Notice d'utilisation des solveurs linéaires

1 But

Les **solveurs linéaires sont en fait omniprésents dans le déroulement des opérateurs de *Code_Aster*** car ils sont souvent enfouis au plus profond d'autres algorithmes numériques : schéma non linéaire, intégration en temps, analyse modale etc. Ils en consomment souvent la majeure partie du temps CPU et de la mémoire. Le choix et le paramétrage complet du solveur linéaire requis s'effectue *via* le mot-clé facteur SOLVEUR. Il est présent dans la plupart des commandes de calcul (STAT_NON_LINE, THER_LINEAIRE, CALC_MODES etc.).
Cette **notice d'utilisation est complémentaire de la documentation Utilisateur** du mot-clé SOLVEUR[U4.50.01]. Elle se veut un maillon intermédiaire entre la «simple» documentation de mot-clé et celle théorique (à chaque solveur est adossée une documentation de Référence).

La première partie de cette note donne une **vision d'ensemble** des différents solveurs linéaires disponibles, de leur périmètre d'utilisation et de leur performances moyennes en terme de robustesse et de consommations CPU/mémoire.
Le chapitre suivant regroupe quelques **conseils** pour utiliser au mieux les paramètres de SOLVEUR en fonction des cas de figure: la taille du système, ses propriétés numériques, les consommations CPU et mémoire... Pour finir, on détaille certains paramètres de SOLVEUR afin d'aider l'utilisateur dans un **usage avancé** de cette fonctionnalité du code.
Les **problématiques connexes** d'amélioration des performances (RAM/CPU) d'un calcul et de l'utilisation du parallélisme sont aussi brièvement abordées. Elles font l'objet de notices spécifiques, respectivement, [U1.03.03] et [U2.08.06].

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Table des Matières

1 But....1
2 Les différents solveurs linéaires disponibles....3
3 Quelques conseils d'utilisation....7
4 Préconisations sur les produits externes....9
5 Liens avec le parallélisme....11
5.1 Généralités....11
5.2 Calculs indépendants....11
5.3 Parallélisation des systèmes linéaires....12
5.4 Distribution de calculs modaux....13
6 Indicateurs de performance d'un calcul....15
7 Informations complémentaires sur les mot-clés....16
7.1 Détection de singularité et mot-clés NPREC/ STOP_SINGULIER/ RESI_RELA....16
7.2 Solveur MUMPS (METHODE='MUMPS')....20
7.2.1 Généralités....20
7.2.2 Périmètre d'utilisation....21
7.2.3 Paramètre RENUM....21
7.2.4 Paramètre ELIM_LAGR2....21
7.2.5 Paramètre RESI_RELA....22
7.2.6 Paramètres pour optimiser la gestion mémoire (MUMPS et/ou JEVEUX)....23
7.2.7 Paramètres pour réduire le temps calcul via différentes techniques d'accélération/compression....26

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

2 Les différents solveurs linéaires disponibles

Ces **solveurs linéaires sont en fait omniprésents dans le déroulement des opérateurs** de *Code_Aster* car ils sont souvent enfouis au plus profond d'autres algorithmes numériques: schéma non linéaire, intégration en temps, analyse modale etc. Ils en consomment souvent la majeure partie du temps CPU et de la mémoire. Le choix et le paramétrage complet du solveur linéaire requis s'effectue *via* le mot-clé facteur SOLVEUR. Il est présent dans la plupart des commandes de calcul (STAT_NON_LINE, THER_LINEAIRE, CALC_MODES...).

Ce mot-clé permet de choisir entre les deux classes de solveurs: les directs et les itératifs. Concernant les **directs**, on dispose du classique algorithme de «Gauss» (SOLVEUR/METHODE='LDLT'), d'une factorisation multifrontale ('MULT_FRONT') et d'une externe ('MUMPS'). Pour les **itératifs**, il est possible de faire appel à un gradient conjugué ('GCPC') ou à certains outils de la librairie publique PETSc ('PETSC').

Seuls MULT_FRONT, MUMPS, PETSC sont **parallélisés**. Le premier en OpenMP, les deux autres en MPI et en OpenMP (parallélisme hybride). Mais tous les solveurs sont compatibles avec un traitement parallèle (*via* MPI) des calculs élémentaires et des assemblages. Et ce, que ces traitements soient initiés juste avant l'utilisation du solveur linéaire proprement dit, ou, dans un autre opérateur (par exemple de pré/post-traitements).

Détaillons un peu le fonctionnement de chacun d'entre eux:

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Solveurs directs

/'MUMPS'

Solveur direct de type multifrontale avec pivotage. Ce solveur est obtenu en appelant le **produit externe MUMPS** développé par CERFACS/IRIT/INRIA/CNRS (cf. Copyright §4). Le stockage matriciel hors solveur est MORSE. En entrée du solveur, on fait la conversion au format interne de MUMPS: i , j , 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 degrés de liberté de type "Lagrange" (éléments incompressibles...).

Cette **méthode est parallélisée en mémoire distribuée** (MPI) et en **mémoire partagée** (OpenMP). Elle peut être exécutée sur plusieurs processeurs (*via* l'interface Astk menu `Options/Optionsdelancement/ncpus&mpi_nbcpu&mpi_nbnoeud`) .

En parallèle MPI (`mpi_nbcpu&mpi_nbnoeud`)**, MUMPS distribue naturellement ses données** (matrice, factorisée...) entre les cœurs des différents nœuds de calcul alloués. Ce qui accélère grandement les calculs et permet de réduire l'occupation mémoire nécessaire, par processus MPI, pour lancer le calcul. Cette consommation RAM peut encore plus être réduite *via* les paramètres `GESTION_MEMOIRE` ou `RENUM` de MUMPS.

En terme de mémoire, le goulet d'étranglement peut alors se trouver au niveau de l'espace `JEVEUX`. Pour réduire ce dernier, on peut distribuer la matrice *Aster* (`MATR_ASSE`) *via* l'option `MATR_DISTRIBUEE`.

MUMPS s'appuie aussi sur une **deuxième niveau de parallélisme imbriqué** au sein du parallélisme MPI (**parallélisme hybride**) et basé sur OpenMP. Ce deuxième niveau de parallélisme est principalement activé dans les appels aux librairies mathématiques sous-jacentes (BLAS, LAPACK). Contrairement au MPI, c'est un parallélisme à mémoire partagée et donc limité aux seuls cœurs d'un nœuds de calcul (`ncpus`).

Pour accélérer les grosses études (N au moins > 2.10^6 ddls) on peut en plus activer des options d'accélération/compression de MUMPS (à partir de la v5.1.0) *via* les mots-clés `ACCELERATION/LOW_RANK_SEUIL`.

/'MULT_FRONT'

Solveur direct de type multifrontale développé en interne EDF R&D. 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.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

/'LDLT' **Solveur direct avec factorisation de Crout** par blocs (sans pivotage) **développé en interne EDF R&D**. 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.
En outre, ce solveur permet de ne factoriser que partiellement la matrice. Cette possibilité est "historique". Elle permet de factoriser la matrice en plusieurs "fois" (plusieurs travaux) voire de modifier à la volée les dernières lignes de cette factorisée. Aujourd'hui, on n'imagine pas bien l'intérêt de cette fonctionnalité hormis pour certaines méthodes (dite discrètes) de contact-frottement où l'on a, à dessein, placé dans les dernières lignes de la matrice les termes concernant les noeuds susceptibles d'être en contact. Ainsi, au fur et à mesure des itérations d'appariement, les relations entre ces noeuds changeant, on efface puis recalcule que ces dernières contributions de la factorisée. C'est un exemple typique où l'usage astucieux d'un algorithme assez frustre peu amener des gains majeurs (en temps).

Solveurs itératifs

/'GCPC' **Solveur itératif de type gradient conjugué avec préconditionnement $ILU(k)$ ou basé sur une factorisée simple précision (*via* MUMPS).**
Le stockage de la matrice est alors MORSE. Avec le Cholesky incomplet, la matrice initiale et sa factorisée incomplète sont stockées, chacune, dans un seul objet JEVEUX.
Avec la factorisée simple précision, le préconditionneur est beaucoup plus coûteux (en CPU/RAM), mais il est effectué en simple précision et son calcul peut être mutualisé pendant plusieurs résolutions (problème de type multiples seconds membres, par ex. STAT_NON_LINE).

/'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/ Options de lancement /mpi_nbcpu&mpi_nboeud).
Lorsque PETSc utilise un préconditionneur basé sur MUMPS (PRECOND='LDLT_SP'), il bénéficie aussi du parallélisme potentiellement hybride du solveur direct. Mais les meilleures accélérations de PETSc se produisent plutôt en privilégiant le premier niveau de parallélisme, MPI.

Attention : les solveurs PETSC et MUMPS étant incompatibles en séquentiel, on privilégie généralement MUMPS. Pour utiliser PETSC, il faut donc souvent lancer une version parallèle de *Code_Aster* (quitte à ne solliciter qu'un seul processeur).

	Périmètre Solveur	Robustesse	CPU	Mémoire	Détails
Direct					
MULT_FRONT	Solveur universel. A déconseiller pour les	+++	Séq: ++ //: ++	+ OOC[1]	Sur 4/8 coeurs.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

	Périmètre Solveur	Robustesse	CPU	Mémoire	Détails
	modélisations nécessitant du pivotage (EF mixtes de X-FEM, incompressible...).		(speed-up~2)		
MUMPS	Solveur universel, solveur de référence	+++	Séq: ++ //: +++ (sp~30)	-- IC + OOC	Jusqu'à 512 coeurs
LDLT	Solveur universel (mais très lent sur de gros cas). A déconseiller pour les modélisations nécessitant du pivotage (EF mixtes de X-FEM, incompressible...) . Factorisation partielle possible (contact discret).	+++	Séq: +	+ OOC	Plutôt les petits cas ou les cas de taille moyenne pouvant tirer partie de factorisation partielle.
Itératif					
GCPC	Problèmes symétriques réels sauf ceux requérant obligatoirement une détection de singularité (calcul modal, flambement).	- (LDLT_INC) + (LDLT_SP)	Séq: ++ +	+++ (LDLT_INC) ++ (LDLT_SP)	Avec LDLT_INC ne pas trop augmenter le niveau de préconditionnement. Parfois très efficace (thermique…) surtout en non linéaire avec LDLT_SP.
PETSC	Idem GCPC mais compatible avec le non symétrique.	- (LDLT_INC) + (LDLT_SP)	Séq: ++ + //: +++ (sp~4)	++ IC	Algorithmes plutôt robustes: GMRES. Souvent très efficace en non linéaire avec LDLT_SP.

Figure 2.-1: Synoptique des solveurs linéaires disponibles dans Code_Aster.

Remarque:

• Pour être complètement exhaustif, on peut préciser que quelques (rares) opérations numériques sont opérées avec un paramétrage solveur «figé ». Celui-ci est donc inaccessible aux utilisateurs (sans surcharge logiciel). Parmi celles-ci, on trouve certaines résolutions des méthodes discrètes de contact-frottement (réservées à LDLT), les recherche de modes rigides en modal, les calculs de modes d'interfaces (réservées à MUMPS ou à LDLT)... A chaque fois des raisons fonctionnelles ou de performance expliquent ce choix peu transparent.

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 pénalisant. 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.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

3 Quelques conseils d'utilisation

Si on souhaite[2] changer de solveur linéaire ou adapter son paramétrage, plusieurs questions doivent être abordées: Quel est le type de problème que l'on souhaite résoudre ? Qu'elles sont les propriétés numériques des systèmes linéaires rencontrés ? etc.
On liste ci-dessous et, de manière non exhaustive, plusieurs questions qu'il est intéressant de se poser lorsqu'on cherche à optimiser les aspects solveurs linéaires. Bien sûr, certaines questions (et réponses) sont cumulatives et peuvent donc s'appliquer simultanément.

En bref:

La méthode par défaut reste la multifrontale interne MULT_FRONT. Mais pour pleinement bénéficier des gains CPU et RAM que procure le **parallélisme**, ou pour résoudre un **problème numériquement souvent difficile** (X-FEM, incompressibilité, THM), on préconise l'utilisation du produit externe **MUMPS**. Plus le problème est de grande taille, plus on conseille d'utiliser le parallélisme MPI et les technique d'accélération/compression (ACCELERATION/LOW_RANK_SEUIL).
Si, malgré tout, sur une plate-forme informatique donnée, le problème ne passe pas en mémoire, le recourt aux solveurs itératifs de PETSC peut être une solution (si leurs périmètres fonctionnels le permet). Notons que par par défaut ils utilisent en «sous-main» MUMPS en tant que préconditionneur (option PRE_COND='LDLT_SP').
Pour aller plus loin dans les **économies en place mémoire** , on peut aussi dégrader le préconditionnement (le calcul sera probablement plus long) en choisissant un des autres préconditionneurs de PETSC ('LDLT_INC'...).

En **non-linéaire, pour gagner en temps** , on peut aussi jouer sur plusieurs paramètres de relaxation (SYME en non symétrique) ou d'intéractions «solveur non linéaire/solveur linéaire» (REAC_PRECOND, NEWTON_KRYLOV..).
Pour des problèmes non linéaires **bien conditionnés** (thermique...), le recours à un MUMPS «relaxé» (MIXER_PRECISION/FILTRAGE_MATRICE) peut amener des gains mémoire très sensibles. De même, en linéaire comme en non linéaire, avec PETSC sans préconditionneur (PRE_COND='SANS').

Pour plus de détails et de conseils sur l'emploi des solveurs linéaires on pourra consulter la notice d'utilisation [U4.50.01] et les documentations de référence associées [R6...]. Les problématiques connexes d'amélioration des performances (RAM/CPU) d'un calcul et, de l'utilisation du parallélisme, font aussi l'objet de notices détaillées: [U1.03.03] et [U2.08.06].

Quel est le type de problème à résoudre ?

- **Résolution de nombreux systèmes avec la même matrice** (problème de type multiples seconds membres[3]) ⇒ solveurs MULT_FRONT ou MUMPS (si possible en désactivant RESI_RELA et avec GESTION_MEMOIRE='IN_CORE').
- **Calcul paramétrique standard en linéaire** ⇒ un premier essai avec MUMPS en laissant les paramètres par défaut, puis tous les autres runs en débranchant RESI_RELA ou avec POSTTRAITEMENTS='MINI'.
- **Calcul paramétrique en non linéaire avec une matrice bien conditionnée** ⇒ un premier essai avec MUMPS en jouant sur les paramètres de relaxation (FILTRAGE_MATRICE/MIXER_PRECISION ou SYME si on est en non symétrique). Puis, si un point de fonctionnement optimisé (consommations CPU/RAM) a été dégagé, l'utiliser pour tous les autres runs.
  On peut aussi essayer MUMPS mais, cette fois, comme préconditionneur simple précision de PETSC/GCPC (LDLT_SP). L'intérêt est alors de ne le réactualiser que périodiquement (REAC_PRECOND).

2 Ou si, tout simplement, il faut jouer sur ce paramètre car le calcul ne passe pas sur la plate-forme logicielle choisie, ou avec des consommations en temps et en mémoire incompatibles avec les contraintes de l'étude.
3 C'est le cas, par exemple, en chaînage thermo-mécanique lorsque les caractéristiques matériaux ne dépendent pas de la température ou, en non linéaire, lorsqu'on réactualise peu souvent la matrice tangente.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Quelles sont ses propriétés numériques ?

- **Système linéaire bien conditionné**[4]($<10^4$) ⇒ solveurs MUMPS+MIXER_PRECISION ou GCPC/PETSC.
- **Système linéaire difficile** (mauvais conditionnement, éléments finis mixtes, prédominances de Lagranges...) ⇒ solveur MUMPS.

A-t-on besoin d'une solution très précise ?

- On peut se contenter d'une **solution très approximée**[5] ⇒ solveurs GCPC ou PETSC avec un RESI_RELA=10^{-3} ou MUMPS avec LOW_RANK_SEUIL<10^{-9} et POSTTRAITEMENTS='SANS'.
- On veut une **solution précise** ou, au moins, un diagnostic sur sa qualité et sur les difficultés numériques du système à résoudre ⇒ solveur MUMPS en activant NPREC et RESI_RELA (en INFO=2).

Comment optimiser les consommations temps/mémoire RAM du solveur linéaire ?

- **Temps** ⇒ solveur MUMPS en désactivant l'OOC (GESTION_MEMORE='IN_CORE') voire RESI_RELA. Calcul parallèle hybride MPI/OpenMP (cf. doc. U2 sur la parallélisme). Sur les problèmes de grande taille usage des compressions low_rank et des accélérations *via* les mots-clés ACCELERATION/LOW_RANK_SEUIL.
- **Mémoire** ⇒ solveur MUMPS en activant GESTION_MEMOIRE='OUT_OF_CORE'/MATR_DISTRIBUE et en mode parallèle distribué. Ou solveur itératif de type Krylov : GCPC/PETSC+LDLT_SP.
  En non linéaire, si la matrice est bien conditionnée et/ou non symétrique, on peut aussi jouer sur les paramètres de relaxation de MUMPS (FILTRAGE_MATRICE, MIXER_PRECISION et SYME) ou utiliser un solveur itératif en réduisant le coût du préconditionnement. On trouvera plus de détails dans la discussion du §7.2.6.
- **Mémoire** ⇒ solveur PETSC en activant MATR_DISTRIBUE et en mode parallèle distribué.

Est-ce un problème frontière de très grande taille (> 5.10^6 degrés de liberté) ?

- **Solveur robuste** ⇒ solveur MUMPS avec les optimisations mémoires précédentes et les compressions low-rank (ACCELERATION/LOW_RANK_SEUIL).
- **Solveurs de la «dernière chance»** ⇒ solveurs itératifs (avec un niveau de préconditionnement faible).

Comment optimiser les performances globales de mon calcul ?

- Notice d'utilisation [U1.03.03].

Comment effectuer, calibrer et optimiser un calcul parallèle ?

- Notice d'utilisation [U2.08.06].

4 Soit la typologie du calcul renseigne sur cette information (on sait par exemple que c'est souvent le cas de la thermique), soit on l'obtient par ailleurs (par exemple, en faisant un test préliminaire avec MUMPS et en activant le mot-clé RESI_RELA+INFO=2).

5 Calculs non linéaires convexe, calcul prospectif ou calcul dont la qualité des solutions mécaniques d'intérêt est maîtrisée par ailleurs.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

4 Préconisations sur les produits externes

Pour résoudre les nombreux systèmes linéaires qu'il produit, *Code_Aster* peut s'appuyer sur des produits externes. Cette section documente l'utilisation de ces produits dans le cadre de *Code_Aster* en se référant à une utilisation et une installation classique.

Les versions préconisées sont :

- **Renuméroteurs/partitionneurs** METIS 5.1.0/PARMETIS 4.0.3 et (PT)SCOTCH 7.0.1 (compilés en entier 64 bits).
- **Solveur direct et préconditionneur :** MUMPS 5.4/5.5 (versions publiques) et 5.4/5.5consortium (versions en accès restreint).
  Les versions consortium en accès restreint sont installées par défaut à EDF. Leurs codes source ne peuvent être redistribués hors EDF. Elles procurent potentiellement un accès à des fonctionnalités exploratoires en avance de phase par rapport aux versions publiques (cf. mots-clés SOLVEUR=_F(ACCELERATION, LOW_RANK_SEUIL et REDUCTION_MPI).
- **Solveurs itératifs et préconditionneurs :** PETSc 3.17.1p0.

  D'autre part, toutes les versions pointent désormais sur les versions 64 bits des rénuméroteurs externes ((PAR)METIS , (PT)SCOTCH et PORD). Cela permet à code_aster de résoudre de très gros modèles éléments finis (> 10^7 degrés de liberté) *via* le choix de MUMPS, soit en tant que solveur direct (METHODE='MUMPS'), soit en tant que préconditionneur (METHODE='PETSC' ou 'GCPC' +PRE_COND='LDLT_SP/DP') .

Pour MUMPS 5.5.1 le Copyright est le suivant (licence CeCILL-C V1) :

```
 Copyright 1991-2022 CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria,
Mumps Technologies, University of Bordeaux.

This version of MUMPS is provided to you free of charge. It is
released under the CeCILL-C license
(see doc/CeCILL-C_V1-en.txt, doc/CeCILL-C_V1-fr.txt, and
https://cecill.info/licences/Licence_CeCILL-C_V1-en.html),
except for variants of AMD ordering and xMUMPS_TRUNCATED_RRQR
derived from the LAPACK package distributed under BSD 3-clause
license (see headers of ana_orderings.F and lr_core.F),
and except for the external and optional ordering PORD provided
in a separate directory PORD (see PORD/README for License information).

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

 [1] P. R. Amestoy, I. S. Duff, J. Koster and  J.-Y. L'Excellent,
 A fully asynchronous multifrontal solver using distributed dynamic
 scheduling, SIAM Journal on Matrix Analysis and Applications,
 Vol 23, No 1, pp 15-41 (2001).

 [2] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent and T. Mary,
 Performance and scalability of the block low-rank multifrontal
 factorization on multicore architectures,
 ACM Transactions on Mathematical Software,
 Vol 45, Issue 1, pp 2:1-2:26 (2019)

As a counterpart to the access to the source code and rights to copy,
modify and redistribute granted by the license, users are provided only
with a limited warranty  and the software's author,  the holder of the
economic rights,  and the successive licensors  have only  limited
liability.

In this respect, the user's attention is drawn to the risks associated
with loading,  using,  modifying and/or developing or reproducing the
software by the user in light of its specific status of free software,
that may mean  that it is complicated to manipulate,  and  that  also
therefore means  that it is reserved for developers  and  experienced
```


Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

```
  professionals having in-depth computer knowledge. Users are therefore
  encouraged to load and test the software's suitability as regards their
  requirements in conditions enabling the security of their systems
  and/or data to be ensured and, more generally, to use and operate it
  in the same conditions as regards security.

  The fact that you are presently reading this means that you have had
  knowledge of the CeCILL-C license and that you accept its terms.
```

Pour MUMPS 5.5.1c, le Copyright est le suivant :

```
    Copyright 1991-2022 CERFACS, CNRS, ENS Lyon, INP Toulouse, Inria,
  Mumps Technologies, University of Bordeaux.

  This version of the MUMPS software is not public and should be
  considered confidential. It is reserved to the members of the MUMPS
  consortium and to the gold subscribers to Mumps Technologies services.
  You should suppress it and any copy you might have obtained
  if you are not or no longer in the perimeter of a consortium membership
  agreement or a gold subscription agreement.

  As defined in the agreements, you are granted access to this version
  of the MUMPS software with limited warranty and with a free
  non-exclusive license limited to the duration of the agreement
  and with limited redistribution conditions: should you wish to
  redistribute this non-public version, it shall only be allowed
  to do so in Object Code form.

  PORD is an external optional ordering package from
  Juergen Schulze (University of Paderborn) provided
  in a separate directory PORD (see PORD/README for License information).
  Variants of AMD ordering and xMUMPS_TRUNCATED_RRQR
  derived from the LAPACK package are distributed under BSD 3-clause license
  (see headers of ana_orderings.F and lr_core.F).
```

Pour PETSc 3.17.1, le Copyright est le suivant (licence BSD clause n°2)

```
Copyright (c) 1991-20 22 , UChicago Argonne, LLC and the PETSc Development
Team
All rights reserved.

Redistribution and use in source and binary forms, with or without
modification,
are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this
  list of conditions and the following disclaimer.
* Redistributions in binary form must reproduce the above copyright notice,
this
  list of conditions and the following disclaimer in the documentation and/or
  other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
AND
ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE LIABLE
FOR
ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
DAMAGES
(INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

METIS est distribué sous licence Apache v2.0 disponible sous http://www.apache.org.

SCOTCH est distribué sous licence CeCILL-C V1 disponible sous http://www.cecill.info .

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

5 Liens avec le parallélisme

Nota : pour plus d'informations on pourra consulter la notice d'utilisation dédiée au parallélisme [U2.08.06].

5.1 Généralités

Souvent une simulation *Code_Aster* peut **bénéficier de gains importants de performance** en distribuant ses calculs sur plusieurs cœurs d'un PC ou sur un ou plusieurs nœuds d'une machine centralisée.

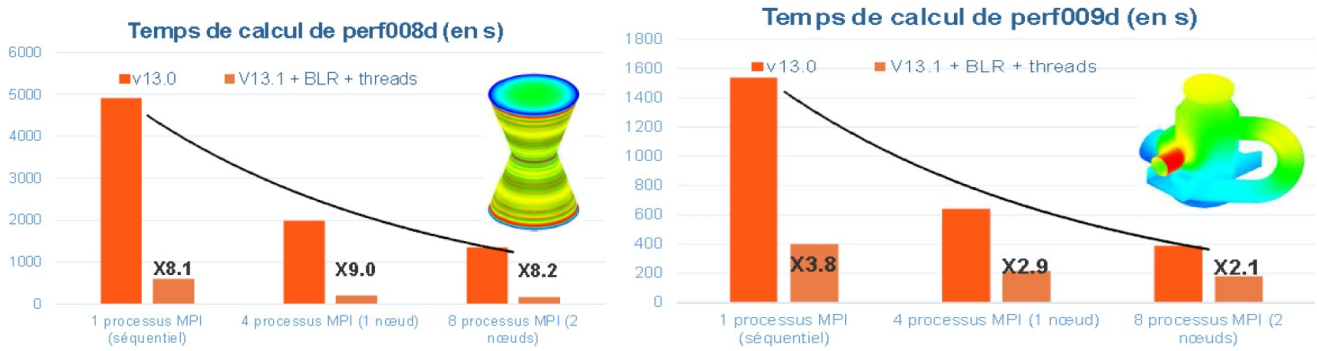


Figure 5.1.1._ Exemple de gains en temps procurés par le parallélisme MPI de Code_Aster v13.0, et avec celui hybride, MPI+OpenMP (+ compressions low-rank cf. [U4.50.01]) de Code_Aster v13.1. Comparaisons effectuées sur les cas-tests de performance perf008d et perf009d et sur la machine centralisée Aster5.

On peut **gagner en temps** (avec le parallélisme MPI et avec le parallélisme OpenMP) comme en **mémoire** (seulement *via* MPI). Ces gains sont variables suivant les fonctionnalités sollicitées, leurs paramétrages, le jeu de données et la plate-forme logicielle utilisée : cf. figure 5.1.1.
Dans *Code_Aster,* par défaut, le calcul est séquentiel. Mais on peut activer **différentes stratégies de parallélisation**. Celles-ci dépendent de l'étape de calcul considérée et du paramétrage choisi. Elles sont souvent cumulatives ou chaînables.
On a trois grandes classes de problèmes parallélisables, la seconde étant la plus courante:

- soit la simulation peut s'organiser en plusieurs **sous-calculs indépendants** (cf. § 5.2),
- soit ce n'est pas la cas mais :
  - celle-ci reste **dominée par des calculs linéaires ou non linéaires** (opérateurs `STAT/DYNA/THER_NON_LINE`, `MECA_STATIQUE`... cf. §5.3),
  - celle-ci reste **dominée par des calculs modaux** divisibles en sous-bandes fréquentielles (`INFO_MODE/CALC_MODES+'BANDE'`, cf. §5.4).

Pour avoir une **estimation du temps passé par un opérateur** et donc des étapes prédominantes d'un calcul, on peut activer le mot-clé `MESURE_TEMPS` des commandes `DEBUT/POURSUITE` (cf. [U1.03.03]) sur une étude type (éventuellement raccourcie ou édulcorée).
Dans tous les cas, on conseillera de **diviser les plus gros calculs en différentes étapes** afin de séparer celles purement **calculatoires**[6], de celles concernant des **affichages**, des **post-traitements** et des **manipulations de champs**[7].

5.2 Calculs indépendants

6 Eventuellement de différents types (catégorie n°2 ou n°3 citée précédemment) et qui gagneront à être effectuées en parallèle.
7 Qui seront souvent plus rapides en séquentiel du fait des risques d'engorgements lors des accès mémoire.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Lorsque la simulation peut s'organiser en différents sous-calculs *Aster* indépendants (cf. figure 5.2.1)**,** l'outil Astk[U1.04.00] propose une fonctionnalité adaptée (cf. [U2.08.07]). Celle-ci distribue ces sous-calculs sur différentes ressources machine et récupère leurs résultats. C'est un schéma parallèle complètement « informatique ».
La limite étant, pour l'instant, que tous ces calculs unitaires doivent pouvoir s'exécuter chacun séquentiellement sur la machine choisie (ce qui peut parfois poser des problèmes de ressources mémoire, cf. [U4.50.01]).

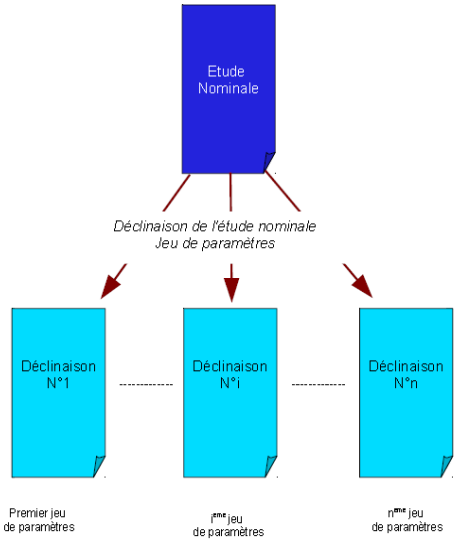


Figure 5.2.1._: Parallélisme des calculs indépendants.

5.3 Parallélisation des systèmes linéaires

Lorsque cette simulation ne peut pas se décomposer en sous-calculs *Aster* similaires et indépendants, mais qu'elle reste **dominée** néanmoins **par des calculs linéaires** ou **non linéaires** (opérateurs STAT/DYNA/THER_NON_LINE , MECA_STATIQUE ... cf. § 5.3), on peut organiser un schéma parallèle spécifique.
Il est fondé sur la **distribution des tâches** et des **structures de données** impliquées dans les manipulations de **systèmes linéaires** . Car ce sont ces étapes de construction et de résolution de systèmes linéaires qui sont souvent les plus sollicitantes en temps de calcul et en ressources mémoire. Elles sont présentes dans la plupart des opérateurs car elles sont enfouies au plus profond d'autres algorithmes «plus métiers»: solveur non linéaire, calcul modal et vibratoire, schéma en temps...

La première étape du schéma parallèle concerne la distribution des éléments finis du modèle sur tous les processus MPI. Chaque processus MPI ne va donc gérer que les traitements et les données associés aux éléments dont il a la charge. La construction des systèmes linéaires dans *Code_Aster* (calculs élémentaires, assemblages) s'en trouve alors accélérée. On parle souvent de « **parallélisme en espace** ». C'est un schéma parallèle plutôt d'ordre « informatique ».

Une fois ces portions de système linéaire construites (cf. figure 5.3.1), deux cas de figures se présentent:

- soit le **traitement suivant est naturellement séquentiel** et donc tous les processus MPI doivent avoir accès à l'information globale. Pour ce faire on rassemble ces bouts de systèmes linéaires et donc l'étape suivante ne sera ni accélérée, ni ne verra baisser ses consommations mémoire. Il s'agit le plus souvent d'une fin d'opérateur, d'un post-traitement ou d'un solveur linéaire non parallélisé en MPI (MULT_FRONT, LDLT, GCPC).
- soit le **traitement suivant accepte le parallélisme MPI**, il s'agit alors principalement des solveurs linéaires HPC MUMPS et PETSC. Le flot parallèle de données construit en amont leur est alors transmis (après quelques adaptations). Ces packages d'algèbre linéaire réorganisent ensuite, en interne, leurs propres schémas paralléles (avec une vision plus algébrique). On parle alors de schéma parallèle d'ordre plutôt « numérique ». Cette combinaison « parallélisme informatique », au niveau de

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

l'assemblage du système linéaire, et, « parallélisme numérique », au niveau de sa résolution, les 2 *via* MPI, est la combinaison la plus courante.

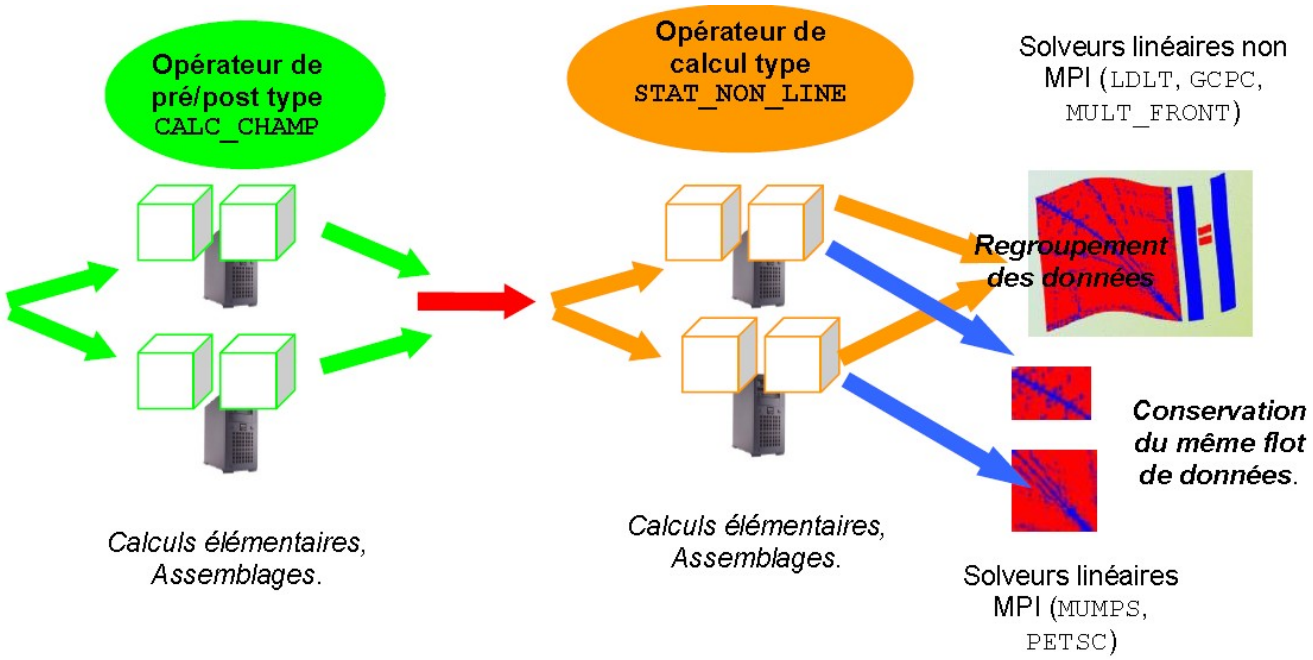


Figure 5.3.1._ Organisation du schéma parallèle MPI de construction et de résolution des systèmes linéaires.

Remarques:

- *Notons qu'à l'issue du cycle «construction de système linéaire – résolution de celui-ci », quelque soit le scénario mis en oeuvre (solveur linéaire séquentiel ou parallèle MPI), le vecteur solution est ensuite transmis, en entier, à tous les processus MPI. Le cycle peut ainsi continuer quelque soit la configuration suivante.*

De plus, on peut **superposer ou substituer à ce parallélisme MPI** (qui fonctionne sur toutes les plate-formes), un autre niveau de parallélisme géré cette fois par le **langage OpenMP**. Celui-ci est cependant limité aux fractions de machine partageant physiquement la même mémoire (PC multi-coeurs ou nœuds de serveur de calcul).
Il ne permet pas de baisser les consommations mémoire mais par contre il accélère certains types de calcul et ce, avec une granularité plus faible que celle du MPI : il procure une meilleure accélération même si le flot de données/traitements n'est pas très important. C'est un schéma parallèle d'ordre « informatique » qui intervient principalement dans les opérations basiques d'algorithmes d'algèbre linéaire (*via* par exemple la librairie BLAS et certaines étapes du solveur MUMPS).

Ce parallélisme peut être :

- soit cumulé avec le parallélisme MPI de MUMPS en accélérant les calculs au sein de chaque processus MPI. On obtient alors un schéma parallèle hybride à 2 niveaux.
- soit se substituer au parallélisme MPI en accélérant la résolution de système linéaire avec `MULT_FRONT`.

5.4 Distribution de calculs modaux

Lorsque la simulation ne peut pas se décomposer en calculs *Aster* indépendants, mais qu'elle reste **dominée** néanmoins **par des calculs modaux généralisés** (opérateurs `INFO_MODE` et `CALC_MODES`), on peut organiser un schéma parallèle spécifique (cf. 5.4.1).

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

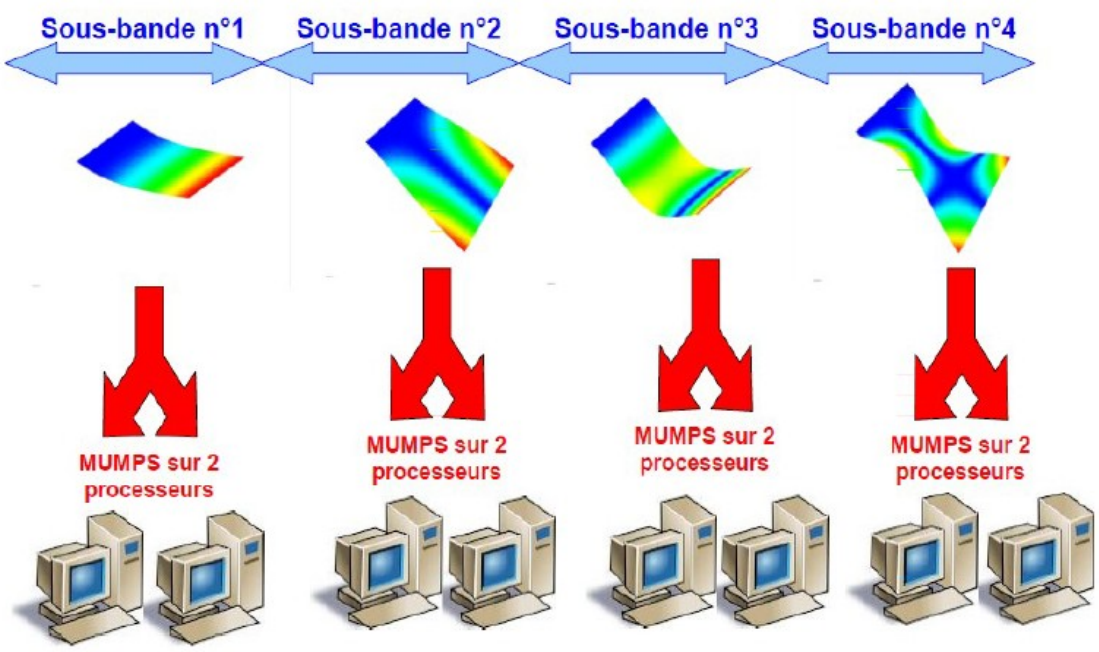


Figure 5.4.1._ Organisation du schéma parallèle MPI de distribution des calculs modaux et de résolutions des systèmes linéaires associés.

Il est fondé sur la **distribution de calculs modaux** indépendants : chacun étant en charge d'une sous-bande fréquentielle .
Ce schéma parallèle d'ordre purement « informatique » ne procure que des gains en temps (sauf si on le cumul avec MUMPS). Il peut toutefois se mixer avec les schémas parallèles précédents :

- **chaînage** entre différents opérateurs : parallélisme MPI de construction de matrices linéaires (dans par exemple CALC_MATR_ELEM) et résolution modale dans CALC_MODES.
- **cumul** , au sein de CALC_MODES , en activant le parallélisme MPI (voire OpenMP) du solveur direct MUMPS . On obtient alors un schéma parallèle hybride à 2 ou 3 niveaux.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

6 Indicateurs de performance d'un calcul

Pour plus d'informations on pourra consulter la notice d'utilisation [U1.03.03]: 'Indicateur de performance d'un calcul (temps/mémoire)'.

Lors d'une simulation *Code_Aster*, des affichages par défaut tracent dans le fichier message (.mess) certaines caractéristiques dimensionnantes du calcul. On retrouve notamment, pour chaque opérateur *Aster* :

- Les caractéristiques du calcul (nombre de nœuds, d'équations, de Lagrange, taille de la matrice...),
- Les mémoires JEVEUX plancher (pour passer en Out-Of-Core) et optimale (pour passer en In-Core),
- La mémoire requise par certains produits externes (par ex. MUMPS),
- Les temps CPU, système et «utilisateur» (elapsed),
- La ventilation des temps consommés suivant les étapes du calcul (calcul élémentaire, assemblage, résolution du système linéaire, déchargement sur disque).

Cette dernière description des temps consommés peut se décliner suivant différents niveaux de lecture (impression synthétique, détaillée et détaillée par incrément de calcul) *via* le paramètre MESURE_TEMPS/NIVE_DETAIL des commandes DEBUT/POURSUITE. En mode parallèle, on rajoute la valeur moyenne, sur tous les processeurs, des temps consommés ainsi que leur écart-type.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

7 Informations complémentaires sur les mot-clés

La signification des différents paramètres du mot-clé SOLVEUR font l'objet de la documentation Utilisateur [U4.50.01]. Celle-ci doit être synthétique pour aider l'utilisateur dans un usage standard du code. Pour une utilisation plus avancée quelques informations complémentaires peuvent s'avérer fort utiles. Ce chapitre récapitule ces éléments.

7.1 Détection de singularité et mot-clés NPREC/ STOP_SINGULIER/ RESI_RELA

L'étape essentielle des solveurs directs (SOLVEUR=_F(METHODE='LDLT'/'MULT_FRONT'/'MUMPS')) en terme de consommation. Or elle peut achopper dans deux cas de figures: problème de construction de la factorisée (matrice structurellement ou numériquement singulière) et détection numérique d'une singularité (processus approximé plus sensible). Le comportement du code va dépendre du cas de figure, du paramétrage de NPREC/STOP_SINGULIER/RESI_RELA et du solveur utilisé. La combinatoire des cas de figures est décrite dans le tableau ci-joint.

Type de solveur/Type de problème	**Construction de la factorisée** *En cas de problème que se passe-t-il ?*	**Détection numérique de singularité(s).** *En cas de singularité que se passe-t-il ?*
LDLT/MULT_FRONT	Arrêt en ERREUR_FATALE	**Cas n°1:** On factorise une matrice dynamique dans un opérateur de dynamique (cf. R5.01.01 § 3.8, CALC_MODES ...): Arrêt en ERREUR_FATALE si il s'agit de la matrice de travail de l'algorithme. Émission d'une ALARME si il s'agit d'une étape du test de Sturm. Dans ces deux cas STOP_SINGULIER n'a aucune incidence sur le processus et NPREC doit être positif. **Cas n°2:** Si STOP_SINGULIER='OUI', Arrêt en ERREUR_FATALE dans la phase de post-traitement du solveur linéaire. **Cas n°3:** Si STOP_SINGULIER='NON', Émission d'une ALARME dans la phase de post-traitement du solveur linéaire. Solution potentiellement imprécise pas détectée par le solveur linéaire. Hormis un éventuel processus englobant (Newton...) on a aucun garde-fou numérique pour garantir la qualité de la solution. **Cas n°4:** Si STOP_SINGULIER='DECOUPE', □ Lancement du processus de découpage du pas de temps. On reconstruit un nouveau problème pour un autre incrément de temps/chargement.
MUMPS	Arrêt en ERREUR_FATALE	**Cas n°1/4:** Si NPREC> 0,même comportement que LDLT / MULT_FRONT (on retrouve les cas

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Type de solveur/Type de problème	Construction de la factorisée *En cas de problème que se passe-t-il ?*	Détection numérique de singularité(s). *En cas de singularité que se passe-t-il ?*
		n°1 à 4) . **Cas n°5:** Si NPREC<0 et RESI_RELA>0, Détection de singularité désactivée mais on mesure la qualité de la solution dans le solveur linéaire. Si le système est singulier, son conditionnement sera très élevé et la qualité de résolution sera très mauvaise. Si cette qualité de résolution est supérieure à la valeur paramétrée dans RESI_RELA: arrêt en ERREUR_FATALE. **Cas n°6:** Si NPREC<0 et RESI_RELA<0, Détection de singularité et de mesure de la qualité de la solution toutes les deux désactivées. Hormis un éventuel processus englobant (Newton...) on a aucun garde-fou numérique pour garantir la qualité de la solution.

Tableau 7.1-1. Comportement du code, en fonction du paramétrage, lorsque la factorisation numérique détecte des problèmes (mauvaise mise en données, instabilités numériques, fort conditionnement...).

On compare quelques détails numériques des deux types de critères de détection (LDLT/MULT_FRONT *versus* MUMPS) de singularité dans le tableau ci-dessous:

Caractéristiques/Type de solveur	LDLT/MULT_FRONT	MUMPS
Critère	Local à chaque degré de liberté (valeur relative)	Global pour tous les degrés de liberté
Terme testé	Valeur absolue du terme diagonal de chaque ligne	Norme infinie de la ligne/colonne du la ligne correspondant au pivot
Détection du numéro de ligne (ISINGU dans les messages)	Toujours	Oui sauf lors de problèmes avec la construction de la factorisée
Fourniture du nombre de décimales perdues	Oui, sauf lors de problèmes avec la construction de la factorisée	Non
Désactivable	Non	Oui

Tableau 7.1-2. Différences dans les processus de détection de singularité suivant les solveurs.

Cependant, au delà des différences de mises en œuvre et des messages d'erreurs (tel solveur pointe un degré de liberté, tel autre solveur un autre degré de liberté), les deux classes de solveurs directs, LDLT/MULT_FRONT et MUMPS, concluent généralement au même type de diagnostics en cas de problèmes[8].
Elles pointent une mise en données déficiente: blocages redondants ou, au contraire, absents; relations linéaires surabondantes dues au contact-frottement; données numériques très hétérogènes (terme de pénalisation trop grands) ou illicites (module de Young négatif...).

Remarques:

8 Cf. Cas-tests erreu03a et erreu04a.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

- *Au pire, il faut ajuster la valeur de NPREC (augmenter ou diminuer de 1) pour conduire au même constat. En général, les singularités sont si flagrantes, que le paramétrage par défaut convient tout à fait.*
- *Contrairement aux deux autres solveurs, MUMPS ne précise pas le nombre de décimales perdues, par contre l'activation de son critère de qualité (par exemple RESI_RELA=1.10^{-6}) constitue un ultime garde-fou efficace contre ce type d'anicroche.*
- *Il est normal et assumé que le degré de liberté détecté soit parfois différent lorsqu'on change un paramètre numérique (solveur, renuméroteur, prétraitements...) ou informatique (parallélisme...). Tout dépend de l'ordre dans lequel chaque processeur traite les inconnues dont il a la charge et des techniques de pivotage/équilibrage éventuellement mises en œuvre. En général, même différents, les résultats concourent au même diagnostic: revoir les blocages de sa mise en données.*
- *Pour obtenir le conditionnement matriciel[9] de son opérateur «brut» (c'est-à-dire sans les éventuels prétraitements opérés par le solveur linéaire), on peut utiliser la combinaison MUMPS+PRETRAITEMENTS='NON'+INFO=2+NPREC<0+RESI_RELA>0 . Une valeur très importante (> 10 ^{12}) trahit alors la présence d'au moins une singularité. Par contre, si le surcoût calcul de la détection de singularité est indolore, celui de l'estimation du conditionnement et de la qualité de la solution l'est moins (jusqu'à 40% en temps elapsed; cf. § 7.2.5).*

Pour être plus précis, on a 9 cas de figures distincts listés dans le tableau ci-dessous. Ils sont basés sur la nullité exacte ou approchée[10] des termes «pivots» (cf. [R6.02.03] §2.3) sélectionnés par la phase de factorisation numérique du solveur direct considéré. Le premier cas de figure apparaît lors de la factorisation numérique proprement dite (matrice numériquement ou structurellement singulière). Le second cas provient de la phase de post-traitement activée à l'issue de cette factorisation numérique.

Prioritairement l'utilisateur doit suivre les conseils prodigués par le message d'alarme (listé dans le tableau ci-dessous). Si cela ne suffit vraiment pas, l'utilisateur avancé peut essayer de jouer sur les paramètres numériques du solveur (renuméroteur...), voire sur le critère NPREC lorsqu'il s'agit d'un pivot presque nul.

Type de problème	Informations disponibles	Conseils
Matrice ne s'appuyant pas sur un maillage. Pivot nul	Numéro de ligne (si MF/LDLT).	Mise en données (conditions limites, caractéristiques matériaux...). Essayer MUMPS (si MF/LDLT).
Matrice ne s'appuyant pas sur un maillage. Pivot presque nul.	Numéro de ligne, Nombre de décimales perdues (si MF/LDLT).	Idem.
Le pivot est un degré de liberté physique hors X-FEM. Pivot nul.	Numéro de ligne/noeud/composante (si MF/LDLT).	Mise en données (conditions limites, caractéristiques matériaux...). Essayer MUMPS (si MF/LDLT).
Le pivot est un degré de liberté physique hors X-FEM. Pivot presque nul.	Numéro de ligne/noeud/composante, Nombre de décimales perdues (si MF/LDLT)	Mode de corps rigide mal bloqué (défaut de blocage). Si le calcul comporte du contact il ne faut pas que la structure ne « tienne » que par les relations de contact.
Le pivot est un degré de liberté physique X-FEM. Pivot presque nul.	Numéro de ligne/noeud/composante (si MF/LDLT).	La level-set (fissure) passe très prés du noeud considéré (augmentez NPREC jusqu'à 10).

9 Au sens résolution de système linéaire creux, cf. papiers de M.Arioli/J.Demmel/I.Duff.
10 Contrôlé par NPREC (cf. [U4.50.01] §3.2).

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

Type de problème	Informations disponibles	Conseils
Le pivot est un Lagrange lié à une relation linéaire entre degrés de liberté. Pivot nul.	Numéro de ligne (si MF/LDLT).	Relations linéaires entre degrés de liberté surabondantes (LIAISON, contact...). Mise en données (conditions limites, caractéristiques matériaux...). Essayer MUMPS (si MF/LDLT).
Le pivot est un Lagrange lié à une relation linéaire entre degrés de liberté. Pivot presque nul	Numéro de ligne, Nombre de décimales perdues (si MF/LDLT).	Relations linéaires entre degrés de liberté surabondantes (LIAISON, contact...).
Le pivot est un degré de liberté de Lagrange lié à un blocage d'un degré de liberté. Pivot nul	Numéro de ligne (si MF/LDLT). Blocage concerné.	Blocage surabondant. Mise en données (conditions limites, caractéristiques matériaux...). Essayer MUMPS (si MF/LDLT).
Le pivot est un degré de liberté de Lagrange lié à un blocage d'un degré de liberté. Pivot presque nul	Numéro de ligne. Blocage concerné. Nombre de décimales perdues (si MF/LDLT).	Blocage surabondant.

Tableau 7.1-3. Différents cas de détection de singularité et conseils associés.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

7.2 Solveur MUMPS (METHODE='MUMPS')

7.2.1 Généralités

Le **solveur MUMPS** actuellement développé par **CNRS/INPT-IRIT/INRIA/CERFACS** 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.
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^{20} 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 et §3.4).
- 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...).
- 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.

Remarque:

- *Même dans le cadre très précis de la résolution de système linéaire, il existe de nombreuses façons de définir la sensibilité aux erreurs d'arrondis du problème considéré (c'est-à-dire son conditionnement). Celle retenue par MUMPS et, qui fait référence dans le domaine (cf. Arioli, Demmel et Duff 1989), est indissociable de la 'backward error' du problème. La définition de l'un n'a pas de sens sans celle de l'autre. Il ne faut donc pas confondre ce type de conditionnement avec la notion de conditionnement matriciel classique.*
  D'autre part, le conditionnement fourni pas MUMPS prend en compte le SECOND MEMBRE du système ainsi que le CARACTERE CREUX de la matrice. En effet, ce n'est pas la peine de tenir compte d'éventuelles erreurs d'arrondis sur des termes matriciels nuls et donc non fournis au solveur ! Les degrés de liberté correspondant ne « se parlent pas » (vu de la lorgnette élément fini). Ainsi, ce conditionnement MUMPS respecte la physique du problème discrétisé. Il ne replonge pas le problème dans l'espace trop riche des matrices pleines.
  ***Ainsi, le chiffre de conditionnement affiché par MUMPS est beaucoup moins pessimiste que le calcul standard que peut fournir un autre produit** (Matlab, Python...). Mais martelons, que ce n'est que son produit avec la 'backward error', appelée 'forward error', qui a un intérêt. Et uniquement, dans le cadre d'une résolution de système linéaire via MUMPS.*

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

7.2.2 Périmètre d'utilisation

C'est un solveur linéaire universel. Il est déployé pour toutes les fonctionnalités de *Code_Aster.*
D'autre part, l'usage de solveur souffre de petites limitations peu fréquentes et que l'on peut contourner aisément le cas échéant.

- En mode POURSUITE on ne sauvegarde sur fichier que les objets FORTRAN de code_aster et donc pas les occurrences de produits externes (MUMPS, PETSc). Donc attention à l'usage des commandes éclatées dans ce cadre (NUME_DDL/FACTORISER/RESOUDRE). Avec MUMPS, il n'est pas possible de FACTORISER ou de RESOUDRE un système linéaire construit lors d'un run Aster précédent (avec NUME_DDL).
- De même, on limite le nombre d'occurrences simultanées de MUMPS à NMXINS=50. Lors de la construction de son problème *via* des commandes éclatées, l'utilisateur doit veiller à ne pas dépasser ce chiffre, sinon le calcul s'arrête en erreur. Pour ce faire, on peut par exemple détruire les concepts utilisateur associés aux matrices (par exemple *via* la commande del dans le .comm, cf. cas-test mumps03a.comm).
- Avec PETSc, par contre, ce chiffre est limité à 5, car la structure de données est potentiellement plus riche et cette bibliothèque de solveurs itératifs n'est communément pas utilisée dans les commandes éclatées (uniquement dans les opérateurs implicites résolvant un ou deux systèmes linéaires simultanément).

7.2.3 Paramètre RENUM

Ce mot clé permet de contrôler l'outil utilisé pour renuméroter le système linéaire[11]. L'utilisateur *Aster* peut choisir différents outils répartis en deux familles: les outils «frustres» dédiés à un usage et fournis avec MUMPS ('AMD', 'AMF', 'QAMD', 'PORD'), et, les bibliothèques plus «riches» et plus «sophistiquées» qu'il faut installer séparément ('PARMETIS'/'METIS', 'PTSCOTCH'/'SCOTCH').

Le **choix du renuméroteur** a une grande importance sur les consommations mémoire et temps du solveur linéaire. Si on cherche à **optimiser/régler les paramètres numériques** liés au solveur linéaire, ce paramètre doit être **un des premiers à essayer**.

Le produit MUMPS décompose ses calculs en trois étapes (cf. [R6.02.03] §1.6): phase d'analyse, de factorisation numérique et de descente-remontée. Dans certains cas, **l'étape d'analyse peut s'avérer prédominante**. Soit parce que le problème est numériquement difficile (nombreux blocages, liaisons ou zones de contact, éléments incompressibles...), soit parce que les deux autres étapes ont été très réduites grâce au parallélisme (cf. [U2.08.06]). Il peut être alors intéressant de paralléliser cette étape d'analyse (*via* MPI). Cela permet de l'accélérer et de baisser sa consommation mémoire. Ceci s'effectue en choisissant un des renuméroteurs parallèles proposés : 'PARMETIS' ou 'PTSCOTCH'.
Le choix d'un tel renuméroteur parallèle, si il améliore souvent les performances de l'étape d'analyse de MUMPS, peut néanmoins dégrader celles des deux autres étapes MUMPS suivantes. Néanmoins, si le problème principal était de réduire la concommation mémoire de cette étape d'analyse ou si les étapes suivantes de MUMPS bénéficient de suffisamment de parallèlisme (ou de compression, cf. paramètres ACCELERATION/LOW_RANK_SEUIL), le bilan peut être globalement positif.

D'autre part, lors de **calculs modaux parallèles** (opérateur CALC_MODES), on a parfois observé des speed-ups décevants du fait d'un choix inapproprié de renuméroteur. Dans ce cas là on a constaté que le choix d'un renuméroteur « sophistiqué » était contre-performant. Il vaut mieux imposer à MUMPS un simple 'AMF' ou 'QAMD', plutôt que 'METIS' ou 'PARMETIS' (souvent pris automatiquement en mode 'AUTO').

7.2.4 Paramètre ELIM_LAGR2

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

11 Afin, notamment, de limiter le phénomène de remplissage de la factorisée, cf. [R6.02.03] §1.7.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

des Lagranges (AFFE_CHAR_MECA/THER…) a été modifiées en introduisant des doubles Lagranges. Formellement, on ne travaille pas avec la matrice initiale $\mathbf{K}_0$

$$\mathbf{K}_0=\begin{bmatrix}\mathbf{K} & \text{blocage}\\ \text{blocage} & \mathbf{0}\end{bmatrix}\begin{matrix}\mathbf{u}\\ \text{lagr}\end{matrix}$$

mais avec sa forme doublement dualisée $\mathbf{K}_2$

$$\mathbf{K}_2=\begin{bmatrix}\mathbf{K} & \text{blocage} & \text{blocage}\\ \text{blocage} & -\mathbf{1} & \mathbf{1}\\ \text{blocage} & \mathbf{1} & -\mathbf{1}\end{bmatrix}\begin{matrix}\mathbf{u}\\ \text{lagr}_1\\ \text{lagr}_2\end{matrix}$$

D'où un surcoût mémoire et calcul.
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**[12]. D'où une matrice de travail $\mathbf{K}_1$ simplement dualisée

$$\mathbf{K}_1=\begin{bmatrix}\mathbf{K} & \text{blocage} & \mathbf{0}\\ \text{blocage} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & -\mathbf{1}\end{bmatrix}\begin{matrix}\mathbf{u}\\ \text{lagr}_1\\ \text{lagr}_2\end{matrix}$$

plus petite car les termes extra-diagonaux des lignes et des colonnes associées à ces Lagranges spectateurs sont alors initialisées à zéro. *A contrario*, avec la valeur 'NON', MUMPS reçoit les matrices dualisées usuelles.

Pour les **problèmes comportant de nombreux Lagranges (jusqu'à 20% du nombres d'inconnues totales)**, l'activation de ce paramètre est souvent payante (matrice plus petite). Mais lorsque **ce nombre explose (>20%)**, ce procédé peut-être contre-productif. Les gains réalisés sur la matrice sont annulés par la taille de la factorisée et surtout par le nombre de pivotages tardifs que MUMPS doit effectuer. Imposer ELIM_LAGR2='NON' peut être alors très intéressant (par exemple : gain de 40% en CPU sur le cas-test mac3c01).
On **débranche** aussi temporairement ce paramètre lorsqu'on souhaite calculer **le déterminant de la matrice**, car sinon sa valeur est faussée par ces modifications des termes de blocage. L'utilisateur est averti de cette modification automatique de paramétrage par un message dédié (visible en INFO=2 uniquement).

7.2.5 Paramètre RESI_RELA

Valeur par défaut=-1.d0 en non linéaire et en modal, 1.d-6 en linéaire.
Ce paramètre est désactivé par une valeur négative.

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 lorsque la solution n'est pas elle même corrigée par un autre processus algorithmique (algorithme de Newton, détection de singularité…) bref dans les opérateurs linéaires THER_LINEAIRE et MECA_STATIQUE. En non linéaire, le critère de détection de singularité et la correction de Newton sont des garde-fous suffisants. On peut donc débrancher ce processus de contrôle (c'est ce qui est fait par défaut *via* la valeur -1). En modal, cette détection de singularité est un outil algorithmique pour capturer les modes propres. Cette détection fait l'objet d'un paramétrage dédié propre à chaque méthode.
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.
L'activation de ce mot-clé initie aussi un processus de raffinement itératif dont l'objectif est d'améliorer la solution obtenue. Ce post-traitement bénéficie d'un paramétrage particulier (mot-clé

12 Pour maintenir la cohérence des structures de données et garder une certaine lisibilité/maintenabilité informatique, il est préférable de «bluffer» le processus habituel en passant de $\mathbf{K}_2$ à $\mathbf{K}_1$, plutôt qu'au scénario optimal $\mathbf{K}_0$.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

POSTTRAITEMENTS). C'est la solution résultant de ce processus d'amélioration itérative qui est testée par RESI_RELA.

Remarque:

- *Ce processus de contrôle implique l'estimation du conditionnement matriciel et quelques descentes-remontées du post-traitement de raffinement itératif. Il peut donc être assez coûteux, notamment en OOC, du fait des I/O RAM/disque lors des descentes-remontées (jusqu'à 40%). Lorsque suffisamment de garde-fous sont mises en œuvre on peut le débrancher en initialisant `resi` à une valeur négative.*

7.2.6 Paramètres pour optimiser la gestion mémoire (MUMPS et/ou JEVEUX)

En général une grande partie des temps calcul et des pics mémoires RAM d'une simulation *Code_Aster* sont imputables aux solveurs linéaires. Le solveur linéaire MUMPS n'échappe pas à la règle mais la richesse de son paramétrage interne et son couplage fin avec *Code_Aster* ménagent une certaine souplesse à l'utilisateur. Notamment en ce qui concerne la gestion de la consommation en mémoire RAM.

En fait, **lors d'un pic mémoire** survenant pour une résolution de système linéaire *via* MUMPS, la mémoire RAM peut se décomposer en 5 parties:

- Les objets JEVEUX hormis la matrice,
- Les objets JEVEUX associés à la matrice (généralement les SD MATR_ASSE et NUME_DDL),
- Les objets MUMPS permettant de stocker la matrice,
- Les objets MUMPS permettant de stocker la factorisée,
- Les objets MUMPS auxiliaires (pointeurs, vecteurs, buffers de communication...).

Grossièrement, la partie 4 est la plus encombrante. En particulier, elle est beaucoup plus grosse que les parties 2 et 3 (facteur d'au moins 30 dû au phénomène de remplissage cf. [R6.02.03]). Ces dernières sont équivalentes en taille mais l'une s'exerce dans l'espace dévolu à JEVEUX, tandis que l'autre, vit dans l'espace complémentaire alloué par le gestionnaire de tâche au job. Quant aux deux autres parties, 1 et 5, elles jouent souvent un rôle marginal[13].

Pour diminuer ces consommations mémoire, l'**utilisateur *Aster* dispose de plusieurs bras de levier** (la plupart du temps cumulables):

- **Le calcul parallèle centralisé** sur n coeurs (partie 4 divisée par n) ou distribué (parties 3 et 4 divisées par n).
- **Le calcul parallèle distribué** + **MATR_DISTRIBUEE** (périmètre d'utilisation limité): parties 3 et 4 divisées par n , partie 2 divisée par un peu moins de n .
- L'activation de l'**OOC de MUMPS** (mot-clé GESTION_MEMOIRE='OUT-OF-CORE'): la partie 4 diminue des 2/3.
  Sachant qu'avant chaque appel au cycle 'analyse+factorisation numérique' de MUMPS, on décharge sur disque systématiquement les plus gros objets JEVEUX de la partie 2.

Conseils:
Au vu des éléments précédent, une tactique évidente consiste à passer le calcul en mode parallèle distribué (valeur par défaut) plutôt qu'en séquentiel. Le mode parallèle centralisé n'apportant rien de ce point de vue[14], il n'est pas à envisager. Si toutefois, on reste tributaire d'un mode particulier, voici les conseils associés à chacun :

- **En mode séquentiel:** les principaux gains viendront de l'activation de GESTION_MEMOIRE='OUT_OF_CORE' (sur des problèmes de taille raisonnable).

13 Mais pas toujours. Ainsi la partie 1 peut finir par être coûteuse lorsqu'on calcule beaucoup de pas de temps, lorsqu'on charge des champs projetés ou lorsqu'une loi de comportement manipule beaucoup de variables internes. De même, la partie 5 peut devenir non négligeable lors d'un calcul parallèle sur un grand nombre de processeur.

14 Il est principalement utilisé pour tester et valider de nouveaux développements ainsi que des études parallèles.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

- **En mode parallèle distribué:** passé une dizaine de processeurs, l'OUT_OF_CORE ne procure plus beaucoup de gain. MATR_DISTRIBUEE peut alors aider dans certaines situations.

Si ces stratégies n'apportent pas suffisamment de gains, on peut aussi essayer en non linéaire (au prix d'éventuels pertes de précision et/ou de temps) de relaxer la résolution du système linéaire (FILTRAGE_MATRICE, MIXER_PRECISION) voire celles du processus englobant (matrice tangente élastique, réduction de l'espace de projection en modal...). Si la matrice est bien conditionnée et si on n'a pas besoin de détecter les singularités du problème (donc pas calcul modal, flambement...), on peut aussi tenter un solveur itératif (GCPC/PETSC+LDLT_SP).

Solution	Gain en mémoire RAM	Surcoût en temps	Perte de précision
Parallélisme	+++	Au contraire, gain en temps	Aucune
GESTION_MEMOIRE= 'OUT_OF_CORE'	++	Faible sauf si nombreuses descente-remontées	Aucune
MATR_DISTRIBUEE (périmètre limité)	+	Aucun	Aucune
Relaxation des résolutions FILTRAGE_MATRICE, MIXER_PRECISION (périmètre limité)	++	Variable	Variable. Possibilité de non convergence.
Changer de solveur: GCPC/PETSC+LDLT_SP (périmètre limité)	+++	Variable	Variable. Possibilité de non convergence.

Tableau 7.2-1. Synoptique des différentes solutions permettant d'optimiser la mémoire lors d'un calcul avec MUMPS.

Mot-clé GESTION_MEMOIRE='OUT_OF_CORE'
Pour activer ou désactiver les facultés OOC de MUMPS qui va alors décharger entièrement sur disque la partie réelle des 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.

Attention: Lors d'un calcul parallèle, si le nombre de processeurs est important, la taille des objets MUMPS déchargeables sur disque devient faible. Le passage en OOC peut alors s'avérer contre-productif (gain faible en RAM et surcoût en temps) par rapport au mode IC paramétré par défaut.
Ce phénomène se produit d'autant plus précocement que la taille du problème est faible et il est d'autant plus sensible qu'on effectue beaucoup de descentes-remontées[15] dans le solveur.
Grosso modo[16], en dessous de 50.10^3 degrés de liberté par processeurs et si on dispose de suffisamment de RAM (3 ou 4 Go) par processeur, on peut sans doute basculer en IC.

Remarques:

- *Pour l'instant, lors d'une exécution MUMPS en OOC, 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 degrés de liberté), même avec cet OOC, des contingences mémoires peuvent empêcher le calcul. On sort en principe avec une ERREUR_FATALE documentée.*

15 Par exemple, dans un STAT_NON_LINE avec beaucoup d'itérations de Newton et/ou des itérations de raffinement itératif.
16 Cela dépend beaucoup de l'étude.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

- *Dans MUMPS, de manière à optimiser l'occupation mémoire, l'essentiel des entiers est codé en INTEGER*4. Seuls les entiers correspondant à une adresse mémoire sont transcrits en INTEGER*8. Cela permet d'adresser des problèmes de plus grande taille, sur des architectures 64 bits. cette cohabitation entiers courts/longs pour optimiser la place mémoire a été étendue à certains gros objets JEVEUX.*
- *Lorsque Code_Aster a fini d'assembler la matrice de travail, avant de passer « le relais » à MUMPS, il décharge sur disque les plus gros objets JEVEUX liés à la résolution du système linéaire (SMDI/HC, DEEQ, NUEAQ, VALM...). Et ce afin de laisser le plus de place RAM possible à MUMPS. En mode GESTION_MEMOIRE='AUTO', si cette place mémoire n'est pas suffisante pour que MUMPS fonctionne en IC, on complète cette libération partielle d'objets JEVEUX par une libération générale de tous les objets libérables (c'est-à-dire non ouvert en lecture/écriture). Cette opération peut procurer beaucoup de gains lorsqu'on « traîne » en mémoire beaucoup d'objets JEVEUX périphériques (projection de champs, long transitoire...). Par contre, ce déchargement massif peut faire perdre du temps. En particulier en mode parallèle du fait d'engorgements des accès coeurs/RAM.*

Mot-clé MATR_DISTRIBUEE

Ce paramètre est utilisable dans les opérateurs MECA_STATIQUE, STAT_NON_LINE, DYNA_NON_LINE, THER_LINEAIRE et THER_NON_LINE avec AFFE_CHAR_MECA ou AFFE_CHAR_CINE. Il n'est actif qu'en parallèle distribué (AFFE_MODELE/ DISTRIBUTION/ METHODE égal autre que **CENTRALISE)**. Cette fonctionnalité est bien sûr cumulable avec GESTION_MEMOIRE='OUT_OF_CORE', d'où une plus grande variété de fonctionnement pour s'adapter aux contingences d'exécution.

En mode parallèle, lorsqu'on distribue les données JEVEUX en amont de MUMPS, on ne redécoupe pas forcément les structures de données concernées. Avec l'option MATR_DISTRIBUEE='NON', tous les objets distribués sont alloués et initialisés à la même taille (la même valeur qu'en séquentiel). Par contre, chaque processeur ne va modifier que les parties d'objets JEVEUX dont il a la charge. Ce scénario est particulièrement adapté au mode parallèle distribué de MUMPS (mode par défaut) car ce produit regroupe en interne ces flots de données incomplets. Le parallélisme permet alors, outre des gains en temps calcul, de réduire la place mémoire requise par la résolution MUMPS mais pas celle nécessaire à la construction du problème dans JEVEUX.

Ceci n'est pas gênant tant que l'espace RAM pour JEVEUX reste très inférieur à celui requis par MUMPS. Comme JEVEUX stocke principalement la matrice et MUMPS, sa factorisée (généralement des dizaines de fois plus grosse), le goulet d'étranglement RAM du calcul est théoriquement sur MUMPS. Mais dès qu'on utilise quelques dizaines de processeurs en MPI et/ou qu'on active l'OOC, comme MUMPS distribue cette factorisée par processeur et décharge ces morceaux sur disque, la «balle revient dans le camp de JEVEUX».

D'où l'option MATR_DISTRIBUEE qui retaille la matrice, au plus juste des termes non nuls dont a la responsabilité le processeur. L'espace JEVEUX requis diminue alors avec le nombre de processeurs et descend en dessous de la RAM nécessaire à MUMPS. Les résultats de la figure 7.2-1 illustrent ce gain en parallèle sur deux études: une Pompe RIS et la cuve Epicure.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

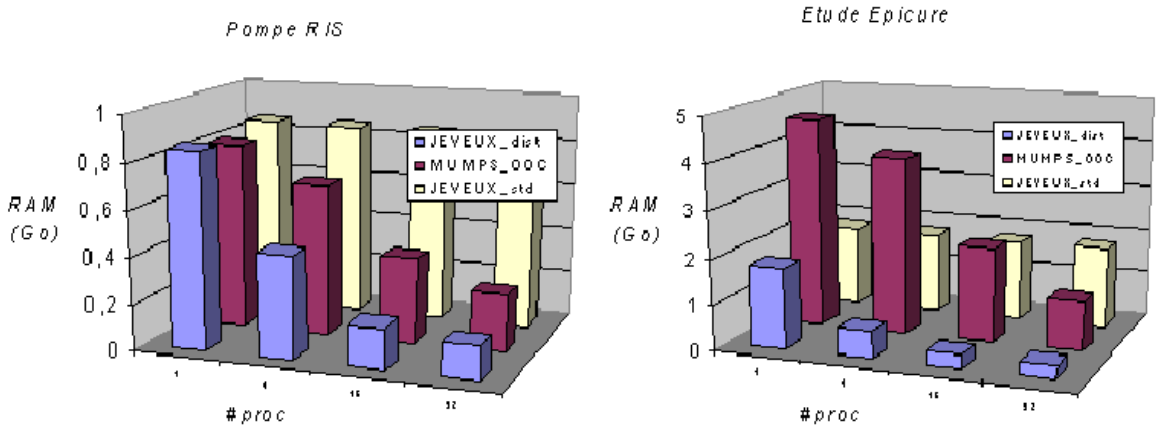


Figure 7.2-1: Évolution des consommations RAM (en Go) en fonction du nombre de processeurs, de Code_Aster v11.0(JEVEUX standard MATR_DISTRIBUE='NON' et distribué, resp. 'OUI') et de MUMPS OOC. Calculs effectués sur une Pompe RIS (perf009) et sur la cuve de l'étude Epicure (perf011).

Remarques:

- *On traite ici les données résultant d'un calcul élémentaire (RESU_ELEM et CHAM_ELEM) ou d'un assemblage matriciel (MATR_ASSE). Les vecteurs assemblés (CHAM_NO) ne sont pas distribués car les gains mémoire induits seraient faibles et, d'autre part, comme ils interviennent dans l'évaluation de nombreux critères algorithmiques, cela impliquerait trop de communications supplémentaires.*
- *En mode MATR_DISTRIBUE, pour faire la jointure entre le bout de MATR_ASSE local au processeur et la MATR_ASSE globale (que l'on ne construit pas), on rajoute un vecteur d'indirection sous la forme d'un NUME_DDL local.*

7.2.7 Paramètres pour réduire le temps calcul *via* différentes techniques d'accélération/compression

Mots-clés ACCELERATION et LOW_RANK_SEUIL

Ces deux mots-clés activent et de pilotent des techniques d'accélération/compression de MUMPS . Celles-ci peuvent réduire significativement le temps de calcul de grosses études, et ce, sans restriction du périmètre d'utilisation, et, avec potentiellement peu ou pas d'impact sur la précision, la robustesse et le comportement global de la simulation.
Elles ne sont généralement intéressantes que sur des problèmes de grandes tailles (N au moins > 2.10 ^{6} ddls) . Les **gains constatés** sur quelques cas-tests de performance et études Aster varient de **20% à 80%** (exemples des figures 7.2-2/7.2-3). Ils augmentent avec la taille du problème, son caractère massif et ils sont complémentaires de ceux procurés par le parallélisme et le renumeroteur.

On complète ici le descriptif de ces fonctionnalités du document [U4.50.01].

Lorsqu'on a choisi une accélération basée sur les compressions 'low-rank' (valeurs 'LR' et 'LR+' du mot-clé ACCELERATION), il faut définir le taux de ces dernières. Ce taux est renseigné par le mot-clé LOW_RANK_SEUIL . Il pilote le critère de troncature de l'algorithme numérique de compression[17]. *Grosso modo* , plus ce chiffre est grand, par exemple 10 $^{-12}$ ou 10 $^{-9}$, plus la compression va être importante et donc plus les gains en temps peuvent être intéressants.
Cependant, si cette valeur est trop grande (par exemple>10 $^{-9}$), la matrice factorisée peut être trop approximée et le vecteur solution s'avérer ainsi trop imprécis ! Dans un processus non linéaire ce n'est pas toujours si grave car l'algorithme de Newton englobant peut corriger le tir !
Par contre, en linéaire ou pour traiter des problèmes numériquement difficiles (éléments finis incompressibles, X-FEM...), il faut alors s'assurer que la résolution inclus bien la procédure de post-

17 Variante avec troncature d'une factorisation QR: RRQR pour 'Rank Revealing QR'.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

traitements corrective (cf. raffinement itératif, mot-clé POSTTRAITEMENTS). En l'occurence, dans ce cas de figure où lorsqu'on cherche à établir un compromis entre performance et précision, la valeur POSTTRAITEMENTS='MINI' (+ RESI_RELA<0) est souvent à privilégier par rapport au choix par défaut (POSTTRAITEMENTS='AUTO' + RESI_RELA>0).

Pour être exhaustif, notons que la valeur réelle de ce mot-clé peut aussi être nulle, la compression se fera alors à la précision machine près, ou devenir négative, la compression utilisera alors le seuil relatif

$$\|\mathbf{K}\| \times |\text{lr_seuil}|$$

La première valeur permet de bénéficier d'un peu de compression sans aucun impact sur la précision (pour des tests fonctionnels ou des expertises).

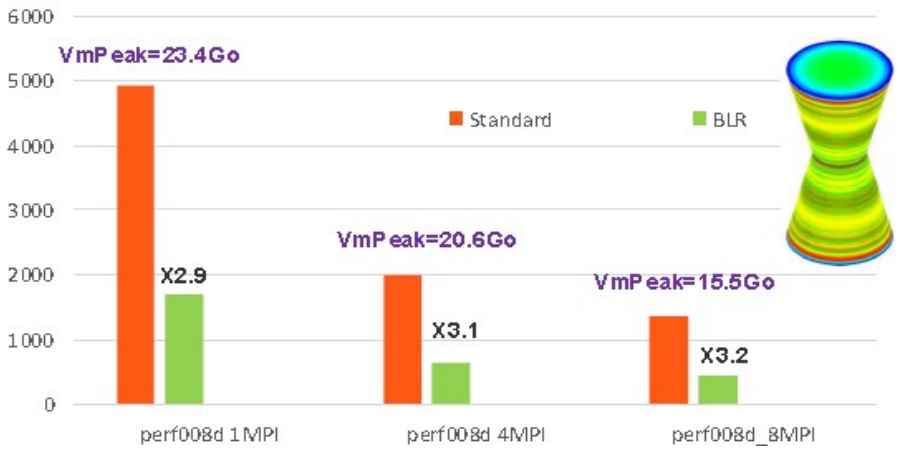


Figure 7.2-2: Exemple de gains procurés par les compressions low-rank sur le cas test de performance perf008d (paramètres par défaut, gestion mémoire en OOC, N=2M, NNZ=80M, Facto_METIS4=7495M, conditionnement=10^7). On trace, en fonction du nombre de processus MPI activés, les temps elapsed consommés par toute l'étape de résolution de système linéaire dans Code_Aster v13.1, son pic mémoire RAM, ainsi que le facteur d'accélération procuré par BLR.

En résumé, le critère d'approximation des algorithmes de compression de MUMPS est fixé suivant la règle suivante :

Si LOW_RANK_SEUIL=0.D0:
troncature à la précision machine près.
Si LOW_RANK_SEUIL>0.D0:
la troncature des algorithmes de compression utilise directement lr_seuil .
Si LOW_RANK_SEUIL<0.D0:
la troncature des algorithmes de compression est basée sur le seuil relatif rappelé ci-dessus.

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)

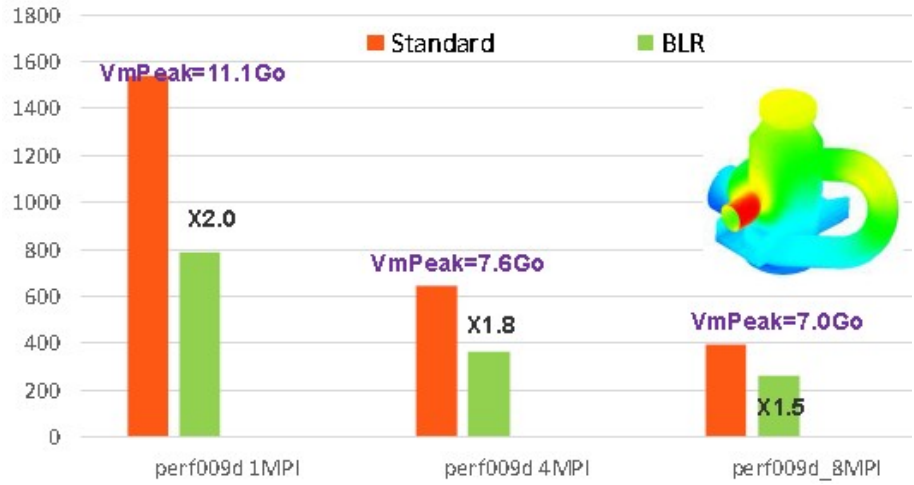


Figure 7.2-3: Exemple de gains procurés par les compressions low-rank sur le cas test de performance perf009d (paramètres par défaut, gestion mémoire en OOC, N=5.4M, NNZ=209M, Facto_METIS4=5247M, conditionnement=10^8). On trace, en fonction du nombre de processus MPI activés, les temps elapsed consommés par toute l'étape de résolution de système linéaire dans Code_Aster v13.1, son pic mémoire RAM, ainsi que le facteur d'accélération procuré par BLR.

Remarques:

• Pour l'instant, les gains des compressions low-rank ne concernent que la deuxième étape de calcul de MUMPS, celle de factorisation numérique, qui est souvent la plus coûteuse. Ces gains dépendent donc de l'importance de cette étape par rapport aux autres étapes du solveur linéaire (construction du `NUME_DDL`, analyse et descente-remontée). Rappelons que l'on peut aisément tracer le coût de cette étape de factorisation (item #1.3) en activant le monitoring détaillé des coûts en temps de chaque étape de calcul (mot-clé `DEBUT/MESURE_TEMPS` [U1.03.03]).

• En dehors des outils de compressions en eux-même, la stratégie low-rank implique deux surcoûts, l'un dans l'étape d'analyse[18] et l'autre en fin de factorisation numérique[19]. Mais, sur les gros problèmes, ceux-ci sont généralement vite compensés par les gains procurés par cette technique.

• Pour l'instant ces gains ne concernent que le temps de calcul, les consommations mémoires restent similaires voire légèrement supérieures (vecteurs auxiliaires pour la compression) à celles d'un calcul 'full-rank' standard.

• Pour des seuils de compressions raisonnables (<10^{-9}) l'impact sur la qualité du résultats et sur les 'outputs' connexes (détection de singularité, calcul de déterminant et du critère de Sturm...) sont souvent négligeables[20] . Au delà, ce n'est plus complètement garanti. Le bon comportement et la robustesse du calcul peuvent en souffrir. Ce paramétrage est à limiter à un usage solveur direct "relaxé" ou préconditionneur[21] pour un solveur itératif de Krylov.

18 Pré-sélection des blocs denses éligibles et construction des structures de données connexes.

19 Les blocs denses compressés sont décompressés après usage avant d'aborder la dernière étape de descente-remontée, car celle-ci est encore traitée en 'full-rank'. A termes, elle sera elle-aussi traitée en 'low-rank' et ce surcoût disparaîtra.

20 Concernant, la qualité du résultat c'est d'autant plus vrai, si la procédure de post-traitement a été activée.

21 Fonctionnalités à venir: par exemple pour un usage en temps que solveur direct "relaxé" (cf. mots-clés `FILTRAGE_MATRICE/MIXER_PRECISION`) ou en tant que préconditionneur (cf. methodes `PETSc/GCPC` + `LDLT_SP`) .

Document diffusé sous licence GNU FDL (http://www.gnu.org/copyleft/fdl.html)