# École d'hiver ICaRE97
# Irrégularité : Capa - Rumeur - Exec

Coordonnateurs

- D. Barth
- J. Chassin de Kergommeaux
- J.-L. Roch
- J. Roman

# Table des matières

# Table des figures

# Table des tableaux

# Première partie

# Titre Partie Journée

# Chapitre 1

# Optimisation des communications et régulation de charge pour la résolution par méthode directe de grands systèmes linéaires creux

P. AMESTOY (ENSEEIHT-IRIT, Toulouse),
F. DESPREZ (LIP INRIA Rhône-Alpes, Lyon),
P. RAMET, J. ROMAN (LaBRI, Bordeaux)

La résolution de systèmes linéaires creux est une brique de base pour traiter numériquement de nombreux problèmes de calcul scientifique. Ces systèmes, qui apparaissent en particulier dans le cadre de la discrétisation d'équations aux dérivées partielles par éléments ou volumes finis, sont de très grande taille pour les applications en vraie grandeur. Le parallélisme est alors une technique incontournable pour résoudre ces très grands systèmes. Dans ce contexte, les méthodes directes jouent un rôle très important car elles sont générales et surtout robustes numériquement ; il existe en effet de nombreux cas où une factorisation numérique s'avère être le seul recours pour une résolution raisonnable du système.

Pour obtenir de bonnes performances globalement, il est obligatoire d'utiliser systématiquement des noyaux de calcul performants implémentés au plus bas niveau pour exploiter au mieux les effets de caches. C'est le cas des primitives de calcul BLAS avec un maximum d'efficacité pour celles de niveau 3 dans le cadre de calculs matriciels denses par bloc ; il faut noter cependant qu'il est important de bien gérer la taille des blocs pour atteindre des vitesses de calcul maximales.

Cet état de fait a un impact considérable sur l'algorithmique des solveurs, sur le prétraitement des données à réaliser et enfin sur l'étude des problèmes liés à la gestion du creux des matrices car tout doit être pensé en termes de blocs : les structures de données seront par bloc et les calculs se feront par bloc.

Un des points essentiels propre aux méthodes directes sur les matrices creuses est le problème du *remplissage*. En effet, la matrice factorisée de Cholesky $L$ est plus pleine que la matrice initiale $A$ de part la création de nouveaux termes non nuls au cours du processus de factorisation. Un outil fondamental pour étudier ce remplissage est le modèle de graphe associé à l'élimination de Gauss. Celui-ci étant directement lié à l'ordre d'élimination des inconnues, il s'agit donc de trouver une renumérotation de ces inconnues, c'est à dire une renumérotation des sommets du graphe associé à la matrice initiale qui minimise le remplissage et en même temps le nombre d'opérations à faire effectivement pour factoriser.

L'arbre d'élimination associé à la numérotation joue un rôle fondamental pour la parallélisation des solveurs creux directs car il indique les dépendances dans les calculs : il ne peut y avoir de dépendances (et donc une séquentialité dans les calculs) entre deux éliminations d'inconnues que si les sommets associés sont sur une même branche de l'arbre d'élimination. Dans un cadre parallèle, le but est donc de trouver une renumérotation des sommets qui *minimise le remplissage* et qui *maximise l'indépendance dans les calculs*, c'est à dire conduisant à des arbres d'élimination *large et de faible hauteur*. Les renumérotations performantes sont celles basées sur les techniques de "degré minimum" et de "dissections emboîtées".

Une autre étape fondamentale est celle dite de *factorisation symbolique* qui va permettre de calculer algorithmiquement la structure creuse de la matrice $L$ à partir de celle de $A$. La résolution générique d'un système linéaire creux sera donc constituée de 4 étapes:

1. renumérotation des sommets du graphe initial,
2. factorisation symbolique pour construire la structure creuse de la matrice factorisée,
3. factorisation numérique sur la structure précédente, les seuls termes non nuls étant ceux de la matrice initiale,
4. résolution des systèmes triangulaires creux par descente-remontée.

Comme il a été dit plus haut, il est indispensable d'utiliser un algorithme de calcul par bloc pour profiter des performances des primitives de type BLAS. Cela veut dire que la structure de donnée pour la matrice $L$ doit être par bloc, et donc que l'algorithme de factorisation symbolique se fera aussi au sens des blocs. Cette notion de bloc dense de coefficients dans la matrice creuse correspond à la notion de *supernode* ; le but est de rechercher dans la matrice factorisée des ensembles de colonnes ayant la même structure de creux et qui pourront donc être "amalgamées" en une "super variable". Ceci se fera de manière couplée avec la factorisation symbolique.

Le cadre parallèle étant celui des machines MIMD à mémoire distribuée, une autre phase de prétraitement des données va précéder l'exécution parallèle du solveur numérique. Avec un modèle de programmation de type SPMD et l'utilisation de bibliothèque de communication de type MPI, ce prétraitement a pour rôle de calculer algorithmiquement un bon partitionnement et une bonne distribution des données ; ceci se fera à partir de la factorisation symbolique et par bloc. Le prétraitement va donc, en plus de la phase de renumérotation, calculer de manière statique une régulation équilibrée pour le solveur (critère d'équilibrage de charge et prise en compte des contraintes de précédence) ; on utilisera ici la structure de l'arbre d'élimination. Le but sera d'exploiter les différents niveaux de parallélisme dans les calculs, à savoir *un premier niveau induit par le creux* présent dans les calculs entre blocs indépendants, et *un deuxième niveau induit par les calculs dans les blocs pleins.* À noter que l'on peut aussi optimiser la localité des communications pour éviter des contentions dans le réseau du calculateur.

Les solveurs numériques eux-mêmes sont basés sur l'approche supernodale (stratégies fan-out, fan-in, fan-both) ou sur l'approche multifrontale. La gestion de l'irrégularité des calculs nécessite la mise en œuvre dans la programmation de techniques assez fines : gestion de l'indéterminisme d'exécution (threads ou régulation algorithmique intégrée dans le code SPMD), utilisation de communications asynchrones avec gestion du recouvrement calcul/communication, ....

Le but des sections qui suivent est de faire le point sur les aspects liés aux phases de renumérotation et de distribution des données, puis d'aborder d'autres aspects de nature différente ; il s'agira tout d'abord de la mise en place d'une nécessaire régulation dynamique lorsque l'on fait du pivotage par exemple dans le cas non-symétrique, puis de la gestion du grain pour optimiser les recouvrements calcul/communication.

## 1.1 Prétraitement statique pour obtenir une bonne régulation

### 1.1.1 L'algorithme du degré minimum

L'algorithme du degré minimum est un des algorithmes locaux de réordonnancement de matrices creuses symétriques les plus classiquement utilisés. Cet algorithme [39, 50] trouve son nom dans son interprétation en tant que manipulation du graphe associé à la matrice creuse; en effet le degré minimum est un algorithme local qui à chaque étape choisit comme pivot le nœud de degré minimum dans le graphe associé à la matrice réduite. Cet algorithme pose en fait de nombreux problèmes d'implémentation et a fait l'objet de nombreux travaux répertoriés dans l'article [1]. Les principales étapes de l'algorithme de degré minimum sont décrites ci-après.

**Pour** $k = 1$ **au** nombre de nœuds **faire**
  1/ Choisir un pivot $p$ non encore sélectionné et de degré minimum
  2/ Construire la liste d'adjacence de $p$
    et mettre à jour la liste d'adjacence des $i \in Adjacence(p)$
  3/ Calculer le degré modifié des $i \in Adjacence(p)$
**Fin Pour**

Sur de nombreux problèmes, l'étape 3 de l'algorithme est l'étape la plus coûteuse. Amestoy, Davis et Duff [1] ont proposé de remplacer le calcul du degré par une approximation, pouvant être calculée durant l'étape 2 sans sur-coût significatif. La précision de l'approximation conduit à un algorithme (*AMD: Approximate Minimum Degree*) qui s'avère de qualité comparable aux algorithmes calculant un degré exact. Pour illustrer l'influence sur le temps de calcul, AMD a été comparé au Multiple Minimum Degree (MMD, voir [38]) qui fait référence en la matière. Les matrices de test sont issues de [18] et du jeu de matrices de Davis (Université de Floride).

| matrice | ordre | non-zéros | AMD | MMD |
|---|---|---|---|---|
| BCSSTK32 | 44609 | 985046 | 3.3 | 6.4 |
| WANG3 | 26064 | 75552 | 3.6 | 5.3 |
| FINAN512 | 74752 | 261120 | 5.6 | 292.6 |
| ORANI678 | 2529 | 85426 | 8.2 | 168.6 |
| BBMAT | 38744 | 1274141 | 16.5 | 111.7 |
| PSMIGR_1 | 3140 | 410781 | 17.0 | 270.9 |
| LHR71 | 70304 | 1199704 | 21.8 | 130.2 |

Tableau 1.1: Comparaison de AMD et MMD (SPARC 10, temps en secondes).

D'autre part, dans le cas de matrices possédant un bloc de lignes pleines ou quasiment pleines, l'étape 2 de l'algorithme du degré minimum peut induire une mise à jour aussi coûteuse qu'inutile de la liste d'adjacence de variables associées au ligne quasiment pleines (voir par exemple [27]).

| Matrice | Ordre | Non-zéros | AMD | | AMDD | |
|---|---|---|---|---|---|---|
| | | | non-zéros dans $L+L^T$ $(\times 10^6)$ | temps sec. | Non-zéros dans $L+L^T$ $(\times 10^6)$ | temps sec. |
| Gupta 1 | 31802 | 2164210 | 4.080 | 483 | 4.066 | 9 |
| Gupta 2 | 62064 | 4248286 | 11.717 | 1414 | 11.911 | 8 |
| Gupta 3 | 16783 | 9323427 | 11.414 | 121 | 11.869 | 28 |

Tableau 1.2: Etude comparative de AMD et AMDD. Temps obtenus sur un processeur HP (PA-RISC 150 MHertz).

Amestoy, Davis et Duff ont exploré deux voies pour traiter ce problème. Une première solution consiste à pré-traiter la matrice pour supprimer les lignes quasiment denses. AMD est appliqué sur la matrice réduite, et des heuristiques pour réordonnancer les lignes quasiment denses peuvent alors être appliquées. Une autre solution (l'algorithme *AMDD*) modifie l'algorithme de degré minimum afin de détecter les lignes quasiment pleines dans la matrice réduite à chaque étape de l'algorithme. A l'étape 2 de l'algorithme du degré minimum, la liste d'adjacence des lignes sélectionnées est partiellement mise à jour afin de permettre un redémarrage récursif de AMD sur les lignes sélectionnées. Un gain significatif en temps peut être obtenu avec les deux approches. Ceci est illustré par les performances obtenues de l'algorithme AMDD (voir Table 1.2) sur des matrices provenant de [27].

### 1.1.2 Méthodes de Dissections Emboîtées

La méthode de dissections emboîtées [23] consiste à déterminer un ensemble de sommets $S$ qui sépare le graphe en deux parties $A$ and $B$ non voisines, numéroter $S$ avec les plus grands numéros disponibles, et recommencer récursivement sur les sous-graphes $A$ et $B$ jusqu'à ce que leurs tailles deviennent inférieures à une valeur limite. Cette numérotation garantit qu'aucun terme non-nul ne peut apparaître lors de la factorisation entre des inconnues de $A$ et des inconnues de $B$.

À la différence des méthodes de degré minimum, il est possible de prédire, au moins en ordre de grandeur, la qualité des renumérotations obtenues par dissections emboîtées pour les graphes usuels. On utilise pour cela les théorèmes de séparation, définis par Lipton, Rose, et Tarjan, qui donnent pour les familles de graphes les plus courantes (graphes planaires, tridimensionnels, graphes à genre borné, ...) des bornes sur la taille des séparateurs sommets et l'équilibrage des sous-parties obtenues [24, 36, 25] ; on a ainsi pu montrer analytiquement que la méthode de dissection emboîtées est proche de l'optimum pour les graphes à degré borné. Malheureusement, souvent ces théorèmes n'utilisent pas de preuves constructives, ni ne proposent d'algorithmes permettant d'atteindre les bornes prouvées.

De nombreuses techniques ont été proposées pour trouver des séparateurs sommets de petite taille qui équilibrent les sous-graphes autant que possible. Une première approche consiste à construire les séparateurs sommets à partir de séparateurs arêtes [34]. Pour trouver le plus petit séparateur sommet constitué d'extrémités d'arêtes de la coupe, on utilise classiquement la décomposition sous forme triangulaire par blocs ("*block triangular form*") du graphe biparti associé à la coupe, qui nécessite au préalable le calcul d'un couplage parfait de ce graphe. D'autres heuristiques de calcul de séparateurs sommets ont été proposées, basées sur la géométrie ou les méthodes spectrales [26], éventuellement dans un contexte multi-niveaux [32].

Jusqu'à récemment, les comparaisons effectuées entre les méthodes de degré minimum et de dissections emboîtées penchaient en faveur des premières. Cependant, le développement d'algorithmes efficaces de partition de graphes a conduit à inverser cette tendance. On a obtenu sur des problèmes de mécanique des structures, pour les méthodes de dissection emboîtées, une réduction moyenne du nombre d'opérations de 15 pour cent pour la factorisation de Cholesky par rapport aux méthodes de degré minimum [9, 47] ; de plus, ces méthodes résistent aux cas pathologiques mis en évidence pour les méthodes de degré minimum, notamment dans le domaine de la modélisation financière [10]. Enfin, le degré minimum donne généralement des arbres d'élimination de bien moins bonne qualité que ceux produits par la dissection emboîtée. En fait, la bonne approche consiste à mixer les deux méthodes pour profiter des avantages des deux numérotations : ainsi, dans la plupart des distributions logicielles proposées, seuls les premiers niveaux sont calculés par dissections emboîtées, les sous-graphes obtenus étant numérotés par degré minimum en dessous d'une certaine taille [32, 35, 45]. Les logiciels de référence dans ce cadre sont METIS [35] et SCOTCH [45, 46, 41]. Le tableau 1.3 compare ces deux logiciels avec MMD; NNZ est le nombre de termes non nuls dans $L$ et OPC est le nombre d'opérations à faire pour factoriser. Le seuil à partir duquel la dissection est arrêtée est de 30 sommets. Les figures 1.1 et 1.2 illustrent la variation de NNZ, OPC et de la hauteur moyenne de l'arbre d'élimination pour les graphes tests 598A et BCSSTK32 quand on augmente la valeur en nombre de sommets du seuil à partir duquel on bascule de la dissection emboîtée au degré minimum.

| Graphe | Scotch 3.2 | | | O-MeTiS 2.0 | | |
|---|---|---|---|---|---|---|
| | NNZ | OPC | Temps | NNZ | OPC | Temps |
| 144 | **5.04302e+07** | **6.30637e+10** | 54.18 | 5.14774e+07 | 6.63160e+10 | 32.07 |
| 3ELT | 9.34520e+04 | 2.91077e+06 | 0.44 | 9.78490e+04 | 3.43192e+06 | 0.27 |
| 598A | 2.80753e+07 | 2.15109e+10 | 32.85 | **2.80390e+07** | **2.10550e+10** | 22.69 |
| AUTO | **2.42459e+08** | **5.45092e+11** | 673.78 | – | – | – |
| BCSSTK29 | 1.95601e+06 | 4.98316e+08 | 5.05 | 2.00891e+06 | 5.22292e+08 | 2.88 |
| BCSSTK30 | 4.70394e+06 | 1.35818e+09 | 16.79 | 4.91682e+06 | 1.48445e+09 | 10.26 |
| BCSSTK31 | **5.22795e+06** | 1.84196e+09 | 12.64 | 5.27408e+06 | **1.72527e+09** | 7.73 |
| BCSSTK32 | 6.31493e+06 | 1.69557e+09 | 20.12 | 6.93198e+06 | 2.23416e+09 | 12.20 |
| BODY | 1.21614e+06 | 1.08353e+08 | 8.45 | 1.10336e+06 | **7.92548e+07** | 4.44 |
| BRACK2 | 7.38112e+06 | **2.54711e+09** | 15.07 | **7.20380e+06** | 2.56784e+09 | 9.19 |
| BUMP | **2.67353e+05** | **1.48797e+07** | 1.02 | 2.73569e+05 | 1.55700e+07 | 0.63 |
| M14B | **6.77465e+07** | **7.08264e+10** | 102.05 | 6.89223e+07 | 7.18790e+10 | 52.67 |
| PWT | **1.43140e+06** | **1.22120e+08** | 6.05 | 1.53126e+06 | 1.40441e+08 | 3.76 |
| ROTOR | **1.76431e+07** | **1.11101e+10** | 28.88 | 1.81358e+07 | 1.16120e+10 | 18.36 |
| TOOTH | **1.31559e+07** | **9.02613e+09** | 19.48 | 1.32536e+07 | 9.44619e+09 | 12.34 |

| Graphe | MMD | | |
|---|---|---|---|
| | NNZ | OPC | Temps |
| 144 | 9.81030e+07 | 2.62934e+11 | 28.30 |
| 3ELT | **8.98200e+04** | **2.84402e+06** | 0.06 |
| 598A | 4.63319e+07 | 6.43375e+10 | 19.96 |
| AUTO | 5.16945e+08 | 2.41920e+12 | 139.14 |
| BCSSTK29 | **1.69480e+06** | **3.930592+08** | 0.76 |
| BCSSTK30 | **3.84344e+06** | **9.28353e+08** | 1.58 |
| BCSSTK31 | 5.30825e+06 | 2.55099e+09 | 1.96 |
| BCSSTK32 | **5.24635e+06** | **1.10873e+09** | 2.14 |
| BODY | **1.00908e+06** | 8.93073e+07 | 1.25 |
| BRACK2 | 7.30755e+06 | 2.99122e+09 | 3.90 |
| BUMP | 2.81482e+05 | 1.76288e+07 | 0.12 |
| M14B | 1.22359e+08 | 2.56226e+11 | 44.07 |
| PWT | 1.59060e+06 | 1.75944e+08 | 0.70 |
| ROTOR | 2.60982e+07 | 2.86029e+10 | 11.27 |
| TOOTH | 1.56226e+07 | 1.65121e+10 | 5.06 |

Tableau 1.3 : Comparaisons entre Scotch 3.2, O-MeTiS 2.0 (arrêt de la dissection en dessous de 30 sommets) et MMD; les meilleures valeurs sont en gras et les temps sont mesurés pour un processeur R10000 190 MHertz.

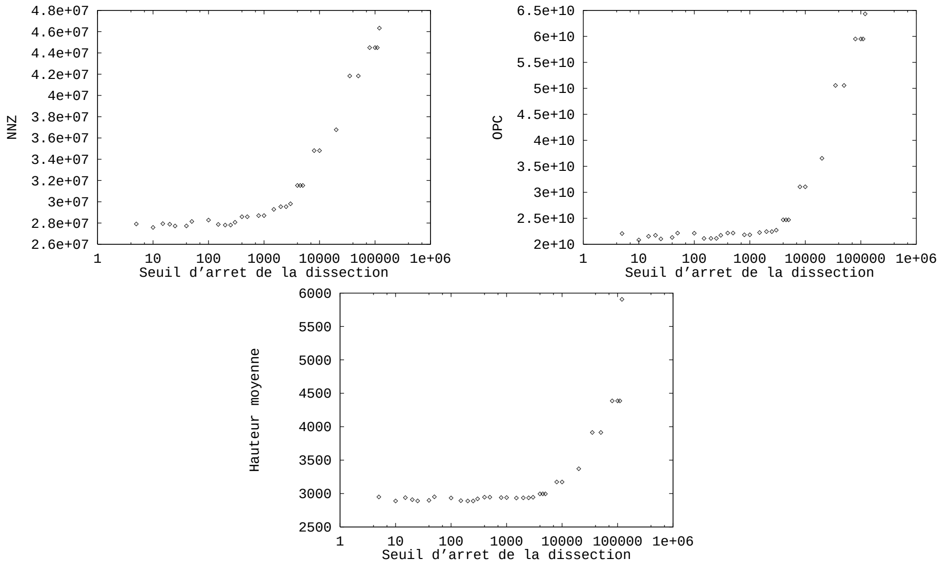


Figure 1.1 : 598A, $|V| = 110971$, $|E| = 741934$

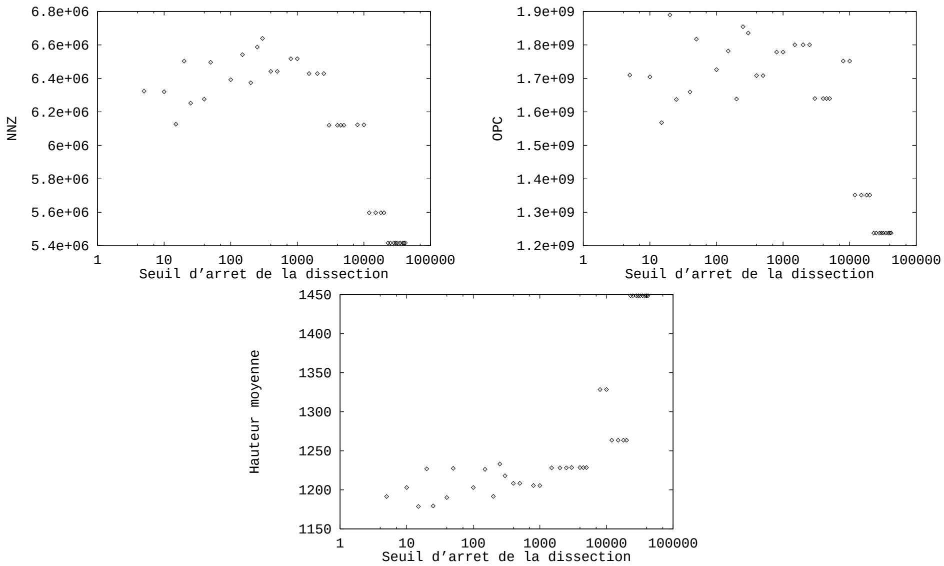


Figure 1.2 : BCSSTK32, $|V| = 44609$, $|E| = 985046$

Un couplage optimisé entre la dissection emboîtée produite par SCOTCH et le degré minimum produit par AMD fait l'objet d'une collaboration entre l'ENSEEIHT et le LaBRI.

### 1.1.3 Distribution des données et solveurs parallèles

Les solveurs séquentiels ou parallèles utilisés aujourd'hui sont nécessairement par bloc pour exploiter la puissance locale des processeurs via les primitives BLAS3.

Il y a essentiellement deux types d'approche algorithmique 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* [5, 6, 7, 8, 31, 42], et l'approche *multifrontale* [19, 3, 14, 31]. En ce qui concerne les méthodes supernodales, l'approche fan-in ou "left-looking" consiste à répercuter sur le bloc colonne courant les modifications des blocs colonnes de gauche concernés, puis à calculer la valeur finale du bloc colonne courant ; de manière symétrique, l'approche fan-out ou "right-looking" consiste à calculer définitivement le bloc colonne courant et à répercuter ses contributions sur les blocs colonnes de droite concernés. En parallèle, l'approche fan-in est plus performante car elle permet de limiter la masse de communication.

A partir de la structure de données par bloc calculée par la factorisation symbolique [12], la phase de prétraitement va donc calculer une bonne distribution des données sur les processeurs. On utilisera l'arbre d'élimination pour distribuer de manière équilibrée les blocs indépendants qui sont les plus bas dans l'arbre, puis on découpera les blocs situés dans les niveaux les plus hauts et qui représentent de manière inhérente la plus grande masse de calcul. Les sous blocs seront alors distribués de manière bloc cyclique pour utiliser le parallélisme contenu dans les calculs en dense. On construit ainsi de manière statique une bonne régulation des calculs [22, 30, 40, 20]. Cette technique a été utilisée selon un schéma de distribution 1D ou l'on distribue de manière élémentaire des blocs colonnes entiers. Schreiber [49] a montré que ce type de schéma limite la scalabilité des solveurs et qu'il faut en fait avoir un schéma de distribution 2D. C'est dans ce dernier cadre que l'on a atteint les meilleures performances connues à ce jour. Avec une approche supernodale, Rotherg et Schreiber [48] ont obtenu 2.8 GFlops sur un Paragon à 196 noeuds pour une matrice de taille 64000 correspondant à une grille 3D d'éléments finis. Par une approche multifrontale, Gupta, Karypis et Kumar [28, 29] ont obtenu entre 4.5 et 15.7 Gflops sur un T3D à 1024 noeuds sur des matrices de tailles variant entre 3136 et 42875. Dans les deux cas, les graphes associés aux matrices ont été renumérotés par dissections emboîtées couplées avec du degré minimum.

## 1.2 Régulation dynamique pour la factorisation des matrices creuses non-symétriques

La factorisation de matrices creuses peut être modélisée par un arbre d'élimination qui exprime les dépendances entre l'élimination des colonnes de la matrice. Dans le cadre de matrices symétriques définies positives la structure de cet arbre peut être calculée avant l'étape de factorisation numérique. Par contre, lors de la factorisation de matrices creuses non-symétriques, uniquement une estimation de la structure de cet arbre peut être connue à priori. En effet, le pivotage numérique peut modifier de façon significative la structure estimée.

Dans cette section nous décrirons l'impact du pivotage numérique aussi bien sur la structure de l'arbre d'élimination que sur les choix algorithmiques pour la parallélisation sur machines à mémoire distribuée de la factorisation. Ce travail (P. Amestoy, I. Duff et J.Y. L'Excellent) est développé dans le cadre du projet PARASOL EU ESPRIT IV LTR. Il s'appuie d'une part sur l'expérience acquise sur machine à mémoire partagée et virtuellement partagée ([2, 4], code MA41 dans la bibliothèque Harwell), d'autre part sur le développement à l'ENSEEIHT d'une version initiale pour machines à mémoire distribuée [21].

### 1.2.1 Arbre d'élimination

L'arbre d'élimination (voir par exemple [37]) exprime les connections/dépendances entre les différentes étapes de la factorisation. Il exprime donc le parallélisme induit par la structure creuse de la matrice.

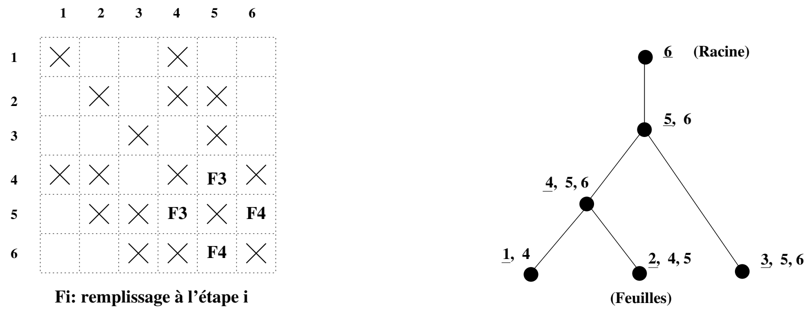


Figure 1.3: Exemple de construction d'un arbre d'élimination.

L'analyse de la structure de la matrice des facteurs ($\mathbf{A}_F$) permet de construire l'arbre d'élimination. Si l'on suppose que les pivots choisis sur la diagonale (et dans l'ordre) constituent une séquence de pivots numériquement stable, alors il est effectivement possible de construire l'arbre d'élimination avant l'étape de factorisation numérique. Chaque colonne correspond à un nœud du graphe. Chaque nœud $k$ de l'arbre correspond à la factorisation d'une matrice frontale dont la structure est celle de la colonne $k$ de $\mathbf{A}_F$. S'il n'y a pas de non-zéro sur la partie gauche de la diagonale de la ligne $j$ alors le nœud $j$ est une **feuille**. Le nœud $j$ est père de $k$ dans l'arbre si et seulement si $j$ est l'indice de ligne du premier élément non-nul sous diagonale dans la colonne $k$.

Pour mieux exploiter l'architecture des processeurs actuels (vectoriels ou scalaires) il est fondamental de détecter/exploiter toutes les sous-structures régulières manipulées par l'algorithme. On peut observer sur notre exemple, que le bloc d'indices $(3, 5, 6)$ constitue une structure pleine de la matrice des facteurs. Extraire les structures pleines de la matrices des facteurs peut aussi être vu comme une amalgamation des nœuds de l'arbre d'élimination. Ce pré-traitement (avant factorisation) du graphe permet effectivement d'accélérer de façon significative l'étape de factorisation. Chaque nœud de l'arbre amalgamé correspond ainsi à l'élimination d'un bloc de colonnes de la matrice initiale (voir nœud $(3, 5, 6)$ de la Figure 1.4). La parallélisation de la factorisation des matrices frontales pleines de l'arbre amalgamé offre un degré de parallélisme supplémentaires qui peut être combiné au parallélisme de l'arbre.

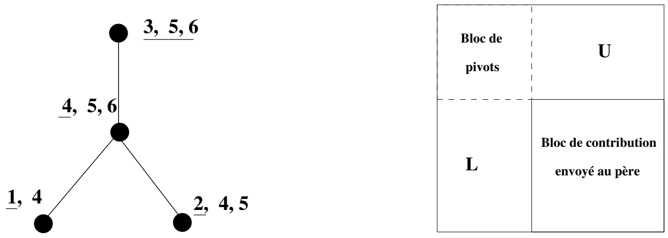


Figure 1.4 : Arbre amalgamé et structure générale d'une matrice frontale.

### 1.2.2 Arbre d'élimination et pivotage numérique

La structure de l'arbre d'élimination est en fait basée sur une estimation de la structure de la matrice des facteurs obtenue en choisissant les pivots sur la diagonale.

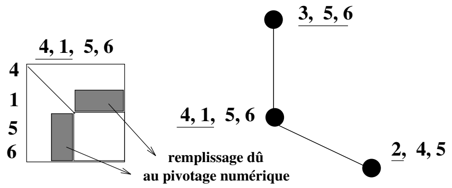


Figure 1.5 : Influence du pivotage numérique sur la structure de l'arbre.

La stabilité de la factorisation d'une matrice creuse non-symétrique est basée sur un contrôle de la taille relative du pivot (pivotage partiel avec critère de seuil, [17]). Si par exemple on suppose que $|a_{1,1}| << |a_{1,4}|$ dans notre matrice de test, alors le pivot $a_{1,1}$ ne peut plus être éliminé lors du calcul associé au nœud 1. Les modifications du graphe initial (voir Figure 1.5) conduisent à une modification de la taille des facteurs (les valeurs (1,5) et (1,6) en plus), de la structure de l'arbre, de la taille et du nombre de pivots des matrices frontales (élimination des pivots 4 et 1 au niveau du nœud 4). Il est à noter que les deux niveaux de parallélisme de la méthode (arbre et traitement d'un nœud) sont ainsi perturbés par le pivotage numérique.

### 1.2.3 Parallélisation sur machine à mémoire distribuée

Dans la première étape de l'algorithme, l'arbre d'élimination est distribué sur les processeurs de la machine cible. Cette étape conduit au regroupement de nœuds d'un sous-arbre sur le même processeur cible (voir Figure 1.6). Le parallélisme induit par l'arbre sera dit de *niveau 1*.

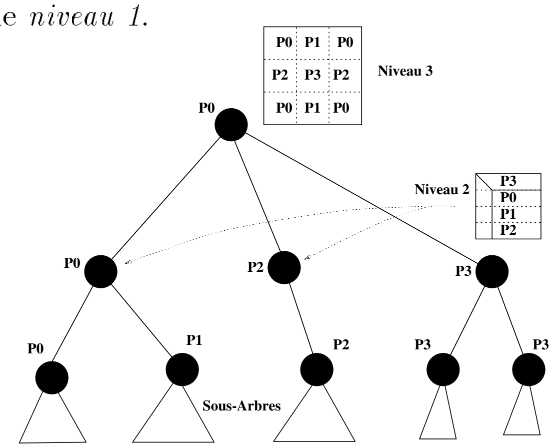


Figure 1.6 : Niveaux de parallélisme exploités durant la factorisation.

Pour pouvoir prendre en compte dynamiquement les problèmes liés au pivotage numérique deux types d'algorithmes de parallélisation des matrices frontales ont été développés:

1. *Niveau 2* : Partitionnement 1D des lignes associées au bloc de contribution de la matrice frontale
2. *Niveau 3* : Partitionnement 2D de la factorisation des matrices frontales.

Lors de l'algorithme de niveau 2, le processus maître associé au traitement du nœud est en charge uniquement de l'élimination du bloc de lignes des pivots. Il distribue alors dynamiquement des tâches par un découpage 1D des lignes associées au bloc de contribution. L'assemblage de la matrice frontale est alors automatiquement distribué sur les processus participant au travail. Le pivotage numérique est effectué localement par le processus maître qui peut factoriser le bloc de pivots indépendamment de la disponibilité des autres processus. Les matrices frontales candidates au niveau 2 sont typiquement celles qui possèdent un bloc de contribution de taille importante par rapport à la taille de la matrice frontale.

Le parallélisme de niveau 3 doit au contraire permettre de traiter les matrices frontales ayant un bloc de contribution de taille relativement faible par rapport à la taille de la matrice frontale (par exemple la racine). Le bloc de pivots estimé à l'étape d'analyse peut être assemblé sur la base d'informations statiques. Une correction (dynamique) sera cependant nécessaire pour prendre en compte les variables non-éliminées envoyées par les nœuds fils.

| | | Arbre complet | | | Arbre sans la racine | |
|---|---|---|---|---|---|---|
| Matrice | Ordre | Niv. 1 | Niv. 1+2 | Niv. 1+2+3 | Niv. 1 | Niv. 1+2 |
| GOODWIN | 7320 | 1.73 | 1.88 | 2.09 | 1.83 | 2.01 |
| K15 | 3948 | 1.98 | 4.58 | 10.0 | 2.35 | 11.1 |
| WANG3 | 26064 | 1.38 | 3.08 | 13.8 | 1.60 | 11.9 |
| WANG4 | 26068 | 1.67 | 2.78 | 20.3 | 2.45 | 18.2 |

Tableau 1.4 : Etude du parallélisme théorique (réordonnancement basé sur AMD).

L'apport théorique de chaque niveau de parallélisme est indiqué dans la Table 1.4. On peut noter le faible degré de parallélisme de l'arbre. Ceci doit pouvoir s'améliorer par l'utilisation d'un algorithme de réordonnancement combinant dissections emboîtées et degré minimum. Cet axe de recherche fait l'objet d'une collaboration en cours avec F. Pellegrini et J. Roman (LaBRI, Bordeaux). Les niveaux 2 et 3 offrent cependant un bon complément au parallélisme de l'arbre même s'il est clair que notre modèle simplifié ne donne qu'une borne supérieure du speed-up réel. Il sera notamment important de contrôler la taille des blocs lors des partitionnements 1D ou 2D des matrices frontales.

La version $\alpha$ du code distribué est développée en FORTRAN90 et s'appuie sur la couche MPI. L'analyse de performance des niveaux 2 et 3 est actuellement en cours.

# 1.3 Calcul adaptatif du grain dans les recouvrements calcul/communication

## 1.3.1 Introduction, problématique et travaux connexes

L'utilisation des machines parallèles à mémoire distribuée apporte un gain important en performances et en taille mémoire mais amène en contre partie un surcoût en communications. Afin d'obtenir des programmes performants et extensibles, il convient de masquer ce surcoût. Plusieurs solutions existent. La première consiste en un choix judicieux de la distribution des données qui réduira au maximum le nombre et la taille des communications. De plus, si les dépendances le permettent, on essaiera d'initialiser les communications au plus tôt et de manière asynchrone : pendant l'exécution de ces communications, on effectuera d'autres calculs, sans rapport avec l'échange de données. Des algorithmes ont été proposés pour recouvrir les communications en particulier dans le cas des transformées de Fourier [51]. Dans [43], la taille optimale des paquets est calculée pour des nids de boucles `DOACROSS` généraux. De nombreux travaux autour des compilateurs data-parallèles traitent ce problème [11, 44, 33].

Quand les calculs sont irréguliers, on fait souvent appel des techniques multithreads. Des threads de calculs sont lancés en parallèle avec des threads de communication. S'ils résolvent le problème des recouvrements dans le cas de calculs et de communications indépendants, ils ne traitent pas le problème du découpage et du calcul du grain dans le cas où les calculs dépendent des communications (et vice-versa). Ils sont malgré tout une solution intéressante et relativement transparente pour l'utilisation de communications asynchrones.

Notre étude concerne donc le cas où calculs et communications sont dépendants entre eux comme dans le schéma de la Figure 1.7.

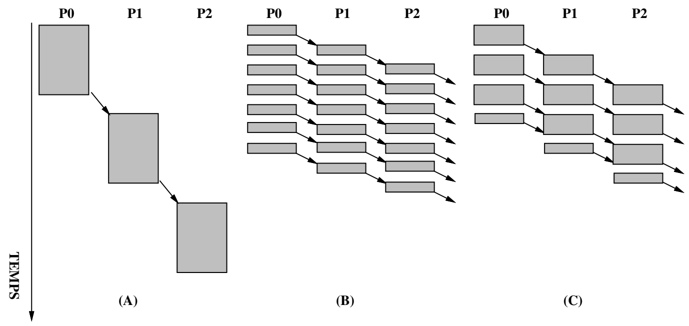


Figure 1.7 : Recouvrement des communications grâce à un pipeline

Il y a une séquentialité dans l'exécution (A). Le processeur P1 doit attendre que le processeur P0 ait fini de calculer pour recevoir les résultats du calcul et commencer à travailler. Une fois qu'il a terminé, il envoie les données résultats au processeur P2 qui peut, à son tour, travailler sur les données reçues. Le temps total est plus important que le temps séquentiel puisque l'on introduit le surcoût des communications. Une première solution consiste à commencer les communications au plus tôt, c'est-à-dire dès qu'un processeur a calculé une donnée. Pour chaque donnée calculée, un envoi est effectué et les processeurs commencent leur travaux dès que possible. On a donc un pipeline à grain fin (B). Cette solution introduit malgré tout un surcoût dû aux temps d'initialisation des communications. Ces temps sont toujours plus importants que le coût d'envoi d'un élément et il en résulte que le temps total peut être encore plus important que le temps séquentiel. Un compromis consiste donc à trouver le grain de calcul (et de communication) qui minimisera le temps total d'exécution. On aura alors un pipeline à grain moyen qui correspond à un schéma de type macro-pipeline (C).

Nous avons développé deux bibliothèques permettant un traitement efficace et portable de ce type de problèmes : la bibliothèque LOCCS[1] [15] qui permet d'effectuer les communications et la gestion de buffers pour les calculs pipelinés et la bibliothèque OPIUM[2] [16] qui permet de calculer le grain de calcul optimal en fonction des paramètres de l'algorithme et les caractéristiques de la machine.

Dans la suite de ce paragraphe, nous allons présenter le calcul de la taille optimale de paquet dans les cas réguliers et "irréguliers".

### 1.3.2 Cas régulier

Dans le cas régulier on peut expliciter une formulation du problème qui conduit à un schéma de calcul numérique du grain optimal. Le modèle permet de prendre en compte à la fois la complexité de l'algorithme et les caractéristiques physiques

[1]Low Overhead Communication and Computation Subroutines

[2]Optimal Packet sIze compUtation Methods

du calculateur. On peut, suivant la complexité de l'algorithme obtenir une solution analytique ou une solution numérique approchée. Notre travail consiste dans une première phase à instrumenter le code par des appels explicites aux primitives OPIUM ; la complexité de l'algorithme est passée comme argument. Il est à noter que, dans le cadre des problèmes d'algèbre linéaire réguliers, les complexités des algorithmes cibles sont généralement connues ou simples à calculer car elles dépendent de la taille des données et non des données elles-mêmes. Pour cela il sera nécessaire d'avoir une analyse du code donnant une estimation fine de la complexité de l'algorithme.

On obtient, sur l'exemple de la factorisation $LU$, des résultats encourageants [16]. L'efficacité du recouvrement nécéssite cependant une connaissance précise des paramètres de communication.

### 1.3.3 Calcul adaptatif

Dans le cas où les calculs n'ont pas tous la même taille, nous devons avoir, au sein d'une même étape de pipeline, des paquets de taille variables afin de garantir un équilibre dans les calculs et les communications et ainsi obtenir les meilleures performances. Nous devons donc calculer la suite des tailles de paquets qui conduit au temps d'exécution minimum. D'autres travaux existent sur des sujets proches comme dans [13], où une solution adaptative pour recouvrir les communications est proposée dans le cas du produit matrice-vecteur.

Nous allons maintenant décrire une méthode pour calculer une suite de taille de paquets qui maximise le recouvrement pour l'algorithme de factorisation de Cholesky. Le point de départ est l'algorithme séquentiel colonne classique que l'on modifie pour mettre en évidence le macro-pipeline :

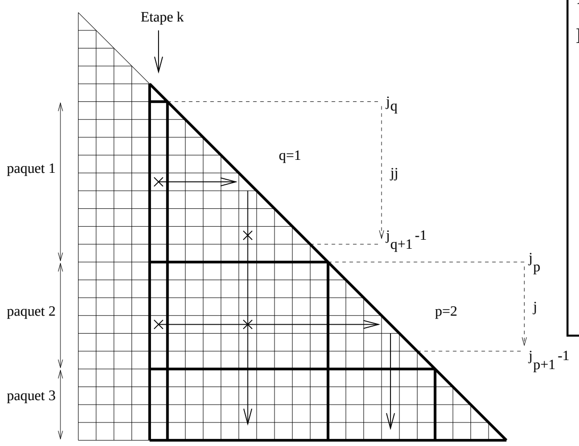


Figure 1.8 : Factorisation de Cholesky

Algorithme modifié par paquet :

Pour $k = 1$ à $n$ faire
 $a_{k,k} = \sqrt{a_{k,k}}$
 Pour $p = 1$ à $P_k$ faire
  Pour $j = j_p$ à $j_{p+1} - 1$ faire
   $a_{j,k} = a_{j,k} / \sqrt{a_{k,k}}$ (BLAS)
  Pour $q = 1$ à $p - 1$ faire
   Pour $jj = j_q$ à $j_{q+1} - 1$ faire
    Pour $j = j_p$ à $j_{p+1} - 1$ faire
     $a_{j,jj} = a_{j,jj} - a_{j,k} * a_{jj,k}$ (BLAS)
  Pour $j = j_p$ à $j_{p+1} - 1$ faire
   Pour $i = j$ à $j_{p+1} - 1$ faire
    $a_{i,j} = a_{i,j} - a_{i,k} * a_{j,k}$ (BLAS)

La redistribution des calculs correspond à une progression “ligne” dans la factorisation. La symétrie conduit à des tailles de paquets variables au cours du pipeline. A chaque étape de la factorisation, on découpe la colonne pivot en paquets de tailles inégales ; c'est cette suite de taille de paquets que l'on cherche à optimiser. Pour l'étape $k$, il y a $P_k$ paquets de taille $\nu_p$ correspondant aux indices de ligne $j_p$ à $j_{p+1}-1$ (Figure 1.8). On calcule les contributions engendrées par le paquet en cours (bloc diagonal) ainsi que le couplage avec les paquets précédents. Il est donc nécessaire de conserver les paquets précédement reçus tant que le pipeline n'est pas terminé.

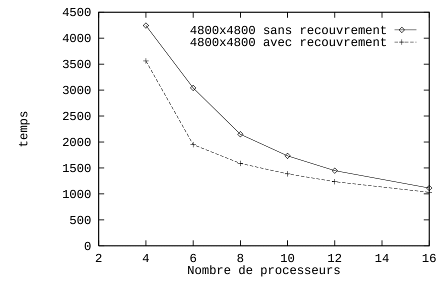


Figure 1.9 : Extensibilité sans BLAS 1

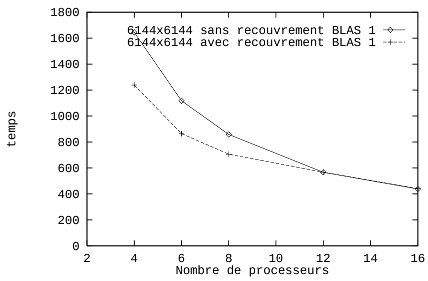


Figure 1.10 : Extensibilité avec BLAS 1

Les figures 1.9 et 1.10 montrent que l'influence du recouvrement devient significative dès que la masse de calcul à traiter par processeur est suffisante (pour de petits macro-pipelines, le nombre de paquets générés est trop faible pour avoir un recouvrement efficace).

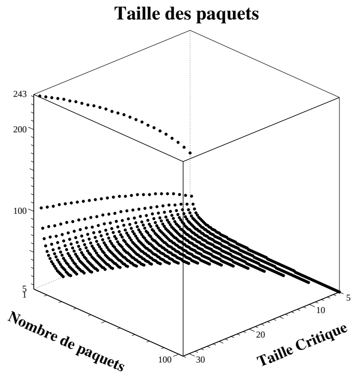


Figure 1.11 : Suite de taille de paquets

On appelle “calcul avant” ($Tcalc_A$) le travail qui doit être effectué sur un paquet avant de communiquer les résultats correspondants. De même, le “calcul après” ($Tcalc_B$) représente le travail qui doit être effectué à l'aide des résultats communiqués. On note $\tau_\times$ le temps élémentaire de mise à jour et $\tau_\div$ le temps élémentaire du calcul de la contribution.

Le processeur émetteur effectue les mises à jour de ses colonnes dans le “calcul avant”, de sorte que le “calcul avant” et le “calcul après” soient asymptotiquement équilibrés.

Pour un paquet $p$ de taille $\nu_p$, pour la colonne $k$, $1 \leq k \leq n$, $1 \leq p \leq P_k$, on a :

$$Tcalc_A(\nu_p) = \frac{\sum_{q=1}^{p-1} \nu_p \nu_q \tau_\times + \frac{\nu_p(\nu_p+1)}{2}\tau_\times}{NProcs} + \nu_p \tau_\div \quad \text{et} \quad Tcalc_B(\nu_p) = \frac{\sum_{q=1}^{p-1} \nu_p \nu_q \tau_\times + \frac{\nu_p(\nu_p+1)}{2}\tau_\times}{NProcs}.$$

En supposant que la taille des paquets est supérieure au nombre de processeurs, et grâce à une *distribution cyclique*, tous les processeurs sont concernés par les contributions.

Quand $p$ croît, la taille du calcul augmente comme $p\nu_p^2$, et donc nécessairement la suite de taille de paquets est décroissante. Pour expliciter le couplage entre les temps de calcul et de communication, qui évoluent de manière opposée, on impose la contrainte : $\forall p,\ Tcalc_A(\nu_p) + Tcomm(\nu_p) = C$, avec $C$ à déterminer (voir proposition 1.3.2). D'autre part, on considère un modèle linéaire de communication : $Tcomm(\nu_p) = \beta + \nu_p \tau$. Sous ces hypothèses, on démontre le résultat suivant :

**Proposition 1.3.1** *Si $L$ est la taille du pipeline, la suite de taille de paquets est donnée par :*

$$\begin{cases} \nu_1(C) = \frac{-b+\sqrt{b^2+4a(C-\beta)}}{2a} \\ \forall p \geq 2, \quad \nu_p(C) = \frac{\sqrt{b^2+4ap(C-\beta)}-\sqrt{b^2+4a(p-1)(C-\beta)}}{2a} \end{cases}, \text{avec} \begin{cases} a = \frac{\tau_\times}{2NProcs} \\ b = \frac{\tau_\times}{2NProcs} + \tau + \tau_\div \end{cases}.$$

*De plus le nombre de paquets du pipeline est donné par :* $P_k(C) = \frac{L(b+aL)}{C-\beta}$.

L'utilisation du noyau de calcul BLAS impose une taille minimale de paquets $\Lambda$. On cherche donc $C$ vérifiant les contraintes : $\begin{cases} \forall p, \nu_p \geq \Lambda, \\ \forall p, Tcalc(\nu_p) \geq Tcomm(\nu_p). \end{cases}$

On démontre le résultat suivant :

**Proposition 1.3.2** $C = \max(C_1, C_2)$ *avec :*

$$\begin{cases} C_1 = \beta + 2aL\Lambda + b\Lambda - a\Lambda^2 \geq 0 & \textit{obtenue avec la } 1^{i\grave{e}re} \textit{ contrainte,} \\ C_2 = \beta + \frac{2a\beta - 2b\tau + 4\tau^2 + 2\tau\sqrt{b^2-4b\tau+4a\beta+4\tau^2}}{2a} & \textit{obtenue avec la } 2^{i\grave{e}me} \textit{ contrainte.} \end{cases}$$

La Figure 1.11 illustre l'évolution des tailles de paquets, pour un macro-pipeline de taille 1024, en fonction de la taille critique imposée $\Lambda$.

La prise en compte des optimisations de type BLAS doit être poursuivie pour les BLAS de niveau 2 et 3, l'objectif est ici d'intégrer ces techniques de recouvrement dans des algorithmes par blocs comme ceux proposés dans ScaLAPACK, et de comparer les 2 versions.

L'utilisation de telles optimisations permet d'avoir de bons gains en performances mais pose de nombreux problèmes. Des problèmes théoriques puisqu'il faut être capable de calculer le grain optimal de calcul et communication. Et des problèmes logiciels puisque l'intégration dans une application réelle doit être la plus transparente, la plus portable et, bien entendu, la plus performante possible. L'optimisation d'algorithmes traitant des structures de données irrégulières complique ces calculs.

La suite de ce travail consiste tout d'abord à généraliser l'approche aux algorithmes par blocs et à traiter la factorisation de Cholesky pour des matrices creuses quelconques; dans ce contexte plus général d'irrégularité, l'objectif sera de calculer, à l'aide d'un *schéma adaptatif*, la suite optimale de taille de paquets. On intégrera ensuite ces techniques dans un schéma de compilation automatique pour le langage HPF (plateforme Adaptor). De plus, nous souhaitons par la suite tenir compte des hiérarchies mémoires et de l'utilisation de routines de calcul performantes comme les BLAS. Du point de vue des bibliothèques LOCCS, nous pensons regarder les approches multi-threads et messages actifs. Un équilibrage dynamique (à l'exécution) est également prévu.

# Bibliographie

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

[2] Amestoy (P.), Daydé (M.) et Duff (I. S.). – Use of level 3 BLAS in the solution of full and sparse linear equations. *In : Proceedings of the International Symposium on High Performance Computing, Montpellier, France, 22–24 March, 1989*, éd. par Delhaye (J.-L.) et Gelenbe (E.). pp. 19–31. – Amsterdam, 1989.

[3] Amestoy (P.) et Duff (I. S.). – Memory allocation issues in sparse multiprocessor multifrontal methods. *International Journal of Supercomputer Applications*, vol. 7, 1993, pp. 64–82.

[4] Amestoy (P. R.), Daydé (M. J.), Duff (I. S.) et Morère (P.). – Linear algebra calculations on a virtual shared memory computer. *Int. Journal of High Speed Computing*, vol. 7, 1995, pp. 21–43.

[5] Ashcraft (C.). – *The fan-both family of column-based distributed cholesky factorization algorithms*. – In Alan George, John R. Gilbert, and Joseph W.-H. Liu, editors, *Graph Theory and Sparse Matrix Computations*. Springer-Verlag, New York, NY, 1993.

[6] Ashcraft (C.), Eisenstat (S. C.) et Liu (J. W. H.). – A fan-in algorithm for distributed sparse numerical factorization. *SIAM Journal on Scientific and Statistical Computing*, vol. 11, 1990, pp. 593–599.

[7] Ashcraft (C.), Eisenstat (S. C.), Liu (J. W. H.), Peyton (B. W.) et Sherman (A. H.). – *A compute-ahead implementation of the fan-in sparse distributed factorization scheme.* – Rapport technique n° ORNL/TM-11496, Oak Ridge National Laboratory, Oak Ridge, TN, 1990.

[8] Ashcraft (C.), Eisenstat (S. C.), Liu (J. W. H.) et Sherman (A. H.). – *A comparison of three column based distributed sparse factorization schemes.* – Rapport technique n° YALEU/DCS/RR-810, Yale University, New Haven, CT, 1990.

[9] Ashcraft (C.) et Liu (J. W. H.). – *Robust ordering of sparse matrices using multisection.* – Rapport technique n° CS 96-01, Departement of Computer Science, York University, Ontario, Canada, 1996.

[10] Berger (A.), Mulvey (J.), Rothberg (E.) et Vanderbei (R.). – *Solving multistage stochastic programs using tree dissection.* – Rapport technique n° SOR-95-07, Princeton University, 1995.

[11] Brandes (T.) et Desprez (F.). – Implementing Pipelined Computation and Communication in an HPF Compiler. *In : Proceedings of Euro-Par'96, LNCS 1123.* pp. 459–462. – Springer Verlag.

[12] Charrier (P.) et Roman (J.). – Algorithmique et calculs de complexité pour un solveur de type dissections emboîtées. *Numerische Mathematik*, vol. 55, 1989, pp. 463–476.

[13] Colombet (L.), Michallon (P.) et Trystram (D.). – Parallel matrix-vector product on rings with a minimum of communications. *In : Parallel Computing*, pp. 289–310.

[14] Conroy (J. M.), Kratzer (S. G.) et Lucas (R. F.). – Multifrontal sparse solvers in message passing and data parallel environments - a comparitive study. *In : Proceedings of PARCO 93.*

[15] Desprez (F.). – A Library for Coarse Grain Macro-Pipelining in Distributed Memory Architectures. *In : IFIP 10.3 Conference on Programming Environments for Massively Parallel Distributed Systems.* pp. 365–371. – Birkhaeuser Verlag AG, Basel, Switzerland.

[16] Desprez (F.), Ramet (P.) et Roman (J.). – Optimal grain size computation for pipelined algorithms. *In : Proceedings of Euro-Par'96, LNCS 1123*. pp. 165–172. – Springer Verlag.

[17] Duff (I. S.), Erisman (A. M.) et Reid (J. K.). – *Direct Methods for Sparse Matrices*. – London: Oxford Univ. Press, 1986.

[18] Duff (I. S.), Grimes (R. G.) et Lewis (J. G.). – Sparse matrix test problems. *ACM Trans. Math. Software*, vol. 15, 1989, pp. 1–14.

[19] Duff (I. S.) et Reid (J. K.). – The multifrontal solution of indefinite sparse symmetric linear equations. *ACM Trans. Math. Software*, vol. 9, n° 3, 1983, pp. 302–325.

[20] Dumitrescu (B.), Doreille (M.), Roch (J. L.) et Trystram (D.). – *Two-dimensional Block Partitionings for the Parallel Sparse Cholesky Factorization : the Fan-in Method*. – Rapport technique n° 3156, INRIA Rhône-Alpes, France, Avr. 1997.

[21] Espirat (V.). – *Développement d'une approche multifrontale pour machines a mémoire distribuée et réseau hétérogène de stations de travail*. – Rapport technique n° Rapport de stage 3ieme Année, ENSEEIHT-IRIT, 1996.

[22] Facq (L.) et Roman (J.). – *Algèbre linéaire creuse : distribution par bloc pour une factorisation parallèle de Cholesky*. – In G. Authié, J. M. Garcia, A. Ferreira, J. L. Roch, G. Villard, J. Roman, C. Roucairol, and B. Virot, eds, Parallélisme et applications irrégulières, 135–147, Hermès, 1995.

[23] George (A.) et Liu (J. W. H.). – *Computer Solution of Large Sparse Positive Definite Systems*. – Englewood Cliffs, New Jersey: Prentice-Hall, 1981.

[24] Gilbert (J. R.). – A separator theorem for graphs of bounded genus. *Journal of Algorithms*, vol. 5, 1984, pp. 391–407.

[25] Gilbert (J. R.). – Some nested dissection order is nearly optimal. *Information Processing Letters*, vol. 26, 1988, pp. 325–328.

[26] Gilbert (J. R.), Miller (G. L.) et Teng (S. H.). – *Geometric mesh partitioning: Implementation and experiments*. – Rapport technique n° CSL-94-13, Xerox Palo Alto Research Center, 1994.

[27] Gupta (A.). – *Fast and effective algorithms for graph partitioning and sparse matrix ordering*. – Rapport technique n° TR RC-20496, IBM research division, T.J. Watson Research Center, Yorktown Heights, 1996.

[28] Gupta (A.), Karypis (G.) et Kumar (V.). – *Highly scalable parallel algorithms for sparse matrix factorization.* – Rapport technique n° 94-63, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994.

[29] Gupta (A.) et Kumar (V.). – *A scalable parallel algorithm for sparse matrix factorization.* – Rapport technique n° 94-19, Department of Computer Science, University of Minnesota, Minneapolis, MN, 1994.

[30] Gupta (A.) et Rothberg (E.). – An efficient block-oriented approach to parallel sparse Cholesky factorization. *SIAM J. Scientific Computing*, vol. 15, 1993, pp. 1413–1439.

[31] Gupta (A.) et Rothberg (E.). – An evaluation of left-looking, right-looking, and multifrontal approaches to sparse cholesky factorization on hierarchical-memory machines. *Int. J. High Speed Comput.*, vol. 5, 1993, pp. 537–593.

[32] Hendrickson (B.) et Rothberg (E.). – *Improving the runtime and quality of nested dissection ordering.* – Rapport technique n° SAND96-0868J, Sandia National Laboratories, Albuquerque, 1996.

[33] Hiranandani (S.), Kennedy (K.) et Tseng (C.). – Evaluating Compiler Optimizations for Fortran D. *Journal of Parallel and Distributed Computing*, vol. 21, 1994, pp. 27–45.

[34] Karypis (G.) et Kumar (V.). – *A fast and high quality multilevel scheme for partitioning irregular graphs.* – Rapport technique n° TR 95-035, Department of Computer Science, University of Minnesota, 1995.

[35] Karypis (G.) et Kumar (V.). – *METIS : Unstructured graph partitioning and sparse matrix ordering system.* – Rapport technique, Department of Computer Science, University of Minnesota, 1995.

[36] Lipton (R. J.), Rose (D. J.) et Tarjan (R. E.). – Generalized nested dissection. *SIAM Journal on Numerical Analysis*, vol. 16, 1979, pp. 346–358.

[37] Liu (J. H. W.). – The role of elimination trees in sparse factorization. *SIAM J. Matrix Anal. Appl.*, vol. 11, 1990, pp. 134–172.

[38] Liu (J. W. H.). – Modification of the minimum-degree algorithm by multiple elimination. *ACM Trans. Math. Software*, vol. 11, n° 2, 1985, pp. 141–153.

[39] Markowitz (M. H.). – The elimination form of the inverse and its application to linear programming. *Management Science*, vol. 3, Avr. 1957, pp. 255–269.

[40] Michel (J.), Pellegrini (F.) et Roman (J.). – Techniques de partitionnement, de renumérotation, et de distribution de données pour la factorisation parallèle de Cholesky de matrices creuses. *In : Second séminaire sur les techniques nouvelles de traitement des matrices creuses pour les problèmes industriels, USTL, Lille.*

[41] Michel (J.), Pellegrini (F.) et Roman (J.). – Unstructured graph partitioning for sparse linear system solving. *In : Proceedings of IRREGULAR'97, Paderborn, LNCS 1253*, pp. 273–286.

[42] Ng (E.) et Peyton (B. W.). – Block sparse cholesky algorithms on advanced uniprocessor computers. *SIAM J. Sci. Comput.*, vol. 14, 1993, pp. 1034–1056.

[43] Ohta (H.), Saito (Y.), Kainaga (M.) et Ono (H.). – Optimal Tile Size Adjustement in Compiling General DOACROSS Loop Nests. *In : International Conference on Supercomputing*, éd. par Press (A.). ACM SIGARCH, pp. 270–279. – Barcelona, Spain, Juil. 1995.

[44] Palermo (D.), Chandy (E. S. A.) et Banerjee (P.). – Communication Optimization used in the Paradigm Compiler for Distributed-Memory Mutlicomputers. *In : International Conference on Parallel Processing*, pp. 1–10. – St. Charles, IL, Août. 1994.

[45] Pellegrini (F.). – SCOTCH 3.1 *User's guide.* – Rapport technique n° 1137-96, LaBRI, Université Bordeaux I, Août. 1996.

[46] Pellegrini (F.) et Roman (J.). – Sparse matrix ordering with SCOTCH. *In : Proceedings of HPCN'97, Vienna, LNCS 1225*, pp. 370–378.

[47] Rothberg (E.). – *Exploring the tradeoff between imbalance and separator size in nested dissection ordering.* – Rapport technique, Silicon Graphics Inc, 1996.

[48] Rothberg (E.) et Schreiber (R.). – Improved load distribution in parallel sparse Cholesky factorization. *In : Supercomputing '94 Proceedings.*

[49] Schreiber (R.). – *Scalability of sparse direct solvers.* – Rapport technique n° RIACS TR 92.13, NASA Ames Research Center, Moffet Field, CA, Mai 1992.

[50] Tinney (W. F.) et Walker (J. W.). – Direct solutions of sparse network equations by optimally ordered triangular factorization. *Proc. of the IEEE*, vol. 55, 1967, pp. 1801–1809.

[51] Walker (D. W.). – Portable programming within a message-passing model: the fft as an example. *In : The Third Conference On Hypercube Concurrent Computers and Applications.* Geoffrey Fox California Institute of Technology.

Un autre *index* et *sous-index* ou *sous-index* ou encore *SOUS-index*.

# Annexe

# Index