
Laboratoire de
Recherche en
Informatique
Amiens

# Etude d'un solveur parallèle pour la simulation de la houle

auteur :

Olivier **SOYEZ**

stage encadré par :

Gil UTARD et Cyril RANDRIAMARO

*dans le cadre de l'obtention du*

**Diplôme d'Etudes Approfondies**
**Informatique Parallèle et Répartie, Combinatoire**
**Université de Picardie - Jules Verne**

Année 2002

# Remerciements

Je remercie Cyril Randriamaro qui m'a orienté vers le domaine passionnant du parallélisme.

Je remercie Cyril Randriamaro et Gil Utard pour leurs conseils avisés, pour m'avoir fait bénéficier de leurs expériences et connaissances, et pour m'avoir fait découvrir le monde de la recherche.

Je remercie Olivier Cozette qui a consacré beaucoup de temps à résoudre de nombreux problèmes techniques qui sont survenus lors de cette étude.

Je remercie Abdou Guermouche pour son aide précieuse dans la compréhension et l'intégration du solveur MUMPS dans le logiciel REFONDE.

Je remercie Pierre Debaillon du CETMEF pour son aide sur la dernière version séquentielle du logiciel REFONDE.

Je remercie tout les membres du LaRIA, pour leur disponibilité, leur sympathie.

Enfin je remercie tout particulièrement mes camarades du **DEA** *Informatique* d'Amiens 2001-2002 avec lesquels j'ai passé une très agréable année, et le climat d'entraide qui a toujours régné tout au long de l'année.

# Table des matières

# Introduction

La connaissance de la houle revêt une grande importance pour l'ingénierie maritime. Son étude est facilitée avant tout par la création d'outils de modélisation opérationnels notamment dans le dimensionnement d'ouvrages maritimes ou encore dans les domaines cotiers, portuaires ou estuaires. En effet, la houle s'avère être un obstacle majeur à la navigation et aux opérations portuaires et elle influe largement sur les mouvements sédimentaires et l'évolution du trait de côte.

De ce fait, un outil informatique disposant d'une grande puissance de calcul et d'une grande souplesse d'utilisation permet de tester rapidement différents projets d'aménagement soumis à une large gamme de sollicitations de houle et de conditions de marées.

Le CETMEF[1] a donc développé un logiciel, nommé REFONDE, dont la fonction est de simuler la houle dans les grands bassins marins. Pour obtenir les différentes simulations, il faut résoudre des systèmes d'équations linéaires de très grande taille. Il est important de souligner que ces simulations étaient jusqu'à présent limitées en taille, et que l'on pensait d'après une étude [1] menée auparavant qu'un solveur pour de tels systèmes était intrinséquement séquentiel.

L'objectif de ce stage a été d'étudier les méthodes pour résoudre des systèmes plus grands afin de traiter précisemment des cas réels. Dans un premier temps, nous décrivons le logiciel Refonde et sa mise à jour, ensuite nous expliquons comment résoudre le problème de stockage des matrices de taille importante, puis comment minimiser le volume de calcul conséquent engendré par le solveur. Enfin nous présenterons une analyse du solveur que nous avons intégré dans Refonde.

1. Centre d'Etude des Techniques Maritimes et Fluviales 

# Chapitre 1

# Logiciel Refonde

## 1.1 Présentation

Ce logiciel est un code de calcul de la houle qui résout l'équation de Berkhoff, prenant en compte les phénomènes de diffraction / réfraction / réflexion, par la méthode des éléments finis [13] (cf annexe A.1). Le logiciel Refonde peut notamment prendre en compte des ouvrages réflechissants. L'utilisateur doit disposer à priori d'une bonne connaissance de la géométrie et de la composition de ces ouvrages afin de déterminer avec précision leur coefficient de réflexion.

Le calcul de la houle est ramené à la résolution de l'équation aux dérivées partielles de Berkhoff. Dans le cas de Refonde, cette équation est transformée en équations algébriques. Un système matriciel de la forme A.x = b est ainsi obtenu et résolu par le solveur.

La précision de résolution de l'équation de Berkhoff par la méthode des éléments finis nécessite bien souvent des maillages très denses : plus de 100 000 nœuds. Le maillage étant l'opération qui consiste à découper le domaine d'étude en un nombre fini de sous-domaines appelés éléments. Des tests de validation effectué au sein du CETMEF ont mis en évidence la nécessité de travailler avec des maillages conséquent. Ceci implique l'obligation de fournir à Refonde des matrices globales A de très grande taille (plusieurs Giga-octets).

Ces matrices ont les particularités suivantes :

- Elles sont composées d'un très faible nombre d'éléments significatifs (différents de zéro), on les nomme des ***matrices creuses***.
- Elles sont ***bande irrégulière homogène***,***symétrique***.

En fait le logiciel Refonde est couplé à un logiciel de Pré-traitement, `Prefonde` qui génère les fichiers de coordonnées des nœuds, les fichiers des éléments nécessaires à la visualisation du résultat et les fichiers de données qui sont les fichiers d'entrée contenant la matrice pour Refonde. `Refonde` traite la matrice afin d'obtenir les fichiers

solutions de la résolution qui seront exploités par un logiciel de post-traitement de visualisation : `postrait`.

Utilisation : le programme fournis se compile sous Linux. Le schéma de compilation et les erreurs majeurs que nous avons corrigées sont décrits respectivement en annexe A.2 et A.3.

Pour executer refonde sur une matrice donnée, il faut indiquer le nom du fichier d'extension 'INP' qui contient la matrice avec les spécifications la concernant. Une des particularité de ce fichier d'entrée est le Bloc SPEC qui contient entre autres les valeurs des différentes périodes de houle. Ce bloc est apparu avec le nouveau format de stockage I10, présenté au chapitre suivant.

A partir de la matrice d'entrée, refonde va générer pour chaque période de houle un fichier solution d'extension 'SOL'

Au début de l'exécution, le programme lit les différentes caractéristiques, paramètres et informations, contenus dans le fichier d'entrée. (cf annexe A.4, Représentation de l'exécution)

Puis le programme appelle la fonction exlinr, qui s'occupe de la phase d'assemblage et résolution de la matrice globale **vkg** :

Phase d'assemblage : c'est la fonction askg qui s'en occupe, à partir des matrices élémentaires **vke(idle,idle)** et de chaque vecteur de coordonnées kloce(idle) associé, voir figure 1.1. La matrice globale **vkg** est stocké dans un format de compression donné. Le vecteur solution est stocké dans **vfg**.

Phase résolution : c'est la fonction qui appelle un solveur (appel_solveur), cette phase correspond à la recherche de la matrice inconnue x, tel que : vkg . x = vfg

Remarque : pour chaque période de houle la structure de la matrice est invariable.

## 1.2 Mise à jour du logiciel

La première version de Refonde dont nous disposons, génère et utilise un Format de stockage SKYLINE " ligne de ciel " et un solveur " méthode de Gauss ", lisant les fichiers INP de format I5. Ces fichiers contiennent les matrices ayant un ordre inférieur à 50000. Cette version a été réalisée au sein du CETMEF.

Une seconde version plus optimisée de Refonde, dispose d'un format de stockage MORSE et un solveur MORSE. Le lecteur des fichiers d'entrée INP de type format I10 supporte les matrices ayant un ordre supérieur à 50000. Il comporte également

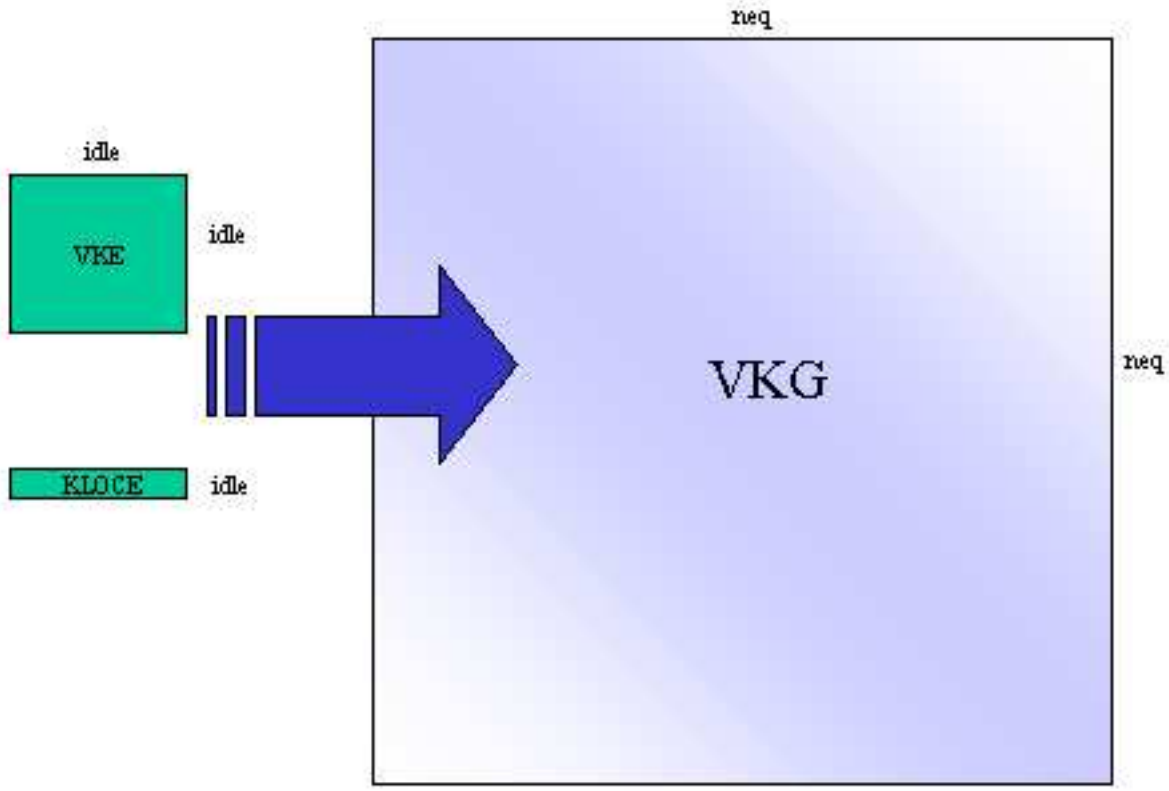



FIG. 1.1 – *Assemblage de la matrice globale vkg*

les fonctions d'assemblages, le solveur MORSE et diverses autres fonctions. Cependant, nous n'avons pas pu obtenir le code source complet, car cette version n'a pas été réalisée par le CETMEF et par conséquent elle est protégée par des droits d'auteurs concernant certaines parties. Nous avons modifié le lecteur initial de fichier INP de type format I5, en lecteur de fichiers I10 pour pouvoir travailler sur de grosses matrices.

Finalement pour pouvoir compiler un programme (" in-core ") : la taille des données en mémoire ne doit pas excéder 2 Giga-octets,par conséquent les deux critères nécessaires pour réduire les temps de calcul et permettre le traitement de cas importants sont les suivants :

- **Un format de compression adapté pour stocker des matrices creuses**; lors de cette étude, diverses versions du logiciel Refonde ainsi que différents formats de compression seront testés.
- **Un solveur parallèle efficace associé**.

# Chapitre 2

# Stockage des matrices

La résolution de très grands systèmes linéaires intervient dans de nombreuses applications scientifiques et industrielles. De ce fait, les matrices correspondant à ces systèmes linéaires nécessitent des tailles de stockage très importantes. Généralement ces matrices sont creuses (composée d'une majorité de zéros), voir fortement creuses, d'où l'obligation d'avoir recours à la compression pour pouvoir calculer / stocker des systèmes linéaires plus grand et échanger ces matrices via le réseau avec des temps de téléchargements raisonnables.

Nous allons donc procéder à une description des formats de compression existants (qui est une version résumée de l'annexe B) couplée à une analyse des matrices du CETMEF issues de simulation de calcul de houle sur la matrice test rot1, puis dans les ports de Brest, et le Havre.

## 2.1 Caractéristiques des Matrices

A chaque simulation correspond une matrice :
- Matrice rot1 : Calcul de houle sur la matrice test rot1.
- Matrice bres : Calcul de houle dans le port de Brest.
- Matrice havre : Calcul de houle dans le port du Havre.

Pour des résultats plus complets, se référer à l'annexe C.

Pourcentage d'éléments non nuls et Pourcentage d'éléments nuls :

| **Matrice** | **Nombre d'éléments non nuls** |
|---|---|
| Rot1 | 0.15% |
| Bres | 0.02% |
| Havre | 0.01% |

*Remarque :* on constate bien que les matrice testées sont fortement creuses, et que l'utilisation d'un format de compression adapté aura pour conséquence de stocker des matrices beaucoup plus importantes.

## 2.2 Définitions

### 2.2.1 Format COO

On notera l'existence du format COO (Coordinate) qui est le format de compression le plus simple à manipuler et à identifier. C'est le premier format de compression pour matrices creuses qui a été créé. De nos jours ce format est devenu obsolète et tend à disparaître.

### 2.2.2 Format SKYLINE

Le format SKYLINE, ou format " ligne de ciel " en français, a pour idée de supprimer les ensembles d'éléments nuls à gauche et à droite respectivement du premier et du dernier élément significatif de chaque ligne de la matrice de départ. Cette compression implique des contraintes au niveau de la structure même de la matrice, pour être performante.

Explication : Si aucun élément nul ne se trouve entre le premier et le dernier élément significatif de chaque ligne de la matrice de départ, alors ce format est le plus efficace parmi ceux étudiés. Si cette condition n'est pas respectée, alors ce format ne sera pas le plus efficace.

### 2.2.3 Format MORSE(CSR) et Format HARWELL-BOEING(CSC)

Pour le format MORSE la matrice des lignes fournit l'indice correspondant aux premiers éléments non-nuls de chaque ligne de la matrice de départ. Tandis que pour le format HARWELL-BOEING [10] la matrice des colonnes fournit l'indice correspondant aux premiers éléments non-nuls de chaque colonne de la matrice de départ.Le format CSC est *le format CRS pour la transposée de A*. Donc dans le cas de Refonde , le gain de compression est le même pour ces deux formats, car il traite des matrices carrées.

## 2.3 Analyses

Après compression dans le format SKYLINE, on étudie les matrices résultantes, ce format est utilisé par la première version de refonde. Pour comparer avec les autres formats, on utilisera la valeur que nous fournira le gain de compression obtenu.

Pour des résultats plus complets, se référer à l'annexe C.3.

Gain de compression en pourcentage du format SKYLINE :

| Matrice | Gain de compression |
|---|---|
| Rot1 | *97.85%* |
| Bres | *99.40%* |
| Havre | *99.66%* |

Rappel : Le format MORSE est utilisé par la dernière version de refonde. Le gain de compression obtenu est le même pour le format Harwell-Boeing.

Pour des résultats plus complets, se référer à l'annexe C.4.

Gain de compression en pourcentage du format Morse ou Harwell-Boeing :

| Matrice | Gain de compression |
|---|---|
| Rot1 | *99.68%* |
| Bres | *99.96%* |
| Havre | *99.97%* |

Bien que le gain de compression soit déjà appréciable dans le format Skyline, la matrice Havre par exemple ne fait plus que 0.34 % de sa taille d'origine; le format de compression SKYLINE n'est pas suffisamment adapté à ce type de matrice, et cela entraîne en conséquence le stockage de nombreux éléments nuls (environ 95% de moyenne d'une matrice compressée).

Le gain de compression du format MORSE ou HARWELL-BOEING est meilleur que pour le format SKYLINE, la matrice Havre par exemple ne fait plus que 0.03 % de sa taille d'origine; le format de compression MORSE ou HARWELL-BOEING est mieux adapté à ce type de matrice, car ces matrices sont bande irrégulière *fortement* creuse.

## 2.4 Conclusion

Dans la première version de Refonde, le format Skyline limitait fortement la taille des problèmes à résoudre.De plus, il faut savoir qu un exécutable ne peut pas traiter plus de 2 Go de données, on doit donc s'assurer que les matrices compressées ne dépassent pas cette limite. Cette restriction impose une limitation au niveau de l'ordre des matrices à traiter. Nous allons voir comment influe cette limitation en fonction des différents formats de compression étudiés :

– Format SKYLINE : on sait qu'une matrice d'ordre 50000 nécessite 1.22 Go. Donc une matrice d'ordre 500000 nécessiterai 12.20 Go. En fait, lorsque l'ordre de la

matrice excède 81967, on dépasse les 2 Go autorisés en mémoire par le compilateur : on ne peut pas générer l'exécutable!
Donc toutes les versions de REFONDE stockant les matrices en Format SKYLINE sont limités à des matrices d'ordre 81967 nœuds maximum.

– Format MORSE ou HARWELL-BOEING : une matrice d'ordre 50000 nécessiterai 0.11 Go. Donc une matrice d'ordre 500000 nécessiterai 1.08 Go. Donc il faudrait dépasser un ordre de 925926 pour dépasser les 2 Go autorisés en mémoire par le compilateur!

Finalement le format Morse ou Harwell-Boeing compressent la matrice par l'élimination de toutes les valeurs non nulles. Ces formats sont donc bien adapté aux matrices testées et permet de traiter des matrices ayant un ordre conséquent.

# Chapitre 3

# Minimisation du volume de calcul

Refonde, fondée sur la méthode des éléments finis, implique la résolution de très grands systèmes d'équations linéaires du type A.x=b, où A est une matrice creuse; notons que le temps de résolution de ces systèmes correspond à la majeure partie du temps global d'execution du programme.

## 3.1 Résolution des systémes linéaires

### 3.1.1 Présentation

Il existe deux grandes familles de méthodes pouvant résoudre les systèmes linéaires :

1. Méthode itérative : Le calcul est effectué plusieurs fois jusqu'à convergence vers la solution. Le nombre d'opérations n'est pas connu à l'avance, il dépend du type de système d'équation considéré. Par contre, la méthode permet de résoudre économiquement des systèmes de grande dimension, via par exemple un stockage partiel sur disque dur.
2. Méthode directe : La résolution du système est directe et rapide, mais sans réellement inverser la matrice, en un nombre fini d'opérations connues à l'avance, qui dépend de la taille de la matrice. Une méthode classique est la méthode de factorisation LU et dans le cas de matrice symétrique la méthode de Cholesky. Cependant ces méthodes ne sont applicables que sur des matrices régulières.

Rappel :

- Une matrice est régulière, si son déterminant est différent de zero.
- Une matrice est singulière, si son déterminant vaut zero.

### 3.1.2 Différentes méthodes directes

#### 3.1.2.1 Factorisation LU

Cette méthode décompose la matrice A sous forme du produit d'une matrice triangulaire inférieure L à diagonale unité par une matrice triangulaire supérieure U :

A=LU

$$L = \begin{pmatrix} L_{00} & 0 \\ L_{10} & L_{11} \end{pmatrix}, U = \begin{pmatrix} U_{00} & U_{01} \\ 0 & U_{11} \end{pmatrix}$$

Les valeurs de L et U sont obtenues par la résolution des équations suivantes :

$$\begin{cases} l_{jj} = 1 & \\ u_{ij} = a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj} & \text{pour } 1 \leq i \leq j \\ l_{ij} = \frac{1}{u_{ij}} \left( a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \right) & \text{pour } j \leq i \leq N \end{cases}$$

Le système initial A=LU devient :

$$LUx = b$$

soit :

$$Ly = b \ (1)$$
$$Ux = y \ (2)$$

On résout le système (1) pour trouver le vecteur y, puis le système (2) pour trouver le vecteur x. La résolution est facilitée par la forme triangulaire des matrices.

L'algorithme 1 représente la factorisation LU d'une matrice donnée A.

**Algorithm 1** algorithme général de la factorisation LU

```
for k=1 to n do
  for i=k+1 to n do
    a_ik = a_ik/a_kk {diviser les elements sous la diagonale dans la colonne k par a_kk}
  end for
  for j=k+1 to n do
    for i=k+1 to n do
      a_ij = a_ij − a_ik ∗ a_kj {mise à jour des élément dont la colonne n'a pas encore été traitée}
    end for
  end for
end for
```

On constate que lors d'une factorisation LU, les colonnes de la matrices sont parcourues de la gauche vers la droite et que les lignes sont parcourues du haut vers le bas.

Il existe deux principaux algorithmes pour la factorisation LU, ces derniers different l'un de l'autre par le moment ou sont effectuées les mises à jour :

- L'algorihme *rigth-looking* qui consiste à mettre à jour pour chaque pivot éliminé les éléments qui n'ont pas encore été traités. (L'algorithme décrivant la factorisation LU donné précédemment est un algorithme *rigth-looking*)
- L'algorithme *left-looking* qui effectue la mise à jour d'une colonne que lorsque le pivot (élément sur la diagonale) de celle-ci doit être éliminé.

**Algorithm 2** algorithme *left-looking*

**for** k=1 to n **do**
  **for** j=1 to k-1 **do**
    **for** i=j+1 to n **do**
      $a_{ik} = a_{ik} - a_{kj} * a_{ij}$ {mise-à-jour des élément de la colonne du pivot qui va être éliminé}
    **end for**
  **end for**
  **for** i=k+1 to n **do**
    $a_{ik} = a_{ik}/a_{kk}$ {diviser les elements sous la diagonale dans la colonne k par $a_{kk}$}
  **end for**
**end for**

Il faut savoir que les algorithmes *rigth-looking* sont plus parallèles que les algorithmes *left-looking*.

- Problème de pivotage : La factorisation LU est le point central de la résolution des systèmes creux. Dans le cas où l'on doit traiter des matrices non "symétrique définie positive", la factorisation doit se faire avec pivotage pour assurer la stabilité numérique du système. Le pivotage consiste à effectuer une permutation entre lignes si l'élément qui se trouve sur la diagonale ( le pivot) est inférieur à un seuil de manière à placer l'élément ayant la valeur la plus importante de la colonne sur la diagonale. Si on considére par exemple la factorisation de la matrice $\begin{pmatrix} 10^{-15} & 1 \\ 1 & 2 \end{pmatrix}$, elle va donner dans le cas où on ne pivote pas la matrice $\begin{pmatrix} 1 & 1 \\ 10^{15} & 10^{15} \end{pmatrix}$. On peut constater alors qu'il y a eu perte d'information au niveau des éléments de la deuxiéme ligne. Cette situation peut etre évitée en permutant les deux lignes. Ainsi le nouveau pivot sera 1 et la stabilité numérique de la factorisation sera assurée.

#### 3.1.2.2 Méthode de Cholesky

Cette méthode s'applique sur des matrices symétriques définies positives. Une matrice symétrique A est dite définie positive si, pour tout vecteur x, le produit $x^T Ax$

est positif. Pour une telle matrice, il est possible de trouver une matrice triangulaire inférieure L, telle que :

$$A = LL^T$$

L peut être considérée comme une sorte de racine carrée de A. Cette décomposition permet de :

- calculer la matrice inverse de A.
- calculer det(A), égal au carré du produit des éléments diagonaux de L.
- résoudre le système Ax = b, selon un principe analogue à celui de la factorisation LU.

Description de la résolution :

Soit à résoudre : A.x =b.

Tout d'abord on transforme la matrice A en un produit de deux matrices triangulaires $A = L.L^T$, cette etape de triangulation est la partie la plus longue du processus.
La seconde étape concerne la résolution proprement dite, on parle de " descente / remontée ". On résout successivement L.y = b, ce qui détermine le vecteur y, puis $L^T.x = y$.
Ces deux résolutions sont immédiates grace à la forme triangulaire des matrices : il suffit de résoudre chaque ligne successivement en commençant par le haut où la ligne se résume à un scalaire non nul, d'où le qualificatif de descente, pour la seconde matrice (transposée de la première) on part du bas, d'où le qualificatif de remontée.

## 3.2 Algorithmes de renumérotation (orderings)

Le remplissage (fill-in) : Soit A une matrice creuse d'ordre N. La factorisation LU de A peut être effectuée en utilisant l'algorithme 1. Il est important de signaler que pour le cas creux, la factorisation peut occasionner une augmentation de la taille mémoire de la matrice . Ce phénomène est appelé *remplissage*. Un exemple illustrant le remplissage occasionné par une factorisation est donné en figure 3.1.

Afin de réduire l'impact du phénoméne de remplissage engendré lors de la phase de factorisation, les solveurs pour matrices creuses utilisent en général des algorithmes de renumérotation (réordonnancement). Leurs fonctions est de trouver des permutations de lignes et de colonnes afin de réduire le remplissage.

L'intérêt des algorithmes de renumérotation est illustré dans les figures 3.2 et 3.3. En effet, dans cet exemple on considère une matrice 5x5 (représentant un cas classique des conséquence du remplissage). Ainsi, la figure 3.2 montre le résultat de la factorisation si la matrice était traitée sans tenter de réduire le remplissage. On remarque que la matrice obtenue après factorisation est pleine. La figure 3.3 quant a elle, montre le résultat de la factorisation de la même matrice sur laquelle on a effectué des permutations visant à minimiser le remplissage. Ainsi, après avoir permuté certaines lignes (resp. colonnes) la matrice résultat a exactement la même taille que la matrice initiale.

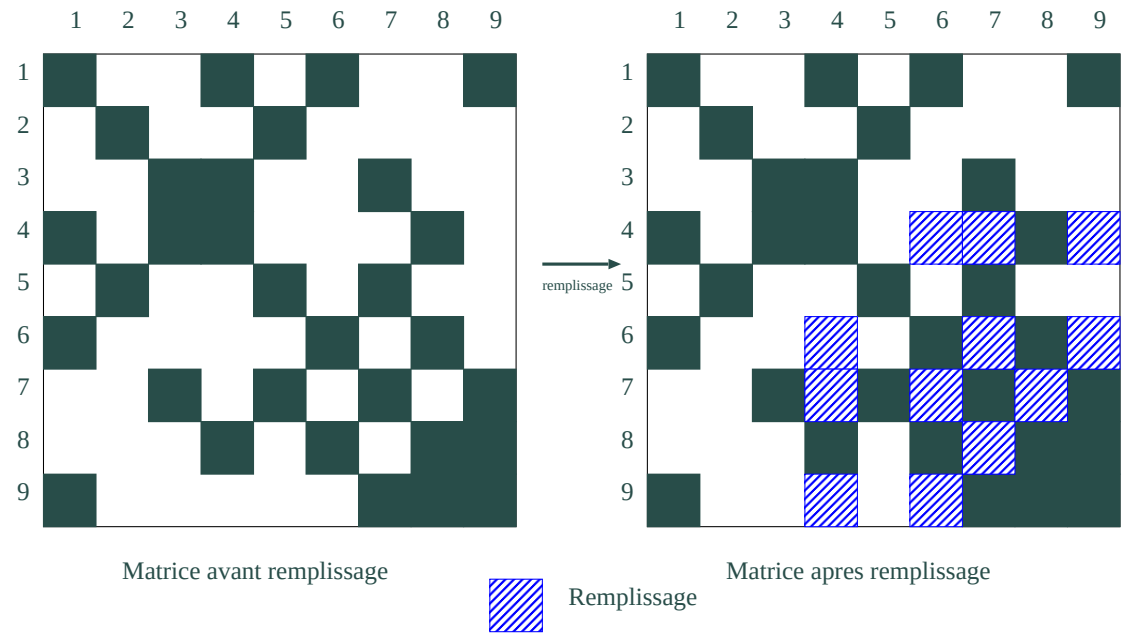



FIG. 3.1 – *Exemple de remplissage sur une matrice creuse 9x9*

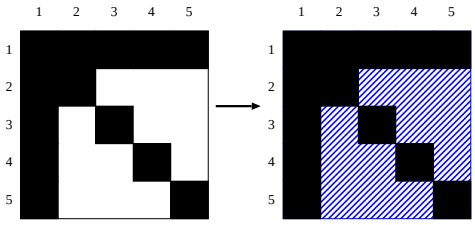


FIG. 3.2 – *sans réordonnancemment*

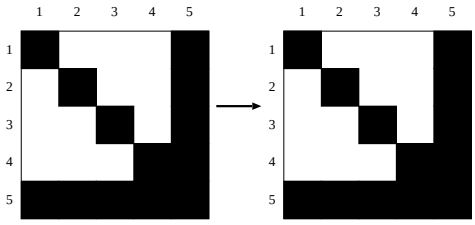


FIG. 3.3 – *avec réordonnancemment*

Les algorithmes de renumérotation sont des heuristiques de permutations de lignes (resp. colonnes) visant essentiellement la réduction du remplissage. Ce sont des méthodes qui manipulent le graphe de la matrice pour trouver les permutations à effectuer.

Il existe trois grandes familles d'algorithmes de renumérotation :

1. Les méthodes globales : sont des méthodes de partitionnement de graphe. Ces méthodes partent du graphe de la matrice pour tenter de trouver un séparateur (un ensemble de colonnes (de noeuds) dont la suppression décompose le graphe en deux ou plusieurs parties indéndantes). Le procédé se répéte recursivement sur les parties indépendantes jusqu'à l'obtention d'un arbre. Exemple de méthode globale : Nested Dissection.
2. Les méthodes locales : (plus connues sous le nom de *minimum degree ordering*) se contentent de choisir les sommets de plus petits degrés en premier dans le processus d'élimination. Les méthodes locales sont les méthodes les plus répandues et il en existe plusieurs variantes telles que AMD [8] (approximate minimum degre), AMF (approximate minimum fill) ou encore MMD (multiple minimum degre).

3. Les méthodes hybrides : sont caractérisées par le fait qu'elles font appel à la fois aux méthodes locales et globales pour générer la séquence des permutations. En effet, les logiciels qui implémentent les méthodes hybrides tels que METIS ou SCOTCH utilisent des méthodes globales pour générer des sous-arbres indépendants et passent suivant certains critères à des méthodes locales pour les sous-graphes.

### 3.2.1 Arbre d'élimination

L'arbre d'élimination joue un rôle important dans plusieurs aspects de la factorisation des matrices creuses. Il a été introduit par J. Liu [7]. L'arbre d'élimination fournit des informations structurelles relatives au processus de factorisation.

Soit A une matrice symétrique d'ordre N. Le graphe associé à la matrice A est un graphe G(A) = ( X(A),E(A)), où les sommets dans X(A) correspondent aux lignes et colonnes de A et les arêtes de E(A) correspondant aux élément non-nuls dans la ligne (resp. colonne). De plus, la matrice F=L+U reprśente donne la structure de la matrice aprés remplissage. Soit $F_t$ la matrice obtenue en supprimant pour chaque colonne (resp. ligne) tout les éléments non-nuls hors-diagonale sauf le premier sous la diagonale. Le graphe associé à la matrice $F_t$ a une structure d'arbre. Ce dernier est appelé arbre d'élimination. Il représente une réduction transitive du graphe associé à la matrice F. De plus, l'arbre d'élimination est la plus petite structure exprimant les dépendances dans le processus de factorisation. Ainsi, le pivot correspondant à un noeud de l'arbre ne peut être éliminé que si tout ses fils dans l'arbre d'élimination l'ont déja été. Un exemple illustrant la structure des matrices A, F et $F_t$ est donné dans la figure 3.4. Les graphes correspondant à ces matrices sont donnés dans la figure 3.5.

La notion d'arbre d'élimination concernait au début les matrices creuses symétriques. Puis elle a été etendue aux matrices non-symétriques. Ceci a été fait en considérant pour toute la phase de construction de l'arbre la matrice $A+A^T$ (qui est symétrique) au lieu de la matrice A. Ainsi, on se raméne au cas symétrique pour générer l'arbre d'élimination.

### 3.2.2 Résultats sur les matrices du CETMEF

Nous avons analysé le résultat de l'ordering sur les Matrices du CETMEF :

- matrice rot1 d'ordre 3836
- matrice boulogne d'ordre 336572
- matrice antifer d'ordre 481470

On obtient les informations suivantes sur leur arbre d'élimination respectif, c'est à partir de cet arbre que l'on peut déterminer le degré de parallélisme.

- matrice rot1 :
  Nombre de noeuds : 1063
  Nombre de feuilles : 650
  Taille maxmium de Front : 82

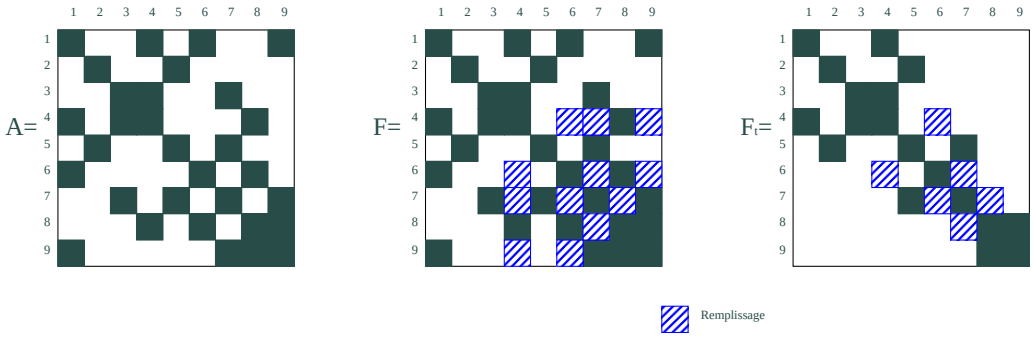



FIG. 3.4 – *Un exemple de structures de matrice*

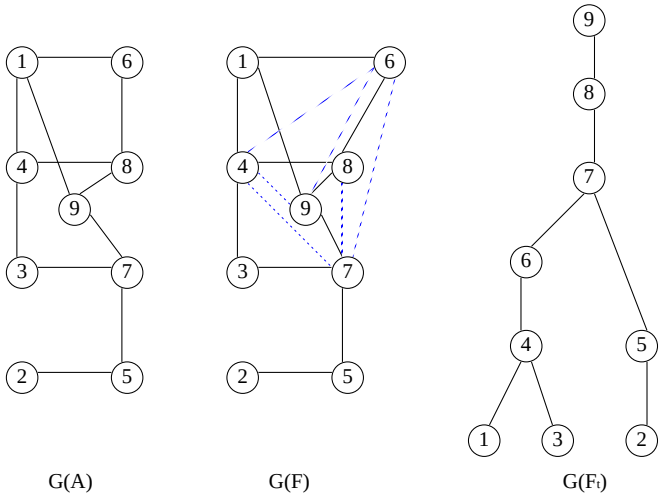



FIG. 3.5 – *Structure des graphes associés aux matrices de la figure3.4*

Profondeur maximum : 12
Variance de la profondeur : 1.47

– matrice boulogne :
Nombre de noeuds : 92137
Taille maximum de Front : 911
Nombre de feuilles : 57189
Profondeur maximum : 19
Variance de la profondeur : 1.530

– matrice antifer :
Nombre de noeuds : 130892
Taille maximum de Front : 972
Nombre de feuilles : 81198
Profondeur maximum : 19
Variance de la profondeur : 1.588

Ces résultats prouvent que les arbres d'élimination des matrices du CETMEF ne sont pas profonds (profondeur maximum de 12 à 19), mais très larges (de 650 à 81198

feuilles); donc Refonde n'est pas intrinséquement séquentiel, mais bien au contraire, Refonde est intrinséquement parallèle : on peut donc utiliser un solveur parallèle.

# Chapitre 4

# Solveur multifrontal parallèle pour Refonde

La méthode la plus connue pour paralléliser les solveurs creux et basée sur la méthode multifrontale . Dans ce chapitre, nous présentons cette méthode, ainsi qu'une bibliothèque de calculs qui l'implémente : MUMPS. Nous l'avons intégré à Refonde, et présentons les résultats ainsi obtenus.

## 4.1 Méthode Multifrontale

La méthode multifrontale [5] est une méthode directe dédiée à la factorisation des systèmes linéaires creux. Elle consiste à réorganiser les calculs de telle sorte que la factorisation totale de la matrice initiale est effectuée à l'aide de factorisations partielles de petites matrices denses ( en relation avec la matrice initiale) appellées *matrices frontales*. La méthode repose sur le concept d'arbre d'élimination introduit dans la partie précédente. La méthode multifrontale peut être alors formulée de la maniére suivante:

Soit A une matrice creuse symétrique d'ordre N. Dans le but de factoriser A en LU, on associe à chaque élément de la diagonale une matrice frontale. La taille de cette dernière est donnée par le nombre d'éléments non nuls dans la colonne qui correspond au pivot dans le facteur L. La factorisation est exprimée algorithmiquement comme suit :

**for** i=1 to N **do**
  former et assembler la matrice frontale associée à i
  effectuer une étape d'élimination dans la matrice frontale pour obtenir $U_{i*}$ et $L_{*i}$
  envoyer le bloc de contribution au père dans l'arbre d'élimination
**end for**

A chaque étape i, la matrice frontale est assemblée à partir de la $i^{eme}$ ligne (ou colonne) de la matrice initiale et de certaines matrices frontales formées dans des étapes précédentes (en l'occurrence les blocks de contributions des fils du noeud courant). La notion de bloc de contribution est illustrée dans la figure 4.1. Une fois la matrice

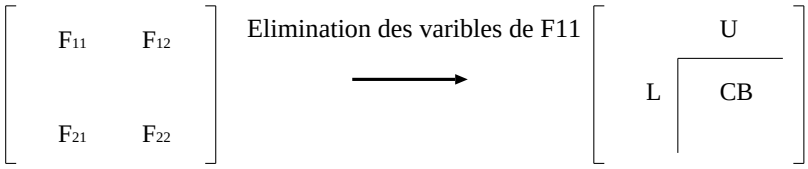



FIG. 4.1 – *composition d'une matrice frontale*

frontale assemblée, la première ligne/colonne de cette dernière est dite complétement formée c'est-à-dire qu'elle a fini de recevoir les contributions des matrices frontales dont elle dépend. Elle peut alors être éliminée immédiatement pour donner la $i^{eme}$ ligne (resp. colonne) du facteur U (resp. L). Le processus d'élimination est déterminé par l' arbre d'élimination. Il est important de signaler qu'en général dans la méthode multifrontale, le parcours de l'arbre se fait en profondeur d'abord ( c'est-à-dire en favorisant la remontée dans l'arbre) pour des raisons de minimisation de la taille mémoire occupée par les blocs de contributions.

La figure 4.2 présente un exemple de matrice avec l'arbre d'élimination associé ainsi que les différentes matrices frontales.

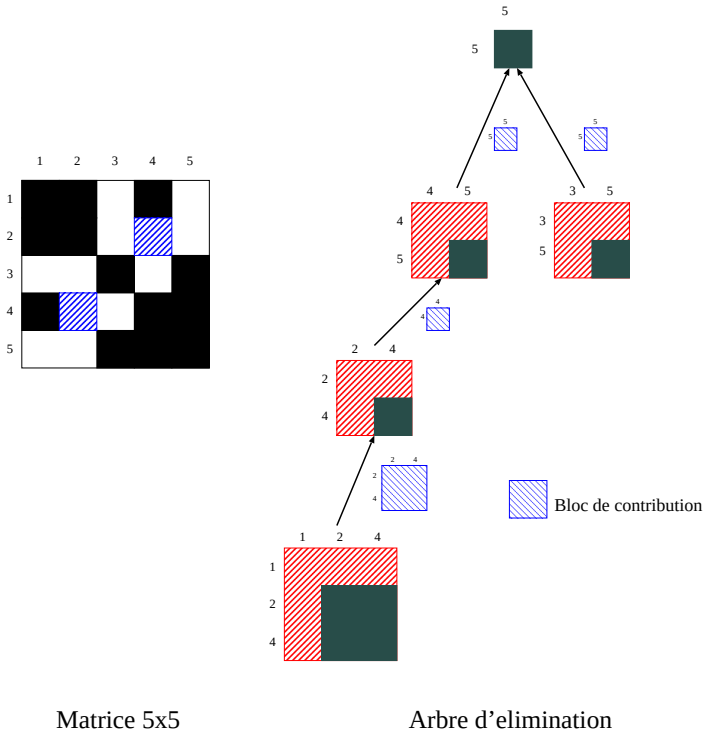
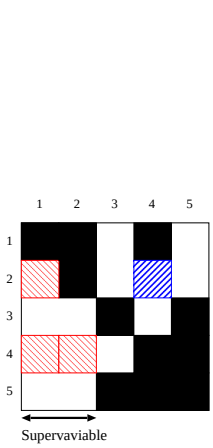


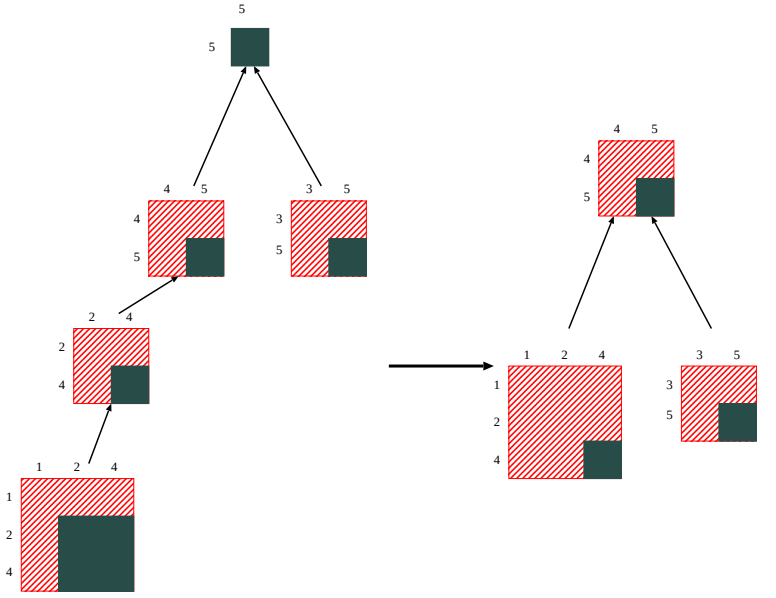

FIG. 4.2 – *Exemple de matrice 5x5 avec arbre d'élimination*

– Arbre d'assemblage et supervariables : Dans le cas général, les solveurs multifrontaux n'utilisent pas l'arbre d'élimination tel que défini précédemment. Pour ce faire, ils partent de l'arbre d'élimination pour générer un arbre d'assemblage

qui n'est qu'une forme compressée de l'arbre d'élimination. En effet, l'arbre d'assemblage est l'arbre d'élimination où on a fusionné un pére avec un ou plusieurs de ces fils. Le critère de choix des noeuds a fusionner est le suivant : Les noeuds à fusionner sont les noeuds pour lesquels le (ou les fils) sont inclus dans le père ou inversement. Les noeuds ainsi fusionnés font partis de la même supervariable (ou supernoeud). Pour rappel, un supernoeud est un ensemble contigu de colonnes tel que la partie correspondante à la supervariable dans le facteur L a un bloc triangulaire inférieur sur la diagonale et la même structure (pour les éléments non-nuls) sous la diagonale. Par exemple, si on considére le cas de l'arbre délimination donné en figure 4.3, les noeuds correpondants aux variables 1 et 2 ont été fusionnés en un seul noeud qui représente la super variable (1,2).




(a) illustration de la notion de supernoeud sur la matrice donnée en figure 4.2




(b) Arbres d'élimination et d'assemblage de la matrice

FIG. 4.3 – *exemple d'arbre d'assemblage et de supervariable*

## 4.2 Solveur Multifrontal MUMPS

MUMPS (**MU**ltifrontal **M**assively **P**arallel **S**olver) [2] est un solveur parallèle de systèmes d'equations linéaires du type A.x = b, ou A est une matrice creuse, elle peut être symetrique,non symetrique, definie positive. MUMPS utilise une version parallèle de la méthode multifrontale pour la factorisation matricielle. MUMPS requiert l'environnement MPI et utilise les routines de BLAS, BLACS, et ScaLAPACK.

La résolution d'un système linéaire avec MUMPS est divisée en trois grandes phases qui sont : l'analyse , la factorisation et la résolution. L'analyse est éxecutée de maniére

sequentielle dans MUMPS (c'est à dire sur un seul processeur : le processeur maitre). Elle se déroule en trois phases :

- Une phase de génération de l'abre d'élimination (appelée aussi factorisation symbolique) dans laquelle MUMPS fait appel à des algorithmes de renumérotation visant à réduire le remplissage lors de la factorisation et à générer l'arbre d'élimination. Dans cette phase, MUMPS part de la structure de $A+A^T$ (A ayant pu subir des permutations non symétrique).
- Une phase de traitement de l'arbre, qui donne une distribution partielle des tâches sur les processeurs.
- Une partie simulation de la factorisation dans la laquelle le programme essaie d'estimer l'espace mémoire et le nombre d'opérations nécessaires à l'éxecution (sans prendre en compte le pivotage numérique qui pourrait avoir lieu lors de la factorisation).

Remarque : Le processeur maitre peut aussi contribuer aux calculs durant la phase de factorisation et solution tout comme un processor esclave. Cette option a été concue pour pouvoir executer MUMPS sur un seul processeur (donc en séquentiel). Dans le cas ou la mémoire est limitée, l'activation de cette option entraine une degradation des performances, car la mémoire du maitre contient la matrice initiale, et le faire participer aux calculs risque d'engendrer une dégradation des performances due à la saturation de sa memoire.

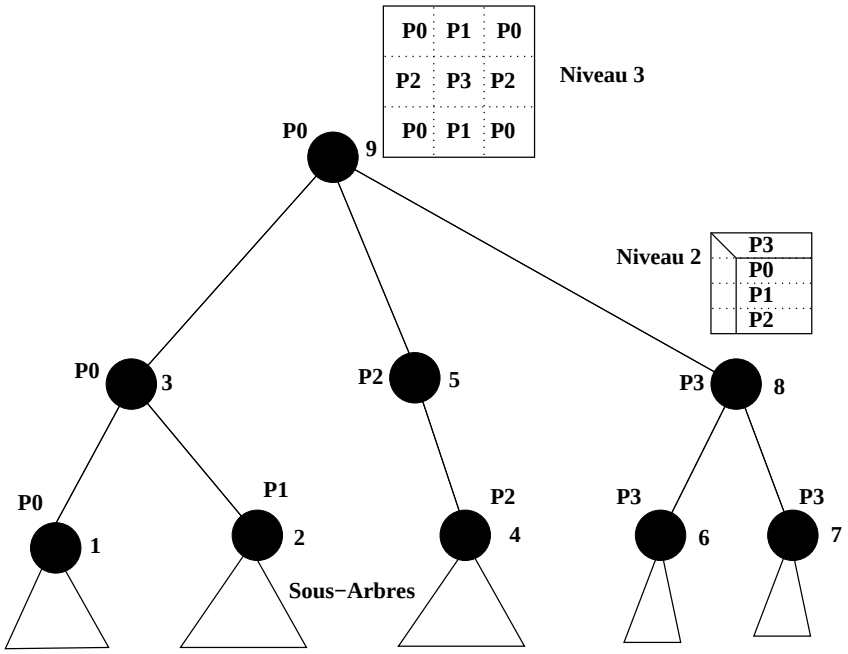



FIG. 4.4 – *Distribution des calculs pour un arbre d'élimination*

MUMPS exploite à la fois le parallèlisme de l'arbre et des noeuds (quand leurs tailles le permet). Pour celà il distingue trois types de noeuds au niveau de l'arbre d'élimination. Les noeuds traités par un seul processeur sont dits de type 1. Les noeuds

de l'abre pour lesquels la matrice frontale associée a un gros bloc de contribution sont dits de types 2. MUMPS fait un découpage 1-D sur ces matrices. Les noeuds sont alors traités en parallèle en distribuant les blocs (issus du découpage 1-D) sur plusieurs processeurs. Enfin, si la matrice frontale associée à la racine de l'arbre d'élimination est assez grande, il est possible de faire un partitionnement 2-D de cette dernière. Dans ce cas, le noeud est dit de type 3. Il est important de noter que pour les noeuds traités en parallèle il existe une notion de processeur maitre et de processeurs esclaves qui est définie comme suit:

- Pour un noeud de type 2, le processeur maitre correspond au processeur sur lequel a lieu l'élimination des pivots. Dans ce cas, les processeurs esclaves ne sont chargés que de mettre a jour les parties qui leurs sont attribuées. Par exemple pour la figure 4.4 le processeur maitre du noeud 3 est le processeur P0.
- Pour le noeud de type 3, le processeur maitre est choisi au niveau de l'analyse parmis tout les processeurs.

Il est à noter que le traitement des noeuds de type 3 est fait à l'aide de SCALAPACK. En effet, une fois la découpage 2-D de la matrice fontale associée au noeud de type 3 effectué, MUMPS fait appel à SCALAPACK pour effectuer la factorisation parallèle.

MUMPS utilise pour la gestion du parallèlisme un ordonnancement complètement dynamique. En effet, seuls les processeurs maitre de chaque noeud de l'arbre sont designés statiquement au niveau de l'analyse (le processeur maitre pour les noeuds de type 2 et 3 et le processeur qui traite le noeud pour les noeuds de type 1). Pour la désignation des processeurs esclaves, elle se fait dynamiquement au moment de l'exécution en tentant d' équilibrer au mieux la charge des processeurs (typiquement ne pas prendre comme esclave un processeur qui a une grosse charge de travail).

## 4.3 Choix de l'ordering

Le solveur MUMPS est couplé à un ordering (par défaut AMD) or il existe plusieurs orderings. Par conséquence des tests pour choisir l'ordering offrant les meilleures performances pour résoudre les matrices du CETMEF par le solveur MUMPS ont été réalisés.

### 4.3.1 Résultats

Les temps indiqués dans ces tests ne representent pas le temps global d'execution du programme, mais mesurent la performance des orderings, les graphiques cidessous on été réalisés à partir des matrices du CETMEF suivantes :

- la matrice boulogne d'ordre 336572, voir figure 4.5
- la matrice antifer d'ordre 481470, voir figure 4.6

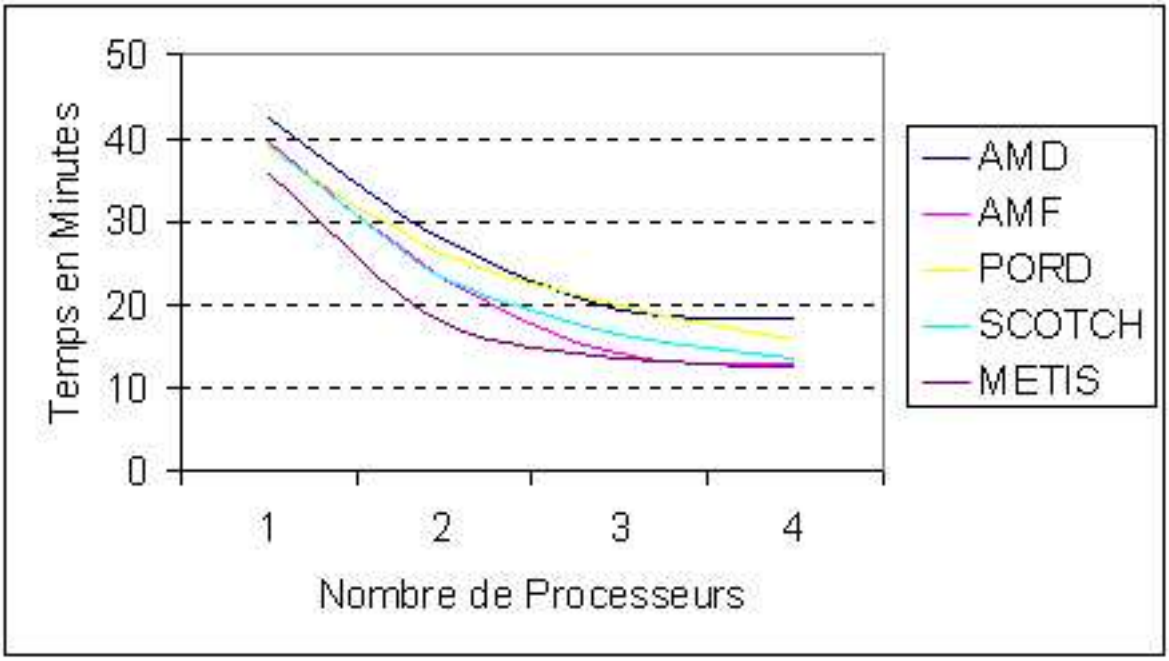



FIG. 4.5 – *Tests sur la matrice boulogne*

### 4.3.2 Conclusion

Grâce à ces tests, nous avons pu établir le classement général des performances des orderings :

1. METIS
2. AMF
3. PORD
4. SCOTCH
5. AMD

On remarque que l'ordering METIS de la famille méthode hybride est le meilleur, suivi de près de l'ordering AMF, ce qui est tout à fait logique puisque METIS est basé sur AMF.Finalement ce sera donc METIS qui sera couplé à MUMPS pour résoudre les matrices du CETMEF.

## 4.4 Performance de MUMPS

Après avoir couplé l'ordering METIS au solveur parallèle MUMPS, des tests pour mesurer le temps global d'exécution ont été effectués sur un et plusieurs processeurs.

### 4.4.1 Résultats

On distinguera les deux séries de tests réalisées comme suit :

– la première série, le maitre participe aux calculs voir figure 4.7, avec P proces-

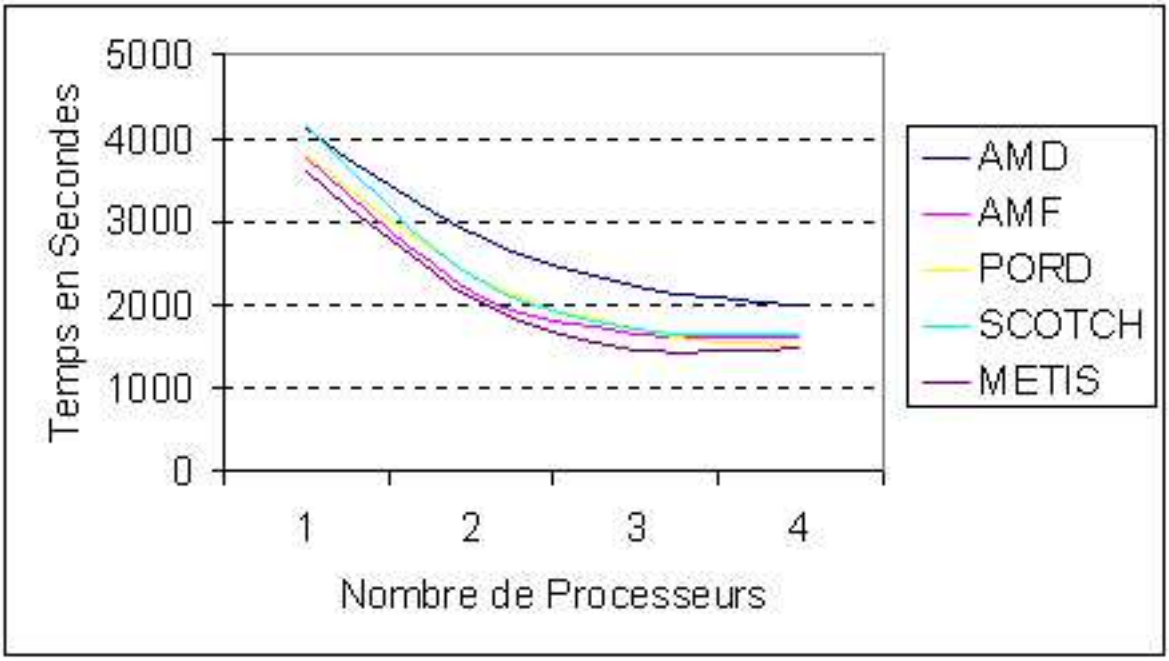



FIG. 4.6 – *Tests sur la matrice antifer*

seurs, on aura le maitre qui envoie des caculs à P-1 processeurs, puis le maitre traite aussi des calculs (comme un esclave), puis il recoit les résultats des P-1 processeurs.Le processus maitre nécessite beaucoup de mémoire, le processus esclave aussi. Si un processeur cumule les deux, alors il swappe.

– la seconde série, le maitre ne participe pas aux calculs : figure 4.8, avec P processeurs, on aura le maitre qui envoie des caculs à P-1 processeurs, puis le maitre attends passivement les résultats des P-1 processeurs.

### 4.4.2 Conclusion

On remarque bien la baisse de performance lors de l'activation du maitre pour les calculs, exemple à 6 processeurs : on obtient le même temps alors que dans la série où le maitre est désactivé, on a 5 processeurs qui travaillent réellement au lieu de 6! Donc il faut veiller à désactiver l'option qui inclut le maitre dans les calculs. On notera d'après les observations que le solveur parallèle est scalable (plus on met de processeurs,plus le temps global va diminuer).

Comparaison des performances globales avec le CETMEF :
D'après les informations que nous avons eu :

– Test sur la matrice Boulogne :
Le CETMEF avec leur dernière version sur un Pentium 3 - 450 Mhz disposant de 384 Mo de mémoire vive a un temps d'execution total de 40 minutes.

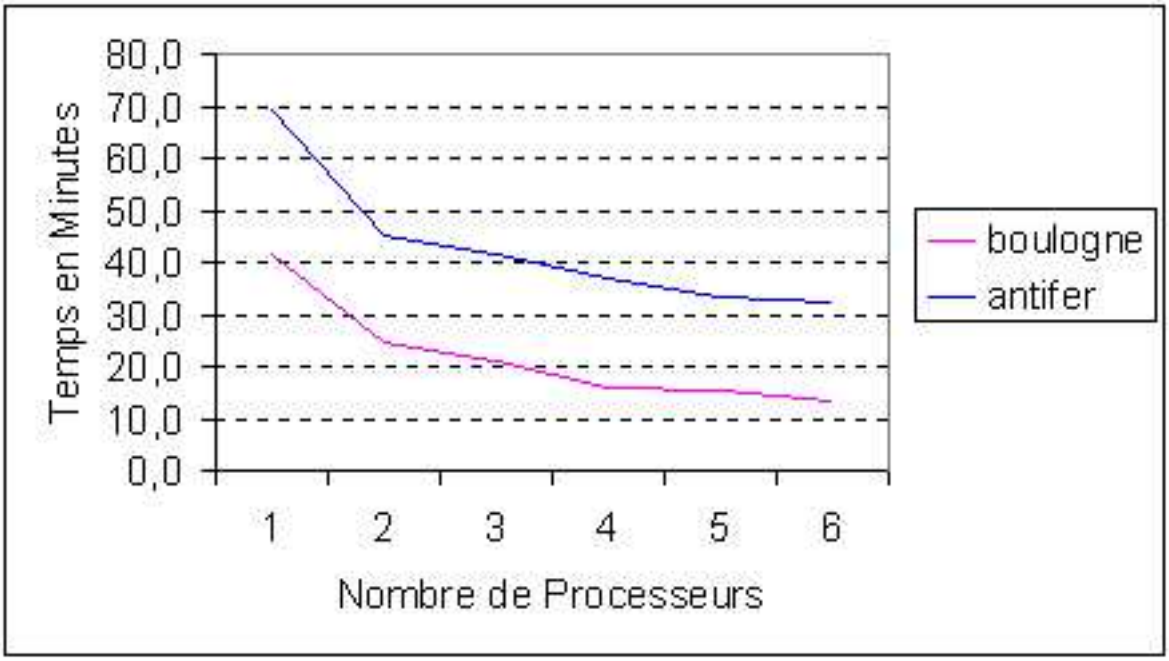



FIG. 4.7 – *Série 1 : le maitre participe aux calculs*

Au LaRIA avec notre version (séquentielle) sur un Alpha 600 Mhz disposant de 128 Mo de mémoire vive a un temps d'execution total de 36 minutes.

– Test sur la matrice Antifer :
  Le CETMEF avec leur dernière version sur un Pentium 3 - 700 Mhz disposant de 1 Go de mémoire vive a un temps d'execution total de 55 minutes.
  Au LaRIA avec notre version (séquentielle) sur un Alpha 600 Mhz disposant de 128 Mo de mémoire vive a un temps d'execution total de 53 minutes.

Remarque :Dans le programme Refonde, la lecture du fichier d'entrée et l'écriture des fichiers solutions se font en séquentiel par le maitre, cela implique un temps total minimum incompressible variable en fonction de la taille du fichier d'entrée lors de l'execution en parallèle sur x processeurs, on peut donc en conclure que le nombre de processeurs sera borné pour chaque matrice.

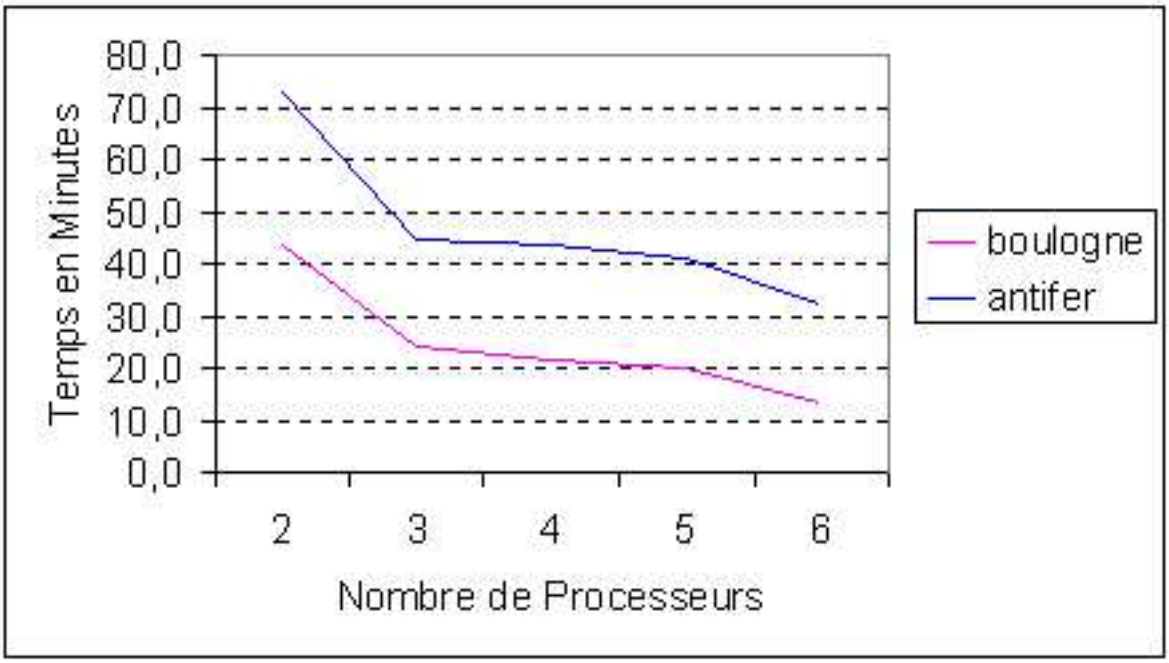



FIG. 4.8 – *Série 2 : le maitre ne participe pas aux calculs*

# Conclusion

Le stage s'est effectué dans le cadre d'une collaboration avec le CETMEF : il est confronté à la résolution de gros problèmes (simulation de la houle en bassin). Le CETMEF a développé la chaîne complète du logiciel de simulation en y intégrant un solveur séquentiel basique. Cette application ramène le problème de simulation à la résolution d'un système linéaire creux (Ax=B).

Lors d'une étude précédente [1],il a été constaté que la représentation des matrices du système limitait la taille des problèmes à résoudre à cause d'un gaspillage mémoire. Entre temps, une nouvelle version a été développée, intégrant un nouveau format de stockage des matrices. Durant ce stage nous avons étudié les gains de tels stockages, et nous avons intégré le format le plus adapté à notre version de Refonde. Malheureusement certains problèmes sont toujours trop grands pour que ce gain substantiel suffise à leur résolution. Ainsi, nous avons étudié comment accroître encore les capacités du solveur en limitant le remplissage, ce qui a été rendu possible en utilisant les techniques de reordering. Nous avons donc étudié les différents algorithmes de la littérature et déterminé expérimentalement lequel était le mieux adapté.

Lors de cette étude, nous avons aussi constaté que la résolution du système pouvait être parallélisé, contrairement à ce que la forme des matrices (matrices bandes) laissait supposer.Nous avons donc recherché les meilleures techniques pour résoudre de manière parallèle les systèmes creux. Nous avons finalement trouvé le solveur MUMPS, que nous avons couplé avec Refonde, de plus grâce au parallélisme nous pouvons désormais traiter les problèmes plus gros. De même grâce à la scabilité, le temps de résolution des matrices du Cetmef peut devenir dérisoire lors de l'utilisation de nombreux processeurs.

Finalement la dernière perspective serait d'étudier une méthode " *out of core* " en vue de l'intégrer dans MUMPS, pour ne plus avoir de limitation dans la taille des matrices.

# Bibliographie


[1] Laurent Josse, Etude préliminaire de la parallélisation d'un solveur pour la simulation de la houle

[2] Patrick. Amestoy, Iain S. Duff, J.Y l'Excellent, J. Koster, M. Turna, MUltifrontal Massively Parallel Solver (MUMPS version 4.2) Specification Sheets

[3] Grégoire Richard, Coupling MUMPS and ordering software

[4] Patrick. Amestoy, Iain S. Duff, J.Y. l'Excellent, Distributed memory multifrontal solver, PARASOL workshop, Sept. 97, CERFACS

[5] Patrick. Amestoy, Iain S. Duff,Memory management issues in sparse multifrontal methods on multiprocessors

[6] Patrick. Amestoy, Iain S. Duff,J. Koster,A fully asynchronous multifrontal solver using distributed dynamic scheduling

[7] J.W.H. Liu,The role of elimination trees in sparse factorization

[8] Patrick. Amestoy, Thimothy A. Davis, Iain S. Duff, An approximate minimum degree ordering algorithm, SIAM J. Matrix Analysis and Applications, Vol. 17 (1996), pp 886-905

[9] Jean Guillaume DUMAS, Thèse : Algorithmes parallèles efficaces pour le calcul formel : Algebre linéaire creuse et extensions algébriques

[10] Iain S. Duff User'Guide for the Harwell-Boeing Sparse Matrix Collection (Release I)

[11] Stéphane DOMAS, Thèse : Contribution à l'écriture et à l'extension d'une Bibliothèque d'Algèbre Linéaire Parallèle

[12] Eddy Caron, Thèse : Calcul numérique de grande taille

[13] Gouri Dhatt, Gilbert Touzot, Une présentation de la méthode des éléments finis

# Annexes

# Annexe A

# Logiciel Refonde

## A.1 Méthode des éléments finis

La méthode des éléments finis est une méthode de résolution approchée d'équations aux dérivées partielles. Il s'agit de remplacer un problème compliqué pour lequel à priori on ne connaît pas de solution, par un problème plus simple que l'on sait résoudre.

## A.2 Compilation de Refonde

Voici une représentation schématique du Makefile associé à la compilation de Refonde.

refonde1.f
(param.inc)
(common.inc)

|

elem14.f elem24.f
(param.inc) (param.inc)
(fonction.f) (fonction1.f)

*Légende :* (nom_fichier) correspond à un include 'nom_fichier'

## A.3 Problème d'exécution

Pour que Refonde puisse fonctionner sans obtenir le message d'erreur : "SEGMENTATION FAULT", qui est du au fait que la taille des données statiques requise par le

programme est supérieure à la taille de mémoire restante (swap+memoire), deux solutions sont possibles :

1. Augmenter la taille de swap.
2. Dans le cas de Refonde,pour des matrices tests de petites tailles, comme la majeure partie des données allouées en statique ne sera jamais utilisée (ni même initialisée), cela ne requerra donc pas d'espace mémoire ou de swap. Il suffit donc de demander à Linux de ne pas faire le test pour savoir s'il reste assez de mémoire (*dans le cas où les données alloué en statique ne sont pas toute utilisées*). Pour réaliser ce blocage, il faut exécuter en tant que Root la ligne de commande suivante :

   ```
   "cat 1 > /proc/sys/vm/overcommit_memory"
   ```

## A.4 Représentation de l'exécution

Ces arbres ci-dessous correspondent au déroulement de l'exécution séquentielle de Refonde en utilisant les différentes "macro-routines". Donc ils sont simplifiés, du fait du nombre elevé de routines employées.

### A.4.1 Arbre principal

```
Main
|
Debut
|
|--------- Bkdata
|--------- Blimag
|--------- Blcount
|--------- Blspec
|--------- Bldata (voir sous-arbre A)
|--------- Blonde
|            |--------- Exonde
|
|--------- Blsor
|            |--------- Exsolr
|
|--------- Bllinr
|            |--------- Tkld_neq
|            |--------- Exlinr (voir sous-arbre B)
|
|--------- Bllint
|            |--------- Tkld_neq
|            |--------- Exlinr (voir sous-arbre B)
|
Fin
```

### A.4.2 Sous-Arbre A

```
Bldata
|
Debut
|
|--------- Rdvcor
|--------- Initi
|--------- Rdkdlnc
|--------- Rdprn
|--------- Rdknec
|--------- Initi
|--------- Rdtypl
|--------- Rdprel
|--------- Rdclim
|--------- Rdsolc
|
Fin
```

### A.4.3 Sous-Arbre B

```
Exlinr
|
Debut
|
|--------- Exonde
|--------- Initc
|--------- Askg (voir sous-arbre C)
|--------- Modkg
|--------- Appel_Solveur
|--------- Move
|--------- Grad
|--------- Deferl
|--------- Prtsol
|
Fin
```

### A.4.4 Sous-Arbre C

```
Askg
|
Debut
|
|--------- Initc
|--------- Elmlib
|            |--------- elem01
```

```
|            |--------- elem02
|            |---------  ...
|            |--------- elem24
|
|--------- Tkone
|--------- Tvcore
|--------- Tvcorec
|--------- Tvpree
|--------- Tloc_neq
|--------- Elmlib
|            |--------- elem01
|            |--------- elem02
|            |---------  ...
|            |--------- elem24
|
|--------- Askec
|--------- Asfec
|
Fin
```

## A.5 Problème d'intégration du solveur MUMPS

Après l'intégration du solveur MUMPS dans le source de la dernière version de Refonde, lors de l'exécution à partir du source ainsi obtenu, le programme génère une erreur due à un problème mémoire :

```
"forrtl: severe (174): SIGSEGV, segmentation fault occurred''
```

Cette erreur est causée par la compilation de MUMPS + Refonde, car MUMPS propose SCOTCH en option, et lors de la compilation de SCOTCH, la librairie mathématique standard ( -lm ) est utilisée, et **le source de Refonde ne doit pas être compilé avec la librairie mathématique standard**, sous peine d'obtenir le message d'erreur indiqué précédemment, du à une incompatibilité avec la librairie mathématique standard causée par des interactions de noms de variables.

Ce problème est réglé sur les nouvelles versions de Refonde créees au LaRIA, grâce à une utilisation de noms de variables différentes.

# Annexe B

# Formats de compression pour matrices creuses

## B.1 Introduction

La résolution de très grands systèmes linéaires intervient dans de nombreuses applications scientifiques et industrielles. De ce fait, les matrices correspondant à ces systèmes linéaires nécessite des tailles de stockage très importantes. Généralement ces matrices sont creuses, voir fortement creuses, d'où l'obligation d'avoir recours à la compression pour pouvoir calculer / stocker des systèmes linéaires plus grand et échanger ces matrices via Internet avec des temps de téléchargements raisonnables.

Dans ce chapitre, les trois formats de compression les plus usuels pour l'utilisation / échange / résolution de matrices creuses sont présentés dans l'ordre chronologique de leur création. Le premier format créé et étudié sera le format COO, puis le format CSR/CSC, puis le format SKYLINE.

Pour étudier et permettre une comparaison entre les formats, on calculera le gain de compression qui sera obtenu par la différence entre le nombre d'éléments avant et après compression par rapport au nombre d'éléments total de la matrice.

*On prendra pour référence la matrice creuse A suivante :*

| | | | | |
|---|---|---|---|---|
| **1** | **0** | **3** | **0** | **0** |
| **0** | **0** | **7** | **-2** | **0** |
| **4** | **9** | **0** | **0** | **0** |
| **0** | **10** | **0** | **0** | **8** |
| **0** | **0** | **0** | **-9** | **0** |

*Notations :*

- x : nombre d'éléments non nuls de la matrice creuse.

– n : nombre total d'éléments de la matrice creuse, correspond pour une matrice de dimension (M,N) à n=M x N.
– nc : nombre d'éléments obtenus après la compression, correspond à la taille mémoire.
– Gc : le gain de compression, correspond à Gc = (n-nc)/n

## B.2 Format COO

### B.2.1 Présentation

Le format COO (Coordinate) est le format de compression le plus simple à manipuler et à identifier. C'est le premier format de compression pour matrices creuses qui a été crée. De nos jours ce format est devenu obsolète et tend à disparaître.

### B.2.2 Définition

A partir de la matrice creuse, 3 tableaux de taille identique x sont créés. On stocke dans le premier tableau les valeurs des éléments non nuls en lisant la matrice d'origine ligne par ligne, et dans les 2 tableaux restants leurs coordonnées.

– A_COO contient les valeurs non nulles de A.
– LA_COO contient les abscisses.
– CA_COO contient les ordonnées.

*Stockage Avant Compression :*

| | | | | |
|---|---|---|---|---|
| **1** | **0** | **3** | **0** | **0** |
| **0** | **0** | **7** | **-2** | **0** |
| **4** | **9** | **0** | **0** | **0** |
| **0** | **10** | **0** | **0** | **8** |
| **0** | **0** | **0** | **-9** | **0** |

*Stockage Après Compression :*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | **3** | **7** | **-2** | **4** | **9** | **10** | **8** | **-9** | **A_COO** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | **1** | **2** | **2** | **3** | **3** | **4** | **4** | **5** | **LA_COO** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | **3** | **3** | **4** | **1** | **2** | **2** | **5** | **4** | **CA_COO** |

### B.2.3 Exemple

La valeur 4 stockée en (3,1) est stockée de la façon suivante :
En parcourant la matrice origine ligne par ligne, on trouve les valeurs non nulles :

1,3,7,-2, **4**, 9, 10, 8, -9

Donc :

A_COO(5) = 4

LA_COO(5) = 3

LA_COO(5) = 1

### B.2.4 Gain de compression

**Nc (GOO) = x + x + x = 3x**
**Gc (COO) = (n-3x)/n**

*Rappel des Notations :*
- x : nombre d'éléments non nuls de la matrice creuse.
- n : nombre total d'éléments de la matrice creuse, correspond pour une matrice de dimension (M,N) à n=M x N.
- nc : nombre d'éléments obtenus après la compression, correspond à la taille mémoire.
- Gc : le gain de compression, correspond à Gc = (n-nc)/n

## B.3 Format CSR et Format CSC

### B.3.1 Présentation

Le format CSR (Compressed Sparse Row) et le format CSC (Compressed Sparse Column) sont les améliorations directes du format de compression COO. Le nom commun du format CSR est le Format MORSE. Le Format CSC lui est nommé Format HARWELL-BOEING. Ces deux formats sont efficaces pour tout les types de matrices creuses existantes, c'est pour cette raison que ces deux formats sont les plus couramment utilisés, en particulier pour l'échange de matrices sur Internet.

*Exemple :* le site web " Matrix Market Ressources "
http://math.nist.gov/MatrixMarket/resources.html.

### B.3.2 Définition

Pour le format CSR la matrice des lignes fournit l'indice correspondant aux premiers éléments non-nuls de chaque ligne de la matrice de départ. Tandis que pour le format CSC la matrice des colonnes fournit l'indice correspondant aux premiers éléments non-nuls de chaque colonne de la matrice de départ.Le format CSC est *le format CRS pour la transposée de A*.

### B.3.3 Conversion CSR ⇒ CSC (Morse ⇒ Harwell-Boeing)

Soit la matrice A stocké au format CSR, effectuer les opérations suivantes :

1. A_CSC ← A_CSR
2. LA_CSC ← CA_CSR
3. CA_CSC ← LA_CSR

ATTENTION : On obtient la transposée de A au format CSC.

### B.3.4 Format CSR

- A_CSR contient les valeurs non nulles de A.
- LA_CSR contient l'indice de début de ligne dans A_CSR.
- CA_CSR contient l'indice de la colonne.

*Stockage Avant Compression :*

| | | | | |
|---|---|---|---|---|
| **1** | **0** | **3** | **0** | **0** |
| **0** | **0** | **7** | **-2** | **0** |
| **4** | **9** | **0** | **0** | **0** |
| **0** | **10** | **0** | **0** | **8** |
| **0** | **0** | **0** | **-9** | **0** |

*Stockage Après Compression :*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | **3** | **7** | **-2** | **4** | **9** | **10** | **8** | **-9** | **A_CSR** |

| | | | | | |
|---|---|---|---|---|---|
| **1** | **3** | **5** | **7** | **9** | **LA_CSR** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | **3** | **3** | **4** | **1** | **2** | **2** | **5** | **4** | **CA_CSR** |

### B.3.5 Exemple

La valeur 4 est stockée de la façon suivante :
En parcourant la matrice origine ligne par ligne,on trouve les valeurs non nulles suivantes :

1,3,7,-2, **4**, 9, 10, 8, -9

Donc :

A_CSR(5) = 4

LA_CSR(3) = 5

CA_CSR(5) = 1

LA_CSR(3) = 5 , car la 3$^{ème}$ ligne commence par la valeur 4 qui est à l'indice 5 du tableau A_CSR.

### B.3.6 Format CSC

- A_CSC contient les valeurs non nulles de A.
- LA_CSC contient l'indice de ligne.
- CA_CSC contient l'indice de début de colonne dans A_CSC.

*Stockage Avant Compression :*

| | | | | |
|---|---|---|---|---|
| **1** | **0** | **3** | **0** | **0** |
| **0** | **0** | **7** | **-2** | **0** |
| **4** | **9** | **0** | **0** | **0** |
| **0** | **10** | **0** | **0** | **8** |
| **0** | **0** | **0** | **-9** | **0** |

*Stockage Après Compression :*

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | **4** | **9** | **10** | **3** | **7** | **-2** | **-9** | **8** | **A_CSC** |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **1** | **3** | **3** | **4** | **1** | **2** | **2** | **5** | **4** | **LA_CSC** |

| | | | | | |
|---|---|---|---|---|---|
| **1** | **3** | **5** | **7** | **9** | **CA_CSC** |

### B.3.7 Exemple

La valeur 4 est stockée de la façon suivante :
En parcourant la matrice origine colonne par colonne, on trouve les valeurs non nulles suivantes :

1, **4**, 9, 10, 3, 7, -2, -9, 8

Donc :

A_CSC(2) = 4

LA_CSC(2) = 3

CA_CSC(1) = 1

CA_CSC (1) = 1 , car la 1$^{ère}$ colonne commence à la valeur 1 qui est à l'indice 1 du tableau A_CSC.

### B.3.8 Gain de compression

Soit L le nombre de lignes de la matrice creuse et C le nombre de colonnes de la matrice creuse :

*Pour le format CSR :*

**Nc (CSR) = x + x + L = 2x + L**
**Gc (CSR) = (n-(2x+L))/n**

*Pour le format CSC :*

**Nc (CSC) = x + x + C = 2x + C**
**Gc (CSC) = (n-(2x+C))/n**

Afin de permettre une comparaison avec les autres formats : on suppose que la matrice creuse est carrée. Cette condition fait office de moyenne pour les deux formats CSR et CSC, car dans le cas où le nombres de colonnes est très important par rapport au nombre de lignes : le format CSC sera préconisé, et inversement pour le format CSR. On obtient donc pour une matrice creuse carrée L=C=$\sqrt{n}$, le gain de compression suivant :

**Nc (CSR) = 2x + $\sqrt{n}$**
**Gc (CSR) = (n-(2x+$\sqrt{n}$))/n**

*Rappel des Notations :*

- x : nombre d'éléments non nuls de la matrice creuse.
- n : nombre total d'éléments de la matrice creuse, correspond pour une matrice de dimension (M,N) à n=M x N.
- nc : nombre d'éléments obtenus après la compression, correspond à la taille mémoire.
- Gc : le gain de compression, correspond à Gc = (n-nc)/n

## B.4 Format SKYLINE

### B.4.1 Présentation

Le format SKYLINE, ou format " ligne de ciel " en français, a pour idée de supprimer les ensembles d'éléments nuls à gauche et à droite respectivement du premier et du dernier élément significatif de chaque ligne de la matrice de départ. C'est la dernière amélioration des formats étudiés auparavant, mais cette amélioration implique des contraintes au niveau de la structure même de la matrice,

Si aucun élément nul ne se trouve entre le premier et le dernier élément significatif de chaque ligne de la matrice de départ, alors ce format est le plus efficace parmi ceux étudiés. Si cette condition n'est pas respectée, alors ce format ne sera pas le plus efficace.

### B.4.2 Définition

- A_SKYLINE contient les éléments significatifs (*pouvant inclure des 0*).
- LA_SKYLINE contient l'indice de début de ligne dans A_SKYLINE.
- CA_SKYLINE contient l'indice de la colonne de l'élément référencé par LA_SKYLINE.

*Stockage Avant Compression :*

| | | | | |
|---|---|---|---|---|
| **1** | **0** | **3** | **0** | **0** |
| **0** | **0** | **7** | **-2** | **0** |
| **4** | **9** | **0** | **0** | **0** |
| **0** | **10** | **0** | **0** | **8** |
| **0** | **0** | **0** | **-9** | **0** |

*Stockage Après Compression :*

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **1** | **0** | **3** | **7** | **-2** | **4** | **9** | **10** | **0** | **0** | **8** | **-9** | **A_SKYLINE** |

| | | | | | |
|---|---|---|---|---|---|
| **1** | **4** | **6** | **8** | **12** | **LA_SKYLINE** |

| | | | | | |
|---|---|---|---|---|---|
| **1** | **3** | **1** | **2** | **4** | **CA_SKYLINE** |

### B.4.3 Exemple

La valeur 4 est stockée de la façon suivante :
En parcourant la matrice origine ligne par ligne, on trouve les valeurs significatives suivantes :

1, 0, 3, 7,-2, **4**, 9, 10, 0, 0, 8, -9

Donc :

A_SKYLINE(6) = 4

LA_SKYLINE(3) = 6

CA_SKYLINE(3) = 1

LA_SKYLINE (3) = 6 , car la 3$^{ème}$ ligne commence par la valeur 4 qui est à l'indice 6 du tableau A_SKYLINE.

CA_SKYLINE (3) = 1 , car la valeur référencée par l'indice 3 du tableau LA_SKYLINE a pour numéro de colonne 1.

### B.4.4 Gain de compression

Soit xbord le nombre d'éléments significatifs (*pouvant inclure des 0*). Soit L le nombre de lignes de la matrice creuse et C le nombre de colonnes de la matrice creuse :

**Nc (SKYLINE) = xbord + 2*min(C,L)**
**Gc (SKYLINE) = (n-(xbord+2*min(C,L)))/n**

On suppose que la matrice creuse est carrée, donc C = L = $\sqrt{n}$ et le gain de compression est :

**Nc (SKYLINE) = xbord + 2$\sqrt{n}$**
**Gc (SKYLINE) = (n-(xbord+2$\sqrt{n}$))/n**

L'efficacité de ce format de compression étant en fonction du patron de la matrice, par conséquent on procède au calcul du gain moyen de compression :

Dans le pire des cas on retrouve les éléments significatifs en priorité sur les extrémités de la matrice, dans ce contexte :

*xbord = n*
**Nc- (SKYLINE) = n + 2$\sqrt{n}$**
**Gc- (SKYLINE) = (n-(n+2$\sqrt{n}$))/n**

*Exemple :*

*xbord = n, pour la matrice suivante :*

| | | | | |
|---|---|---|---|---|
| **1** | **0** | **0** | **0** | **2** |
| **25** | **0** | **0** | **0** | **4** |
| **4** | **0** | **0** | **0** | **6** |
| **16** | **0** | **0** | **0** | **8** |
| **12** | **0** | **0** | **0** | **10** |

Dans le meilleur des cas les éléments sont groupés entre eux et aucune séquence d'éléments nuls ne se trouve entre deux éléments significatifs :

*xbord = x*
**Nc+ (SKYLINE) = x + 2$\sqrt{n}$**
**Gc+ (SKYLINE) = (n-(x+2$\sqrt{n}$))/n**

*Exemple :*

*xbord = x, pour la matrice suivante :*

| | | | | |
|---|---|---|---|---|
| **1** | **2** | **0** | **0** | **0** |
| **0** | **5** | **54** | **0** | **0** |
| **0** | **0** | **8** | **10** | **0** |
| **0** | **66** | **6** | **0** | **0** |
| **0** | **0** | **13** | **11** | **10** |

*Rappel des Notations :*

- x : nombre d'éléments non nuls de la matrice creuse.
- n : nombre total d'éléments de la matrice creuse, correspond pour une matrice de dimension (M,N) à n=M x N.
- nc : nombre d'éléments obtenus après la compression, correspond à la taille mémoire.
- Gc : le gain de compression, correspond à Gc = (n-nc)/n

# Annexe C

# Description des matrices

## C.1 Ordre et nombre total d'éléments

| Matrice | Ordre | Nombre total d'éléments |
|---|---|---|
| Rot1 | 3836 | 14714896 |
| Bres | 32715 | 1070271225 |
| Havre | 49754 | 2475460516 |

## C.2 Nombre d'éléments non nuls et Nombre d'éléments nuls

| Matrice | Nombre d'éléments non nuls | Nombre d'éléments nuls |
|---|---|---|
| Rot1 | 21608 | 14693288 |
| Bres | 188039 | 1070083186 |
| Havre | 288328 | 2475172188 |

## C.3 Matrices format SKYLINE

### C.3.1 Nombre d'éléments stockés et Nombre d'éléments nuls stockés

| Matrice | Nombre d'éléments stockés | Nombre d'éléments nuls stockés |
|---|---|---|
| Rot1 | 307990 | 286382 |
| Bres | 6403783 | 6215744 |
| Havre | 8402794 | 8114466 |

### C.3.2 Pourcentage d'éléments nuls stockés

| Matrice | Nombre d'éléments nuls stockés |
|---|---|
| Rot1 | *92.98%* |
| Bres | *97.06%* |
| Havre | *96.57%* |

### C.3.3 Nombre d'éléments stockés

| Matrice | NC |
|---|---|
| Rot1 | 315662 |
| Bres | 6469213 |
| Havre | 8502302 |

## C.4 Matrices format MORSE/Harwell-Boeing

### C.4.1 Nombre d'éléments stockés

| Matrice | NC |
|---|---|
| Rot1 | 47052 |
| Bres | 408793 |
| Havre | 626410 |

# Prototype Out-of-Core de solveur creux basé sur le contrôle de la pagination

Olivier Cozette, Abdou Guermouche, Cyril Randriamaro, Olivier Soyez, Gil Utard
LaRIA
Université de Picardie Jules Verne
5, rue du Moulin Neud
80000 Amiens

Décembre 2003

## 1 Introduction

Ce document d´ecrit le prototype de solveur out-of-core que nous avons r´ealis´e en combinant le solveurs multifrontale MUMPS d´evelopp´e par le CERFACS et l'INRIA avec le controleur de pagination MMUM-/MUSSEL d´evelopp´e au LaRIA dans la thèse de Mr. Olivier Cozette.

Une première id´ee pour traiter des grand problèmes est d'utiliser le système de pagination pr´esent sur tous les système. Cependant les politiques de gestion de m´emoire virtuelle mises en oeuvre dans les systèmes d'exploitation, ne sont pas adapt´ees au sch´ema d'ex´ecution des applications telles que la factorisation de matrices creuses. Deux approches sont alors envisageables :

**La restructuration du code :** les performances d'un code *out-of-core* pourront être am´elior´ees si le programmeur peut mettre à profit ses connaissances sur le système de pagination. En effet, le programmeur peut, si cela lui est possible, ordonnancer l'accès aux donn´ees afin de diminuer les d´efauts de page.

**Modification de la politique de pagination :** cette approche est difficile à mettre en oeuvre. En effet, une des rares approches consiste en g´en´eral à r´e´ecrire des parties du noyau du système d'exploitation qui gère la strat´egie de gestion m´emoire.

Restructurer le solveur MUMPS est une op´eration qui n'est pas envisageable dans l'imm´ediat. En effet ce solveur est constitu´e de plusieurs milliers de ligne de code. De plus comme le code est extr´ement irr´egulier, il n'est pas a priori possible de trouver une restructuration efficace pour toutes les matrices trait´ees. La solution retenue consiste donc à modifier la strat´egie de pagination en reportant celle ci au niveau de l'application. Nous avons donc d´efini un moniteur de pagination qui s'ex´ecute concurremment au solveur et qui gère la pagination de ce dernier. Ce moniteur a ´et´e ´ecrit en utilisant la bibliothèque de controle de la pagination de MMUM/MMUSSEL qui a ´et´e d´evelopp´e au LaRIA. Le solveur MUMPS a ´et´e annot´e par des directives à destination du moniteur de pagination.

Dans la première partie de ce rapport nous d´ecrivons le fonctionnement de MMUM/MMUSSEL. Ensuite nous d´ecrirons le comportement m´emoire de MUMPS, et le fonctionnement du moniteur de pagination ainsi que les interactions avec le solveurs. Nous donnerons les premiers r´esultats exp´erimentaux.

# 2 Gestion de la mémoire virtuelle au niveau utilisateur

Il y a d´ejà eu quelques approche de contrôle de la pagination avec des micronoyau (L3/L4[2], Chorus [1] ou mach[13]), mais ils ne sont pas ais´ement transposable sur des systèmes d'exploitation courants tels que *Linux* ou *Windows*.

Nous pr´esentons notre outil *MMUM/MMUSSEL* qui permet cette gestion sous *Linux* en mode utilisateur. Nous d´ecrivons son fonctionnement, puis montrons sur la factorisation LU, d´ecrite dans le chapitre pr´ec´edent, la mise en œuvre d'une politique permettant la modification de la pagination et ainsi une diminution importante de l'activit´e de pagination (´evincements de page et d´efauts de page).

## 2.1 Librairie *MMUM* et le module MMUSSEL

Les politiques usuelles de gestion de m´emoire virtuelle ne sont pas toujours adapt´ees aux besoins. La strat´egie de pagination optimale serait de connaître à l'avance l'ordre d'accès des pages et ainsi de calculer à chaque instant quelles pages doivent être ´evinc´ees et quelles pages doivent être charg´ees. En offrant la possibilit´e à l'utilisateur de pouvoir modifier la strat´egie de pagination en fonction de son application, on tend à se rapprocher de la solution optimale puisque sp´ecifique à une application donn´ee. C'est à partir de ce constat que nous avons d´evelopp´e un nouvel outil de gestion de la m´emoire virtuelle au niveau utilisateur.

### 2.1.1 Description

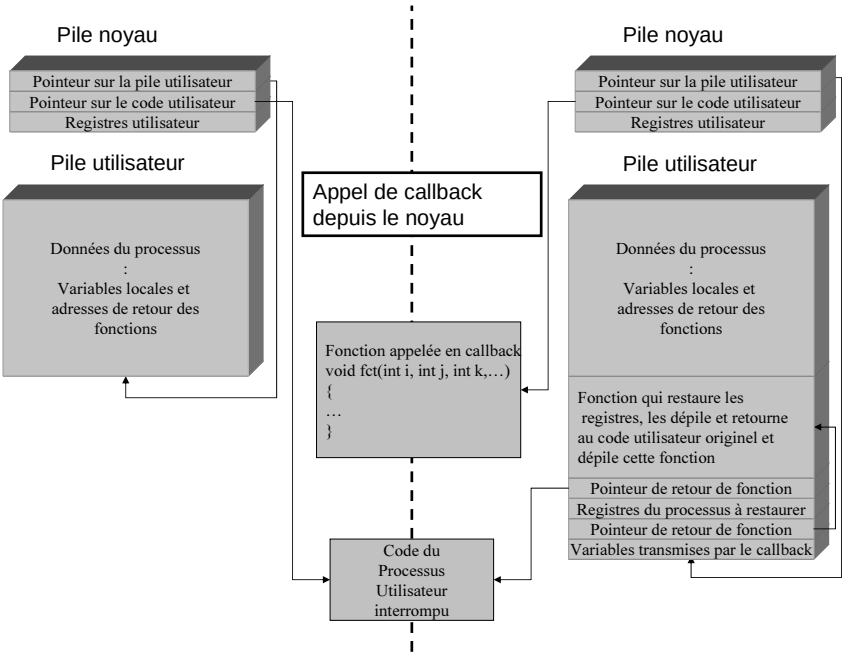
L'outil de gestion de la m´emoire virtuelle est compos´e d'une librairie appel´ee MMUM[1] (700 lignes en C), et d'un module noyau appel´e MMUSSEL[2][7] (2200 lignes en C). Le module noyau est ´ecrit pour LINUX, mais pourrait être ´ecrit pour n'importe quel Unix utilisant une organisation de la m´emoire virtuelle de type MACH[13] (BSD, OSF/1, Hurd,...). *MMUM* est une librairie consacr´ee au contrôle de la m´emoire virtuelle et à l'´ecriture de gestionnaire de m´emoire virtuelle au niveau applicatif. La librairie est combin´ee avec le module MMUSSEL. Ce module interagit avec le système d'exploitation pour g´erer les correspondances entre les pages physiques et les pages virtuelles (fig. 3).

Comme nous l'avons dit pr´ec´edemment, Linux, comme d'autres systèmes d'exploitation, s´eparent la m´emoire en r´egions auxquelles sont associ´ees des fonctions relatives à la pagination. Lorsque l'on demande à *MMUM/MMUSSEL* de g´erer une r´egion, le module Linux *MMUM* d´etourne la fonction *nopage* qui gère les d´efauts de page de cette r´egion. Ainsi, lorsqu'un d´efaut de page se produit sur cette r´egion, *MMUM* le reçoit et le transmet par un signal *SIGSEGV* ou un *callback* à *MMUSSEL* qui avertit le programme. N´eanmoins, Linux a besoin d'avoir imm´ediatement une page pour chaque d´efaut de page. Pour cela *MMUM* associe une page physique temporaire pour la page virtuelle fautive jusqu'à ce que l'application r´eagisse ou que *MMUM* reçoive d'autres ´ev´enements du système.

La première impl´ementation de *MMUM/MMUSSEL* utilisait un signal SIGSEGV pour communiquer, l'adresse de la page fautive ´etant mise dans des donn´ees statiques. L'inconv´enient de cette m´ethode ´etait la non r´eentrance : un second d´efaut de page avant le traitement du premier faisait perdre les informations du premier. Pour limiter ce problème, un module permettant de faire directement (sans passage par des signaux) des callbacks en mode utilisateur a ´et´e ajout´e par la suite.

**Module *callback*** La technique de callback est impl´ement´ee dans un module (250 lignes en C). Elle permet d'appeler une fonction en mode utilisateur depuis le noyau. L'int´erêt principale par rapport aux signaux est le passage d'un nombre variables de paramètre et la r´eentrance : plusieurs appels peuvent être effectu´es alors que les pr´ec´edents ne sont pas termin´es. Elle utilise la pile et les registres du mode

[1] Memory Management in User Mode.
[2] Memory Management at USer SpacE Level.




shadowbox

Figure 1: Callback : modification de la pile système et utilisateur

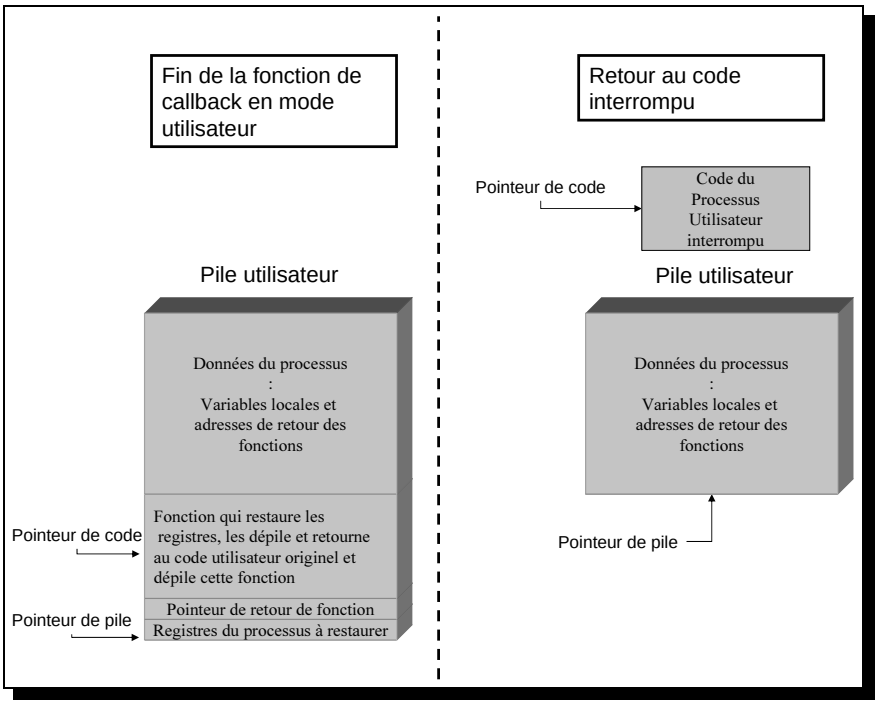



Figure 2: Callback : retour au code interrompu

utilisateur, qui sont stock´es dans le haut de la pile lorsque l'on est dans le mode noyau. Un callback est effectu´e par l'appel, en mode noyau, de la fonction *make_callback(handler,param,length)*. Le paramètre *handler* est une fonction r´eentrante appel´ee en mode utilisateur, *param* est un pointeur sur les paramètres qui seront envoy´es à la fonction utilisateur, le nombre de paramètres est d´etermin´e par *length*.

La fonction *make_callback()* ajoute à la pile du mode utilisateur le code d'une fonction de retour, les paramètres et une copie des registres utilisateurs (fig. 1). Ainsi, lorsque l'on sort du mode noyau, l'ex´ecution continue par la fonction *handler* avec ses paramètres. A la fin de cette fonction, notre fonction est ex´ecut´ee dans la pile, restaure les registres et redonne le contrôle au code interrompu (fig. 2). Cette technique est d´ependante de l'*Application Binary Interface* du processeur (*ABI*), mais peut être ais´ement ´ecrite grâce aux informations des fichiers `arch/TARGET/kernel/signal.c` et la structure `pt_regs` du noyau Linux

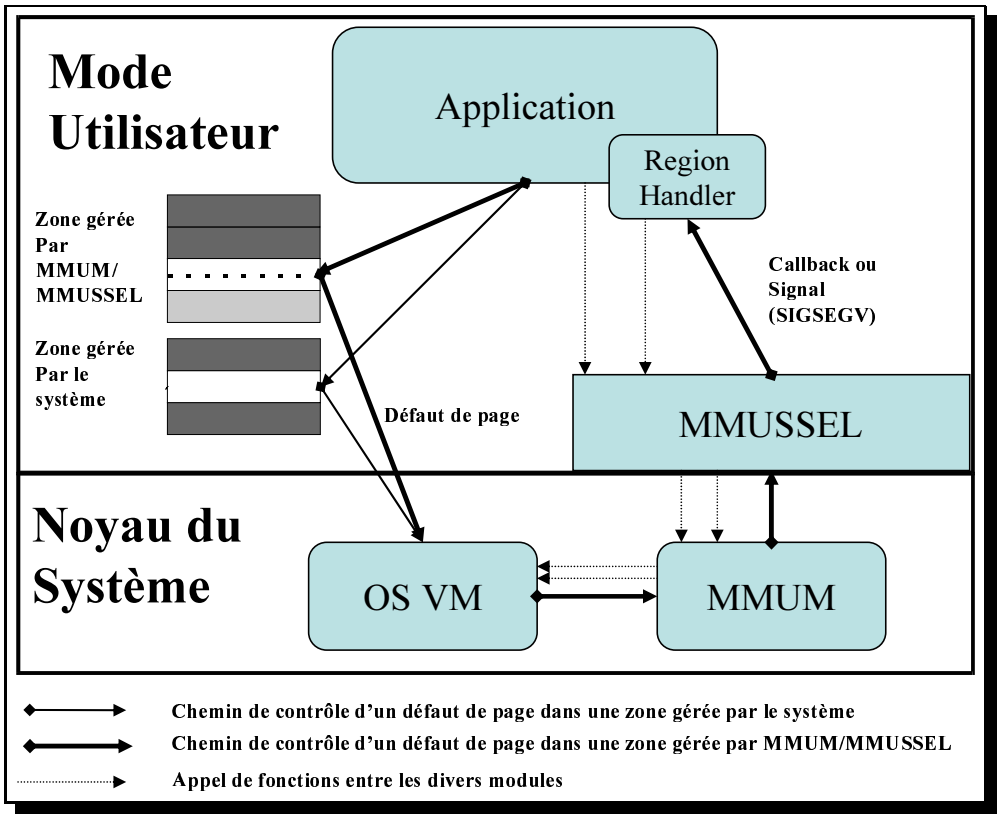



Figure 3: Sch´ema g´en´eral de *MMUM/MMUSSEL*

**Utilisation de *MMUM/MMUSSEL*** Nous allons maintenant d´ecrire l'interface de *MMUM/MMUSSEL*. L'application commence par cr´eer une r´egion par l'appel à la fonction `mmum create()`. La pagination de cette r´egion sera ensuite g´er´ee par l'application.

```
int mmum_create(long size,
        int (*init)(void * data, int région),
        int (*nopage)(MMUSSEL_ZONE région ,long address))
```

Le paramètre `size` est la taille en octets de la nouvelle r´egion. La fonction `init` r´ealise la cr´eation d'une nouvelle r´egion. La fonction `nopage` est appel´ee à chaque fois qu'un d´efaut de page est provoqu´e dans cette r´egion m´emoire.

Afin de d´efinir une nouvelle strat´egie de gestion de la m´emoire virtuelle, il faut dans un premier temps ´ecrire les deux proc´edures `init` et `nopage`. Plusieurs fonctions de *MMUM* sont conçues pour permettre la gestion de la m´emoire physique avec la nouvelle r´egion cr´e´ee. Nous pouvons parler plus sp´ecifiquement de :

- `mmum_newpage(void *a)`: associe une nouvelle page de la m´emoire physique à la page r´ef´erenc´ee par l'adresse (virtuelle) `a`. Les bits de lecture et d'´ecriture de la nouvelle page sont positionn´es à 0, c'est-à-dire non lue et non modifi´ee. Une fois la page r´ef´erenc´ee, elle est verrouill´ee en m´emoire et ne peut pas être ´evinc´e par le système.
- `mmum_releasepage(void *a)`: libère la page physique associ´ee à l'adresse (virtuelle) `a`.
- `mmum_xchg(void *a,void *b)`: ´echange les deux pages physiques associ´ees respectivement aux adresses (virtuelles) `a` et `b` (fig 4). Il n'est pas n´ecessaire d'associer une page physique à l'adresse virtuelle. Si l'adresse `a` est associ´ee à la page physique et qu'il n'y a pas de page associ´ee à l'adresse `b`, après l'appel à `mmum_xchg(a,b)`, l'adresse `b` est associ´ee avec la page pr´ec´edemment point´ee par l'adresse `a`, et aucune page n'est associ´ee à l'adresse `a`.
- `mmum_use_rw(void *a)`: retourne la valeur du bit de lecture/´ecriture de la page associ´ee à l'adresse (virtuelle) `a` et la remet à z´ero.

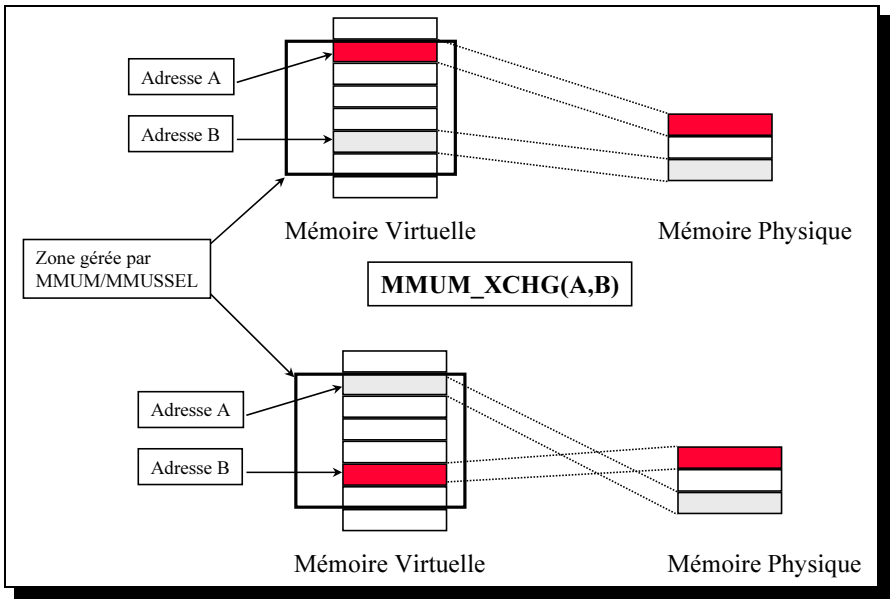



Figure 4: Echange de deux pages avec mmum_xchg

### 2.1.2 Projeter un fichier dans la mémoire virtuelle

Pour illustrer les applications potentielles de *MMUM/MMUSSEL*, nous pr´esentons un gestionnaire de m´emoire virtuelle qui permet de projeter un fichier dans une nouvelle r´egion m´emoire[3]. Le système de

[3]Notons que dans les syst`emes d'exploitation actuels la fonction `mmap` permet d'effectuer cette op´eration.

```
void main()  {
double *v;

int i;

  ...
/* Le fichier A est projeté sur le vecteur v */


v = mmum_create(N,fm_init,fm_nopage);

for (i=0;i<N;i++) v[i]=f(v,i);

...

}
```

Figure 5: Application n´ecessitant l'accès à un fichier projet´e dans une r´egion m´emoire.

gestion de m´emoire virtuelle est utilis´e par une application qui effectue quelques calculs sur un vecteur de taille $N$ comme le d´ecrit le programme de la figure5. Le contenu du vecteur est en fait le contenu du fichier plac´e dans une nouvelle r´egion m´emoire. La variable `V` est un pointeur sur le d´ebut de cette nouvelle r´egion. Les it´erations du programme modifient les valeurs du vecteur par la fonction `f`. Les accès aux fichiers sont transparents et sont effectu´es par le système de gestion de la m´emoire virtuelle, c'est-à-dire par la fonction `fm_init` et `fm_nopage`.

La figure 6 pr´esente le code source des fonctions `fm_init` et `fm_nopage` qui impl´emente les m´ecanismes pour projeter le fichier.

Dans cette impl´ementation, il n'y a qu'une seule page pr´esente en m´emoire physique à la fois. Du fait de l'initialisation, cette page est la première page du fichier (fonction `fm_init`). Quand un d´efaut de page survient, c'est-à-dire qu'une autre page de la r´egion m´emoire est acc´ed´ee, la page courante est ´ecrite sur le disque si elle a ´et´e modifi´ee, et la nouvelle page est charg´ee en m´emoire (fonction `fm_nopage`).

### 2.1.3 Copie mémoire rapide

Initialement, nous avons cr´e´e les fonctions de *MMUM/MMUSSEL* pour contrôler les m´ecanismes de gestion de la pagination, mais un autre int´erêt se trouve dans les fonctions de copies. Ainsi, lors d'un d´eplacement de donn´ees, les donn´ees sources ne sont plus utiles et peuvent être remplac´ees par n'importe quelle autre donn´ee, cette op´eration peut ainsi être remplac´ee par l'´echange des pages virtuelles sources et destinations.

Par exemple, si les donn´ees de d´epart ne sont pas r´eutilis´ees, `memmove(b,a,PAGE_SIZE)` est ´equivalent à `mmum_xchg(a,b)` dans le cas ou $a$ et $b$ correspondent au d´ebut d'une page ou commencent aux mêmes offsets d'une page. L'int´erêt de cette technique r´eside dans le fait que seuls les deux pointeurs de la table des pages sont modifi´es et seuls les quatre pointeurs des r´epertoires de pages et des r´epertoires de pages secondaires seront lus. Chacun de ces pointeurs seront charg´es dans le cache du processeur. Dans le cas où les lignes de caches ont 64 octets, nous devront lire 64*6 octets et les deux pointeurs modifi´es devront être r´e´ecrit. Nous obtenons 384 octets lus et 128 ´ecrits, soit 512 octets ´echang´es avec la m´emoire. Or, dans le cas d'un processeur x86, une page fait 4096 octets, et la copie correspond au mieux à la lecture de 4096 octets et l'´ecriture de 4096 octets, soit 8192 octets ´echang´es avec la m´emoire. Nous obtenons ainsi un rapport de 16 du point de vue du d´ebit m´emoire, ce qui semble int´eressant à priori.

```
void fm_start; / Adresse de la nouvelle région mémoire
*/
void fm_current; / Adresse virtuelle de la page courante */

/* Chargement dans la région */
int fm_size;  /* Taille de la nouvelle région mémoire */
int fm_fd; /* Descripteur de fichier */
void fm_init(void v,int size)
/* La nouvelle région mémoire est initialisée */
{
fm_start=v; /* Enregistre l'adresse de la nouvelle région */
fm_size=size; /* Enregistre la taille de la nouvelle région */

fm_fd=fopen(FILENAME,rw); /* Ouvre le fichier
à projeter dans la région */
/* mémoire  */


mmum_newpage(fm_start); /* Association d'une nouvelle page physique
*/
/* avec le début de la région */
/* Lit la première page de la région */
fread(fm_start,PAGE_SIZE,1,fm_fd);
fm_current=fm_start; /* Enregistre l'adresse virtuelle courante */
/* de la page physique */
}

void fm_nopage(void a) / La page adressée n'est pas présente
en */

/* mémoire physique */
{
/* Vérifie si la page a été modifiée */
/* Si oui, elle est inscrite sur le disque */

if (WRITTEN(mmum_use_rw(fm_current)))
{

fseek(fm_fd,(fm_current-fm_start)/PAGE_SIZE);
fwrite(fm_start,PAGE_SIZE,1,fm_fd);


}

mmum_xchg(fm_current,a);


/* La page physique associée à l'adresse */
/* virtuelle fm_current est à présent */
/* associée à l'adresse virtuelle a */
/* Lit la page correspondant à l'adresse */
/* virtuelle a */
fseek(fm_fd,(a-fm_start)/PAGE_SIZE);
fread(a,PAGE_SIZE,1,fm_fd);
fm_current=a; /* Enregistre l'adresse virtuelle courante */
/* de la page physique  */
}
```

Figure 6: Fonctions associ´ees à la nouvelle r´egion m´emoire pour l'impl´ementation de la projection de fichier.

Malheureusement, l'utilisation de `mmum_xchg()` requiert le passage en mode noyau suivi de quelques traitements du système avant d'arriver dans MMUM. Ce passage en mode noyau, l'arriv´ee dans *MMUM* et le retour en mode utilisateur coûtent 300 cycles sur un Pentium II 300Mhz. Le passage en mode noyau est in´evitable, mais il peut être acc´el´er´e sur les processeurs x86 par l'utilisation de l'instruction `SYSENTER/SYSEXIT` qui limite les v´erifications, et r´eduit à 150 cycles l'appel et le retour à une fonction de *MMUM* (sur le Pentium II 300Mhz). Cette am´elioration (400 lignes en assembleur) est accompagn´e d'une r´eduction maximale des traitements (moins de 40 instructions assembleurs avec la fonction `mmmum_xchg`). Une autre am´elioration consiste en la d´efinition de *mmum xchg str(a,b,NB PAGE)* qui permet d'´echanger $NB\_PAGE$ à la fois, ce qui r´eduit le nombre de transitions mode noyau/mode utilisateur.

Ces am´eliorations ayant ´et´e impl´ement´ees dans *MMUM/MMUSSEL*, nous obtenons les r´esultats exp´erimentaux suivants pour notre fonction de d´eplacement de donn´ees:

| | d´eplacement de 4Ko | d´eplacement de 4Mo |
|---|---|---|
| memmove | moins de 300Mo/s | moins de 300Mo/s |
| mmum_xchg_str | 600Mo/s | 1.2Go/s |

# 3 Moniteur de pagination

Nous venons de voir l'outil *MMUM/MMUSSEL* qui permet de contrôler la pagination en mode utilisateur, la question est maintenant comment utiliser au mieux cet outil pour faire une pagination adapt´e à MUMPS. Pour cela nous avons deux possibilit´es :

- Une première possibilit´e consiste à d´ecrire une politique de pagination g´en´erale, adapt´e à une classe d'applications et sans communication avec l'application. C'est l'option prise pour le gestionnaire de flux par Glass [10] et dans une version plus large par notre gestionnaire de flux du chapitre pr´ec´edent.

- Une deuxième possibilit´e consiste à d´ecrire une technique d'interaction entre l'application et la pagination. Cette option a ´et´e prise par Mowry [12], l'interaction avec l'application se faisant par l'indication par le compilateur des donn´ees qui vont être utiles et des donn´ees qui ne le seront pas ou plus.

Nous choisirons la deuxième solution. A cette fin, nous analyserons la pagination de MUMPS avec un outil bas´e sur une seconde version de *MMUM/MMUSSEL*. Nous en d´eduirons une première approche du comportement m´emoire de *MUMPS*. Nous d´efinirons une m´ethode pour am´eliorer la pagination de *MUMPS* et nous validerons cette m´ethode par une pr´esentation des gains obtenus.

## 3.1 Contrôleurs de pagination

Nous avons montr´e l'outil *MMUM/MMUSSEL* dans le chapitre pr´ec´edent, cet outil à ´evolu´e en devenant ind´ependant du processus qui l'utilise. Nous pressentons cette ´evolution, puis nous montrons son int´erêt sur l'exemple de la collecte de l'activit´e de la pagination sans modification de l'application.

Le principal problème de *MMUM/MMUSSEL* est la limitation de l'interaction avec le système d'exploitation. En effet, *MMUM/MMUSSEL* se contente d'intercepter les d´efauts de page et de verrouiller les pages utilis´ees en m´emoire, le nombre de pages maximum qui peuvent être verrouill´ees ´etant d´etermin´es au chargement du module. Ainsi, dans le cas où le système a besoin de plus de m´emoire pour d'autres programmes, il ne pourra pas en obtenir de *MMUM/MMUSSEL* ce qui peut ralentir ou bloquer le système. De la même manière, si le système a besoin de moins de m´emoire *MMUM/MMUSSEL* ne pourra l'utiliser. La solution adopt´ee considère que le système ´evince les pages les moins r´ecemment utilis´ees et qu'il doit toujours pouvoir ´evincer une page de la m´emoire s'il en a un besoin urgent.

Pour satisfaire cette dernière condition nous utiliserons deux processus, le client (g´er´e par *MMUM-/MMUSSEL*) et le moniteur. Ce dernier est un processus standard avec sa pagination g´er´ee par le système. Lorsqu'un d´efaut de page se produit dans le client, le moniteur reçoit un signal et doit charger la page concern´ee et la d´eplacer dans l'espace du client. Lorsque le système veut ´evincer une page du client, il la d´eplace dans le moniteur qui doit la sauvegarder sur le disque, cette page sera r´ef´erenc´ee comme une page r´ecemment utilis´ee, afin que le système ne l'´evince pas du moniteur. Le moniteur influence l'ordre dans lequel seront ´evinc´ees les pages du client, soit par l'´evincement explicite, soit par le changement de l'âge de dernière utilisation des pages (car le système ´evince en premier les pages les plus âg´ees). N´eanmoins, le moniteur ´etant un processus standard, si le système a un besoin urgent de m´emoire, le moniteur sera pagin´e avec les pages qui lui viennent du client.

Les nouvelles fonctions de *MMUM/MMUSSEL* sont :

**mmum_attach(pid,deb,fin,message()) :** cette fonction permet de r´ecup´erer la pagination du processus *pid* dans la zone comprise entre *deb* et *fin*. C'est-à-dire qu'a chaque d´efaut de page dans cette zone, le système appelle la fonction *message()*, qui devra lire la file de message, charger la page et red´emarrer le processus. Cette fonction est ´egalement appel´ee lors de la r´eception d'un *swapout*.

**mmum_xchg_monitor(pid1,addr1,pid2,addr2) :** cette fonction permet de retirer une page virtuelle de la m´emoire virtuelle de pid1 et de la mettre à l'adresse virtuelle addr2 dans pid2, cette fonction se fait par ´echange de pointeurs entre les tables de translation d'adresse des deux processus. Les utilisations de cette fonction sont l'´evincement explicite d'une page d'un client vers le moniteur, l'ajout dans le client d'une page qui vient d'être demand´ee (d´efauts de page). Cette fonction est utilis´ee par le système pour ´evincer une page d'un client vers un moniteur.

**mmum_restart(pid) :** cette fonction red´emarre un client qui s'est arrêt´e suite à un d´efaut de page. Il est n´ecessaire d'avoir plac´e la page demand´ee dans le client avant de le red´emarrer.

**mmum_userw_monitor(pid,addr) :** cette fonction permet de savoir si une page a ´et´e modifi´ee ou acc´ed´ee depuis le dernier appel à cette fonction.

**mmum_touch_monitor(pid,addr) :** cette fonction permet de modifier l'âge de dernier accès à une page. Sachant que le système ´evince les pages les moins r´ecemment utilis´ees, cette fonction permet de forcer les pages les plus importantes à rester en m´emoire (si possible).

**mmum_release_monitor(pid,addr) :** cette fonction libère la page physique associ´ee à la page virtuelle sans la stocker sur le disque.

Comme avec la version pr´ec´edente de *MMUM/MMUSSEL*, le moniteur reçoit les informations du système par callback ou par signaux, n´eanmoins lorsque le moniteur reçoit des pages ´evinc´ees du client, leur nombre peut être important et ne peut être pass´e en simple paramètre. Pour cette raison, une r´egion de m´emoire du moniteur est verrouill´ee et est utilis´ee comme file de r´eception des messages du système. Ces messages sont de deux types diff´erents :

**Défaut_de_page(pid,addr) :** signale qu'un client pid est suspendu suite à un d´efaut de page à l'adresse addr. Dans ce cas, le moniteur doit charger la page demand´ee, la transf´erer dans l'espace virtuel du processus et red´emarrer le processus.

**Swapout(pid,addr,addr_monitor) :** signale que le système a ´evinc´e la page virtuelle addr du client pid et l'a transf´er´ee à l'adresse addr monitor du moniteur. Le moniteur devra sauvegarder cette page avant de lib´erer son emplacement par *mmum_release_monitor(getpid(),addr_monitor)*. Il faut noter que si le moniteur ne libère pas suffisamment rapidement les pages, le système peut d´ecider de paginer le moniteur, ce qui fait perdre l'int´erêt de *MMUM/MMUSSEL*, mais ne bloque pas le système.

Un premier exemple de l'utilisation de *MMUM/MMUSSEL* 2 est d´ecrit dans la figure8 pour le moniteur et la figure 9 pour le client qui lance le moniteur. Dans cet exemple, un programme, que nous nommerons client, lance un moniteur qui prendra en charge la gestion d'un fichier projet´e en m´emoire. Le moniteur commence par demander le contrôle de la pagination du client par *mmum_attach()*, et attends les messages de pagination du clients dans la zone comprenant *a[]*. Lorsque le client accède à une page non pr´esente du tableau, il est arrêt´e et un message est envoy´e au moniteur. Celui-ci charge la page en m´emoire et la transfère dans l'espace virtuel du client (avec *mmum_xchg()*). Lorsqu'une page est ´evinc´ee par le système (swapout), il la transfère dans l'espace virtuel du moniteur, qui l'´ecrit sur le disque et la retire de son espace virtuel (*mmum_release()*.

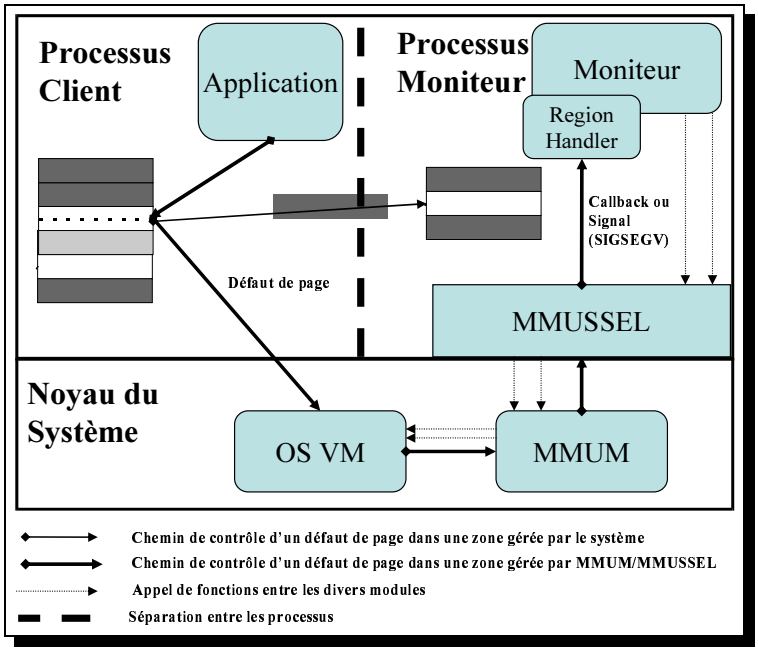



Figure 7: Interaction entre le moniteur, *MMUM/MMUSSEL* et le client

Cette version de *MMUM/MMUSSEL* impl´ement´ee avec un noyau 2.4 de Linux a ´et´e plus difficile a impl´ementer qu'avec le 2.2, car les op´erations associ´ees aux zones m´emoires ont ´et´e r´eduites depuis les versions 2.2 et 2.0. Il n'y a ainsi plus de fonction swapout associ´e à une page, ce qui n´ecessite de cr´eer un p´eriph´erique de stockage secondaire pour r´ecup´erer les swapout. Pour pouvoir transf´erer *MMUM* sur un noyau 2.4, nous avons du commencer par l'impl´ementer sous forme d'une partie du noyau, avant de cr´eer un pilote de p´eriph´erique d'´echanges qui a permis de le transf´erer en module.

Nous pouvons noter que nous gardons les fonctionnalit´es de la première version, ce qui permet de faire fonctionner les anciens programmes. Cette nouvelle impl´ementation a ensuite ´et´e utilis´ee pour donner les d´efauts de page d'une application et ainsi essayer d'am´eliorer l'application nomm´ee *MUMPS*.

# 4 Contrôle de la pagination de MUMPS

Dans cette section, nous allons analyser le programme *MUMPS* : *MUMPS* [4, 5] est un solveur pour systèmes lin´eaires creux. Il est bas´e sur la m´ethode multifrontale[8, 9] qui est une m´ethode directe efficace pour la factorisation de matrices creuses. Cependant, comme les autres m´ethodes directes, elle est connue pour sa consommation m´emoire importante en comparaison avec les m´ethodes it´eratives. Ainsi, lorsque la m´emoire est limit´ee, on observe une forte activit´e de la pagination. Nous avons donc voulu ´etudier cette pagination pour concevoir un moniteur qui am´eliore les performances. A cette fin, nous

```
#include <stdio.h>
#include <mmussel2.h>

int fd ;
long address_begin;

void message(
  int swno,
  int pid,
  long address,
  long local_address)
{
if (swno==NOPAGE)
  {
  char * t=malloc_aligned(PAGE_SIZE);
// malloc_aligned donne une adresse
// PAGE_SIZE alignée
  llseek(fd,address-address begin,SEEK_SET);
  read(fd,t,PAGE_SIZE);
  mmum_xchg(getpid(),t,pid,a);
  free(t);
  }
if (swno==SWAPOUT)
{
// Quand Linux/MMUM évince une page,
// elle est transféré dans cet espace virtue
// à l'adresse local_address
llseek(fd,address-address_begin,SEEK_SET);
write(fd,local_address,PAGE_SIZE);
mmum_release(getpid(),local_address);
}
}

int main(int argc, char ** argv)
{
fopen("/tmp/mappedfile",O_RDWR);
int pid;
if (argc<2)
 printf("Utilisation :\
  ./monitor pid_of_monitored_process\
  address_of_the_mapped_file\n");
fd=fopen("/tmp/mappedfile",O_RDWR);
pid=atoll(argc[1]);
address_begin=atoll(argv[2]);

mmum_attach(pid,address_begin,address_begin

    + 400*PAGE_SIZE,pf,swo);
//Signale que le moniteur est demaré
kill(pid,SIGUSR1);

while (1) sleep(100);
}
```

Figure 8: Un exemple de projection en m´emoire d'un fichier dans un autre processus : le moniteur

```
#include <stdio.h>

main()
{
char a=malloc_aligned(400*PAGE_SIZE);
char pid[10];

long w;
char address[20];
int status;
sprintf(pid,"%d",getpid());

sprintf(address,"%ld",(long)a);
if (fork()==0)

  execlp("./monitor",
     pid,address);


//Attende du démarrage du moniteur

wait(&status) ;



for (w=0;w<400*PAGE_SIZE;w++)
  {
  if (a[w]<10) a[w]++;
// a[] is the mapped file
}
}
```

Figure 9: Un exemple de projection en m´emoire d'un fichier dans un autre processus : le client

analysons *MUMPS* avec l'outil d'analyse de trace dont nous venons de parler. Nous validons ensuite cette analyse par une ´etude du code de l'application. Nous utilisons cette ´etude pour d´efinir un moyen de communication entre l'application et la pagination. A cette fin, nous ´ecrivons un gestionnaire de pagination impl´ementant cette interface. Enfin, nous montrons les gains obtenus grâce à cette m´ethode.

## 4.1 Analyse de la pagination de *MUMPS*

Nous allons commencer par ´etudier la trace avec notre programme d'analyse de la trace. Pour cela nous avons utilis´e une machine avec 32Mo de m´emoire afin d'avoir le maximum de d´efauts de page et donc d'information. Nous avons analys´e *MUMPS* avec deux matrices (TWOTONE et SHIP 003).

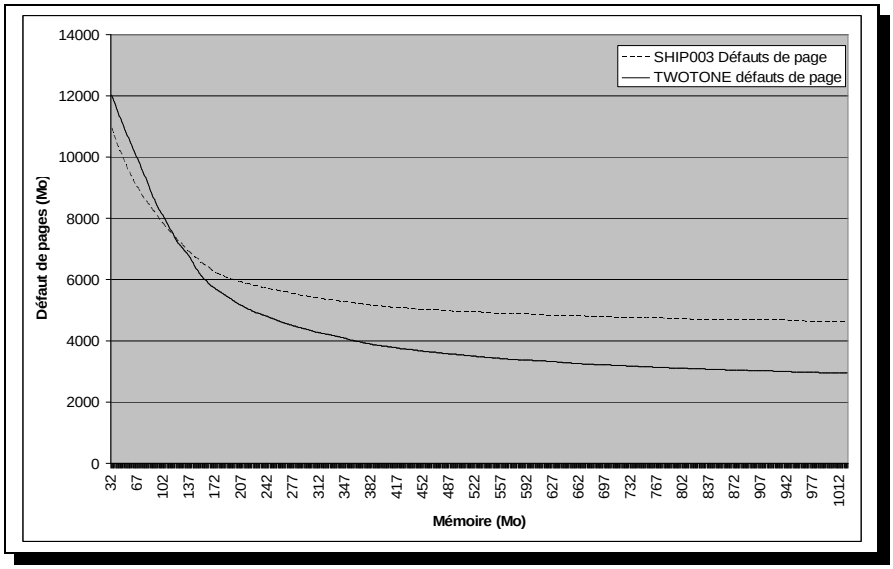



Figure 10: Taille m´emoire optimale pour *MUMPS*

La figure 10 donne le nombre estim´e de d´efaut de page en fonction de la m´emoire de la machine. On constate que le nombre de d´efauts de page est divis´e par deux vers 160Mo pour *TWOTONE* et vers 184Mo pour *SHIP 003*. Il est ainsi possible de d´eduire quelle serait l'augmentation de performances par rapport à la taille de la m´emoire.

Cette première ´etude ne nous renseigne pas sur la possibilit´e d'am´eliorer la pagination. Pour cela, nous avons calcul´e l'espace de travail utilis´e à chaque seconde. Nous pouvons voir sur la figure 11 que l'espace de travail est fortement variable, n´eanmoins l'ensemble des abscisses sans point correspond à un espace de travail contenu en m´emoire, c'est-à-dire aucun d´efaut de page. Nous pouvons en d´eduire que le pr´echargement de page et le d´echargement de pages inutiles pendant ces p´eriodes est non p´enalisant, car le disque n'est pas utilis´e pour la pagination. Le problème est de connaître quel est l'espace non utilis´e pendant ces p´eriodes, et quels sont les pages à pr´echarger et d´echarger. Pour connaître ces paramètres, nous allons ´etudier plus en profondeur *MUMPS*.

## 4.2 Description des accès mémoire de *MUMPS*

La m´ethode multifrontale est bas´ee sur un arbre d'´elimination [11] qui est une r´eduction transitive du graphe de d´ependance des tâches de factorisation. En pratique, nous utilisons une structure appel´ee arbre d'assemblage obtenue par fusion de certains nœuds de l'arbre d'´elimination.

Un exemple d'arbre assemblage est donn´e dans la figure 13. L'arbre est construit à partir de la matrice donn´ee dans la figure. Les deux feuilles, sont ind´ependantes les unes des autres alors que la

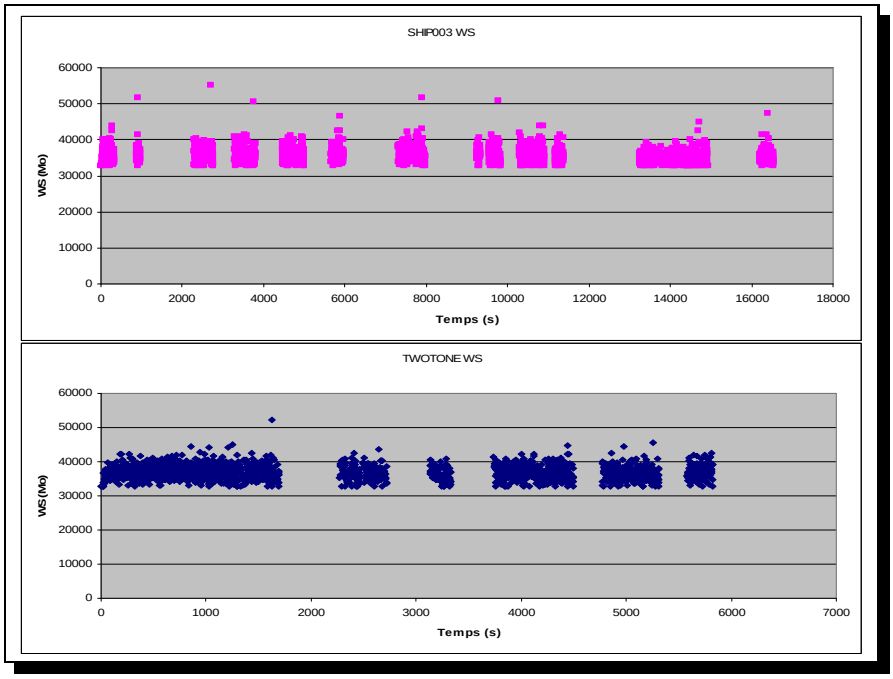



Figure 11: Espace de travail (*Working Set*) de *MUMPS* au cours du temps pour *MUMPS*

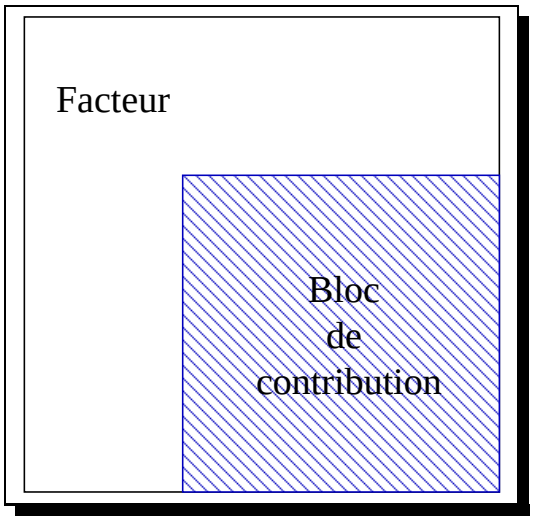



Figure 12: structure d'une matrice frontale.

racine doit recevoir des contributions des feuilles.

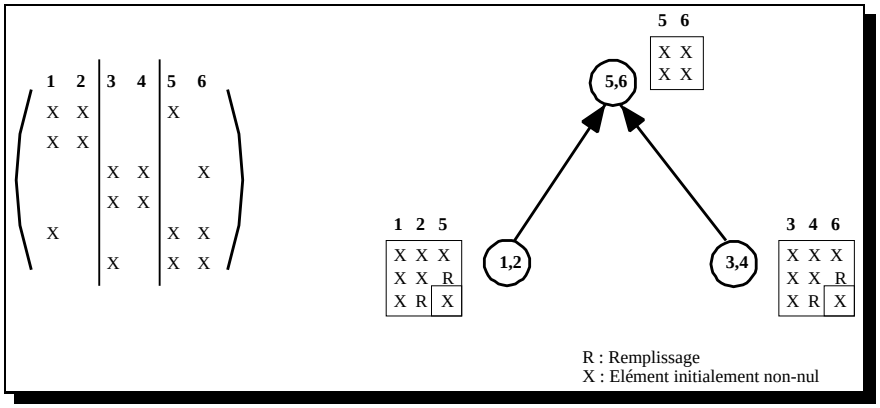



Figure 13: exemple de matrice et d'arbre d'assemblage associ´e.

Dans la m´ethode multifrontale, la factorisation de la matrice est effectu´ee par une succession de factorisations partielles de petites matrices denses, associ´ees à chaque nœud de l'arbre appel´e *matrices frontales*. La structure de la matrice frontale est donn´ee dans la figure12. La matrice est compos´ee de deux parties. La première correspond au bloc *facteur*, appel´e aussi bloc des variables complètement assembl´ees, qui contient les variables qui ont ´et´e ´elimin´ees dans le processus de factorisation de la matrice frontale. Le *bloc de contribution* correspond à la deuxième partie de la matrice frontale, il correspond aux variables mises à jour (compl´ement de Schur) pendant le processus de factorisation.

Une fois la factorisation partielle de la matrice frontale effectu´ee, le bloc de contribution est envoy´e au père du nœud auquel elle correspond. Une fois que toutes les contributions des fils sont disponibles au niveau du père, elles peuvent être assembl´ees (i.e. somm´ees avec les valeurs contenues dans la matrice du père). L'algorithme d'´elimination est un parcours en remont´ee d'abord de l'arbre d'assemblage où un père est trait´e d´es que ses fils l'ont ´et´e. L'algorithme utilise trois emplacements de stockage dans une r´egion m´emoire contigu, une r´egion pour stocker les facteurs, une r´egion g´er´ee comme une pile pour stocker les blocs de contribution, et une dernière r´egion pour stocker la matrice frontale active (courante). Une description d´etaill´ee de la gestion m´emoire est donn´ee par P. Amestoy[3].

Pendant le processus de factorisation, l'espace m´emoire n´ecessaire au stockage des facteurs ne fait que croître en taille alors que celui correspondant à la pile (espace contenant les blocs de contribution) varie suivant la nature de l'op´eration effectu´ee. En effet, quand la factorisation partielle d'une matrice frontale est effectu´ee, un bloc de contribution est empil´e ce qui augmente la taille de la pile. D'un autre cot´e, lors de la phase d'assemblage d'une matrice frontale, les blocs de contribution des fils du nœud courant sont consomm´es et ce qui occasionne une diminution de la taille de la pile. La structure de l'espace de travail de *mumps* d´ecrite pr´ec´edemment est donn´ee à la figure14.

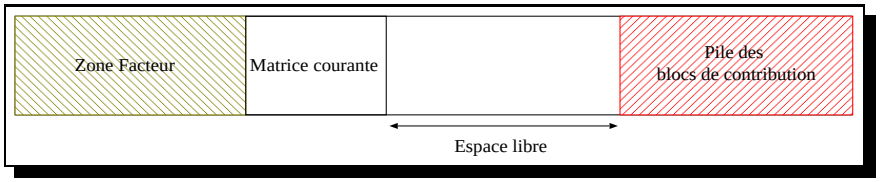



Figure 14: structure de la m´emoire utilis´ee par *mumps*.

## 4.3 Stratégie d'optimisation de la pagination

Nous venons d'exposer le fonctionnement de *MUMPS*, nous pouvons en d´eduire certaines caract´eristiques int´eressantes pour l'am´elioration de la pagination :

**Temps d'inactivité de la pagination :** le temps de calcul d'un front peut-être ´elev´e tout en rentrant en m´emoire, on observe peu de pagination pendant ce temps. Cette caract´eristique rend ce calcul propice au pr´echargement des donn´ees utiles dans le futur et au d´echargement des donn´ees inutiles.

**Prévisibilité des données utiles :** l'arbre d'assemblage est connu avant la factorisation, il est ainsi possible de connaître les donn´ees qui seront n´ecessaires pour les prochains calculs de facteurs, ainsi que les donn´ees inutiles. Ces informations permettent un pr´echargement ou un d´echargement des donn´ees.

Afin d'utiliser ces caract´eristiques, nous avons cr´e´e un deuxième processus nomm´e moniteur. Ce processus utilise *MMUM/MMUSSEL* pour influer sur la pagination de *MUMPS* et ainsi pr´echarger ou d´echarger des pages. Le mode de communication entre ces deux processus est bas´e sur le concept de priorit´e : *MUMPS* donne une priorit´e diff´erente aux r´egions de m´emoire suivant leur utilit´e pr´esente et à venir.

### 4.3.1 Description du moniteur de pagination

Le moniteur est un programme (1800 lignes en C) lanc´e par *MUMPS* qui utilise *MMUM/MMUSSEL* pour contrôler la pagination (fig. 15) reçoit de *MUMPS* des priorit´es pour des plages d'adresse m´emoire par la fonction *monitor_regmem(adr_debut,adr_fin,priorite)*. Ces priorit´es sont ensuite utilis´ees pour connaître, suivant la m´emoire disponible quelles sont les donn´ees à conserver en m´emoire et quelles sont les donn´ees à ´evincer de la m´emoire. L'algorithme cherche quelle est la priorit´e la plus faible qui permet de stocker en m´emoire toutes les pages de priorit´es identiques ou sup´erieures. Le but est ensuite de stocker et d'´evincer les pages pour obtenir cette configuration dite stable. Le problème du moniteur est qu'il n'a qu'une vue partielle des priorit´es, car *MUMPS* donne les informations au fur et à mesure du calcul. Le moniteur doit donc recalculer un ´etat stable à chaque fois, ce qui peut engendrer des comportements inadapt´es.

Parmi les priorit´es que peut passer *MUMPS* au moniteur, quelques unes sont sp´ecifiques : la priorit´e nulle indique qu'une page doit être d´echarg´ee de la m´emoire car elle ne sera pas utilis´ee avant longtemps, la priorit´e maximale indique qu'une page doit absolument être pr´esente en m´emoire (à moins qu'il n'y ai pas assez de m´emoire). Ces deux priorit´es sont les seules utiles lorsque *MUMPS* ne fait pas de pr´ediction sur les prochains accès m´emoires.

Une primitive d'interaction entre *MUMPS* et le moniteur est *monitor_release(addr_debut,addr_fin)*. Cette commande signifie qu'une plage d'adresses contient des donn´ees qui ne seront plus jamais utilis´ees, c'est-à-dire que ces donn´ees peuvent être ´evinc´ees de la m´emoire sans être sauvegard´ees sur le disque. Ceci peut procurer des gains en terme de volume d'entr´ees/sorties et de latence. Cette primitive est principalement utilis´ee pour la gestion de la pile dans *MUMPS*. En effet, lorsque la zone m´emoire en tête de la pile est lib´er´ee au niveau de l'application, les donn´ees qu'elle contenait peuvent être d´etruites. Ainsi, il est possible de la lib´erer avec *monitor_release*.

Une autre utilisation de cette commande est le d´eplacement de donn´ees. Ainsi, lors d'un d´eplacement de donn´ees, une première page est lue et copi´ee dans une autre, et ainsi de suite. La page qui vient d'être lue pour être copi´ee restera en m´emoire du fait de l'algorithme LRU. Or, non seulement cette page n'est plus utile en m´emoire mais en plus, il est inutile de la sauvegarder. Pour cela nous avons remplac´e la fonction `memmove(adresse1,adresse2,taille)` par une fonction qui emploie la primitive *mum_release()* (voir fig. 16).

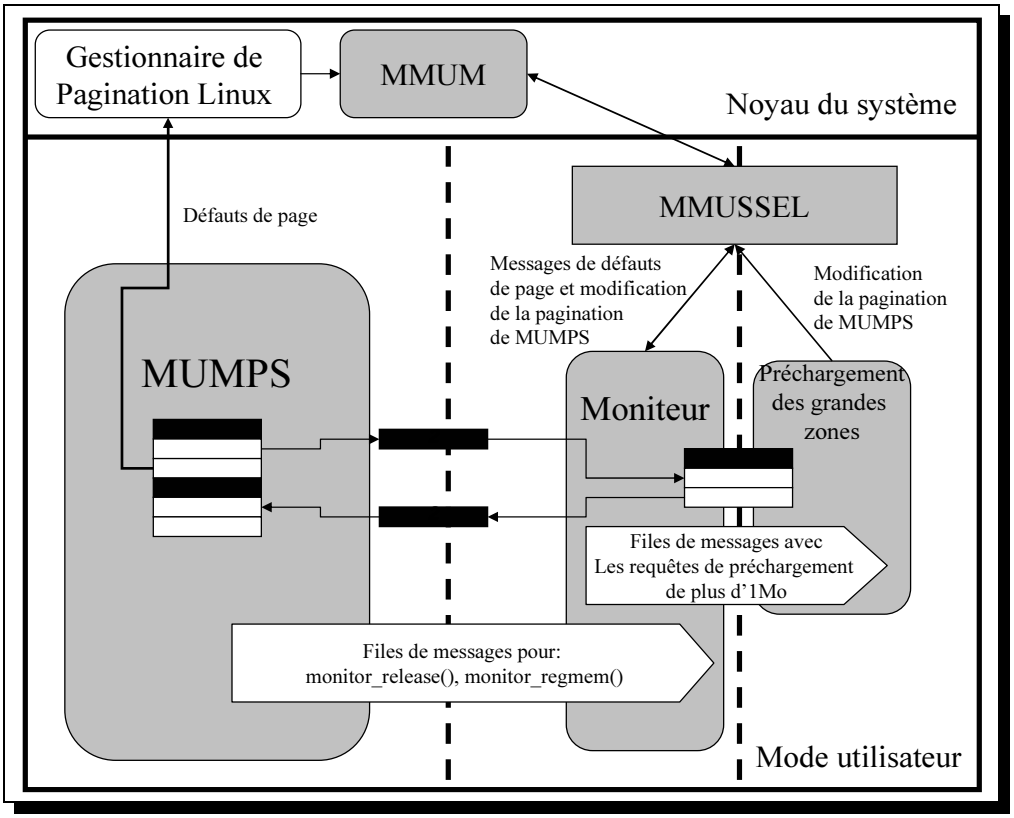



Figure 15: Relation entre le moniteur et *MUMPS*

1. Si ADRESSE1 est au milieux d'une page copier la page alors
   (a) B=TAILLE_PAGE – (ADRESSE1 & TAILLE_PAGE )
   (b) ADRESSE1=ADRESSE1+B
   (c) ADRESSE2= ADRESSE 2+B
   (d) TAILLE=TAILLE-B
2. Tant que TAILLE ¿ TAILLE_PAGE faire
   (a) Copier TAILLE_PAGE octets de ADRESSE1 vers ADRESSE2
   (b) ADRESSE1=ADRESSE1+TAILLE_PAGE
   (c) ADRESSE2=ADRESSE2+TAILLE_PAGE
   (d) TAILLE=TAILLE-TAILLE_PAGE
   (e) monitor_release(ADRESSE1,ADRESSE1+TAILLE_PAGE-1)
3. Copier TAILLE octets de ADRESSE1 vers ADRESSE2

Figure 16: D´eplacements de donn´ees avec *mmum release()*

Le moniteur et *MUMPS* utilisent un système de passage de messages pour communiquer. Ce système est bas´e sur une r´egion de m´emoire partag´ee et des s´emaphores. Le moniteur lit les messages et les traite un à un. N´eanmoins, pour des raisons de performances, les chargements de grande r´egion de m´emoire (plus de 1Mo) sont confi´es à un troisième processus.

Certaines commandes demandent une synchronisation, c'est-à-dire qu'elles peuvent ´evincer certains messages de la file des messages ou attendre la fin de leur traitement. C'est le cas avec les *monitor_releases()* plac´es après les *monitor_regmem()*. En effet, il n'est plus n´ecessaire de charger une r´egion si elle a ´et´e ´evinc´e par un *monitor release()*. Un cas semblable se produit avec *monitor regmem()* qui efface les messages pr´ec´edents concernant une r´egion qui est inclue. Ainsi, lorsque *MUMPS* donne l'information qu'une r´egion à une faible priorit´e puis que cette r´egion à une forte priorit´e, le moniteur ne traitera pas la première commande. Un autre cas de synchronisation concerne les *monitor_release()*. Par exemple, si le moniteur ne libère pas imm´ediatement les pages, *MUMPS* peut r´eutiliser ces pages avant la prise en compte du *monitor_release()*. Puis le moniteur d´etruira le contenu de ces pages, les donn´ees de cette page auront ainsi ´et´e perdues alors qu'elles sont encore utiles. Ce problème est r´esolu par la synchronisation du moniteur avec *monitor_release_sync()*

### 4.3.2 Interactions *mumps*-moniteur

Les communications entre *mumps* et le moniteur sont faites à des moments pr´ecis du processus de factorisation. Ainsi suivant le type de l'op´eration effectu´ee, assemblage ou factorisation partielle, *mumps* ´emet des requêtes visant à aider la gestion m´emoire au niveau du moniteur. Un sch´ema r´esumant les diff´erentes phases d'appels des fonctions de communication entre *mumps* et le moniteur sont donn´ees à la figure 17.

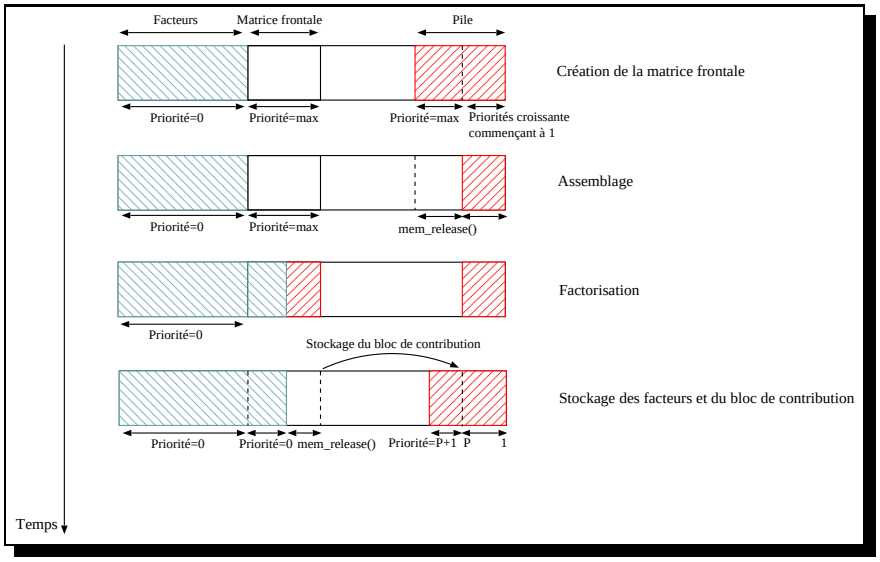



Figure 17: Sch´ema d'interaction *mumps*-moniteur.

Les interactions entre *mumps* et le moniteur sont faites à chaque ´etape de la vie de la matrice frontale. Une description des diff´erentes phases du traitement de la matrice frontale est donn´ee ci-dessous :

**Création de la matrice frontale.** Avant la cr´eation de la matrice frontale, la priorit´e de la r´egion qui doit la contenir est mise à la priorit´e maximale à l'aide d'un *reg_mem* (cr´eation de la matrice frontale à la figure 17).

**Assemblage.** Avant chaque ´etape d'assemblage, *mumps* ´emet des requêtes de type *reg_mem* sur les r´egions qui seront acc´ed´ees durant cette phase. Ainsi il positionne la priorit´e de ces r´egions à la valeur maximale pour assurer leur pr´esence en m´emoire physique. Dès que l'assemblage d'un bloc de contribution est termin´e, la r´egion m´emoire correspondante dans la pile est lib´er´ee à l'aide d'un *mem_release*.

**Factorisation.** La priorit´e de la r´egion m´emoire qui correspond à la matrice frontale ´etant d´ejà maximale, la factorisation se d´eroule sans communication entre *mumps* et le moniteur. Une fois la factorisation finie, le bloc de contribution de la matrice frontale est stock´e dans la pile. Il prend alors une priorit´e ´egale à la priorit´e du bloc de contribution qui ´etait en tête de pile plus un (priorit´e cb=priorit´e tête pile+1). En ce qui concerne les facteurs de la matrice frontale, leur priorit´e est mise à z´ero puisqu'ils ne seront pas r´eacc´ed´es pendant la factorisation.

## 4.4 Résultats

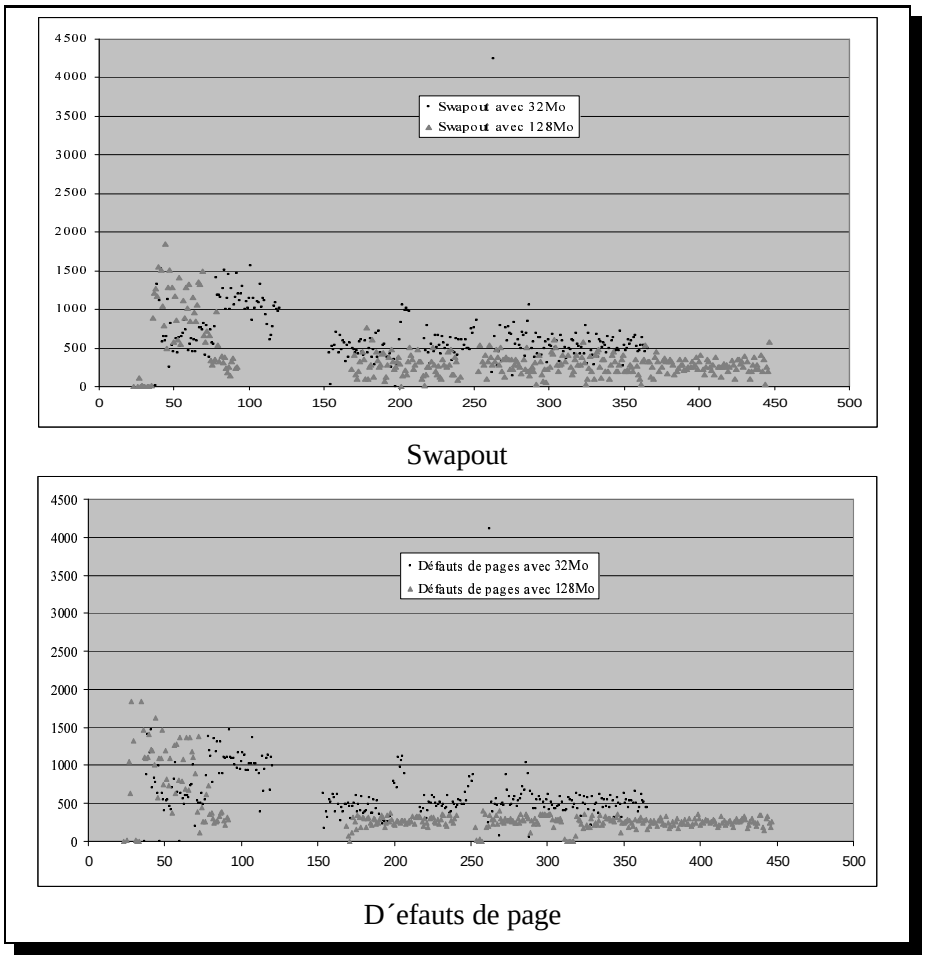



Figure 18: Gestion de la m´emoire pour la matrice gupta3 avec 32Mo et 128Mo de m´emoire

Nous avons test´e le moniteur et *MUMPS* sur plusieurs matrices sym´etriques et non sym´etriques. Sur chacune de ces matrices, nous avons effectu´e les optimisations que nous venons de d´ecrire. Nous exposons les gains dans la figure 21 et les temps dans la figure 20 et 22.

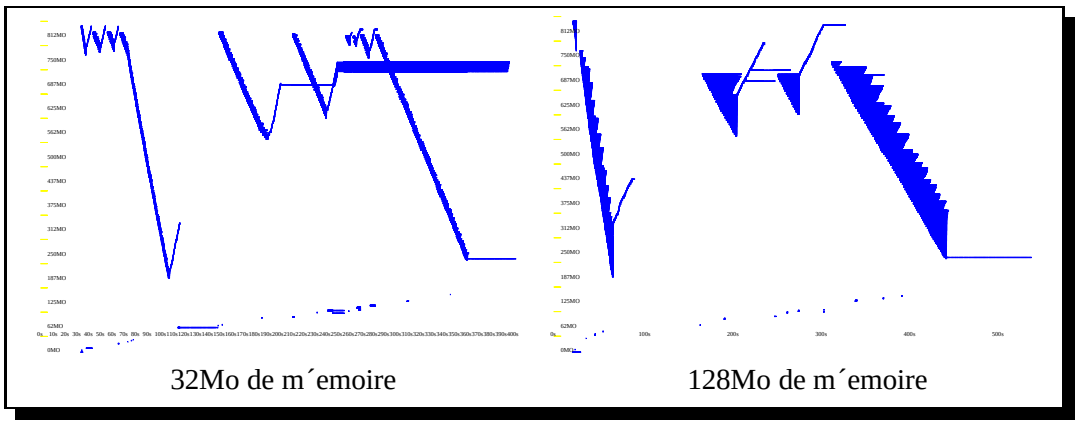



Figure 19: Utilisation de la m´emoire pour la matrice gupta3 avec 32Mo et 128Mo de m´emoire

### 4.4.1 Matrice GUPTA3

Une première remarque que nous pouvons faire sur la matrice GUPTA3 est la diminution de m´emoire acc´elère *MUMPS*. La figure. 18 montre que vers 100 secondes, il n'y a plus d'´evincement de page (swapout) avec 128Mo de m´emoire, mais qu'il continue à en avoir avec 32Mo. Cela s'explique par le fait que la m´emoire est insuffisante pour 32Mo et qu'ainsi les donn´ees utilis´ees dans la phase de calcul pr´ec´edente sont lib´er´ees. Entre 100 et 150 secondes, un calcul sur de nouvelles donn´ees pouvant r´esider en m´emoire est effectu´e. Ensuite, comme l'indique la figure 19, *MUMPS* utilise une autre r´egion de m´emoire et certaines donn´ees qui ´etaient en m´emoire deviennent inutiles, il faut donc les ´evincer de la m´emoire. Avec 32Mo de m´emoire les donn´ees utilis´ees avant le calcul avaient d´ejà ´et´e ´evinc´ees pendant le calcul, avec 128Mo de m´emoire il faut les ´evincer après le calcul. Le même ph´enomène se produit par la suite sur de plus petites dur´ees, ce qui explique qu'il y a un taux d'´evincements (swapout) inf´erieur avec 128Mo. Ainsi, ces d´echargements en avance compensent celui venant d'un nombre de chargements/´evincements des pages moins importants (environ 300000 avec 128Mo et 400000 avec 32Mo).

Pour r´esumer, avec 32Mo les donn´ees commencent à être ´evinc´ees plus tôt de la m´emoire qu'avec 128Mo, or les donn´ees utilis´e dans la phase de calcul pr´ec´edent ne seront pas r´eutilis´ees, elles doivent donc être ´evinc´ees le plus tôt possible de manière à profiter au maximum du d´ebit du disque.

| Matrice | M´emoire | Temps (s) avec le moniteur | Temps sans le moniteur |
|---|---|---|---|
| GUPTA3 | 32Mo | 374 | 424 |
| SHIP 003 | 32Mo | 85168 | 92164 |
| TWOTONE | 32Mo | 5626 | 6067 |
| XENON2 | 32Mo | 5737 | 6647 |
| GUPTA3 | 64Mo | 408 | 444 |
| SHIP 003 | 64Mo | 2764 | 3896 |
| TWOTONE | 64Mo | 1867 | 1571 |
| XENON2 | 64Mo | 1876 | 2118 |
| GUPTA3 | 128Mo | 524 | 576 |
| SHIP 003 | 128Mo | 1114 | 1207 |
| TWOTONE | 128Mo | 429 | 292 |
| XENON2 | 128Mo | 734 | 678 |

Figure 20: Temps d'ex´ecutions s´equentiel de *MUMPS* avec et sans le moniteur sur diverses matrices

### 4.4.2 MUMPS séquentiel

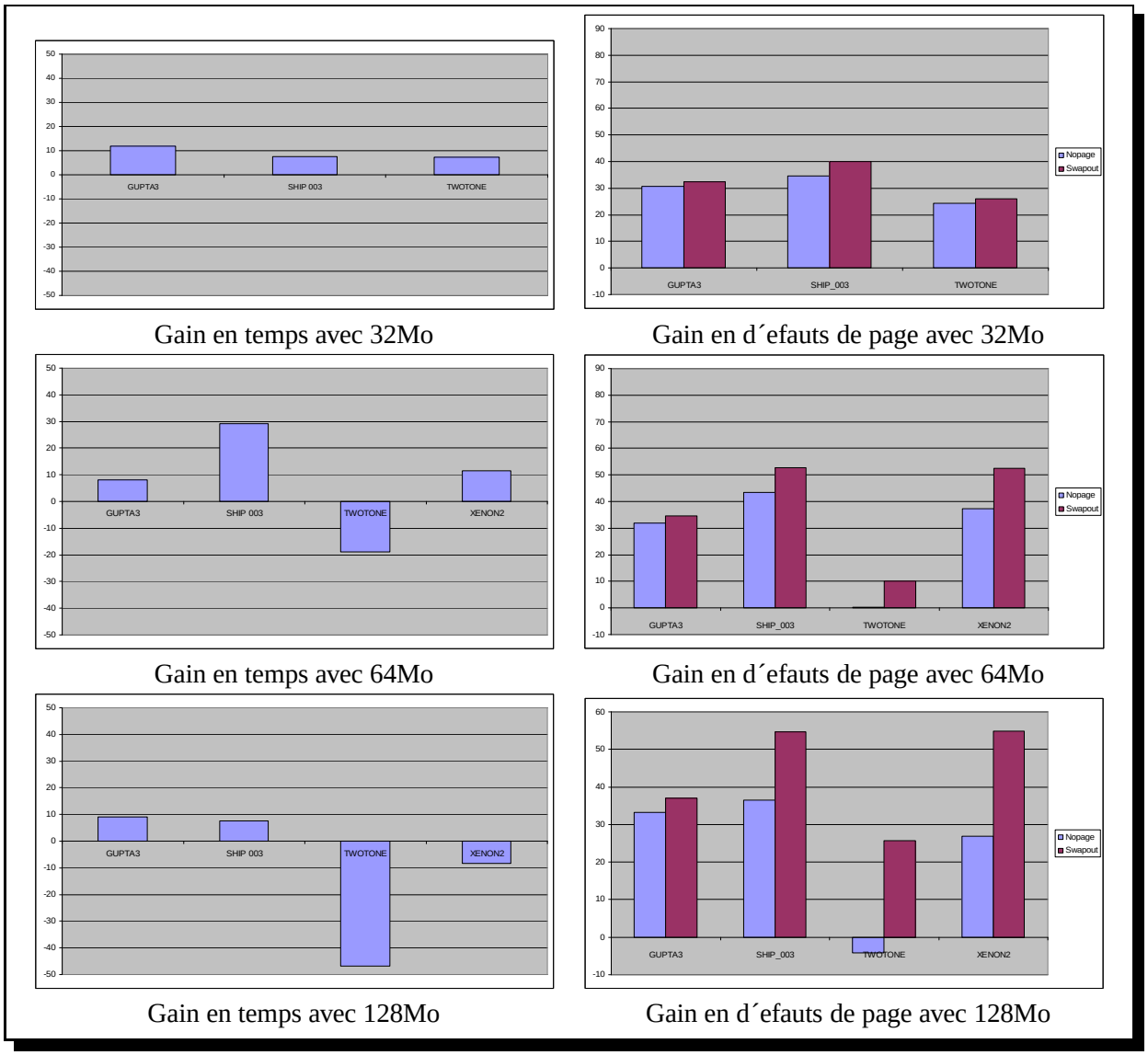



Figure 21: Gain de *MUMPS* s´equentiel (en pourcentage) par l'utilisation du moniteur

Nous pouvons constater (fig. 21) que nous obtenons une r´eduction des ´evincements (swapout) dans tout les cas. C'est principalement dû au m´ecanisme de lib´eration de la m´emoire *monitor_release()* d´ecrit pr´ec´edemment. Concernant le nombre de d´efauts de page, nous observons un gain dans la plupart des cas à l'exception de la matrice TWOTONE avec 64Mo et 128Mo. En effet, comme le moniteur consomme approximativement 5 Mo de m´emoire, les matrices frontales qui ´etaient contenues en m´emoire sans le moniteur, ne peuvent plus l'être. De ce fait, la pr´esence du moniteur augmente consid´erablement l'activit´e de pagination pour de telles matrices frontales.

Habituellement, une r´eduction du nombre de d´efaut de page diminue le temps d'ex´ecution (fig. 20), ce qui n'est pas v´erifi´e sur la matrice XENON2 avec 128Mo. En fait, l'augmentation du temps de factorisation est dû au surcoût du moniteur. En effet, il y a un grand nombre de communication pour des op´erations à grains fins (moins de 1Mo). Comme illustration, nous avons mesur´e 95 000 requêtes faite au moniteur en 600 secondes d'ex´ecution, alors que dans le cas de SHIP_003 avec 128Mo, nous obtenons 35 000 requêtes pour 1 200 secondes.

Finalement, nous observons de meilleurs gains pour 64Mo que pour 32Mo ou 128Mo. Pour 32Mo le nombre de matrices frontales qui ne peuvent être contenu en m´emoire est sup´erieur à 64Mo. En effet, les factorisations repr´esentent la majeure partie du temps d'ex´ecution et il n'y a aucune optimisation de la pagination pendant la factorisation d'une matrice frontale. Pour 128Mo, le ralentissement dû a la pagination est faible, car il y a peu de d´efauts de page et leur diminution n'a que peut d'impact sur les temps d'ex´ecution.

### 4.4.3 MUMPS parallèle

| Matrice | M´emoire | Processeur | Temps (s) avec le moniteur | Temps sans le moniteur |
|---|---|---|---|---|
| SHIP 003 | 64Mo | 4 | 1044 | 1205 |
| BMWCRA 1 | 64Mo | 4 | 335 | 393 |
| XENON2 | 64Mo | 4 | 762 | 801 |
| PRE2 | 64Mo | 4 | 5902 | 6349 |
| ULTRASOUND3 | 64Mo | 4 | 6768 | 8029 |
| SHIP 003 | 128Mo | 4 | 566 | 552 |
| BMWCRA 1 | 128Mo | 4 | 243 | 241 |
| XENON2 | 128Mo | 4 | 561 | 540 |
| PRE2 | 128Mo | 4 | 3098 | 3634 |
| ULTRASOUND3 | 128Mo | 4 | 3657 | 6179 |
| PRE2 | 128Mo | 6 | 2779 | 3172 |
| ULTRASOUND3 | 128Mo | 6 | 3228 | 3307 |

Figure 22: Temps d'ex´ecutions parallèle de *MUMPS* avec et sans le moniteur sur diverses matrices

Nous avons ´egalement fait des tests avec une version parallèle de *MUMPS*. Les figures23.et 22 donnent les temps et les gains en pourcentages d'une ex´ecution sur 4 et 6 nœuds avec 64Mo et 128Mo de m´emoire par nœud.

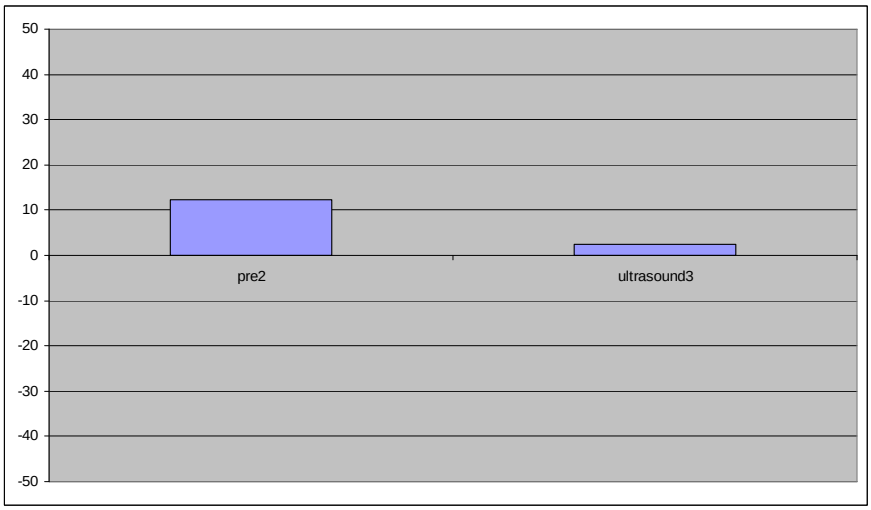
Nous obtenons de l´egères pertes pour les petites matrices (XENON2 et BMWCRA. 1) avec 4 nœuds et 128Mo. Cet effet provient principalement du moniteur. Nous obtenons des gains int´eressant pour les autres problèmes, particulièrement pour le problème ULTRASOUND3 qui expose un gain de 45% avec 4 nœuds de 64Mo. Concernant les r´esultats sur 6 nœuds, nous constatons des gains atteignant 12% du temps d'ex´ecution.

Il est important de remarquer que le moniteur à un impact important sur la distribution des taches de *MUMPS* sur les nœuds, car elle est dynamique. En effet, l'ex´ecution avec et sans le moniteur donne une distribution diff´erente des taches. Afin d'illustrer ce point, nous avons fait l'exp´erience suivantes : nous avons forc´e la strat´egie statique de s´election des esclaves, la distribution des taches d´ecoulant de cette d´ecision dont d´ecoule. Dans ce cas, nous obtenons un gain de 18% avec ULTRASOUND3 sur 6 nœuds avec 64Mo, alors qu'avec la strat´egie fix´ee nous n'avions aucun gain. Nous pouvons ainsi d´eduire qu'il est int´eressant de prendre en consid´eration l'activit´e de pagination dans la distribution des taches.

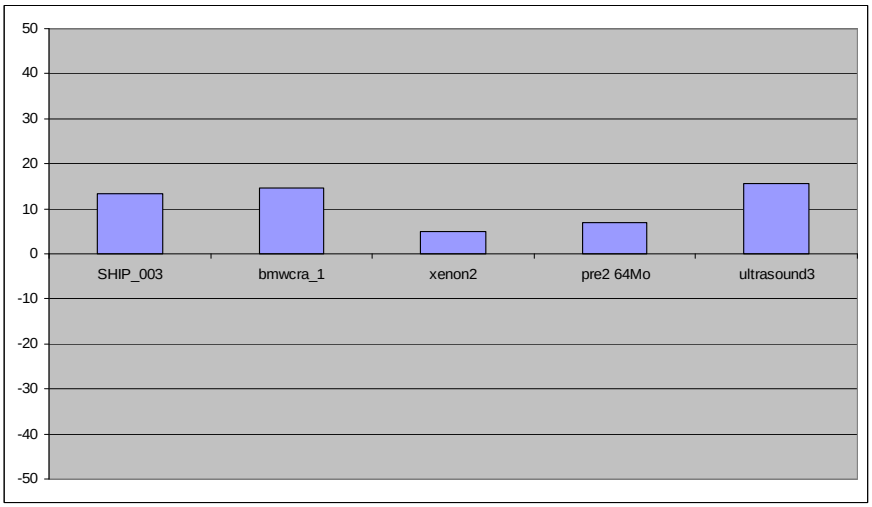
# 5 Conclusion

Nous venons de d´ecrire une am´elioration de MUMM/MMUSSEL qui perturbe moins le système que la pr´ec´edente. L'am´elioration repose sur l'utilisation d'un processus ext´erieur à l'application pour contrôler sa pagination Nous avons montr´e l'int´erêt du contrôle de la pagination pour augmenter la vitesse des copies et nous avons utilis´e *MMUM/MMUSSEL* en coop´eration avec *MUMPS*.

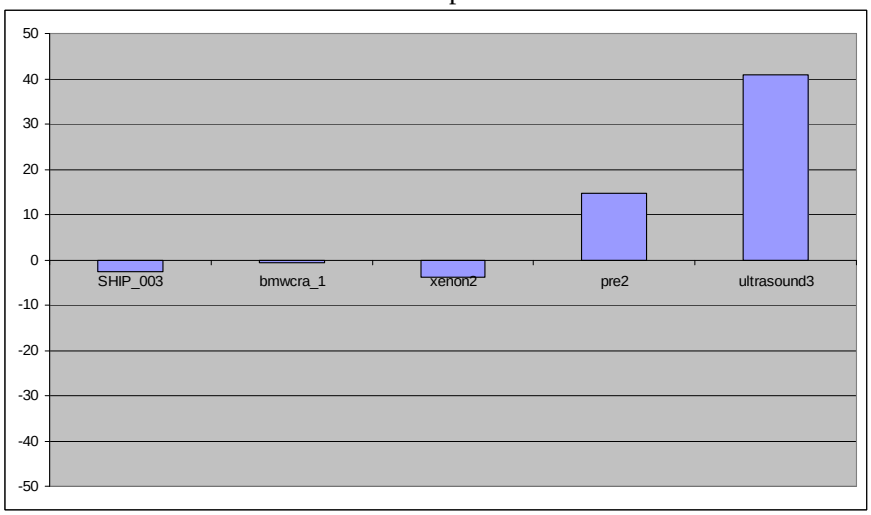
L'interface entre *MMUM/MMUSSEL* et *MUMPS* est standardis´ee et permet de faire coop´erer d'autres programmes avec *MMUM/MMUSSEL*. Ceci avait ´et´e r´ealis´e par Todd Mowry[12] [6], mais ce dernier avait une interface moins riche qui ne permettait que d'indiquer les donn´ees à pr´echarger ou à d´echarger,




128Mo avec 6 processeurs




64MB avec 4 processeurs




128Mo avec 4 processeurs

Figure 23: Gain de *MUMPS* (en pourcentage) par l'utilisation du moniteur avec une grappe

c'est-à-dire l'´etat de la m´emoire voulu à un instant donn´e. L'avantage de notre approche est de donner plus d'information pour la pagination. Ces informations, permettent de ne pas sauvegarder inutilement des pages qui ne seront plus jamais utilis´ees et elles permettent d'orienter l'ordre du pr´echargement par des priorit´es. Nous avons ainsi une meilleure gestion de la m´emoire disponible et des accès au disque.

Un d´eveloppement futur pourrait être l'am´elioration des entr´ees/sorties, car les gains en d´efauts de page et ´evictions ne se traduisent pas directement en gain de temps.

L'interface avec *MUMPS* s'est fait dans le cadre s´equentiel et montre des gains int´eressants. Les premiers tests sur une version parallèle sont encourageants, sur quatre ou six processeurs nous obtenons des gains significatifs, malgr´e un ordonnancement qui ne prend pas en compte la charge m´emoire. Une perspective future sera le perfectionnement du moniteur et le coordonnacement des taches de *MUMPS*.

Actuellement, la version de d´eveloppement de MUMPS intègre les annotations nec´essaires au fonctionnement du moniteur de pagination. Elles devraient prochainement être int´egr´ees dans la version distribu´ee.

## References


[1] V. Abrossimov, M. Rozier, and M. Shapiro. Generic virtual memory management for operating system kernels. In *Proceedings of the 12th ACM Symposium on Operating Systems Principles (SOSP)*, volume 23, pages 123–136, 1989.

[2] Rexe Di Bona Alan Dearle, Anders Lindström, John Rosenberg, and Francis Vaughan. User-level management of persistent data in the grasshopper operating system. Technical report, University of Sidney, 1994.

[3] P. R. Amestoy and I. S. Duff. Memory management issues in sparse multifrontal methods on multiprocessors. *Int. J. of Supercomputer Applics.*, 7:64–82, 1993.

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

[5] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.*, 184:501–520, 2000.

[6] Angela Demke Brown, Todd C. Mowry, and Orran Krieger. Compiler-based I/O prefetching for out-of-core applications. *ACM Transactions on Computer Systems*, 19(2):111–170, 2001.

[7] Eddy Caron, Olivier Cozette, Dominique Lazure, and Gil Utard. Virtual memory management in data parallel applications. In *HPCN'99, High Performance Computing and Networking Europe – Workshop on High Performance Computation on Very Large Data Sets*, volume 1593 of *LNCS*. Springer, 11-13 April 1999.

[8] I. S. Duff and J. K. Reid. The multifrontal solution of indefinite sparse symmetric linear systems. *ACM Transactions on Mathematical Software*, 9:302–325, 1983.

[9] I. S. Duff and J. K. Reid. The multifrontal solution of unsymmetric sets of linear systems. *SIAM Journal on Scientific and Statistical Computing*, 5:633–641, 1984.

[10] Gideon W. Glass. Adaptative page replacement, master thesis. Master's thesis, University of Wisconsin - Madison, 1998.

[11] J. W. H. Liu. The role of elimination trees in sparse factorization. *SIAM Journal on Matrix Analysis and Applications*, 11:134–172, 1990.

[12] Todd C. Mowry, Angela K. Demke, and Orran Krieger. Automatic compiler-inserted I/O prefetching for out-of-core applications. In *Proceedings of the 1996 Symposium on Operating Systems Design and Implementation*, pages 3–17. USENIX Association, October 1996.

[13] Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Baron dan David Black, William Bolosky, and Jonathan Chew. Machine-independent virtual memory management for uniprocessor and multiprocessor architectures. In *Second International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 31–9, 1987.

# Étude de solveurs efficaces pour la simulation de la houle dans les grands bassins

LaRIA/UPJV

Décembre 2003

## 1 Rappel des objectifs du projet

La connaissance de la houle revêt une grande importance pour l'ingénierie maritime. Elle intervient en premier lieu dans le dimensionnement des ouvrages maritimes. Elle s'avère aussi un obstacle majeur à la navigation et aux opérations portuaires. Elle influe enfin largement sur les mouvements sédimentaires et l'évolution du trait de côte.

La connaissance de la houle dans les domaines côtiers, portuaires ou estuaires passe avant tout par la construction d'outils de modélisation opérationnels. L'outil informatique s'avère à cet égard, d'une grande puissance et d'une grande souplesse d'utilisation. Il permet de tester rapidement différents projets d'aménagement soumis à une large gamme de sollicitations de houle et de conditions de marée.

Le code de houle REFONDE est un code d'agitation de houle qui résout l'équation de réfraction-diffraction de Berkhoff par une méthode de calcul d'éléments finis. Il peut prendre en compte des ouvrages réfléchissants. Il a été développé au sein du CETMEF.

Dans une modélisation par éléments finis, on se ramène généralement à la résolution d'un système linéaire généralement creux. Certains problèmes traités par le CETMEF (par exemple l'aménagement du port du Havre), peuvent induire la résolution de systèmes creux de plusieurs dizaines de Gigaoctets. Malheureusement, dans certains cas, les systèmes à résoudre sont extrêmement coûteux en terme de temps de calcul et/ou d'espace mémoire. Le solveur employé jusqu'à présent ne permet pas de traiter de tels problèmes.

L'objet de ce projet est d'étudier d'autres types de solveurs directes pour l'application REFONDE.

L'objectif est d'étudier les différentes méthodes pour améliorer les performances et les capacités de traitement du solveur de REFONDE. En particu-

lier :

- exploiter la hiérarchie mémoire des machines ;
- employer le parallélisme ;
- employer les techniques de calcul `out-of-core`.

Le résultat attendu étant de trouver une combinaison de ces différentes méthodes permettant d'obtenir un solveur directe efficace. Ce type de solveur devrait pouvoir s'exécuter sur une architecture de type grappe, i.e. qu'il soit capable d'exploiter un parc de machines existantes sans investissements supplémentaires.

# 2 Résultats

Notre travail s'est déroulé en trois temps : une analyse préliminaire des systèmes à résoudre permettant de déterminer la méthode de résolution la plus adaptée, puis la recherche et l'intégration d'un outil de résolution basé sur cette méthode, enfin la dérivation d'un prototype out-of-core.

## 2.1 Analyse préliminaire

Dans un premier temps, nous avons étudié la structure des systèmes à résoudre dans Refonde à partir des matrices exemples fournies par le CETMEF. La caractéristique principale de ces systèmes est qu'ils sont composés de matrices creuses bandes irrégulières. Nous avons cherché quelles pouvaient être les méthodes les plus efficaces pour résoudre de tels systèmes. Classiquement, la résolution de matrices bandes est considérée comme peu parallélisable. La méthode usuelle pour résoudre les matrices bandes est l'emploi de solveurs frontaux. Pour améliorer les performance, du parallélisme peut être introduit par des décompositions de type *Single Width Separator*.

Cependant, une étude plus fine des matrices considérées a montré une irrégularité très forte. Il est connu que l'ordre dans lequel un système creux est résolu peut réduire significativement le nombre de calculs et les besoins en mémoire. Nous avons donc étudié les différentes techniques d'ordonnancement des calculs (*ordering*) et leur influence sur les performances pour les systèmes considérés. Lors de cette analyse, nous avons constaté que l'irrégularité des matrices considérées exhibe un fort degré de parallélisme à grain fin.

Il s'avère que la méthode de résolution directe la plus adaptée pour les matrices considérées semble être la méthode multifrontale. Cette méthode a deux avantages immédiats :

– elle exploite au mieux la hiérarchie mémoire par un découpage des calculs en blocs permettant d'employer des primitives de calcul de type BLAS3 ;
– elle se parallélise naturellement.

## 2.2 Intégration de MUMPS dans Refonde

Nous avons cherché s'il n'existait pas un solveur multifrontal opérationnel. Nous avons choisi MUMPS (*MUltifrontal Massively Parallel Solver*) qui a été développé principalement par le CERFACS et l'INRIA dans le cadre d'un programme européen. Ce logiciel est actuellement maintenu au LIP à l'ENS Lyon.

Dans un premier prototype, nous avons modifié le code initial de Refonde afin de pouvoir intégrer MUMPS. Les modifications ont consisté à :
– changer les structures de données internes de Refonde afin de réduire l'occupation mémoire et permettre l'interopérabilité avec MUMPS ;
– intégrer le support d'exécution parallèle MPI dans Refonde.

Nous avons testé et validé ce premier prototype en séquentiel et en parallèle.

MUMPS est proposé avec un *ordering* générique (AMD) qui permet d'obtenir de bonnes performances dans le cas général. Lorsque AMD n'est pas adapté, MUMPS permet d'employer des *ordering*s externes. Actuellement, nous cherchons l'ordering le mieux adapté : par exemple l'*ordering* hybride METIS donne de meilleurs résultats que AMD.

La description de l'implémentation ainsi que les premiers résultats se trouvent dans le rapport de stage d'Olivier Soyez joint en annexe.

## 2.3 Traitement de grands problèmes

En terme de capacité de traitement, i.e. la taille maximale des problèmes qui peuvent être résolus, l'intégration d'un nouveau format de stockage des systèmes et l'utilisation d'*ordering*s adéquats, permettent d'envisager des problèmes qui étaient hors de porté de la version initiale de Refonde. Le parallélisme permet de repousser encore plus loin les limites.

Dans le cas de traitements séquentielles, nous avons donc aussi abordé l'emploi de techniques out-of-core. Le problème était de savoir dans qu'elle mesure il était possible de dériver une version out-of-core explicite (i.e. avec des instructions d'entrée sorties). Cependant MUMPS est un gros code (120000 lignes de Fortran 90) et il ne paraissait donc pas raisonnable de faire une nouvelle version out-of-core car cela nécessiterait de maintenir deux versions du code (in-core & out-of-core).

Nous avons donc choisi une autre voie qui consiste à modifier la pagination de MUMPS grâce aux outils de contrôle de la pagination développé au LaRIA dans la thèse de M. Olivier Cozette. De concert avec les développeurs de MUMPS, nous y avons donc intégré des directives de pagination et nous avons développé un moniteur de pagination dédié. Nous obtenons des améliorations notables des performances. La versions de développement de MUMPS à l'INRIA intégre les modifications que nous avons effectuées, elles devraient être disponibles dans la version diffusée prochainement.

Une description du prototype out-of-core se trouve dans le second rapport joint en annexe.

# 3 Conclusion

Grâce à une analyse des systèmes à résoudre, nous avons choisi la méthode de résolution directe qui semble la plus adaptée à Refonde. Nous avons intégré un solveur multifrontal générique (MUMPS) dans un nouveau prototype de REFONDE : ce nouveau prototype est opérationnel et permet déjà d'obtenir de bonnes performances. De plus il peut s'exécuter en parallèle. D'autre part, nous avons aussi dérivé un prototype de version out-of-core de MUMPS qui peut s'intégrer dans Refonde à partir du code précédent.

La version parallèle et la version out-of-core de MUMPS que nous avons intégré dans Refonde devrait permettre d'aborder les problèmes de grande taille sans difficulté. Le prototype de Refonde peut donc être déployé sur le site du CETMEF.