Mot clé SOLVEUR

1 But

Ce mot clé facteur se retrouve dans un certain nombre de commandes conduisant à la résolution de systèmes linéaires. Ces «**solveurs linéaires**» sont en fait omniprésents dans le déroulement des opérateurs *Code_Aster* car ils sont souvent enfouis au plus profond d'autres algorithmes numériques: schéma non linéaire, intégration en temps, analyse modal etc. Ils en consomment la majeure partie du temps CPU et de la mémoire.
Ce mot-clé permet de choisir entre les trois classes de solveurs: **les directs, les itératifs et les hybrides**.
Concernant les directs, on dispose du classique algorithme de "Gauss" (METHODE='LDLT'), d'une factorisation multifrontale «maison» ('MULT_FRONT') et d'une externe ('MUMPS'). Seule la dernière propose des fonctionnalités avancées: équilibrage, pivotage, raffinement itératif, calculs d'erreurs etc.
Pour les itératifs, on peut faire appel à un gradient conjugué «maison» (préconditionné par un Cholesky Incomplet ILU(k) *via* le mot-clé 'GCPC') ou à certains outils de la librairie PETSC ('PESTC').
Lorsqu'on mixte les deux premières approches, on fait de l'hybride. C'est le cas du solveur multidomaines FETI ('FETI').
D'un point de vue HPC, seules MULT_FRONT, MUMPS et FETI sont parallélisées. Le premier en OpenMP, les deux autres en MPI.

Pour chaque type de solveur, ce document décrit leurs paramètres accessibles. 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 conseil préalable de l'équipe de développement.

Table des Matières

2 Syntaxe

```
◊  SOLVEUR = _F (

   #Solveur direct «maison» de type multifrontal:
   /METHODE='MULT_FRONT'  ,                                   [DEFAUT]

      #Paramètre numérique
            ◊   RENUM          =/'METIS',                     [DEFAUT]
                                  /'MD',
                                  /'MDA'.
      #Paramètres fonctionnels
            ◊  STOP_SINGULIER=/'OUI',                         [DEFAUT]
                              /'NON'.
            ◊  NPREC=/8,                                      [DEFAUT]
                     /nprec.                                  [I]


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

      #Paramètre numérique
            ◊  RENUM=/'RCMK',                                 [DEFAUT]
                     /'SANS'.
      #Paramètres fonctionnels
            ◊  STOP_SINGULIER=/'OUI',                         [DEFAUT]
                              /'NON'.
            ◊  NPREC=/8,                                      [DEFAUT]
                     /nprec.                                  [I]


   #Solveur direct de type multifrontal basé sur le produit externe MUMPS:
   /METHODE='MUMPS' ,

      #Paramètres fonctionnels
            ◊  TYPE_RESOL=/'AUTO',                            [DEFAUT]
                          /'NONSYM',
                          /'SYMGEN',
                          /'SYMDEF'.
            ◊  PCENT_PIVOT=/10,                               [DEFAUT]
                           /pcent.                            [R]
            ◊  ELIM_LAGR2=/'OUI',                             [DEFAUT]
                          /'NON'.
            ◊  RESI_RELA=/-1.d0,                              [DEFAUT]
                         /resi.                               [R]
      #Paramètres numériques
            ◊  PRETRAITEMENTS=/'AUTO',                        [DEFAUT]
                              /'SANS'.
             ◊  RENUM=/'AUTO',                                [DEFAUT]
                     /'AMD',
                     /'AMF',
                     /'QAMD',
                     /'PORD',
                     /'METIS'.
      #Paramètres pour le parallélisme
            ◊  PARALLELISME=/'CENTRALISE',                    [DEFAUT]
                            /'DISTRIBUE_MC',
                            /'DISTRIBUE_MD',
                            /'DISTRIBUE_SD'.
            ◊  PARTITION=sdfeti
            ◊  CHARGE_PROC0_MA=/100,                          [DEFAUT]
```

```
                              /chargema.                    [I]
        ◊  CHARGE_PROC0_SD=/0           ,                   [DEFAUT]
                              /chargesd.                    [I]
    #Paramètres pour la gestion mémoire de MUMPS
        ◊  OUT_OF_CORE=/'NON',                              [DEFAUT]
                       /'OUI'.

  #Solveur itératif «maison» du gradient conjugué préconditionné par un
Cholesky Incomplet ILU(k):
  /METHODE='GCPC',

    #Paramètres numériques
        ◊  PRE_COND='LDLT_INC'.                             [DEFAUT]
        ◊  NIVE_REMPLISSAGE=/0,      ,                      [DEFAUT]
                              /niv.
         ◊  RENUM=/'RCMK',                                  [DEFAUT]
                /'SANS'.
    #Paramètres fonctionnels
         ◊ NMAX_ITER=/0,                                    [DEFAUT]
                    /niter.                                 [I]
        ◊  RESI_RELA=/10-6,                                 [DEFAUT]
                    /resi.                                  [R]

  #Solveurs itératifs basés sur la librairie externe PETSc:
  /METHODE='PETSC',

    #Paramètres numériques
        ◊  ALGORITHME=/'CG',                                [DEFAUT]
                     /'BCGS',
                     /'BICG',
                     /'CR',
                     /'GMRES',
                     /'TFQMR'.
        ◊  PRE_COND=/'LDLT_INC',                            [DEFAUT]
                   /'JACOBI',
                   /'SOR'.
        ◊  NIVE_REMPLISSAGE=/0,                             [DEFAUT]
                           /niv.
        ◊  REMPLISSAGE=/1.0,                                [DEFAUT]
                        /rem.
         ◊  RENUM=/'RCMK',                                  [DEFAUT]
                /'SANS'.
    #Paramètres fonctionnels
        ◊  NMAX_ITER=/0,                                    [DEFAUT]
                    /niter.                                 [I]
        ◊  RESI_RELA=/10-6,                                 [DEFAUT]
                    /resi  .                                [R]

  #Solveur hybride multidomaines de type FETI:
  /METHODE='FETI',

    #Paramètres fonctionnels
        ♦  PARTITION=sdfeti
        ◊  NMAX_ITER=/0,                                    [DEFAUT]
                    /niter.                                 [I]
        ◊  REAC_RESI=/0,                                    [DEFAUT]
                    /nreac.                                 [I]
        ◊  RESI_RELA=/10-6,                                 [DEFAUT]
                    /resi.                                  [R]
    #Paramètres du problème d'interface
        ◊  PRE_COND=/'LUMPE',                               [DEFAUT]
                      /'SANS'.
```

```
                ◊  SCALING=/'MULT',                                  [DEFAUT]
                           /'SANS'.
                ◊  TYPE_REORTHO_DD=/'GSM',                           [DEFAUT]
                                   /'GS',
                                   /'IGSM',
                                   /'SANS'.
                ◊  NB_REORTHO_DD=/0,                                 [DEFAUT]
                                  /nb_reortho.                       [I]
          #Paramètres des problèmes locaux
                ◊  RENUM=/'METIS',                                   [DEFAUT]
                         /'MD',
                          /'MDA'.
                 ◊  STOP_SINGULIER=/'OUI',                           [DEFAUT]
                                  /'NON'.
                 ◊ NPREC=/8,                                         [DEFAUT]
                        /nprec.                                      [I]
          #Paramètres de tests, de profiling et de monitoring. Divers.
                ◊  VERIF_SDFETI=/'OUI',                              [DEFAUT]
                                /'NON'.
                ◊  TEST_CONTINU=/10-6,                               [DEFAUT]
                                /test_continu.                       [R]
                ◊  STOCKAGE_GI=/'CAL',                               [DEFAUT]
                               /'OUI',
                               /'NON'.
                ◊  INFO_FETI=/'FFFFFFFFFFFFFFF',                     [DEFAUT]
                             /info_feti.                             [K15]
          #Paramètre pour le parallélisme
                ◊  NB_SD_PROC0=/0,                                   [DEFAUT]
                               /nb_sdproc0.                          [I]
          #Paramètres pour les accélérations (problème de type multiples
    seconds membres)
                ◊  ACCELERATION_SM=/'OUI',                           [DEFAUT]
                                   /'NON'.
                ◊  NB_REORTHO_INST=/0,                               [DEFAUT]
                                   /nb_reortho_inst.                 [I]


        #Paramètre commun à tous les solveurs
        ◊  SYME=/'NON',                                              [DEFAUT]
                /'OUI'.

                        ),
```

3 Opérandes

3.1 Opérande METHODE

◊ METHODE='MULT_FRONT'(**défaut**)/'LDLT'/'MUMPS'/'GCPC'/'FETI'/'PETSC'
Ce mot clé permet de choisir la méthode de résolution des systèmes linéaires:

/'MULT_FRONT' Solveur direct de type multifrontal. Le stockage matriciel est 'MORSE' (ou 'CSC' pour 'Compressed Sparse Column') 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/ncpus). La matrice initiale est stockée dans un seul objet JEVEUX et sa factorisée est répartie sur plusieurs, donc elle peut être déchargée partiellement et automatiquement sur disque.

/'LDLT' Solveur direct avec factorisation de Crout par blocs (sans pivotage). Le stockage matriciel hors solveur est MORSE. En entrée du solveur, on fait la conversion au format interne de LDLT: 'ligne de ciel' ('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 multifrontale avec pivotage. Ce solveur est obtenu en appelant le produit externe MUMPS développé par CERFACS-ENSEEIHT-INRIA-Parallab (voir copyright plus bas). Le stockage matriciel hors solveur est MORSE. En entrée du solveur, on fait la conversion au format interne de MUMPS: *i*, *j*, $\mathbf{K}_{ij}$, centralisée ou distribuée.
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.
Cette méthode est parallélisée en mémoire distribuée (MPI) et peut être exécutée sur plusieurs processeurs (*via* l'interface Astk menu Options/ Optionsdelancement/mpi_nbcpu&mpi_nbnoeud). On peut aussi profiter du parallélisme en mémoire partagée des librairies d'algèbre linéaire de bas niveau (BLAS). Pour ce faire il faut modifier le fichier de configuration du calcul (config.txt).

/'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.

/'PETSC' Solveurs itératifs issus de la librairie externe PETSc (Laboratoire Argonne). Le stockage matriciel hors solveur est MORSE. En entrée du solveur, on fait la conversion au format interne de PETSc: 'CSR' pour 'Compressed Sparse Row'. PETSc alloue des blocs de lignes contigus pas processeur. Cette méthode est parallélisée en mémoire distribuée (MPI) et peut être exécutée sur plusieurs processeurs (*via* l'interface Astk menu Options/Optionsdelancement/mpi_nbcpu&mpi_nbnoeud).

/'FETI' Solveur par décomposition de domaines de type FETI: gradient conjugué préconditionné projeté (GCPPC) pour le problème d'interface et solveur direct multifrontal pour les inversions des matrices de rigidité locales.
Les problèmes locaux étant inversés par la multifrontale 'MULT_FRONT', leurs matrices associées, matrices de rigidité locales et factorisées, sont traitées comme telles (cf. ci-dessus). Cette méthode est parallélisée en mémoire distribuée (MPI). Pour ce faire, contacter l'équipe de développement.

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 modélisations (XFEM, incompressible…)	+++	Séq: +++ //: + (speed-up~2)	++ OOC[1]	Solveur de référence. Sur 4 proc.
MUMPS CENTRALISE	Sauf modal et STOP_SINGULIER='DECOUPE' Vigilance en mode POURSUITE[2].	+++	Séq: +++ //: ++ (sp~8)	+++ IC/OOC	Plutôt en linéaire Sur 16 proc.
MUMPS DISTRIBUE	Idem centralisé moins cmde éclatées et DYNA_LINE_TRAN	++	Séq: +++ //: +++ (sp~16)	+++ IC/OOC	Surtout en non linéaire Sur 32 proc. Meilleur équilibrage *via* DISTRIBUE_SD
LDLT	Idem MULT_FRONT	+++	Séq: +	++ OOC	Plutôt les petits cas
Itératif					
GCPC	Sauf modal En symétrique réel	-	Séq: +	+++ OOC	Ne pas trop augmenter le niveau de préconditionnement. Parfois très efficace (thermique…).
PETSC	Idem GCPC + systèmes linéaires non symétriques. Vigilance en mode POURSUITE.	-	Séq: + //: + (sp~4)	+++ IC/OOC	Algorithmes robustes: GMRES,BCGS.
Hybride					
FETI	MECA_STATIQUE STAT_NON_LINE en symétrique réel	---	Séq : ++ + //:+++ (sp~8)	+++ OOC	Gros cas plutôt linéaire. Sur 16 proc. Paramètrage pointu Solveur de recherche.

Tableau 3.1-1. Synopsis des solveurs linéaires disponibles dans Code_Aster.

1 OOC pour 'Out-Of-Core'. C'est-à-dire qu'on va libérer de la mémoire RAM en déchargeant sur disque une partie des objets. Cela permet de suppléer au swap système et de traiter des problèmes bien plus gros. Suivant l'algorithmique, ces accès disques supplémentaires peuvent être pénalisants. Le mode de gestion opposé est «l'In-Core» (IC). Tous les objets informatiques restent en RAM. Cela limite la taille des problèmes accessibles (au swap système près) mais privilégie la vitesse.

2 En mode POURSUITE on ne sauvegarde sur fichier que les objets FORTRAN77 *Code_Aster*, pas les occurrences de produits externes. Donc attention à l'usage des commandes éclatées dans ce cadre (NUME_DDL/FACTORISER/RESOUDRE).

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 ddls).
/'METIS' Autre méthode de numérotation basée sur une dissection emboîtée. 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 > 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.

Remarques:
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 (en stockage SKYLINE) 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-ENSEEIHT-INRIA-Parallab est un solveur direct de type multifrontal, 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 GCPC: on ne peut pas l'utiliser dans les opérateur 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 $\mathbf{K}\,\mathbf{u}=\mathbf{f}$ *via* les notions d'*erreur directe relative* ('relative forward error') et d'erreur *inverse* ('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'):

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

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 $\kappa(\mathbf{K})\cdot\eta(\mathbf{K},\mathbf{f})$ 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 le 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. L'algorithme est plus rapide et moins coûteux en mémoire.
/' 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 prétraitement 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):
/'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 (jusqu'à la 4.7.3) 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.Schulse, 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 choisit 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 EFFEUR_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 modifiées 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_MC/MD/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_MC/MD/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 propres à *Code_Aster*.
Les trois 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. complètement en interne de l'opérateur concerné, par paquets de mailles contiguës ('DISTRIBUE_MC') ou par distribution cyclique ('DISTRIBUE_MD'). Ou soit par sous-domaines ('DISTRIBUE_SD') *via* une décomposition du modèle construite 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_MC/MD': **le parallélisme débute, en amont de MUMPS**, dès la phase de calcul élémentaire de *Code_Aster* . Chaque processeur alloue la matrice entière (et les structures de données Aster connexes NUME_DDL, MATR_ELEM...), n'en calcule et 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, soit **par paquets de mailles contiguës** (..._MC), soit **par distribution cyclique** (..._MD). Par exemple avec un modèle comportant 8 mailles et pour un calcul sur 4 processeurs, on a les répartitions de charge suivantes:

Mode de distribution	*Maille 1*	*Maille 2*	*Maille 3*	*Maille 4*	*Maille 5*	*Maille 6*	*Maille 7*	*Maille 8*
***_MC	Proc. 0	Proc. 0	Proc. 1	Proc. 1	Proc. 2	Proc. 2	Proc. 3	Proc. 3
***_MD	Proc. 0	Proc. 1	Proc. 2	Proc. 3	Proc. 0	Proc. 1	Proc. 2	Proc. 3

Tableau 3.4-1. Distribution des mailles de calcul suivant l'option du mot-clé PARALLELISME.

/ ' DISTRIBUE_SD ' : parallélisme similaire aux options ' DISTRIBUE_MC/MD ' , 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 «retailler» les objets FORTRAN manipulés au plus juste. Comme pour le solveur FETI, on tarit le plus tôt possible le flot de données et d'instructions par processeur. Par contre, il ne s'agit plus de raisonner en terme de sous-domaines et de problème d'interface, mais de traiter sélectivement des blocs matriciels/vectoriels du problème global que MUMPS va rassembler. Cette stratégie 'DISTRIBUE_***' est donc une voie intermédiaire entre FETI et MUMPS centralisé ou la multifrontale Aster. D'où des gains en terme de robustesse, de généricité et d'efforts d'implantation dans le code.

•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, de zones particulières (non linéarités…). La distribution par sous-domaines est plus souple et peut s'avérer plus efficace en permettant d'adapter le flot de données à sa simulation.
•Une autre cause de déséquilibrage 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 déséquilibrage plus marqué. L'utilisateur peut le compenser en renseignant un des mot-clés CHARGE_PROC0_*. 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.

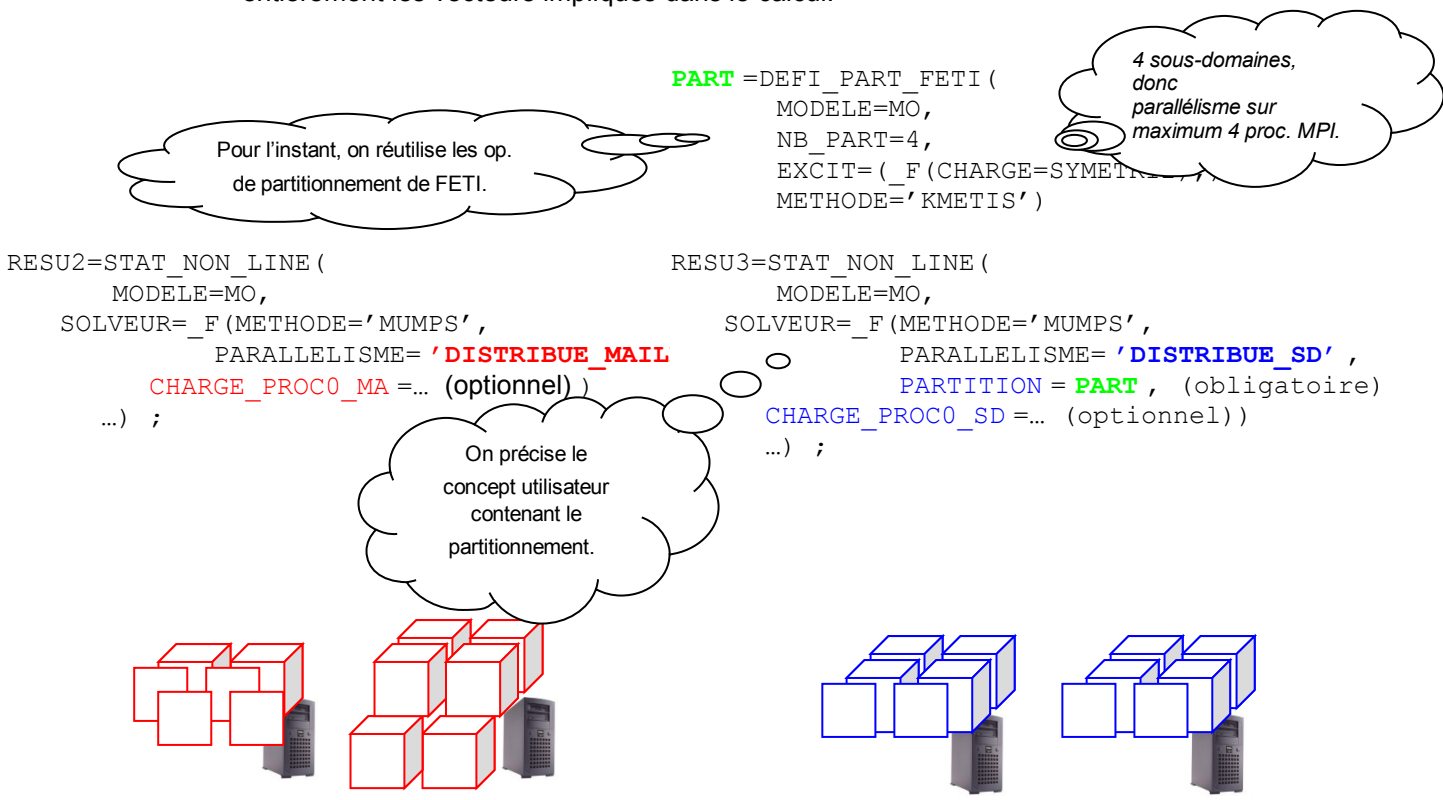

Figure 3.4-1. Mode de distribution de MUMPS sur 2 processeurs :
par paquets de mailles contigües ou par sous-domaines.
Extrait du cas-test mumps02a (8 HEXA+ 4 QUAD).

Remarque:
•De même, la matrice est pour l'instant dupliquée: dans l'espace JEVEUX et dans l'espace F90 de MUMPS. A terme, du fait du déchargement sur disque de la factorisée produite par

MUMPS (lorsque OUT_OF_CORE='OUI' valeur conseillée), elle va devenir un objet dimensionnant de la RAM.

◊ 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_SD=chargesd (**défaut** =0)
Nombre de sous-domaines (par rapport au nombre prévu normalement par l'heuristique) attribué au processeur maître. Dans l'exemple 3.4-1, si chargesd=1, le processeur maître ne sera plus responsable que de un 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'.

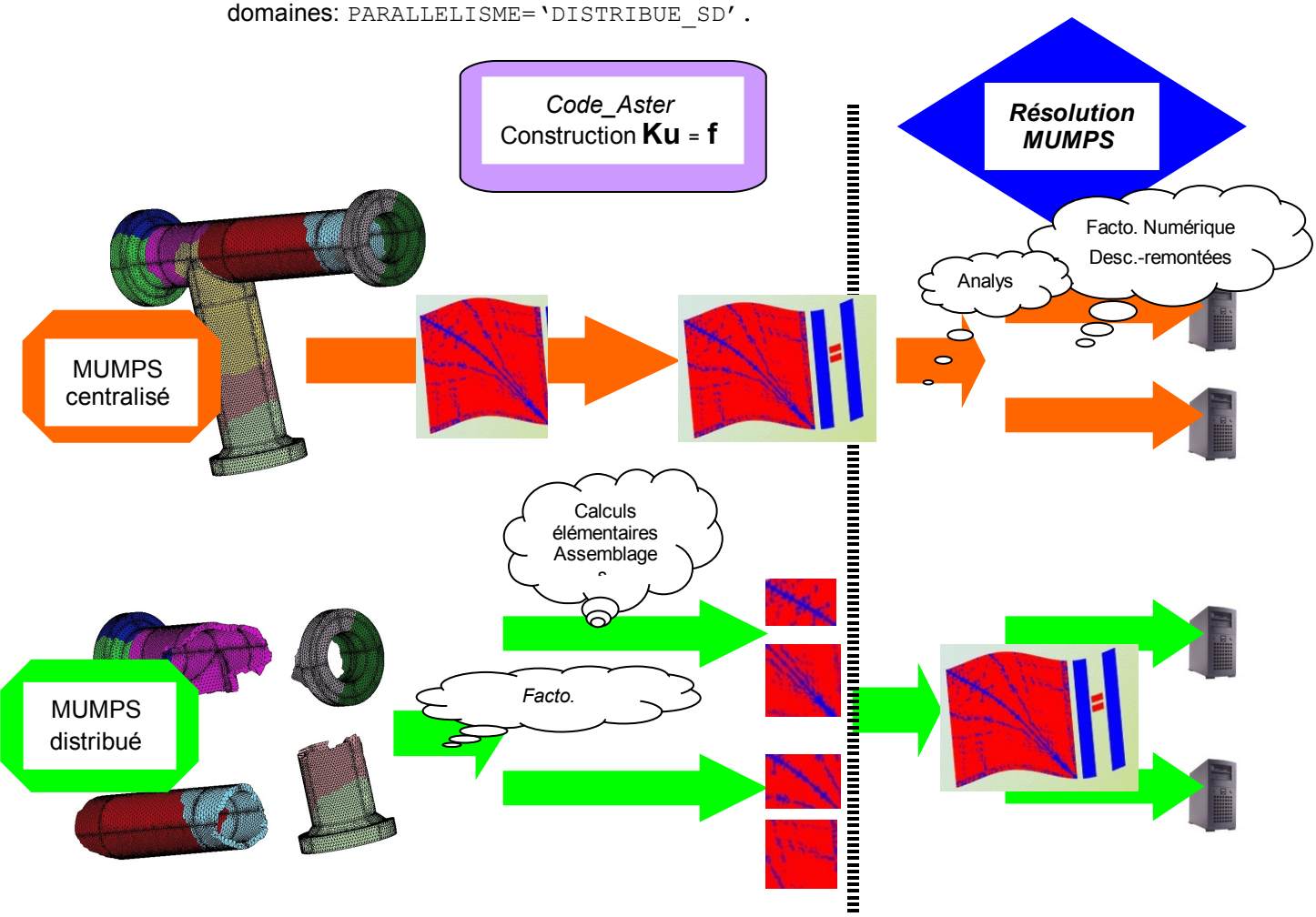

Figure 3.4-2. Flots de données/traitements parallèles de MUMPS centralisé/distribué.

◊ 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 3.4-1, 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 contiguës ou par répartition cyclique: PARALLELISME='DISTRIBUE_MC/MD'.

◊ OUT_OF_CORE ='OUI'/'NON' (**défaut**)

Pour activer ou désactiver les facultés OOC de MUMPS qui va alors décharger entièrement sur disque les blocs de factorisée gérés par chaque processeur. Cette fonctionnalité est bien sûr cumulable avec le parallélisme d'où une plus grande variété de fonctionnement pour s'adapter aux contingences d'exécution. L'OOC, tout comme le parallélisme, contribue à réduire la mémoire RAM requise par processeur. Mais bien sûr (un peu) au détriment du temps CPU: prix à payer pour les I/O pour l'un, pour les communications MPI pour l'autre.
Pour un petit cas linéaire, le mode «séquentiel IC» suffit; pour un plus gros cas toujours en linéaire, le mode «centralisé IC ou OOC» peut convenir; en non linéaire, avec réactualisation fréquente de la matrice tangente, le mode «distribué OOC» est vivement conseillé.

Remarques:

•Pour l'instant, seuls les vecteurs de réels contenant la factorisée sont (entièrement) déchargés sur disque. Les vecteurs d'entiers accompagnant cette structure de données (de taille tout aussi importante) ne bénéficient pas encore de ce mécanisme. D'autre part, ce déchargement ne s'opère qu'après la phase d'analyse de MUMPS. Bref, sur de très gros cas (plusieurs millions de ddls), même avec cet OOC, des contingences mémoires peuvent empêcher le calcul. On sort en principe avec une ERREUR_FATALE documentée.

•En dehors des aspects stockage précédents, les entiers de MUMPS peuvent aussi poser problème du fait de leur codage sur 4 octets (32 bits). Contrairement à ceux utilisés par défaut dans Code_Aster qui sont sur 8 octets. Les structures de données de MUMPS sont donc limitées à des tailles de l'ordre de 10^9 éléments. Compte-tenu de la largeur de bande et du facteur de remplissage moyens constatés, cela peut conduire à des arrêts de MUMPS faute d'espace d'adressage, à partir de quelques millions de ddls par processeur. Tout dépend du stockage requis par la phase d'analyse, pour l'instant séquentielle et In-Core. Pour réduire ce goulet d'étranglement, elle va être prochainement parallélisée et ses structures de données distribuées.

COPYRIGHT de MUMPS

3.5 METHODE:'GCPC'

◊ PRE_COND='LDLT_INC' (**défaut**)
Méthode de préconditionnement: la matrice de préconditionnement est obtenue par une décomposition **LDL**^{T} incomplète (par niveau) 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 a.taille(**K**):
- a =1 pour niv=0
- a =3,5 pour niv=1
- a =7,5 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...

◊ RENUM='RCMK' (**défaut**)/ 'SANS'
Renuméroteur de type 'Reverse Cuthill-Mackee' (comme 'LDLT') pour limiter le remplissage de la factorisée incomplète. D'autre part, ce renuméroteur a été adapté pour tenir compte des Lagranges, et permettre ainsi la factorisation de systèmes indéfinis sans technique de pivotage.

◊ 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 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='PETSC'

◊ ALGORITHME='BCGS'/'BICG'/'CG'(**défaut**)/'CR'/'GMRES'/'TFQMR'
Nom des solveurs itératifs (de type Krylov) de PETSc accessibles depuis *Code_Aster:*
- 'BCGS' pour gradient bi-conjugué stabilisé,
- 'BICG' pour gradient bi-conjugué,
- 'CG' pour gradient conjugué standard,
- 'CR' pour résidu conjugué,
- 'GMRES' pour 'Generalised Minimal RESidual',
- 'TFQMR' pour 'Transpose Free Quasi Minimal Residual'.

Les méthodes CG et CR sont à réserver aux modélisations d'*Aster* conduisant à des matrices symétriques (l'idéal serait SPD mais du fait des conditions limites de type Lagrange, on perd souvent cette propriété !). En non symétrique, il faut faire appel aux autres méthodes. GMRES et TFQMR, malgré leur surcoût, semblent les plus robustes.

◊ PRE_COND='LDLT_INC' (**défaut**) /'JACOBI'/'SOR'
Nom des préconditionneurs de PETSc accessibles depuis *Code_Aster:*
- 'LDLT_INC' pour factorisation incomplète par niveau,
- 'JACOBI' pour préconditionneur diagonal standard,
- 'SOR' pour Successive Over Relaxation.

Tous les préconditionneurs de PETSc proposent une kyrielle de paramètres. Pour l'instant, seul ceux liés aux remplissage du ILU sont accessibles à l'utilisateur Code_Aster. Pour les autres, on garde les valeurs par défaut. D'autre part, en mode parallèle, seul JACOBI offre un préconditionnement équivalent au mode séquentiel. Les deux autres, LDLT_INC et SOR, modifient un peu le calcul en utilisant des blocs diagonaux locaux aux processeur. C'est plus simple à mettre en oeuvre mais sous-optimal d'un point de vue algorithmique. Ces problèmes de 'parallel preconditioning' biaisent ainsi tout évaluation de speed-up.

Remarque:
•Avec les options par défaut ('CG'+'ILU'+'NIVE_REMPLISSAGE=0'+ 'RENUM='RCMK') on retrouve un algorithme très proche de celui du GCPC natif d'Aster (cf. §3.5).

◊ NIVE_REMPLISSAGE=niv (**défaut=0**)
Niveau de remplissage du préconditionneur si PRE_COND='LDLT_INC' (cf §3.5).

◊ REMPLISSAGE= a (**défaut=1.0**)
Facteur d'accroissement de la taille du préconditionneur en fonction du niveau de remplissage (cf §3.5). La référence est fixé à niv=0 pour lequel k=1. Ce paramètre n'est pris en compte que si PRE_COND='LDLT_INC'. Ce chiffre permet à PETSc de prévoir “à la louche” la taille nécessaire pour stocker le préconditionneur. Si cette estimation est trop erronnée ou non fournie, la librairie redimensionne d'elle-même les objets, mais cette opération est plus coûteuse.

◊ RENUM='RCMK' (**défaut**)/ 'SANS'
Renumeroteur de type 'Reverse Cuthill-Mackee' (comme 'LDLT') pour limiter le remplissage de la factorisée incomplète. D'autre part, l'usage de ce renumeroteur adapté pour tenir compte des Lagranges, permet de “bluffer” les préconditionneurs de PETSc et de les utiliser sur des systèmes indéfinis.

◊ NMAX_ITER=niter (**défaut=**-1)
Nombre d'itérations maximum de l'algorithme de résolution itératif. Si niter<=0 alors il est fixé automatiquement par PETSc (*via* PETSC_DEFAULT/maxits=10^5).

◊ 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 la norme $\|\ \|$ euclidienne

3.7 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].

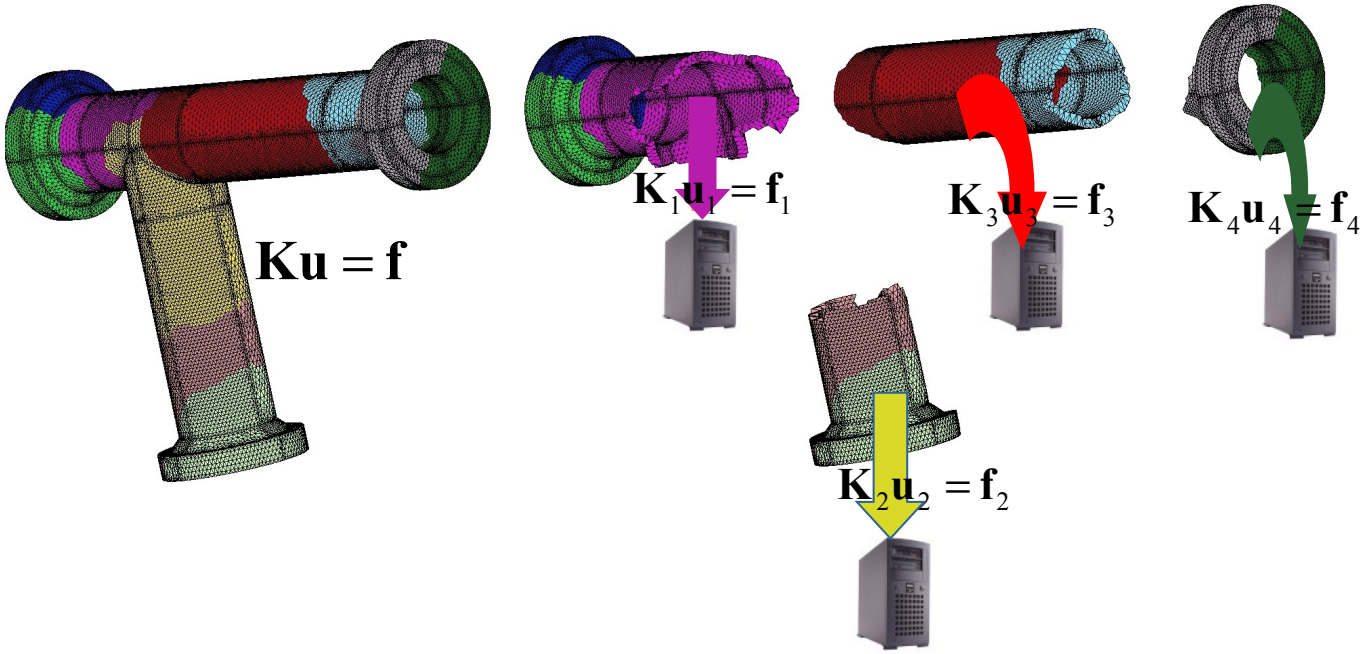

Figure 3.6-1. Principe de la décomposition de domaines de type FETI illustré sur une zone de mélange d'un circuit RRA.

◊ 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) où nbi 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

$$\frac{\|\mathbf{P}\mathbf{r}_m\|}{\|\mathbf{b}\|} \leq resi$$

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

$\mathbf{P}$ l'opérateur de projection

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

$$\frac{\|\mathbf{P}\,\mathrm{r}_m\|}{\|\mathbf{b}\|} \leq resi$$

$\mathbf{r}_m$ est le résidu à l'itération m
$\mathbf{P}$ l'opérateur de projection
$\mathbf{b}$ est le second membre et $\|\ \|$ 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)**
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 fournit 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 $\mathbf{G}_i$ 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 + 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)**='FFFFFFFFFFFFFFF')
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 nmax_freq=min(nbi-2, niter) valeurs propres *via* l'algorithme IRAM[R5.01.01] en projetant sur un espace de taille dim_sous_espace=min(nbi, 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.

Figure 3.5-1. Extrait du fichier de message (.mess) lorsqu'on active ce test.

•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).
•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 rempli 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 rempli 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:

nb_reortho_inst=max(nb_pas_temps/5,5)

où nb_pas_temps est le nombre de pas de temps du problème.
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 5ième pas de temps on ne peut utiliser que les 4 pas de temps antérieurs) on le fixe dynamiquement à cette valeur:

nb_reortho_inst=num_pas_temps – 1.

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

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

Attention:

•La symétrisation de la matrice $\mathbf{K}$ *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.*
•Lorsque le problème est purement linéaire, on ne peut attendre aucune compensation d'un quelconque processus englobant. Si le solveur ne gère pas les systèmes non symétriques (par ex. 'GCPC' Aster ou PETSc), l'appel à ce critère est illicite. A contrario, pour ce type de solveur à périmètre restreint, ce critère est obligatoire en non linéaire.
•De manière générale, l'opérateur détecte le caractère symétrique ou non de la matrice de travail et oriente automatiquement vers la bonne variante du solveur.

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_REMPLISSAGE=1`).

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