Résolution parallèle de grands systèmes linéaires creux

David Goudin, Pascal Hénon, François Pellegrini, Pierre Ramet et Jean Roman.
ENSERB & LaBRI, UMR CNRS 5800
Université Bordeaux I
351 cours de la Libération, 33405 TALENCE, FRANCE
{goudin|henon|pelegrin|ramet|roman}@labri.u-bordeaux.fr

15 septembre 2000

Résumé

Cet article présente un tour d'horizon concernant les principes et techniques pouvant être utilisées pour la résolution de grands systèmes linéaires creux. Il présente en particulier les travaux réalisés au LaBRI au sein du thème ALiENor consacrés à la résolution parallèle haute performance par méthodes directes.

1 Introduction

De nombreux problèmes scientifiques et industriels peuvent être modélisés par des systèmes linéaires, de forme $Ax = b$, dont la taille sans cesse croissante impose le recours au parallélisme pour obtenir des temps de réponse acceptables. Un des cas les plus intensivement étudiés est la résolution de systèmes où A est une matrice symétrique, définie, positive, et creuse, c'est-à-dire avec un nombre de termes non-nuls de la matrice qui est faible par rapport à la taille de celle-ci.

La théorie des graphes constitue un des outils les plus efficaces de modélisation des problèmes apparaissant dans ces résolutions, en ce sens qu'ils permettent de représenter de façon naturelle la structure de la matrice, et donc les dépendances entre inconnues : à toute matrice symétrique peut être associé son graphe d'adjacence, dont les sommets représentent les inconnues, et tel qu'il existe une arête entre deux sommets v_i et v_j associés respectivement aux inconnues i et j si le terme $a_{i,j}$ de la matrice est non nul.

Les deux familles de méthodes les plus classiques de résolution de tels systèmes sont les méthodes itératives, qui calculent une solution du système en faisant converger itérativement une solution approchée, et les méthodes directes qui résolvent le système en factorisant la matrice A.

Dans le cas des méthodes itératives, la parallélisation s'effectue en répartissant de façon équilibrée les calculs itératifs associés à chaque inconnue sur les processeurs de l'architecture parallèle, tout en cherchant à minimiser le surcoût de communication induit par la mise à jour sur les processeurs voisins des copies des valeurs courantes des inconnues. La minimisation du temps de résolution peut alors être reformulée comme un problème d'affectation *sous contraintes* des sommets du graphe d'adjacence aux processeurs, connu sous le nom de *placement statique*.

Dans le cas des méthodes directes, la parallélisation nécessite de répartir sur les processeurs les calculs de factorisation des blocs colonnes, c'est-à-dire de répartir des ensembles de blocs colonnes ou de blocs, afin que les calculs affectés à chaque processeur soient les plus indépendants possibles vis-à-vis de ceux effectués sur les autres processeurs. Un problème crucial de ces méthodes est la minimisation du nombre d'opérations à réaliser pour factoriser la matrice, qui dépend du remplissage de celle-ci (c'est-à-dire du nombre de termes non-nuls qui sont créés lors de la factorisation), et découle directement de la numérotation des inconnues au sein du système. L'amélioration de la parallélisation de ces méthodes nécessite donc le calcul de renumérotations efficaces minimisant à la fois le nombre absolu d'opérations à réaliser (qui est aussi un objectif pour la résolution séquentielle) et la dépendance entre calculs, afin de disposer du parallélisme le plus grand possible. Ce problème de *renumérotation* peut lui aussi s'exprimer comme un problème de *partitionnement sous contraintes* de graphes lorsque l'algorithme de renumérotation utilise le principe des dissections emboîtées. Une fois le système renuméroté, il est nécessaire d'ordonnancer les calculs de factorisation numérique et de résolution du système triangulaire sur les processeurs de l'architecture parallèle, en tenant compte si possible des latences de communication entre processeurs.

La suite de ce document est consacrée à la résolution par méthodes directes, et présente les travaux réalisés au LaBRI au sein du thème ALiENor avec comme objectif la parallélisation d'un code de mécanique des structures. En guise de conclusion, nous fournissons un ensemble de liens électroniques pour rechercher de l'information sur ces sujets.

2 Résolution parallèle de grands systèmes linéaires creux par méthodes directes

La résolution par méthodes directes de grands systèmes linéaires creux nécessite plusieurs phases :

- la renumérotation de la matrice, qui permet de minimiser le remplissage de celle-ci lors de la factorisation numérique ;
- la factorisation symbolique, qui permet de connaître à l'avance la structure de la matrice factorisée. Afin d'économiser la place mémoire et d'accélérer le calcul de cette factorisation symbolique, celle-ci peut s'effectuer par blocs [5] ;
- le placement des blocs de la matrice sur les processeurs de la machine parallèle, en tenant compte de leurs dépendances mutuelles, afin d'obtenir un parallélisme maximum lors de la factorisation numérique ;
- la factorisation numérique proprement dite ;
- la résolution des systèmes triangulaires ainsi construits (phase de descente-remontée).

L'application parallèle qui nous sert à valider nos algorithmes est un code de mécanique des structures par éléments finis qui résout les équations de la mécanique des milieux continus. Une autre application cible est un code d'électro-magnétisme ; ceci nous a conduit à choisir une factorisation LDL^T plutôt que LL^T, afin de pouvoir traiter des matrices à coefficients complexes.

La mise en œuvre parallèle du code de mécanique des structures nécessite les phases supplémentaires suivantes, s'intégrant entre le placement des blocs et la factorisation numérique :

- le placement des éléments du maillage sur les processeurs afin de maximiser la localité des calculs sur les éléments et de minimiser les communications ;
- l'assemblage parallèle pour construire la matrice à factoriser et le ou les vecteurs de second membre.

Nous allons présenter dans la suite de cette section les différentes solutions que nous avons apportées à ces problèmes pour réaliser une chaîne logicielle parallèle complète.

2.1 Renumérotation

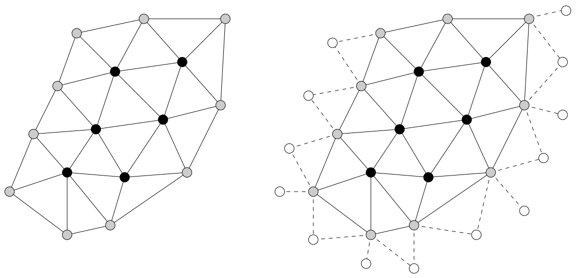
Lorsqu'on cherche à résoudre par méthodes directes des systèmes linéaires creux de la forme $Ax = b$, il est courant de faire précéder la phase de factorisation numérique de la matrice A par une phase de renumérotation des inconnues. Cette renumérotation est choisie de telle sorte que la mise à jour des termes non-diagonaux dans l'ordre de la matrice permutée PAP^T obtenue par renumérotation provoque beaucoup moins de remplissage et d'opérations que si la matrice avait été conservée telle quelle (le remplissage est l'ensemble des termes nuls de A qui deviennent non-nuls dans la matrice factorisée L).

Les deux méthodes de renumérotation les plus couramment utilisées sont le *degré minimum* et les *dissections emboîtées*. L'algorithme de degré minimum [23] est une heuristique locale qui choisit la prochaine inconnue à éliminer en prenant un nœud du graphe de plus petit degré. L'algorithme de dissections emboîtées [8] est une heuristique récursive globale qui calcule un ensemble de sommets S séparant le graphe en deux parties A et B et numérote S avec les plus grands numéros disponibles. Il s'applique alors récursivement aux parties A et B jusqu'à ce que leurs tailles deviennent inférieures à une valeur de seuil prédéfinie. Cette numérotation garantit qu'aucun terme non nul ne peut apparaître au cours de la factorisation entre les inconnues de A et les inconnues de B.

L'algorithme de degré minimum est réputé pour sa vitesse et sa généralité, et a fait l'objet de nombreuses études au cours des trente dernières années (voir par exemple [1, 9, 19]). Cependant, cet algorithme est intrinsèquement séquentiel, et peu de choses ont pu être prouvées sur son efficacité théorique.

En revanche, de nombreux résultats théoriques ont été obtenus sur la renumérotation par dissections emboîtées [5, 18], et sa formulation de type *diviser pour résoudre* la rend facilement parallélisable. En pratique, les dissections emboîtées calculent des renumérotations qui, tant en terme de remplissage que de nombre d'opérations, sont en moyenne plus efficaces que celles obtenues par le degré minimum [12, 16, 21]. De plus, les arbres d'élimination induits par les dissections emboîtées sont plus larges, plus courts, et mieux équilibrés, et de fait permettent un meilleur parallélisme dans le cadre de la factorisation de Cholesky parallèle.

Du fait de la nature complémentaire de ces méthodes, plusieurs techniques ont été proposées pour les coupler [4, 13, 17]. Dans les premières tentatives, seulement des couplages faibles ont été



a. Sans halo. **b.** Avec halo.

Fig. 1 – Connaissance des degrés du sous-graphe par l'algorithme de degré minimum sans et avec halo. Les sommets noirs sont les sommets internes du sous-graphe, les sommets gris sont les sommets appartenant à la frontière du sous-graphe, et les sommets blancs sont les sommets du halo, vus par l'algorithme de degré minimum HAMD mais n'appartenant pas au sous-graphe.

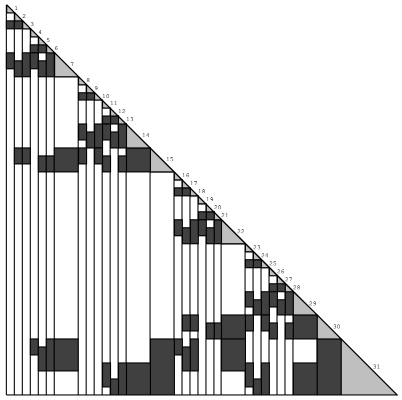


Fig. 2 – Un exemple de matrice factorisée structurée par blocs colonnes.

réalisés : des dissections emboîtées étaient calculées sur le graphe initial, et les sous-graphes finaux (avec un simple critère de taille) étaient traités au moyen de variantes de l'algorithme de degré minimum. Il en résultait que l'algorithme de degré minimum ne disposait pas des valeurs exactes des degrés pour les sommets situés sur les bords du sous-graphe, conduisant à un comportement erroné de l'algorithme de choix des sommets. Plusieurs couplages plus forts ont été proposés par la suite, qui suppriment ce comportement [13, 21].

Notre approche [21, 20] consiste à ajouter aux sous-graphes passés à l'algorithme de degré minimum une description topologique de leur voisinage immédiat (voir figure 1). Cette information supplémentaire, appelée *halo*, est alors exploitée par un algorithme de degré minimum approché [1]. L'arbre d'élimination (c'est-à-dire l'arbre des super-variables) est alors construit (voir plus bas). Nous appellerons dans la suite cet algorithme Halo-AMD, ou HAMD.

Afin de réaliser la factorisation numérique de façon efficace, il est nécessaire d'utiliser des routines de calcul par blocs de type BLAS 3, sur des blocs de taille suffisante. De fait, il faut amalgamer les colonnes de la matrice L renumérotée afin d'obtenir des blocs suffisament larges, sans pour autant pénaliser le remplissage.

L'algorithme d'amalgamation que nous avons utilisé n'opère que sur les sous-graphes passés à l'algorithme de degré minimum, car les séparateurs sont presque toujours de taille suffisante. Le critère d'amalgamation que nous avons choisi est basé sur trois paramètres, appelés respectivement cmin, cmax, et frat : un nœud de l'arbre d'élimination est fusionné avec son père, soit s'il contient moins de cmin colonnes, soit si la fusion induit moins de remplissage qu'une fraction frat de la surface du bloc fusionné, et ceci tant que le bloc colonne obtenu a moins de cmax colonnes [21].

La partition du graphe en super-variables est obtenue en fusionnant la partition induite par les séparateurs avec la partition des super-variables amalgamées localement dans chaque sous-graphe.

2.2 Factorisation symbolique

La factorisation symbolique de la matrice est réalisée par blocs, afin de réduire la taille des structures de données manipulées par l'algorithme, ainsi que le temps pris par celui-ci ; il a été montré que lorsque la factorisation symbolique s'effectue par blocs sa complexité en temps est quasi-linéaire par rapport au nombre de blocs (la linéarité étant prouvée dans le cas de la dissection complète [5]).

L'algorithme de factorisation symbolique opère sur la matrice renumérotée et partitionnée en blocs-colonnes par la phase de renumérotation, et produit en sortie la matrice L décomposée par blocs, stockant l'ensemble des termes de remplissage qui seront produits au cours de la phase de factorisation numérique (voir figure 2).

2.3 Repartitionnement et distribution des blocs

Certains solveurs numériques prennent entièrement à leur charge le séquencement dynamique des calculs de factorisation des blocs sur les processeurs de l'architecture parallèle, en se basant sur le placement des blocs préalablement calculé par un algorithme de pré-traitement de type *subtree-to-subcube*.

L'approche que nous avons choisie consiste à décharger entièrement le solveur de cette tâche, et donc à pré-calculer totalement l'ordonnancement des calculs de factorisation des blocs de façon statique, en simulant le plus précisément possible le comportement de l'architecture parallèle utilisée [14, 10]. Notre algorithme comprend deux phases distinctes.

La première est la phase de repartitionnement, dans laquelle les plus gros blocs sont redécoupés afin de pouvoir faire participer plusieurs processeurs à leur traitement. On exhibe ainsi un parallélisme de type *calcul parallèle sur des blocs pleins*, qui s'applique principalement aux plus gros séparateurs de l'arbre d'élimination, à la fin de la factorisation. Au cours de cette phase, on parcourt l'arbre d'élimination, et on calcule pour chaque bloc un ensemble de processeurs candidats susceptibles de l'accueillir. Cette notion de processeur candidat est indispensable pour préserver la bonne localité des communications ; celle-ci suit la logique de l'arbre d'élimination, car ne seront candidats pour un nœud de l'arbre d'élimination que les processeurs ayant participé à la factorisation du sous-arbre dont il est la racine.

La deuxième phase est la phase de distribution proprement dite, dans laquelle on attribue à chaque bloc le processeur qui le calculera, parmi ses processeurs candidats. Pour ce faire, on remonte l'arbre d'élimination des feuilles vers la racine et, à mesure que l'on distribue les nœuds de l'arbre sur les processeurs, on calcule le temps (estimé) pris par leur factorisation et l'envoi de leurs contributions. Ainsi, le choix de placement d'un bloc sur l'un de ses processeurs candidats est décidé en connaissant la date à laquelle potentiellement le processeur aura terminé de le traiter. Le processeur élu sera alors celui qui pourra avoir fini de traiter le bloc le plus tôt.

2.4 Distribution des éléments

La distribution des éléments est directement induite par la distribution des blocs issue de l'étape précédente [11]. Cette distribution permet de connaître à l'avance les différentes communications à effectuer dans le module d'assemblage (voir section 2.5).

Pour effectuer la distribution des éléments, il est nécessaire de connaître non seulement la distribution des blocs, mais aussi des informations relatives au maillage : géométrie de chaque élément, numéros des nœuds composant cet élément, et nombre de degrés de liberté par nœud. Le traitement effectué se décompose en deux parties :

– l'attribution des éléments totalement locaux à un processeur selon un critère d'équilibrage de charge en termes de nombre d'éléments (par élément local, on entend un élément dont tous les nœuds appartiennent à des blocs locaux à un processeur au sens de la distribution des blocs) ;
– la répartition des éléments restants.

L'algorithme que nous utilisons équilibre la masse de calculs pour chaque processeur, c'est-à-dire que chaque processeur doit calculer un nombre équivalent de matrices élémentaires. Les communications ne sont pas prises en compte pour l'équilibrage, car elles sont supposées *recouvertes* par les calculs.

Comme la phase de distribution des nœuds prend en compte les coûts de calcul associés aux nœuds, et que ces coûts sont inégaux suivant la connexité du nœud dans le maillage, il est nécessaire de faire les remarques suivantes :

– il ne faut placer qu'une fraction des éléments locaux afin de ne pas déséquilibrer le placement dès la première phase si un processeur en possède beaucoup plus que les autres, sans que ce déséquilibre puisse être compensé par le placement judicieux des autres éléments ;
– on détermine, pour chaque élément, sur quel processeur il doit être placé, en fonction de la charge courante de chaque processeur candidat, en privilégiant le processeur qui possède le plus de nœuds appartenant à l'élément considéré ;
– on cherche systématiquement à minimiser la distance entre les processeurs possédant les nœuds de l'élément et le processeur sur lequel celui-ci sera placé, afin d'augmenter la localité des communications.

2.5 Assemblage de la matrice

Cette phase utilise les informations qui lui sont données par la distribution des éléments. On commence par traiter les éléments qui ont été attribués au processeur courant mais dont un ou plusieurs nœuds sont dans des blocs appartenant à un autre processeur. On prend en compte, en priorité, les éventuels messages de contributions distantes. On calcule la matrice élémentaire d'un élément non local, on prépare les messages à envoyer aux processeurs concernés, et on effectue les additions de contributions locales. Pour réduire le nombre de communications, les messages sont vectorisés au niveau d'un élément.

Les éléments totalement locaux sont traités en dernier pour que leurs calculs recouvrent les communications induites par les éléments non locaux. On calcule leur matrice élémentaire, puis on additionne les contributions (locales) dans la partie de matrice correspondante.

2.6 Résolution numérique

La phase de résolution numérique réalise une factorisation parallèle de Crout LDL^T avec une distribution 1D des blocs de la matrice (c'est-à-dire par bloc-colonne) et/ou 2D (les blocs extra-diagonaux d'un bloc-colonne ne seront pas forcément tous distribués sur le même processeur).

Il existe essentiellement deux types d'approche qui sont l'approche *supernodale*, qui provient directement des algorithmes d'élimination par colonne réécrits par blocs et avec les variantes que l'on appelle *fan-in* ou *fan-out* [2, 3, 22], et l'approche *multifrontale* [7, 6]. On s'intéresse à l'algorithme supernodal de la factorisation parallèle creuse LDL^t où toutes les contributions distantes sont localement agrégées dans une structure bloc.

Des expérimentations parallèles ont été réalisées sur le calculateur IBM SP2 du CINES doté de 192 nœuds fins 120 MHz Power2SC (480 MFlops de puissance crête) disposant chacun de 256 MBytes de mémoire physique et d'un réseau d'interconnexion basé sur un switch TB3.

Nous présentons une comparaison des performances obtenues entre notre chaîne logicielle PaStiX [1] et la bibliothèque PSPASES [2] version 1.0.3 basée sur une approche multifrontale [15]. PaStiX utilise la version 3.4 de Scotch [20]; PSPASES est configuré pour utiliser la renumérotation de la version 4.0 de MeTiS [17]. La taille critique des blocs est fixée à 64 et la bibliothèque IBM ESSL est utilisée dans les deux cas.

Les jeux de tests utilisés sont ceux pour lesquels on disposait d'une description au format RSA, le format commun accepté par les deux chaînes logicielles. Les mesures données dans la table 1 ont été obtenues à l'aide d'une factorisation symbolique *scalaire* pour une renumérotation issue de Scotch mais sans amalgamation dans la partie degré minimum : $\mathrm{NNZ_A}$ est le nombre de termes extra-diagonaux dans la partie triangulaire inférieure de la matrice A, $\mathrm{NNZ_L}$ est le nombre de termes extra-diagonaux dans la matrice factorisée L et OPC est le nombre d'opérations à réaliser.

La table 2 présente les performances obtenues pour la factorisation parallèle (le temps est donné en secondes et la puissance en gigaflops). Pour chaque matrice, la première ligne donne le temps de PaStiX (factorisation LDL^t) et la seconde celui de PSPASES (factorisation LL^t). Les tirets correspondent à des exécutions faussées par des phénomènes de “swapping” ou d'allocation mémoire impossible.

[1] Parallel Sparse matriX package.
[2] Parallel SPArse Symmetric dirEct Solver.

Les expérimentations effectuées montrent que PaStiX se compare très favorablement à PSPASES et obtient de meilleurs temps de factorisation dans la plupart des cas jusqu'à 32 processeurs. Pour les plus gros cas tests, la comparaison est favorable à PaStiX jusqu'à 64 processeurs. Pour les cas plus petits, les performances sur 64 processeurs sont sensiblement équivalentes alors que l'on atteint les limites acceptables de la granularité des calculs. En plus d'une bonne scalabilité des performances, on remarque que l'approche retenue entraîne une bonne scalabilité mémoire ; une étude est en cours pour réduire encore, algorithmiquement, les surcoûts mémoires.

3 Pour en savoir plus...

Les articles [11, 14, 21] décrivent plus précisement les algorithmes mis en œuvre dans les différentes étapes de l'enchaînement, ainsi que l'environnement de développement et d'expérimentation. Les travaux de l'équipe ALiENor consacrés à la résolution parallèle par méthodes directes sont accessibles à l'adresse `http ://dept-info.labri.u-bordeaux.fr/~ramet/alienor/`.

Les travaux relatifs aux solveurs parallèles PSPASES et SuperLU sont disponibles aux adresses `http ://www-users.cs.umn.edu/~mjoshi/pspases/` et `http ://www.cs.berkeley.edu/~demmel/SuperLU.html/`.

Le projet Européen PARASOL concernant la résolution de grands systèmes linéaires creux est présenté à l'adresse `http ://www.genias.de/projects/parasol/`.

Un site de référence sur le parallélisme en général est le site Parascope, à l'adresse `http ://www.computer.org/parascope/`. Un autre site de référence sur les logiciels et bibliothèques est le site du NHSE, à l'adresse `http ://www.nhse.org/`.

Références

[1] P. Amestoy, T. Davis, and I. Duff. An approximate minimum degree ordering algorithm. *SIAM J. of Matrix Analysis and Applications*, 17 :886–905, 1996.

[2] C. Ashcraft, S. Eisenstat, J. W.-H. Liu, and A. Sherman. A comparison of three column based distributed sparse factorization schemes. In *Proc. Fifth SIAM Conference on Parallel Processing for Scientific Computing*, 1991.

[3] C. Ashcraft, S. C. Eisenstat, and J. W.-H. Liu. A fan-in algorithm for distributed sparse numerical factorization. *SIAM J. Sci. Stat. Comput.*, 11(3) :593–599, May 1990.

[4] C. Ashcraft and J. W.-H. Liu. Robust ordering of sparse matrices using multisectors. *SIAM Journal of Matrix Analysis and Applications*, 19 :816–832, 1998.

Nom	Taille	NNZ_A	NNZ_L (Scotch)	OPC (Scotch)	NNZ_L (MeTiS)	OPC (MeTiS)
B5TUER	162610	3873534	2.542e+07	1.531e+10	2.404e+07	1.237e+10
BMWCRA1	148770	5247616	6.597e+07	5.702e+10	6.981e+07	6.124e+10
MT1	97578	4827996	3.115e+07	2.109e+10	3.455e+07	2.269e+10
OILPAN	73752	1761718	8.912e+06	2.985e+09	9.065e+06	2.751e+09
QUER	59122	1403689	9.119e+06	3.281e+09	9.586e+06	3.448e+09
SHIP001	34920	2304655	1.428e+07	9.034e+09	1.481e+07	9.462e+09
SHIP003	121728	3982153	5.873e+07	8.008e+10	5.910e+07	7.587e+10
SHIPSEC5	179860	4966618	5.650e+07	6.952e+10	5.256e+07	5.509e+10
THREAD	29736	2220156	2.404e+07	3.884e+10	2.430e+07	3.583e+10
X104	108384	5029620	2.634e+07	1.713e+10	2.728e+07	1.412e+10

TAB. 1 – Description des cas tests de comparaison entre PaStiX et PSPASES.

Nom	Nombre de processeurs					
	2	4	8	16	32	64
B5TUER	**29.54** (0.52)	**15.52** (0.99)	**8.86** (1.73)	**5.25** (2.91)	**3.96** (3.87)	**2.91** (5.25)
	-	-	14.04 (0.88)	7.32 (1.69)	3.91 (3.16)	2.58 (4.79)
BMWCRA1	-	-	**30.97** (1.84)	**17.28** (3.30)	**9.89** (5.76)	**6.94** (8.21)
	-	-	-	-	24.87 (2.49)	13.48 (4.61)
MT1	**37.92** (0.56)	**20.35** (1.04)	**11.29** (1.87)	**6.65** (3.17)	**4.33** (4.87)	**3.51** (6.01)
	-	-	-	10.71 (2.12)	5.70 (3.98)	3.59 (6.32)
OILPAN	**7.28** (0.41)	**3.81** (0.78)	**2.15** (1.39)	**1.39** (2.14)	**1.00** (3.00)	**0.87** (3.42)
	-	5.23 (0.53)	2.79 (0.99)	1.73 (1.59)	1.25 (2.20)	0.93 (2.96)
QUER	**8.35** (0.39)	**4.46** (0.74)	**2.57** (1.28)	**1.67** (1.96)	**1.16** (2.83)	**0.93** (3.53)
	23.80 (0.14)	13.11 (0.26)	3.22 (1.07)	2.01 (1.72)	1.30 (2.65)	0.96 (3.59)
SHIP001	**20.98** (0.43)	**10.91** (0.83)	**6.07** (1.49)	**3.63** (2.49)	**2.43** (3.72)	**1.96** (4.60)
	-	15.32 (0.62)	8.11 (1.17)	4.48 (2.11)	2.98 (3.17)	2.14 (4.42)
SHIP003	-	**109.78** (0.73)	**45.83** (1.75)	**25.75** (3.11)	**16.60** (4.83)	**12.03** (6.66)
	-	-	-	-	21.28 (3.57)	14.08 (5.39)
SHIPSEC5	-	**79.12** (0.88)	**35.68** (1.95)	**20.51** (3.39)	**13.99** (4.97)	**11.58** (6.00)
	-	-	-	-	21.80 (2.52)	11.81 (4.66)
THREAD	**78.04** (0.50)	**41.14** (0.94)	**22.85** (1.70)	**13.50** (2.88)	**10.42** (3.73)	**6.70** (5.80)
	-	-	-	21.02 (1.70)	11.24 (3.19)	6.41 (5.59)
X104	**31.53** (0.54)	**19.69** (0.87)	**9.98** (1.72)	**7.49** (2.29)	**5.09** (3.37)	**3.85** (4.44)
	-	-	-	8.32 (1.70)	4.92 (2.87)	3.18 (4.44)

TAB. 2 – Comparaison des performances entre PaStiX et PSPASES.

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

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

[7] I. S. Duff. Sparse numerical linear algebra : direct methods and preconditioning. Technical Report TR/PA/96/22, CERFACS, 1996.

[8] A. George and J. W.-H. Liu. *Computer solution of large sparse positive definite systems*. Prentice Hall, 1981.

[9] A. George and J. W.-H. Liu. The evolution of the minimum degree ordering algorithm. *SIAM Review*, 31 :1–19, 1989.

[10] D. Goudin, P. Hénon, F. Pellegrini, P. Ramet, and J. Roman. Parallel sparse linear algebra and application to structural mechanics. *Accepted in Numerical Algorithms, Baltzer Science Publisher*, 2000.

[11] D. Goudin and J. Roman. A scalable parallel assembly for irregular meshes based on a block distribution for a parallel block direct solver. In *Proceedings of PARA'2000*, June 2000.

[12] A. Gupta, G. Karypis, and V. Kumar. Highly parallel algorithms for sparse matrix factorization. Technical Report 94-63, Dept. of Computer Science, Univeristy of Minnesota, 1994.

[13] B. Hendrickson and E. Rothberg. Improving the runtime and quality of nested dissection ordering. *SIAM J. Scientific Computing*, 20(2) :468–489, 1998.

[14] P. Hénon, P. Ramet, and J. Roman. PaStiX : A parallel sparse direct solver based on a static scheduling for mixed 1D/2D block distributions. In *Proceedings of Irregular'2000, LNCS 1800*, pages 519–525, May 2000.

[15] M. Joshi, G. Karypis, V. Kumar, A. Gupta, and Gustavson F. PSPASES : Scalable Parallel Direct Solver Library for Sparse Symmetric Positive Definite Linear Systems. Technical report, University of Minnesota and IBM Thomas J. Watson Research Center, May 1999.

[16] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. TR 95-035, University of Minnesota, June 1995.

[17] G. Karypis and 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, 1998. `http ://www-users.cs.umn.edu/~karypis/metis/`.

[18] R. J. Lipton, D. J. Rose, and R. E. Tarjan. Generalized nested dissection. *SIAM Journal of Numerical Analysis*, 16(2) :346–358, April 1979.

[19] J. W.-H. Liu. Modification of the minimum-degree algorithm by multiple elimination. *ACM Transactions on Mathematical Software*, 11(2) :141–153, 1985.

[20] F. Pellegrini. SCOTCH v3.1 user's guide, Static Mapping Package. `http ://www.labri.u-bordeaux.fr/~pelegrin/scotch/`, September 1996.

[21] F. Pellegrini, J. Roman, and P. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency : Practice and Experience*, 12 :69–84, 2000.

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

[23] W. F. Tinney and J. W. Walker. Direct solutions of sparse network equations by optimally ordered triangular factorization. *J. Proc. IEEE*, 55 :1801–1809, 1967.