**RENPAR'14 / ASF / SYMPA**
**Hamamet, Tunisie, 10 – 13 avril 2002**

# Optimisation de l'occupation mémoire pour un solveur parallèle creux direct hautes performances de type supernodal *

Pascal Hénon et Pierre Ramet

LaBRI (UMR CNRS 5800, ENSEIRB, Université Bordeaux 1) et action de recherche INRIA ScAlApplix
351 cours de la Libération, 33405 TALENCE, FRANCE
{henon|ramet}@labri.fr

**Résumé**
Cet article décrit une partie de nos travaux sur la résolution directe parallèle de grands systèmes linéaires creux. Il présente sa mise en œuvre sur des architectures à base de nœuds SMP, et plus particulièrement des techniques de gestion efficace de la mémoire.

**Mots-clés :** Algèbre linéaire creuse, solveur direct parallèle, ordonnancement statique, gestion mémoire.

## 1. Introduction

La faisabilité de la résolution des systèmes linéaires creux de grandes tailles est déterminante pour les applications industrielles. Nous avons présenté dans des travaux précédents [2, 3, 4], un solveur direct parallèle pour la factorisation de Cholesky basé sur un ordonnancement des calculs et des communications et acceptant une distribution par bloc mixte 1D/2D. Nous avons obtenu ainsi des performances de très bonne qualité sur une machine IBM SP2 [4]. Nous présentons tout d'abord dans la section 2 une extension de ces travaux pour des architectures à base de nœuds SMP qui a été validée sur une machine IBM SP3 [2]. Les deux sections suivantes présentent des mécanismes d'optimisation de la gestion mémoire qui permettront de traiter des cas industriels en vraie grandeur.

## 2. Stratégie d'ordonnancement et distribution mixte adaptées à une machine à nœuds SMP

### 2.1. Algorithmique

Notre algorithme parallèle de factorisation est une version supernodale de la factorisation $L.L^t$ avec agrégation locale des contributions. De ce point de vue, on se rapproche de la technique *Fan-In* [1] où les processeurs communiquent uniquement par des blocs agrégés de contributions. L'algorithme que nous proposons peut traiter à la fois des distributions par blocs-colonnes (1D) ou des distributions par blocs (2D) selon leur position dans l'arbre d'élimination par blocs [2].
Le code de la factorisation parallèle est découpé en tâches qui doivent être ordonnancées par la phase de distribution. Les tâches que nous avons définies peuvent être liées au calcul d'un bloc-colonne (distribution 1D) ou alors liées à une opération élémentaire entre blocs (distribution 2D).
Avant de définir ces tâches, nous introduisons quelques notations pour le bloc-colonne $k$, $1 \leq k \leq N$ :

- $\mathtt{BCol}(k)$ est l'ensemble des indices $i > k$ de blocs-colonnes en face des blocs extra-diagonaux du bloc-colonne $k$ dans la matrice $L$ ;
- $\mathtt{BRow}(k)$ est l'ensemble des indices $j < k$ de blocs-colonnes tels que $k \in \mathtt{BCol}(j)$ ;
- $map(i,k)$ est le processeur sur lequel est distribué le bloc $L_{ik}$ ;
- le symbole $\star$ signifie $\{j \in \mathtt{BCol}(k)\}$ ;
- soit $j \geq k$ ; la notation $[j]$ signifie $\{i \in \mathtt{BCol}(k) \cup \{k\}$ avec $i \geq j\}$ ;
- $AUB_{ij}$ : tout bloc agrégé de contribution pour le bloc $A_{ij}$. Nous noterons par extension $AUB_{[j]j}$ l'ensemble compacté des blocs agrégés de contribution d'un processeur donné pour le bloc-colonne $j$.

*Ce travail est soutenu par le CEA/CESTA et par le GDR ARP du CNRS.

Les calculs sur les blocs peuvent ainsi être classés en quatre types élémentaires : `COMP_1D` qui assure le traitement d'un bloc-colonne dans le cas d'une distribution 1D, et `FACTOR - BDIV - BMOD`, qui sont les tâches élémentaires pour le cas d'une distribution 2D. Les tâches associées sont alors définies comme suit dans la factorisation $L.L^t$ ($p$ est le numéro du processeur courant) :

– `COMP1D(k)` : *factoriser le bloc-colonne* $k$ *et calculer les contributions pour les blocs-colonnes de* $\mathrm{BCol}(k)$
  . Factoriser $A_{kk}$ en $L_{kk}L_{kk}^t$
  . Résoudre $L_{kk}L_{\star}^t = A_{\star k}^t$
  . **Pour** $j \in \mathrm{BCol}(k)$ **Faire**
  . Calculer $C_{[j]} = L_{[j]k}L_{jk}^t$ — *Calcul d'une contribution*
  . **Si** $map([j], j) == p$ **Alors** $A_{[j]j} = A_{[j]j} - C_{[j]}$
  . **Sinon** $\mathrm{AUB}_{[j]j} = \mathrm{AUB}_{[j]j} + C_{[j]}$ — *Agrégation d'une contribution*

– `FACTOR(k)` : *factoriser le bloc diagonal* $k$
  . Factoriser $A_{kk}$ en $L_{kk}L_{kk}^t$

– `BDIV(j,k)` : *mettre à jour le bloc extra-diagonal* $j$ *dans le bloc-colonne* $k$
  . Résoudre $L_{kk}L_{jk}^t = A_{jk}^t$

– `BMOD(i,j,k)` : *calculer la contribution du bloc* $i$ *dans le bloc-colonne* $k$ *pour le bloc* $i$ *dans le bloc-colonne* $j$
  . Calculer $C_i = L_{ik}L_{jk}^t$
  . **Si** $map(i, j) == p$ **Alors** $A_{ij} = A_{ij} - C_i$
  . **Sinon** $\mathrm{AUB}_{ij} = \mathrm{AUB}_{ij} + C_i$

Les performances du solveur sont liées d'une part à un choix pertinent pour la distribution 1D ou 2D d'un bloc-colonne, et d'autre part à un ordonnancement de l'exécution des tâches de calcul associées à la distribution qui permettra de recouvrir au mieux les communications par du calcul. L'approche que nous avons choisie consiste à décharger entièrement le solveur de ce travail, 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 [2, 4]. 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 super-nœuds 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 super-nœud un ensemble de processeurs candidats susceptibles de participer à son calcul. On peut décider qu'un super-nœud sera parallélisé selon un schéma 1D ou selon un schéma 2D. En règle générale, plus le nombre de processeurs est important, plus une parallélisation selon un schéma 2D est payante par rapport à une parallélisation en 1D [8]. Nous avons constaté expérimentalement que sur une machine donnée et pour un nombre de processeurs qui n'est pas aberrant par rapport à la taille du problème, *le nombre de processeurs candidats* était un bon critère pour passer d'une distribution 1D à une distribution 2D.Cette notion de processeur candidat est aussi 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 à partir d'une modélisation en temps des primitives BLAS et des communications MPI, l'estimation du temps 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].

L'ordonnancement des communications est compatible avec celui des tâches de calcul. Quand un bloc d'agrégation AUB reçoit sa dernière contribution, on dit que celui-ci est "complétement agrégé" et est donc prêt à être envoyé. Un ordonnancement efficace consiste à envoyer les AUBs dans l'ordre d'exécution des tâches qui les reçoivent ; ainsi l'envoi d'un AUB pourra éventuellement être retenu pour respecter cet ordre. En effet, ceci permettra de ne pas retarder le calcul de tâches urgentes par une mise-à-jour réalisées par des tâches moins prioritaires.

L'adaptation de nos algorithmes à une architecture SMP tient essentiellement dans la modélisation des communications. La première modification par rapport à une architecture classique à une seule hiérarchie mémoire consiste à prendre en compte, pour le calcul du coût d'une communication, le fait que les processeurs appartiennent ou non au même nœud SMP.
Un autre phénomène qui peut intervenir dans une architecture à base de nœuds SMP est que le débit et la latence varient selon le nombre de processeurs communiquant en même temps dans un nœud SMP. Notre algorithme de distribution ne nous permet pas de connaître le nombre de communications simultanées à un moment donné de la factorisation. Nous avons donc introduit un modèle de communication qui utilise des valeurs de latence et de bande passante qui varient selon le nombre de processeurs candidats qui participent au calcul d'un super-nœud de l'arbre d'élimination [2].

### 2.2. Validation expérimentale

| Nom | Colonnes | $NNZ_A$ | $NNZ_L$ | OPC | $NNZ_L$/OPC | Description |
|---|---|---|---|---|---|---|
| **GRID1023** | 1046529 | 4179980 | 5.615e+07 | 2.083481e+10 | 2.69e-3 | Regular 2D Mesh |
| **CUBE47** | 103823 | 1290898 | 4.828e+07 | 6.963850e+10 | 0.69e-3 | Regular 3D Mesh |
| **OILPAN** | 73752 | 1761718 | 8.912e+06 | 2.984944e+09 | 2.98e-3 | PARASOL |
| **MT1** | 97578 | 4827996 | 3.114e+07 | 2.109265e+10 | 1.48e-3 | PARASOL |
| **BMW3_2** | 227362 | 5530634 | 4.420e+07 | 3.007981e+10 | 1.47e-3 | PARASOL |
| **BMWCRA_1** | 148770 | 5247616 | 6.597e+07 | 5.701988e+10 | 1.16e-3 | PARASOL |
| **CRANKSG1** | 52804 | 5280703 | 3.142e+07 | 3.007141e+10 | 1.05e-3 | PARASOL |
| **SHIPSEC8** | 114919 | 3269240 | 3.572e+07 | 3.684269e+10 | 0.97e-3 | PARASOL |
| **CRANKSG2** | 63838 | 7042510 | 4.190e+07 | 4.602878e+10 | 0.91e-3 | PARASOL |
| **SHIPSEC5** | 179860 | 4966618 | 5.649e+07 | 6.952086e+10 | 0.81e-3 | PARASOL |
| **SHIP003** | 121728 | 3982153 | 5.872e+07 | 8.008089e+10 | 0.73e-3 | PARASOL |
| **THREAD** | 29736 | 2220156 | 2.404e+07 | 3.884020e+10 | 0.62e-3 | PARASOL |
| **AUDI** | 943695 | 39e+06 | 1.214e+09 | 5.266e+12 | 2.30e-4 | PARASOL |
| **COUP8000T** | 5.3e+6 | 374.5e+06 | 32.128e+08 | 20.90e+11 | 1.54e-3 | 3D Coupole CEA |
| **COUP5000T** | 3.3e+6 | 233.2e+06 | 20.269e+08 | 13.10e+11 | 1.54e-3 | 3D Coupole CEA |
| **COUP2000T** | 1326483 | 93734775 | 6.888e+08 | 5.00924e+11 | 1.37e-3 | 3D Coupole CEA |
| **COUP3000T** | 1989966 | 140.6e+06 | 10.333e+08 | 7.68324e+11 | 1.35e-3 | 3D Coupole CEA |

TAB. 1 – Description des cas tests. $NNZ_A$ est le nombre de termes non nuls dans la partie triangulaire inférieure de la matrice A, $NNZ_L$ est le nombre de termes non nuls dans la matrice L et OPC est le nombre d'opérations requises pour la factorisation. Les matrices sont triées par ordre suivant le rapport $NNZ_L$/OPC qui est une mesure de la réutilisabilité potentielle des données [6] .

Les algorithmes de repartitionnement et de distribution ainsi que le solveur parallèle ont été implémentés dans la bibliothèque PASTIX [2, 4] qui fait appel à la version 3.4 du logiciel SCOTCH [7] pour la renumérotation des matrices creuses (dissections emboîtées couplées au degré minimum approché).
La table 1 présente les caractéristiques de nos matrices tests et les résultats de leur renumérotation par SCOTCH. La plupart de ces matrices sont des matrices de mécanique des structures et de mécanique des fluides. Certaines dépassent un million d'inconnues et un téra-opérations pour la factorisation.
Les résultats que nous présentons sur la table 2 ont été obtenus sur l'IBM SP3 du CINES (28 nœuds SMP à 16 processeurs Power 3+/375 Mhz interconnectés par un switch Colony. Chaque nœud est doté de 16 Go de mémoire et développe une puissance crête théorique de 24 Gflops).
On peut voir que l'on a une scalabilité d'ensemble raisonnable. D'une manière générale l'efficacité est comparable, voir supérieure, aux résultats sur IBM SP2 [4] relativement aux puissances de crête, ce qui montre la bonne adaptation de l'algorithmique aux architectures SMP. Nous obtenons ainsi sur le cas COUP8000T près de 100 Gflops soit près de 50% de la puissance crête. D'autre part, ces résultats se comparent favorablement [2] à ceux obtenus par le solveur multifrontal de référence PSPASES [5], en particulier en termes d'occupation mémoire globale.

## 3. Optimisation de l'occupation mémoire par la technique d'agrégation partielle

Un problème important qui intervient dans les applications industrielles de grandes tailles est celui de la complexité en espace mémoire, en particulier à cause du surcoût mémoire dû aux structures supplémentaires qui interviennent dans la gestion des données distribuées et des communications. Dans le cas de la factorisation supernodale $L.L^t$ ou $L.D.L^t$ avec agrégation totale des contributions, ce surcoût

| Nom | Nombre de processeurs | | | |
|---|---|---|---|---|
| | 16 | 32 | 64 | 128 |
| **GRID1023** | **2.59** (8.03) | **1.72** (12.11) | **1.18** (17.58) | **.94** (21.98) |
| **CUBE47** | **7.08** (9.83) | **4.94** (14.07) | **3.91** (17.80) | **3.34** (20.84) |
| **OILPAN** | **.44** (6.66) | **.37** (8.05) | **.32** (9.28) | **.29** (10.29) |
| **MT1** | **1.91** (11.03) | **1.24** (16.96) | **1.04** (20.14) | **.94** (22.42) |
| **BMW3_2** | **3.30** (9.11) | **2.23** (13.47) | **1.69** (17.70) | **1.46** (20.48) |
| **BMWCRA_1** | **4.84** (11.76) | **2.90** (19.29) | **1.83** (29.72) | **1.48** (37.31) |
| **CRANKSEG1** | **3.08** (9.73) | **1.91** (15.70) | **1.58** (19.02) | **1.29** (23.17) |
| **SHIPSEC8** | **4.56** (8.07) | **3.64** (10.10) | **2.73** (13.48) | **2.50** (14.71) |
| **CRANKSEG2** | **3.97** (11.58) | **2.45** (18.73) | **1.93** (23.78) | **1.59** (28.93) |
| **SHIPSEC5** | **6.26** (11.09) | **4.43** (15.68) | **3.33** (20.82) | **3.03** (22.87) |
| **SHIP003** | **7.20** (10.28) | **4.82** (16.59) | **3.49** (22.89) | **2.84** (27.78) |
| **THREAD** | **4.16** (9.33) | **3.12** (12.41) | **2.77** (13.98) | **2.40** (16.14) |
| **COUP2000T** | **40.52** (12.36) | **19.67** (25.46) | **9.93** (50.43) | **5.80** (86.32) |
| **COUP3000T** | - | **29.48** (26.06) | **16.36** (46.96) | **8.67** (88.59) |
| **COUP5000T** | - | - | **26.49** (49.50) | **14.15** (92.57) |
| **COUP8000T** | - | - | **42.65** (49.00) | **22.49** (92.93) |

TAB. 2 – Performances de la factorisation sur IBM SP3 en double précision : le **temps** est en secondes et la puissance en Gigaflops.

consiste essentiellement dans l'allocation des blocs agrégés de contributions. En effet, un bloc agrégé $AUB_{ij}$ sur un processeur (voir paragraphe 2.1) consiste en un bloc recouvrant de toutes les contributions de ce processeur pour le bloc $(i, j)$ appartenant à un autre processeur. Ainsi un AUB doit être alloué dès le calcul de sa première contribution, et reste en mémoire jusqu'à ce que sa dernière contribution lui ait été ajoutée et qu'il soit effectivement envoyé. La somme en mémoire de tous les AUBs alloués simultanément peut devenir très importante par rapport à la mémoire nécessaire pour stocker les données locales de la matrice distribuée, et cette somme dépend naturellement du nombre total de processeurs. La table 3 montre le pourcentage de mémoire nécessaire pour le stockage des AUBs au cours du calcul par rapport à la mémoire nécessaire pour le stockage des cœfficients de la matrice. Comme on peut le voir, ce pourcentage est très important, surtout pour les cas issus de maillage en 3D (CUBE47, SHIP003, BMWCRA1), et a tendance à augmenter avec le nombre de processeurs (à l'exception du cas THREAD qui présente une géométrie atypique).

| | Nombre de processeurs | | | |
|---|---|---|---|---|
| Nom | 16 | 32 | 64 | 128 |
| **THREAD** | **264.0** | **325.5** | **366.0** | **172.2** |
| **SHIP003** | **95.0** | **129.5** | **205.9** | **330.9** |
| **CUBE47** | **103.7** | **233.5** | **332.8** | **344.9** |
| **BMWCRA1** | **9.1** | **23.2** | **41.4** | **55.0** |

TAB. 3 – Pourcentage de mémoire nécessaire pour allouer les AUBs par rapport à celle nécessaire pour les cœfficients de la matrice.

### 3.1. Algorithmique

Une solution à ce problème consiste à réduire le nombre d'AUBs alloués simultanément en *envoyant certains d'entre eux avant qu'ils ne soient totalement agrégés*. Ainsi un AUB partiellement agrégé pourra être envoyé et libérera de la mémoire jusqu'à ce qu'il soit réalloué lors du calcul de sa prochaine contribution. Cette méthode est appelée *agrégation locale partielle*.

La technique d'agrégation partielle est pénalisante en temps par rapport à l'agrégation totale : elle provoque plus d'envois de messages, de réallocations dynamiques et oblige à recevoir et ajouter en plusieurs fois les AUBs. Il convient donc de minimiser le nombre d'AUBs partiels, selon la contrainte mémoire, pour que le temps de factorisation parallèle ne soit pas trop augmenté. Nous proposons une solution qui, *avant la factorisation parallèle* et en fonction de l'ordre des tâches de calcul et de l'ordre d'envoi des AUBs (selon le schéma d'agrégation totale), modifie pour chaque processeur le schéma d'envoi de ses AUBs afin d'anticiper l'envoi de certains d'entre eux (le plus petit nombre possible) partiellement agrégés. On choisit systématiquement d'envoyer une contribution partiellement agrégée qui sera ré-accédée *le plus tard possible* selon l'ordre d'exécution des tâches de calcul.

### 3.2. Validation expérimentale

La figure 1 présente pour le cas test BMWCRA1 et par processeur, l'évolution de la mémoire allouée pour les AUBs au cours d'une factorisation sur 32 processeurs ; on utilise différentes contraintes de

réduction mémoire (par exemple 50% signifie que la mémoire nécessaire aux AUBs a été réduite de 50% par rapport à une agrégation totale). On peut voir sur cette figure qu'en agrégation totale (courbe 0% de réduction), le pic du surcoût mémoire est atteint ($\approx$ 5.5 Mo) vers le 3500$^{\text{ème}}$ ajoût d'une contribution dans un AUB. Les courbes 25%, 50% montrent le comportement de l'évolution de la mémoire obtenu lorsqu'on applique l'algorithme que nous avons décrit. On peut remarquer que cet algorithme permet de réduire la mémoire au plus juste pour respecter la limite mémoire imposée.
La figure 3 présentent pour 4 cas tests différents la pénalité en temps pour différents niveaux de réduction de la mémoire allouée aux AUBs. Elle montre l'intérêt de l'agrégation totale ; lorsque l'agrégation partielle est poussée jusqu'à un comportement proche du "fan-out" alors la dégradation en temps devient très importante. Ainsi pour la plupart des cas, lorsque l'on contraint les AUBs à une limite mémoire réduite de plus de 80% par rapport à l'agrégation totale, on observe une dégradation du temps de la factorisation supérieure à 100% du temps obtenu en agrégation totale. Par contre, comme on peut le voir sur ces exemples, une réduction de 50% du surcoût mémoire entraîne un surcoût en temps de factorisation qui varie sur 128 processeurs entre 2,69% et 28,4% par rapport au temps de factorisation en agrégation totale (ligne 0%) ; cette limite apparaît donc comme un bon compromis entre réduction du surcoût mémoire et augmentation du temps de factorisation. Dans ce contexte, notre régulation de la mémoire supplémentaire conduit globalement à une bonne scalabilité mémoire.
Cette technique nous a permis de factoriser le cas AUDI (plus de 5 tera-opérations), réputé difficile à cause de l'occupation mémoire qu'il occasionne. La matrice a été factorisée en 188s sur 64 processeurs Power3 avec 50% de réduction du surcoût mémoire, soit une puissance totale de 28 Gflops.

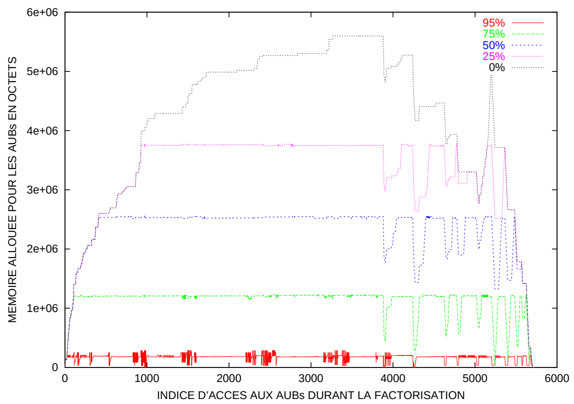


FIG. 1 – Courbe de réduction mémoire par agrégation partielle pour la BMWCRA1 sur 32 processeurs.

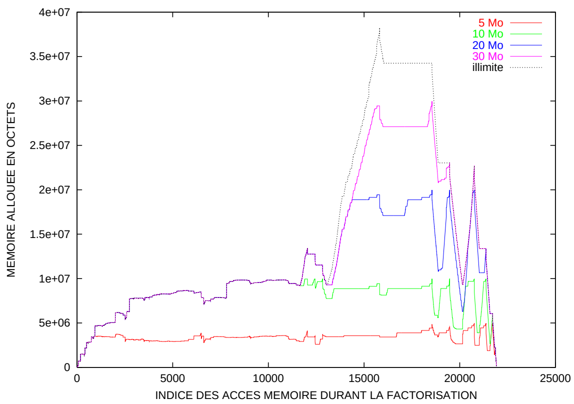


FIG. 2 – Courbes mémoires pour différentes limites mémoires lors de la factorisation "Out-of-Core" de la CUBE_47 sur 32 processeurs.

## 4. Vers une mise en œuvre "Out-Of-Core" hautes performances

Dans le paragraphe précédent, nous avons présenté une technique pour contrôler le surcoût mémoire induit par notre algorithme parallèle de factorisation avec agrégation des contributions. Le but était de contrôler la scalabilité mémoire via un mécanisme optimisé d'agrégation partielle en visant des temps de calcul du même ordre que pour une factorisation avec agrégation totale des contributions. Cependant, les grands challenges numériques actuels concernent des résolutions de systèmes qui comportent des dizaines de millions d'inconnues. La mémoire physique des machines n'est alors toujours pas suffisante pour des systèmes d'une telle grandeur. Pour pouvoir les traiter, il faut nécessairement mettre en place un mécanisme "Out-of-Core".

Le mécanisme "Out-of-Core" consiste à n'avoir en mémoire vive que les données nécessaires au traitement des calculs courants. Lorsqu'un calcul est terminé, les données qui lui sont associées sont écrites sur disque libérant ainsi de la mémoire pour recharger des données qui serviront aux calculs ultérieurs.

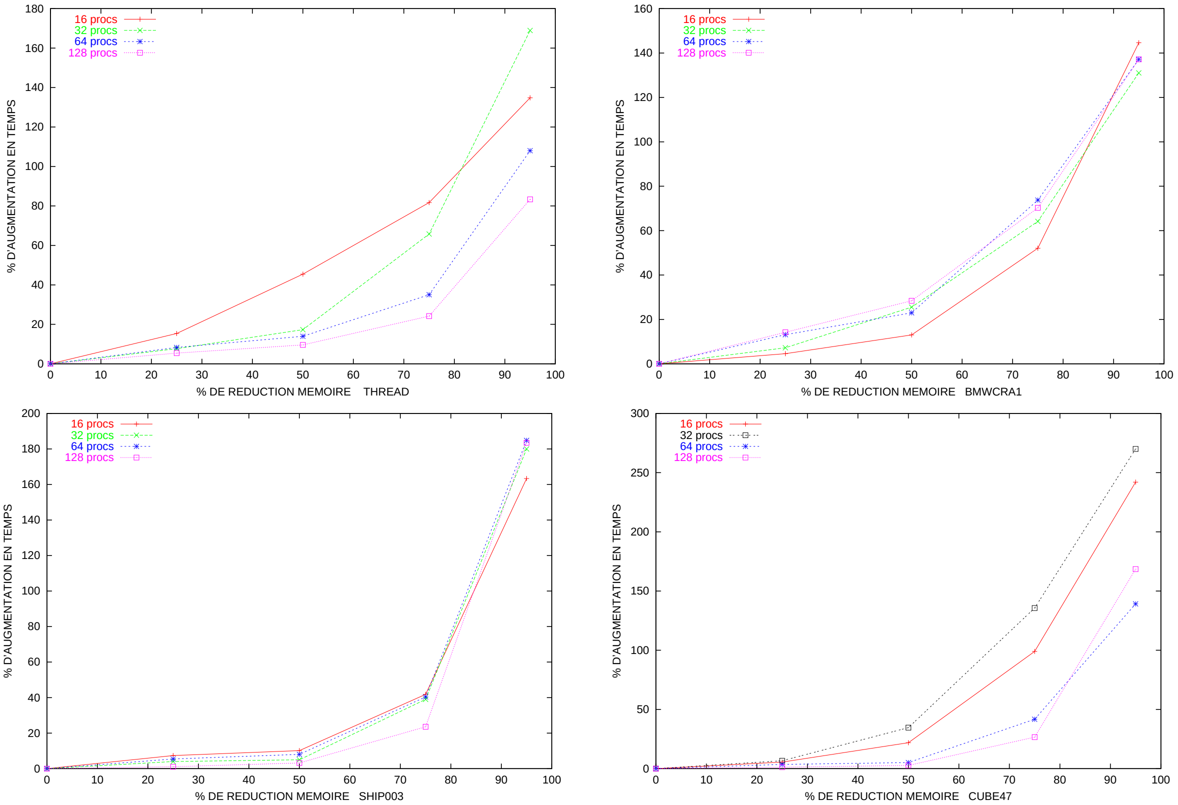


FIG. 3 – % d'augmentation du temps de factorisation / % de réduction mémoire

La technique "Out-of-Core" s'adapte naturellement à notre algorithmique : en effet, comme dans le cas de l'agrégation partielle, tous les accès aux données peuvent naturellement être anticipés par l'ordonnancement des tâches de calcul ; ainsi l'algorithmique utilisée sera très similaire.

Dans notre cas, le temps de lecture d'un bloc sera généralement plus coûteux que le temps de calcul associé à ce bloc. La performance de la factorisation "Out-of-Core" dépendra donc essentiellement du volume de données lues sur disque et du débit en lecture et en écriture *asynchrone* des disques. L'objectif de cette étude est de recouvrir les temps de lecture/écriture des blocs sur disque par les temps de calcul à l'aide d'un mécanisme performant de lecture/écriture asynchrone.

### 4.1. Algorithmique

Dans cette étude préliminaire, nous nous sommes limités au cas d'une distribution 1D de la matrice ; ce premier choix se justifie par le fait que la granularité des accès au disque doit être suffisante pour pouvoir profiter de toute la bande passante dans les échanges entre les disques et la mémoire.

Comme nous l'avons vu au paragraphe 3, les blocs de contributions agrégées peuvent représenter une part non négligeable des données allouées en mémoire. Cependant, le mécanisme d'agrégation partielle est incompatible avec un solveur "Out-of-Core" efficace ; en effet, les AUBs partiellement agrégés sont généralement émis par anticipation pour des tâches de calcul qui ne seront pas traitées tout de suite sur le processeur récepteur. Un tel mécanisme aurait donc pour effet d'obliger le chargement en mémoire de blocs-colonnes dont on n'a pas besoin pour les calculs imminents, et générerait donc beaucoup de va-et-vient entre la mémoire et le disque. On considèrera donc, dans le contexte "Out of Core", uniquement la stratégie d'agrégation totale.

Le mécanisme "Out-of-Core" doit donc prendre en compte à la fois les accès aux blocs-colonnes locaux de la matrice, et les accès aux blocs de contributions agrégées. Ainsi nous désignerons indifféremmemt par *bloc mémoire* les AUBs et les blocs-colonnes de la matrice. A partir de l'ordonnancement des tâches de calcul et de communication, l'ordre d'accès à tous les blocs mémoires est donc connu. Ces données accédées au cours du calcul d'un bloc-colonne $k$ sont :

– les blocs-colonnes locaux appartenant à $\mathtt{BCol}(k)$ qui reçoivent les contributions calculées au cours de la tâche de calcul `COMP_1D(k)` ;
– les AUBs dans lesquels sont ajoutées les contributions du bloc-colonne $k$ pour les blocs-colonnes non locaux.

La stratégie qui consisterait à ne remettre sur disque que les blocs-colonnes dont on ne se sert plus dans la suite du calcul (i.e. les blocs-colonnes déjà factorisés) peut se révéler dans beaucoup de cas insuffisante. En effet, l'allocation simultanée des AUBs et des blocs-colonnes appartenant à $\mathtt{BCol}(k)$ pour un bloc-colonne $k$ déjà factorisé, peut rapidement saturer la mémoire s'il n'y a pas de renvoi sur disque. Pour minimiser le nombre de va-et-vient des blocs mémoires entre le disque et la mémoire vive, il convient alors de choisir parmi les blocs présents en mémoire ceux qui seront réaccédés *le plus tard dans le calcul*.

La méthode, proche de celle utilisée pour l'agrégation partielle, consiste en prévoyant la taille de la mémoire occupée tout au long des accès mémoire, à choisir parmi les blocs présents en mémoire ceux qui devront lors de la factorisation être remis sur disque pour respecter la limite mémoire choisie. Ces blocs sont choisis en prenant ceux qui seront réaccédés (et auront donc besoin d'être rechargés en mémoire) le plus tard possible. Ce critère permet de minimiser, en fonction de la limite mémoire, le volume de données écrites et lues sur le disque.

### 4.2. Quelques estimations de notre étude préliminaire

Le tableau 4 montre la quantité de données lues sur disque en fonction de la limite mémoire que l'on se fixe, et ce pour une factorisation parallèle sur 32 processeurs. Les valeurs en mégaoctets sont par processeur. La colonne "Pic sans OOC" donne le pic de mémoire vive atteint lors de la factorisation avec agrégation totale des contributions et sans "Out-of-Core" ; les valeurs de cette colonne représentent donc la quantité de mémoire minimum nécessaire pour factoriser la matrice sans "Out-of-Core". La colonne "Pic avec OOC " correspond à une factorisation "Out-of-Core" qui ne lit les données locales de la matrice *qu'une fois* (les blocs sont lus à la volée et sont ré-écrits sur disque après leur factorisation). La valeur la plus à droite dans cette colonne correspond à la taille de la matrice locale. Dans les colonnes 50%, 25% et 10%, la valeur à gauche correspond au pic mémoire de la factorisation avec "Out-of-Core" (colonne "Pic avec OOC") réduit du pourcentage indiqué en haut de la colonne ; la valeur de droite correspond au volume de données lues sur le disque (nécessairement plus que la totalité de la matrice locale puisqu'il faut mettre en œuvre les allers-retours sur disque calculés par notre algorithme).

Une première constatation (naturelle) que l'on peut faire est que la quantité de mémoire nécessaire par rapport au volume des lectures nécessaires sur le disque est très variable selon le type de la matrice : dans le cas d'une matrice issue d'un (vrai) maillage 3D volumique, comme par exemple la CUBE47 ou l'AUDI, le rapport est beaucoup plus important que dans le cas des matrices issues de maillages 2D ou 3D surfacique, comme par exemple les maillages COUPOLE. Ceci est dû à deux phénomènes qui interviennent de façon plus prononcée sur les matrices de maillage 3D volumique. Tout d'abord la masse d'AUBs est beaucoup plus importante pour ces matrices (sur la AUDI la mémoire nécessaire pour une exécution est presque 2 fois supérieure au volume des données locales de la matrice). D'autre part, les premiers super-nœuds d'un graphe 3D volumique sont de grande taille (asymptotiquement en $\mathcal{O}(n^{\frac{2}{3}})$ ) ; ainsi la taille du premier super-nœud correspond, pour notre renumérotation, au nombre d'inconnues dans le dernier bloc plein de la matrice (c'est la racine de l'arbre d'élimination avant redécoupage). Lorsque l'on a besoin d'ajouter des contributions dans des blocs-colonnes appartenant à ce super-nœud redécoupé, il est nécessaire d'y accéder en mémoire. Ainsi la taille de bloc plein et le couplage des autres blocs-colonnes avec ces blocs pleins favorisent pour ces cas 3D volumique, une plus grande consommation de mémoire.

On constate ainsi sur le tableau 4 (colonne 50%), que pour diviser la taille mémoire nécessaire par environ 3.4 (622.8 / 183.8) pour l'AUDI, il faut lire environ 13.4 fois la taille de la matrice locale (4070.1 / 303.5) sur chaque processeur. Par contre, sur le cas COUP3000T, pour diviser la taille mémoire nécessaire par 100, il faut seulement lire environ 21 fois la taille de la matrice locale sur un processeur.

La figure 2 montre sur le cas CUBE_47, pour différentes limites mémoire et en appliquant le mécanisme de va-et-vient que nous venons de décrire, les courbes d'occupation de la mémoire vive obtenues lors d'une factorisation sur 32 processeurs. Ces courbes sont indicées par l'accès aux blocs-colonnes et aux

| Nom | Pic sans OOC | Pic avec OOC | | 50 % | | 25 % | | 10 % | |
|---|---|---|---|---|---|---|---|---|---|
| **CUBE47** | 46.1 | 38.3 | 12.1 | 19.1 | 74.3 | 9.6 | 139.5 | 3.8 | 379.3 |
| **SHIPSEC5** | 34.6 | 24.1 | 14.1 | 12.0 | 106.17 | 6.0 | 372.3 | 2.4 | 442.5 |
| **SHIP003** | 34.9 | 23.0 | 14.7 | 11.5 | 100.5 | 5.8 | 407.1 | 2.3 | 557.4 |
| **BMWCRA1** | 27.1 | 12.2 | 16.5 | 6.1 | 235.5 | 3.05 | 364.6 | 1.2 | 472.6 |
| **COUP2000T** | 207.3 | 24.5 | 172.2 | 12.3 | 873.3 | 6.1 | 3175.4 | 2.5 | 4312.1 |
| **COUP3000T** | 298.7 | 29.7 | 258.3 | 14.9 | 2226.7 | 7.4 | 3592.8 | 3.0 | 6264.4 |
| **AUDI** | 622.8 | 367.3 | 303.5 | 183.8 | 4070.1 | 91.9 | 10165.1 | 36.73 | 14756.5 |

TAB. 4 – Evolution (mémoire nécessaire / volume de lecture sur disque) pour une factorisation $L.L^t$ sur 32 processeurs.

AUBs. La courbe “illimité” correspond au pic mémoire obtenu pour une factorisation “Out-of-Core” sans limite mémoire. On voit sur cette courbe que le pic mémoire est atteint vers le 15000$^{\text{ème}}$ accès à un bloc mémoire et vaut environ 38Mo. Les autres courbes correspondent à l’occupation de la mémoire vive lorsqu’on fixe une limite mémoire qui oblige à utiliser le va-et-vient calculé au plus juste par l’algorithme “Out-of-Core”. On peut voir sur la courbe avec limite à 5Mo que pour ne pas dépasser la limite mémoire imposée, on est obligé de commencer à remettre des blocs mémoires sur disque très tôt dans le calcul.

## 5. Conclusion

Nous avons présenté dans ce papier une extension de notre solveur direct PASTIX aux architectures à base de nœuds SMP ainsi que sa validation par des expérimentations sur des cas de références et issus de problèmes industriels. Nous avons aussi présenté deux moyens de réguler la consommation de mémoire au cours de la factorisation parallèle. Le mécanisme d’agrégation partielle permet de contrôler les surcoûts mémoire dûs à l’agrégation totale des contributions et ainsi de préserver une certaine scalabilité mémoire. Le mécanisme “Out-of-Core”, dont nous avons fait ici l’étude préliminaire, devrait permettre de résoudre des systèmes de tailles très importantes. Dans ces cas, c’est l’ordonnancement statique des calculs qui permet d’anticiper l’évolution de la mémoire au cours de la factorisation et ainsi de choisir de manière optimale les blocs agrégés de contributions ou les blocs-colonnes qui doivent temporairement libérer la mémoire. Un autre problème intéressant sera d’adapter la structuration des blocs sur disque afin d’optimiser la localité des accès. La mise en œuvre de nos stratégies ”Out-of-Core” reposeront sur des primitives AIO (*Asynchronous Input/Output*) et sont en cours de validation.

## Bibliographie


1. 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.
2. P. Hénon. *Distribution des Données et Régulation Statique des Calculs et des Communications pour la Résolution de Grands Systèmes Linéaires Creux par Méthode Directe*. PhD thesis, LaBRI, Université Bordeaux I, Talence, France, November 2001.
3. P. Hénon and P. Ramet. PaStiX : Un solveur parallèle direct pour des matrices creuses symétriques définies positives basé sur un ordonnancement statique performant et sur une gestion mémoire efficace. In *ACTES RenPar’2001, Paris, France*, April 2001.
4. P. Hénon, P. Ramet, and J. Roman. PaStiX : A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2) :301–321, January 2002.
5. 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.
6. X. S. Li. *Sparse Gaussian Elimination on High Performance Computers*. PhD thesis, University of California at Berkeley, 1996.
7. F. Pellegrini, J. Roman, and P. R. Amestoy. Hybridizing nested dissection and halo approximate minimum degree for efficient sparse matrix ordering. *Concurrency : Practice and Experience*, 12 :69–84, 2000. Preliminary version published in *Proceedings of Irregular’99*, LNCS 1586, 986–995.
8. R. Schreiber. Scalability of sparse direct solvers. Technical Report TR 92.13, RIACS, NASA Ames Research Center, May 1992.