RENPAR/CFSE/SYMPA
Paris, cité des sciences et de l'industrie, 24 – 27 avril 2001

PaStiX : Un solveur parallèle direct pour des matrices creuses symétriques définies positives basé sur un ordonnancement statique performant et sur une gestion mémoire efficace.

Pascal Hénon, Pierre Ramet

LaBRI, UMR CNRS 5800
ENSEIRB et Université Bordeaux 1
351 cours de la Libération, 33405 TALENCE, FRANCE
{henon|ramet}@labri.u-bordeaux.fr

Résumé
La résolution de grands systèmes linéaires creux est un point crucial dans de nombreuses applications industrielles et scientifiques. Notre travail porte sur le partionnement et la distribution de grandes matrices creuses pour la factorisation LDL^t en parallèle sur machine de type MIMD. Nous présentons dans cet article notre technique de factorisation parallèle basée sur un ordonnancement statique des calculs et des communications, et nous la validons sur des systèmes de plus d'un million d'inconnues pour des problèmes d'éléments finis 3D.

Mots-clés : calcul hautes performances, factorisation de Cholesky parallèle, matrice creuse, partitionnement et distribution statique.

---

1. Introduction

La résolution de grands systèmes linéaires creux est une brique de base essentielle pour le traitement de nombreux problèmes de simulation numérique en vraie grandeur. Pour de telles applications, par exemple basées sur une modélisation par éléments finis, le recours au calcul parallèle hautes performances est incontournable. Dans ce cadre-là, on est souvent amené à utiliser des solveurs directs réputés pour leur robustesse numérique en particulier lorsqu'il y a des problèmes de conditionnement qui font que les solveurs itératifs préconditionnés se comportent de manière instable.
La résolution par méthode directe 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 et de maximiser l'indépendance dans les calculs du solveur. Nous utilisons dans ce travail un "couplage fin" entre l'algorithme du degré minimum et l'algorithme de dissections emboîtées : une fois que les premiers niveaux de récursion de la dissection emboîtée sont effectués, les sous-graphes restants sont alors numérotés en utilisant l'algorithme du "Halo Approximate Minimum Degree" [13]. A l'issue de cette étape on dispose d'une partition initiale des inconnues de la matrice ;
- 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, cette étape s'effectue par blocs sur la partition issue de la renumérotation [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 [8, 9] ;
– la factorisation numérique proprement dite et la résolution des systèmes triangulaires ainsi construits (phase de descente-remontée). 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* [3, 4, 15], et l'approche *multifrontale* [1, 2, 6]. On s'intéresse ici à un algorithme supernodal de factorisation parallèle creuse LDL^t où toutes les contributions distantes sont localement agrégées dans une structure bloc. On utilise 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) calculée par l'étape de placement statique [8, 9].

Nous allons présenter dans la suite de ce papier le résultat de nos travaux portant sur le repartitionnement et la distribution de la matrice par blocs, ainsi que des résultats expérimentaux pour la factorisation numérique parallèle sur de très grands systèmes creux.

2. Repartitionnement et distribution des blocs

Certains solveurs numériques prennent entièrement à leur charge le séquencement dynamique des calculs de factorisation des blocs sur l'architecture parallèle, en se basant sur le placement des blocs préalablement calculé par un algorithme de prétraitement par exemple 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 et des communications de manière statique, en simulant le plus précisément possible le comportement de l'architecture parallèle utilisée [7, 8]. 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 de haut en bas et on calcule pour chaque bloc un ensemble de processeurs candidats susceptibles de le traiter ; cette méthode est basée sur la technique de "proportional mapping" introduite par Pothen et Sun [14]. Cette notion de processeur candidat est indispensable pour préserver la bonne localité des communications ; celle-ci s'obtient en suivant 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, pour distribuer chaque nœud de l'arbre sur les processeurs, on calcule le temps (estimé) pris par sa factorisation et l'envoi de ses contributions. Ainsi, le choix de placement d'un bloc sur l'un de ses processeurs candidats est décidé en connaissant la date à laquelle le processeur aura potentiellement terminé de le traiter. Le processeur élu sera alors celui qui pourra avoir fini de traiter le bloc le plus tôt. Cette étape de simulation permet un ordonnancement fin des calculs et des communications ; elle nécessite le calibrage des opérations BLAS3 et des opérations de communication MPI pour l'architecture cible. Il est intéressant de noter que cette stratégie peut s'adapter via les modèles de coût à des

TAB. 1 – Description de nos cas tests.

Nom	Taille	NNZ_A	NNZ_L	OPC	Description
GRID1023	1046529	4179980	5.615708e+07	2.083481e+10	Maillage régulier 2D
CUBE39	59319	730778	2.210534e+07	2.240674e+10	Maillage régulier 3D
CUBE47	103823	1290898	4.828456e+07	6.963850e+10	Maillage régulier 3D
BCSSTK32	44609	985046	5.239146e+06	1.162900e+09	Rutherford-Boeing
M14B	214765	1679018	6.236747e+07	6.112540e+10	Rutherford-Boeing
OCEAN	143437	409593	2.029997e+07	1.301477e+10	Rutherford-Boeing
BBMAT	38744	1274141	1.716094e+07	1.250040e+10	Rutherford-Boeing
TOOTH	78136	452591	1.031143e+07	6.267094e+09	Rutherford-Boeing
SMDOOR	162610	3873534	2.541937e+07	1.530774e+10	PARASOL
OILPAN	73752	1761718	8.912337e+06	2.984944e+09	PARASOL
BMWCRA_1	148770	5247616	6.597301e+07	5.701988e+10	PARASOL
INVEXTR1	30412	906915	7.256566e+06	3.766788e+09	PARASOL
MIXTANK	29957	982542	9.280247e+06	7.316933e+09	PARASOL
MT1	97578	4827996	3.114873e+07	2.109265e+10	PARASOL
QUER	59122	1403689	9.118592e+06	3.280680e+09	PARASOL
SHIP001	34920	2304655	1.427916e+07	9.033767e+09	PARASOL
SHIP003	121728	3982153	5.872912e+07	8.008089e+10	PARASOL
SHIPSEC5	179860	4966618	5.649801e+07	6.952086e+10	PARASOL
SHIPSEC8	114919	3269240	3.572761e+07	3.684269e+10	PARASOL
THREAD	29736	2220156	2.404333e+07	3.884020e+10	PARASOL
X104	108384	5029620	2.634047e+07	1.712902e+10	PARASOL
BMW3_2	227362	5530634	4.420244e+07	3.007981e+10	PARASOL
CRANKSG1	52804	5280703	3.142730e+07	3.007141e+10	PARASOL
CRANKSG2	63838	7042510	4.190437e+07	4.602878e+10	PARASOL
COLOGB30	20373	1394292	7.849464e+06	4.359803e+09	Cologne 3D
COLOGB75	50208	3473292	2.248613e+07	1.532035e+10	Cologne 3D
COUP1500T	994983	70303275	5.219997e+08	3.79847e+11	Coupole 3D
COUP2000T	1326483	93734775	6.888813e+08	5.00924e+11	Coupole 3D

architectures hétérogènes et plus particulièrement celles de type SMP comme l'IBM SP3.
Comme il a été constaté que des distributions hybrides (1D en bas de l'arbre d'élimination et 2D en haut) [8] donnent de meilleurs résultats, l'algorithme de repartitionnement a été modifié pour que le choix du type de distribution soit réalisé lors de la descente récursive de l'arbre d'élimination par blocs en fonction du nombre de processeurs et du coût des sous-arbres à traiter.
De plus, nous avons intégré dans notre phase de régulation statique la possibilité de réduire le surcoût mémoire lié aux agrégations locales de contributions par l'anticipation d'envoi d'agrégats partiels. En effet, le surcoût mémoire lié à la stratégie d'agrégation locale peut dépasser dans certains cas le coût de stockage de la matrice elle-même. Pour réduire ce surcoût, nous modifions le schéma d'ordonnancement des communications en anticipant l'envoi d'agrégats partiels afin de ne pas dépasser une limite mémoire fixée par l'utilisateur. Ce recalcul du schéma d'ordonnancement est optimisé pour affecter le moins possible les performances en temps du solveur.

3. Résultats expérimentaux

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.
Les mesures données dans la table 1 ont été obtenues à l'aide d'une factorisation symbolique *scalaire* pour notre renumérotation qui utilise l'algorithme du "Halo Approximate Minimum Degree" [13] sans amalgamation dans la partie degré minimum : NNZ_A est le nombre de termes extra-diagonaux dans la partie triangulaire inférieure de la matrice A, NNZ_L est le nombre de

TAB. 2 – Performances de la factorisation sur un IBM SP2.

Nom	Nombre de processeurs				
	4	8	16	32	64
GRID1023	-	**14.57** (1.43)	**8.61** (2.42)	**5.56** (3.74)	**3.83** (5.44)
CUBE39	**24.20** (0.93)	**13.20** (1.70)	**8.88** (2.52)	**6.32** (3.55)	**5.03** (4.46)
CUBE47	**73.31** (0.95)	**38.23** (1.82)	**24.08** (2.89)	**17.15** (4.06)	**12.59** (5.53)
BCSSTK32	**2.46** (0.47)	**1.41** (0.82)	**1.08** (1.08)	**0.78** (1.49)	**0.71** (1.63)
M14B	-	**46.95** (1.30)	**25.84** (2.37)	**14.60** (4.19)	**9.21** (6.64)
OCEAN	**18.39** (0.71)	**9.78** (1.33)	**6.26** (2.08)	**4.12** (3.16)	**4.17** (3.12)
BBMAT	**14.73** (0.85)	**8.55** (1.46)	**5.24** (2.39)	**3.39** (3.69)	**2.72** (4.60)
TOOTH	**10.70** (0.59)	**6.60** (0.95)	**4.01** (1.56)	**2.77** (2.26)	**2.49** (2.52)
SMDOOR	**15.52** (0.99)	**9.10** (1.68)	**5.25** (2.91)	**4.07** (3.76)	**2.91** (5.25)
OILPAN	**3.81** (0.78)	**2.23** (1.34)	**1.51** (1.98)	**1.03** (2.90)	**0.92** (3.24)
BMWCRA_1	-	**30.97** (1.84)	**18.75** (3.04)	**9.55** (5.97)	**5.94** (9.60)
INVEXTR1	**6.31** (0.60)	**4.03** (0.93)	**2.52** (1.49)	**1.76** (2.14)	**1.57** (2.40)
MIXTANK	**10.02** (0.73)	**6.33** (1.16)	**3.80** (1.93)	**3.32** (2.20)	**3.10** (2.36)
MT1	**20.35** (1.04)	**12.91** (1.63)	**6.96** (3.03)	**4.33** (4.87)	**3.51** (6.01)
QUER	**4.46** (0.74)	**2.77** (1.18)	**1.90** (1.73)	**1.35** (2.43)	**1.04** (3.16)
SHIP001	**10.91** (0.83)	**6.13** (1.47)	**3.78** (2.39)	**2.34** (3.86)	**1.96** (4.60)
SHIP003	**109.78** (0.73)	**45.83** (1.75)	**26.86** (2.98)	**16.69** (4.80)	**12.03** (6.66)
SHIPSEC5	**79.12** (0.88)	**40.36** (1.72)	**21.46** (3.24)	**15.36** (4.53)	**11.76** (5.91)
SHIPSEC8	**43.40** (0.85)	**27.71** (1.33)	**15.40** (2.39)	**11.93** (3.09)	**9.23** (3.99)
THREAD	**41.14** (0.94)	**22.85** (1.70)	**13.45** (2.89)	**11.79** (3.29)	**6.70** (5.80)
X104	**19.69** (0.87)	**11.02** (1.55)	**6.32** (2.71)	**5.30** (3.23)	**3.85** (4.44)
BMW3_2	**34.02** (0.88)	**20.78** (1.45)	**11.52** (2.61)	**7.90** (3.81)	**6.67** (4.51)
CRANKSG1	**33.54** (0.90)	**19.20** (1.57)	**10.54** (2.85)	**6.82** (4.41)	**5.34** (5.63)
CRANKSG2	**47.99** (0.96)	**25.33** (1.82)	**14.39** (3.20)	**8.45** (5.45)	**6.04** (7.62)
COLOGB30	**5.25** (0.83)	**3.23** (1.35)	**2.06** (2.12)	**1.69** (2.58)	**1.34** (3.25)
COLOGB75	**18.72** (0.82)	**8.99** (1.70)	**5.69** (2.69)	**3.89** (3.94)	**3.14** (4.88)
COUP1500T	-	-	-	-	**33.01** (11.51)
COUP2000T	-	-	-	-	**43.12** (11.62)

termes extra-diagonaux dans la matrice factorisée L et OPC est le nombre d'opérations à réaliser.

Nous avons expérimenté plusieurs critères de seuil pour basculer, dans l'étape du placement statique, entre une distribution 1D et 2D. Les meilleurs résultats obtenus sont ceux pour lesquels on utilise un simple critère sur le nombre de processeurs candidats affectés à un sous arbre de l'arbre d'élimination.

La table 2 présente les performances obtenues pour la factorisation parallèle (le temps est donné en secondes et la puissance en gigaflops). Les tirets correspondent à des exécutions faussées par des phénomènes de "swapping" ou d'allocation mémoire impossible. Une bonne scalabilité est obtenue pour l'ensemble des cas tests jusqu'à 64 processeurs. On obtient en moyenne environ 35% de la puissance crête théorique, pour des calculs effectués en double précision.

Dans [8] nous avons présenté une comparaison des performances obtenues entre notre chaîne logicielle PaStiX [1] (utilisant la version 3.4 de Scotch [12]) et la bibliothèque PSPASES [2] version 1.0.3 basée sur une approche multifrontale [10] (utilisant la version 4.0 de MeTiS [11]). 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 64 processeurs.

La table 3 présente, pour un cas test, la dégradation en temps de la factorisation parallèle induite

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

TAB. 3 – Influence de l'agrégation partielle sur le temps de factorisation de la matrice BMW-CRA_1.

% réduction	Nombre de processeurs							
BMWCRA_1	8		16		32		64	
0%	**35.04**	**9%**	**20.23**	**22%**	**12.37**	**35%**	**7.30**	**59%**
20%	+2.5%	**7%**	+2.9%	**18%**	+3.2%	**28%**	+1.4%	**47%**
30%	+8.4%	**6%**	+9.9%	**15%**	+6.5%	**24%**	+6.2%	**41%**
40%	+18.0%	**5%**	+30.1%	**13%**	+32.2%	**21%**	+38.6%	**35%**
50%	+28.7%	**4%**	+51.7%	**11%**	+55.9%	**17%**	+71.4%	**29%**

par la technique d'agrégation partielle (cf. dernier paragraphe de la section 2). Rappelons que cette technique a pour but de réduire le surcoût mémoire lié à l'agrégation.
Les résultats sont obtenus pour différents nombres de processeurs et pourcentages de réduction mémoire. La première colonne donne le pourcentage de réduction mémoire, fixé par l'utilisateur, du surcoût total associé aux blocs d'agrégation. Les autres colonnes donnent, en utilisant ce pourcentage, l'augmentation en temps (par rapport au temps en secondes du schéma avec agrégration totale donné à la ligne 0%) de la factorisation parallèle, ainsi que le pourcentage de la mémoire nécessaire pour allouer l'ensemble des blocs d'agrégation par rapport à la mémoire nécessaire pour allouer les coefficients de la matrice. Ces résultats montrent qu'une réduction mémoire de l'ordre de 30% (par rapport au surcoût total associé aux blocs d'agrégation) est acceptable en terme de performance (sur 64 processeurs, le surcoût mémoire passe de 59% à 41% pour une augmentation en temps de l'ordre de seulement 6%). Au delà, la perte d'efficacité augmente notoirement plus avec le nombre de processeurs du fait des communications et manipulations mémoire supplémentaires. Dans ce contexte, notre gestion des surcoûts mémoire conduit à une amélioration significative de la scalabilté mémoire.
On peut remarquer que le temps correspondant à 0% (schéma avec agrégration totale) est supérieur à celui donné dans la table 2. Ceci est du au fait que les allocations des blocs d'agrégration sont effectuées ici de manière *dynamique* (à la demande lors de la factorisation numérique) alors qu'elles étaient réalisées de manière *statique* pour les résultats présentés dans la table 2.

Des résultats expérimentaux récents ont eté réalisés, pour les deux plus gros cas tests, sur l'IBM SP3 de l'université du Minnesota, constitué de 78 nœuds WinterHawk+, chaque nœud disposant de 4 processeurs (375 MHzPower3) avec plus de 4 GB de mémoire physique. Sur 16 nœuds (64 processeurs) la factorisation de la matrice COUP1500T est effectuée en 11.8s (32.2 gigaflops), et la matrice COUP2000T en 15.1s (33.3 gigaflops). Nous pensons que les performances obtenues sont plutôt satisfaisantes et confirment que notre stratégie d'ordonnancement est adaptable aux architectures hétérogènes de type SMP.

4. Conclusions et perspectives

Nous avons présenté dans cet article un solveur direct LDL^t basé sur un ordonnancement statique des calculs et des communications. Nous avons vu que cette stratégie s'avère très efficace puisqu'elle permet d'obtenir une factorisation très performante tout en permettant une gestion efficace des problèmes de mémoire rencontrés habituellement pour cette classe de solveur parallèle. En outre, nos stratégies de placement, d'ordonnancement et de résolution s'adaptent spécifiquement à des architectures hétérogènes notamment celles de type SMP (IBM SP3). Les prochaines améliorations de PaStiX porteront sur une version autorisant des matrices à coefficients complexes du solveur ainsi qu'une version *out-of-core*. Pour des précisions sur les travaux de l'équipe ALiENor consacrés à la résolution parallèle par méthodes directes vous pouvez consulter l'adresse `http ://dept-info.labri.u-bordeaux.fr/~ramet/alienor/`.

Bibliographie

1. P. R. Amestoy and I. S. Duff. Memory management issues in sparse multifrontal methods on multiprocessors. *Int. J. Supercomputer Applics*, 7 :64–82, 1993.
2. P. R. Amestoy, I. S. Duff, and J. Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.*, 2000.
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, 1990.
4. C. Ashcraft, S. C. Eisenstat, J. W.-H. Liu, and A. Sherman. A comparison of three column based distributed sparse factorization schemes. In *Proc. Fifth SIAM Conf. on Parallel Processing for Scientific Computing*, 1991.
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. D. Goudin, P. Hénon, F. Pellegrini, P. Ramet, J. Roman, and J.-J. Pesqué. Parallel sparse linear algebra and application to structural mechanics. *Numerical Algorithms, Baltzer Science Publisher*, 24 :371–391, 2000.
8. 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 *Proccedings of Irregular'2000*, number 1800 in Lecture Notes in Computer Science, pages 519–525. Springer Verlag, May 2000.
9. P. Hénon, P. Ramet, and J. Roman. PaStiX : A Parallel Direct Solver for Sparse SPD Matrices based on Efficient Static Scheduling and Memory Managment. In *Proccedings of Tenth SIAM Conference on Parallel Processing for Scientific Computing*, March 2001.
10. 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.
11. 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, September 1998.
12. F. Pellegrini and J. Roman. Sparse matrix ordering with SCOTCH. In *Proceedings of HPCN'97*, number 1225 in Lecture Notes in Computer Science, pages 370–378. Springer Verlag, April 1997.
13. F. Pellegrini, J. Roman, and P. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. In *Proceedings of Irregular'99*, number 1586 in Lecture Notes in Computer Science, pages 986–995. Springer Verlag, April 1999. Extended paper appeared in Concurrency : Practice and Experience, 12 :69-84, 2000.
14. A. Pothen and C. Sun. A mapping algorithm for parallel sparse Cholesky factorization. *SIAM J. Sci. Comput.*, 14(5) :1253–1257, September 1993.
15. 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.