Mise en œuvre de la méthode multifrontale MULT_FRONT

Résumé :

On présente dans cette note, la description informatique de la version native dans Code_Aster, du solveur linéaire multi-frontal, appelé communément MULT_FRONT.

Cette version, développée à EDF R&D est stabilisée, avec un périmètre quasi-complet.

Cette note complète la documentation de référence de Code_Aster: « Solveur linéaire par la méthode multi-frontale » [R6.02.02].

On rappelle les principales notions utilisées par la méthode : renumérotation, arbre d'élimination, super-nœuds. On décrit les structures informatiques créées par l'algorithme. On signale les principales difficultés et particularités informatiques. On envisage les différents développements possibles.

Un compte-rendu publié à part est dédié à un état des lieux et aux perspectives de maintenance applicative : [RB12] « Méthode multi-frontale native dans Code_Aster : état des lieux et perspectives », CR-I23-2012-001.

Table des matières

1 Description de la méthode....3
1.1 Méthode LDLT pour les matrices pleines....3
1.2 Matrice creuse et remplissage....6
1.3 Méthode multifrontale....7
1.4 Descente - Remontée....13
2 Implantation et utilisation dans Code_Aster....14
3 Bibliographie....15
4 Description des versions du document....15
Annexe 1 Documentation de référence de la méthode « Approximate Minimum degree »....17
Annexe 2 Documentation de référence METIS....17

1 Introduction

On décrit dans cette note l'implémentation informatique du solveur multifrontal, dans C*ode_Aster*. Il est destiné aux développeurs du code et n'est pas un document « autoporteur ».

Cette méthode a été développée sous la forme de prototype au sein du groupe ISA en 1993 [RO93], et introduite ensuite dans *Code_Aster* en 1994, dans une forme adaptée à ce code :
•Factorisation d'une matrice non nécessairement en mémoire centrale (« out of core »), basée sur le progiciel JEVEUX.
•Traitement des conditions aux limites avec multiplicateurs de Lagrange.
•On supposait en outre ne pas avoir besoin d'activer une recherche de pivot.

Dans un premier temps, seule une version séquentielle pour matrice réelle symétrique a été implémentée dans *Code_Aster*.

Par la suite les versions non-symétriques et complexes ont été introduites.

Le parallélisme complet a été développé dans le prototype [RO95], il n'a pas été retenu pour la version développée dans Code_aster. Les raisons en sont les suivantes :
•L'objectif prioritaire à l'époque était d'optimiser l'occupation en mémoire afin de traiter des systèmes linéaires de grande taille. Or le parallélisme nécessite plus de mémoire que le séquentiel.
•Le parallélisme de la multi-frontale nécessite une gestion de la mémoire parallèle propre à cette méthode, qui aurait dû cohabiter avec le progiciel JEVEUX, intrinsèquement séquentiel.
•Le parallélisme n'était pas non plus une priorité à cette époque où l'on ne disposait que de calculateurs faiblement parallèles.

On trouvera une base théorique sur la méthode dans [DU86], [AS87], [RO93] & [RO95], ainsi que dans la documentation de référence de *Code_Aster* : « Solveur linéaire par la méthode Multi-Frontale », [R6.02.02].

La version double précision REAL*8 (resp. COMPLEX*16) est réalisée par l'appel à la routine FORTRAN MULFR8 (resp. MLFC16).

Par la suite on décrira en détail la version pour matrice symétrique de variable réelle, des paragraphes seront consacrés aux versions non symétrique et complexe. Pour plus de détails sur les structures de données de C*ode_Aster*, on se référera aux documentations [D4.06.07] (`sd_ nume_ddl`) et [D4.06.10] (`sd_matr_asse`).

1.1 Principes généraux.

On veut remplacer la factorisation d'une matrice creuse, par une suite de factorisation de matrices pleines. C'est le principe cartésien : remplacer un problème difficile par un ensemble de problèmes simples.

Pour cela on dispose d'abord d'un algorithme de renumérotation des variables du système. Cet algorithme (de degré minimum, par exemple), permet de minimiser l'ordre des matrices pleines à factoriser, c'est le premier axe d'efficacité de la méthode.

En outre, de cette renumérotation, on peut extraire les deux produits suivants :
•Un arbre d'élimination, obtenu à partir du « creux » (sparsity) de la matrice initiale et du remplissage optimum issu de la renumérotation. Ce graphe (en forme d'arborescence) fournit l'ordre d'élimination des variables et induit le parallélisme. (En effet un nœud « fils » doit être éliminé avant un nœud « père », mais tous les « fils » d'un même « père » peuvent être éliminés indépendamment, c'est à dire en parallèle.) Ce parallélisme est le deuxième axe d'efficacité de la méthode.
•La notion de super-nœud. Cette notion a été conçue, initialement, pour minimiser le coût (non négligeable) de la renumérotation. En simplifiant, on appelle super-nœud, un ensemble de nœuds (au sens de la théorie des graphes), ayant les mêmes voisins. L'arborescence citée ci-dessus, est en fait formée de super-nœuds dont la taille augmente quand on monte dans l'arborescence, cette augmentation est une illustration du remplissage dû à la méthode d'élimination de Gauss. Ce regroupement a l'intérêt suivant : on élimine les variables par groupe, et non une par une, permettant ainsi l'utilisation de méthodes par bloc efficaces (à l'aide de BLAS de niveau 2 ou 3, produits matrice*vecteur ou matrice*matrice). Ce travail par bloc est le troisième axe d'efficacité.

En résumé, les 3 lignes de force de la méthode sont :
•Une renumérotation,
•Un arbre d'élimination,
•Un regroupement des variables.

La factorisation numérique se déroule alors comme suit :

On parcourt l'arbre d'élimination, à chaque étape (élimination du super-nœud), on dispose d'une matrice pleine associée au super-nœud et à ses voisins, on effectue l'élimination de Gauss limitée aux variables du super-nœud. On obtient ainsi d'une part les colonnes de la factorisée finale (variables du super-nœud), et une matrice frontale modifiée et réduite aux voisins (« complément de Schur »), qui sera assemblée dans la matrice frontale des variables du super-nœud « père ».

1.2 Programmation de la méthode.

En règle générale, la programmation de la méthode multi-frontale est divisée en 2 phases :
•une phase de factorisation dite « symbolique », incluant la renumérotation,
•une phase de factorisation numérique proprement dite.

Dans *Code_Aster*, la renumérotation et la première phase sont réalisées par l'appel à la subroutine MLTPRE (appelé via MLFC16 et MULFR8, dans les routines « chapeau »TLDLG2 et TLDLG3).

La deuxième phase est réalisée via l'appel à ces routines MLFC16 et MULFR8, dans les cas complexe ou réels.

2 Renumérotation et factorisation symbolique : MLTPRE

Par commodité, on a omis le préfixe NU, devant les noms de structure.

Structures de données lues.
De la structure de données : NUME_DDL, on extrait les enregistrements :

NUME.DELG : indique si le degré de liberté est de « Lagrange » ou non.
NUME.DEEQ ,
NUME.NUEQ,
NUME.PRNO : Ces 3 enregistrements établissent les liens entre les degrés de liberté soumis à une condition aux limites et les Lagranges correspondant.
Les 3 enregistrements suivants décrivent une matrice « Morse ».
SMOS.SMHC : Numéro de colonne du terme
SMOS.SMDI : Adresses des termes diagonaux
SMOS.SMDE. : Nombres d'inconnues, et de termes de la matrice

Structures de données écrites.
(En face de chaque structure *Code_Aster*, on a écrit le nom du tableau FORTRAN correspondant, qui sera utilisé, par la suite pour plus de commodités.)
MLTF.ADNT ADINIT : adresses des termes initiaux dans la matrice factorisée
MLTF.LGBL LGBLOC : longueur des blocs de la matrice factorisée
MLTF.LGSN LGSN : longueur des super-nœuds
MLTF.LOCL LOCAL : utilisé dans l'assemblage des matrices frontales, correspondance entre les numéros locaux du super-nœuds père et fils.
MLTF.ADRE ADRESS
MLTF.SUPN SUPND : définition des super-nœuds
MLTF.FILS FILS : fils des super-nœuds dans l'arbre d'élimination
MLTF.FRER FRERE : frère des super-nœuds dans l'arbre d'élimination
MLTF.LGSN LGSN : longueur des super-nœuds
MLTF.LFRN LFRONT : ordre des matrices frontales (après élimination)
MLTF.NBAS NBASS : pointeur lié à l'assemblage des matrices frontales

MLTF.ADPI ADPILE : adresses dans la pile des matrices frontales
MLTF.ANCI ANC : tableau inverse de la renumérotation
MLTF.NBLI NBLIGN : ordre des matrices frontales (avant élimination)
MLTF.NCBL NCBLOC : nombre de degrés de liberté de chaque bloc
MLTF.DECA DECAL : décalage entre le début de colonne d 'un super-nœud et le début du bloc JEVEUX qui le contient
MLTF.SEQU SEQ : ordre d'élimination des super-nœuds (parcours de l'arborescence).
Ces structures de données seront décrites dans les paragraphes suivants.

Description du code.
On distingue 2 phases, la première consiste en la renumérotation, la seconde en ce que l'on appelle dans le langage de la méthode multi-frontale, la factorisation symbolique.

La structure de données NUME_DDL décrit une numérotation des NEQ degrés de liberté, comprenant, la plupart du temps, des degrés de liberté dits de Lagrange. Ces degrés de liberté doivent satisfaire la relation d'ordre suivante :
(1) $L1<U<L2$, si $L1$ et $L2$ sont les « Lagranges » associés à la condition aux limites sur U .

Afin de conserver cet ordre, on va soumettre à l'algorithme de renumérotation uniquement les degrés de liberté non « Lagranges ». Les degrés de liberté « Lagranges sont donc ôtés et les informations les concernant sont stockées afin de les réintégrer dans la nouvelle numérotation, tout en satisfaisant (1).

Routines appelées (un première description succincte du code).

Pour la première phase, les routines appelées sont les suivantes :
PREML0 : extraction des informations sur les « Lagranges »
PREMLA : traitement pour les « Lagranges » de type relation linéaire
PREML1 : pré traitement et renumérotation.
PREMLC : post-traitement de la renumérotation (ajout des « Lagranges »).
PREMLD : idem

Pour la seconde :
PREML2 : factorisation symbolique.

2.1 PREML0

On stocke dans les tableaux $LBD1$, $LBD2$, les «Lagranges» de blocage, et dans RL les «Lagranges» de relation linéaire.

On obtient une nouvelle numérotation de 1 à $N2$ des degrés de liberté sans Lagrange, définie par :
P de $[1:\mathrm{NEQ}]\Rightarrow[1-N2]$, et Q Inverse de P .

Pour I de 1 à NEQ , si I est un degré de liberté bloqué, $LBD1(I)$, (resp. $LBD2(I)$), est le numéro de son L1 (resp. $L2$).

Dans le cas de NRL relations linéaires, on aura :
$RL(1,I)$: $L1$ de la relation I ,
$RL(2,I)$: $L2$ de la relation I .

N.B. Programmation
Dans certaines routines, on utilisera le nom de variable $N1$ en place de NEQ .

2.2 PREMLA

Pour tous les degrés de liberté $L2$ de relation linéaire, (second « Lagrange »), ses voisins de numéro inférieur sont connus en extrayant l'information de la structure de données SMOS.SMHC (tableau COL).

A partir de ces informations, PREMLA crée des listes chaînées (tableaux DEB, VOIS, SUIT) pour les degrés de liberté de relation linéaire $L2$.
Soit J un degré de liberté inclus dans une relation linéaire, $j0=DEB(j)$ est l'adresse du premier voisin de J dans le tableau $VOIS$, $J1=SUIV(J0)$ est l'adresse du voisin suivant, $J2=SUIV(J1)$ est l'adresse du suivant et ainsi de suite.

Cette liste chaînée sera utilisée par la suite dans PREML1, pour la fabrication de $ADJNCY$.

N.B. Allocation de mémoire
La liste chaînée consiste en 2 structures locales, de noms NOMVOI et NOMSUI, (créées par WKVECT et détruites en fin de MLTPRE).
Leur longueur : $LGLIST$, est estimée a priori par PREML0 (paramètre de sortie LT), s'il est insuffisant, PREMLA fournit un code d'erreur, et est appelé de nouveau avec des structures de longueur 2 fois plus grandes.

2.3 PREML1

PREML1 appelle les routines suivantes :
PRMADJ, GENMMD, AMDBAR, ONMETL.

Le rôle essentiel de PREML1 est de lancer la renumérotation. Celle-ci, effectuée au choix par: METIS (méthode de dissection), GENMMD (minimum degree) ou AMDBAR (minimum degree amélioré), nécessite 2 actions préalables.

2.3.1 Reconstitution des nœuds (au sens de la discrétisation par éléments finis).

Dans une première version du code, l'algorithme de renumérotation travaillait sur les $N2$ degrés de liberté (sans Lagranges).

On a voulu ensuite préserver l'ordre interne des inconnues au sein des nœuds de discrétisation élément finis (par exemple ux, uy, uz, p).

Un développement a été effectué à cet effet. **On soumet désormais à l'algorithme de renumérotation l'ensemble des nœuds et non les $N2$ degrés de liberté**.

On forme donc un nouvel ensemble de $NBND$ nœuds et 2 pointeurs associés :
$NŒUD[1:N2]\Rightarrow[1:NBND]$, et le pointeur inverse $DDL[1:NBND]$

On a ainsi 3 numérotations :
•Numérotation initiale de 1 à NEQ,
•Celle des degrés de liberté sans Lagranges 1 à $N2$,
•Celle des nœuds d'interpolation 1 à $NBND$.

Munies des tableaux P et Q entre la première et le deuxième, $NŒUD$ et DDL entre la deuxième et la troisième.

2.3.2 Création de la structure ($ADJNCY$, $XADJ$)

Cette structure est la structure standard de données des algorithmes de renumérotation cités plus haut. On a 2 tableaux $ADJNCY$ et $XADJ$ définis comme suit :

Soit un nœud I, ses voisins se trouvent stockés dans le tableau $ADJNCY$, entre les adresses $XADJ(I)+1$ et $XADJ(I+1)$.

On crée d'abord une structure $(ADJNCY, XADJ)$ relative aux $N2$ degrés de liberté, puis la routine PRMADJ la transforme en une structure $(ADJNCY, XADJD)$ relative aux $NBND$ nœuds.

Pour créer $ADJNCY$, on utilise la liste chaînée $(DEB, VOIS, SUIT)$ créée par PREMLA, en forçant les degrés de liberté d'une relation linéaire à être voisins. Ainsi ces degrés de liberté seront dans le même super nœud, de même que leur $L2$. La relation d'ordre sera ainsi respectée.

On effectue ensuite l'appel à GENMMD, AMDBAR ou METIS. On obtient les résultats habituels de cet algorithme :

1) Renumérotation,
2) Définition des super-nœuds,
3) Arborescence

Les sources de ces 3 méthodes sont dans le domaine public.
Concernant les 2 derniers points, il a fallu les adapter. La renumérotation concerne les nœuds (1 à $NBND$), elle a la forme de 2 tableaux inverses l'un de l'autre: $INVPND$, $PERMND$. ($INVPND(I)$ est le nouveau numéro du nœud I).

L'arborescence, quant à elle, porte sur les super-nœuds créés par la renumérotation. Le super-nœud SND est défini comme suit : il est constitué du segment de nœuds $[SPNDND(SND-1)+1, SPNDND(SND)]$. $SPNDND$ est ainsi une injection du segment $[1:NBSND]$ vers $[1:NBND]$, $NBSND$: nombre de super-nœuds.

L'arborescence est définie comme suit : $PARENT(SND)$ est le nœud « père » du super-nœud SND.

N.B. Programmation
L'arborescence des super-nœuds de la multi-frontale, apparaît sous 2 noms : $PARENT$ et $PAREND$
Le premier contient l'arborescence des SN hors Lagrange, il est fourni par la renumérotation (GENMMD/AMDBAR/METIS.)
Le second est le complété du premier en y ajoutant les Lagranges, il est calculé par PREMLC à partir du premier.

Dans le programme appelant MLTPRE, $PARENT$, qui est un intermédiaire, est stocké dans ZI à l'adresse SEQ, réutilisé plus tard (gain de place).

Ces tableaux précédents sont définis en fonctions des $NBND$ nœuds de discrétisation de *Code_Aster*, (1 à $NBND$). Ils sont reconstruits ensuite dans la numérotation des $N2$ degrés de liberté non « Lagrange ».
$INVPND$ devient ainsi $INVP$, $PERMND$ devient $PERM$ et $SPNDND$ devient $SUPND$. (Cf. boucles 400 405 406 407 de PREML1).

N.B. Allocation de mémoire

La structure $(XADJ, ADJNCY)$ de PREML1, est appelée $(XADJ2, ADJNC2)$ dans MLTPRE. Elle est créée avant PREML1 et détruite après. La longueur de $ADJNC2$, $LGADJN$, est estimée à $2\times(lmat-\mathrm{neq}+lglist)$, avec $lglist$ = longueur de la liste des voisins utilisée dans PREML0
$lmat$ = longueur de la matrice initiale (fournie par SMOS).
Si $LGADJN$ est insuffisant, PREML1 fournit la longueur nécessaire en code retour, et est relancé.

NB. METIS

Auparavant, on produisait un exécutable du logiciel METIS, écrit en C, compilé et linké séparément. Cet exécutable était appelé à travers APLEXT, couche d'appel d'un exécutable externe par Code_Aster. Désormais les routines de METIS sont compilées et appelées par un appel à ONMETL, fonction C appelante.

2.4 PREMLC

En sortie de PREML1, les résultats sont exprimés en terme de degrés de liberté non « Lagrange » (1 à $N2$), la routine PREMLC complète la numérotation de 1 à $N2$ en ajoutant les «Lagrange» et en créant de nouveaux super-nœuds.

Les «Lagrange» sont inclus ainsi dans la nouvelle numérotation :

Pour chaque $L1$, on crée un nouveau super-nœud SN , constitué d'un seul degré de liberté ($L1$), et fils du premier degrés de liberté dont il est le « Lagrange » (bloqué ou relation linéaire). En étant le fils de ce degré de liberté, il sera éliminé auparavant, ce qui est la règle.

Pour les $L2$, on ne crée pas de nouveau SN , on se contente d'ajouter un degré de liberté supplémentaire ($L2$) au super-nœud contenant le degré de liberté (bloqué ou R.L.) dont il est le « Lagrange ».

En cas de relation linéaire, tous les degrés de liberté de la relation sont mis par l'algorithme dans le même super-nœud. Cela est réalisé dans le paragraphe précédent (PREML1), lors de la fabrication de la structure $ADJNCY$, en forçant les degrés de liberté des relations linéaires à être voisins entre eux, au moyen de la liste chaînée (DEB , $VOIS$, $SUIV$).

On obtient les nouvelles renumérotations $NOUV$ et ANC sur $[1:\mathrm{NEQ}]$, pour tous les degrés de liberté dans la nouvelle numérotation, $ANC(I)$ fournira son numéro dans la numérotation initiale. $NOUV$ est le tableau inverse.

$NBSN$ le nouveau nombre de super-nœuds et $SUPND[1:nbsnd+1]$ définit ces SN .
Le super-nœud SN est constitué du segment de nœuds $[SUPND(SN-1)+1, SUPND(SN)]$.
$SUPND$ est ainsi une application du segment $[1:NBSN]$ vers $[1:\mathrm{NEQ}]$.

N.B. Allocation de mémoire

$LGIND$ est la longueur estimée des tableaux $GLOBAL$ ET $LOCAL$, (Cf. plus loin 2.6) qui sont alloués avant PREML2 ($NOPGLO$ et $NOPLOC$)
$LGIND$ est d'abord un résultat de la renumérotation, fourni par PREML1 et augmenté par PREMLC en fonction des conditions aux limites.

$LGIND$ est une surestimation de $GLOBAL$ et $LOCAL$ on ne veut pas conserver une structure trop importante.

Ainsi, après PREML2, on connaît la longueur réelle de ces tableaux, on redéfinit $LGIND$ à cette valeur, et on recopie $GLOBAL$ et $LOCAL$ dans des structures $NOMGLO$ et $NOMLOC$ de longueur exacte.

2.5 PREMLD

C'est une étape technique : cette routine fabrique une structure de type $ADJNCY$ comme celles utilisées par les algorithmes de renumérotation, cette fois-ci tous les degrés de liberté de 1 à NEQ sont pris en compte.

N.B. Allocation de mémoire

La structure $(XADJ, ADJNCY)$ fabriquée par PREMLD, est appelée $(XADJ1, ADJNC1)$ dans MLTPRE.
On utilise la même longueur $LGADJN$ calculée précédemment.

2.6 PREML2

C'est la seconde phase de MLTPRE : on effectue entre autres, la factorisation symbolique qui est une simulation de la factorisation réelle, sans opération flottante, destinée à faciliter le travail de la factorisation réelle, en fabriquant certains pointeurs nécessaires.

PREML2 appelle les routines suivantes :

- FACSMB: factorisation symbolique proprement dite.
- MLTPOS : fabrication de la séquence de factorisation, c'est à dire, remontée de l'arborescence définie par le tableau $PAREND$, qui définit ainsi l'ordre d'élimination des super-nœuds.

- MLTBLC : On définit ici le découpage en blocs de la matrice factorisée, virtuellement « out of core », car découpée en blocs libérables par le progiciel JEVEUX.
- MLTPAS : On calcule ici l'adresse des coefficients initiaux dans la matrice factorisée.

2.6.1 FACSMB

Données (Elles sont issues de la renumérotation) :
SUPND
PAREND ,
(Et de la topologie initiale) :
ADJNCY .

Résultats :

GLOBAL
LOCAL
ADRESS
NBASS
DEBFAC
DEBFSN
LGSN
LFRON
FILS
FRERE .

Ces tableaux sont décrits ci-dessous.

FILS et *FRERE* sont les tableaux « inverse » du tableau *PAREND* , ils permettent de connaître tous les super-nœuds « fils » d'un super-nœud donné *SN* , ce sont $FILS(SN)$, $FRERE(FILS(SN))$, et ainsi de suite. Pour les autres résultats, il est nécessaire de revenir sur les principes de base de la méthode multi-frontale, au moyen des figures suivantes :

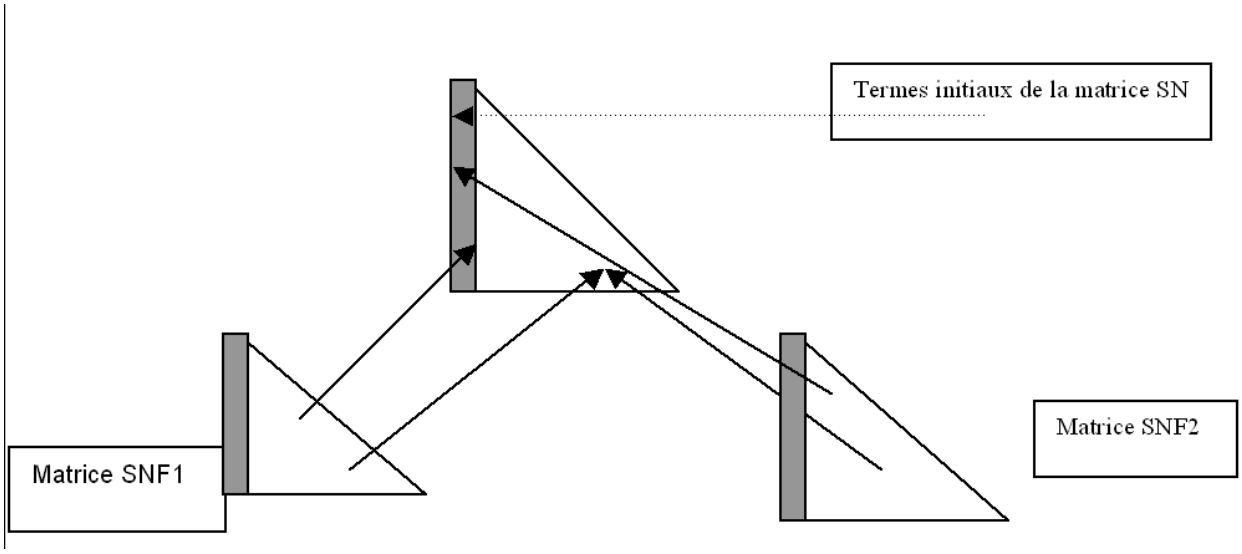


Figure 1 Assemblage de 2 matrices filles dans une matrice mère

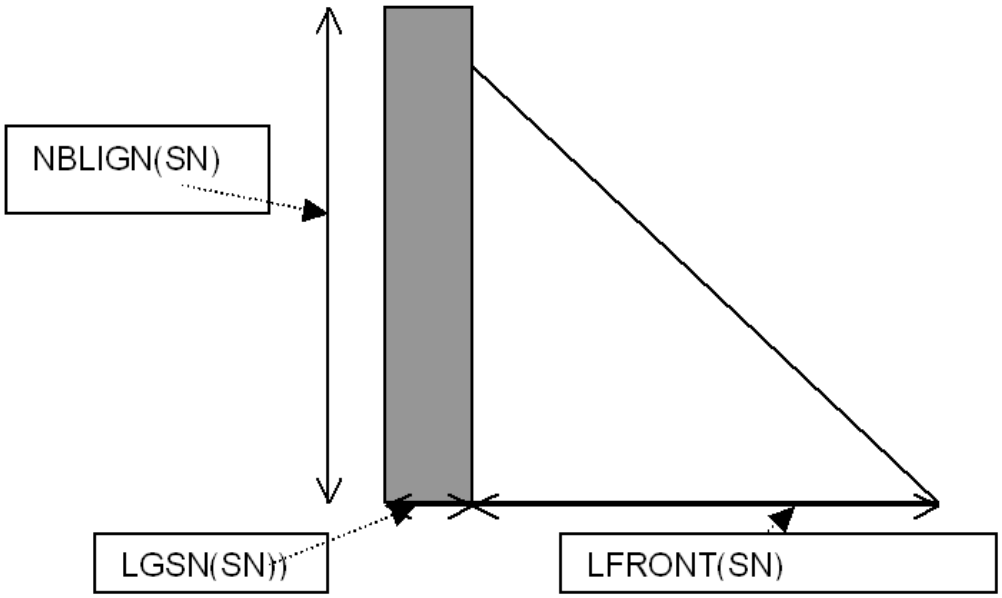


Figure 2 Matrice frontale SN

On voit sur la Figure 1, la matrice frontale SN (du super-nœud SN), constituée des termes provenant de la matrice initiale, ainsi que des termes provenant des matrices filles $SNF1$ et SNF2, après élimination des super-nœuds $SNF1$ et $SNF2$. Après élimination, la partie grisée de la matrice frontale va constituer les colonnes de la matrice factorisée, et la partie blanche restante rangée dans la pile des matrices frontales, ira s'agglomérer à la matrice mère lors de l'assemblage de cette dernière.

La factorisation symbolique effectuée par FACSMB va simuler ce processus d'assemblage/élimination. Pour chaque super-nœud SN, on va fabriquer une liste chaînée contenant les numéros d'inconnues du super nœud lui-même, ainsi que les numéros d'inconnues des voisins de tous les fils de SN. On inclut les voisins des fils, mais non les fils eux-mêmes qui ont été éliminés.

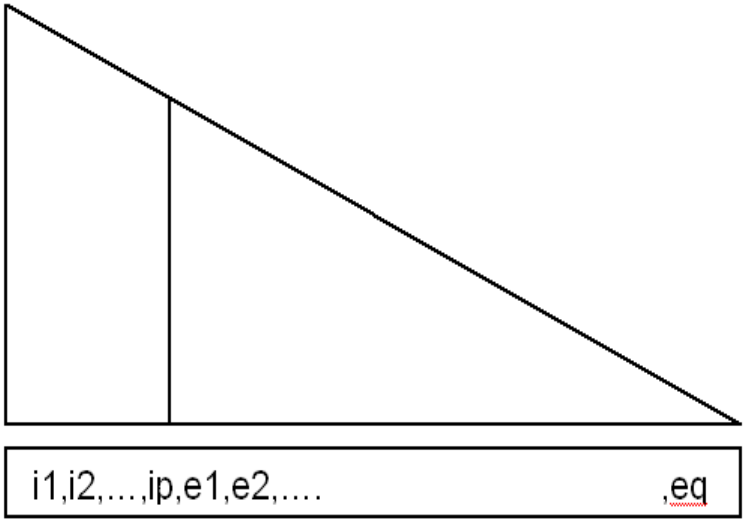
En langage FORTRAN, FACSMB produit les tableaux suivants :

$LGSN(SN)$: nombre d'inconnues du super-nœud SN (en fait = $supnd(sn+1)-supnd(sn)$),
$NBLIGN(SN)$: nombre d'inconnues total de la matrice frontale SN
$LFRONT(SN)=NBLIGN(SN)-LGSN(SN)$, ordre de la matrice frontale après élimination qui sera stockée dans la pile.

FACSMB fabrique aussi un tableau $GLOBAL$, muni d'un tableau de pointeurs $ADRESS$.
$GLOBAL$ contient les numéros des inconnues de la matrice frontale SN (avant élimination), entre les adresses $ADRESS(SN)$ et $ADRESS(SN+1)-1$.

Ces inconnues sont au nombre de $NBLIGN(SN)=(ADRESS(SN+1)-ADRESS(SN))$.
On accède au tableau $LOCAL$ comme $GLOBAL$, au moyen du tableau $ADRESS$. Soit $FSN1$ le super-nœud « fils » du super-nœud SN, comme sur la figure ci-dessus, et soit I un indice compris entre $ADRESS(FSN1)+LGSN(FSN1)$ et $ADRESS(FSN1+1)$, $LOCAL(I)$ sera le numéro local (compris entre 1 et $NBLIGN(SN)$) de cette inconnue dans la matrice du super-nœud « père » SN (on a $PAREND(FSN1)=SN$).

$LOCAL$ sera utilisé dans l'assemblage de la matrice frontale « fille » dans la matrice frontale « mère », il donne directement l'adresse de destination. On illustre ceci à l'aide de l'exemple suivant :



La matrice frontale SN a $(p+q)$ inconnues : $i1, i2, .., ip, e1, e2, ..., eq$, avec

$NBLIGN(SN)=p+q$
$LGSN(SN)=p$
$LFRONT(SN)=q$
$ADRESS(SN+1)-ADRESS(SN)=p+q$,
$GLOBAL=[i1, i2, ...ip, e1, e2, ...eq]$, aux adresses variant entre $ADRESS(SN)+1$ et $ADRESS(SN+1)$,
$LOCAL=[0,0, .., 0, l1, l2, ... lq]$

Aux mêmes adresses que ci-dessus, $l1, l2, lq$ sont les numéros locaux dans la matrice frontale de $PAREND(SN)$.(valeurs comprises entre 1 et $NBLIGN(PAREND(SN))$).

$NBASS$, compteur utilisé lui aussi pour l'assemblage, est défini ainsi : $NBASS(FSN1)$ est le nombre d'inconnues INC tel que $INC<SUPND(SN+1)$, où $PAREND(FSN1)=SN$. C'est donc le nombre d'inconnues de $FSN1$, assemblées dans la matrice mère SN et qui seront éliminées lors de l'élimination de SN .
$DEBFAC$ et $DEBFSN$, ces 2 tableaux donnent les mêmes valeurs, le premier est indicé par inconnue, le second par super-nœud, ils fournissent, pour $DEBFAC$, les adresses des coefficients diagonaux des inconnues dans la matrice factorisée. Pour $DEBFSN$, il s'agit de l'adresse du coefficient diagonal de la première inconnue du super-nœud.

On a vu sur la Figure 2, que les colonnes correspondant aux inconnues du super-nœud éliminé, (en grisé), constituaient les colonnes de la matrice factorisée que l'on appelle $FACTOR$. Les adresses des coefficients diagonaux fournies par DEBFAC sont des adresses dans un tableau virtuel $FACTOR$ contenant la matrice factorisée. En fait un ensemble de blocs JEVEUX contiendra la matrice factorisée, et on maniera un pointeur supplémentaire pour chaque bloc.

N.B. Programmation

On voit sur les figures précédentes que l'on stocke les colonnes du SN , toutes entières, (rectangle grisé), sans prendre en compte la symétrie du bloc diagonal. On consomme ainsi un peu plus de mémoire, mais on a affaire à un stockage régulier nécessaire à l'utilisation des routines BLAS qui demandent des stockages dans des tableaux FORTRAN à 2 dimensions.

2.6.2 MLTPOS

On rappelle que les matrices frontales issues des éliminations sont stockées dans une pile. Toutes les matrices filles d'un même super nœud SN sont stockées (empilées), consécutivement. Une fois lues (dépilées), elles ne sont plus réutilisées, et la matrice frontale issue de l'élimination de SN est stockée à leur place. La longueur de cette pile diminue ou augmente au fil des éliminations, Elle atteint un maximum qu'il est nécessaire de connaître.

MLTPOS parcourt l'arborescence, calcule $ESTIM$, longueur maximum de la pile pour son allocation ultérieure, construit un tableau $SEQ(1:NBSN)$ qui fournit l'ordre d'élimination des super-nœuds en parcourant de bas en haut l'arborescence. Il fournit $ADPILE$ qui contient l'adresse dans la pile des matrices frontales des SN.
MLTPOS contient un algorithme de renumérotation de l'ordre d'élimination des super-nœuds, destiné à minimiser la longueur de la pile des matrices frontales([AS87])

2.6.3 MLTBLC

Cette routine construit les pointeurs du découpage en blocs de la matrice factorisée. Chaque bloc contiendra les colonnes d'un nombre entier de super-nœud. On ne rencontrera jamais la configuration où les différentes colonnes d'un même super-nœud appartiendraient à 2 blocs.

Données :
$MAXBLOC$: longueur maximum des blocs. Chaque bloc contiendra les colonnes d'un maximum de super nœuds. $MAXBLOC$ est calculé avant l'appel à MLTBLC, c'est en fait la longueur (en colonnes) du plus gros des super-nœuds, et ce dernier (en général le plus haut dans l'arborescence) occupera un bloc à lui seul.

Résultats :
$NBLOC$: Nombre de blocs.
$LGBLOC(1:NBLOC)$: longueurs de chaque bloc.
$NCBLOC(1:NBLOC)$: définition des blocs, comme suit le bloc IB est constitué des colonnes des super-nœuds $SEQ(NCBLOC(IB-1)+1)$, à $SEQ(NCBLOC(IB))$.
$DECAL(1:NBSN)$ tableau de décalage qui permet d'accéder au premier terme diagonal de chaque super-nœud dans la matrice factorisée. On va illustrer cela par le schéma suivant :

```
ISN=0
Pour IB = 1,NBLOC                                                    !
Boucle sur les blocs de la factorisée
      JEVEUO(JEXNUM(FACTOL,IB),'L',IFACL)
                          ! Le bloc IB de la collection FACTOL est chargé en IFACL
     Pour NC = 1, NCBLOC(IB)              ! Boucle sur les Super-nœuds du bloc IB
           ISN = ISN +1
           SN = SEQ(ISN)
!          On suit la séquence d'élimination
           ADFAC= IFACL + DECAL(SN) -1
!          ADFAC est l'adresse dans ZR du premier coefficient du super-noeud SN
```

2.6.4 MLTPAS

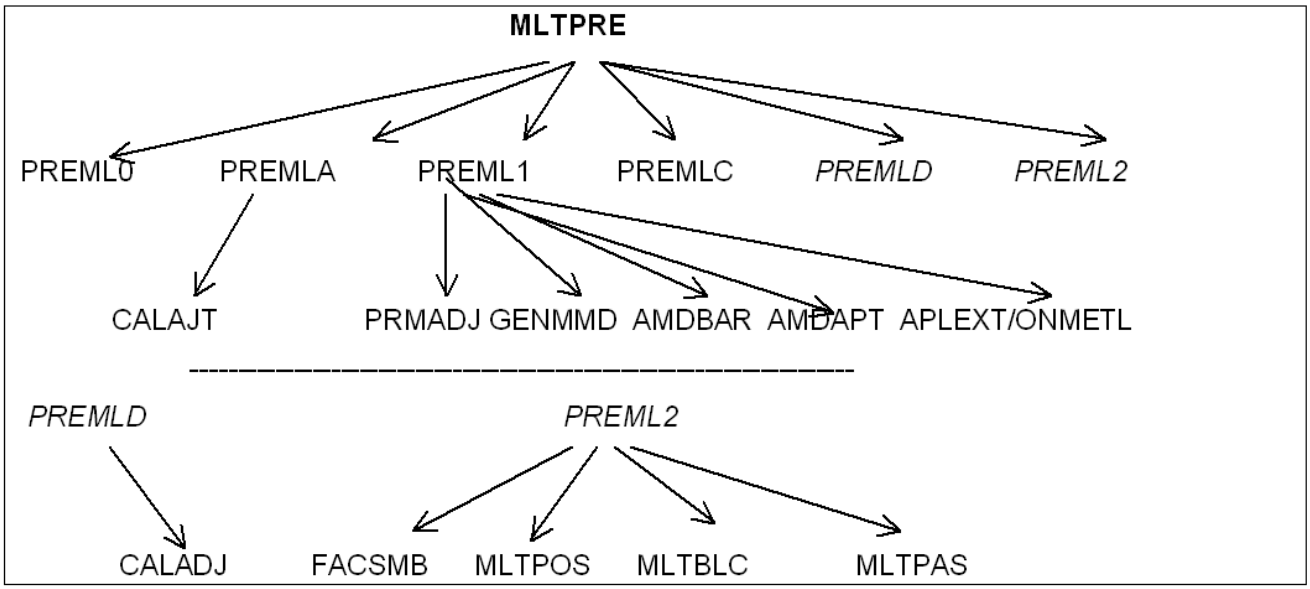
Cette routine calcule les adresses dans la matrice factorisée, des coefficients de la matrice initiale.
Ces adresses sont stockées dans la structure MLTF.ADNT, de nom FORTRAN $ADINIT$ ou COL de longueur $LMAT$.

N.B. Particularité de programmation.

Certaines adresses calculées de ADINIT sont négatives. En effet dans le cas non-symétrique, on a 2 matrices initiales MATI et MATS qui ont la même topologie, (mêmes voisins, même creux), la factorisation symbolique est la même.
Cependant, il peut arriver qu'un coefficient de MATI (partie inférieure de la matrice initiale), ait une destination dans la partie supérieure de la matrice factorisée. (De façon identique entre MATS et la partie inférieure de la factorisée). Dans ce cas l'adresse du coefficient initial est stockée négative et la routine d'injection des coefficients de la matrice initiale, MLTASA le prend en compte et adresse correctement les coefficients.

```
IF( CODE.GT.0 ) THEN
                   ZR(IFACL+ADPROV-DEB) = ZR(MATI+I1-1): MATI injectée dans IFACL(ower)
                   ZR(IFACU+ADPROV-DEB) = ZR(MATS+I1-1)
                ELSE
                   ZR(IFACL+ADPROV-DEB) = ZR(MATS+I1-1)
                   ZR(IFACU+ADPROV-DEB) = ZR(MATI+I1-1) MATI injectée dans IFACU(pper)
                ENDIF
```

2.7 Arbre d'appel de MLTPRE



3 Factorisation numérique MULFR8

On a affaire à la factorisation proprement dite, les routines appelées sont :

1) MLTPRE
2) MLTASA
3) MLTFC1

MLTPRE, vu précédemment, est appelé si nécessaire. On vérifie qu'il n'a pas déjà été appelé en amont, dans ce cas on sort de la routine.

3.1 MLTASA

Cette routine effectue l'injection des termes initiaux de la matrice dans la matrice factorisée. Le tableau ADINIT calculé précédemment par MLTPRE fournit les adresses.

Données :
L'enregistrement .VALM de la structure NOMMAT de type sd_matr_asse fournie.
$ADINIT$ et $LGBLOC(1:NBLOC)$ calculés par MLTPRE.

Résultats :
MLTASA crée une collection dispersée de nom $FACTOL$ ayant $NBLOC$ blocs de longueurs données par $LGBLOC(1:NBLOC)$.

Dans le cas non –symétrique, la routine lit 2 « .VALM » de noms respectifs $MATI$ et $MATS$, parties inférieure et supérieure de la matrice initiale, et crée 2 collections $FACTOL(ower)$ et $FACTOU(pper)$.

3.2 MLTFC1

On est dans le cas où la pile des matrices frontales est rangée dans un tableau FORTRAN nommé $PILE$ et alloué par WKVECT dans l'appelant MULFR8, ce tableau est détruit en fin de routine.

Une version avec les matrices frontales en collection dispersée a été écrite (MLTFCB) , mais n'est pas utilisée.
MLTFC1 a la structure suivante :

Dans une boucle externe sur les blocs de la factorisée,
Chargement du bloc en mémoire
Boucle sur les nœuds du bloc
SEQ fournit l'ordre d'élimination
On élimine donc le super-nœud SNI en différentes phases :
Assemblages des matrices « filles » par appel à MLTAFF et MLTAFF
Élimination des colonnes de SNI qui sont les colonnes de la factorisée : MLTFLM
Mise à jour des inconnues restantes dans la matrice frontale (complément de Schur) : MLTFMJ
Fin de boucle
Fin de boucle

```
Pour IB = 1,NBLOC                                                              !
Boucle sur les blocs de la factorisée
      JEVEUO(JEXNUM(FACTOL,IB),'L',IFACL)
      ! Le bloc IB de la collection FACTOL est chargé en IFACL
     Pour NC = 1, NCBLOC(IB)   ! Boucle sur les Super-nœuds du bloc IB
           ISN = ISN +1
           SNI= SEQ(ISN)
           Pour tous les SN  fils de SNI ! assemblage des matrices filles
            mltafp
              mltaff
           ! fin de l'assemblage des matrices filles

            mltflm  ! élimination du SN : factorisation sur les colonnes du SN
            mltfmj  ! apport des colonnes éliminées sur les autres degrés de liberté du
front, mise à jour de la matrice frontale
                    !        fin de boucle sur les super nœuds
!        fin de boucle sur les blocs
```

Avant de décrire les routines appelées, il faut préciser la façon dont la matrice est rangée. On regarde le schéma suivant :

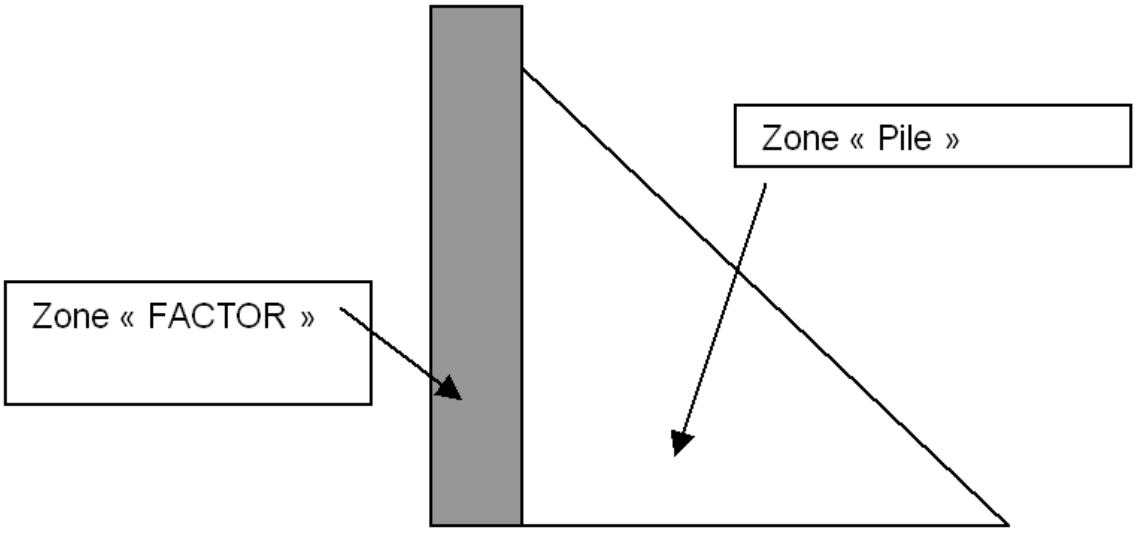


Figure 3 Décomposition de la matrice frontale en 2 zones de stockage

La partie rectangulaire grisée représente la zone « active de la matrice frontale », ce sont les colonnes du SN qui vont être éliminées, et vont constituer les colonnes de la matrice factorisée $FACTOR$. Pour éviter une copie de mémoire entre la matrice frontale et $FACTOR$, on stocke directement les coefficients dans $FACTOR$, on dira que c'est la partie « $FACTOR$ » de la matrice frontale. A coté, on a la zone « passive » de la matrice frontale, (complément de Schur en fait), qui sera stockée dans la pile, de même, pour éviter une copie de mémoire, on stocke directement cette partie de la matrice dans la pile.

MLTAFP et MLTAFF sont les routines qui assemblent les matrices frontales « filles » dans la matrice frontale « mère ».

3.2.1 mltafp

Cette routine assemble les coefficients de la matrice frontale « fille », dans la zone « $FACTOR$ » de la matrice frontale « mère ».

3.2.2 mltaff

Cette routine copie les coefficients de la matrice frontale « fille », dans la zone « pile » de la matrice frontale « mère ».

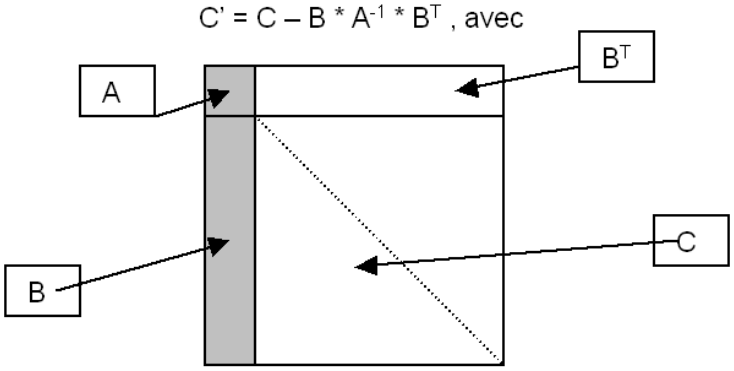
3.2.3 mltflm

Elimination du super nœud SN, i.e. factorisation partielle de la matrice frontale, travail dans la zone « $FACTOR$ ».

MLTFLM appelle :
DGEMV : routine Blas qui effectue les produits matrice*vecteur
MLTFLD (appel à DGEMV)
MLTFLJ (appel à DGEMM).

3.2.4 mltfmj

Mise à jour de la zone « $PILE$ » qui est modifiée par l'élimination du SN.
MLTFMJ appelle :
DGEMM: routine Blas qui effectue les produits matrice-matrice.
MLTFMJ est la routine la plus consommatrice de temps de calcul, elle effectue la mise à jour de la matrice frontale sur les lignes et les colonnes des degrés de liberté voisins de ceux éliminés, c'est une opération que l'on peut écrire :



Afin de diminuer les coûts de calcul, on divise virtuellement les matrices A , B et C en sous blocs d'ordre NB et l'opération (1) ci-dessus est effectuée par blocs en appelant la routine Blas DGEMM. Cette routine optimisée par le constructeur sur chaque machine tire profit du fait que les petits blocs d'ordre NB ainsi multipliés tiennent dans le cache primaire, diminuant ainsi les coûts d'accès à la mémoire. NB est fixé actuellement à 96, cette valeur généralement de l'ordre de quelques dizaines dépend de la taille du cache $L1$, le plus proche du processeur et de la taille des problèmes traités. Il pourrait être ré estimer de temps en temps en fonction des nouvelles acquisitions de machines, sur un ensemble de cas-test significatifs. Les essais effectués précédemment ont montré que les performances variaient peu quand NB variait autour de 96.

MLTFLM utilise aussi cette division en blocs et utilise DGEMM par l'intermédiaire de MLTFLJ. (Dans le cas où le nombre de colonnes à éliminer est inférieur à un certain seuil ($pmin=10$), l'appel à ces 2 dernières routines est remplacé par l'appel à MLFT21). Ceci a été fait par souci d'optimisation de l'élimination des « petits » super-nœuds, il conviendrait de vérifier que cela est toujours justifié. Ainsi, par souci de simplification, après vérification de l'absence de dégradation des performances pour les petits cas, on pourrait supprimer ce seuil et ainsi éliminer un branche d'appel de sous-programme.

Pour illustrer ceci, l'arborescence des routines appelées dans le cas symétrique est le suivant :

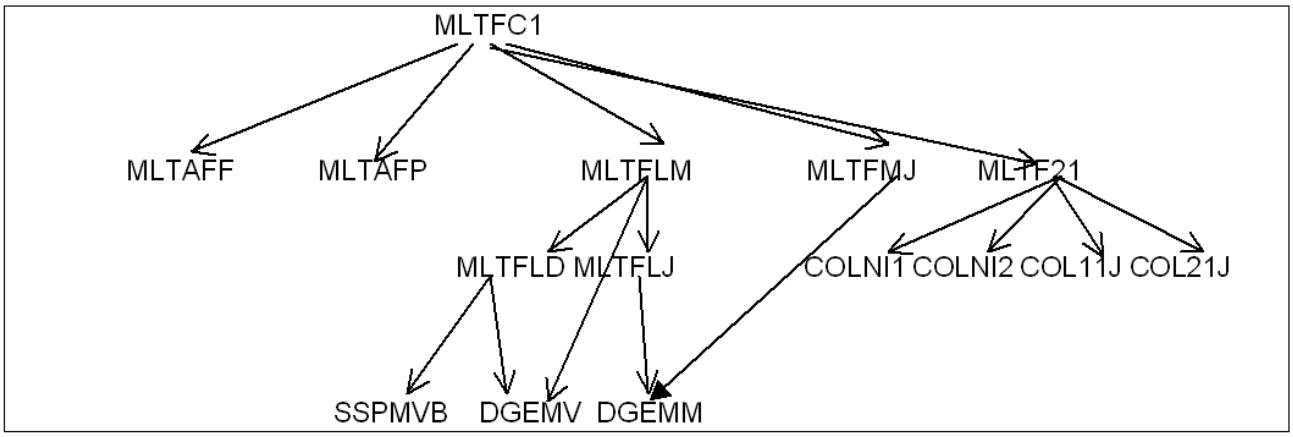


Figure 4 Factorisation (cas réel symétrique).

4 Remontée du système : RLTFR8

Si le nombre de seconds membres à résoudre simultanément est supérieur à 4, on fait appel à une méthode par blocs implémentée par la routine RLBFR8, sinon MLTDRA est appelé pour chaque second membre.
Données : ce sont les pointeurs issus de MLTPRE :

ADRESS ,
GLOBAL ,
SUPND ,
LGSN ,
ANC ,
NOUV ,
SEQ ,
LGBLOC ,
NCBLOC ,
DECAL .

Et la matrice factorisée issue de MULFR8
$FACTOL$: .VALF (et $FACTOU$: .WALF en non symétrique)

Les seconds membres :
$XSOL(1:\neq,1:NBSOL)$

Résultats : Ils se retrouvent dans le tableau qui a contenu les seconds membres : $XSOL$.

4.1 MLTDRA

Cette routine effectue successivement, pour un seul second membre, la descente, la division par le terme diagonal, et la remontée. Dans la descente, on fait appel au produit matrice-vecteur $DGEMV$ des bibliothèques Blas, au-delà d'un certain seuil. En dessous on optimise « à la main » dans la routine SSPMVB.
(Cette optimisation est une vectorisation écrite pour le CRAY, elle pourrait donc être revue voire supprimée !)

4.2 RLBFR8

En présence de plusieurs seconds membres, les appels à DGEMV cités au paragraphe précédent, sont remplacés par des appels à DGEMM, produit matrice*matrice, sur des blocs matriciels d'ordre NB . On vise ainsi la performance fournie par ces Blas de niveau 3, comme dans la factorisation numérique.
Les routines appelées sont :

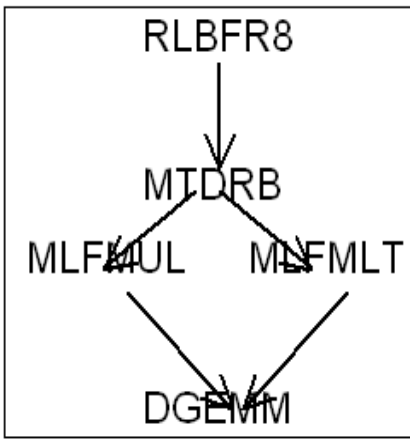


Figure 5 Descente-remontée par blocs

5 Version non-symétrique

La divergence entre les versions symétriques et non-symétrique se trouve à l'intérieur de la routine MLTFC1, l'arborescence des routines appelées dans le cas non-symétrique est le suivant .
(MLNFLM, MLNFMJ, MLNFLD, MLNFLJ remplacent respectivement MLTFLM, MLTFMJ, MLTFLD, MLTFLJ).

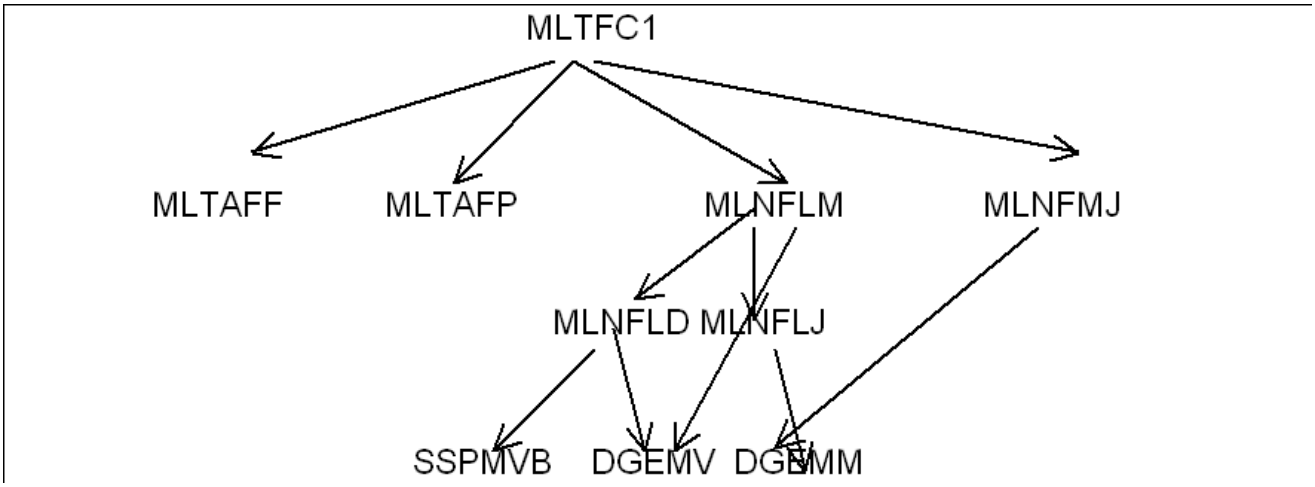


Figure 6 Arborescence réelle non –symétrique.

6 Version complexe

6.1 Factorisation complexe symétrique

La factorisation complexe symétrique est représentée par l'organigramme suivant, les changements dans les noms de routines sont marqués en gras. MLFC16 remplace MULFR8 et MLTASC effectue l'injection des termes initiaux de façon analogue à MLTASA.

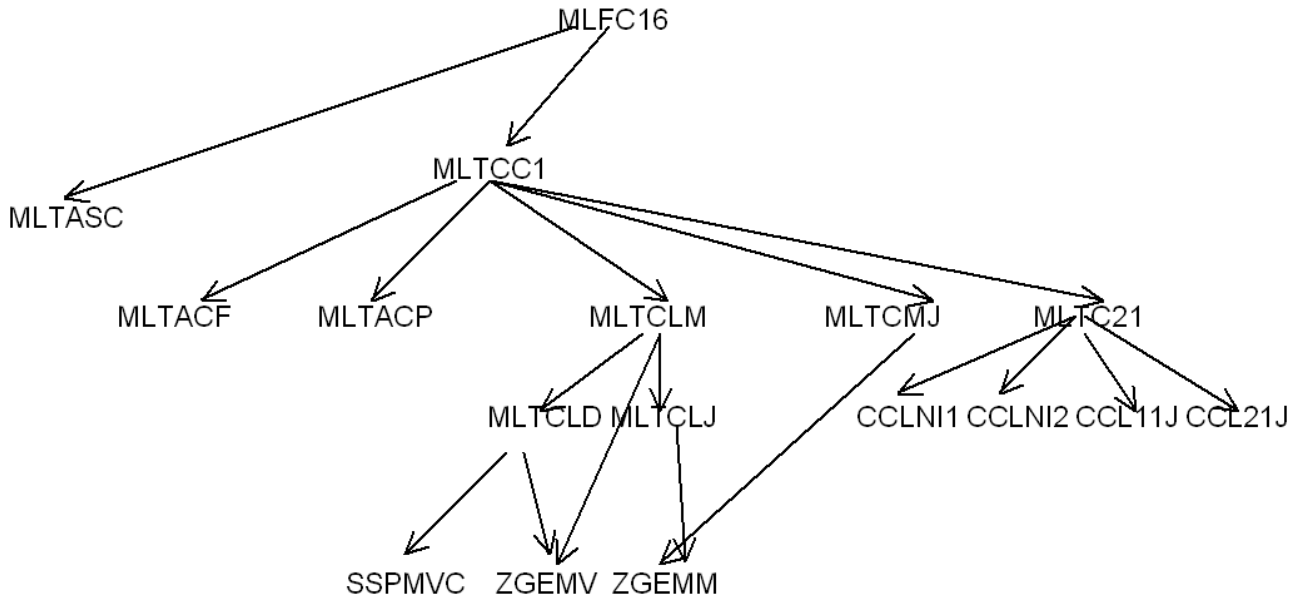


Figure 7 Version complexe symétrique

6.2 Factorisation complexe non-symétrique

La factorisation complexe non symétrique est représentée par l'organigramme suivant, les changements dans les noms de routines sont marqués en gras.

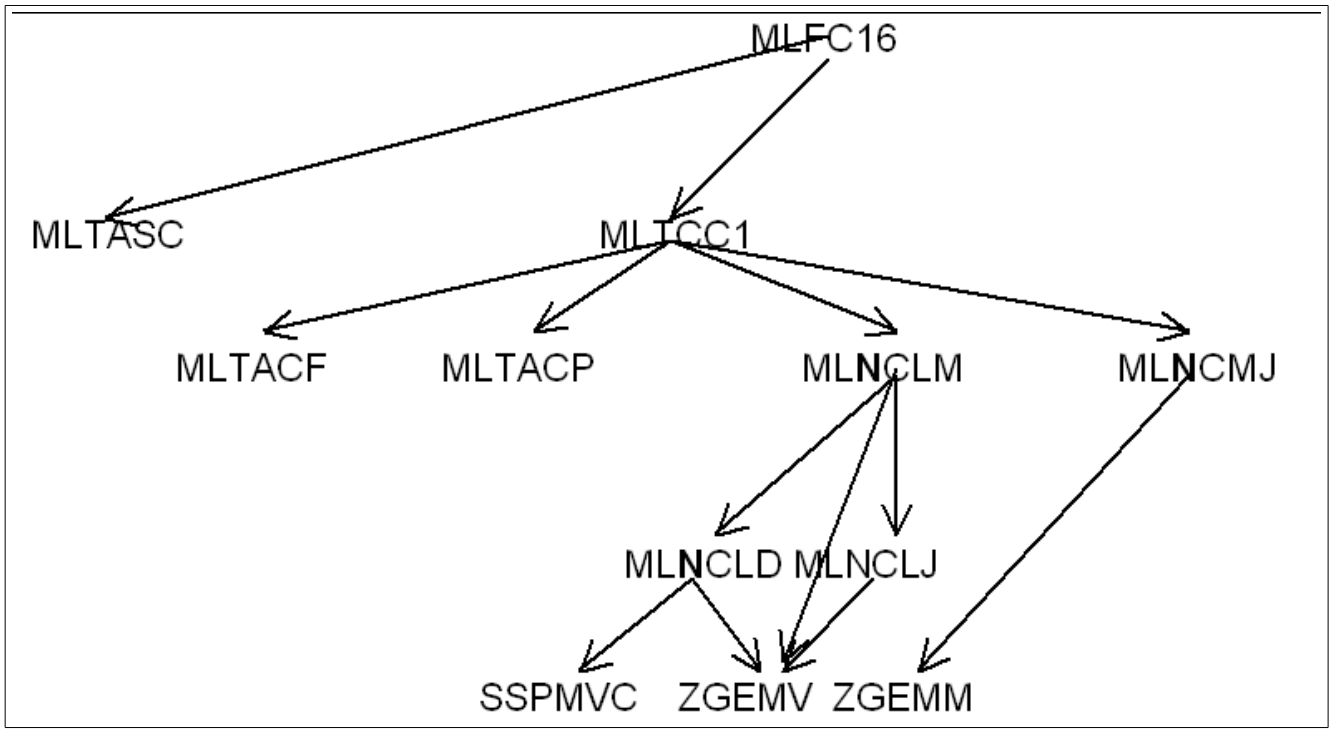


Figure 8 Factorisation complexe non symétrique

6.3 Descente-remontée complexe

La routine RLFC16 est l'équivalent complexez de RLTFR8, et MLTDCA, l'équivalent de MLTDRA.
La version par blocs pour plusieurs seconds membres RLBFR8 en réels n'existe pas en complexe.

MLTDCA appelle SSPMVC et ZGEMV, cette dernière, de la bibliothèque Blas, est la version complexe de DGEMV. SSPMVC est la version complexe de SSPMVB.

7 Parallélisation

La parallélisation dans MULT_FRONT est réalisée par directives Openmp. Ces directives sont placées dans les routines suivantes :

Cas réel symétrique : MLTFMJ, MLTFLJ
Cas complexe symétrique : MLTCMJ, MLTCLJ
Cas réel non symétrique : MLNFMJ, MLNFLJ
Cas complexe non symétrique : MLNCMJ, MLNCLJ

La parallélisation implémentée est « dite » interne, elle a lieu à l'intérieur du processus d'élimination d'un super nœud. L'élimination des super nœuds, dite « externe » peut aussi être parallélisée. Mais elle n'est pas implémentée pour la raison suivante : L'élimination simultanée de plusieurs super nœuds nécessite une occupation de la mémoire supérieure à celle de la version séquentielle. Or le souci de l'occupation de la mémoire a toujours prédominé celui du gain de temps de restitution lors des développements précédents de MULT_FRONT.

La parallélisation interne, elle, ne nécessite aucune mémoire supplémentaire à celle de la version séquentielle, elle ne nécessite non plus aucun développement de code particulier, seul un simple ajout de directives.
On va décrire la parallélisation de la routine MLTFMJ, celle de MLTFLJ suit les mêmes principes, ainsi que les autres routines citées plus haut.

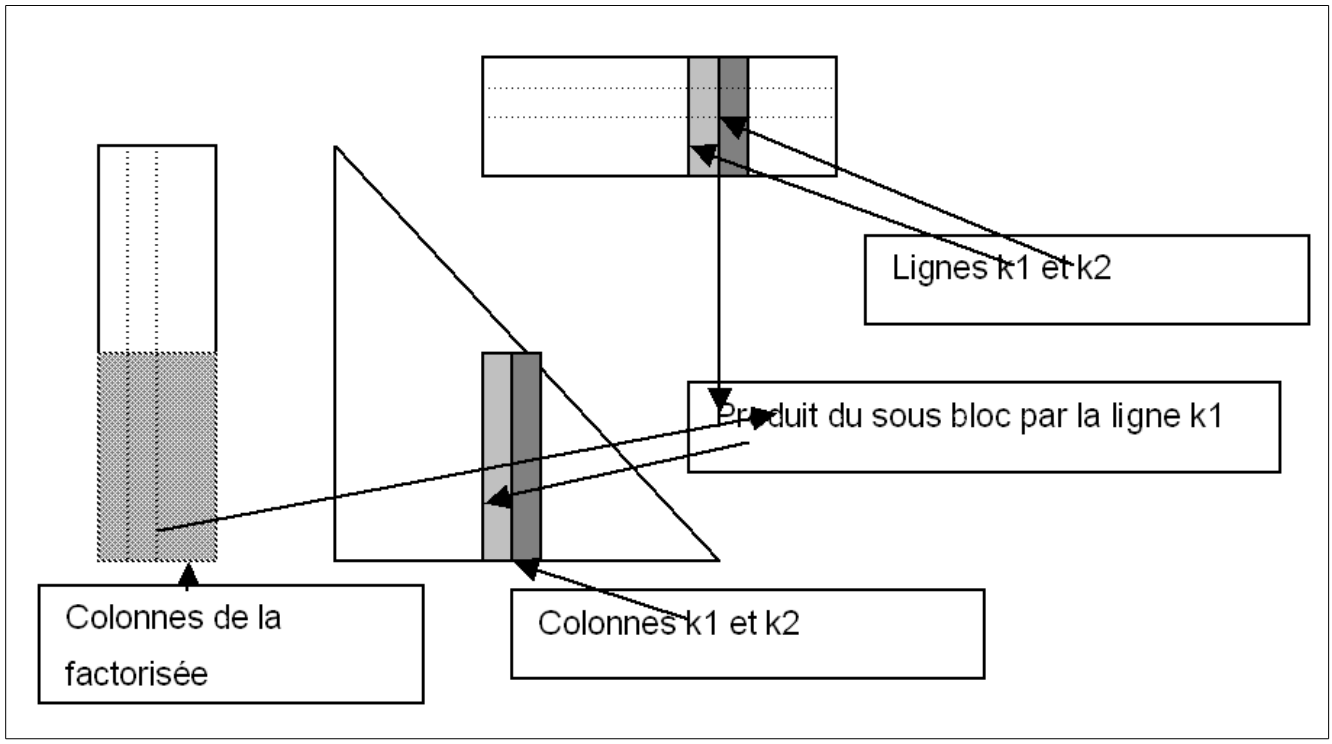


Figure 9 Schéma de la mise à jour de la matrice frontale

On voit sur la Figure ci-dessus, un schéma de la mise à jour de la matrice frontale lors de l'élimination d'un nœud. Le triangle représente la partie inférieure de la matrice frontale, et les 2 rectangles représentent les colonnes de la matrice factorisée, colonnes correspondant uniquement au super nœud éliminé. Dans un premier temps, on raisonnera par colonnes. On sait que l'on met à jour la colonne $k1$ de la matrice frontale en lui soustrayant le produit du sous-bloc de colonnes (en grisé ci-dessus) par la ligne $k1$ (multipliée elle-même par les termes diagonaux). (opération de type Blas2). Cette opération de mise à jour est parallélisée sans problème par une directive OpenMP, les colonnes de la matrice frontale sont adjacentes, il n'y a aucun conflit d'adresses. Toutes les variables sont partagées, sauf les indices de boucles et les variables locales. On a vu ci-dessus que par souci de performance, on travaille par bloc de colonnes, afin d'utiliser des Blas3, la parallélisation évoquée ci-dessus se fait donc en fait sur les blocs de colonnes et non sur les colonnes elles-mêmes.

Aperçu du code :

```
C$OMP PARALLEL DO DEFAULT(PRIVATE)
C$OMP+SHARED(N,M,P,NMB,NB,RESTM,FRONT,ADPER,DECAL,FRN,TRAV,C)
C$OMP+SHARED(TRA,TRB,ALPHA,BETA)
C$OMP+SCHEDULE(STATIC,1)
     DO 1000 KB = 1,NMB
     NUMPRO=MLNUMP()
C     K : INDICE DE COLONNE DANS LA MATRICE FRONTALE (ABSOLU DE 1 A N)
       K = NB*(KB-1) + 1 +P
       DO 100 I=1,P
         S = FRONT(ADPER(I))
         ADD= N*(I-1) + K
         DO 50 J=1,NB
           TRAV(I,J,NUMPRO) = FRONT(ADD)*S ! TRAV contient les produits : terme diagonal*ligne
                 ADD = ADD + 1
 50          CONTINUE
 100      CONTINUE

          DO 500 IB = KB,NMB
```

```
            IA = K + NB*(IB-KB)
            IT=1
            CALL DGEMM( TRA,TRB,NB,NB,P,ALPHA,FRONT(IA),N,
     &                  TRAV(IT,1,NUMPRO), P,BETA,C(1,1,NUMPRO), NB)
C     RECOPIE

C
            DO 501 I=1,NB
               I1=I-1
               IF(IB.EQ.KB) THEN
                  J1= I
                  IND = ADPER(K + I1) - DECAL
               ELSE
                  J1=1
                  IND = ADPER(K + I1) - DECAL + NB*(IB-KB)  - I1
               ENDIF
               DO 502 J=J1,NB
                     FRN(IND) = FRN(IND) +C(J,I,NUMPRO)! Recopie dans la matrice
frontale frn
                  IND = IND +1
 502           CONTINUE
 501        CONTINUE
 500     CONTINUE

 1000 CONTINUE
C$OMP END PARALLEL DO
```

N.B. Parallélisation.

La version séquentielle nécessite un tableau de travail TRA , écrit pour chaque bloc, il est nécessaire de le dupliquer pour la parallélisation. Dans la boucle parallèle, on fait appel à la fonction MLNUMP qui fournit le numéro de thread traitant l'itération de boucle ; NUMPRO , on travaille ensuite dans le plan NUMPRO du tableau TRAV .

8 Remarques diverses

Quelques paramètres, valeurs de seuil, sont utilisés dans MULT_FRONT. Ils sont destinés à optimiser le code, en termes de temps de calcul. Ce sont :
NB : longueur des blocs pour l'utilisation de DGEMM. *NB* est fourni par la fonction LLBLOC() qui renvoie la valeur 96. Il serait opportun à chaque changement de processeur de vérifier si cette valeur est optimale. *PMIN* (fixée à 10 dans MLTFC1). Cette valeur est liée à la précédente. Les super nœuds dont la taille (nombre de degrés de liberté, donné par *LGSN*) est inférieur à *PMIN* , sont traités par MLTF21 et non par MLMTFLM, MLTFMJ. Cela veut dire que les inconnues sont traitées colonne par colonne et non par bloc avec appel à SDGEMM).

Ce seuil vise à simplifier le traitement des petits cas-tests et de ne pas activer la machinerie par blocs pour les petits super nœuds.

NBSOL : C'est le nombre de seconds membres traités lors de la descente/remontée, si *NBSOL* est inférieur à 4, les Descentes/remontées sont effectuées les unes après les autres par MLTDRA, sinon on travaille par BLOC dans RLBFR8. *SEUIN SEUIK* : valeurs utilisées dans MLTDRA : elles déterminent l'usage de DGEMV pour le produit matrice-vecteur, sous ces seuils, on appellera SSPMVB « optimisé » à la main. (Étant donné le peu de poids de MLTDRA, ces 2 seuils pourraient être supprimés).

Comme on l'a déjà remarqué ci-dessus, il conviendrait de temps en temps de vérifier si ces seuils sont adéquats.

9 Conclusions, développements envisageables.

Le compte-rendu [RB12] a été diffusé. Il fait office de conclusion détaillée. Il offre un état des lieux et des perspectives plus détaillées sur la maintenance et les développements éventuels de MULT_FRONT, dans le contexte de la disponibilité dans *Code_Aster*, d'un autre solveur, MUMPS. On résume cependant dans les 2 paragraphes suivants, le périmètre d'exécution et le bilan des performances de MULT_FRONT.

9.1 Périmètre d'exécution

Le périmètre de la méthode multi-frontale est quasi-complet depuis les développements de 2011 qui permettent de factoriser les matrices dites « généralisées ». Seuls les systèmes numériquement instables, nécessitant une méthode de pivot (X-FEM par exemple) sont à exclure du choix du solveur MULT_FRONT. Un développement moins récent (renumérotation des nœuds d'interpolation et non plus des degrés de liberté), permet de conserver l'ordre de ces degrés de liberté à l'intérieur d'un nœud et permet de traiter les éléments incompressibles. Le recours à MUMPS est la solution pour les systèmes nécessitant une méthode avec pivot. Cependant un développement de MULT_FRONT est envisageable : il est possible d'effectuer, à l'intérieur de chaque super nœud, c'est à dire parmi les inconnues éliminées à chaque étape, une recherche (évidemment partielle) de pivot. Une recherche plus complète, impliquerait alors la fusion de plusieurs super nœuds (fusion de matrices fille et mère) et paraît difficilement envisageable. Ce pivotage partiel nécessite une quantité de travail non négligeable et constituerait un risque pour la stabilité du code.

9.2 Performance

Les performances séquentielles (mode mono-processeur) sont comparables à celles de MUMPS dans beaucoup de cas. Seuls les cas présentant beaucoup de relations linéaires entre les degrés de liberté peuvent être très défavorables à MULT_FRONT qui crée du remplissage matriciel dans ces cas. Parfois il a suffi de réordonner la données de ces conditions aux limites pour régler le problème.

En ce qui concerne le parallélisme (localisé comme on l'a vu plus haut, à l'intérieur de l'élimination de chaque super nœud), il est peu utilisé et peu offrir un facteur d'accélération entre 2 et 3 pour une exécution sur quatre processeurs (ou cœurs de processeurs). Ce parallélisme est intéressant pour les gros cas nécessitant beaucoup de mémoire et devant s'exécuter pratiquement seuls en machine. Cela permet de réduire notablement leur durée de résidence en machine. L'arrivée de processeurs dotés de plus en plus de cœurs devrait fournir l'occasion de nouveaux tests de performances pour de tels gros cas. Cependant cette parallélisation en mémoire partagée n'offre pas de gain en mémoire. Ce que permet la parallélisation en mémoire distribué (MUMPS). Souvent ce problème de mémoire est un point bloquant pour les gros systèmes linéaires.
La méthode multi-frontale offre la possibilité d'une autre parallélisation plus élaborée sur les différentes branches de l'arbre d'élimination. Il n'est pas envisageable de la programmer dans MULT_FRONT, ce serait « refaire » ce qui est fait dans MUMPS. Le code n'a pas été conçu pour la parallélisation par envoi de message.

10 Références.

1 [DU86] Duff, I.S., REID J.K., « Parallel implementation of multi-frontal schemes », Parallel Computing,3, (1986)

2 [AS87] Ashcraft C., "A vector implementation of the multi-frontal method for large, sparse symmetric positive definite systems". Boeing Computer Service Technical Report ETA-TR 51, (1987)

3 [RO93] Rose C., "Une méthode multi-frontale pour la résolution directe des systèmes linéaires", EDF R&D, note HI-76/93/008

4 [RO95] Rose C., "Une méthode multi-frontale parallèle pour la résolution directe des systèmes linéaires", EDF R&D, note HI-76/95/021

5 [RB12] Rose C., Boiteau O., Méthode multi-frontale native dans Code_Aster : état des lieux et perspectives, CR-I23-2012-001