

Manuel d'Utilisation
Fascicule U4.5- : Méthodes de résolution
Document : U4.50.01**Mot clé SOLVEUR**

1 But

Choisir le mode de stockage des matrices et l'algorithme de résolution. Ce mot clé facteur se retrouve dans un certain nombre de commandes conduisant à la résolution de systèmes linéaires. Pour les algorithmes de résolution il permet de choisir entre factorisation "classique" de type "GAUSS" ('LDLT'), factorisation multi-frontale ('MULT_FRONT' ou 'MUMPS'), gradient conjugué préconditionné ILU(k) ('GCPC') ou solveur FETI par décomposition de domaines ('FETI').

Pour chaque type de solveur, certains paramètres numériques facultatifs sont accessibles et sont décrits ici. Par défaut, c'est le solveur 'MULT_FRONT' qui est utilisé. Le solveur 'FETI', quant à lui, est encore en phase de développement, il n'est donc pas conseillé de l'utiliser sans conseils préalables de l'équipe de développement.

2 Syntaxe

```

◇ SOLVEUR = _F (

# Factorisation de type "multi-frontale" :
/  METHODE = 'MULT_FRONT' ,                                [DEFAULT]

# Paramètre numérique
    ◇ RENUM = / 'METIS' ,                                [DEFAULT]
                                / 'MD' ,
                                / 'MDA' .

# Paramètres fonctionnels
    ◇ STOP_SINGULIER = / 'OUI' ,                                [DEFAULT]
                                / 'NON' .

    ◇ NPREC = / 8 ,                                [DEFAULT]
                                / nprec .                [I]

# Factorisation "classique" de type Gauss :
/  METHODE = 'LDLT' ,

# Paramètre numérique
    ◇ RENUM = / 'RCMK' ,                                [DEFAULT]
                                / 'SANS' .

# Paramètres fonctionnels
    ◇ STOP_SINGULIER = / 'OUI' ,                                [DEFAULT]
                                / 'NON' .

    ◇ NPREC = / 8 ,                                [DEFAULT]
                                / nprec .                [I]

# Factorisation de type "multi-frontale" avec MUMPS:
/  METHODE = 'MUMPS' ,

# Paramètres fonctionnels
    ◇ TYPE_RESOL = / 'AUTO' ,                                [DEFAULT]
                                / 'NONSYM' ,
                                / 'SYMGEN' ,
                                / 'SYMDEF' .

    ◇ PCENT_PIVOT = / 10 ,                                [DEFAULT]
                                / pcent .                [R]

    ◇ ELIM_LAGR2 = / 'OUI' ,                                [DEFAULT]
                                / 'NON' .

    ◇ RESI_RELA = / -1.d0 ,                                [DEFAULT]
                                / resi .                 [R]

# Paramètres numériques
    ◇ PRETRAITEMENTS = / 'AUTO' ,                                [DEFAULT]
                                / 'SANS' .

    ◇ RENUM = / 'AUTO' ,                                [DEFAULT]
                                / 'AMD' ,
                                / 'AMF' ,
                                / 'QAMD' ,
                                / 'PORD' ,
                                / 'METIS' .

# Paramètres pour le parallélisme
    ◇ PARALLELISME = / 'CENTRALISE' ,                                [DEFAULT]
                                / 'DISTRIBUE_MAILLE' ,
                                / 'DISTRIBUE_SD' .

    ◇ PARTITION = sdfeti

    ◇ CHARGE_PROC0_MA = / 0 ,                                [DEFAULT]
                                / chargema .                [I]

    ◇ CHARGE_PROC0_SD = / 0 ,                                [DEFAULT]
                                / chargesd .                [I]

```

Titre : Mot clé SOLVEUR
Auteur(s) : O.BOITEAU

Date : 22/02/08
Clé : U4.50.01-G1 Page : 3/22

```
# Méthode itérative du gradient conjugué :
/  METHODE = 'GCPC' ,

# Paramètres numériques
    ◇ PRE_COND      =      'LDLT_INC' .          [DEFAULT]
    ◇ NIVE_REPLISSAGE = / 0 ,                    [DEFAULT]
                                     / niv .

# Paramètres fonctionnels
    ◇ NMAX_ITER      =      / 0 ,                [DEFAULT]
                                     / niter.      [I]
    ◇ RESI_RELA      =      / 10-6 ,           [DEFAULT]
                                     / resi .       [R]

# Méthode de décomposition de domaines FETI :
/  METHODE = 'FETI' ,

# Paramètres fonctionnels
    ◆ PARTITION      =      sdfeti
    ◇ NMAX_ITER      =      / 0 ,                [DEFAULT]
                                     / niter.      [I]
    ◇ REAC_RESI      =      / 0 ,                [DEFAULT]
                                     / nreac.      [I]
    ◇ RESI_RELA      =      / 10-6 ,           [DEFAULT]
                                     / resi .       [R]

# Paramètres du problème d'interface
    ◇ PRE_COND      =      / 'LUMPE' ,          [DEFAULT]
                                     / 'SANS' .
    ◇ SCALING        =      / 'MULT' ,           [DEFAULT]
                                     / 'SANS' .
    ◇ TYPE_REORTHO_DD=      / 'GSM' ,            [DEFAULT]
                                     / 'GS' ,
                                     / 'IGSM' ,
                                     / 'SANS' .
    ◇ NB_REORTHO_DD =      / 0 ,                [DEFAULT]
                                     / nb_reortho . [I]

# Paramètres des problèmes locaux
    ◇ RENUM          =      / 'METIS' ,          [DEFAULT]
                                     / 'MD' ,
                                     / 'MDA' .
    ◇ STOP_SINGULIER =      / 'OUI' ,           [DEFAULT]
                                     / 'NON' .
    ◇ NPREC          =      / 8 ,               [DEFAULT]
                                     / nprec .     [I]

# Paramètres de tests, de profiling et de monitoring. Divers.
    ◇ VERIF_SDFETI   =      / 'OUI' ,           [DEFAULT]
                                     / 'NON' .
    ◇ TEST_CONTINU    =      / 10-6 ,           [DEFAULT]
                                     / test_continu . [R]
    ◇ STOCKAGE_GI     =      / 'CAL' ,           [DEFAULT]
                                     / 'OUI' ,
                                     / 'NON' .
    ◇ INFO_FETI       =      / 'FFFFFFFFFFFFFFF' , [DEFAULT]
                                     / info_feti.   [K15]

# Paramètre pour le parallélisme
    ◇ NB_SD_PROC0     =      / 0 ,               [DEFAULT]
                                     / nb_sdproc0 . [I]

# Paramètres pour les accélérations (problème de type multiples seconds membres)
    ◇ ACCELERATION_SM =      / 'OUI' ,           [DEFAULT]
                                     / 'NON' .
    ◇ NB_REORTHO_INST =      / 0 ,               [DEFAULT]
                                     / nb_reortho_inst . [I]
```

Titre : Mot clé SOLVEUR
Auteur(s) : O.BOITEAU

Date : 22/02/08
Clé : U4.50.01-G1 Page : 4/22

Paramètre commun à tous les solveurs
◇ SYME = / 'NON' , [DEFAULT]
/ 'OUI' .
),

3 Opérandes

3.1 Opérande METHODE

◇ METHODE = 'MULT_FRONT' (défaut) / 'LDLT' / 'MUMPS' / 'GCPC' / 'FETI'

Ce mot clé permet de choisir la méthode de résolution des systèmes linéaires :

/ 'MULT_FRONT'	Solveur direct de type 'multi-frontale'. Le stockage matriciel est 'MORSE' et donc proscrit tout pivotage. Cette méthode est parallélisée en mémoire partagée (OpenMP) et peut être exécutée sur plusieurs processeurs (via l'interface Astk menu Options – Options de lancement). La matrice initiale est stockée dans un seul objet JEVEUX et sa factorisée est répartie sur plusieurs, donc peut être déchargée partiellement et automatiquement sur disque.
/ 'LDLT'	Solveur direct avec factorisation de Crout par blocs (sans pivotage). Le stockage matriciel est 'ligne de ciel' ou 'SKYLINE'. On a une pagination de la mémoire complètement paramétrable (la matrice est décomposée en blocs gérés en mémoire de façon indépendante et déchargés sur disque au fur et à mesure) qui permet de passer de gros cas mais qui se paye par des accès disques coûteux.
/ 'MUMPS'	Solveur direct de type 'multi-frontale' avec pivotage. Ce solveur est obtenu en se "branchant" sur la bibliothèque MUMPS développée par CERFACS-ENSEEIH-IRIA-Parallab (voir copyright plus bas). Pour Code_Aster, son intérêt principal réside dans sa capacité à pivoter lignes et/ou colonnes de la matrice lors de la factorisation en cas de pivot petit. Cette possibilité est utile (voire indispensable) pour les modèles conduisant à des matrices non définies positives (hors conditions aux limites) ; par exemple, les éléments "mixtes" ayant des ddls de type "Lagrange" (éléments incompressibles...). Informatiquement, ce solveur pose deux problèmes : il nécessite un compilateur fortran90 et il faut partager la mémoire entre JEVEUX et la bibliothèque MUMPS (usage du bouton "mem_aster" du menu OPTIONS d'ASTK). Cette méthode est parallélisée en mémoire distribuée (MPI) et peut être exécutée sur plusieurs processeurs (via l'interface Astk menu Options – Options de lancement). On peut aussi profiter du parallélisme en mémoire partagée des librairies d'algèbre linéaire de bas niveau (BLAS).
/ 'GCPC'	Solveur itératif de type gradient conjugué avec préconditionnement ILU(k). Le stockage de la matrice est alors 'MORSE'. La matrice initiale et sa factorisée incomplète sont stockées, chacune, dans un seul objet JEVEUX.
/ 'FETI'	Solveur par décomposition de domaines de type FETI : gradient conjugué préconditionné projeté (GCPPC) pour le problème d'interface et solveur direct multi-frontale pour les inversions des matrices de rigidité locales. Les problèmes locaux étant inversés par la multi-frontale 'MULT_FRONT', leurs matrices associées, matrices de rigidité locales et factorisées, sont traitées comme tel (cf. ci-dessus). Cette méthode est parallélisée en mémoire distribuée (MPI).

Les valeurs par défaut des autres mots clés sont alors prises automatiquement en fonction de la méthode choisie.

Solveur	Périmètre	Robustesse	CPU	Mémoire	Détails
Direct					
MULT_FRONT	Entier Sauf certaines	+++	Séq : +++ // : +	++ out-of-core	Solveur de référence.

Solveur	Périmètre	Robustesse	CPU	Mémoire	Détails
	modélisations (XFEM, incompressible...)		(speed-up~2)		Sur 4 proc.
MUMPS CENTRALISE	Sauf modal et STOP_SINGULIER= 'DECOUPE'	+++	Séq : +++ // : ++ (sp~8)	--- in core	Plutôt en linéaire Sur 16 proc.
MUMPS DISTRIBUE	Idem centralisé moins : cmde éclatées et DYNA_LINE_TRAN	++	Séq : +++ // : +++ (sp~16)	--- in core	Surtout en non linéaire Sur 32 proc. Meilleur équilibrage via DISTRIBUE_SD
LDLT	Idem MULT_FRONT	+++	Séq : +	++ out-of-core	Plutôt les petits cas
Itératif ou hybride					
FETI	MECA_STATIQUE STAT_NON_LINE en symétrie réel	---	Séq : +++ // : +++ (sp~8)	+++ out-of-core	Gros cas plutôt linéaire. Sur 16 proc. Paramétrage pointu Solveur de recherche.
GCPC	Sauf modal En symétrie réel	-	Séq : + Parfois très efficace (thermique...)	+++ out-of-core	Ne pas trop augmenter le niveau de préconditionnement

Tableau 3.1-a : Synopsis des solveurs linéaires disponibles dans Code_Aster.

3.2 METHODE = 'MULT_FRONT'

◇ RENUM = 'MD' / 'MDA' / 'METIS' (défaut)

Cet argument permet de renuméroter les nœuds du modèle :

/ 'MD' ("Minimum Degré") cette numérotation des nœuds minimise le remplissage de la matrice lors de sa factorisation.

/ 'MDA' ("Minimum Degré Approché") cette numérotation est en principe moins optimale que 'MD' en ce qui concerne le remplissage mais elle est plus économique à calculer. Elle est toutefois préférable à 'MD' pour les gros modèles ($\geq 50\,000$ dds).

/ 'METIS' Autre méthode de numérotation basée sur une dissection emboîtée. Cette méthode n'est possible que sur le serveur de calcul dédié au projet Aster sauf à installer soi-même l'exécutable METIS. Sur cette machine, c'est la méthode la plus efficace (en temps CPU et en mémoire).

◇ STOP_SINGULIER = 'OUI' (défaut) / 'NON' / 'DECOUPE'

Lorsqu'au terme de la factorisation, on constate qu'un terme diagonal d' est devenu très petit (par rapport à ce qu'il était avant la factorisation d), c'est que la matrice est (probablement) presque singulière.

Soit $n = \log \left| \frac{d}{d'} \right|$, ce rapport de magnitude indique que sur une équation (au moins) on a perdu n chiffres significatifs.

Si $n > \text{nprec}$ (mot clé NPREC ci-dessous), on considère que la matrice est singulière. Si l'utilisateur a indiqué : STOP_SINGULIER = 'OUI', le code s'arrête alors en ERREUR FATALE, si 'NON' l'exécution se poursuit avec émission d'une alarme, si 'DECOUPE' et qu'on est dans l'opérateur 'STAT_NON_LINE' ou 'DYNA_NON_LINE', le processus de découpe automatique du pas de temps est déclenché.

Remarque :

Toute perte importante de chiffres significatifs lors d'une factorisation est un indicateur d'un problème mal posé. Plusieurs causes sont possibles (liste non exhaustive) :

- des conditions aux limites de blocage de la structure insuffisantes,
- des relations linéaires redondantes,
- des données numériques très hétérogènes (termes de pénalisation trop grands), ...

◇ NPREC = nprec (défaut=8)

C'est le nombre qui sert à déterminer si la matrice est singulière (ou non) (cf. mot clé STOP_SINGULIER ci-dessus).

3.3 METHODE = 'LDLT'

◇ RENUM = 'SANS' / 'RCMK' (défaut)

Cet argument permet de renuméroter si on le désire les nœuds du modèle :

/ 'SANS' On garde l'ordre initial donné dans le fichier de maillage,

/ 'RCMK' "Reverse Cuthill-MacKee", cet algorithme de renumérotation est souvent efficace pour réduire la place nécessaire au stockage "ligne de ciel" de la matrice assemblée et pour réduire le temps nécessaire à la factorisation de la matrice.

◇ STOP_SINGULIER

Voir [§3.2].

◇ NPREC

Voir [§3.2].

3.4 METHODE = 'MUMPS'

Le solveur MUMPS développé par CERFACS-ENSEEIH-Parallab est un solveur direct de type multi-frontale, parallélisé (en MPI) et robuste. car il permet de pivoter les lignes et colonnes de la matrice lors de la factorisation numérique.

Bien qu'il soit un solveur direct, son usage dans *Code_Aster* s'apparente plus au solveur itératif GPCP : on ne peut pas l'utiliser dans les opérateurs modaux, ni en conjonction du mot clé STOP_SINGULIER='DECOUPE'. La raison en est que MUMPS (appelé par *Code_Aster*) ne donne pas directement d'information sur la « quasi-nullité » des pivots lors de la factorisation (comme le font MULT_FRONT et LDLT : voir le mot clé NPREC). On ne sait donc pas déterminer si une matrice est "proche" de la singularité lors de sa factorisation.

Mais par contre, MUMPS fournit une estimation de la qualité de la solution \mathbf{u} (cf. mot-clé RESI_RELA) du problème matriciel $\mathbf{Ku}=\mathbf{f}$ via les notions d'erreur relative *a priori* ('relative forward error') et d'erreur *a posteriori* ('backward error'). Cette 'backward error', $\eta(\mathbf{K},\mathbf{f})$, mesure le comportement de l'algorithme de résolution (quand tout va bien, ce réel est proche de la précision machine, soit 10^{-15} en double précision). MUMPS calcule aussi une estimation du conditionnement de la matrice, $\kappa(\mathbf{K})$, qui traduit le bon comportement du problème à résoudre (réel compris entre 10^4 pour un problème bien conditionné jusqu'à 10^{12} pour un très mal conditionné). Le produit des deux est un majorant de l'erreur relative sur la solution ('relative forward error') :

$$\frac{\|\delta\mathbf{u}\|}{\|\mathbf{u}\|} < C^{st} \kappa(\mathbf{K}).\eta(\mathbf{K},\mathbf{f})$$

En précisant une valeur strictement positive au mot-clé RESI_RELA (par ex. 10^{-6}), l'utilisateur indique qu'il souhaite tester la validité de la solution de chaque système linéaire résolu par MUMPS à l'aune de cette valeur. Si le produit $\kappa(\mathbf{K})\eta(\mathbf{K},\mathbf{f})$ est supérieur à RESI_RELA le code s'arrête en ERREUR_FATALE, en précisant la nature du problème et les valeurs incriminées. Avec l'affichage INFO=2, on détaille chacun des termes du produit : $\eta(\mathbf{K},\mathbf{f})$ et $\kappa(\mathbf{K})$.

Pour poursuivre le calcul, on peut alors :

- Augmenter la tolérance de RESI_RELA. Pour les problèmes mal conditionnés, une tolérance de 10^{-3} n'est pas rare. Mais elle doit être prise au sérieux car ce type de pathologie peut sérieusement perturber un calcul (Cf. remarque suivante sur le conditionnement).
- Si c'est la 'backward error' qui est trop importante : il est conseillé de modifier l'algorithme de résolution. C'est-à-dire, dans notre cas, de jouer sur les paramètres de lancement de MUMPS (TYPE_RESOL, PRETRAITEMENTS, PARALLELISME).
- Si c'est le conditionnement de l'opérateur qui est en cause, il est conseillé d'équilibrer les termes de la matrice, en dehors de MUMPS ou via MUMPS (PRETRAITEMENTS='OUI'), ou de changer la formulation du problème.

◇ TYPE_RESOL = 'NONSYM' / 'SYMGEN' / 'SYMDEF' / 'AUTO' (défaut)

Ce mot clé permet de choisir le type de résolution MUMPS :

- / 'NONSYM' doit être choisi pour les matrices non symétriques.
- / 'SYMGEN' doit être choisi pour les matrices symétriques non définies positives. C'est la cas le plus général dans *Code_Aster* du fait de la dualisation des conditions aux limites par des coefficients de Lagrange.
- / 'SYMDEF' peut être choisi pour les matrices symétriques définies positives. Il n'y a pas de pivotage.
- / 'AUTO' , le code choisira 'NONSYM' pour les matrices non symétriques et 'SYMGEN' pour les matrices symétriques.

Il n'est pas interdit de choisir 'NONSYM' pour une matrice symétrique. Cela doublera probablement le coût de calcul mais cette option donne à MUMPS plus de possibilités algorithmiques (pivotage, scaling...).

◇ PRETRAITEMENTS = 'SANS' / 'AUTO' (défaut)

Ce mot clé permet de contrôler le type de prétraitements à opérer au système pour améliorer sa résolution (diverses stratégies d'équilibrage des termes de la matrice et de permutation de ses lignes et de ses colonnes):

Titre : Mot clé SOLVEUR

Date : 22/02/08

Auteur(s) : O.BOITEAU

Clé : U4.50.01-G1 Page : 10/22

/ 'SANS' : pas de prétraitement.

/ 'AUTO' : MUMPS choisit la meilleure combinaison de paramètres en fonction du problème (TYPE_RESOL).

Remarque :

- En mode parallèle distribué (PARALLELISME='DISTRIBUE_**'), MUMPS n'opère quasiment pas de prétraitements. Tout se passe comme si PRETRAITEMENT était fixé à 'SANS'.

◇ RENUM = 'AMD' / 'AMF' / 'QAMD' / 'PORD' / 'METIS' / 'AUTO' (défaut)

Ce mot clé permet de contrôler la renumérotation et l'ordre d'élimination. Les différents outils proposés ('AMD', 'AMF', 'PORD', 'METIS', 'QAMD') ne sont pas forcément tous disponibles. Cela dépend de l'installation de MUMPS/Code_Aster (fichier config.txt).

/ 'AMD' : 'Approximate Minimum Degree' (Minimum Degré Approché ; Disponible aussi avec METHODE='MULT_FRONT').

/ 'AMF' : 'Approximate Minimum Fill' (Remplissage Minimum Approché).

/ 'QAMD' : variante de 'AMD' (détection automatique de ligne quasi-dense).

/ 'PORD' : outil externe de renumérotation (cf. J.Schulze, distribué avec MUMPS).

/ 'METIS' : outil externe de renumérotation (cf. G.Karypis et V.Kumar ; Disponible aussi avec METHODE='MULT_FRONT').

/ 'AUTO' : MUMPS choisit la meilleure combinaison de paramètres en fonction du problème et des packages disponibles. Si l'utilisateur spécifie un renuméroteur particulier et que ce dernier n'est pas disponible, le solveur choisi le plus adéquat dans la liste des disponibles et une ALARME est émise.

◇ RESI_RELA = resi (défaut=-1.d0)

En précisant une valeur strictement positive à ce mot-clé (par ex. 10^{-6}), l'utilisateur indique qu'il souhaite tester la validité de la solution de chaque système linéaire résolu par MUMPS à l'aune de cette valeur. Cette démarche prudente est conseillée (malgré un surcoût de l'ordre de 10%) lorsque la solution n'est pas elle-même corrigée par un autre processus algorithmique (algorithme de Newton, Schéma de Newmark...), bref dans les opérateurs linéaires THER_LINEAIRE et MECA_STATIQUE. Si l'erreur relative sur la solution estimée par MUMPS est supérieure à resi le code s'arrête en ERREUR_FATALE, en précisant la nature du problème et les valeurs incriminées.

◇ PCENT_PIVOT = pcent (défaut=10%)

Ce mot-clé permet de choisir un pourcentage de mémoire que MUMPS réservera en début de calcul pour ses pivotages. La valeur par défaut est de 10% qui correspond à un nombre de pivotages raisonnable. Si par exemple MUMPS estime à 100 la place nécessaire à une factorisation sans pivotage, il allouera *in fine* 120. Par la suite, si l'espace mémoire requis par les pivotages s'avère plus important, la place mémoire allouée sera insuffisante et le code s'arrêtera en ERREUR_FATALE en demandant d'augmenter ce critère. Une valeur dépassant les 50% doit rester exceptionnelle.

◇ ELIM_LAGR2 = 'OUI' (défaut)/'NON'

Historiquement, les solveurs linéaires directs de Code_Aster ('MULT_FRONT' et 'LDLT') ne disposaient pas d'algorithme de pivotage (qui cherche à éviter les accumulations d'erreurs d'arrondis par division par des termes très petits). Pour contourner ce problème, la prise en compte des conditions limites par des Lagranges (AFFE_CHAR_MECA/THER...) a été modifiée en introduisant des doubles Lagranges. D'où un surcoût mémoire et calculs.

Comme MUMPS dispose de facultés de pivotage, ce choix de dualisation des conditions limites peut être remis en cause. En initialisant ce mot-clé à 'OUI', on ne tient plus compte que d'un Lagrange, l'autre étant spectateur. *A contrario*, avec la valeur 'NON', MUMPS reçoit les matrices dualisées usuelles.

◇ PARALLELISME = 'CENTRALISE' (défaut)/'DISTRIBUE_MAILLE' / 'DISTRIBUE_SD'

MUMPS est un solveur linéaire parallélisé. Ce parallélisme peut être activé de plusieurs manières notamment suivant les aspects séquentiels ou parallèles du code qui l'utilise. Ainsi, ce parallélisme peut être limité aux flots de données/traitements internes à MUMPS, ou, *a contrario* s'intégrer à un flot de données/traitements parallèles déjà organisés en amont du solveur, dès les

phases de calculs élémentaires de *Code_Aster*. Le premier mode ('CENTRALISE') a pour lui la robustesse et un plus large périmètre d'utilisation, le second ('DISTRIBUE_MAILLE' et 'DISTRIBUE_SD') est moins générique mais plus efficace.

Car les phases souvent les plus coûteuses en temps CPU d'une simulation sont : la construction du système linéaire (purement *Code_Aster*, découpée en trois postes : factorisation symbolique, calculs élémentaires et assemblages matriciels/vectoriels) et sa résolution (dans MUMPS : analyse, factorisation numérique et descente-remontée). Le premier mode de parallélisation ne profite que du parallélisme des étapes 2 et 3 de MUMPS, alors que les deux autres parallélisent aussi les calculs élémentaires et les assemblages de *Code_Aster*.

Les deux stratégies 'DISTRIBUE_**' ont besoin d'organiser un flot de « données/traitements » parallèle en distribuant les données initiales. *Code_Aster* étant un code éléments finis, la distribution naturelle de données est celle par groupe de mailles. En début d'opérateur, ces stratégies distribuent donc les mailles du modèle aux processeurs. Soit par paquets de mailles contiguës ('DISTRIBUE_MAILLE') en interne de l'opérateur concerné, soit par sous-domaines ('DISTRIBUE_SD') via une décomposition du modèle en sous-domaines construites en amont de l'opérateur (cf. DEFI_PART_FETI/OPS). Dans les deux cas, les processeurs ne remplissent que les morceaux de matrices et de second membres qui leurs correspondent puis les transmettent à MUMPS. Ce dernier les assemble en interne avant d'effectuer la résolution proprement dite.

- / 'CENTRALISE' : le parallélisme ne commence qu'au niveau de MUMPS. Chaque processeur construit et fournit à MUMPS l'intégralité du système à résoudre (matrice, second membre).
- / 'DISTRIBUE_MAILLE' : le parallélisme débute, en amont de MUMPS, dès la phase de calcul élémentaire de *Code_Aster*. Chaque processeur alloue la matrice entière, n'en remplit que les termes *ad hoc* et fournit les valeurs non nulles à MUMPS. Ce dernier va les regrouper avant d'effectuer la résolution parallélisée du système. La distribution des mailles du modèle est initiée en début d'opérateur, par paquets de mailles contiguës. Par exemple avec un modèle comportant 100 mailles et pour un calcul sur 2 processeurs : les mailles 1 à 50 sont assignées au premier processeur, les autres mailles au second.
- / 'DISTRIBUE_SD' : parallélisme similaire à l'option 'DISTRIBUE_MAILLE', mais cette fois la distribution initiale des mailles en début d'opérateur se base sur une décomposition en sous-domaines construite en amont, via les opérateurs DEFI_PART_FETI/OPS. Par exemple, avec une structure de données SD_FETI comportant 5 sous-domaines et un calcul sur 2 processeurs : les mailles des sous-domaines 1 et 2 sont assignées au premier processeur, les mailles des sous-domaines restant au second.

Remarques :

- En terme d'occupation mémoire les trois stratégies sont équivalentes car, pour des raisons de maintenance et de lisibilité, on a pas cherché à « retailer » les objets FORTRAN manipulés au plus juste.
- La distribution par mailles est très simple mais peut conduire à des déséquilibres de charge car elle ne tient pas compte des mailles spectatrices, des mailles de peau (cf. figure 3.4-a sur un exemple comportant 8 mailles volumiques et 4 mailles de peau), de zones particulières (non linéarités...). La distribution par sous-domaines est plus souple et peut s'avérer plus efficace en permettant d'adapter son flot de données à sa simulation.
- Une autre cause de déséquilibre peut provenir des conditions de Dirichlet par dualisation (DDL_IMPO, LIAISON_***...). Par soucis de robustesse, leur traitement est affecté seulement au processeur maître. Cette surcharge de travail, souvent négligeable, introduit cependant dans certains cas, un déséquilibre plus marqué. L'utilisateur peut le compenser en renseignant un des mot-clés CHARGE_PROCO_*. Ce traitement différencié concerne en fait tous les cas de figures impliquant des mailles dites « tardives » (Dirichlet via des Lagranges, mais aussi force nodale, contact méthode continue...).
- Pour l'instant, la distribution par sous-domaines est basée sur un partitionnement au sens FETI générés par les opérateurs DEFI_PART_*. Ces derniers génèrent en fait une structure de données de type SD_FETI (sous-domaines, interface, charges projetées...) trop riche pour les seuls besoins de MUMPS. Pour éviter des surcoûts calculs et des failles de robustesse, on peut juste se contenter (pour des contingences purement informatiques) de préciser dans la liste des charges fournie à ces opérateurs de partitionnement (mot-clé EXCIT), une seule charge, voire une

charge « bidon », n'impliquant si possible aucun Lagrange (DDL_IMPO, LIAISON_*...). Cette dernière ne servira alors pas forcément dans le calcul effectif.

- En mode distribué, chaque processeur ne manipule que des matrices partiellement remplies. Par contre, afin d'éviter d'introduire trop de communications MPI dans le code (critères d'arrêt, résidu...), ce scénario n'a pas été retenu pour les seconds membres. Leur construction est bien parallélisé, mais en fin d'assemblage, les contributions de tous les processeurs sont sommées et envoyées à tous. Ainsi tous les processeurs connaissent entièrement les vecteurs impliqués dans le calcul.

Pour l'instant, on réutilise les op. de partitionnement de FETI.

```

PART=DEFI_PART_FETI(
  MODELE=MO,
  NB_PART=4,
  EXCIT=( _F( CHARGE=SYMETRI2, , , , ,
  METHODE='KMETIS' )

```

4 sous-domaines, donc parallélisme sur maximum 4 proc. MPI.

```

RESU2=STAT_NON_LINE(
  MODELE=MO,
  SOLVEUR=_F(METHODE='MUMPS',
  PARALLELISME='DISTRIBUE_MAILLE',
  CHARGE_PROC0_MA=... (optionnel))
...);

```

```

RESU3=STAT_NON_LINE(
  MODELE=MO,
  SOLVEUR=_F(METHODE='MUMPS',
  PARALLELISME='DISTRIBUE_SD',
  PARTITION=PART, (obligatoire)
  CHARGE_PROC0_SD=... (optionnel))
...);

```

On précise le concept utilisateur contenant le partitionnement. Ici **PART**.



Figure 3.4-a : Mode de distribution de MUMPS sur 2 processeurs : par paquets de mailles contigües ou par sous-domaines.
Extrait du cas-test mumps02a.

◇ PARTITION = sdfeti

Nom utilisateur de l'objet SD_FETI décrivant le partitionnement en sous-domaines. Il est généré par un appel préalable aux opérateurs DEFI_PART_FETI/OPS [U4.23.05]. Paramètre utilisé uniquement (et indispensable) lorsqu'on active un parallélisme distribué par sous-domaines : PARALLELISME='DISTRIBUE_SD'.

◇ CHARGE_PROC0_MA = chargema (défaut=0%)

Pourcentage de charge (par rapport au nombre de mailles prévu normalement par l'heuristique) attribué au processeur maître. Dans l'exemple 3.4-a, si chargema=50%, le processeur maître n'est plus responsable que de 3 mailles. A contrario, on peut aussi le surcharger par ce biais. Paramètre utilisé uniquement lorsqu'on active un parallélisme distribué par paquets de mailles contigües : PARALLELISME='DISTRIBUE_MA'.

◇ CHARGE_PROC0_SD = chargesd (défaut=0)

Nombre de sous-domaines (par rapport au nombre prévu normalement par l'heuristique) attribué au processeur maître. Dans l'exemple 3.4-a, si chargesd=1, le processeur maître ne sera plus responsable que de 1 sous-domaine (2 HEXA + 1 QUAD). A contrario, on peut aussi le surcharger par ce biais. Paramètre utilisé uniquement lorsqu'on active un parallélisme distribué par sous-domaines : PARALLELISME='DISTRIBUE_SD'.

**Figure 3.4-b : Flots de données/traitements parallèles
de MUMPS centralisé versus MUMPS distribué.**

Titre : *Mot clé SOLVEUR*

Date : 22/02/08

Auteur(s) : *O.BOITEAU*

Clé : U4.50.01-G1 Page : 14/22

COPYRIGHT de MUMPS

COPYRIGHT (c) 1996-2003 P. R. Amestoy, I. S. Duff, J. Koster,
J.-Y. L'Excellent

CERFACS , Toulouse (France) (<http://www.cerfacs.fr>)
ENSEEIH-IRIT, Toulouse (France) (<http://www.enseeiht.fr>)
INRIA (France) (<http://www.inria.fr>)
PARALLAB , Bergen (Norway) (<http://www.parallab.uib.no>)
All rights reserved.

Your use or distribution of the package implies that you agree with this License. Up-to-date copies of the MUMPS package can be obtained from the Web page <http://www.enseeiht.fr/apo/MUMPS/>. This package is provided to you free of charge. It was initially based on public domain software developed during the European Esprit IV project PARASOL (1996-1999). THIS MATERIAL IS PROVIDED AS IS, WITH ABSOLUTELY NO WARRANTY EXPRESSED OR IMPLIED. ANY USE IS AT YOUR OWN RISK.

Permission is hereby granted to use or copy this package provided that the Copyright and this License is retained on all copies and that the package is used under the same terms and conditions. User documentation of any code that uses this software should include this complete Copyright notice and this License.

You can modify this code but, at no time shall the right or title to all or any part of this package pass to you. All information relating to any alteration or addition made to this package for the purposes of extending the capabilities or enhancing the performance of this package shall be made available free of charge to the authors for any purpose.

You shall acknowledge (using references [1] and [2]) the contribution of this package in any publication of material dependent upon the use of the package. You shall use reasonable endeavours to notify the authors of the package of this publication.

[1] P. R. Amestoy, I. S. Duff and J.-Y. L'Excellent (1998), Multifrontal parallel distributed symmetric and unsymmetric solvers, in Comput. Methods in Appl. Mech. Eng., 184, 501-520 (2000). An early version appeared as a Technical Report ENSEEIH-IRIT (1998) and is available at <http://www.enseeiht.fr/apo/MUMPS/>.
[2] P. R. Amestoy, I. S. Duff, J. Koster and J.-Y. L'Excellent, A fully asynchronous multifrontal solver using distributed dynamic scheduling, SIAM Journal of Matrix Analysis and Applications, Vol 23, No 1, pp 15-41 (2001). An early version appeared as a Technical Report ENSEEIH-IRIT, RT/APO/99/2 (1999) and is available at <http://www.enseeiht.fr/apo/MUMPS/>.

None of the text from the Copyright notice up to and including this line shall be removed or altered in any way.

3.5 METHODE : 'GCPC'

◇ PRE_COND = 'LDLT_INC' (défaut)

Méthode de préconditionnement : la matrice de préconditionnement est obtenue par une décomposition \mathbf{LDL}^T incomplète de la matrice assemblée.

◇ NIVE_REMPLISSAGE = niv (défaut=0)

La matrice de préconditionnement (**P**) utilisée pour accélérer la convergence du gradient conjugué est obtenue en factorisant de façon plus ou moins complète la matrice initiale (**K**).

Si niv = 0

P a le même stockage que **K**. La factorisation est incomplète car on n'utilise pour les calculs que les termes que l'on peut stocker dans **K**. **P** représente donc une approximation (médiocre) de \mathbf{K}^{-1} ; son stockage est donc plus raisonnable.

Si niv = 1

On stocke dans **P** en plus des termes qui avaient leur place dans le stockage initial, les "descendants" de première génération des termes initiaux. En effet lors de la factorisation, un terme nul dans **K** peut devenir non nul dans **P**. On obtient ainsi le remplissage de niveau 1.

Si niv = 2, ...

Le même procédé est repris : la matrice **P** remplie au niveau niv-1 crée les termes de la matrice **P** au niveau niv.

Plus niv est grand, plus la matrice **P** est proche de \mathbf{K}^{-1} et donc plus le gradient conjugué converge vite (en nombre d'itérations). En revanche, plus niv est grand plus le stockage de **P** devient volumineux (en mémoire et sur disque) et plus les itérations sont coûteuses en CPU.

Les premiers essais ont montré (approximativement) que la taille de **P** valait :

- 1*taille(**K**) pour niv = 0
- 3,5*taille(**K**) pour niv = 1
- 7,5*taille(**K**) pour niv = 2

Notre expérience de ce mot clé est encore limitée et nous conseillons d'utiliser la valeur par défaut (niv = 0). Si niv = 0 ne permet pas au gradient conjugué de converger, on essaiera successivement les valeurs niv = 1, 2, 3...

◇ NMAX_ITER = niter (défaut=0)

Nombre d'itérations maximum de l'algorithme de résolution itératif. Si niter = 0 alors le nombre maximum d'itérations est calculé comme suit :

niter = nequ/2 où nequ est le nombre d'équations du système.

◇ RESI_RELA = resi (défaut=10⁻⁶)

Critère de convergence de l'algorithme. C'est un critère relatif sur le résidu :

$$\frac{\|\mathbf{r}_m\|}{\|\mathbf{f}\|} \leq resi$$

\mathbf{r}_m est le résidu à l'itération m

\mathbf{f} est le second membre et $\|\cdot\|$ la norme euclidienne

Remarque :

- Pour reprendre la terminologie des estimateurs de qualité introduite pour MUMPS (cf. §3.4), ce critère d'arrêt peut être interprété comme une 'backward error' particulière $\eta(\mathbf{K}, \mathbf{f})$.

3.6 METHODE : 'FETI'

♦ PARTITION = sdfeti

Nom utilisateur de l'objet SD_FETI décrivant le partitionnement en sous-domaines. Il est généré par un appel préalable aux opérateurs DEFI_PART_FETI/OPS [U4.23.05].

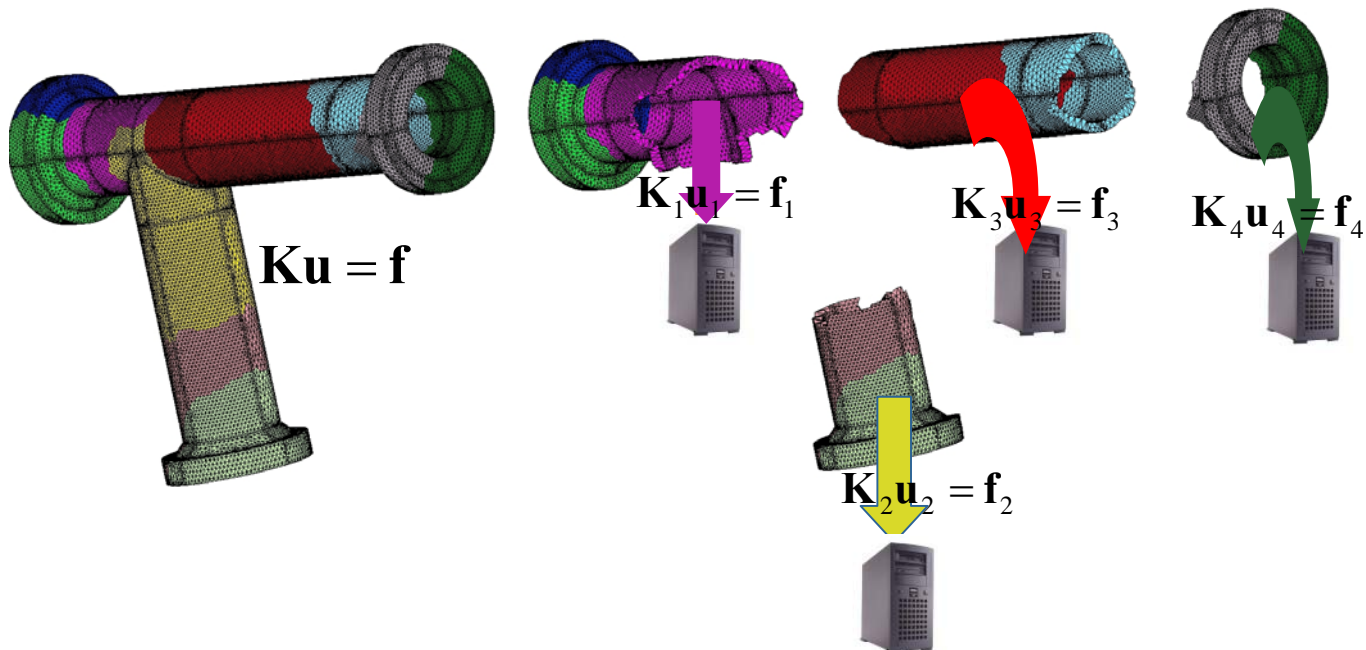


Figure 3.6-a : Principe de la décomposition de domaines illustré sur une zone de mélange d'un circuit RRA.

de type FETI

♦ NMAX_ITER = niter (défaut=0)

Nombre d'itérations maximum du GCPPC résolvant le problème d'interface. Si niter = 0 alors le nombre maximum d'itérations est calculé comme suit :

$$niter = \max(nbi/100, 10) \text{ où } nb_i \text{ le nombre d'inconnues du problème d'interface.}$$

♦ REAC_RESI = nreac (défaut=0)

Fréquence de réactualisation du calcul du résidu. Valeur conseillée : 10 ou 20.

♦ RESI_RELA = resi (défaut=10⁻⁶)

Critère de convergence de l'algorithme : c'est un critère relatif sur le résidu projeté du problème d'interface

$$\frac{\|Pr_m\|}{\|b\|} \leq resi$$

r_m est le résidu à l'itération m

P l'opérateur de projection

b est le second membre et $\| \cdot \|$ la norme euclidienne

♦ PRE_COND = 'SANS' / 'LUMPE' (défaut)

Cet argument permet de choisir le type de préconditionneur pour le GCPPC :

/ 'SANS' Pas de préconditionnement.

/ 'LUMPE' Préconditionnement lumpé.

Normalement le préconditionneur lumpé conduit à un gain en itérations et en CPU, sans surcoût mémoire.

♦ SCALING = 'SANS' / 'MULT' (défaut)

Titre : *Mot clé SOLVEUR*
Auteur(s) : **O.BOITEAU**

Date : 22/02/08
Clé : U4.50.01-G1 Page : 17/22

Cet argument permet de choisir le type de scaling (mise à l'échelle) adopté pour le préconditionneur. Il n'est donc pris en compte que si `PRE_COND` est différent de 'SANS'.

/ 'SANS' Pas de phase de scaling.

/ 'MULT' Mise à l'échelle par la multiplicité des nœuds d'interface.

Normalement la phase de scaling conduit à un gain en itérations et en CPU, sans surcoût mémoire. Surtout lorsque le partitionnement produit beaucoup de points de jonction (points appartenant à plus de deux sous-domaines).

◇ `TYPE_REORTHO_DD = 'SANS' / 'GS' / 'GSM' (défaut) / 'IGSM'`

Cet argument permet de choisir le type de réorthogonalisation des directions de descente (au sein d'une résolution de système linéaire ou entre différentes résolutions cf. `ACCELERATION_SM`). Il est lié au paramètre `NB_REORTHO_DD`.

/ 'SANS' Pas de réorthogonalisation des méthodes de descente.

/ 'GS' Réorthogonalisation de Gram-Schmidt.

/ 'GSM' Réorthogonalisation de Gram-Schmidt Modifié.

/ 'IGSM' Réorthogonalisation de Gram-Schmidt Modifié Itératif.

Cette phase permet de lutter contre la propension des directions de descente du GCPPC à perdre leur orthogonalité. En théorie, IGSM est meilleure que GSM qui est lui même supérieur à GS. En pratique, le meilleur compromis « surcoût calcul/qualité d'orthogonalité » est souvent réalisé par GSM.

◇ `NB_REORTHO_DD = nb_reortho (défaut=0)`

Nombre de directions de descente initiales utilisées dans la phase de réorthogonalisation. En principe, plus il est grand, meilleure est la convergence, mais plus grand est aussi le surcoût calcul et mémoire. Il faut donc trouver un compromis entre ces éléments.

Si `nb_reortho = 0` alors ce nombre est calculé comme suit :

$$nb_reortho = \max(niter/10, 5)$$
 où `niter` le nombre maximal d'itérations définies ci-dessus.

◇ `RENUM`

Voir [§3.2].

◇ `STOP_SINGULIER`

Voir [§3.2].

◇ `NPREC`

Voir [§3.2].

◇ `VERIF_SDFETI = 'OUI' (défaut) / 'NON'`

On comptabilise les incohérences en terme de nom de modèle et de noms de chargement, entre le paramétrage de l'opérateur appelant le mot-clé `SOLVEUR` et celui fourni à l'opérateur de partitionnement qui reste stocké dans la `SD_FETI`. Il faut que les noms de modèles soient identiques et que la liste des chargements de l'opérateur appelant soit égale à celle de `DEFI_PART_OPS`. Si ce n'est pas le cas et si `VERIF_SDFETI = 'OUI'`, on s'arrête en `ERREUR_FATALE`, sinon on émet une `ALARME`.

◇ `TEST_CONTINU = test_continu (défaut=10-6)`

Critère du test de continuité à l'interface : c'est un critère relatif sur les valeurs (non nulles) des inconnues aux interface. Si on est en dessus du critère, il y'a émission d'une `ALARME`. Ce critère est pour l'instant désactivé.

◇ `STOCKAGE_GI = 'CAL' (défaut) / 'OUI' / 'NON'`

Lorsque le nombre de sous-domaines augmente, un objet devient proéminent, c'est G_I la matrice des traces des modes de corps rigides sur l'interface. Elle est utilisée dans la phase de projection, c'est-à-dire $10 + \text{nombre_itérations_FETI} * 4$. Pour permettre à l'utilisateur d'adapter le compromis « taille mémoire/temps CPU », son stockage est paramétrable :

/ 'OUI', elle est calculée et stockée une fois pour toute. Cela nécessite plus de mémoire mais moins de temps calcul lorsqu'on s'en sert.
/ 'NON', c'est l'inverse, elle est recalculée à chaque fois que cela est nécessaire.
/ 'CAL', le choix 'OUI' ou 'NON' va être calculé automatiquement. Si la taille de la matrice est inférieure à la taille moyenne des matrices de rigidité locales, on stocke ('OUI'), sinon, on recalcule ('NON').

◇ INFO_FETI = info_feti (**défaut(1:15)='FFFFFFFFFFFFFFFF'**)

Chaîne de caractères permettant de paramétrer les affichages de l'algorithme FETI, de ses pré et post-traitements ainsi que des tests de cohérence. Ce monitoring est indépendant du mot-clé INFO. Etant souvent très verbeux et parfois coûteux en mémoire et en CPU, il doit être utilisé de préférence sur des petits cas et plutôt pour des activités de développements.

- Si INFO_FETI(1:1)='T' : déroulement général de l'algorithme.
- Si INFO_FETI(2:2)='T' : contenu des structures de données hors CHAM_NO et MATR_ASSE.
- Si INFO_FETI(3:3)='T' : contenu des structures de données CHAM_NO et MATR_ASSE.
- Si INFO_FETI(4:4)='T' : affichage de variables intermédiaires.
- Si INFO_FETI(5:5)='T' : détails des routines d'assemblages.
- Si INFO_FETI(6:6)='T' : tests de validité des modes de corps rigides.
- Si INFO_FETI(7:7)='T' : test de la définie-positivité de l'opérateur d'interface. On calcule alors les $n_{max_freq} = \min(n_{bi}-2, n_{iter})$ valeurs propres via l'algorithme IRAM[R5.01.01] en projetant sur un espace de taille $dim_sous_espace = \min(n_{bi}, nb_reortho)$. Option licite uniquement en séquentiel.
- Si INFO_FETI(8:8)='T' : test des orthogonalités du GCPPC.
- Si INFO_FETI(9:9)='T' : profiling (temps CPU + système) des différentes étapes de résolution du problème d'interface FETI, le GCPPC, (projection, opérateur FETI, réorthogonalisation, redémarrages...).
- Si INFO_FETI(10:10)='T' : affichages dédiés aux parallélisme MPI + profiling des surcoûts dus aux communications.
- Si INFO_FETI(11:11)='T' : affichages généraux (taille de l'interface, des matrices de rigidité locales et de leurs factorisées, nombre total de modes rigides...) et profiling des étapes amonts du solveur FETI (temps CPU + système des phases de calculs élémentaires, d'assemblages, de factorisations symbolique et numérique) détaillé par sous-domaine ou par processeur.
- Si INFO_FETI(12:12)='T' : tests directs et indirects de la cohérence des objets JEVEUX de description de l'interface. Lorsqu'on veut faire un calcul FETI il est conseillé de valider son découpage en sous-domaines en utilisant ce test avec MECA_STATIQUE. Mettre un RESI_RELA assez dur (par ex. 10^{-12}) pour obliger l'algorithme à faire quelques itérations. Mettre de préférence qu'un seul chargement dans EXCIT et ne comportant pas de Lagrange (une pression par exemple).

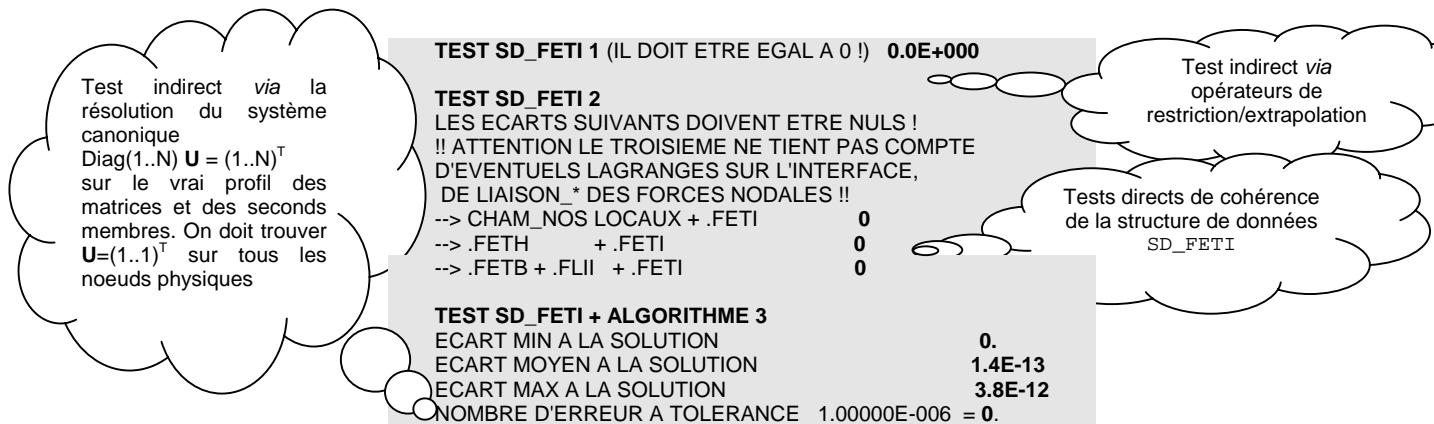


Figure 3.5-b : Extrait du fichier de message (.mess) lorsqu'on active ce test.

- Si INFO_FETI(13:13)= 'T': sortie fichier (fort.17) des évolutions en fonction du numéro d'itération des résidus projetés absolu et relatif.
- Si INFO_FETI(14:14)= 'T': sortie fichier (fort.18) des matrices de rigidité locales, des seconds membres locaux, des solutions locales et de la solution globale. Chaque processeur remplit son fichier avec les données afférentes aux sous-domaines dont il a la charge.
- Si INFO_FETI(15:15)= 'T': sortie fichier (fort.18) des matrices de rigidité locales et de leurs éventuels modes de corps rigides. Chaque processeur remplit son fichier avec les données afférentes aux sous-domaines dont il a la charge.

◇ NB_SD_PROC0 = nb_sdproc0 (défaut=0)

Paramètre utilisé en mode parallèle MPI, permettant d'attribuer un nombre de sous-domaines arbitraire au processeur 0 (le « maître »). Ce nombre peut ainsi être inférieur à celui qui lui serait attribué par la procédure de répartition automatique « sous-domaines/processeurs ». Cela permet de le soulager, en CPU et en espace mémoire, par rapport aux autres processeurs car il doit gérer des étapes supplémentaires et des objets JEVEUX potentiellement volumineux (phase de réorthogonalisation, projections du problème grossier...).

Il n'est actif que si il est licite : nb_sdproc0>0 et nb_sdproc0<nbsd-nbproc+1 (nbproc, nombre de processeurs et nbsd, nombre de sous-domaines).

◇ ACCELERATION_SM = 'OUI' (défaut)/'NON'

Cet argument permet d'activer la phase d'accélération d'un problème avec multiples seconds membres (par exemple, un calcul d'élasticité avec des chargements thermiques dépendant du temps). Lorsqu'il est activé, le GCPPC du solveur FETI ne va pas démarrer son processus en partant de zéro, mais va au contraire s'appuyer sur une information *a priori*, celle des directions de descente stockées au nb_reortho_inst pas de temps précédents. On va donc gagner beaucoup en nombre d'itérations et donc en CPU, en concédant assez peu en mémoire (si l'interface est faible devant la taille du problème).

/'OUI' Accélération activée si les conditions sont réunies (voir plus bas).
 /'NON' Accélération désactivée.

Cet argument est lié au paramètre NB_REORTHO_INST et n'est activé que si le problème est une succession de systèmes linéaires à seconds membres différents et si la réorthogonalisation des directions de descente, au sein de chaque pas de temps, est activée (TYPE_REORTHO_DD différent de 'SANS').

◇ NB_REORTHO_INST = nb_reortho_inst (défaut=0)

A un pas de temps donné, c'est le nombre de pas de temps précédents dont on va utiliser les directions de descente pour la procédure d'accélération. En principe, plus il est grand, meilleure est la convergence, mais plus grand est aussi le surcoût calcul et mémoire. Il faut donc trouver un compromis entre ces éléments.

Si nb_reortho_inst = 0 alors ce nombre est calculé comme suit :

$$\text{nb_reortho_inst} = \max(\text{nb_pas_temps} / 5, 5)$$

où nb_pas_temps est le nombre de pas de temps du problème.

Titre : *Mot clé SOLVEUR*

Date : 22/02/08

Auteur(s) : **O.BOITEAU**

Clé : U4.50.01-G1 Page : 20/22

Et si, au pas de temps `num_pas_temps`, `nb_reortho_inst` est supérieur au nombre de pas de temps précédents disponibles (par ex. au 5^{ième} pas de temps on ne peut utiliser que les 4 pas de temps antérieurs) on le fixe dynamiquement à cette valeur :

$$\text{nb_reortho_inst} = \text{num_pas_temps} - 1.$$

3.7 Mot clé SYME

◇ SYME = 'OUI' / 'NON' (défaut)

Si la matrice du système linéaire **K** est non-symétrique, le mot clé SYME = 'OUI' permet de symétriser cette matrice avant la résolution du système. La matrice alors est remplacée par

$$\mathbf{K}' = \frac{1}{2}(\mathbf{K} + \mathbf{K}^T).$$

/ 'OUI' Symétrisation de la matrice activée.
/ 'NON' Symétrisation désactivée.

Attention:

*La symétrisation de la matrice **K** conduit donc à résoudre un autre problème que celui que l'on cherche à résoudre ! En réalité, cette possibilité (SYME = 'OUI') n'est utile que dans les commandes non-linéaires (comme STAT_NON_LINE par exemple), pour lesquelles la convergence vers la solution est obtenue par itérations successives. Chaque itéré est obtenu par "estimation" et l'on vérifie ensuite qu'il est "solution". Dans ce cas, une erreur légère sur les itérés n'empêche pas de converger vers la bonne solution. L'intérêt de ce mot clé est de gagner du temps lors de la résolution des systèmes linéaires. Le tout est de savoir si la symétrisation perturbe beaucoup (ou non) la solution du système linéaire ? On peut citer (par exemple) le cas des modèles 3D (ou coque) avec pression suiveuse pour lesquels la symétrisation fait gagner beaucoup de temps.*

4 Exemples

4.1 Solveur par défaut

Il n'y a rien à écrire ! Mais on peut aussi écrire : `SOLVEUR=_F ()`

4.2 Gradient conjugué

On veut utiliser le gradient conjugué. On pense que la convergence sera plus efficace si l'on autorise un pré-conditionnement plus poussé (`NIVE_REEMPLISSAGE=1`).

```
SOLVEUR=_F(METHODE = 'GCPC' , NIVE_REEMPLISSAGE=1 , )
```