1,114	1,153	1,252	1,454
330	1,243	1,221	1,195	1,349
617	1,148	1,242	1,223	1,437
1127	1,123	1,231	1,334	1,572

Conclusion

Dans ce chapitre, nous avons présenté deux algorithmes pour construire un préconditionneur pour le système réduit du complément de Schur en cherchant à minimiser le coût mémoire. Le premier algorithme permet d'exploiter une structure bloc dense des facteurs. Comme cela a été vérifié expérimentalement, le pic mémoire peut être très fortement réduit. Le second algorithme introduit un paramètre de seuillage numérique. La factorisation a été adaptée pour prendre en compte la structure scalaire des facteurs. Une première comparaison entre les méthodes $\mathscr{R}_C + \tau = 0$ et $\mathscr{R}_S + \tau > 0$ a permis de justifier l'intérêt de ce second algorithme. Une comparaison plus complète des différentes méthodes de résolution est réalisée dans le chapitre suivant.

La parallélisation des algorithmes exploite naturellement la partition en domaines et la décomposition hiérachique de l'interface. La robustesse du préconditionneur permet d'utiliser un schéma d'équilibrage de charge fondé sur l'attribution de plusieurs sous-domaines par processeurs. Une modélisation statique sous forme de graphe d'adjacence entre sous-domaines permet d'exploiter un partitionneur de graphe pour réaliser un bon équilibrage mémoire.

Chapitre 4

Etude expérimentale et validation

Sommaire

4.1 Etude séquentielle **80**
4.2 Etude parallèle en temps et en mémoire **87**
4.3 Passage à l'échelle sur de grands cas tests industriels **91**

Fig. 4.1 – Le logo de Hips

Introduction

Nous présentons dans ce chapitre des résultats expérimentaux obtenus sur une sélection de cas tests irréguliers. Dans une première partie, nous analysons l'influence des paramètres de notre méthode de résolution grâce à une étude séquentielle. Nous comparons aussi notre approche avec une méthode classique de Schwarz additive. Une deuxième partie est consacrée aux résultats obtenus en parallèle. Nous étudions alors la capacité de notre méthode à passer à l'échelle en temps et en mémoire. Dans une troisième partie, nous nous consacrons plus spécifiquement à la résolution d'un problème industriel de très grande taille.

Les méthodes hybrides présentées dans cette thèse ont été implémentées dans la bibliothèque Hips[5] (« Hierarchical Iterative Parallel Solver »). Hips permet également d'utiliser les algorithmes de décomposition de domaine et les factorisations multi-niveaux ILUT présentées dans [47].

Ce logiciel est téléchargable sur son site internet, sous la licence CeCILL-C. Hips permet de traiter des cas tests réels symétriques et non symétriques. Une version symétrique complexe est aussi disponible.

Présentation des cas tests

Nous avons choisi ici quatre grands problèmes irréguliers pour illustrer les deux premières parties de ce chapitre :

- Le cas test Audi est une matrice symétrique réelle issue d'un problème de mécanique des structures de la collection PARASOL [6].
- Le cas test Mhd1 est une matrice non-symétrique réelle. Elle concerne un problème 3D de magnétohydrodynamique, discrétisé sur un maillage d'éléments finis composés de tétraèdres et d'hexaèdres [47].
- Les cas tests Haltere, Amande sont des matrices symétriques complexes issues de problèmes 3D d'électromagnétisme (opérateur d'Helmholtz). Ces matrices nous ont été soumises par le CEA/CESTA dans le cadre d'une collaboration scientifique.

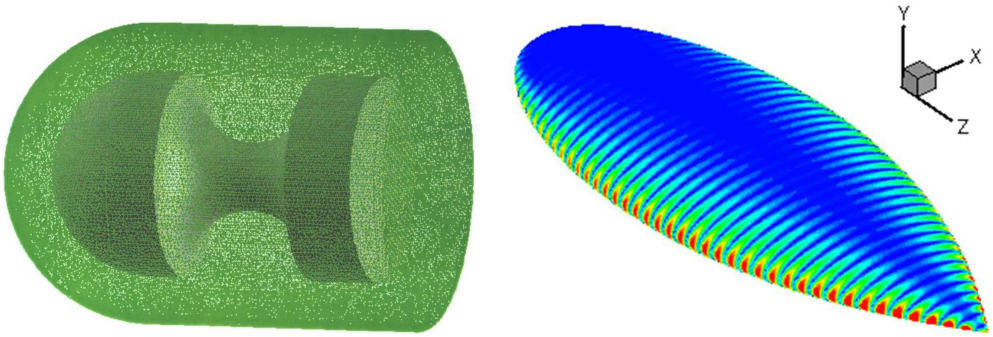
Ces cas tests sont disponibles dans la collection de matrices de GRID-TLSE[7]. Le tableau 4.1 donne les caractéristiques générales de ces matrices. Les deux dernières colonnes du tableau donnent respectivement le remplissage des facteurs et le nombre d'opérations nécessaires à une factorisation directe de la matrice renumérotée par la bibliothèque de partitionnement Scotch [66]. Les remplissages des facteurs sont donnés sous forme de ratio par rapport au nombre initial de termes non nuls dans la matrice A.

Ces cas tests sont traités de manière purement algébrique : le solveur n'a pour information que la matrice du problème. Le membre de droite de l'équation est généré pour la recherche de

[5] http://hips.gforge.inria.fr/
[6] http://www.parallab.uib.no/projects/parasol/data
[7] http://tlse.enseeiht.fr

TAB. 4.1 – Propriétés générales des cas tests

Matrices	Nombre d'inconnues	Nombre de termes	Factorisation directe	
			nnz(L,U)	Nb d'op.
AUDI	943 695	39 297 771	$28,1$	$5,3 \,.\, 10^{12}$
MHD1	485 597	24 233 141	$55,2$	$9,0 \,.\, 10^{12}$
HALTERE	1 288 825	10 476 775	$38,7$	$7,5 \,.\, 10^{11}$
AMANDE	6 994 683	58 477 383	$53,6$	$1,5 \,.\, 10^{13}$



(a) Le cas test HALTERE (b) Le cas test AMANDE

FIG. 4.2 – Illustration des cas tests du CEA/CESTA

la solution $x = \mathbb{1}$.

Conditions expérimentales

Nous ne considérons dans ce chapitre que la variante M_2 du préconditionneur (cf. section 2.2.2). Rappelons que cette variante consiste à n'itérer que sur le système réduit et à utiliser un produit de Schur $S.x$ implicite dans la résolution itérative. La méthode itérative utilisée est un GMRES. Pour simplifier l'analyse de la convergence, aucun paramètre de réinitialisation de la base de Krylov n'est ici utilisé. En pratique, bien que l'on itère sur un nombre restreint d'inconnues, il peut être préférable d'en utiliser un pour limiter la consommation mémoire de la phase itérative. La précision est mesurée par la norme résiduelle relative $||b - A.x||/||b||$.

Dans notre méthode, nous utilisons deux fois un partitionneur de graphe :

- pour construire une décomposition de domaine de manière algébrique (cf. section 2.1.4),
- pour effectuer la répartition de charge (cf. section 3.2).

HIPS permet d'utiliser indifféremment les bibliothèques de renumérotation et partitionnement METIS et SCOTCH. Les résultats présentés ici ont été obtenus en utilisant la bibliothèque SCOTCH.

L'étude séquentielle a été réalisée sur la machine de calcul `Borderline` du projet `Grid5000`[8]. Les nœuds de cette machine sont composés de quatre processeurs AMD Opteron dual-core cadencés à 2,6 GHz. L'étude parallèle a été conduite sur le supercalculateur `Jade` du GENCI-

[8] http://www.grid5000.fr

CINES[9]. Ses nœuds sont composés de deux processeurs Intel Xeon Quad-Core cadencés à 3,0 GHz. Le réseau d'interconnexion rapide de cette machine est un réseau Infiniband. Sur ces deux clusters, chaque nœud dispose de 32 Go de mémoire, ce qui permet de traiter en séquentiel l'ensemble des cas tests du tableau 4.1.

Comparaison avec la méthode de Schwarz additive

Nous avons choisi de comparer notre préconditionneur à la méthode de Schwarz additive. Ce choix est naturel car c'est la méthode algébrique classique pour coupler une résolution directe et une résolution itérative par une décomposition de domaine. De plus, PETSc [5] offre une implémentation efficace de cette méthode et permet d'utiliser le solveur direct MUMPS [2] comme solveur local. Les sous-domaines sont alors éliminés par une méthode directe efficace, utilisant une renumérotation des inconnues adaptée et effectuant les calculs par blocs. Comme pour Hips, la méthode itérative employée est un GMRES. Le remplissage mémoire correspond au coût du stockage des facteurs locaux de chaque sous-domaine. La décomposition de domaines utilisée par les deux solveurs est identique, afin d'obtenir une comparaison équitable entre les préconditionneurs. En effet, par défaut PETSc utilise un partitionnement naïf (par bandes) qui produit de très mauvais résultats sur les problèmes choisis. Le recouvrement entre les sous-domaines est de un nœud. Pour la méthode de Schwarz, des résultats complémentaires sont présentés pour un recouvrement élargi à cinq nœuds.

Une comparaison avec la méthode présentée dans [42] fera l'objet d'études futures.

4.1 Etude séquentielle

L'étude séquentielle est illustrée sur le cas test Audi. Les résultats des autres matrices sont fournis en annexe, page 95.

Historiques de convergence

Dans un premier temps, nous comparons la convergence des différentes versions du préconditionneur lorsque la décomposition de domaine est fixée. Nous nous intéressons au nombre d'itérations et au temps nécessaire pour atteindre une certaine précision. La figure 4.3 présente ainsi l'historique de convergence du cas test Audi :

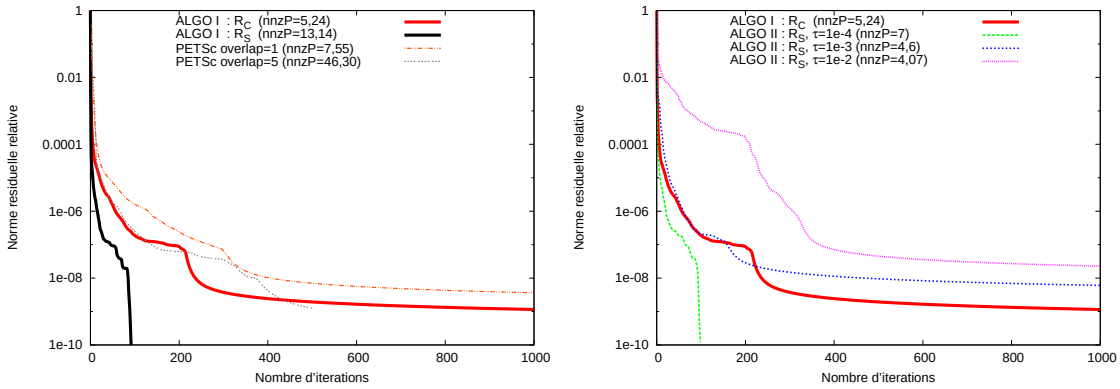
- les courbes de gauche correspondent aux résultats de la version du préconditionneur qui n'utilise aucun seuillage numérique (algorithme 18, page 61). Dans la légende, cette version du préconditionneur est notée Algo. I. Les résultats sont présentés pour un remplissage $\mathscr{R}_C$ et $\mathscr{R}_S$. Sur le même graphique sont reportés les résultats obtenus pour la méthode de Schwarz additive et pour une méthode directe.
- les courbes de droite correspondent à la version du préconditionneur qui utilise un seuillage numérique (algorithme 19, page 68). Notons Algo. II cette version du préconditionneur. Les résultats sont présentés pour différentes valeurs du seuil τ. Au vu des résultats du chapitre précédent, seule la version $\mathscr{R}_S$ est présentée. Sur le même graphique, nous reportons aussi la courbe de convergence de la méthode $\mathscr{R}_C + \tau = 0$ de la première version du préconditionneur.

Pour chaque courbe, la légende indique le nombre de non-zéros du préconditionneur, en terme de ratio par rapport au remplissage initial de la matrice. Le décalage à l'origine des courbes des graphiques 4.3(b) correspond à la durée du calcul du préconditionneur.

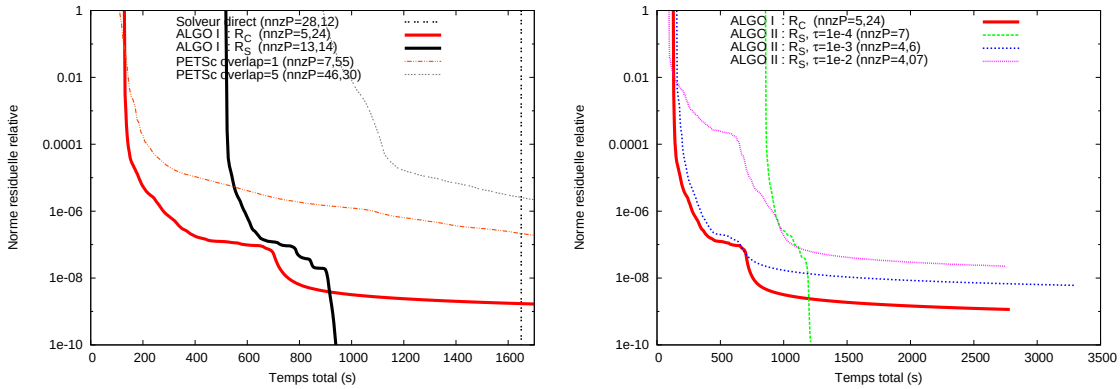
[9] http://www.genci.fr - http://www.cines.fr/

HIPS prend en paramètre une taille de domaine cible N_{cible}. Elle est ici fixée à 4000 nœuds et on obtient alors 181 sous-domaines. Avec ce paramètre, le complément de Schur représente $16,74\%$ de la dimension du problème initial. La méthode $\mathscr{R}_S+\tau=0$ est très robuste. Elle permet ici d'atteindre une précision de 10^{-10} en 92 itérations. Par contre, le calcul du préconditionneur prend beaucoup de temps. La variante $\mathscr{R}_C+\tau=0$ permet de réduire à la fois le coût mémoire et le temps de calcul du préconditionneur. En contrepartie, la précision stagne vers 10^{-8}. Lorsqu'on a besoin d'une précision plus faible, la résolution est beaucoup plus rapide avec l'approche $\mathscr{R}_C$. L'utilisation de différents paramètres de seuillage numérique influence assez fortement la convergence. Lorsque celui-ci est trop élevé, la convergence est fortement dégradée (pour $\tau = 10^{-2}$). Lorsque le remplissage autorisé est proche de celui de $\mathscr{R}_C+\tau=0$, on obtient avec ces deux variantes des résultats similaires (pour $\tau = 10^{-3}$). Avec un remplissage plus important, on retrouve un historique de convergence proche de celui de la méthode $\mathscr{R}_S+\tau=0$. Le temps de préconditionnement est plus important (car le calcul de S et de ses facteurs est scalaire) mais la méthode est très économique en mémoire : pour $\mathscr{R}_S+\tau = 10^{-4}$, le remplissage est de $7,00$ contre $13,14$ pour $\mathscr{R}_S+\tau=0$.

La comparaison des résultats obtenus avec la méthode de Schwarz additive a pour but de souligner la robustesse des préconditionneurs proposés dans cette thèse. Le cas test AUDI est relativement difficile et, avec la méthode de Schwarz additive, il s'avère impossible d'obtenir une précision de 10^{-7} en un temps raisonnable sur ce cas test. Sur des cas tests plus faciles, le comportement de la convergence de cette méthode est souvent proche de la variante $\mathscr{R}_C$. L'utilisation d'un recouvrement augmenté permet de se rapprocher du comportement de $\mathscr{R}_S$ en nombre d'itérations mais le remplissage devient prohibitif. On pourra par exemple consulter en annexes les résultats du cas test HALTERE.



(a) Historique de convergence en nombre d'itérations



(b) Historique de convergence en temps (s)

FIG. 4.3 – Matrice AUDI : historique de convergence

Influence de la taille des sous-domaines

On s'intéresse maintenant à l'influence du nombre de domaines sur les performances du solveur. La précision demandée est fixée à 10^{-7}. En fonction du nombre de sous-domaines, la figure 4.4 présente le nombre d'itérations (figure 4.4(a)) et le temps (figure 4.4(b)) nécessaire pour atteindre cette précision sur le cas Audi. Les temps de préconditionnement (temps de la factorisation incomplète) et de résolution (temps du GMRES préconditionné) sont détaillés figure 4.4(d) et figure 4.4(b).

En augmentant le nombre de sous-domaines, on réduit la proportion de résolution directe et le coût du calcul du préconditionneur. Pour l'Algo. I, si le nombre de domaines est trop important, on perd en effet BLAS en morcelant la structure du préconditionneur. Dans le cas de l'Audi, cela se remarque surtout pour le remplissage $\mathscr{R}_S$. Pour l'Algo. II, l'augmentation du nombre de sous-domaines réduit aussi le temps de préconditionnement, mais cette tendance est contrebalancée par l'augmentation de la taille du complément de Schur et donc par l'augmentation de la proportion de calcul scalaire, particulièrement quand le seuil numérique τ est faible.

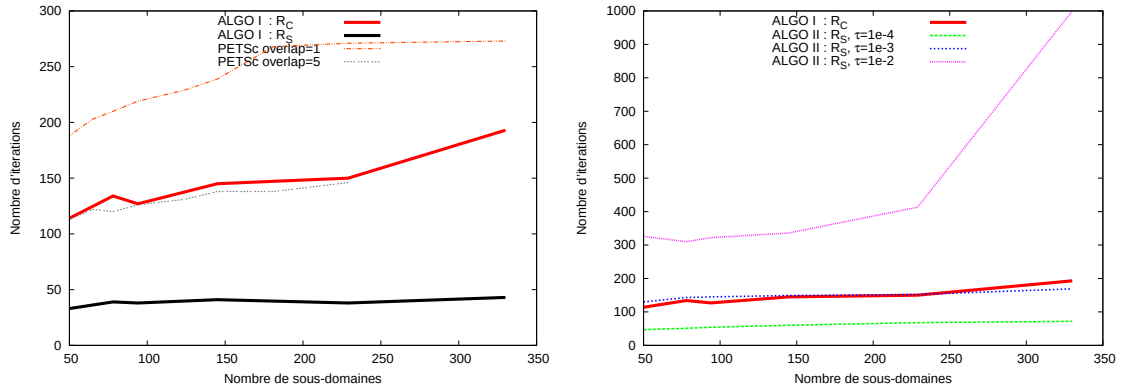
Le coût d'une itération est lié au remplissage et diminue avec la taille des sous-domaines. L'évolution du temps de calcul de la phase de résolution dépend de l'augmentation du nombre d'itérations. Comme nous l'avions déjà remarqué dans les résultats du chapitre 2, l'évolution du nombre d'itérations est asymptotiquement linéaire avec l'augmentation du nombre de sous-domaines. Avec la stratégie $\mathscr{R}_S+\tau = 0$, le nombre d'itérations de Hips croît particulièrement lentement. En passant de 50 à 330 domaines, le nombre d'itérations pour la stratégie $\mathscr{R}_S+\tau = 0$ passe par exemple de 33 à 43 dans le cas de l'Audi. Dans le cas de l'Haltere, il faut 2 itérations pour converger avec 80 domaines et seulement 3 avec 1882 domaines.

Pour un problème et une précision donnés, il existe en fait un nombre « optimal » de sous-domaines pour lequel le solveur est le plus performant en temps. Avec Hips, le nombre de domaines « optimal » pour un problème donné est bien supérieur au nombre de processeurs qu'il est raisonnable d'utiliser pour résoudre le problème. Par exemple, sur le cas test Audi, le meilleur temps séquentiel est obtenu avec le préconditionneur $\mathscr{R}_S+\tau = 0,001$ et 145 domaines. Avec ces paramètres, le remplissage du préconditionneur est de $5,12$ et il faut alors 167 secondes et 12 itérations pour atteindre une précision de 10^{-7}. Ce résultat illustre l'importance de pouvoir utiliser plusieurs sous-domaines par processeur dans le schéma de parallélisation de Hips. Un meilleur compromis temps / mémoire est obtenu avec des sous-domaines de faible taille (en comparaison à la taille du problème). Contrairement à Hips, le préconditionneur Schwarz additif est peu adapté à l'utilisation d'un nombre important de sous-domaines. Le calcul du préconditionneur est aussi très efficace mais comme le préconditionnement est moins robuste, la phase itérative est coûteuse et, pour en limiter le coût, il faut utiliser moins de sous-domaines (et donc plus de mémoire). Le meilleur temps sur le cas test Audi est obtenu avec 40 sous-domaines. Il faut alors 173 itérations et 983 secondes pour atteindre une précision de 10^{-7}. La phase itérative représente 77% du temps total et le remplissage est de $11,92$.

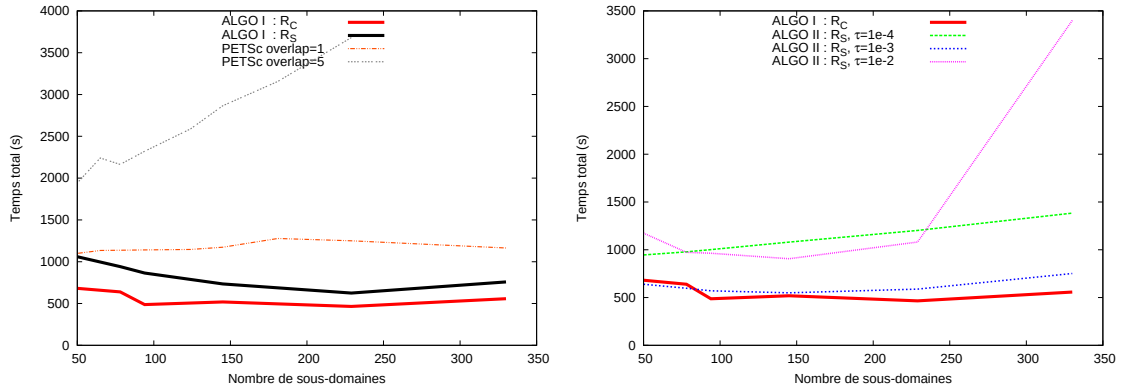
Les courbes de la figure 4.5 présentent l'influence du nombre de sous-domaines sur la mémoire nécessaire au stockage du préconditionneur (figure 4.5(a)) et sur le pic mémoire (figure 4.5(b)). Comme attendu, l'utilisation de sous-domaines de faible taille diminue la mémoire nécessaire au préconditionneur. Les courbes du pic mémoire montrent la même tendance. Le surcoût mémoire de la phase de préconditionnement correspond à la différence entre le stockage du préconditionneur et le pic mémoire. Il reste quasiment constant lorsque le nombre de domaines augmente. Lorsqu'on utilise le préconditionneur Schwarz additif avec un recouvrement augmenté, son coût mémoire augmente avec le nombre de sous-domaines en raison de l'augmentation de la taille des

interfaces.

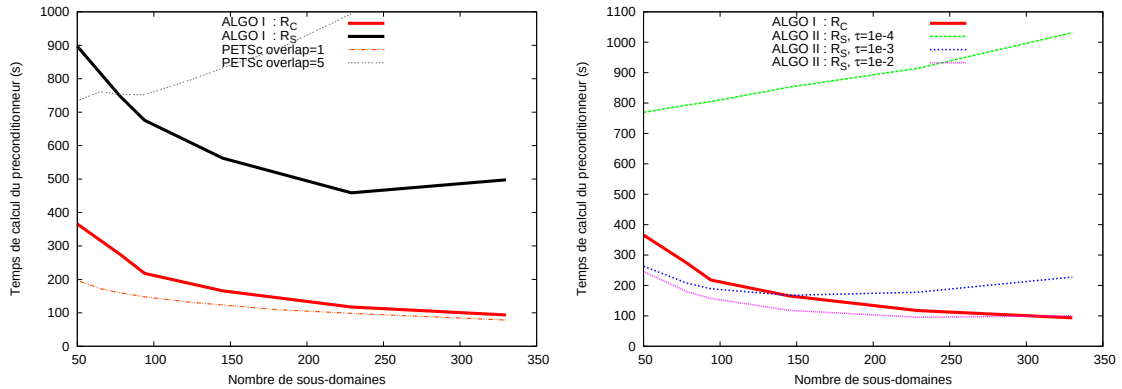
Comme le cas test MHD1 est moins difficile numériquement, les meilleurs résultats sont obtenus avec le préconditionneur le plus rapide à calculer. Il s'agit du préconditionneur $\mathscr{R}_C$ de l'ALGO. I. Pour résoudre ce problème, il faut alors 136 itérations et 124 secondes pour 274 domaines et un remplissage de $3,62$. Le cas test HALTERE peut aussi être considéré comme facile. Mais pour ce problème, il est plus intéressant d'utiliser un préconditionneur de l'ALGO. II qui réduit fortement le nombre d'itérations. Avec $\tau = 0,01$ et 551 domaines, il faut 87 secondes et un remplissage de $7,46$ pour atteindre la précision demandée. La cas test AMANDE est à la fois le cas test le plus grand et le plus difficile numériquement de cette étude. La méthode $\mathscr{R}_C+\tau = 0$ ne permet pas de converger à 10^{-7} sur ce cas test en un temps raisonnable. Avec l'algorithme ALGO. II, il faut utiliser un seuil τ élevé et les calculs scalaires demandent alors beaucoup de temps. Le plus efficace est donc d'utiliser la version $\mathscr{R}_S+\tau = 0$ de l'ALGO. I. Avec 3034 domaines, il faut alors seulement 8 itérations et 1178 secondes pour résoudre le problème, avec un remplissage de $15,41$. Ce cas test est aussi trop difficile pour la méthode additive de Schwarz. Il n'a pas été possible de résoudre ce problème avec la limite mémoire de 32 Go d'un nœud. En effet, nous sommes limités dans le choix du nombre de sous-domaines, qui doit être suffisamment grand pour que le préconditionneur tienne en mémoire et, pour un nombre important de sous-domaines, la convergence à 10^{-7} n'est obtenue que pour un nombre élevé d'itérations, ce qui nécessite le stockage d'une base de Krylov trop importante. Sur 16 processeurs, le meilleur résultat de PETSc a été obtenu pour 164 sous-domaines, en 374 secondes et pour un remplissage de $21,32$. Avec un solveur direct, il faut 1679 secondes sur 16 processeurs pour le résoudre.



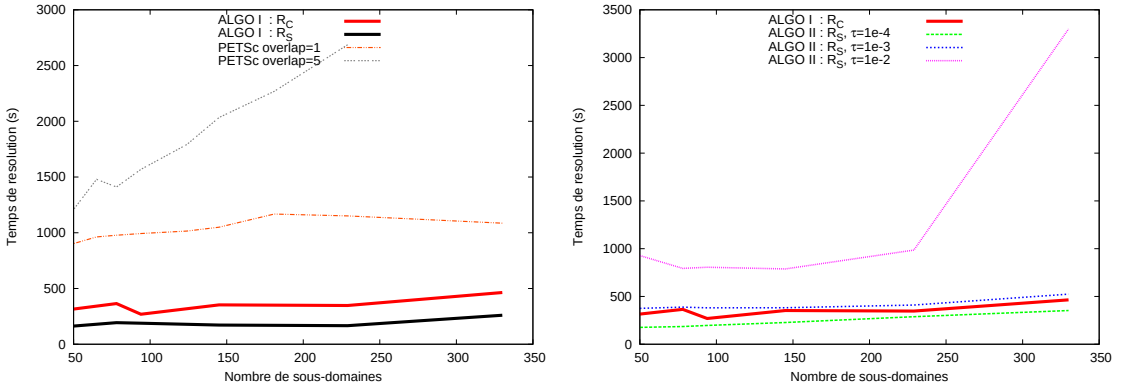
(a) Nombre d'itérations



(b) Temps total (s)

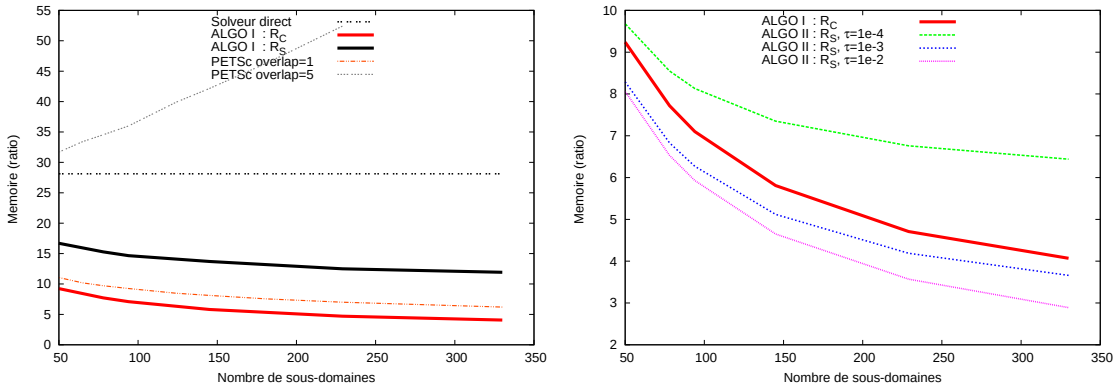


(c) Temps de préconditionnement (factorisation) (s)

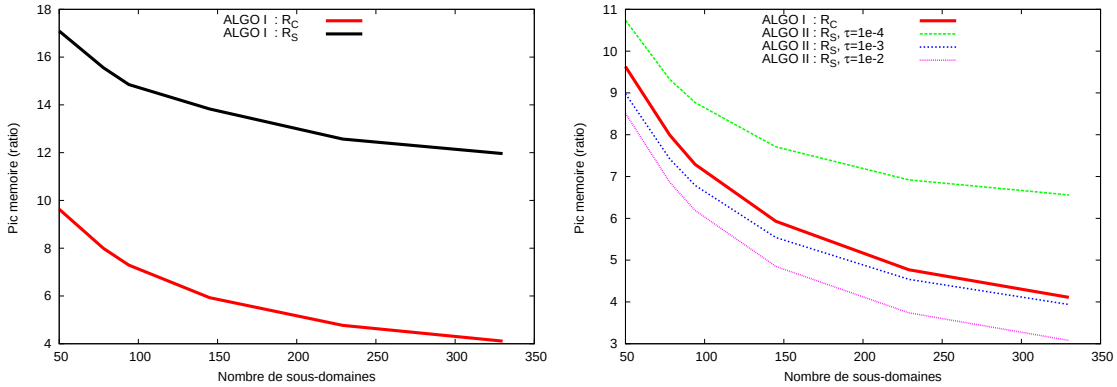


(d) Temps de résolution (s)

FIG. 4.4 – Matrice AUDI : influence du nombre de sous-domaines



(a) Remplissage du préconditionneur (ratio)



(b) Pic mémoire (ratio)

FIG. 4.5 – Matrice AUDI : influence du nombre de sous-domaines sur la mémoire

4.2 Etude parallèle en temps et en mémoire

Dans cette partie, on étudie la scalabilité en temps et en mémoire de HIPS. Le nombre de domaines est fixé. Le nombre d'itérations reste donc constant lorsque le nombre de processeurs augmente. La figure 4.6 présente la scalabilité en temps obtenue sur les quatre cas tests en parallèle. L'échelle est logarithmique. La scalabilité parfaite correspond donc à une droite. Dans le cas de l'AUDI, la taille des sous-domaines cible N_{cible} est fixée à 4000 nœuds, comme précédemment. Pour les autres cas tests, nous avons choisi une taille de 2000 nœuds. On obtient alors respectivement 133, 551 et 3034 domaines pour les cas tests MHD1, HALTERE et AMANDE. Pour chaque cas test, la scalabilité de deux variantes du préconditionneur est présentée. Elles correspondent aux variantes les mieux adaptées pour atteindre la précision de 10^{-7}, en accord avec la discussion de la section précédente. On reporte sur le même graphique les résultats de la méthode de Schwarz additive. La distribution des domaines est identique à celle utilisée dans HIPS. La décomposition de domaine est indépendante du nombre de processeurs et le nombre d'itérations reste donc constant. La méthode passe bien à l'échelle en temps et d'autant mieux que le nombre de sous-domaines est élevé par rapport au nombre de processeurs (cas test AMANDE par exemple). Il n'y a pas de différence notable entre les stratégies $\mathscr{R}_C$ et $\mathscr{R}_S$, bien qu'elles ne présentent pas le même parallélisme (cf. section 2.1.3). A partir de huit processeurs, on utilise plusieurs nœuds et le temps de communication entre les processus est plus long. Cela explique le changement de pente.

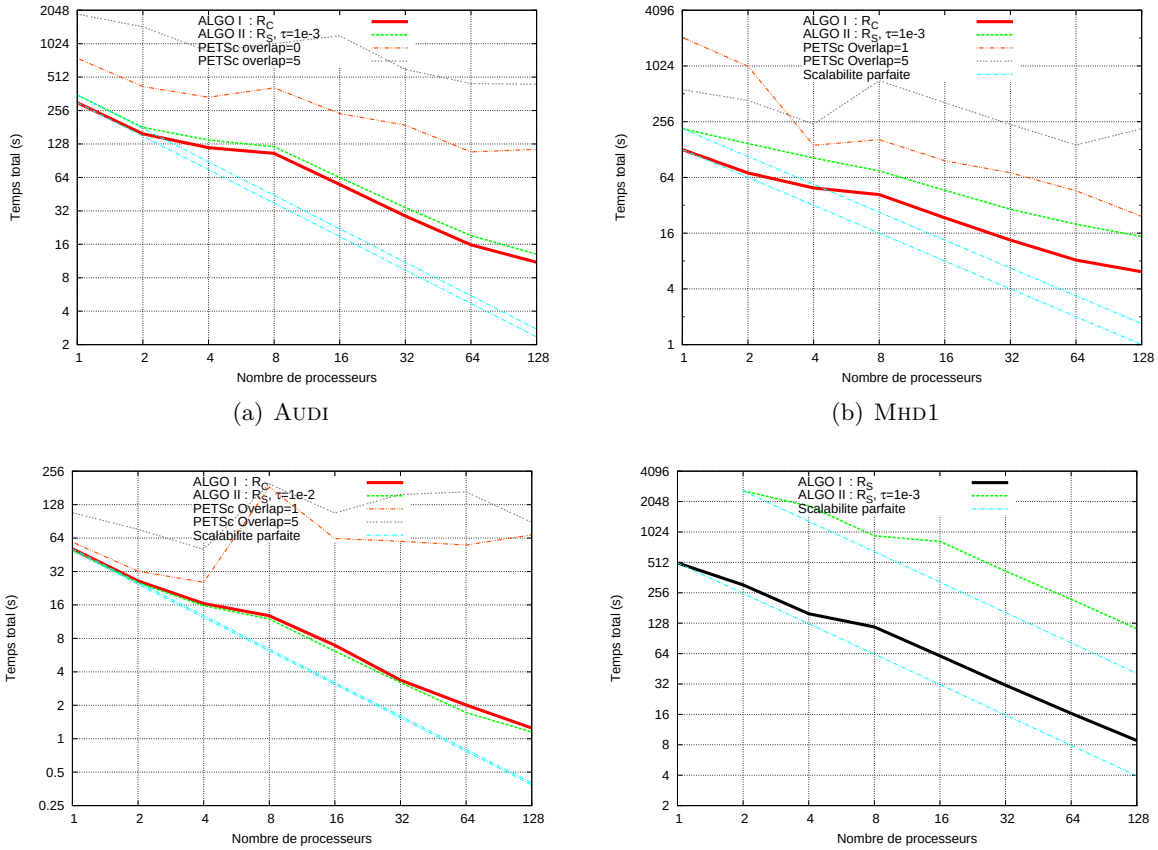
A nombre d'itérations constant, la méthode de Schwarz additive passe normalement facilement à l'échelle. Pour chaque processeur, le calcul du préconditionneur est indépendant et l'application du préconditionneur pendant la phase itérative consiste à simplement sommer les contributions de chaque sous-domaine. Les résultats de PETSc ne sont pas aussi bons qu'escomptés. Bien que PETSc offre cette possibilité, l'implémentation du Schwarz additif avec plusieurs sous-domaines par processeur ne semble pas optimisée pour cet usage particulier.

Les figures 4.7 et 4.8 concernent la scalabilité en mémoire. La figure 4.7 renseigne sur la taille des préconditionneurs locaux. On s'intéresse au processeur utilisant le plus de mémoire. La figure 4.8 présente le plus grand pic mémoire local atteint par un processeur pendant le calcul du préconditionneur (matrices temporaires incluses). La consommation mémoire est comparée dans les deux cas au coût du stockage de la matrice initiale A. Idéalement, les processeurs se partagent équitablement les coûts mémoires et en doublant le nombre de processeurs, on divise exactement par deux le ratio mémoire du processeur le plus chargé. L'approche présentée en section 3.2 favorise l'équilibrage mémoire. On constate que la scalabilité mémoire du solveur est excellente, même sur ces problèmes aux structures irrégulières. La répartition des sous-domaines bénéficie aussi à la méthode de Schwarz additive.

Le tableau 4.2 présente des résultats jusqu'à 1024 processeurs sur le cas test AMANDE. Avec 3034 domaines, il faut 9 itérations pour atteindre 10^{-7} pour la méthode $\mathscr{R}_S+\tau = 0$. On distingue le temps de préconditionnement et le temps de résolution. Chaque étape du solveur passe bien à l'échelle grâce, notamment, à un bon équilibrage des sous-domaines entre les différents processeurs. Pour mesurer la qualité de l'équilibrage mémoire, nous comparons la mémoire nécessaire au processeur le plus chargé à l'équilibrage idéal, sous la forme du ratio :

$$\beta = \max_i \, nnz(P_i) \, / \Big(\frac{nnz(A_{global})}{n} \Big) \tag{4.1}$$

où P_i est le préconditionneur local du processeur i et n est le nombre de processeurs. Idéalement, β reste constant et correspond au ratio de remplissage en séquentiel $(15,5)$. La colonne « Préc. » correspond au plus grand pic mémoire local atteint par un processeur pendant le calcul du

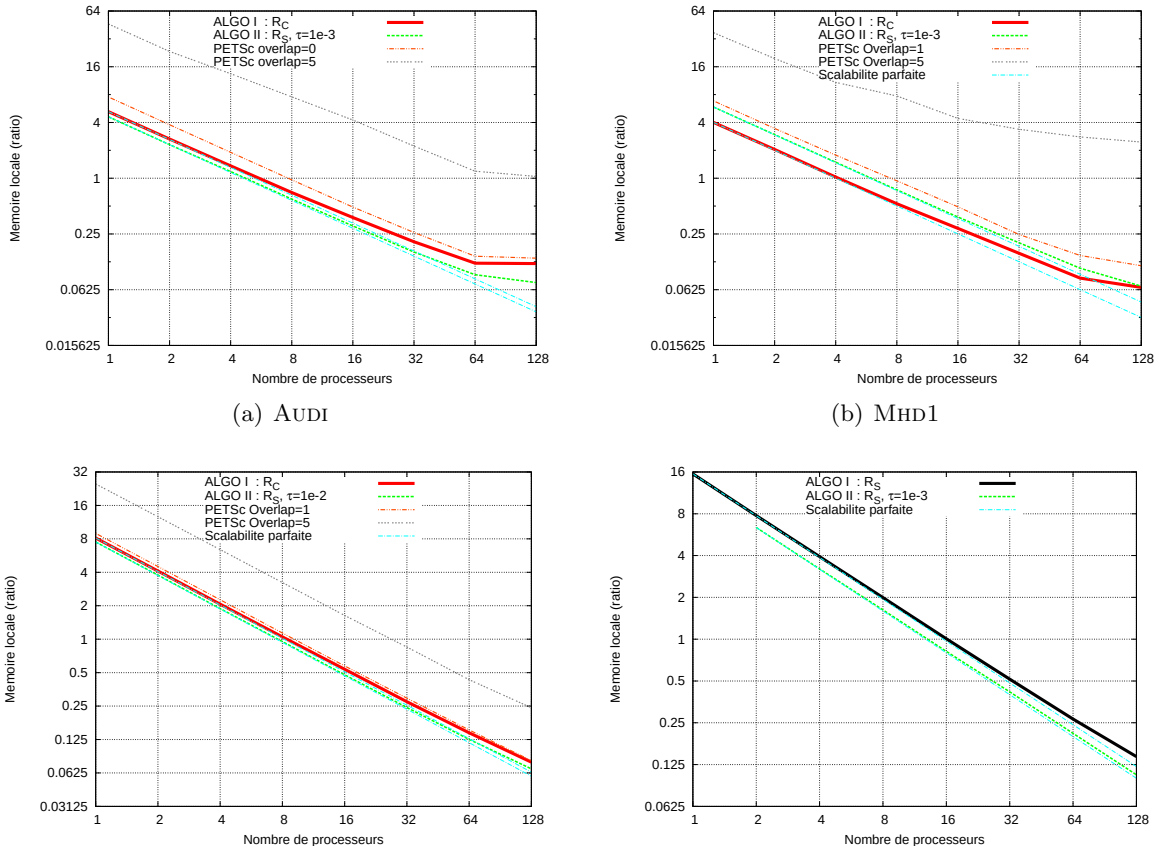


(a) Audi (b) Mhd1

(c) Haltere (d) Amande

Fig. 4.6 – Scalabilité en temps

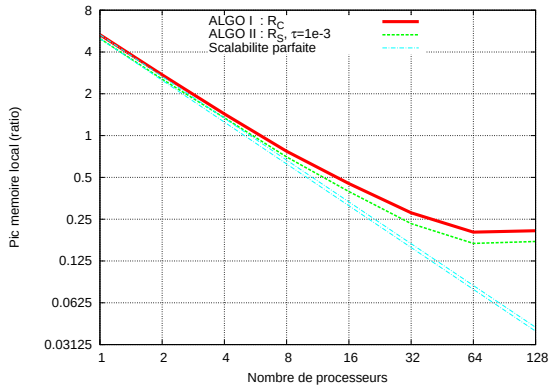
préconditionneur et la colonne « Résol. » correspond à la taille du préconditionneur local pendant la phase de résolution. On peut constater que jusqu'à 256 processeurs, l'équilibrage mémoire est quasiment parfait car β croît faiblement (si β reste constant, cela correspond à diviser par deux la taille maximale d'un préconditionneur local lorsque le nombre de processeurs double). On met ensuite en évidence le déséquilibrage de la partition initiale (comme à la page 75, tableaux 3.2). L'efficacité mémoire reste très acceptable car la taille des problèmes locaux est alors faible par rapport au nombre de processeurs.



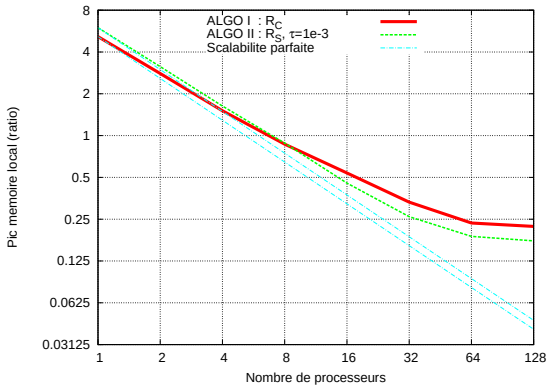
(a) Audi (b) Mhd1

(c) Haltere (d) Amande

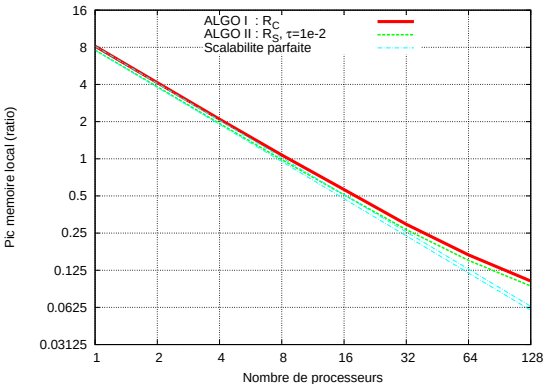
Fig. 4.7 – Scalabilité mémoire



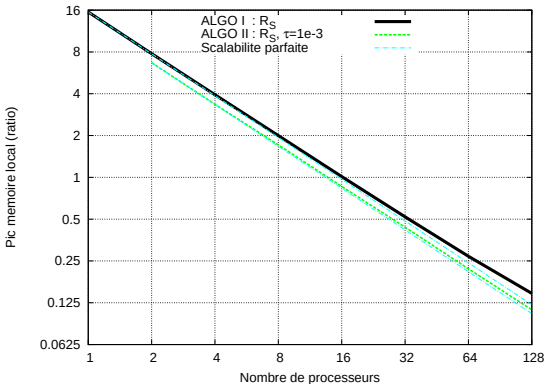
(a) Audi



(b) Mhd1



(c) Haltere



(d) Amande

Fig. 4.8 – Scalabilité du pic mémoire

Tab. 4.2 – Amande - 3034 domaines - $\mathscr{R}_S+\tau = 0$

# proc	Précond. (s)	Résol. (s)	Total (s)	Mémoire	
				Préc.	Résol.
16	35,32	25,43	60,75	16,20	16,11
32	18,45	12,72	31,17	16,63	16,50
64	9,94	6,45	16,39	17,30	17,02
128	5,59	3,24	8,82	18,70	18,27
256	3,48	1,65	5,13	19,71	18,82
512	2,99	0,84	3,83	22,55	20,43
1024	3,33	0,53	3,85	28,21	24,87

4.3 Passage à l'échelle sur de grands cas tests industriels

Nous terminons ce chapitre par une évaluation de Hips sur un cas challenge à 10 millions d'inconnues issu de notre collaboration avec le CEA/CESTA. Le cas test provient d'un code d'électromagnétisme qui travaille avec des maillages de cônes 3D. Cette simulation utilise habituellement un solveur direct pour la résolution des sytèmes linéaires sous-jacents. Les caractéristiques de ce problème symétrique à coefficients complexes sont décrites figure 4.3. Le solveur direct PaStiX [29] nécessite 104 Go de mémoire pour stocker les facteurs et ce problème est résolu en une centaine de secondes sur 512 processeurs.

Les résultats obtenus sur la machine `Jade` sont présentés tableau 4.4. Avec 512 processeurs, une vingtaine de secondes sont nécessaires pour atteindre une précision de 10^{-7}. En séquentiel, le remplissage est de $23,13$ fois la matrice initiale, ce qui correspond à $30,7$ Go. Sur 512 processeurs, un remplissage de $35,66$ correspond à n'utiliser que 94 Mo sur le processeur le plus consommateur de mémoire. Sur la dernière ligne du tableau, il y a trop peu de domaines par processeurs pour pouvoir équilibrer la mémoire entre les processeurs : le déséquilibre est alors dû à l'irrégularité initiale de la décomposition de domaines. Il est à noter que dans ce cas extrême la taille d'un sous-domaine traité par un processeur est très petite ; c'est donc la granularité de calcul plus que la méthode qui limite alors la scalabilité.

Tab. 4.3 – Propriétés du cas test 10Millions

Matrices	Nombre d'inconnues	Nombre de termes	Factorisation directe	
			nnz(L,U)	Nb d'op.
10Millions	10 423 737	89 072 871	$75,49$	$4,4\,.\,10^{13}$

Tab. 4.4 – 10Millions - 2193 domaines - $\mathscr{R}_S+\tau = 0$

# proc	Précond. (s)	Résol. (s)	Total (s)	Mémoire	
				Préc.	Résol.
16	125,01	359,91	484,92	24,12	23,94
32	65,06	181,25	246,31	24,75	24,40
64	35,42	94,57	129,99	26,27	25,61
128	19,34	48,43	67,77	27,75	26,76
256	10,20	31,78	41,97	31,13	29,22
512	6,33	17,38	23,71	38,77	35,66
1024	7,08	10,52	17,60	48,08	41,86

Conclusion et perspectives

Conclusion

Les systèmes linéaires creux apparaissent dans de nombreux problèmes de simulation de grandes tailles. L'objectif de cette thèse était de concevoir un solveur linéaire creux algébrique et parallèle capable de résoudre des cas tests difficiles et/ou avec une grande précision à moindre coût mémoire. Dans le premier chapitre, nous nous sommes attachés à effectuer un état de l'art des techniques de résolution. Les méthodes directes sont très efficaces : les calculs sont notamment réalisables par blocs denses et prédictibles. Mais elles sont consommatrices de mémoire. Les méthodes itératives n'ont pas leur robustesse mais deviennent incontournables lorsqu'il s'agit de traiter de grands problèmes 3D. Les méthodes de décomposition de domaine sont souvent utilisées, en raison de leur simplicité de mise en œuvre et du parallélisme qu'elles offrent naturellement.

L'approche envisagée a ainsi été de construire un solveur hybride direct-itératif couplant une résolution directe à une méthode itérative et réutilisant le parallélisme des méthodes de décomposition de domaine. Le domaine de calcul initial est subdivisé en sous-domaines ; l'intérieur des sous-domaines est éliminé par une méthode directe, ce qui permet de se concentrer sur la résolution du problème restreint à l'interface par une méthode itérative. De récents travaux [47] sur les factorisations multi-niveaux ILUT ont proposé de partitionner l'interface entre les sous-domaines d'une décomposition de manière à construire une élimination parallèle des inconnues. Nous avons généralisé ces travaux pour construire une décomposition de domaine utilisant une renumérotation compatible avec les solveurs directs. Nous avons ensuite présenté des algorithmes pour coupler efficacement le calcul de la factorisation exacte à l'intérieur des domaines et celui de la factorisation incomplète du complément de Schur sur les interfaces.

Nous avons décrit des algorithmes permettant de conjuguer, selon la difficulté d'un problème, à la fois des factorisations par blocs denses et des factorisations scalaires ILUT. Cela nous permet d'obtenir un bon compromis de performance et de robustesse numérique dans la majorité des cas. Nous nous sommes particulièrement attachés à diminuer la consommation mémoire de la phase de préconditionnement. Un couplage algorithmique fin permet en effet de réduire fortement le pic mémoire dû au stockage de matrices temporaires.

Une autre contribution porte sur la parallélisation de ces méthodes en utilisant plusieurs sous-domaines par processeur. Ce schéma permet d'exploiter des décompositions dont le nombre de domaines est adapté à la difficulté du problème et indépendant du nombre de processeurs envisagés pour sa résolution. Ici encore, nous offrons la possibilité d'affiner le compromis entre convergence, mémoire et performance, sans être contraint par le nombre de processeurs utilisés. Pour ces algorithmes, nous nous sommes aussi attachés à obtenir un équilibrage mémoire entre les processeurs qui permet de repousser les limites du passage à l'échelle.

Enfin, une étude expérimentale a permis de valider l'ensemble de ce travail, notamment sur de très grands cas tests irréguliers. Nous avons montré l'efficacité de cette approche en la

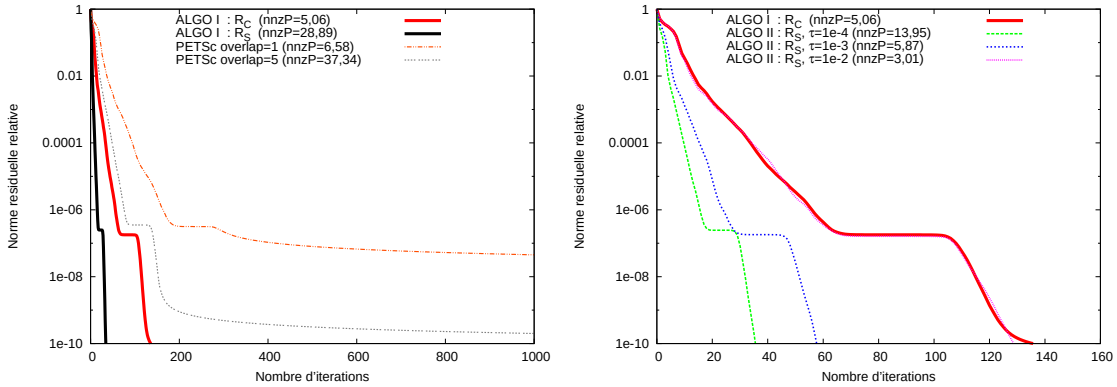
comparant à une méthode de décomposition de domaine classique. La convergence obtenue par notre méthode est beaucoup moins dépendante du nombre de sous-domaines et utilise moins de mémoire. Pour finir, nous avons fourni des résultats sur un cas test de 10 millions d'inconnues, issu d'un code de simulation d'électromagnétisme du CEA/CESTA, qui résistait jusqu'alors aux méthodes itératives classiques.

Perspectives

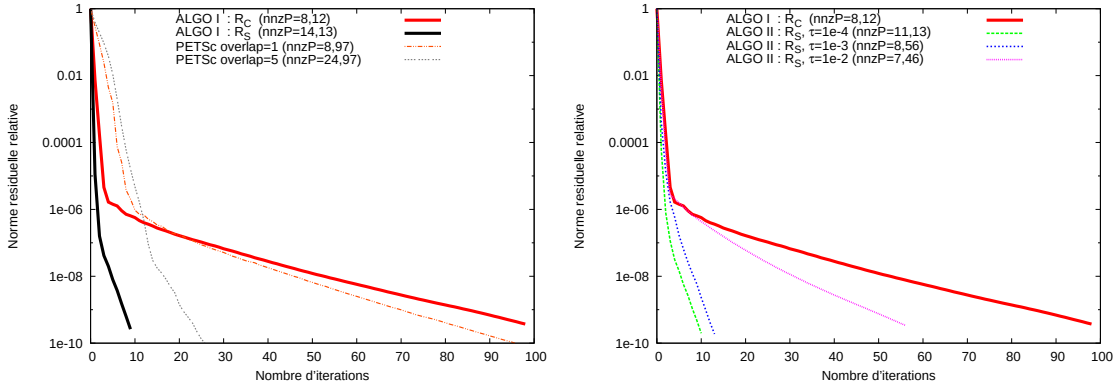
La méthode présentée dans cette thèse permet de couvrir efficacement un large spectre de problèmes car nous pouvons régler la robustesse et le coût du préconditionneur en fonction de la difficulté du problème grâce à de nombreux paramètres, comme la taille des sous-domaines, le choix du schéma de remplissage ou celui d'un seuil numérique. Une extension possible de ce travail serait de proposer un réglage automatique des principaux paramètres pour obtenir un solveur *boîte noire*. Nous pourrions par exemple nous concentrer sur l'approche par blocs denses qui ne propose pas de seuillage numérique. Une étude de la structure de l'arbre d'élimination et du remplissage induit par la taille des sous-domaines pourrait permettre de sélectionner une taille de sous-domaines adaptée au problème. Les coûts du préconditionnement et d'une itération sont prévisibles grâce à un modèle de coûts, et ce, pour les deux motifs de remplissage $\mathscr{R}_C$ et $\mathscr{R}_S$. En faisant une hypothèse sur le nombre d'itérations, nous pourrions donc trouver les paramètres qui permettent de résoudre le problème le plus efficacement possible. Cette hypothèse pourrait être faite par apprentissage (sur un ensemble de problèmes représentatifs) ou en permettant à l'utilisateur d'indiquer la difficulté de son problème.

Actuellement, la phase de prétraitement de la méthode est réalisée sur un seul processeur, en séquentiel. Pour construire la décomposition de domaine, nous utilisons en effet un partitionneur de graphe dont la renumérotation est ensuite modifiée par l'algorithme de décomposition hiérarchique. De récents travaux proposent des logiciels de partitionnements parallèles efficaces [17]. Ils permettent de distribuer le graphe d'adjacence de la matrice et donc de traiter de très grands cas tests. Pour paralléliser la phase de prétraitement de notre solveur, il suffirait donc de proposer une version distribuée de l'algorithme de décomposition hiérarchique. Pour construire la partition de l'interface en connecteurs, l'algorithme regroupe les nœuds de l'interface en fonction des sous-domaines auxquels ils appartiennent. Ensuite, seules des modifications locales de la renumérotation sont effectuées, pour garantir que les connecteurs d'un niveau jouent le rôle de séparateur pour le niveau inférieur [47]. L'algorithme offre donc un potentiel de parallélisation important.

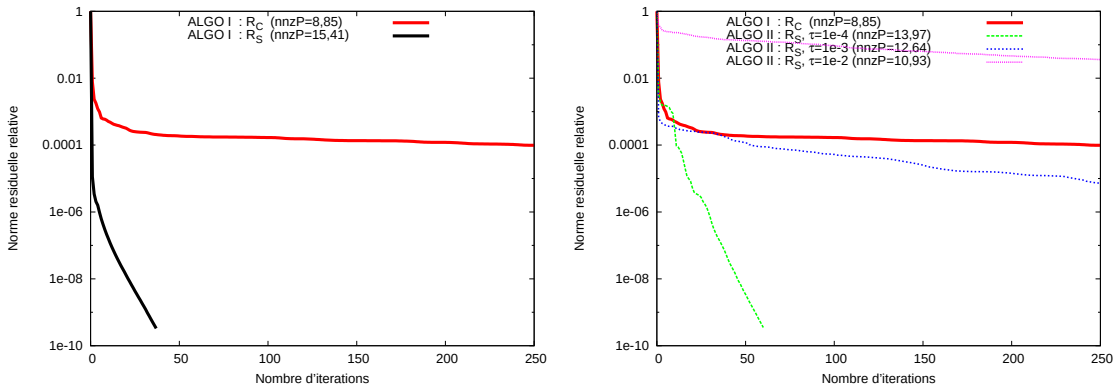
Résultats expérimentaux complémentaires



(a) Mhd1

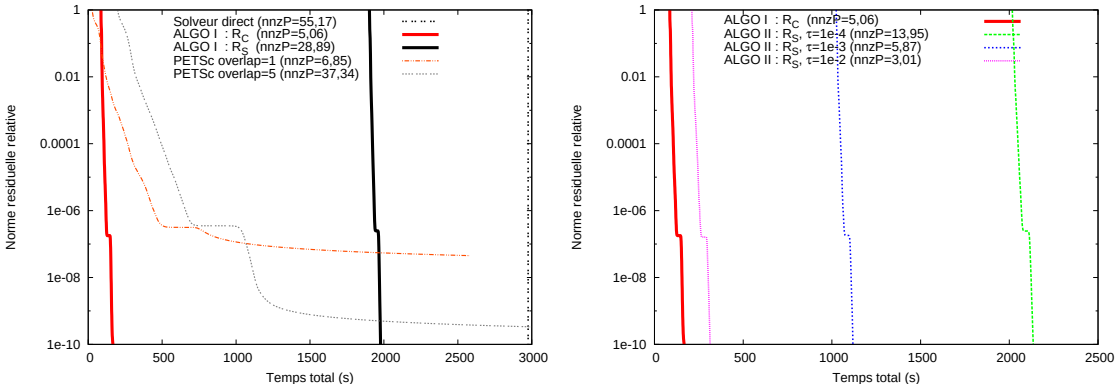


(b) Haltere

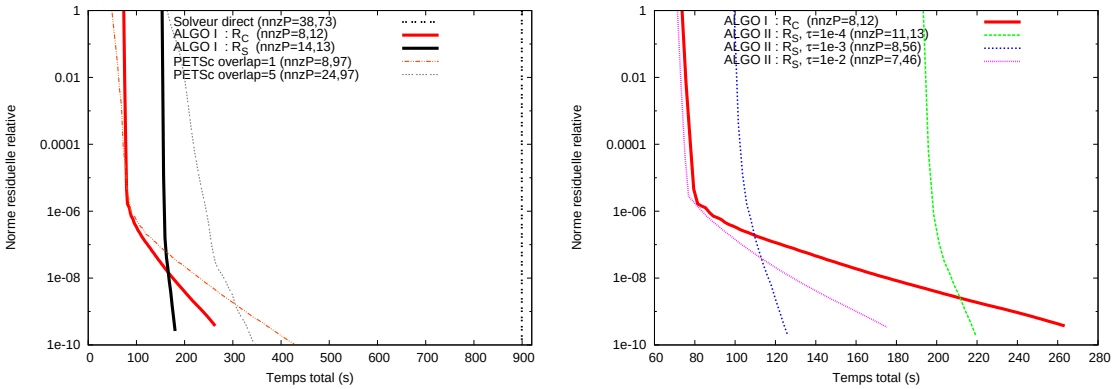


(c) Amande

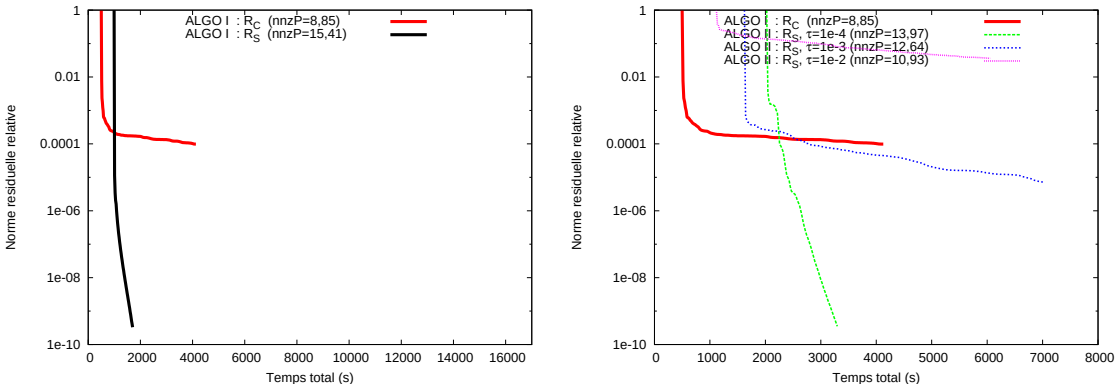
Fig. 1 – Historique de convergence à taille de sous-domaines fixée (en nombre d'itérations) - $N_{cible} = 2000$



(a) MHD1

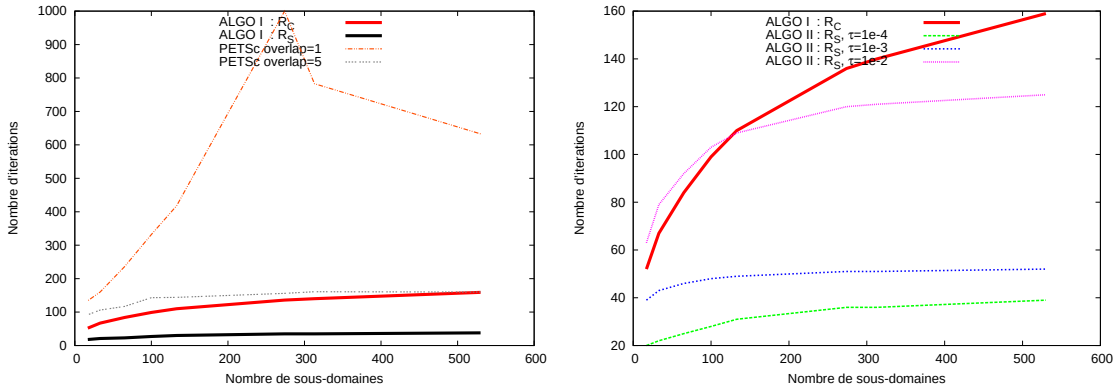


(b) HALTERE

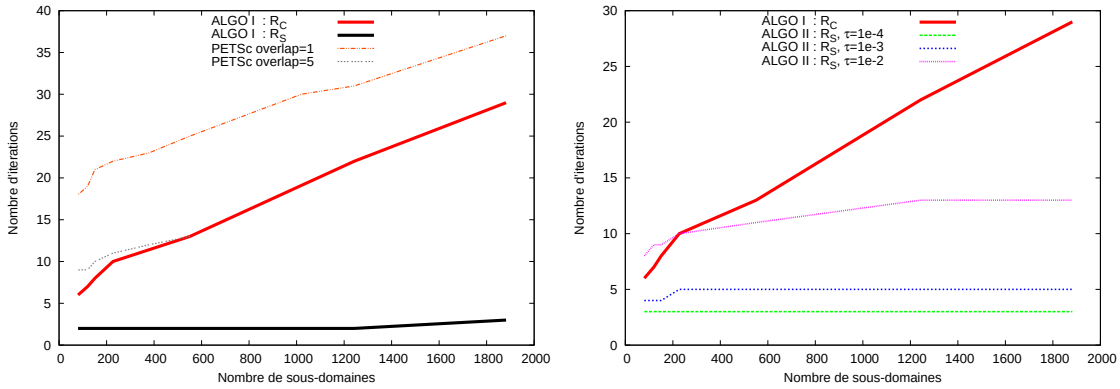


(c) AMANDE

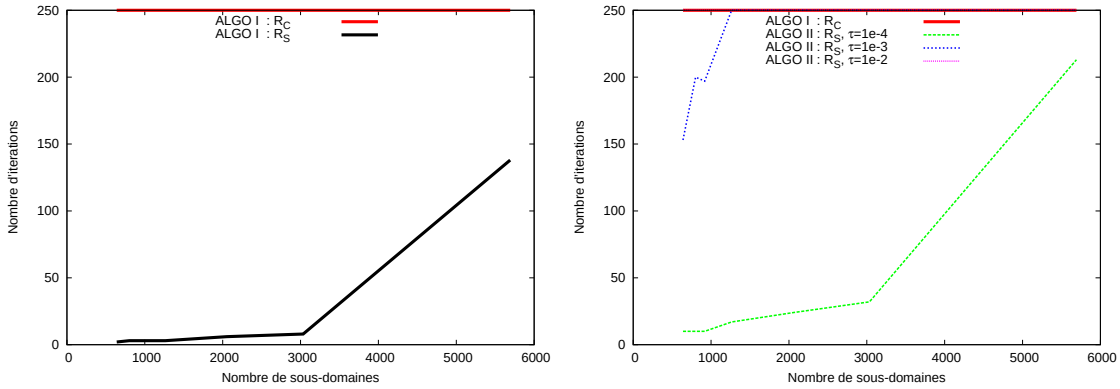
FIG. 2 – Historique de convergence à taille de sous-domaines fixée (en temps) - $N_{cible} = 2000$



(a) MHD1

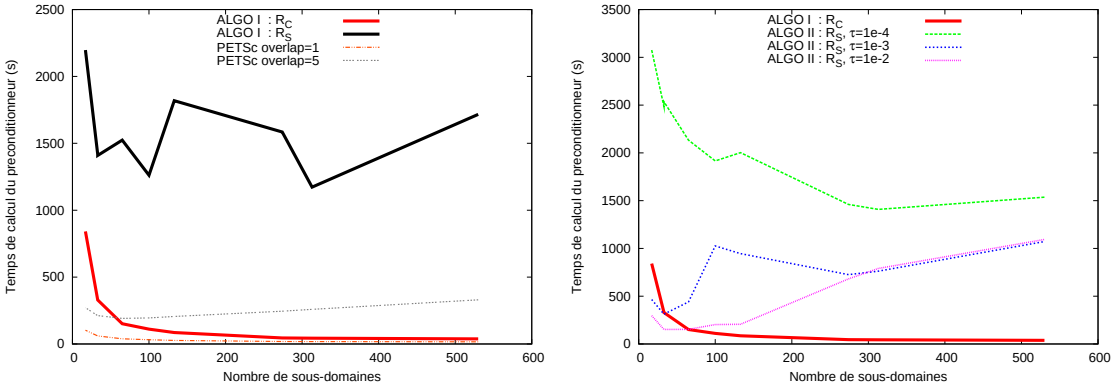


(b) HALTERE

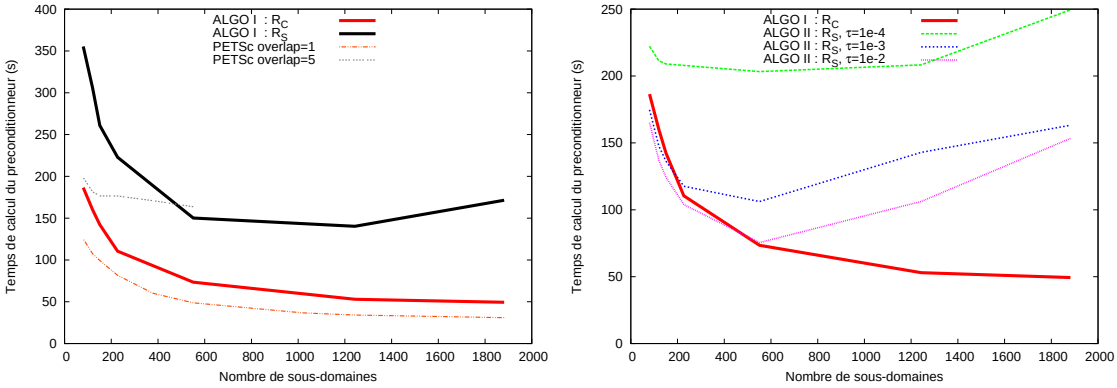


(c) AMANDE

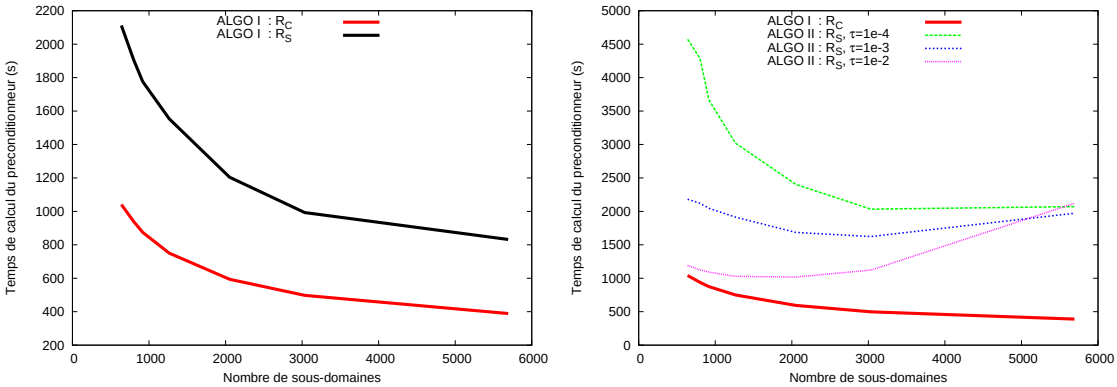
FIG. 3 – Nombre d'itérations en fonction de la taille des sous-domaines (10^{-7})



(a) MHD1

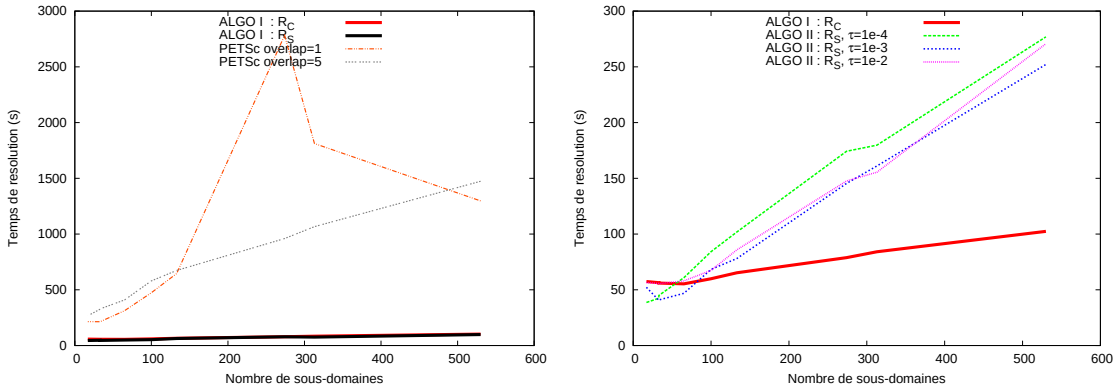


(b) HALTERE

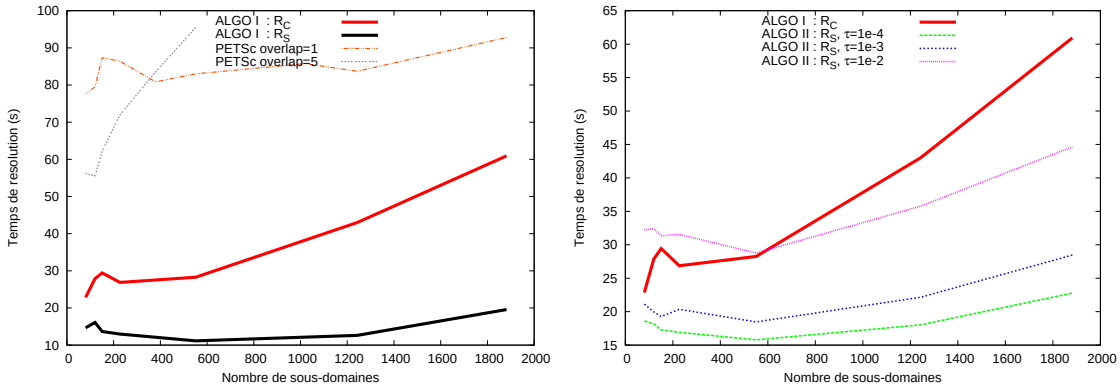


(c) AMANDE

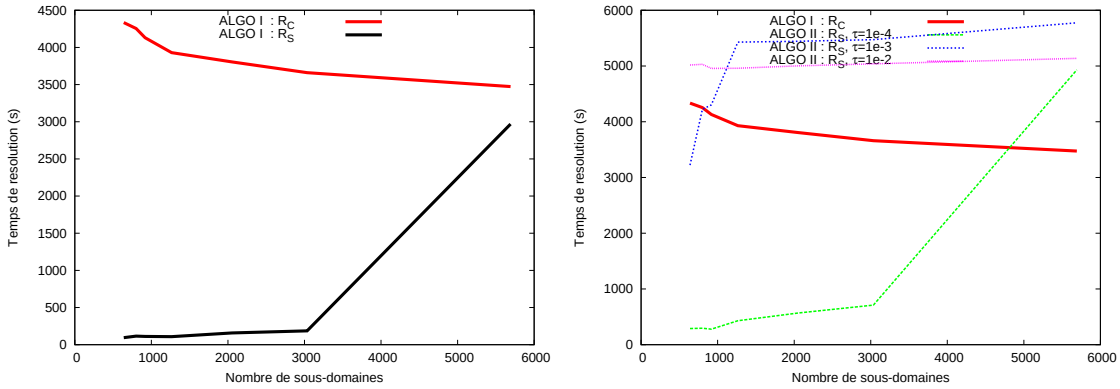
FIG. 4 – Temps séquentiel pour le calcul du préconditionneur, en fonction de la taille des sous-domaines (10^{-7})



(a) MHD1

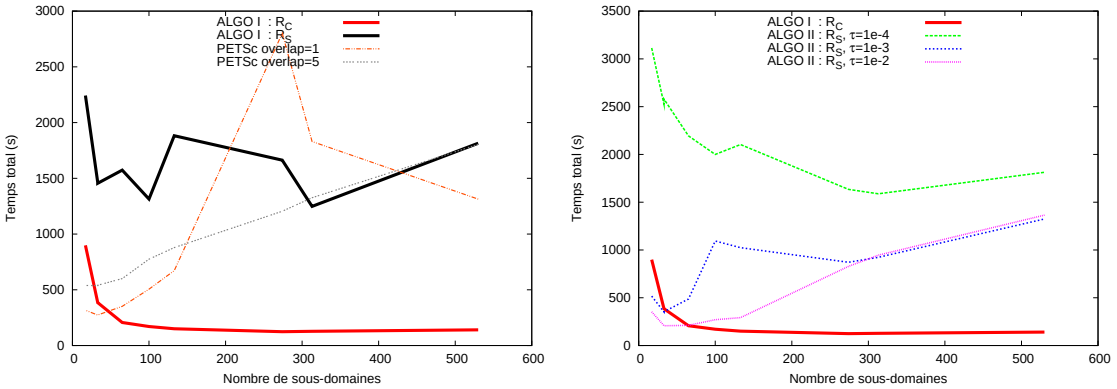


(b) HALTERE

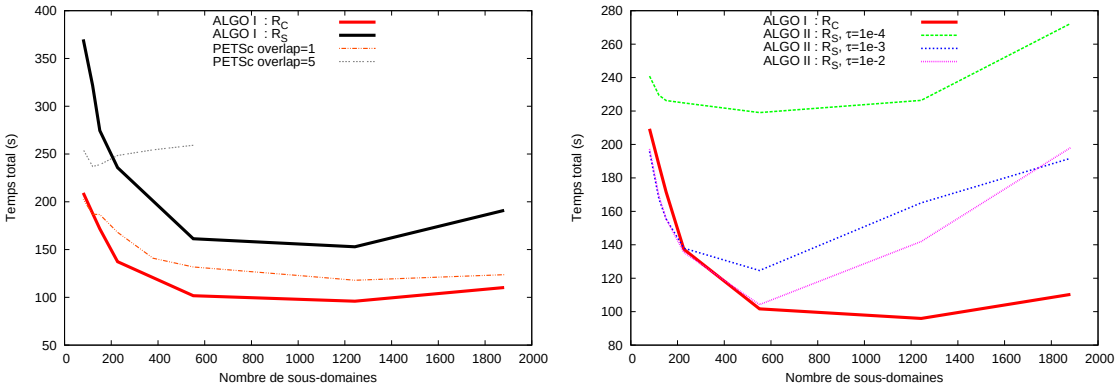


(c) AMANDE

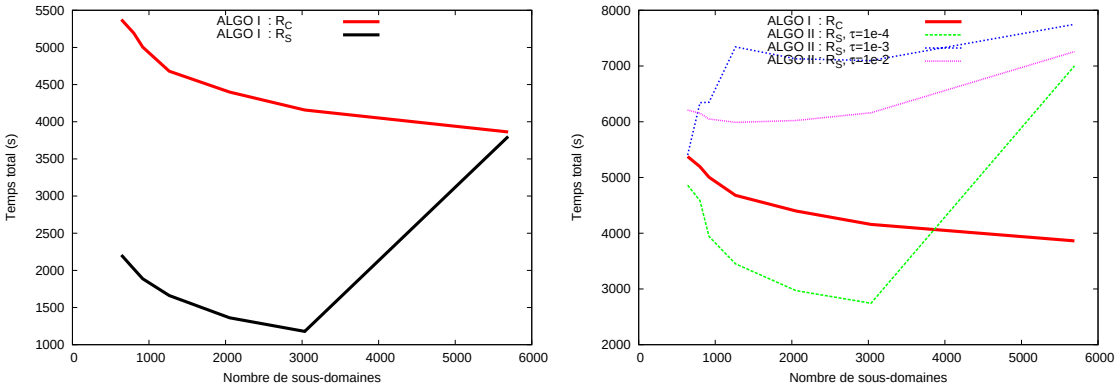
FIG. 5 – Temps séquentiel de résolution, en fonction de la taille des sous-domaines (10^{-7})



(a) Mhd1

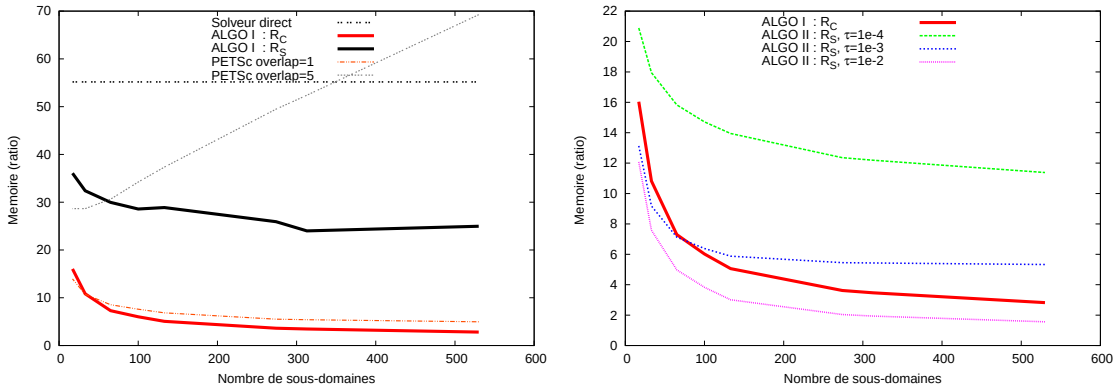


(b) Haltere

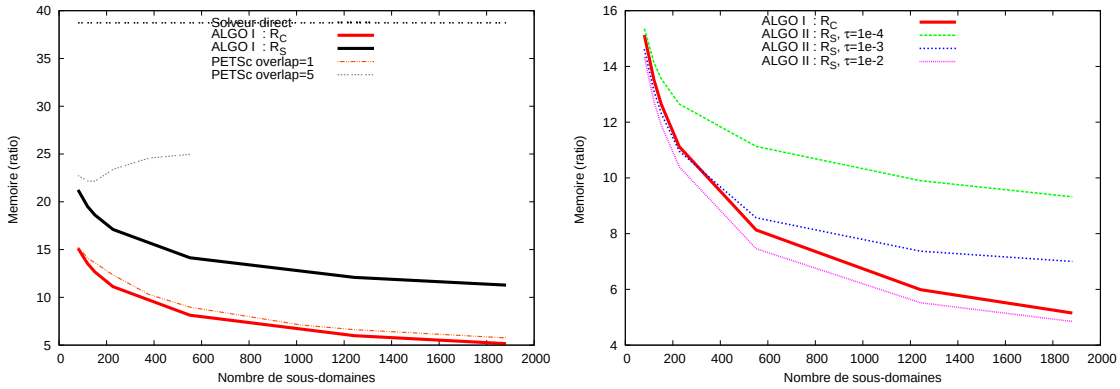


(c) Amande

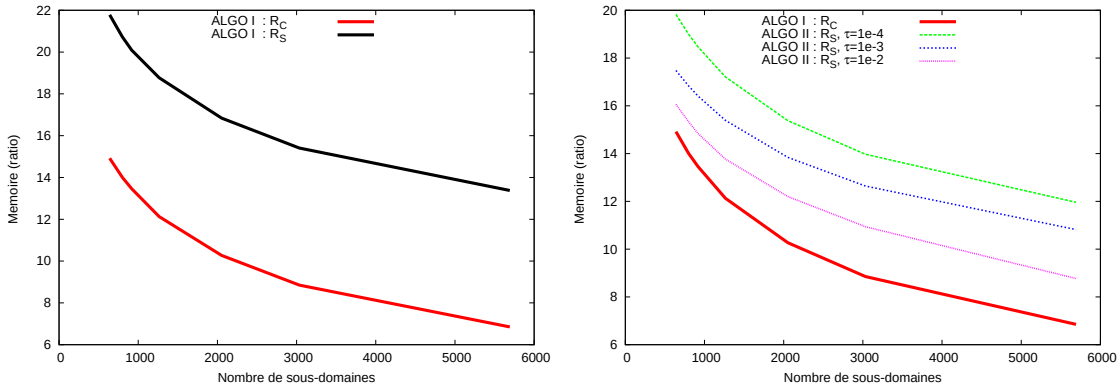
Fig. 6 – Temps séquentiel total en fonction de la taille des sous-domaines (10^{-7})



(a) MHD1

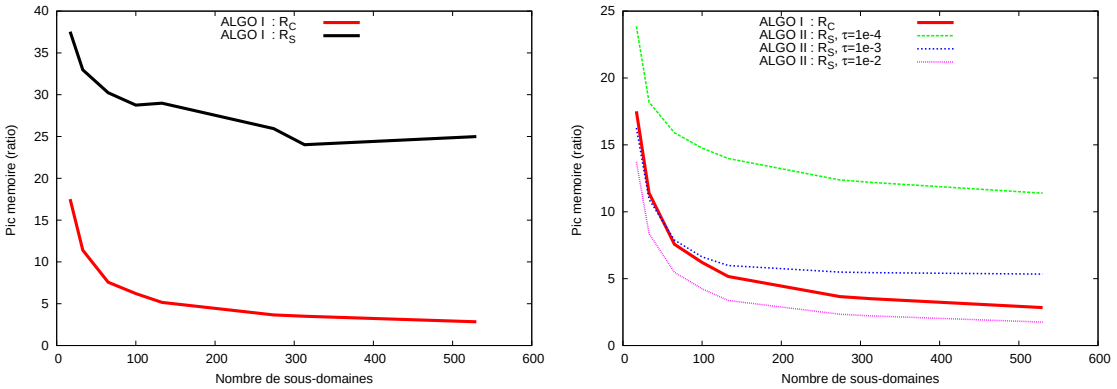


(b) HALTERE

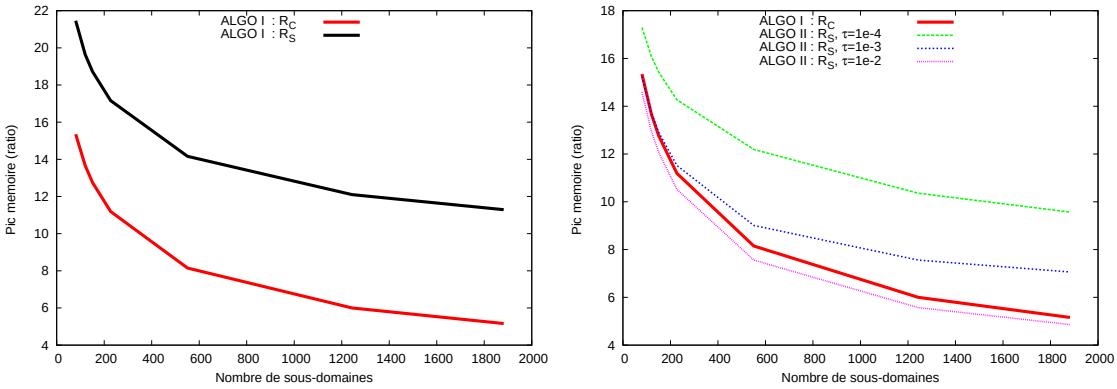


(c) AMANDE

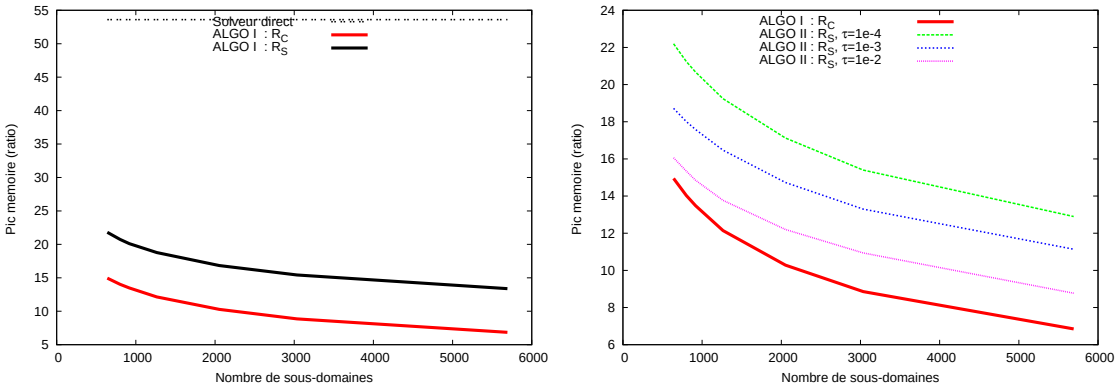
FIG. 7 – Ratio mémoire en fonction de la taille des sous-domaines (10^{-7})



(a) MHD1



(b) HALTERE



(c) AMANDE

FIG. 8 – Pic mémoire en fonction de la taille des sous-domaines (10^{-7})

Bibliographie

[1] P. R. Amestoy, T. A. Davis et I. S. Duff : An approximate minimum degree ordering algorithm. *SIAM J. Matrix Anal. Appl.*, 17(4):886–905, 1996. 8

[2] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent et J. Koster : A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM Journal on Matrix Analysis and Applications*, 23(1):15–41, 2001. 4, 13, 80

[3] C. Ashcraft : The fan-both family of column-based distributed Cholesky factorization algorithms. *Graph Theory and Sparse Matrix Computation, IMA*, 56:159–190, 1993. 13

[4] O. Axelsson : A general incomplete block-matrix factorization method. *Lin. Alg. Appl.*, 74:179–190, 1986. 23

[5] S. Balay, W. D. Gropp, L. Curfman McInnes et B. F. Smith : *PETSc home page.* Argonne Natl. Lab., Argonne, IL, 1998. 80

[6] M. Benzi : Preconditioning techniques for large linear systems : a survey. *J. Comput. Phys.*, 182(2):418–477, 2002. 5

[7] M. Benzi, J. Marín et M. Tůma : Parallel preconditioning with factorized sparse approximate inverses. *In PPSC*, 1999. 23

[8] M. Benzi et M. Tůma : A comparative study of sparse approximate inverse preconditioners. *Applied Numerical Mathematics : Transactions of IMACS*, 30(2–3):305–340, juin 1999. 18

[9] M. Bollhöfer et Y. Saad : A factored approximate inverse preconditioner with pivoting. *SIAM Journal on Matrix Analysis and Applications*, 23(3):692–705, 2002. 18

[10] J. H. Bramble : *Multigrid Methods*, volume 294 de *Pitman Research Notes in Mathematical Sciences.* Longman Scientific & Technical, Essex, England, 1993. 15

[11] J. H. Bramble, J. E. Pasciak et J. Xu : Parallel multilevel preconditioners. *Math. Comp.*, 55:1–22, 1990. 23

[12] A. Brandt : Multi-level adaptive solutions to boundary-value problems. *Math. Comp.*, 31:333–390, 1977. 15

[13] W. L. Briggs, V. E. Henson et S. F. McCormick : *A Multigrid Tutorial.* SIAM Books, Philadelphia, 2000. Second edition. 15

[14] N. I. Buleev : Numerical method for solving two- and three-dimensional diffusion equations. *Mat. Sb. (N.S.)*, 51(93):227–238, 1960. 19

[15] A. Chapman, Y. Saad et L. Wigton : High order ILU preconditioners for CFD problems. Rapport technique, mai 16 1996. 23

[16] P. Charrier et J. Roman : Algorithmique et calculs de complexité pour un solveur de type dissections emboîtées. *Numerische Mathematik*, 55:463–476, 1989. 11

[17] C. Chevalier et F. Pellegrini : PT-scotch. *Parallel Computing*, 34(6-8):318–331, 2008. 94

[18] E. Chow et M. A. Heroux : An object-oriented framework for block preconditioning. *ACM Trans. Math. Softw*, 24(2):159–183, 1998. 23

[19] E. Chow et Y. Saad : Approximate inverse techniques for block-partitioned matrices. *SIAM Journal on Scientific Computing*, 18(6):1657–1675, novembre 1997. 23

[20] E. Cuthill : *Several strategies for reducing the bandwidth of matrices*, pages 157–166. Plenum Press, New York, 1972. 29

[21] J. E. Dendy : Black box multigrid. *J. Comp.*, 48:366–386, 1982. 16

[22] J. J. Dongarra, J. Du Croz, S. Hammarling et I. S. Duff : A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 16(1):1–17, 1990. 11

[23] J. J. Dongarra, I. S. Duff, D. C. Sorensen et H. A. Van der Vorst : *Numerical Linear Algebra for High Performance Computers.* SIAM, Philadelphia, PA, USA, 1998. 6

[24] Jack J. Dongarra, J. Du Croz, S. Hammarling et R. J. Hanson : An extended set of fortran basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14(1):1–17, 1988. 11

[25] I. S. Duff : A review of frontal methods for solving linear systems. *Computer Physics Communications*, 1996. 6

[26] I. S. Duff : Sparse numerical linear algebra : direct methods and preconditioning. Rapport technique TR/PA/96/22, CERFACS, 1996. 6

[27] I. S. Duff, A. M. Erisman et J. K. Reid : *Direct Methods for Sparse Matrices.* Oxford University Press, London, 1986. 4, 7

[28] L. Facq et J. Roman : Algèbre linéaire creuse : distribution par bloc pour une factorisation parallèle de Cholesky. *In Parallélisme et applications irrégulières*, pages 135–147. Hermès, 1995. 13

[29] M. Faverge : Vers un solveur de systèmes linéaires creux adapté aux machines NUMA, 2009. 91

[30] J. Gaidamour et P. Hénon : A parallel direct/iterative solver based on a schur complement approach. *In CSE*, pages 98–105. IEEE Computer Society, 2008. 65

[31] K. A. Gallivan, M. Heath, E. Ng, B. W. Peyton, R. J. Plemmons, J. Ortega, C. Romine, A. H. Sameh et R. G. Voigt : Parallel algorithms for matrix computations. *SIAM, Philadelphia, PA*, 1990. 6

[32] G. A. Geist et E. Ng : Task scheduling for parallel sparse Cholesky factorization. *Internat. J. Parallel Programming*, 18(4):291–314, 1989. 13

[33] A. George : Nested dissection of a regular finite element mesh. *SIAM J. Numer. Anal.*, 10:345–363, 1973. 8

[34] A. George et J. W.-H. Liu : *Computer Solution of Large Sparse Positive Definite.* Prentice Hall Professional Technical Reference, 1981. 4, 6, 7, 8

[35] A. George et J. W.-H. Liu : The evolution of the minimum degree ordering algorithm. *SIAM Rev.*, 31(1):1–19, 1989. 8

[36] A. Greenbaum : *Iterative Methods for Solving Linear Systems.* SIAM, Philadelphia, PA, USA, 1997. 17

[37] A. Greenbaum, V. Pták et Z. Strakoš : Any nonincreasing convergence curve is possible for GMRES. *SIAM Journal on Matrix Analysis and Applications*, 17(3):465–469, juillet 1996. 18

[38] A. Gupta, G. Karypis et V. Kumar : Highly scalable parallel algorithms for sparse matrix factorization. IEEE Trans. on Parallel and Distributed Systems, 8(5):502–520, mai 1997. 13

[39] A. Gupta et V. Kumar : A scalable parallel algorithm for sparse matrix factorization. Rapport technique 94-19, Department of Computer Science, University of Minnesota, Minneapolis, 1994. 13

[40] I. Gustafsson : A class of first order factorization methods. *BIT*, 18:142–156, 1978. 20

[41] W. Hackbusch : *Multi-Grid Methods and Applications.* Springer, Berlin/New York, 1985. 15

[42] A. Haidar : *On the parallel scalability of hybrid linear solvers for large 3D problems.* Thèse de doctorat, Institut national polytechnique de Toulouse, France, 2008. 1, 28, 80

[43] M. T. Heath, E. Ng et B. W. Peyton : Parallel algorithms for sparse linear systems. *SIAM Rev.*, 33(3):420–460, 1991. 6

[44] P. Hénon : *Distribution des Données et Régulation Statique des Calculs et des Communications pour la Résolution de Grands Systèmes Linéaires Creux par Méthode Directe.* Thèse de doctorat, LaBRI, Université Bordeaux I, Talence, France, novembre 2001. 13

[45] P. Hénon, P. Ramet et J. Roman : PaStiX : A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing, à paraître*, 2002. 4

[46] P. Hénon, P. Ramet et J. Roman : On finding approximate supernodes for an efficient block-ILU(k). *Parallel Computing*, 34(6-8):345–362, 2008. 13, 21

[47] P. Hénon et Y. Saad : A parallel multistage ILU factorization based on a hierarchical graph decomposition. *SIAM J. Scientific Computing*, 28(6):2266–2293, 2006. 2, 32, 33, 34, 36, 47, 78, 93, 94

[48] M. R. Hestenes et E. Stiefel : Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952. 15

[49] A. S. Householder : *The Theory of Matrices in Numerical Analysis.* Blaisdell Publishing Co., 1964. 15

[50] D. Hysom et A. Pothen : Parallel computation of ILU(k) preconditioners without serial bottlenecks. *In PPSC*, 1999. 23

[51] D. Hysom et A. Pothen : Level-based Incomplete LU factorization : Graph Model and Algorithms. Nov 2002. 20

[52] G. Karypis et V. Kumar : A fast and high quality multilevel scheme for partitioning irregular graphs. Technical Report 95-035, Department of Computer Science, University of Minnesota, 1995. 9

[53] G. Karypis et V. Kumar : MeTiS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0.* University of Minnesota, septembre 1998. 9

[54] C. Lanczos : Solution of systems of linear equations by minimized iterations. *J. Res. Natl. Bur. Stand*, 49:33–53, 1952. 15

[55] C. L. Lawson, R. J. Hanson, D. R. Kincaid et F. T. Krogh : Basic linear algebra subprograms for fortran usage. *ACM Trans. Math. Softw.*, 5(3):308–323, 1979. 11

[56] M. R. Leuze : Independent set orderings for parallel matrix factorization by gaussian elimination. *Parallel Computing*, 10(2):177–191, 1989. 28

[57] X. S. Li : An overview of superLU : Algorithms, implementation, and user interface. *ACM Trans. Math. Softw*, 31(3):302–325, 2005. 4

[58] Z. Li, Y. Saad et M. Sosonkina : pARMS : a parallel version of the algebraic recursive multilevel solver. *Numerical linear algebra with applications*, 10(5–6):485–509, juillet/septembre 2003. 29

[59] R. J. Lipton, D. J. Rose et R. E Tarjan : Generalized nested dissection. Rapport technique, Stanford, CA, USA, 1977. 8

[60] R. J. Lipton et R. E. Tarjan : A separator theorem for planar graphs. *SIAM J. Appl. Math.*, 36:177–199, 1979. 11

[61] J. W.-H. Liu : The role of elimination trees in sparse factorization. *Siam J. Matrix Anal. Appl.*, 11:134–172, 1990. 6, 8

[62] J. W.-H. Liu, E. G. Ng et B. W. Peyton : On finding supernodes for sparse matrix computations. *SIAM J. Matrix Anal. Appl.*, 14(1):242–252, 1993. 10

[63] T. A. Manteuffel : An incomplete factorization technique for positive definite linear systems. *Mathematics of Computation*, 34(150):473–497, avril 1980. 19

[64] J. A. Meijerink et H. A. van der Vorst : An iterative solution method for linear systems of which the the coefficient matrix is a symmetric M–matrix. *Math. Comp.*, 31:148–162, 1977. 19, 20

[65] E. Ng et B. W. Peyton. : Block sparse Cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput.*, 14:1034–1056, 1993. 13

[66] F. Pellegrini : Scotch 3.1 user's guide. Rapport technique 1137-96, LaBRI, Université Bordeaux I, août 1996. 78

[67] F. Pellegrini et J. Roman : Sparse matrix ordering with scotch. *In Proceedings of HPCN'97*, numéro 1225 de Lecture Notes in Computer Science, pages 370–378. Springer Verlag, April 1997. 8

[68] F. Pellegrini, J. Roman et P. Amestoy : Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency : Practice and Experience*, 12:69–84, 2000. Preliminary version published in *Proceedings of Irregular'99*, LNCS 1586, 986–995. 8, 9

[69] A. van der Ploeg, E. F. F. Botta et F. W. Wubs : Nested grids ILU-decomposition (NGILU). *J. Comput. Appl. Math.*, 66:515–526, 1996. 28

[70] J. Reid : On the method of conjugate gradients for the solution of large sparse systems of linear equations. *In Proceedings Conference Large Sparse Sets of Linear Equations*, London, 1971. Academic Press. 15

[71] J. K. Reid : Direct methods for sparse matrices. *In* Evans, D. J., éditeur : *Software for Numerical Mathematics*, pages 29–47. Academic Press, 1974. 6

[72] Y. Robert : Regular incomplete factorizations of real positive definite matrices. *Linear Algebra and its Applications*, 48:105–117, 1982. 19

[73] J. Roman : Partitionnement algorithmique des données pour la factorisation de Cholesky par bloc de grands systèmes linéaires creux sur des calculateurs MIMD. *Lettre du transputer et des calculateurs parallèles*, 6(24):115–120, 1994. 13

[74] D. J. Rose, R. E. Tarjan et G. S. Lueker : Algorithmic aspects of vertex elimination on graphs. *Siam J. Comput.*, 5:266–283, 1976. 9

[75] E. Rothberg et A. Gupta : An efficient block-oriented approach to parallel sparse Cholesky factorization. *SIAM J. Sci. Comput.*, 15(6):1413–1439, novembre 1994. 13

[76] J. W. Ruge et K. Stüben : Algebraic multigrid. *In* S. F. McCormick, éditeur : *SIAM Frontiers on Applied Mathematics, Volume 3, Multigrid Methods*, pages 73–130. SIAM, 1987. 16

[77] Y. Saad : ILUT : a dual threshold incomplete *LU* factorization. *Numerical linear algebra with applications*, 1(4):387–402, 1994. 22

[78] Y. Saad : ILUM : A multi-elimination ILU preconditioner for general sparse matrices. *SIAM Journal on Scientific Computing*, 17(4):830–847, juillet 1996. 28

[79] Y. Saad : *Iterative methods for sparse linear systems.* SIAM, Philadelphia, PA, second édition, 2003. 5, 17, 19, 44

[80] Y. Saad et M. Schultz : GMRES : A generalized minimal residual algorithm for solving nonsymmetric linear systems. *SIAM Journal of Scientific and Statistical Computing*, 7:856–869, 1986. 15

[81] Y. Saad et M. Sosonkina : Enhanced parallel multicolor preconditioning techniques for linear systems. *In PPSC*, 1999. 23

[82] Y. Saad et B. Suchomel : ARMS : an algebraic recursive multilevel solver for general sparse linear systems. *Numerical linear algebra with applications*, 9(5):359–378, juillet/août 2002. 29

[83] H. A. Schwarz : Über einen Grenzübergang durch alternierendes Verfahren. *Vierteljahrsschrift der Naturforschenden Gesellschaft in Zürich*, 15:272–286, mai 1870. 24

[84] B. F. Smith, P. E. Bjørstad et W. Gropp : *Domain Decomposition : Parallel Multilevel Methods for Elliptic Partial Differential Equations.* Cambridge University Press, 1996. 25, 26, 49

[85] U. Trottenberg, C. W. Oosterlee, A. Schuller, contributions by A. Brandt, P. Oswald et K. Stüben : *Multigrid.* Academic Press, San Diego, CA, 2001. 15

[86] R. S. Tuminaro, M. Heroux, S. A. Hutchinson et J. N. Shadid : Official Aztec user's guide version 2.1. Technical Report SAND99-8801J, Sandia National Laboratories, 1999. 24

[87] R. S. Varga : Factorization and normalized iterative methods. *In* R. E. Langer, éditeur : *Boundary Problems in Differential Equations*, pages 121–142. University of Wisconsin Press, Madison, WI, 1960. 19

[88] R. S. Varga, E. B. Saff et V. Mehrmann : Incomplete factorizations of matrices and connections with H-matrices. *SIAM Journal on Numerical Analysis*, 17(6):787–793, décembre 1980. 19

[89] J. S. Warsa, T. A. Wareing et J. E. Morel : Solution of the discontinuous P_1 equations in two-dimensional Cartesian geometry with two-level preconditioning. *SIAM Journal on Scientific Computing*, 24(6):2093–2124, novembre 2003. 5

[90] O. Widlund : Some Schwarz methods for symmetric and nonsymmetric elliptic problems. Technical Report TR1991-581, New York University, septembre, 1991. 23

[91] M. Yannakakis : Computing the minimum fill-in is NP-Complete. *SIAM J. Alg. and Discr. Meth.*, 2(1):77–79, 1981. 8

Liste des publications

Congrès internationaux avec comité de sélection :

[1] Jérémie Gaidamour et Pascal Hénon. Algorithms to build robust hybrid direct/iterative sparse linear solvers. *International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications*, Hong Kong, Août 2009.

[2] Jérémie Gaidamour et Pascal Hénon. A parallel direct/iterative solver based on a Schur complement approach. *11th IEEE International Conference on Computational Science and Engineering*, pages 98–105, Sao Paulo Brésil, Juillet 2008.

[3] Jérémie Gaidamour et Pascal Hénon. HIPS : a parallel hybrid direct/iterative solver based on a Schur complement approach. *PMAA'08*, Neuchâtel Suisse, Juin 2008.

[4] Jérémie Gaidamour, Pascal Hénon, Jean Roman, et Yousef Saad. Parallel resolution of sparse linear systems by mixing direct and iterative methods. *International Symposium on Iterative Methods in Scientific Computing (IMACS)*, Lille France, Mars 2008.

[5] Jérémie Gaidamour, Pascal Hénon, Jean Roman, et Yousef Saad. An hybrid direct-iterative solver based on a hierarchical interface decomposition. *International Conference on Preconditioning Techniques for Large Sparse Matrix Problems in Scientific and Industrial Applications*, Toulouse France, Juillet 2007.

Workshops internationaux avec comité de sélection :

[6] Jérémie Gaidamour et Pascal Hénon. Comparison of algorithms to build an efficient schur complement preconditioner in hips. *Sparse Days, CERFACS*, Toulouse France, Juin 2009.

[7] Jérémie Gaidamour et Pascal Hénon. HIPS : a parallel hybrid direct/iterative solver based on a Schur complement approach. *Sparse Days, CERFACS, Workshop de VECPAR'08*, Toulouse France, Juin 2008.

[8] Jérémie Gaidamour, Pascal Hénon, Jean Roman et Yousef Saad. An hybrid direct-iterative solver based on the Schur complement approach. *8th Workshop of the ERCIM Working group*, Salerne Italie, Septembre 2006.