Décomposition de Domaine et parallélisme : la méthode FETI

Résumé

Dans le cadre des simulations thermo-mécaniques avec *Code_Aster* , l'essentiel des coûts calcul provient souvent des systèmes linéaires. Pour y pallier, il peut alors être judicieux de scinder le maillage initial en plusieurs sous-domaines sur lesquels vont être conduits des calculs élémentaires, des assemblages et des résolutions parallèles plus efficaces: c'est la **Décomposition de Domaine** (**DD**). Parmi les méthodes DD envisageables, *Code_Aster* a choisi d'intégrer la méthode **FETI-1**.

Dans la première partie de ce document nous résumons **quelques aspects du calcul parallèle** qu'il faut avoir à l'esprit avant d'aborder la problématique des méthodes DD. Puis on essaye de donner les **limitations théoriques et pratiques de l'approche FETI ainsi que des approches concurrentes en calculs hautes performances** pour la mécanique des structures: parallélisme informatique, parallélisme numérique éventuellement *via* une librairie externe, méthodes DD de type Schwarz et Schur. Une fois ces différentes approches explicitées, les **critères de choix de la méthode FETI** sont discutés et pondérés à la lumière de notre expérience de développement et d'utilisation. Nous abordons la problématique connexe du **partitionnement de domaine** , puis nous détaillons les **aspects informatiques et fonctionnels de l'utilisation de FETI** dans *Code_Aster* . Enfin nous concluons par quelques résultats numériques, un bilan du chantier logiciel et des solveurs linéaires disponibles dans le code et des perspectives.

Pour conclure, il faut garder à l'esprit que FETI est un solveur de recherche, donc encore en phase de validation et de fiabilisation. En cas de problème de robustesse et/ou de limite de périmètre d'utilisation, il est conseillé d'utiliser plutôt le solveur MUMPS [U2.08.03/U4.50.01][R6.02.03]. Son niveau de parallélisme est moins ambitieux que celui proposé par la DD mais elle peut souvent procurer des gains intéressants par rapport au solveur linéaire par défaut du code.

Table des Matières

1 Introduction

1.1 Contexte de la simulation en mécanique des structures

Question: Pourquoi croyez vous que les surfaces anguleuses et les grands aplats du bombardier furtif F117, le «diamant volant», ont cédé la place aux courbes harmonieuses et aérodynamiques du non moins célèbre B-2 ? Dans des domaines tout aussi en pointe mais bien moins belliqueux que sont l'automobile ou le nautisme, ce «bio-design» est d'ailleurs devenu la règle.

Réponse: Tout simplement grâce aux fantastiques progrès réalisés ces dernières décennies en modélisation mécanique (avec l'analyse numérique et l'algorithmique associées) et en informatique (aussi bien machine de calcul que technique de programmation).

C'est une «tautologie» que de rappeler que ces avancées ont permis d'accéder souvent au 3D, en intégrant certaines non-linéarités, des problèmes en temps, différentes physiques/échelles/modélisations… D'où des moteurs diesels à injection directe plus économes et plus nerveux, des voitures moins bruyantes, des paquebots aux formes improbables, des ouvrages d'art toujours plus audacieux !
Dans le cadre du développement des mécanismes et des structures, on l'aura donc compris, la part de cette phase simulation devient désormais incontournable. Elle permet la réduction des temps de développement, la limitation du nombre de prototypes et d'essais, avant les tests finaux de validation. Néanmoins, il convient de rester lucide et d'être conscient de la qualité et des limites des modélisations et des calculs qui sont menés. **Même si elle se banalise, la méthode des éléments finis reste un outil complexe et gourmand en capacité de traitement.**
Même avec des maillages et des modélisations optimisés (maillage adaptatif, symétrie, éléments structuraux…), les problèmes à résoudre continuent d'engorger les ressources informatiques. En particulier en statique ou dynamique tridimensionnelles non-linéaires, ou, lorsque le chargement est complexe (cyclique, dynamique rapide…), ou encore, pour effectuer des analyses particulières (recalage ou assimilation de données, optimisation, sensibilité, fiabilité, étude paramétrique…) qui n'utilisent ces problèmes, déjà très coûteux, que comme brique élémentaire d'un processus plus global.
Ces coûts calcul sont généralement issus des deux étapes principales: l'intégration des lois de comportement et les inversions de systèmes linéaires . Le coût de la première dépend du nombre de points d'intégration et de la complexité des lois, tandis que celui de la seconde est lié au nombre d'inconnues, aux modélisations retenues et à la topologie (largeur de bande, conditionnement). Lorsque le nombre d'inconnues explose, la seconde étape devient souvent prépondérante[1] et c'est donc cette dernière qui va principalement nous intéresser ici. D'ailleurs, lorsqu'il est possible d'être plus performant sur cette phase de résolution, grâce à l'accès à une machine parallèle, nous verrons que cet atout se propagera à la phase de constitution du système proprement dite (calculs élémentaires et assemblages).

Ces inversions de systèmes linéaires sont en fait omniprésentes dans le code et souvent enfouies au plus profond d'autres algorithmes numériques: schéma non-linéaire, intégration en temps, analyse modale…. On cherche, par exemple, le vecteur des déplacements nodaux (ou des incréments de déplacement) $\boldsymbol{u}$ vérifiant un système linéaire du type

$$\mathbf{Ku=f} \qquad (1.1\text{-}1)$$

avec $\boldsymbol{K}$ la matrice de rigidité, creuse, assez mal conditionnée[2] et souvent réelle, symétrique définie positive (SPD en anglais). Le vecteur $\boldsymbol{f}$ traduisant l'application des forces généralisées au système mécanique.
Lorsque ce caractère SPD n'est plus de mise avec, par exemple, une matrice $\boldsymbol{K}$ indéfinie du fait des Lagranges utilisés pour les blocages de Dirichlet ou une perte de symétrie due aux modélisations de géomatériaux, la plupart des méthodes abordées dans ce document sont transposables[3] . Par soucis de clarté et de concision, nous resterons néanmoins dans ce cadre SPD plus confortable.

1 Pour *Code_Aster*, voir l'étude de 'profiling' conduite par N.Anfaoui[Anf03].

2 En mécanique des structures le conditionnement $\eta(K)$ est connu pour être beaucoup plus mauvais que dans d'autres domaines, telle la mécanique des fluides par exemple. Il peut varier, typiquement, de 10^5 à 10^{12} et le fait de raffiner le maillage, d'utiliser des éléments étirés ou des éléments structuraux a des conséquences dramatiques sur ce chiffre (cf. B.Smith. *A parallel implementation of an iterative substructuring algorithm for problems in 3D.* SIAM J.Sci.Comput., **14** (1992), pp406-423. §3.1 ou I.FRIED. *Condition of finite element matrices generated from nonuniform meshes*. AIAA J., **10** (1972), pp219-221.)

3 En substituant, par exemple, à un solveur d'interface basé sur le GCPC, un algorithme de GMRES.

Depuis 50 ans, deux types de techniques se disputent la suprématie dans le domaine, les solveurs directs et ceux itératifs (cf. [Che05][Duf06][Gol96][Las98][Meu99][Saa03]). Les premiers sont robustes et aboutissent en un nombre fini d'opérations connu par avance. Leur théorie est relativement bien achevée et leur déclinaison suivant moult types de matrices et d'architectures logicielles est très complète. Cependant, ils requièrent des capacités de stockage qui croissent rapidement avec la taille du problème et ils ne permettent pas un parallélisme massif efficace.

En revanche, les méthodes itératives sont complètement «scalables» lorsque l'on augmente le nombre de processeurs. Leur théorie regorge de nombreux «problèmes ouverts», surtout en arithmétique finie. En pratique, leur convergence en un nombre «raisonnable» d'itérations, n'est pas toujours acquise, elle dépend de la structure de la matrice, du point de départ, du critère d'arrêt…

Ce type de solveurs a plus de mal à percer en mécanique des structures industrielles où l'on cumule souvent hétérogénéités, non-linéarités et jonctions de modèles qui gangrènent le conditionnement de l'opérateur de travail.
Contrairement à leurs homologues directs, il n'est pas possible de proposer LE solveur itératif qui va résoudre n'importe quel système linéaire. L'adéquation du type d'algorithme à une classe de problèmes se fait au cas par cas. Ils présentent, néanmoins, d'autres avantages qui historiquement leur ont donné droit de cité pour certaines applications. A gestion mémoire équivalente, ils en requièrent moins que les solveurs directs, car on a juste besoin de connaître l'action de la matrice sur un vecteur quelconque, sans devoir véritablement la stocker. D'autre part, on n'est pas soumis au «diktat» du phénomène de remplissage qui détériore le profil des matrices, on peut exploiter efficacement le caractère creux des opérateurs et contrôler la précision des résultats [4] .
Bref, l'utilisation de solveurs directs relève plutôt du domaine de la technique alors que le choix du bon couple méthode itérative/préconditionneur est plutôt un art ! En dépit de sa simplicité biblique sur le papier, la résolution d'un système linéaire, même symétrique défini positif, n'est pas «un long fleuve tranquille». Entre deux maux, remplissage/pivotage et préconditionnement, il faut choisir !

1.2 Les solveurs DD

D'où l'idée séduisante de mixer ces deux types de solveur, direct et itératif, au sein d'une troisième catégorie, les solveurs par Décomposition de Domaine (DD) , afin de tirer profit de leurs avantages respectifs. Basées sur le principe (universel !) «diviser pour mieux régner», ces techniques proposent de partitionner[5] un domaine global Ω en plusieurs sous-domaines[6] Ω_i , pour lesquels on va pouvoir résoudre, de préférence sur des processeurs distincts, des problèmes locaux quasi indépendants (cf. figures 1.2-1)

$$K_i u_i = f_i \qquad (1.2\text{-}1)$$

Ces problèmes locaux sont de même nature que le problème initial, on va donc pouvoir tous les inverser par le même solveur. En général, par soucis de robustesse, on fait appel à un solveur direct creux car les conditionnements de ces K_i sont souvent aussi défavorables que celui du problème initial et les tailles de ces matrices restent conséquentes. Parfois un découpage réfléchi peut diminuer significativement la largeur de bande de ces matrices de rigidité locales et donc accélérer d'autant leurs inversions.
Ces inversions locales étant faites, on opère ensuite la jointure des solutions locales u_i , à la frontière des domaines, *via* un solveur d'interface *ad hoc*: inversion du complément de Schur S pour les méthodes primales ou d'un opérateur de type FETI, F_I , pour les duales.
La matrice de ce problème d'interface est par contre bien mieux conditionnée que le problème initial, de plus petite taille et elle combine les données condensées de part et d'autre. Pour faciliter le parallélisme et ne pas avoir à la construire explicitement, on préconise alors souvent d'avoir recourt à une méthode itérative.

4 Ce qui peut être très intéressant dans le cadre de solveurs emboîtés (par ex. Newton + GCPC), cf. V.Frayssé. *The power of backward error analysis*. HDR de l'Institut National Polytechnique de Toulouse (2000).
5 En mécanique des structures, on utilise essentiellement un partitionnement par maille (ou EOD pour 'Element Oriented Decomposition'). Les interfaces entre sous-domaines sont alors constituées d'arêtes ou de facettes. Par opposition au découpage orienté nœuds (VOD pour 'Vertex…').
6 Pour éviter des «allitérations» disgracieuses, nous employons indifféremment dans ce document les termes domaine/structure, sous-domaine/sous-structure, CPU/processeur voire méthode de Schur/sous-structuration. Compte-tenu de leurs contextes d'utilisation, ces amalgames sont naturels.

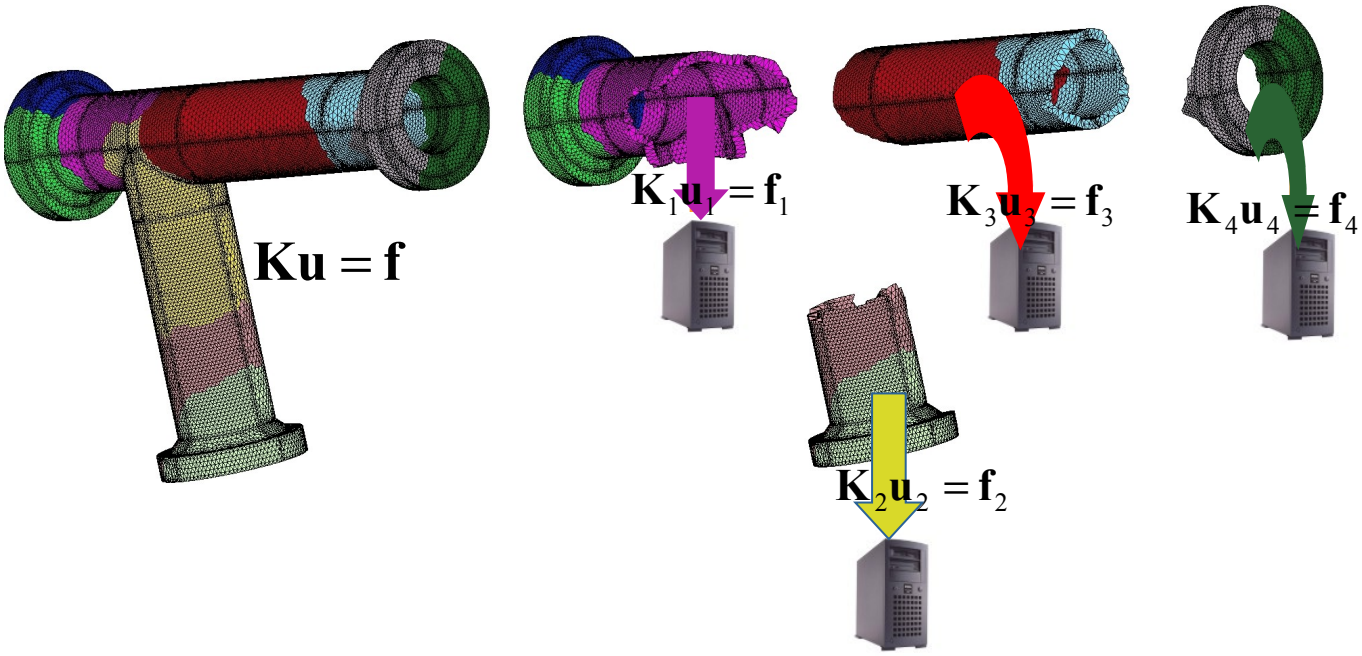


Figure 1.2-1._ Principe de la décomposition de domaine implantée dans Code_Aster, illustré sur une zone de mélange d'un circuit RRA.

Les gains en performances de ces méthodes jouent sur plusieurs facteurs:

•Les consommations mémoire et CPU des solveurs directs qui sont *grosso modo* des fonctions puissances de la taille du problème: c'est le fameux «diviser pour mieux régner ! »,

•Le découpage des flots d'informations en paquets de tailles variables (solveur d'interface, solveur local, Lapack, Blas) pour nourrir plus efficacement les différents niveaux de caches. Leurs temps d'accès étant toujours très supérieurs à la fréquence d'horloge. On essaye ainsi de «doper» l'efficacité des processeurs pour ce type de calculs qui requièrent beaucoup d'accès aux données.

•La limitation au strict minimum du flot de données autour de chaque processeur (celles afférentes aux seules sous-domaines) pour améliorer leurs localisations: «penser globalement mais agir localement».

•Un parallélisme «gros grain»[7] et potentiellement multiniveaux (solveur d'interface, solveur local…) pour tirer la performance et s'adapter au cas d'étude, à la variabilité des machines parallèles et de leurs librairies optimisées.

Finalement, les méthodes DD jouent sur le fait que chaque processeur peut traiter plus efficacement et indépendamment une grande part du travail, et que la reconstruction de la solution globale, *via* le solveur d'interface, n'impacte que faiblement ces gains. Mais elles ont aussi d'autres cordes à leurs arcs ! **L'autre grand intérêt des méthodes DD, outre les aspects performance et stockage, réside dans la modularité et la souplesse qu'elles introduisent dans le processus de modélisation** . Cet aspect «boite à outil» d'appariement de calculs distribués est souvent méconnu. Pourtant, il peut s'avérer tout aussi utile d'un point de vue opérationnel. Car comme les méthodes de sous-structuration dont elles sont issues, les méthodes DD peuvent faciliter:

- Une préparation et une vérification des modèles liés à chaque sous-domaine, indépendamment les uns des autres. C'est particulièrement intéressant pour l'étude de sous-ensembles confiés à différents contractants.
- Des modifications dans un sous-ensemble sans nécessairement entraîner une ré-analyse complète du reste de la structure (modifications structurales, sous-domaines spectateurs).
- Une minimisation des calculs et du stockage des données lorsque la structure présente des répétitivités, des symétries, des translations (cette potentialité est exploitée dans *Code_Aster* en sous-structuration statique et dynamique, cf. par exemple l'opérateur `MACR_ELEM_STAT` §3.2).
- Le traitement de problèmes de tailles quelconques, quelles que soient les capacités mémoires ; Sur le papier du moins, en enchaînant suffisamment de niveaux (DD multiniveau [Esc92]).
- De part l'existence d'une interface, la jonction au sein d'un même calcul, de différents maillages, modélisations, échelles etc.

7 Le volume de calcul à réaliser par processeur reste élevé vis-à-vis de la quantité d'informations à échanger avec les processeurs voisins.

1.3 DD et parallélisme

Revenons un peu sur les aspects parallèles. D'après la fameuse loi de Moore, les capacités calculs des ordinateurs doublent tous les 18 mois. Cette prédiction est toujours d'actualité pour les machines individuelles malgré le défi technologique croissant que cela représente[8] , mais ces gains ne sont plus aussi faciles à réaliser. Cette dynamique d'amélioration était jusqu'à présent obtenue par l'augmentation de la fréquence d'horloge, de la finesse de gravure associée à l'optimisation de l'exécution des instructions et à la mise en œuvre de mémoires caches. Dernièrement, cette course à la performance s'est plutôt traduite par l'intégration de plusieurs processeurs sur la même puce (multi-coeurs) avec un gestion simultanée de flots d'exécutions concurrents (mécanisme de multi-threading).
Cette conversion au parallélisme s'est opérée bien plus tôt pour les calculateurs scientifiques ! Elle leur a permis de surclasser de plusieurs ordre de grandeur la loi de Moore. Ainsi, la puissance installée des 500 plus puissants calculateurs (le fameux Top 500) a explosé de 1 téraflops (soit 10^{12} opérations par seconde) en 1993 à plus de 3 pétaflops (3000 fois plus) en 2006. Soit un surfacteur de plusieurs centaines dû au parallélisme massif des nouvelles architectures regroupant des milliers, voire des centaines de milliers de cœurs.

Encore faut-il exploiter ces performances déduites de cas-tests canoniques sur de véritables études industrielles. ***A minima,* une réflexion doit être menée sur le type de décomposition à retenir** . Tout d'abord, elle doit ménager des flots de données et d'instructions indépendantes, ici centrées sur les sous-domaines. Pour résoudre le problème d'interface, ces sous-domaines doivent échanger des données. Il faut donc prévoir des communications inter ou intra-processeurs, de plus faibles volumes possibles, localisées (par exemple un sous-domaine ne communique qu'avec ses voisins), équi-réparties et pas trop fréquentes. On essaie même parfois de les désynchroniser pour pouvoir recouvrir du temps de communications par du temps calcul. **Mais fondamentalement, les critères dimensionnants sont le volume et la fréquence des communications.**

Au vu de ces critères, **la décomposition par sous-domaines est une bonne approche**: les calculs par sous-domaines sont indépendants et le solveur itératif d'interface ne requiert que des échanges de données entre sous-domaines voisins et limités à leurs frontières. Dans le cadre de la résolution des EDP, cette stratégie DD, dite de Schur-Krylov, n'est pas la seule voie. La décomposition d'opérateurs (cf. algorithme d'Uzawa) ou celle d'espaces fonctionnels (méthodes EF, modales) conduisent aussi à des formulations élégantes qui découplent complètement les problèmes. Cependant, leurs synchronisations ne se limitent pas à des volumes de l'ordre d'une interface. La taille de l'information a transmettre est celle du domaine complet.

Mais même si la stratégie de résolution parallèle des méthodes DD est basée sur le multi-domaines, elle sous-entend les deux autres ! Puisqu'on combine la décomposition d'espace fonctionnel par éléments finis pour discrétiser le problème en espace et celle d'opérateur pour la résolution non-linéaire du problème de mécanique et le préconditionnement du problème d'interface. Nous nous limiterons aux discrétisations par éléments finis qui sont prédominantes dans *Code_Aster* , mais il faut savoir que ces approches DD ont aussi essaimé dans d'autres configurations (méthodes 'meshless', EF discrets etc.).

1.4 Une kyrielle de méthodes

Initialement, l'objectif des méthodes de DD était de pouvoir résoudre analytiquement certaines EDP sur des domaines complexes maillés uniformément. *Via* la sous-structuration, elles sont revenues au goût du jour très longtemps après[9] , pour cette fois résoudre numériquement et plus efficacement des EDPs sur des domaines quelconques maillés non uniformément. L'analyse par éléments finis a ainsi conduit au développement de techniques et d'algorithmes nouveaux.

Classiquement, on les scinde en deux catégories: décomposition en sous-domaines recouvrants (méthodes de Schwarz) ou non (méthodes de Schur) . La deuxième est, de loin, celle qui a connu le plus d'essor en mécanique des structures. Elle se subdivise en méthode primale ou duale suivant le positionnement

8 Notamment en terme de finesse de gravure, de puissance électrique consommée et, par voie de conséquence, de chaleur dissipée par unité de surface.

9 Il est intéressant de remarquer, qu'une de leur première déclinaison industrielle fut en mécanique des structures, pourtant non dominée par les EDP. L'année 1987 marque vraiment le retour sur le devant de la scène, tant en analyse numérique et qu'en mécanique numérique, de ces techniques DD avec la tenue du premier congrès international dédié ('International Conference on DDM') et la création de l'association éponyme[Ddm].

du problème d'interface : en variables primales avec le complément de Schur (champ de déplacement pour la mécanique des structures) ou duales avec l'opérateur FETI (son pendant dual, les inter-efforts à l'interface).
L'arborescence se poursuit avec le type de solveur utilisé à l'interface, **on parlera alors de méthode, directe ou itérative, de Schur primale ou duale.**

Structurellement, ces méthodes sont riches, elles comportent des ingrédients empruntés à l'analyse mathématique des EDP, à l'algèbre linéaire, à la théorie des graphes et bien sûr à la mécanique. Celles qui se sont imposées en mécanique des structures sont la méthode itérative de Schur primale BDD (pour 'Balanced Domain Decomposition', ou sa variante de Neumann-Neumann[LeT91] [LeT94]) et la méthode de Schur duale FETI ('Finite Element Tearing and Interconnecting'[FR91][FR94]). Elles ont fait leurs preuves sur des études frontières pour traiter des problèmes comportant des millions d'inconnues sur des machines à plusieurs milliers de processeurs[Far02]. Leurs périmètres d'utilisation est large: mécanique non linéaire, analyse modale et vibratoire, problème d'évolution, acoustique, dynamique rapide etc. Corollaire de ce succès, chaque adaptation suscite sa variante et plusieurs tentatives de généralisation ont tenté de les réunir, d'où une kyrielle d'approches: FETI-1, FETI-2, FETI-DP, hybride, mixte etc. Dans ce document nous nous limiterons à la méthode FETI initiale, FETI-1, qui est celle développée effectivement dans le code.

1.5 Description du document

Au début du chantier logiciel autour de FETI dans *Code_Aster* , s'est posée la question de l'adaptation de la méthode à l'architecture et aux structures de données du code. C'est une question récurrente en calcul haute performance [10]: Faut il adapter le logiciel existant ou, au contraire, faut il investir dans la réécriture complète dudit code en une version optimisée pour telle ou telle machine ou activité ?
Compte-tenu de l'aboutissement des grands codes actuels, du caractère de plus en plus «consommable» des machines de calcul et des risques de dysfonctionnement entre les différentes versions, séquentielle et parallèle, d'un code (défaut de complétude, résultats faux, non-régression …) **on préfère souvent optimiser les codes industriels existants même si cette refonte n'est pas complètement «up-to-date»** . Les coûts de développement et de maintenance sont évidemment moindres et l'exploitation du logiciel n'a pas besoin de se dédoubler (AQ, versionnement, diffusion, documentation…).

Dans cette optique d'optimisation d'un code existant, des techniques de **parallélisation informatique** (§2.5) et **numérique** (§2.6; multifrontale OpenMP, MUMPS…) ont été introduites. Nous les abordons dans la première partie du document. Mais avant, nous résumons pour les lecteurs néophytes en la matière, quelques **aspects du calcul parallèle** (§2.1/2.4 ; hardware, méthodologie, langages, indicateurs de performance) qu'il faut avoir à l'esprit avant d'aborder la problématique des méthodes DD.
Puis on survole **les grandes classes de méthodes DD (**§3; Schwarz, Schur, Neumann-Neumann et sous-structuration statique *via* l'opérateur *Aster* `MACR_ELEM_STAT`) et la problématique du **partitionnement de domaine (**§6, algorithme, outils et opérateurs `DEFI_PART_FETI/OPS`) avant d'aborder les différents aspects de la **méthode FETI**. On détaille ses grandes étapes, ses variantes algorithmiques, son parallélisme intrinsèque et quelques éléments de théorie (cf. §4 et §5).
Dans une septième partie nous traitons **sa mise en oeuvre et son utilisation dans *Code_Aster*** avant de conclure par quelques **résultats et perspectives**.

Parmi l'abondante bibliographie disponible, on ne saurait que trop conseiller les articles de synthèse qui font référence dans le domaine: ceux de P. Le Tallec[LeT94] et de C.Farhat & F.X.Roux[FR94], puis dernièrement ceux de P.Gosselet & C.Rey[GRe06] ou de Y.Fragakis & M.Papadrakakis[FPa03]. Pour des aspects plus numériques et concernant tous les domaines d'application on recommande les ouvrages de B.Smith et al[Smi96], de J.Kruis[Kru06], de F.Magoulès[Mag07] ou de A.Toselli & O.Widlund[TWi05]. Les portails internet[Ddm][Ora] dédiés à la DD et au HPC sont aussi de vraies mines d'informations (base bibliographique, workshop, articles, annuaires…). Pour une vue plus synthétique sur les méthodes, leurs déclinaisons dans les grands codes de calcul de structures et dans les travaux réalisés en interne EDF R&D on pourra consulter la note [Boi03]. Sur les aspects parallélisme, l'ouvrage de M.J.Quinn[Qui03] propose une entrée en matière aussi riche que ludique.

Figure 1.5 -1._ Logo du site internet ddm.org représentant le problème de Schwarz original

10 HPC pour 'High Performance Computing'.

et celui de l'association ORAP.

2 Parallélisme informatique et numérique

2.1 Architectures parallèles

Qu'est ce que le parallélisme ? Le calcul parallèle consiste à décomposer un calcul en un flot de tâches et/ou de données indépendantes et à gérer leurs interactions afin de tirer parti d'un groupe de processeurs. Ces opérations sont effectuées sur des architectures parallèles variées (PC multi-cœurs, clusters, centre de calcul…) constitués de plusieurs processeurs reliés par un réseau.

2.1.1 Hardware

Bien qu'il existe de nombreuses architectures de machines, on peut souvent les classer en se référant à deux modèles: les multiprocesseurs et les multicomputers.

Les **multiprocesseurs** sont constitués de plusieurs CPU partageant la même mémoire. Ainsi une adresse mémoire sur deux processeurs distincts concerne la même donnée. Ce qui pose des problèmes de cohérence de cache et de synchronisation. Ce modèle se subdivise ensuite en multiprocesseur centralisé ou distribué appelés aussi UMA et NUMA ('Non/Uniform Memory Access'). En effet, pour grouper plus d'une dizaine de CPU sans détériorer la bande passante des accès mémoire, il faut distribuer ces CPU en grappes. Mais ils continuent à partager le même espace mémoire d'où des temps d'accès variables (ce qui justifie le qualificatif de non uniforme pour les multiprocesseurs distribués) et toujours des problèmes de cohérence de cache cornéliens.

Pour pallier ces problèmes de cache est né l'autre modèle de mémoires distribuées, celui des **multicomputers**, ensemble d'ordinateurs à mémoires disjointes reliés par un réseau d'interconnection[11] . On distingue là encore deux sous-classes: les symétriques et les asymétriques suivant que les CPU peuvent remplir toutes la même fonction ou qu'il faille d'abord passer un nœud de calcul dédié qui répartie ensuite le travail aux autres.
Actuellement à EDF R&D, la machine centralisée *Aster* peut donc plutôt se définir comme une collection de multiprocesseurs NUMA alors que les clusters sont des multicomputers.

2.1.2 Taxinomie de Flynn

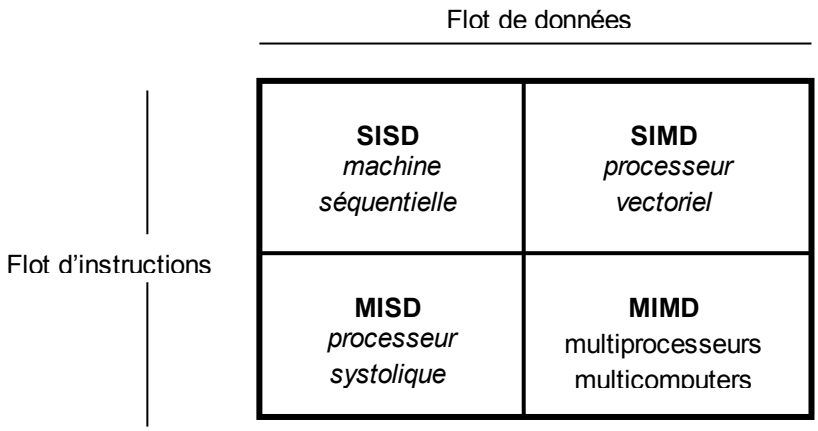


Figure 2.1-1._ Taxinomie de Flynn.

11 Comme pour les CPU, il existe de nombreuses topologies de réseaux (Ethernets, Myrinet, Infiniband…) dont les principales caractéristiques sont la latence (temps d'accès incompressible pour établir une communication) et la bande-passante (vitesse de transmission de l'information). Par exemple, le Gigabit Ethernet a une latence de 100.10^{-6}s et une bande passante de 1 Gbit/s à comparer, respectivement, aux 7.10^{-6} s et 2 Gbit/s du Myrinet. Il ne s'agit ici que de performances matérielles crêtes, en pratique il faut aussi composer avec d'autres contingences, par exemple les différentes implémentations d'un langage parallèle qui orchestrent lesdites communications.

Dès le début du calcul parallèle (les années 60 !) on a classifié les machines parallèles en quatre catégories suivant le parallélisme qu'elles exhibent sur les flots de données et les flots d'instructions (cf. figure 2.1-1): les SISD (pour 'Single Instruction Single Data') qui n'exploitent qu'un seul flot d'instructions et un flot de données (machine à monoprocesseur), les SIMD ('…Multiple Data') tels les anciennes machines vectorielles, les MISD et les MIMD. Cette classification attribuée à Flynn est souvent reprise dans la littérature mais n'a plus guère d'utilité pratique car la plupart des ordinateurs actuels sont des MIMD et peuvent donc gérer concurremment plusieurs flots d'instructions et de données.

2.2 Méthodologie

Pour paralléliser un code existant ou en concevoir un nouveau, il est toujours profitable d'avoir en tête quelques principes méthodologiques, quelques bonnes recettes ! Parmi celles qui ont fait leur preuve, on retrouve la **méthode de Foster** [Fos95] (appelée encore modèle «tâche/canal» ou 'task/channel' cf. figure 2.2-1) basée sur quatre étapes: le **partitionnement** , les **communications**, **l'agglomération** et **l'assignation des tâches à chaque processeur** (le 'mapping').
La première phase consiste à décomposer une exécution en un ensemble de tâches élémentaires associées à des données locales (par exemple, les instructions codant l'intégration d'une loi de comportement sur un sous-domaine, c'est l'aspect «tâche» du modèle) qui peuvent communiquer ensemble (d'où le canal de communication). Si ce partitionnement est basé sur les données (comme c'est la cas en mécanique des structures), on parle d'algorithme par décomposition de domaine. La stratégie complémentaire étant de diviser d'abord le flot d'instructions en différents paquets indépendants et ensuite de leurs associer des données: c'est la décomposition fonctionnelle. Celle-ci est plutôt utilisé pour les applications temps-réel ou des IHMs.

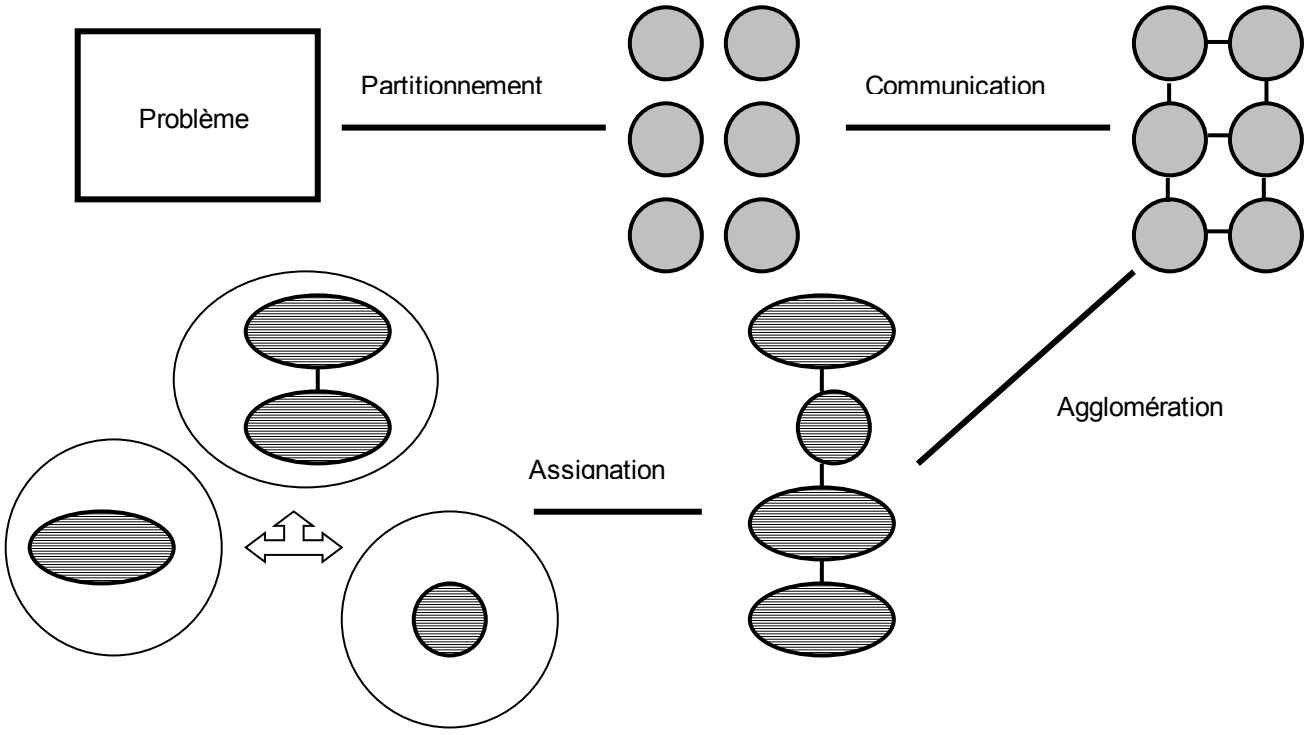


Figure 2.2-1._ Méthodologie de construction d'un code parallèle: modèle «tâche/canal» de Foster.

Cette décomposition doit suivre un cahier des charges précis:
- Etre la plus fine possible car c'est sa granularité qui va borner les gains en performance. Typiquement, un ordre de magnitude supérieur au nombre de processeurs disponibles pour faciliter par exemple l'équilibrage de charge.
- Eviter les redondances de données et d'instructions.
- Etre relativement équilibrée.
- Etre proportionnelle à la taille du problème pour permettre la résolution de problèmes plus gros en augmentant le nombre de CPUs ('scale up').

Ensuite on va établir le schéma de communication entre ces tâches élémentaires en essayant de les équi-répartir, de les localiser (chaque tâche ne communique qu'à ses voisines) et, éventuellement, de les désynchroniser afin qu'elles puissent s'effectuer concurremment.

Ces deux premières phases doivent dégager le plus de parallélisme possible. Il faut ensuite adapter ce parallélisme à la plate-forme matérielle retenue et au code cible. C'est l'objectif des deux dernières phases qui vont agglomérer et répartir par CPU les tâches élémentaires définies précédemment. La combinaison de ces tâches en tâches dites consolidées a plusieurs avantages:

1) Diminuer les surcoûts de communications et accroître la localisation des données (une tâche n'a plus forcément à attendre l'achèvement d'une précédente pour disposer de la donnée).
2) Ménager un équilibrage de charge empirique entre les processeurs.
3) Optimiser l'utilisation des CPU tout en minimisant leurs communications.
4) Réaliser un compromis entre ces contraintes informatiques et les coûts d'adaptation du code.

Ce modèle théorique est très souple, proche de la philosophie de développement en MPI (cf. §2.3) et peut s'adapter à des approches parallèles hybrides (parallélisme par directives et par messages). Il a le mérite de formaliser l'écriture d'algorithmes parallèles maximisant le ratio calcul/communication de chaque processeur, ce qui est donc particulièrement adapté aux architectures distribuées actuelles. Nous reviendrons sur ce modèle lorsque nous décrirons l'architecture du calcul FETI dans *Code_Aster*.

2.3 Langages parallèles

D'un point de vu langage, pour mettre en œuvre le parallélisme dans un code, plusieurs voies sont prospectées:

- **Etendre un compilateur existant pour qu'il traduise automatiquement des programmes séquentiels** en programmes parallèles. C'est l'objectif du parallélisme informatique que nous décrirons au §2.5.
- **Etendre un langage existant en y incorporant de nouvelles fonctionnalités décrivant et organisant le parallélisme.** C'est par exemple l'incorporation de directives OpenMP[Omp] ('Open Multi Processing') ou de routines MPI[Mpi] ('Message Passing Interface') dans un code C ou Fortran. Cela semble de loin la voie la plus répandue par sa facilité de mise en œuvre, sa portabilité et sa réutilisation de l'existant. Mais l'insertion de ces fonctions de bas niveau dans un code doit s'appuyer sur une approche méthodologique et une étude d'impact rigoureuses pour rester pérenne, évolutive et robuste. Et si elles perturbent peu l'environnement de travail du développeur, celui-ci ne bénéficie plus de l'aide du compilateur pour débroussailler les bugs spécifiquement parallèles qui peuvent s'avérer particulièrement tortueux et instables à reproduire (cf. la qualification MPI de *Code_Aster*) !
- **Ajouter une nouvelle couche de langage parallèle au dessus de langages existants** . Suivant ce modèle, un programme parallèle se dissocie en deux niveaux: la couche de bas niveau reprend le code séquentiel existant, à charge pour la couche supérieure de créer et de gérer les flots parallèles d'instructions et de données. Pour finir, un compilateur *ad hoc* traduit ces deux niveaux en code exécutable par une machine parallèle. Cette approche se retrouve dans les langages CODE ('Computationally Oriented Display Environment') et HENCE ('Heterogeneous Network Computing Environment'). Comme elle impose au programmeur d'apprendre un nouveau langage de programmation non normalisé, cette démarche n'a pas connue de grands succès.
- **Créer de nouveaux langages** exprimant explicitement le parallélisme. Il faut alors développer son code «from scratch» à partir des nouveaux concepts mis en œuvre dans ces langages. Les exemples les plus connus sont OCCAM, HPF ('High Performance Fortran') et C*. Malgré de grandes potentialités, une trop faible portabilité limite encore leur progression. Quitte à s'appuyer sur un nouveau langage parallèle, les nouveaux codes préfèrent souvent se lier à des bibliothèques d'algèbre linéaire de plus haut niveau, intégrant structures de données réparties et solveurs (linéaire, modaux…) parallèles. Parmis ces toolkits on compte, PETSc, TRILINOS et HYPRE pour les américaines, Numerical Platon et Arcane pour les hexagonales ! Ces aspects seront détaillés dans le paragraphe sur le parallélisme numérique (cf. §2.6).

Actuellement, bien que de nombreux travaux continuent de faire progresser la cause des compilateurs et des langages parallèles, l'approche véritablement plébiscitée reste la deuxième: enrichir un code C, C++ ou Fortran de fonctions de bas niveaux, type OpenMP ou MPI. Ces langages peuvent être utilisés sur tout type de plate-formes: grappe de multiprocesseurs, PC multi-coeurs ou multicomputers.

2.3.1 Parallélisme par directives

OpenMP est adapté aux machines à mémoire partagée ou globalement adressable (multiprocesseurs). Il regroupe un ensemble de directives de compilation qui précisent les zones séquentielles et parallèles du

programme (‘multithreading’). Les séquentielles étant les parties qui ne doivent être exécutées qu’en séquentiel pour respecter la logique du programme. A l’intérieur des zones parallèles, d’autres directives répartissent la charge entre processeurs et jouent un rôle dans l’allocation mémoire. En effet, certaines variables doivent rester globales et n’exister qu’en un seul exemplaire, alors que d’autres seront allouées à raison d’une copie par processeur.

2.3.2 Parallélisme par messages

Plus général et plus flexible, **MPI ou PVM** (‘Parallel Virtual Machine system’) s’accorde avec toutes les plate-formes, qu’elles partagent de la mémoire ou non, puisque ce partage de données est remplacé par des envois de messages. Cette démarche a beaucoup de liens avec le modèle de programmation de Foster: chaque tâche élémentaire devient un process, chaque process pouvant communiquer avec les autres.

A gros traits, l’utilisateur spécifie lors du lancement du programme le nombre de process qu’il souhaite. Chacun exécute le même programme mais peut réaliser des opérations différentes car il dispose d’un identificateur distinct. Ils réalisent concurremment des calculs sur leurs variables locales et s’échangent des résultats. Leur synchronisation est en fait souvent rythmée par cette arbre de communications.
OpenMP et MPI désignent avant toute chose des normes. Il existe plusieurs implémentations de ces API et certains constructeurs de machines proposent, comme pour les librairies mathématiques BLAS et LAPACK, leurs versions optimisées.

2.3.3 Parallélisme multiniveau

Beaucoup de plate-formes parallèles étant des grappes de multiprocesseurs, on peut souvent améliorer les performances d’un code utilisant seulement MPI, en mixant les deux approches: OpenMP au niveau de chaque nœud à mémoire partagée pour gérer les interactions à faible grains entre certains processeurs et MPI pour organiser les communications gros grains inter-nœuds. Cette stratégie est d’ailleurs tout à fait envisageable pour l’algorithme FETI implanté dans *Code_Aster* , puisqu’il est parallélisé *via* MPI et utilise pour ces inversions de matrices de rigidité locales, la multifrontale parallélisée en OpenMP. Le solveur MUMPS cumule aussi potentiellement un algorithme parallélisé *via* MPI et des routines mathématiques en OpenMP (Scalapack, Blas).

Figure 2.3-1._ Logos des API OpenMP et MPI.

2.4 Mesure de performances

2.4.1 Speed-up et scale-up

Une fois que l’on a validé et stabilisé un code parallèle[12] , il est toujours instructif d’en mesurer les performances et d’en faire un profiling. Deux cas de figures complémentaires peuvent alors se présenter. Résolvant un problème de taille N sur p processeurs en un temps donné, on souhaite:

•Diminuer ce temps elapsed en augmentant p . On espère une diminution quasi-linéaire et on parle alors de scalabilité ou d’extensibilité forte (‘strong scalability’). Cette notion est le pendant informatique de la scalabilité numérique des préconditionneurs DD que l’on abordera dans les chapitres suivants. Elle se mesure par un critère appelé **speed-up**

$$S_p(N)=\frac{temps\ séquentiel}{temps\ sur\ p\ processeurs}:=\frac{T_1(N)}{T_P(N)} \qquad (2.4\text{-}1)$$

A ce critère, on préfère parfois la notion **d’efficacité parallèle**

$$E_p(N):=\frac{S_p(N)}{p} \qquad (2.4\text{-}2)$$

•Augmenter la taille du problème à résoudre en augmentant p . On parle de scalabilité ou d’extensibilité faible (‘weak scalability’). Cette notion est le pendant de l’optimalité des préconditionneurs DD et son critère est le **scale-up**

12 Pas avant ! Il faut savoir résister à la tentation du speed-up !

$$C_p(N)=\frac{temps\ séquentiel}{temps\ sur\ p\ processeurs\ du\ pb\ de\ taille\ pN}:=\frac{T_1(N)}{T_P(pN)} \quad (2.4\text{-}3)$$

2.4.2 Loi d'Amdhal & Co

Idéalement ces critères doivent être linéaires. En pratique ce n'est malheureusement pas toujours le cas car une exécution parallèle est inévitablement handicapée par des portions séquentielles et des surcoûts de communications qui croissent non linéairement avec le nombre de processeurs. Différents critères permettent d'introspecter plus finement ces performances parallèles:

•La **loi d'Amdhal** qui évalue, à partir d'une exécution séquentielle et de parallèles, les performances optimales d'un code sur une plate-forme donnée. Si f est la portion de code irrémédiablement séquentielle, le speed-up théorique atteignable sur p processeurs en négligeant les communications et les calculs redondants, s'écrit

$$S_p^A(N):=\frac{1}{f+(1-f)/p} \quad (2.4\text{-}4)$$

Mais ce critère suppose que la taille est constante lorsqu'on augmente le nombre de processeurs, il ne tient donc pas compte de la scalabilité faible. En plus, il dépend fortement de la taille du problème initial, car la portion séquentielle décroît souvent avec la taille du problème. Bref, il peut être biaisé et pour cette raison on lui préfère le critère suivant.

•La **loi de Gustafson-Barsis** qui prédit un speed-up optimal équilibré ('scaled speed-up'), partant d'une exécution parallèle et sachant que bien souvent on ne peut réaliser l'exécution séquentielle par manque de mémoire. Connaissant empiriquement la portion séquentielle f, ce **speed-up équilibré** vaut, sur p processeurs,

$$S_p^{GB}(N):=p+(1-p)\ f \quad (2.4\text{-}5)$$

•La **métrique de Karp-Flatt** qui, partant des valeurs de speed-up d'un problème, permet d'évaluer si les faiblesses du speed-up sont dues aux parties séquentielles ou aux communications. Pour chaque valeur de p et de $S_p(N)$ on déduit la fraction séquentielle théorique

$$f_p^{KF}(N):=\frac{1/S_p(N)-1/p}{1-1/p} \quad (2.4\text{-}6)$$

Si cette fraction reste constante avec p, c'est la partie séquentielle qui est en cause. Sinon il faut plutôt incriminer les communications.

2.5 Parallélisme informatique

Pour tirer parti des nouvelles machines à architecture parallèle, une des premières approches a consisté à **paralléliser de manière automatique les codes** de calcul exécutés jusqu'à présent sur machine séquentielle. Les différentes techniques utilisées vont, de l'analyse automatique de dépendance de sections de code, au déroulage de boucle en vue de leur vectorisation et de leur parallélisation, à la génération automatique de code parallèle à partir d'un code séquentiel.

On peut ainsi réutiliser au maximum un code séquentiel existant pour en produire une version capable de s'exécuter sur machines modérément parallèles. Ce sont aujourd'hui les techniques les plus transparentes pour le développeur mais elles font montre de piètres performances dès que le nombre de processeurs dépasse quelques unités et ne sont pratiquement disponibles que sur des architectures à mémoire partagée. Elles font toujours l'objet de recherche et par cette aspect un peu expérimental, on pourrait les comparer aux outils de différentiation automatique de code. Comme ces derniers, ces techniques ne fonctionnent en aveugle que sur des codes présentant une architecture et une structuration des données compatibles.

D'un point de vue moins transparent au programmeur, il est aussi possible de «profiler» les zones de codes très consommatrices en CPU et potentiellement segmentables en tâches indépendantes, et de les paralléliser *via* des directives OpenMP (cf. §2.3).

Remarque:

- *Ces techniques ont été mis en place dans le version 4 du Code_Aster[Ber97][Gom96] sur CRAY. Les calculs élémentaires étaient parallélisés via des directives de compilation ce qui pouvait procurer des speed-ups intéressants sur ces parties de code. Cependant, ces directives ont du être essaimées dans bon nombre de sources et cela compliquait les chantiers de maintenance/développement. En outre, le développeur lambda, souvent mécanicien de*

formation, devait tenir compte des contraintes de programmation qu'elles impliquaient. Cette expérience n'a pas survécu au portage sur SGI et a du être résorbée.

Ces parallélisations presque exclusivement informatiques sont bien adaptées pour intégrer des lois de comportement ou pour post-traiter des résultats. C'est ce type d'approche qui a été retenue pour **alimenter le parallélisme numérique de MUMPS** (cf. §2.6). En activant le **parallélisme distribué** (mot-clé PARALLELISME de l'opérateur AFFE_MODELE [U2.08.06]) on débute ainsi bien en amont du solveur linéaire, la distribution de calculs et de données.

En début d'opérateur *Aster*, on distribue des paquets de mailles par processeur (suivant différentes stratégies, automatiques, semi-automatiques ou manuelles) et les flots de données/traitements de chaque processeur se limitent par la suite à ces mailles. Chaque processeur effectue les intégrations de loi de comportement, les calculs élémentaires et les assemblages matriciels/vectoriels dans ce périmètre et va remplir juste la zone correspondante dans les structures de données globales (matrices, vecteurs, champs initialisés à zéro). Le tout est transmis de manière distribuée à MUMPS qui doit alors communiquer pour reconstituer le problème global et conduire la résolution en parallèle (avec sa propre distribution).
L'approche est moins ambitieuse en terme de performances CPU/mémoire et d'appariement de calculs distribués que les techniques DD, mais elle est plus robuste et plus facile à implanter dans un code (on résout le même problème qu'en séquentiel, pas un problème d'interface). Elle mutualise d'ailleurs certains chemins logiciels avec la méthode FETI implantée dans *Code_Aster*: décomposition en sous-domaines et distribution des calculs élémentaires/assemblages. Ce parallélisme informatique (du ressort du code applicatif *Code_Aster*) est très complémentaire du parallélisme numérique du solveur linéaire (ici MUMPS).

2.6 Parallélisme numérique

Pour améliorer l'efficacité parallèle d'un code, il faut donc plus s'investir dans la résolution numérique du problème modélisé et, par exemple, **résoudre en parallèle les grands systèmes creux** du type (1.1-1) dont l'assemblage et la résolution consomme souvent la majeure partie du temps: c'est ce qu'on appelle le parallélisme numérique.
Si on fait le choix de réutiliser les sources d'un solveur linéaire séquentiel intégré au code, différentes techniques permettent de répartir la charge de travail entre les processeurs et d'améliorer la «scalabilité» du processus (renumérotation de type «static pivoting», super-nœuds, partionnement de graphe, produits scalaires à poids ou par masque….). Mais malgré tout, si il s'agit d'un **solveur direct** , sa scalabilité risque de marquer le pas à partir de quelques dizaines de processeurs du fait des nombreux échanges de blocs matriciels de la factorisation numérique. Si il s'agit d'un **solveur itératif** , la difficulté va provenir du préconditionneur qui a du mal à combiner scalabilité, robustesse et faible coût calcul (on parle de problème de 'parallel preconditionning'). D'où le recours à une approche hybride de type DD qui essaie de jouer sur ces deux classes de méthodes pour passer à l'échelle sur un plus grand nombre de processeurs.

L'exercice présente souvent une efficacité parallèle bridée à quelques dizaines de CPU et une gestion mémoire peu optimale. Mais c'est déjà très intéressant pour bon nombre d'études et de configurations machines. D'autre part, ce parallélisme affiche souvent une robustesse et un périmètre d'utilisation similaires à ceux du séquentiel, il est peu invasif et facile d'emploi.
Ce type de parallélisme a donc été retenu dans *Code_Aster* pour répondre à certains besoins: gagner un ordre de grandeur en temps CPU sur de grosses études qui pourraient à la limite passer en séquentiel avec les solveurs natifs. Parmi les choix proposés on retrouve la méthode directe par défaut du code, la **multifrontale,** le solveur direct externe **MUMPS** et la librairie de solveurs itératifs/directs **PETSc**.

Remarque:

- *Ce parallélisme numérique peut s'avérer très précieux et complémentaires du parallélisme mécanique des méthodes DD. C'est tout l'attrait du parallélisme multiniveau de ces méthodes qui peuvent finalement être appréciées comme une surcouche des solveurs standards. On peut donc envisager à terme de coupler un algorithme FETI parallèle avec une multifrontale, un MUMPS ou un PETSc (ou les trois à la fois sur des sous-domaines différents), eux-mêmes parallèles.*

2.6.1 La multifrontale

Revenons un peu sur la **multifrontale du code** (METHODE='MULT_FRONT' dans le mot-clé facteur SOLVEUR [U4.50.01/R6.02.02]) parallélisée en OpenMP. On peut isoler deux niveaux de parallélisme:

•**un externe**, avec une élimination concurrente des super-noeuds en bas de l'arborescence d'élimination des inconnues.
•**un interne,** avec la gestion concurrente de certaines phases (assemblage des matrices filles, mise à jour de la matrice frontale…) de l'élimination des autres super-nœuds, ceux du haut de l'arbre.

Le découpage entre les deux niveaux est paramétré et cela permet d'opérer un compromis entre la granularité des calculs et l'équilibrage de charge des processeurs. Pour l'instant, seul le parallélisme interne a été activé et il permet de bénéficier de speed-ups appréciables (<5) sur la phase de factorisation numérique[13] lorsqu'on le déploie sur une machine multiprocesseur.

Remarque:

- *Le gradient conjugué «natif» de Code_Aster, un GCPC standard préconditionné par un Cholesky incomplet multiniveau ICC(k) (METHODE='GCPC' [R6.05.02]), n'a pu bénéficier des mêmes avancées pour des problèmes de 'parallel preconditionning'. Dans d'autres codes d'EDF R&D, tels Code_Saturne et TELEMAC, une telle méthode rend pourtant de très grands services et passe souvent à l'échelle sur des centaines de processeurs. Le meilleur conditionnement de leurs problèmes et leur gros volume de données leur permettent et leur imposent de travailler avec des préconditionneurs plus frustres (par exemple blocs diagonaux) et donc plus facilement parallélisables. Mais se mettre en condition de bien passer à l'échelle se paye parfois au prix fort en terme d'itérations: des centaines voire des milliers par résolution de système linéaire. D'autre part, ces codes peuvent se passer de certains services inhérents aux solveurs directs: le traitement de problème de type multiple seconds membres (calcul non linéaire avec mutualisation de la matrice tangente, calcul modal) et la détection de singularité (validation de la mise en données, calcul modal/flambement). Un code de mécanique des structures généraliste a du mal à s'en passer !*

2.6.2 MUMPS, PETSc et les solveurs parallèles externes

On peut aussi s'appuyer sur des **librairies scientifiques externes** qui ont optimisé «l' implémentation» de solveurs linéaire, en séquentiel et en parallèle. Cela permet de conjuguer à moindre frais portabilité et performance tout en s'appuyant sur l'expérience d'équipes internationales reconnues. Souvent, ces librairies proposent bien plus que des solveurs, en permettant de constituer toute une séquence de calcul parallèle avec ses propres structures de données, sans véritablement se préoccuper de la répartition de ces flots d'instructions et de données. Leurs larges périmètres d'utilisation et certains paramètres d'ajustement automatiques prennent en compte bien des facteurs lourds à gérer dans un solveur natif: le type de données (réel, complexe, symétrique…), leur représentation (creux, bande…), le choix d'algorithmes adaptés (scaling, renuméroteur…), les opérations algébriques basiques suivant les données (optimisation type BLAS), l'architecture de la machine cible, les librairies disponibles (BLACS, ScaLAPACK, ParMETIS…).

Pour une utilisation multi plate-formes, on préfère souvent s'appuyer sur des librairies du domaine public (PETSc, HYPRE…) mais bien sûr chaque constructeurs propose sa version (en principe) optimisée: MKL pour Bull/Intel, PESSL pour IBM, ASL et MathKeisan chez NEC etc.
On citera ainsi, de manière non exhaustive:

- **Toolkit PETSc**: 'Portable Extensive Toolkit for Scientific computation' du laboratoire Argonne avec ses utilitaires de manipulation de structures de données et ses solveurs parallèles variés (linéaire, modaux et EDO). C'est un produit du domaine public, interfacé avec Fortran 77/90, C et C++, parallélisé *via* MPI et qui a atteint une certaine maturité (le projet a débuté en 1991). Certains grands codes, tel MEF++ [Tar04], n'ont pas hésité à s'y adosser complètement pour distribuer leurs données et accéder au parallélisme numérique. *Code_Aster* propose différentes approches pour tirer partie de cette librairie (METHODE='PETSC' dans le mot-clé facteur SOLVEUR [U4.50.01/R6.01.02][Des07]).

  Dans la même veine, d'autres toolkits d'algèbre linéaire parallèles ont vu le jour: TRILINOS (Sandia), HYPRE (Livermore), PSARSLIB (Université du Minnesota), ScaLAPACK (Université d'Oak Ridge)…

- **Solveur direct MUMPS:** 'Multifrontal Massively Parallel sparse direct Solver', solveur direct multifrontal pour la résolution de grands systèmes creux en parallèle. Ces facultés de pivotage et ces outils de pré et post-traitement (mise à l'échelle, raffinement itérative, erreur *a posteriori*) ont déjà séduit *Code_Aster* , qui le propose pour beaucoup de modélisations (METHODE='MUMPS' [U4.50.01/R6.02.03]) et le conseille fortement pour certaines (XFEM, incompressible). Il est aussi

13 La factorisation symbolique et la descente-remontée sont pour l'instant traitées en séquentiel.

possible **d'exploiter son parallélisme, en mode distribué**[Boi07b] ou **centralisé**[Des07] suivant que l'on parallélise la construction du système (parallélisme informatique cf. §2.5) ou juste sa résolution (parallélisme numérique cf. figure 2.6-1). Cette approche procure en moyenne des gains en temps d'un facteur 6 en centralisé (12 en distribué) sur 32 processeurs. Dans des configurations favorables, ces gains peuvent dépasser la vingtaine.
Par contre, MUMPS standard (c'est-à-dire In-Core[14]) requiert beaucoup plus de mémoire vive que les méthodes natives du code, car il travaille exclusivement en RAM. Il est conseillé d'activer le **mode Out-Of-Core** (mot-clé OUT_OF_CORE='OUI' [Boi08]) pour décharger entièrement sur disque la factorisée et rester ainsi dans des consommations mémoires plus raisonnables.

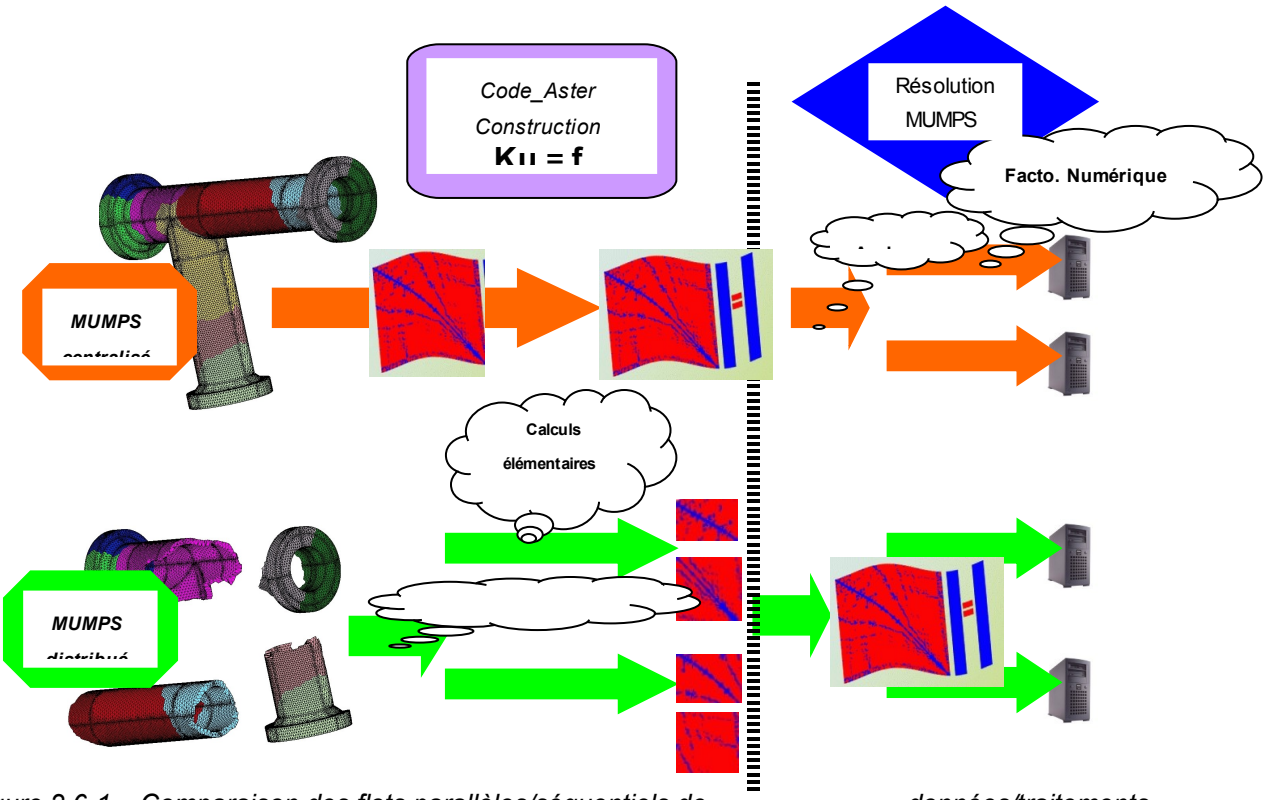


Figure 2.6-1._ Comparaison des flots parallèles/séquentiels de données/traitements des approches «MUMPS centralisée» et «distribuée».

Dans la même veine, on trouve pour les directs: WSMP (IBM), SPOOLES, SuperLU, PaStiX etc. et pour **les itératifs**: BlockSolve95, SPAI, pARMS…

Pour plus d'informations sur cette problématique on pourra consulter l'article de synthèse de I.S.Duff et H.A. Van der Vorst[DV99] retraçant les évolutions des solveurs linéaires parallèles et la liste exhaustive des packages disponibles mise à jour sur le site web d'un des pères fondateurs du domaine, Jack Dongarra[Don].

Remarque:

- *Des plate-formes de développement, telles Numerical Platon ou Arcane, permettent de masquer la diversité et la complexité de ces librairies de solveurs. Ces sur-couches logicielles ne formalisent que des objets de haut niveaux (variable, maillage…) et fournissent des outils d'analyse, de mise au point et d'optimisation des performances du calcul indépendants des aspects numérique et architecture.*

14 Une gestion mémoire «In-Core(IC)» conserve tous ses objets en RAM. C'est la cas standard retenu par beaucoup d'applications. A l'inverse, une gestion «Out-Of-Core(OOC)» (par exemple, celle du progiciel JEVEUX de *Code_Aster*) décharge ses objets les plus encombrants sur un disque annexe. Cela obère un peu les performances CPU lors des accès aux données mais décuple les capacités de stockage.

3 De Schwarz à Schur

3.1 Méthodes de Schwarz

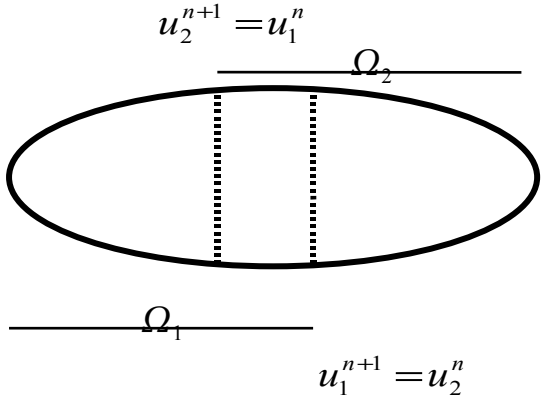
3.1.1 Description générale

La première méthode de DD est généralement attribuée au mathématicien Suisse du 19ième siècle, H.A.Schwarz[15]. Pour résoudre analytiquement une EDP sur une géométrie complexe, il a proposé de scinder ce domaine en deux sous-domaines plus simples et recouvrants, donnant ainsi son nom à une grande classe de méthodes DD: les méthodes de Schwarz alternées ('alternating Schwarz method') appelées aussi méthodes de Schwarz avec recouvrement ('overlapping Schwarz method').
Recouvrement et processus alternés, les deux idées sont effectivement là puisqu'il s'agit de résoudre alternativement sur chacun des sous-domaines, une EDP identique à celle du problème initial, en rajoutant juste sur l'interface une nouvelle condition de Dirichlet. Suivant le type de mise à jour que l'on va imposer sur cette interface, la dernière ou l'avant-dernière solution du sous-domaine voisin, on distingue la Méthode de Schwarz Multiplicative (MSM) ou Additive (MSA).

Par exemple, considérons un domaine connexe $\Omega=\Omega_1\cup\Omega_2$, décomposable en deux sous-domaines recouvrants, sur lequel on veut résoudre l'EDP suivante:

$$\begin{cases} Lu=f & \text{dans } \Omega \\ u=g & \text{sur } \partial\Omega \end{cases} \tag{3.1-1}$$

MSA resoud itérativement ce problème modèle en construisant une double suite de solutions approchées $\left(u_1^n, u_2^n\right)_n$ suivant le schéma suivant (pour MSM on change la sixième condition en $u_2^{n+1}=u_1^{n+1}$)



$$\begin{cases} Lu_1^{n+1}=f & \text{dans } \Omega_1 \\ u_1^{n+1}=g & \text{sur } \partial\Omega_1\cap\partial\Omega \\ u_1^{n+1}=u_2^n & \text{sur } \partial\Omega_1\cap\Omega_2 \end{cases}$$

puis (3.1-2)

$$\begin{cases} Lu_2^{n+1}=f & \text{dans } \Omega_2 \\ u_2^{n+1}=g & \text{sur } \partial\Omega_2\cap\partial\Omega \\ u_2^{n+1}=u_1^n & \text{sur } \partial\Omega_2\cap\Omega_1 \end{cases}$$

Figure 3.1-1._ Méthode de Schwarz additif (MSA).

3.1.2 Algorithme

Après discrétisation, en notant $\boldsymbol{A}$ l'opérateur de travail, $\boldsymbol{A_i}$ sa restriction au sous-domaine Ω_i , $\boldsymbol{F}$ le vecteur second membre et $\boldsymbol{R_i}$ l'opérateur de restriction de Ω à Ω_i , ces méthodes peuvent se reformuler sous la forme d'un problème de point fixe : on cherche la suite d'itérés $\left(u^n\right)_n$ vérifiant (avec $\boldsymbol{u}$ le vecteur solution)

$$\begin{aligned} u^{n+1}-u&=\left(I-R_1^T A_1^{-1} R_1 A-R_2^T A_2^{-1} R_2 A\right)\left(u^n-u\right) \quad (\text{MSA}) \\ u^{n+1}-u&=\left(I-R_2^T A_2^{-1} R_2 A\right)\left(I-R_1^T A_1^{-1} R_1 A\right)\left(u^n-u\right) \quad (\text{MSM}) \end{aligned} \tag{3.1-3}$$

D'où le vocable «Schwarz additif» et «Schwarz multiplicatif» de ces méthodes.

Remarques:

- *Pour ne pas alourdir le propos, nous nous sommes limités à une condition limite de type Dirichlet simple, mais évidemment ces considérations se généralisent aux Dirichlet généralisés, à Neumann ou à Robin.*

15 H.A.SCHWARZ. *Uber einige Abbildungensaufgaben (traduction littérale: «Sur quelques problèmes de projection»)*. J. Reine Angew. Math., 70 (1869), pp105-120.

- *Avec MSM, du fait de l'entrelacement des solutions de domaines contigus entre deux itérations successives, un parallélisme à forte granularité dédié à chaque sous-domaine est difficile à mettre en place. Il faut alors user de «techniques de coloriage» pour distinguer des familles de sous-domaines complètement séparées. Cela dégrade notablement le parallélisme et la lisibilité du processus.*

Les méthodes de point fixe convergent au moins linéairement, mais on cherche néanmoins à les accélérer en utilisant, soit l'algorithme du gradient conjugué pour MSA, soit GMRES pour MSM. Par exemple pour MSA avec P sous-domaines, la formule (3.1-3a) peut se réécrire sous la forme

$$u^{n+1}=u^n-\left(R_1^T A_1^{-1} R_1+\cdots+R_P^T A_P^{-1} R_P\right)\left(\mathbf{Au^n}-f\right) \quad (3.1\text{-}4)$$

Ce qui n'est rien d'autre qu'une méthode de Richardson (appelée aussi méthode du gradient à pas constant) associée au problème discrétisé initial $\mathbf{Au}=f$ et préconditionnée par

$$M_{MSA}^{-1}:=\sum_{i=1}^{P} R_i^T A_i^{-1} R_i \quad (3.1\text{-}5)$$

avec un pas unité $\rho^n=1$. Par ce jeu d'écriture, on a donc substitué à un processus alterné une méthode de descente de type gradient exprimée sur le domaine tout entier et dont seul le préconditionnement laisse entrevoir la DD.
Lorsque le problème est symétrique elliptique (ce qui est fréquent en mécanique des structures), les matrices A et M^{-1} sont SPD et un GCPC standard peut alors avantageusement se substituer à la méthode de Richardson:

Initialisation u^0 donné, $r^0=\mathbf{Au^0}-f$, $p^0=r^0$, $g^0=M_{MSA}^{-1}r^0$
Boucle en n
(1) $z^n=\mathbf{Ap^n}$
(2) $\alpha^n=\dfrac{\langle r^n, g^n\rangle}{\langle z^n, p^n\rangle}$
(3) $u^{n+1}=u^n+\alpha^n p^n$ (nouvel itéré)
(4) $r^{n+1}=r^n-\alpha^n z^n$ (nouveau résidu)
(5) Test d'arrêt via $\langle r^{n+1}, r^{n+1}\rangle$
(6) $g^{n+1}=M_{MSA}^{-1}r^{n+1}$ (résidu préconditionné)
(7) $\beta^{n+1}=\dfrac{\langle r^{n+1}, g^{n+1}\rangle}{\langle r^n, g^n\rangle}$
(8) $p^{n+1}=g^{n+1}+\beta^{n+1} p^n$ (nouvelle direction de descente)

Algorithme 3.1-1._ Algorithme de MSA.

Cette mise sous forme de GCPC, tout comme les notions de solveurs grossiers ou inexacts, sont appelées techniques d'accélération, car elles permettent d'améliorer le convergence du processus global en mélangeant les itérés successifs de la solution. Pour l'autre grande variante de Schwarz, MSM, l'algorithme est similaire avec un GMRES standard puisque l'opérateur n'est pas symétrique.

Remarques:

- *Malheureusement les performances de cet algorithme s'étiolent lorsque le nombre de sous-domaines augmente. Pour pallier cette difficulté on introduit un problème grossier ('coarse grid problem' par référence au multigrille) dont la contribution va s'ajouter à celle du préconditionneur. Comme son nom l'indique, ce problème consiste à résoudre le même opérateur mais discrétisé sur un espace fonctionnel plus petit. Il n'est pas toujours facile à construire mais on sait caractériser certaines de ses propriétés théoriques. Son objectif étant de rendre la convergence de l'algorithme peu dépendante du nombre de sous-domaines. Cette notion de préconditionnement par un problème grossier est récurrent en DD. On le retrouvera pour les méthodes sans recouvrement et en particulier pour FETI-1, dont c'est un ingrédient crucial.*
- *La part la plus coûteuse de l'algorithme étant la construction du préconditionneur qui nécessite P résolutions de problèmes locaux du genre*

$$\langle A_i u_i , v_i \rangle = \langle R_i \; r^{n+1} , v_i \rangle \tag{3.1-6}$$

une pratique courante en DD consiste à soulager les processeurs en remplaçant les opérateurs locaux A_i par des opérateurs approchés B_i plus simples à calculer (mais «spectralement équivalentes»). Comme il ne s'agit que de la phase de préconditionnement, l'introduction de ces «résolutions inexactes» ('inexact solves') n'est pas véritablement préjudiciable.

3.1.3 Convergence

En empruntant le canevas théorique des méthodes de correction par sous-espace ('subspace correction methods', P.L.Lions 1987,), on montre que sous certaines hypothèses (théorème 2.9 [LeT94]) que le conditionnement de MSA (qui pilote la convergence de l'algorithme 3.1-1) vérifie asymptotiquement

$$\begin{aligned} \eta\left(M_{MSAEG}^{-1} A \right) &= \theta\left(\frac{H}{\delta} k \right) \quad \text{avec solveur grossier} \\ \eta\left(M_{MSA}^{-1} A \right) &= \theta\left(\frac{1}{\delta \; H} k \right) \quad \text{sans} \end{aligned} \tag{3.1-7}$$

en notant H le diamètre moyen des sous-domaine, δ leur recouvrement moyen et k leur nombre moyen de voisins. Sans solveur grossier, la convergence du processus est altérée par l'augmentation du nombre de sous-domaines tandis qu'avec ce type de technique d'accélération, elle ne dépend plus que de la proportion de recouvrement et du nombre de voisins. La construction d'un solveur grossier, malgré le surcoût qu'il engendre, semble donc être le prix à payer pour traiter des dizaines de sous-domaines. D'autre part, la convergence s'améliore avec la proportion de recouvrement ou est inversement proportionnelle au nombre de voisins.

Remarque:

•Le fait que ces méthodes convergent d'autant mieux que les sous-domaines ont peu de voisins se retrouve dans d'autres méthodes DD. Cela privilégie les sous-domaines élancés. Un compromis est donc à rechercher entre les contraintes informatiques du parallélisme qui souhaitent minimiser le volume des communications aux interfaces et celles, numériques, qui tendent au contraire à les favoriser pour améliorer la convergence.

Ces méthodes de Schwarz sont souvent efficaces et parallélisables. Elles sont plus simples à incorporer dans un code que les autres méthodes DD car elles réutilisent l'opérateur initial sans avoir à lui substituer un nouvel opérateur d'interface et un préconditionneur adapté. Cependant subsistent quelques difficultés telles le découpage avec recouvrement de structures quelconques ou la construction d'espace grossier adapté. D'autre part, leur convergence est très sensible aux hétérogénéités et elles dupliquent une part non négligeable du travail dans les zones de recouvrement (plus celles-ci sont importantes, meilleures est la convergence, d'où un compromis à trouver !). Elles semblent ainsi peu utilisées en mécanique des structures, c'est un domaine où on leur préfère les méthodes sans recouvrement de type Schur primale ou duale. Elles font l'objet des paragraphes suivant.

3.2 Méthodes de Schur primale

3.2.1 Description générale

Les techniques de DD sans recouvrement ('nonoverlapping decomposition method') simplifient la prise en compte des zones frontières. En éliminant tous les degrés de liberté qui n'appartiennent à aucune interface de sous-domaines, elles réduisent le problème initial global en un problème, plus petit et mieux conditionné, dont les inconnues sont précisément les degrés de liberté localisés sur cette interface.

A l'origine, les premières méthodes DD développées dans les années 60 par les mécaniciens des structures, résolvaient ce problème d'interface avec un solveur direct: c'est la sous-structuration statique appelée aussi méthode des sous-structures, des macro ou super-élements[16], condensation statique[17] ou méthode directe de Schur primale ('primal substructuring method'). Le terme primal est requis pour deux raisons:

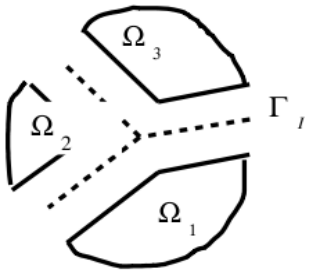
- Le problème d'interface est formulé dans les mêmes variables que le problème initial, c'est-à-dire en déplacement,

16 Terminologie que l'on retrouve pour le solveur linéaire frontal. On explicitera cet connivence par la suite.

17 En fait, la notion de condensation statique recoupe un plus large périmètre. C'est une technique qui permet l'élimination des degrés de liberté internes à un élément ou à un groupe d'éléments afin d'en simplifier la formulation. Mais elle peut bien sûr s'appliquer à des pans entiers d'une structure, d'où l'amalgame des terminologies.

- On veut distinguer cette méthode des méthodes de type FETI (cf. §4) qui écrivent leur problème d'interface en terme d'inter-efforts ou efforts de réactions entre les sous-structures;

Depuis une quinzaine d'années, la taille des problèmes traités et donc celle des problèmes d'interfaces associés, tend à substituer aux solveurs directs d'interface (le plus souvent une multifrontale) des méthodes de type gradient conjugué préconditionné. La pierre angulaire de l'édifice étant alors la construction d'un préconditionneur efficace, fiable, parallélisable et facilement implantable pour résoudre des problèmes non-linéaires sur des géométries complexes et pour des découpages quelconques. En effet la qualité du partitionnement en sous-domaines peut perturber le parallélisme intrinsèque du GC (cf. §2.6). Sa robustesse est dépendante du type de découpage (régulier ou élancé), de la régularité du maillage et des partitions (les fameux «ratios d'aspect»), de la présence de sous-structures «flottantes» (sous-domaines non bloqués) ou de points de jonction entre sous-structures[18] (arêtes en 3D).
Même si l'idée de décomposer une structure afin d'en simplifier le calcul était dans l'air depuis le début de la simulation, sa paternité est souvent attribuée à J.S.Przemieniecki[19] (1963). Il s'agissait surtout à l'époque de s'affranchir des faibles capacités mémoire des calculateurs en utilisant au maximum les invariances de symétrie ou de translation de structures et de permettre un travail collaboratif sur des sous-structures distinctes.
Une problématique, qui paraît bien lointaine maintenant, concernait la nécessité de pouvoir mixer les deux types d'approches qui s'affrontaient alors dans la communauté 'computational structural analysis': la méthode des forces et celles des déplacements[20]. Par contre, une caractéristique majeure de cette méthode des sous-structures que son auteur ne pouvait suspecter, est son parallélisme intrinsèque, véritable «pain béni» pour les machines actuelles.
Illustrons notre propos avec le partitionnement du domaine 2D quelconque de la figure 3.2-1 en différents sous-domaines géométriquement indépendants, Ω_1 , Ω_2 et Ω_3 . On note Γ_I l'interface géométrique qui les relie entre eux. L'approche algébrique consiste alors en une renumérotation formelle des nœuds en vue de présenter un découpage par blocs du problème initial $\boldsymbol{Ku}=\boldsymbol{f}$: on numérote d'abord les degrés de libertés rattachés strictement à l'intérieur du premier sous-domaine, puis ceux du second et ainsi de suite… jusqu'à traiter en dernier ceux de l'interface. Le système renuméroté se présente alors sous la forme



$$\begin{bmatrix} K_{11} & 0 & 0 & K_{1\Gamma} \\ 0 & K_{22} & 0 & K_{2\Gamma} \\ 0 & 0 & K_{33} & K_{3\Gamma} \\ K_{\Gamma 1} & K_{\Gamma 2} & K_{\Gamma 3} & K_{\Gamma\Gamma} \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_\Gamma \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ f_\Gamma \end{Bmatrix} \tag{3.2-1}$$

Figure 3.2-1._ Exemple de décomposition en sous-domaines et interface géométrique (dite primale)

En substituant les trois premières inconnues dans la quatrième, ce que le mécanicien dénomme par «condensation statique du problème discrétisé sur l'interface», on se ramène au problème d'interface

$$\mathbf{S}\mathbf{u}_\Gamma = b \tag{3.2-2}$$

avec

$$\begin{aligned} S &:= \sum_i S^i := K_{\Gamma\Gamma} - \sum_i K_{\Gamma i} K_{ii^{-1}} K_{i\Gamma} \\ b &:= \sum_i b^i := f_\Gamma - \sum_i K_{\Gamma i} K_{ii^{-1}} f_i \end{aligned} \tag{3.2-3}$$

$\boldsymbol{S}$ est la matrice du complément de Schur appelée aussi parfois matrice de capacitance ou de rigidité condensée. Elle est en fait la discrétisation de l'opérateur $\boldsymbol{SP}$ dit de Poincaré-Steklov[21] (ou 'Dirichlet-to-Neumann') exhumé dans ce cadre par V.Aghoskov et V.Lebedev au début des années 80. En mécanique des

18 Appelés aussi points de croisement ('cross points') ou points multiples.
19 J.S.Przemieniecki. *Matrix structural analysis of substructures*. Am. Inst. Aero. Astro. J., **1** (1963), pp138-147.
20 La dernière approche, la 'displacement method', du fait de sa facilité de mise en œuvre, de son efficacité numérique et de son universalité a occupé quasi-exclusivement le devant de la scène. Tandis que l'autre approche, la 'force method', n'a que récemment gagnée quelques points… sur le terrain des indicateurs d'erreurs spatiaux !

structures, cet opérateur est souvent continu, symétrique, défini positif et rend le problème bien posé. Le vecteur $\boldsymbol{b}$, quant à lui, est le résultat de la condensation des efforts généralisés $\boldsymbol{f}$ sur Γ_I .
Par cette astuce de redistribution/substitution des inconnues, on a transformé un problème global en un problème d'interface plus petit, SPD et mieux conditionné (par contre plus dense que le problème creux initial) donc plus facile à résoudre avec un GCPC.
Une fois exhumée la solution d'interface $\boldsymbol{u_\Gamma}$, on remonte aux contributions internes *via* des relations du type

$$u_i = K_{ii}^{-1}\left(f_i - K_{i\Gamma} u_\Gamma\right) \tag{3.2-4}$$

3.2.2 Convergence

Cet atout des méthodes de Schur en terme de conditionnement s'illustre par les estimations suivantes. On passe, sous certaines hypothèses (§3.2.5 [LeT94]) et en reprenant les notations usuelles du §3.1, asymptotiquement de

$$\eta(K) = \theta\left(\frac{1}{h^2}\right) \tag{3.2-5}$$

à

$$\eta(S) = \theta\left(\frac{1}{H^2}\left(1+\frac{H}{h}\right)\right) \tag{3.2-6}$$

D'où une meilleure précision si on le résout *via* un solveur direct (Schur direct primal) ou une meilleure convergence si on fait appel à de l'itératif (Schur itératif primal). Dans les deux cas, les algorithmes qui encapsulent souvent cette inversion de système sont gagnants: cela facilite les calculs modaux ou les résolutions non linéaires de Newton.
Cependant, ce conditionnement dépend fortement de la finesse du maillage ou du nombre de sous-domaines. Dans l'état la méthode de Schur n'est donc ni optimale, ni numériquement extensible (cf. notions de speed-up et de scale-up du §2.4). On verra dans la version itérative que certains préconditionneurs permettent d'arranger les choses. Ou alors, il faut changer la formulation du problème et c'est FETI and Co(§4) !

3.2.3 Méthode directe de Schur primale

La résolution du problème condensé (3.2-2) peut s'effectuer directement par un solveur direct standard de type $\boldsymbol{LDL^T}$, mais il est plus judicieux de jouer de l'analogie entre les sous-domaines de la sous-structuration et les super-éléments de la multifrontale. On peut ainsi opérer la résolution, dès l'assemblage, des composantes locales S^i de la matrice de Schur de Ω_i

$$S^i := K^i_{\Gamma\Gamma} - K_{\Gamma i} K_{ii}^{-1} K_{i\Gamma} \tag{3.2-5}$$

en appliquant la multifrontale aux deux super-éléments regroupant les degrés de liberté internes (notés i) et frontières (notés Γ). Il «suffit» d'inverser la matrice de rigidité associé à ce $i^{ème}$ sous-domaine en conservant comme dernier front, l'ensemble des nœuds maître de Γ_I

$$K^i := \begin{bmatrix} K_{ii} & K_{i\Gamma} \\ K_{\Gamma i} & K^i_{\Gamma\Gamma} \end{bmatrix} \tag{3.2-6}$$

On peut ainsi voir la méthode frontale comme une forme récursive de condensation des nœuds internes sur les nœuds maîtres. Cette opération peut bien sûr être réalisée simultanément sur plusieurs sous-domaines. Tout comme les condensations locales des second membres

$$b^i := f^i_\Gamma - K_{\Gamma i} K_{ii}^{-1}\left(f_i\right) \tag{3.2-7}$$

Si l'interface est de petite taille, les phases d'assemblage des compléments de Schur et des efforts condensés locaux ainsi que la résolution du système condensé (3.2-2) peuvent rester séquentielles. Néanmoins, si celle-ci augmente, on peut être tenté de mettre en place un parallélisme numérique à ce niveau et on se retrouve alors dans la configuration évoquée précédemment d'un parallélisme «emboîté». D'où un canevas possible

Par sous-domaine (en parallèle niveau 1)	**Par le proc. maître** (en séquentiel ou parallèle niveau 2)

21 V.Aghoskov & V.Lebedev. *The Poincaré-Steklov's operators and the domain decomposition methods in vartiational problems*. Computational processes and systems, Ed. Nauka (1985), pp173-227.

•Constitution des rigidités élémentaires K^i
•Condensation locale de S^i et b^i
•Envoi par messages ou partage par directives de ces condensés
• Assemblage de S et b
• Résolution du problème d'interface $\mathbf{S}u_\Gamma = b$
• Découpage et envoi des u_Γ^i
•Réception des u_Γ^i
•Restitution aux nœuds internes *via* $u_i = K_{ii}^{-1}\left(f_i - K_{i\Gamma}u_\Gamma\right)$

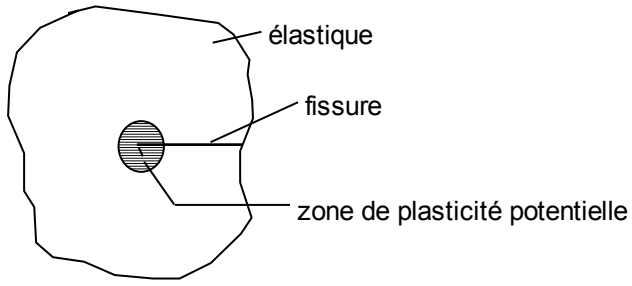
Algorithme 3.2-1_ Méthode directe de Schur primale.

On constate néanmoins que ce type d'approche nécessite le stockage des matrices de rigidité locales $\boldsymbol{K}^i$, de la matrice de Schur S (pleine) et des inverses $\mathbf{K}_{ii}^{-1}$. Ces dernières, notamment, sont stockées sous forme $\boldsymbol{LDL}^T$ pendant tout le processus pour ne pas avoir à refaire plusieurs fois cette étape de factorisation (qui reste coûteuse malgré le fait que ces sous-matrices soient de taille et de largeur de bande plus petite que la matrice initiale). C'est ce type de contrainte qui a longtemps pénalisé la sous-structuration statique en la rendant parfois moins attrayante qu'une résolution globale.
Pour pallier cet inconvénient, on peut envisager des condensations successives sur des problèmes de taille de plus en plus réduites (DD multiniveau) ou tirer profit, lorsque cela est possible, de répétitivité ou de symétrie. Au delà des aspects performances, cette souplesse en terme d'appariement de calculs, potentiellement distribués, est un point fort de la sous-structuration statique.

Remarques:
•Depuis les origines de la sous-structuration, des parallèles ont été faits entre cette méthode mécanique et le solveur multifrontal. Le premier considère les sous-structures géométriques quand le second traite des groupes de nœuds, les «super-nœuds». Mais un des avantages de la sous-structuration est justement cette connaissance de la géométrie sous-jacente qui évite de calculer des quantités redondantes[22] ou propose des astuces matricielles pour les recouvrer.
•Des considérations sur les numérotations des différentes matrices évoquées précédemment permettent aussi de minimiser leur encombrement. D'autre part, contrairement aux solveurs itératifs, les solveurs directs telle la multifrontal bénéficient d'heuristiques estimant leur complexité calcul. Ces estimations déduites à la volée peuvent nourrir des stratégies de partitionnements dynamiques de domaine (on transfert des groupes de degrés de liberté d'un sous-domaine à l'autre en cours de calcul) afin d'équilibrer la charge des processeurs. Car celle ci n'est pas rigoureusement proportionnelle au nombre de mailles d'un sous-domaine, même en linéaire. Ces différentes stratégies ont nourries de nombreux travaux de la communauté mécanicienne de l'UTC autour de leur code SIC pour Système Interactif de Conception (cf. notamment papiers de G.Touzot, P.Breitkopf & Y.Escaig).

3.2.4 Implantation dans Code_Aster: MACR_ELEM_STAT

En séquentiel, *Code_Aster* propose cette méthode de Schur directe primale *via* la commande MACR_ELEM_STAT[U2.07.02] de constitution de macro-éléments (sous-structures). Elle propose notamment la gestion de sous-structures multiniveau et/ou déduites d'autres sous-domaines par une transformation géométrique simple (rotation, translation). Son périmètre recouvre tous les éléments finis de mécanique avec les chargements associés, mais seulement en linéaire ou pour des parties linéaires de modèles non linéaires.



22 Les matrices frontales sont considérées comme des matrices pleines contrairement aux compléments de Schur qui peuvent comporter des zéros dont on connaît l'emplacement et que l'on évite de manipuler.

Figure 3.2-2._ Structure entièrement élastique sauf une petite zone plastique[U2.07.02].

On voit tout l'intérêt de cette approche lorsque par exemple une structure élastique comporte une petite zone plastique à la frontière de laquelle on va condenser la zone linéaire (cf. figure 3.2-2). Le nombre de degrés de liberté s'en trouve considérablement réduit.
En dynamique, le code propose des méthodes DD adaptées et approchées telles que celles de Craig-Bampton ou Mac Neal[R4.06.02/R4.06.3].

Comme c'est souvent le cas en DD, cette méthode de Schur directe primale peut procurer, même en séquentiel, des gains en complexité calcul et en encombrement mémoire appréciables. Mais en plus, elle

Initialisation u_Γ^0 donné, $r_\Gamma^0 = \mathbf{S}\mathbf{u}_{\boldsymbol{\Gamma}}^{\mathbf{0}} - b$, $p_\Gamma^0 = r_\Gamma^0$, $g_\Gamma^0 = M^{-1} r_\Gamma^0$

Boucle en n

(1) $z_\Gamma^n = \mathbf{S}\mathbf{p}_{\boldsymbol{\Gamma}}^{\boldsymbol{n}}$ (assemblage de P problèmes de Dirichlet)

(2) $\alpha^n = \dfrac{\langle r_\Gamma^n, g_\Gamma^n \rangle}{\langle z_\Gamma^n, p_\Gamma^n \rangle}$

(3) $u_\Gamma^{n+1} = u_\Gamma^n + \alpha^n p_\Gamma^n$ (nouvel itéré)

(4) $r_\Gamma^{n+1} = r_\Gamma^n - \alpha^n z_\Gamma^n$ (nouveau résidu)

(5) Test d'arrêt via $\langle r_\Gamma^{n+1}, r_\Gamma^{n+1} \rangle$

(6) $g_\Gamma^{n+1} = M^{-1} r_\Gamma^{n+1}$ (résidu préconditionné)

(7) $\beta^{n+1} = \dfrac{\langle r_\Gamma^{n+1}, g_\Gamma^{n+1} \rangle}{\langle r_\Gamma^n, g_\Gamma^n \rangle}$

(8) $p_\Gamma^{n+1} = g_\Gamma^{n+1} + \beta^{n+1} p_\Gamma^n$ (nouvelle direction de descente)

présente des avantages en terme de souplesse d'utilisation, de modélisation et de manipulations numériques hérités de la sous-structuration statique et des solveurs directs. Cependant cette adhérence au solveur direct limite leur scalabilité (cf. § 2.6) et requiert des capacités de stockage qui croissent quadratiquement avec la taille de l'interface. D'où le recours à de l'itératif pour résoudre le problème d'interface.

3.2.5 Méthode itérative de Schur primale

Lorsque le problème initial est SPD, cette version itérative de Schur primale est basée sur un GCPC standard ('substructure based GC algorithm') en notant bold M^{-1} une matrice de préconditionnement *ad hoc*:

Algorithme 3.2-2._ Méthode itérative de Schur primale.

L'étape (1) peut se décrire comme la résolution de P problèmes de Dirichlet indépendants (un par sous-domaine) car le produit matrice-vecteur $\mathbf{S}\mathbf{p}_\Gamma^n$ (à l'itération n du GCPC) se décompose sur chaque sous-domaine i

$$\mathbf{S}\mathbf{p}_{\boldsymbol{\Gamma}}^{\mathbf{n}} = \underbrace{\sum_{i=1}^{p} \mathbf{S}^{\mathbf{i}} \left(\mathbf{R}_{\mathbf{i}} \mathbf{p}_\Gamma^n \right)}_{\mathbf{p}_\Gamma^{n,i}} \tag{3.2-8}$$

avec $\boldsymbol{R_i}$ l'opérateur de restriction qui extrait, dans un vecteur d'interface quelconque, les degrés de liberté appartenant au $i^{ème}$ sous-domaine. Chaque produit matrice-vecteur local $S^i p_\Gamma^{n,i}$ se reformule comme un problème similaire au problème global sur le $i^{ième}$ sous-domaine mais enrichi d'une condition de Dirichlet non homogène $\boldsymbol{p}_\Gamma^{n,i}$

$$K^i p^{n,i} := \begin{bmatrix} K_{ii} & K_{i\Gamma} \\ K_{\Gamma i} & K^i_{\Gamma\Gamma} \end{bmatrix} \begin{Bmatrix} p_i^{n,i} \\ p_\Gamma^{n,i} \end{Bmatrix} = \begin{Bmatrix} 0 \\ T_\Gamma^{n,i} \end{Bmatrix} \quad (3.2\text{-}9)$$

avec $T_\Gamma^{n,i}$ les tractions d'interface associées à ladite condition de Dirichlet. En substituant la première équation dans la seconde, on retrouve ainsi

$$T_\Gamma^{n,i} = \left(K^i_{\Gamma\Gamma} - K_{\Gamma i} K_{ii}^{-1} K_{i\Gamma} \right) p_\Gamma^{n,i} = S^i p_\Gamma^{n,i} \quad (3.2\text{-}10)$$

D'où l'interprétation mécanique de méthode de Schur primale. A chaque itération n du GCPC, un champ de déplacement bord $p_\Gamma^{n,i}$ est imposé comme seul chargement sur chaque sous-domaine i et le complément de Schur local S^i, lui fait correspondre un champ d'effort bord $T_\Gamma^{n,i}$ associé par le seul comportement dudit sous-domaine Ω_i.

Puis on assemble ces tractions d'interface et on teste leur équilibre (*via* le résidu). Si il est atteint, le processus s'arrête, sinon on met à jour dans chaque sous-domaine un nouveau champ de déplacement bord $p_\Gamma^{n+1,i}$ et on réitère le processus. Finalement, à un champ de déplacement bord, le complément de Schur assemblé $\boldsymbol{S}$ fait correspondre le déséquilibre bord associé.

D'un point de vue parallélisme, la structure distribuée de l'étape (1), de loin la plus gourmande en CPU, oriente fortement les choix. En suivant le canevas méthodologique évoqué au §2.2, un parallélisme par envoi de message paraît tout indiqué. Chaque processeur gère les instructions et les données associées à quelques sous-domaines (en particulier les calculs élémentaires et les assemblages matriciels et vectoriels) puis calcule localement les composantes de (3.2-8) associées. Ensuite deux approches sont possibles: «maître-esclave» ou «esclave-esclave».

Soit un processeur «maître» centralise ces contributions $S^i p_\Gamma^{n,i}$ pour construire $\mathbf{Sp}_\Gamma^n$ (avec un surcoût dû aux communications collectives dimensionnées à l'interface). Mise à part l'étape (6) de préconditionnement elle aussi est distribuée, ce processeur maître gère seul les autres tâches mineures d'ordonnancement du GC. C'est le choix sous-optimal mais plus simple à mettre en œuvre retenu pour l'implantation de FETI dans *Code_Aster*.

Soit chaque processeur ne communique les données d'interfaces $\boldsymbol{S}^i \boldsymbol{p}_\Gamma^{n,i}$ qu'aux processeurs «voisins» concernés[23] et conduit lui-même les produits scalaires et les «`daxpy`» des étapes mineures. Ces différentes approches seront détaillées dans les chapitres suivants concernant FETI-1.

Remarque:

- *Le critère d'arrêt (5) est basé sur le résidu d'interface. Or ce dernier vérifie*

$$\underbrace{\|\mathbf{Ku}^n - f\|}_{\langle r^n, r^n \rangle^2} = \underbrace{\|\sum_{i=1}^{P} S^i u_\Gamma^{n,i} - b^i\|}_{\sum_{i=1}^{P} \langle r_\Gamma^{n,i}, r_\Gamma^{n,i} \rangle^2} := \underbrace{\|\mathbf{Su}_\Gamma^n - b\|}_{\langle r_\Gamma^n, r_\Gamma^n \rangle^2} \quad (3.2\text{-}11)$$

En méthode itérative de Schur primale, on peut donc contrôler le résidu global via le résidu d'interface. D'un point de vue mécanique, l'égalité entre les deux résidus n'a rien de surprenant compte-tenu du fait, qu'à chaque itération, les DDLs internes de chaque sous-domaine sont supposés être à l'équilibre. On verra qu'avec l'algorithme FETI, étant en dual, on n'a pas cette chance !

3.2.6 Neumann-Neumann

A chaque itération du GCPC il faut résoudre autant de problèmes de Dirichlet que de domaines ! Pour être compétitive, la complexité[24] calcul du solveur itératif doit donc se transcender et la convergence doit s'opérer en beaucoup moins d'itérations que d'inconnues d'interface. Sinon, autant utiliser un solveur direct sur le problème global ! Heureusement, depuis une quinzaine d'années des efforts de recherche importants ont produits une kyrielle de préconditionneurs d'interface: les préconditionneurs induits qui utilisent les préconditionneurs

23 C'est à dire des processeurs auxquels ont été attribués un ou plusieurs sous-domaines partageant une interface avec un ou plusieurs sous-domaines du processeur donné.

24 $\theta(kcN)$ avec k le nombre d'itérations, c la taille moyenne de largeur de bande (très importante pour $\boldsymbol{S}$) et N le nombre d'inconnues d'interface.

classiques du problème initial, les préconditionneurs «sonde» ('probing preconditioner'), les préconditionneurs «Wire-basket»…
Parmi eux, le préconditionneur dit de Neumann-Neumann[25] semble sortir du lot. Son principe est déduit de l'interprétation mécanique précédente. De (3.2-9) il apparaît que l'inverse du complément local de Schur $\left(S^i\right)^{-1}$, est l'opérateur qui à un champ local de traction d'interface $T_\Gamma^{n,i}$ (qui joue cette fois le rôle d'une condition de Neumann non homogène) associe le champ local bord résultant $p_\Gamma^{n,i}$

$$\left(S^i\right)^{-1} T_\Gamma^{n,i} = \left(K_{\Gamma\Gamma}^i - K_{\Gamma i} K_{ii}^{-1} K_{i\Gamma}\right)^{-1} T_\Gamma^{n,i} = p_\Gamma^{n,i} \tag{3.2-12}$$

Un conditionneur global peut alors être construit en approximant l'inverse de la somme par la somme des inverses

$$M_{NN}^{-1} = \sum_{i=1}^{P} \left(S^i\right)^{-1} \tag{3.2-13}$$

Si on récapitule, cet algorithme de Neumann-Neumann peut être vu comme un solveur d'interface à deux niveaux basé sur un GCPC (cf. figure 3.2-3) :

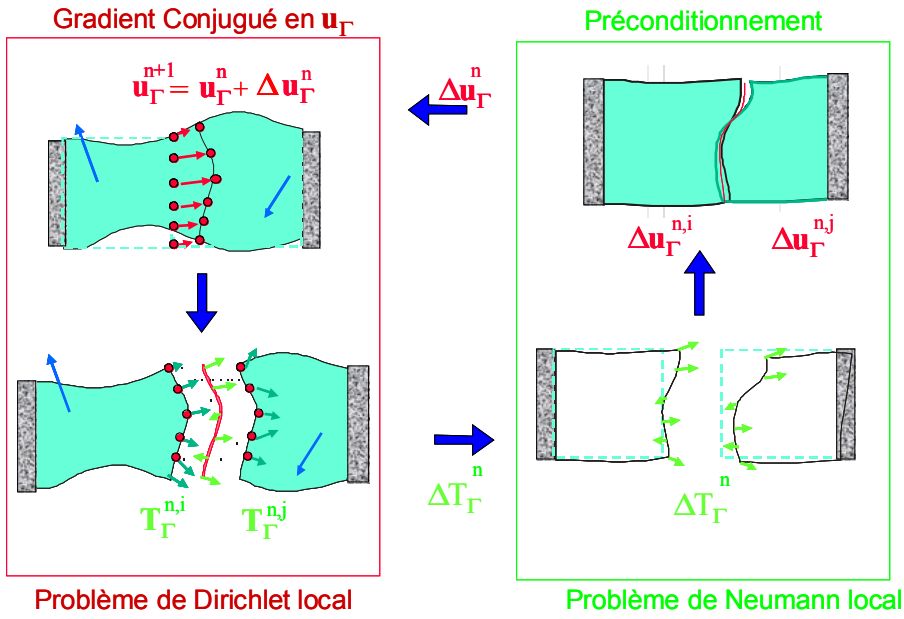


Figure 3.2-3._ Illustration de la méthode de Schur itératif primal[26].

- **A chaque itération du GCPC, on va résoudre P problèmes indépendants de Dirichlet**: des champs de déplacement bord $u_\Gamma^{n,i}$ sont prescrits aux interfaces des sous-domaines à qui on va faire correspondre, sous-domaine par sous-domaine, des champs d'effort bord $T_\Gamma^{n,i} := S^i u_\Gamma^{n,i} - b^i$. Puis on moyenne ces contributions interface par interface. Par exemple dans le cas le plus simple d'une interface partagée entre les deux sous-domaines i et j cela peut s'écrire $\Delta T_\Gamma^n := \left(T_\Gamma^{n,i} + T_\Gamma^{n,j}\right)/2$.
- Puis, **P problèmes indépendants de Neumann sont inversés**: le champ de traction moyenné ΔT_Γ^n est imposé à l'interface et chaque sous-domaine évalue la correction du champ de déplacement correspondante $\Delta u_\Gamma^{n,i} := \left(S^i\right)^{-1} \Delta T_\Gamma^n$. Puis on moyenne à l'interface ces corrections, par exemple $\Delta u_\Gamma^n := \left(\Delta u_\Gamma^{n,i} + \Delta u_\Gamma^{n,j}\right)/2$, afin de mettre à jour le champ de déplacement bord: $u_\Gamma^{n+1} = u_\Gamma^n + \Delta u_\Gamma^n$.

25 Son nom provient du premier cas de figure dans lequel il a été utilisé: deux sous-domaines en vis-à-vis donc deux problèmes de Neumann à résoudre.
26 Illustration inspirée d'un papier de D.J.Rixen[Rix02].

Et, ainsi de suite itérativement jusqu'à convergence: équilibre des champs de déplacement et annulation du saut d'effort bord. En présence de sous structures flottantes[27], les compléments de Schur locaux des sous-domaines concernés ne sont plus inversibles et il faut alors prendre en compte une contrainte supplémentaire liée à leurs modes rigides. Cette contrainte est traitée par une technique de Krylov-augmenté en rajoutant un projecteur dans la mise à jour du champ de déplacement bord $u_\Gamma^n = u_\Gamma^n + P\Delta u_\Gamma^n$. C'est le fameux problème grossier tant recherché qui permet de faire communiquer globalement des informations d'un bout à l'autre de la structure.
On détaillera ces ingrédients pour FETI au §4 et on verra que la philosophie de cette famille d'algorithme est exactement duale de celle-ci, en cherchant à équilibrer les champs de traction pour annuler les sauts de déplacement à l'interface.

La plus grande part des calculs qu'impliquent ce processus peut bien sûr être réalisée en parallèle, sous-domaine par sous-domaine. Cela conduit à un haut niveau de parallélisme mais réduit singulièrement le flot d'informations aux structures contiguës. Comme on l'a déjà mentionné, ce type de schéma pâtit d'une absence d'échange entre tous les sous-domaines.
Pour y remédier, des variantes avec équilibrage (BDD pour 'Balanced Neumann-Neumann') ou enrichies d'un espace grossier ont vu le jour. Elles concurrencent parfois les méthodes de type FETI qui elles l'incluent structurellement. Dans les deux cas, le conditionnement du problème s'améliore grandement par rapport à l'estimation (3.2-6) et devient

$$\eta\left(M_{NNEG}^{-1}S\right) = \vartheta\left(\left(1+\log\left(\frac{H}{h}\right)\right)^2\right) \tag{3.2-14}$$

Il n'est plus que logarithmiquement dépendant du nombre de mailles par sous-domaines, ce qui confère à ces méthodes DD les qualités de scalabilité numérique et d'optimalité.

Remarque:

- *Lorsqu'un découpage induit des sous-domaines flottants, ce qui est souvent le cas en pratique, ce préconditionneur de Neumann-Neumann prend la forme*

$$\left(M_{NN}^{-1}\right)_{\text{flottant}} = \sum_{i=1}^{P}\left(S^i\right)^+ \tag{3.2-15}$$

où $\left(S^i\right)^+$ est un inverse généralisé du $i^{ème}$ complément de Schur local. Il est associé à un projecteur ad hoc et son calcul explicite n'est pas requis. Sa bonne qualité est moins cruciale que pour FETI-1, car il n'intervient ici que dans la phase de préconditionnement.

27 'Floating substructure', c'est-à-dire sous-structure hypostatique dont tous les modes de corps rigides potentiels n'ont pas été bloqués.

4 Méthode FETI: principe et algorithme

4.1 Description générale

Dans toutes les méthodes DD que nous avons introduites jusqu'à présent, les variables primales sont les déplacements. En particulier, dans les méthodes de Schur du chapitre précédent, on cherche à équilibrer les champs de déplacement pour annuler le saut d'inter-efforts à l'interface Γ. Les méthodes de Schur duales introduites par C.Farhat & F.X.Roux[FR91], méthodes de type FETI pour 'Finite Element Tearing and Interconnecting'[28] (1991), font exactement l'inverse.
Ce sont aussi des méthodes itératives qui réduisent le problème initial en un problème d'interface plus petit et mieux conditionné, mais elles équilibrent cette fois le champ de traction pour annuler le saut de déplacement à l'interface. Interviennent aussi nativement (c'est-à-dire sans même regarder les aspects préconditionneurs) des ingrédients numériques supplémentaires: variables de Lagranges, modes rigides, projecteur et problème grossier. Mais avant d'en détailler le principe et se souvenant du canevas introduit pour Neumann-Neumann (cf. §3.2), on peut déjà en brosser à gros traits le fonctionnement. La version initiale de FETI[29] avec préconditionneur de Dirichlet peut être vue comme un solveur d'interface à deux niveaux basé sur un GCPC (cf. figure 4.1-1):

- **A chaque itération du GCPC, sont résolus P problèmes indépendants de Neumann**: partant d'une estimation initiale d'inter-efforts λ^n vérifiant une certaine contrainte d'admissibilité on lui fait correspondre un saut de déplacement bord du type $r_\Gamma^n = P^T\left(d - \left(K_i^{-1} + K_j^{-1}\right)\lambda^n\right)$. Ces composantes sont calculées sous-domaine par sous-domaine. Puis, on moyenne ces sauts interface par interface. Par exemple dans le cas simple d'une interface partagée entre les deux sous-domaines i et j, cela peut s'écrire $\Delta u_\Gamma^n = \left(r_\Gamma^{n,i} + r_\Gamma^{n,j}\right)/2$. L'interface de i se voit alors imposée le déplacement bord $\Delta u_\Gamma^{n,i} = \Delta u_\Gamma^n$ et l'interface de j, $\Delta u_\Gamma^{n,j} = -\Delta u_\Gamma^n$.
- **Puis l'étape de préconditionnement requiert l'inversion de P problèmes indépendants de Dirichlet**: partant des déplacement bord précédents $\Delta u_\Gamma^{n,i}$, chaque sous-domaine détermine les inter-efforts locaux correspondants $\Delta T_\Gamma^{n,i} = S^i \Delta u_\Gamma^{n,i}$. Il n'y a plus qu'à les moyenner, interface par interface, par exemple en imposant $\Delta\lambda^n = \left(\Delta T_\Gamma^{n,i} - \Delta T_\Gamma^{n,j}\right)/2$ pour corriger le champ de Lagrange modélisant les efforts de traction à l'interface $\lambda^{n+1} = \lambda^n + P\Delta\lambda^n$.

Et ainsi de suite, itérativement, jusqu'à convergence: équilibre du champ de traction et annulation du saut de déplacement bord. En présence de sous structures flottantes, les matrices de rigidité locales ne sont plus inversibles et il faut alors prendre en compte une contrainte supplémentaire liée à leurs modes rigides. Cette contrainte d'admissibilité est traitée par une technique de Krylov-augmenté en rajoutant un projecteur P dans le calcul du saut de déplacement bord et sa mise à jour. C'est le fameux problème grossier[30] qui est très recherché par toutes les méthodes DD car il permet de faire communiquer globalement des informations d'un bout à l'autre de la structure. Avec FETI-1, il est résolu deux fois par itération. Il propage ainsi l'erreur globalement à tout le domaine, accélérant la convergence du processus[31]. Sa bonne résolution est cruciale pour le déroulement du processus. C'est d'ailleurs le traitement des modes rigides sous-tendant ce problème grossier qui distingue FETI des autres méthodes DD utilisant des multiplicateurs de Lagrange.

28 Son nom rend mémoire aux travaux de G.Kron (*A set of principles to interconnect the solutions of physical systems*. J. Appl. Phys., **24** (1953), pp965-980) sur les méthodes de 'tearing' (déchirement) pour les modèles de circuit électrique. Le principe consiste bien ici aussi à «déchirer» un domaine en sous-domaines, puis à les «interconnecter» *via* les inter-efforts aux interfaces.
29 Appelée aussi FETI niveau un ('one-level FETI') ou simplement FETI-1.
30 Problème de petite taille positionné sur toute l'interface.
31 Convergence constatée empiriquement dans C.Farhat et al (*Optimal convergence properties of the FETI domain decomposition method*. Computer Meth. Appl. Mech. And Engrg., **115-3/4** (1994), pp367-388) et prouvée mathématiquement par J.Mandel et al (*On the convergence of a substructuring methode with Lagrange multipliers*. Numerische Mathematik (1994)).

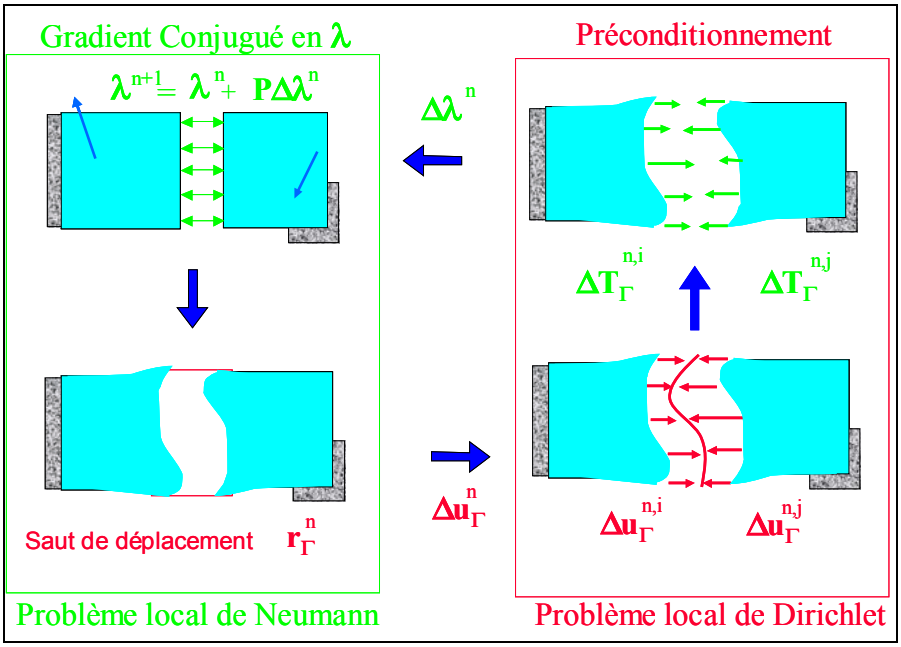


Figure 4.1-1._ Illustration de la méthode FETI-1[32] .

Comme on l'a déjà observé l'enchaînement des tâches est «dual» de celui d'une méthode itérative de Schur primale avec préconditionnement de Neumann-Neumann (cf. §3.2). On verra par la suite que ce parallèle entre les deux méthodes DD n'est pas fortuit.
Cette méthode DD affiche une scalabilité numérique intrinsèque indépendante de tout préconditionneur et elle offre des possibilités de réduction de modèle en adaptant, localement et dynamiquement, la discrétisation polynomiale des multiplicateurs sur l'interface.
Cette discrétisation séparée peut aussi être déployée pour combiner des sous-domaines maillés par des éléments incompatibles ou non conformes. De manière générale, l'introduction d'une interface traitée en dual confère à ce type de méthode la souplesse d'un toolkit d'appariement de calculs distribués (DD multiniveau, modification structurale, multiéchelle, multiphysique…).

4.2 Problème mécanique

4.2.1 Problème de point selle

En mécanique des structures, la résolution du système linéaire[33] $\mathbf{Ku}=f$ découle en fait d'un problème de minimisation

$$(P_1) \quad u = Arg \min_{v \in H(\Omega)} J(v) \tag{4.2-1}$$

d'une forme quadratique positive représentant l'énergie du domaine

$$J(v) := \frac{1}{2} a(v,v) - (v, f)_\Omega - (v, h)_{\Gamma_2} \tag{4.2-2}$$

Où $a(.,.)$, $(.,.)_\Omega$ et $(.,.)_{\Gamma_2}$ représentent, respectivement, la forme variationnelle et les produits scalaires $L^2(\Omega)$ et $L^2(\Gamma_2)$ habituels: Ω est l'ouvert de $\Re^\alpha$ modélisant la structure[34], Γ_1 sa partie de frontière à déplacement imposé et Γ_2 celle des efforts imposés, $\Gamma_1 \cup \Gamma_2 = \partial\Omega$. Etant donnés des chargements volumique f et surfacique h, on cherche donc le champ de déplacement u minimisant J sur l'espace fonctionnel des champs cinématiquement admissibles

32 Illustration inspirée d'un papier de D.J.Rixen[Rix02].
33 Pour simplifier les écritures on va rester dans le cadre des systèmes SPD. C'est celui que l'on rencontre le plus fréquemment dans *Code_Aster* et c'est aussi celui pour lequel a été mis au point FETI-1. Depuis sa genèse, le périmètre de cette méthode a été étendu à la plupart des configurations: non symétrique, indéfinie, complexe…
34 $\alpha = 2$ en 2D et 3 en 3D.

$$H(\Omega) := \left\{ v : \Omega \to \Re^{\alpha} / v \in \left(H^1(\Omega)\right)^{\alpha} \text{ et } v_{|\Gamma_1} = 0 \right\} \tag{4.2-3}$$

On dissocie le domaine initial en P sous-domaines et on introduit les restrictions des fonctionnelles précédentes à chaque sous-domaine. Lorsque le partitionnement fait apparaître des points multiples, deux descriptions de l'interface sont possibles: l'interface purement géométrique, dite primale, et la duale prenant en compte certaines connectivités entre les sous-domaines. Plusieurs stratégies sont associées à cette description duale des connectivités liées aux points multiples: la redondante qui prend en compte tous les couples possibles, la non-redondante qui n'en prend que $m-1$ pour un point de multiplicité m et l'orthogonale. Elles ont fait l'objet de nombreux papiers et sous-tendent autant de variantes de FETI, mais aucune ne semble se détacher des autres en terme de performances. Dans la suite du document nous nous référerons à l'interface duale redondante de l'algorithme FETI initial. C'est la variante pour l'instant retenue dans *Code_Aster*.

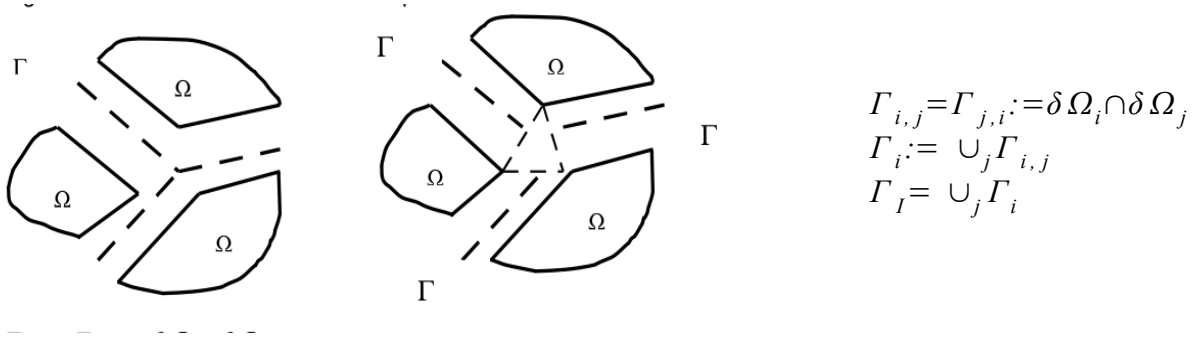


Figure 4.2-1._ Définition de l'interface primale (géométrique) et duale (par connectivité).

On montre alors que le problème (P_1) est équivalent au problème (P_2) qui s'intéresse au P-uplet $u := \left\{u_i\right\}_{i=1}^{P}$ minimisant

$$(P_2) \quad u_i = Arg \min_{v_i \in H_i(\Omega)} J_i(v_i) \quad (i = 1 \ldots P) \tag{4.2-4}$$

avec

$$\begin{aligned} J_i(v_i) &:= \frac{1}{2} a_i(v_i, v_i) - (v_i, f)_{\Omega_i} - (v_i, h)_{\Gamma_2^i} \\ H_i(\Omega) &:= \left\{ v_i = v_{|\Omega_i},\ v \in \left(H(\Omega)\right)^{\alpha} / v_i = 0 \text{ sur } \overline{\Omega - \Omega_i} \right\} \end{aligned} \tag{4.2-5}$$

et lui rajoutant la contrainte de continuité du champ de déplacement à l'interface

$$u_i = u_j \text{ sur } \Gamma_I \tag{4.2-6}$$

Ces P sous-problèmes de minimisation sous contrainte sont traités par une technique de dualisation avec introduction de multiplicateurs de Lagrange idoines: λ représentant le champ d'inter-efforts à l'interface entre les sous-domaines. Bref, cela conduit à un problème mixte ou primal-dual dont on cherche un point selle[35]. On cherche alors les déplacements $u := \left\{u_i\right\}_{i=1}^{P} \in \bigotimes_i H_i(\Omega)$ et les multiplicateurs d'interconnexion $\lambda := \left\{\lambda_i\right\}_{i=1}^{T} \in \bigotimes_{k,l} H^{-1/2}\left(\Gamma_{k,l}\right)$ qui constituent un point-selle (u, λ) du Lagrangien

$$(P_3) \quad J^{¿}(v_1, \ldots v_P, \mu_1, \ldots \mu_T) := \sum_{i=1}^{P} J_i(v_i) + \sum_{i,j=1}^{P} (v_i - v_j, \mu_k)_{\Gamma_I} \tag{4.2-7}$$

avec

$$(v_i - v_j, \mu_k)_{\Gamma_I} := \int_{\Gamma_{i,j}} \mu_k \, Tr_{\Gamma_{i,j}}(v_i - v_j) \, d\Gamma \tag{4.2-8}$$

Ces composantes vérifient les relations classiques, avec des champs admissibles $v := \left\{v_i\right\}_{i=1}^{P}$ et $\mu := \left\{\mu_i\right\}_{i=1}^{T}$

,

$$J^{¿}(u_1, \ldots u_P, \mu_1, \ldots \mu_T) \le J^{¿}(u_1, \ldots u_P, \lambda_1, \ldots \lambda_T) \le J^{¿}(v_1, \ldots v_P, \lambda_1, \ldots \lambda_T) \tag{4.2-9}$$

35 Il existe dès que les formes bilinéaires considérées sont SPD. Elles héritent en cela des propriétés de la forme non décomposée.

L'inégalité de gauche assure la continuité (4.2-6) et celle de droite, la minimisation de la solution retenue parmi toutes les solutions admissibles, c'est-à-dire parmi celles vérifiant la condition de continuité. Evidemment, au final, seule la première partie des arguments du point-selle nous intéresse. Une analyse théorique détaillée de cette formulation mixte a été conduite par F.X.Roux dans sa thèse[Rou89].

4.2.2 Problème discrétisé

La discrétisation sur l'ensemble des fonctions de forme conduit ensuite au système matriciel

$$(P_4)\begin{cases} K_i u_i = f_i - \sum_{k=1}^{k_i} \left(R_k^i\right)^T \lambda \quad (i=1\dots P) \\ \sum_{i=1}^{P}\sum_{j=1}^{k_i} \left(R_j^i u_i + R_i^j u_j\right) = 0 \end{cases} \tag{4.2-10}$$

où

- $\boldsymbol{K}_i$, $\boldsymbol{u}_i$ et $\boldsymbol{f}_i$ sont, respectivement, les matrices de rigidité, les vecteurs de déplacement recherchés et de forces imposées sur les domaines Ω_i.
- k_i le nombre de sous-domaines voisins de Ω_i et λ le vecteur des multiplicateurs de Lagrange.
- Les matrices R_j^i sont des matrices booléennes signées d'interconnexion entre les $i^{ème}$ et $j^{ème}$ sous-domaines. Elles permettent de localiser une quantité globale au sous-domaine Ω_i sur son interface avec Ω_j.

Les équations de continuité (4.2-10b) assurent la solvabilité du système qui, sinon, comporterait plus d'inconnues que d'équations ! Les matrices d'interconnexion ont une structure du type

$$R_j^i = \begin{bmatrix} & 0_{kl}^1 & \\ 0_{kl}^3 & I_{kl} & 0_{kl}^4 \\ & 0_{kl}^2 & \end{bmatrix} \begin{matrix} \updownarrow m_1(i,j) \\ \updownarrow m_C(i,j) \\ \updownarrow m_2(i,j) \end{matrix} \tag{4.2-11}$$
$$\begin{matrix} \leftrightarrow & \leftrightarrow & \leftrightarrow \\ m_3(i,j) & m_C(i,j) & m_4(i,j) \end{matrix}$$

où on note

- n_I le nombre total de degrés de liberté d'interface,
- n_i^i et n_Γ^i, respectivement, les nombres de degrés de liberté intérieurs et d'interface du sous-domaine Ω_i,
- 0_{kl}^1 et 0_{kl}^2, des matrices identiquement nulles de tailles, respectivement, $\left(n_i^i + n_\Gamma^i\right) \times m_1(i,j)$ et $\left(n_i^i + n_\Gamma^i\right) \times m_2(i,j)$,
- $\mathbf{0}_{kl}^3$ et $\mathbf{0}_{kl}^4$, des matrices identiquement nulles de tailles, respectivement, $m_3(i,j) \times m_C(i,j)$ et $m_4(i,j) \times m_C(i,j)$,
- $\mathbf{I}_{kl}$ une matrice identité signée carrée de taille $m_C(i,j)$ (dans *Code_Aster* on a pris la convention $I_{kl} = \delta_{kl}$ si $j<i$, sinon $I_{kl} = -\delta_{kl}$).

Avec les relations

$$\begin{aligned} n_I &= m_1(i,j) + m_C(i,j) + m_2(i,j) \\ n_i^i + n_\Gamma^i &= m_3(i,j) + m_C(i,j) + m_4(i,j) \end{aligned} \tag{4.2-12}$$

Remarque:

- *Dans le cas particulier de deux sous-domaines*
  $$m_1(i,j) = m_2(i,j) = m_4(i,j) = 0 \text{ et } m_C(i,j) = n_\Gamma^l .$$

Pour des contingences mémoires, ces matrices d'interconnexion ne sont pas construites explicitement. Il faut les considérer comme une formalisation confortable d'un processus d'extraction plutôt que comme un véritable produit matrice-vecteur.

4.2.3 Pseudo-inverse[36]

Dans le cas le plus simple (peu fréquent en pratique), les matrices de rigidité sont inversibles (*i.e.* elles ne correspondent pas à des sous-domaines flottants) et (4.2-10a) se réécrit :

$$u_i=\left(K_i\right)^{-1}\left\{f_i-\sum_{k=1}^{k_i}\left(R_k^i\right)^T\lambda\right\}\ \ (i=1\ldots P) \tag{4.2-13}$$

Malheureusement, dans la plupart des cas industriels , le découpage automatique conduit à des sous-domaines flottants. Leurs matrices de rigidité locales sont alors singulières et une technique d'inversion particulière doit être mise en place : la construction de pseudo-inverses. Si le système initial est consistant, ces pseudo-inverses (nous verrons comment les calculer effectivement par la suite) s'écrivent

$$u_i=\left(K_i\right)^{+}\left\{f_i-\sum_{k=1}^{k_i}\left(R_k^i\right)^T\lambda\right\}-B_i\alpha_i\ \ (i=1\ldots Q) \tag{4.2-14}$$

avec

- $Q<P$ le nombre de sous-domaines flottants,
- $\boldsymbol{B}_i$ la matrice $\left(n_i^i+n_\Gamma^i\right)\times n_r^i$ des n_r^i modes de corps rigides du $i^{\text{ème}}$ sous-domaine,
- α_i un vecteur d'amplitude de ces modes.

Idéalement, c'est-à-dire dans les cas de figure ne présentant pas de modes rigides parasites (*i.e.* générés par des mécanismes internes à la sous-structures, des degrés de liberté non alimentés en raideur, des modes cinématiques dus à la sous-intégration…), les sous-structures 3D flottantes connexes comportent au maximum $n_r^i=6$ modes rigides. En `2D-PLAN` et en `2D-AXI`, ce chiffre tombe respectivement, à $n_r^i=3$ et $n_r^i=1$. Les colonnes de $\boldsymbol{B}_i$ constituent ainsi une des bases possibles de $ker(\boldsymbol{K}_i)$.

Remarque:

- *Ce type de pseudo-inverse n'a pas à être calculé explicitement. FETI étant une méthode de sous-structuration basée sur un GC, on a « uste» besoin de connaître l'action du pseudo-inverse sur un vecteur $\left(K_i\right)^{+}u$, avec un coût calcul équivalent à $\left(\mathbf{K}_i\right)^{-1}\mathbf{u}$.*

Le remplacement de l'équation (4.2-13) par (4.2-14) introduit un nouvel ensemble d'inconnues $\alpha:=\left\{\alpha_i\right\}_{i=1}^{Q}$ rendant le problème (P_4) indéterminé. Cependant, puisque la matrice de raideur est symétrique et réelle, l'équation singulière

$$K_iu_i=f_i-\sum_{k=1}^{k_i}\left(R_k^i\right)^T\lambda\ \ (i=1\ldots P) \tag{4.2-15}$$

admet au moins une solution si et seulement si (alternative de Fredholm)

$$f_i-\sum_{k=1}^{k_i}\left(R_k^i\right)^T\lambda\in\mathrm{Im}\left(K_i\right)=\ker\left(K_i^*\right)^{\mathrm{ortho}}=\ker\left(K_i\right)^{\mathrm{ortho}} \tag{4.2-16}$$

ce qui s'écrit, puisque $\boldsymbol{B}_i$ engendre $ker(\boldsymbol{K}_i)$

36 Appelé aussi inverse généralisé. Il existe plusieurs procédés pour le déterminer (SVD, Q -méthode …) mais son unique définition est de vérifier les quatre conditions de Moore-Penrose. Pour $\boldsymbol{K}_i$, c'est la matrice X vérifiant:

$$K_i\mathbf{X}\mathbf{K}_i=K_i\quad\left(K_iX\right)^T=K_iX$$
$$\mathbf{X}\mathbf{K}_iX=X\quad\left(\mathbf{X}\mathbf{K}_i\right)^T=\mathbf{X}\mathbf{K}_i$$

Formellement $\left(\mathbf{K}_i\right)^{+}:=\left(\left(\mathbf{K}_i\right)^T\mathbf{K}_i\right)^{-1}\left(\mathbf{K}_i\right)^T$, mais on ne constitue pas explicitement cette matrice car elle est trop coûteuse à mettre en place.

$$\left(B_i\right)^T\left(f_i-\sum_{k=1}^{k_i}\left(R_k^i\right)^T\lambda\right)=0\quad\left(i=1\ldots P\right) \tag{4.2-17}$$

D'où un jeu de nouvelles équations permettant de trouver les nouvelles inconnues α_i. Cela signifie que le second membre de (4.2-15) ne doit pas comporter de composante dans le noyau de la $i^{ème}$ matrice de rigidité.

Remarque:

•La relation (4.2-17) traduit le fait que les modes de corps rigides ne produisent pas d'énergie interne car

$$\left(B_i\right)^T\left(f_i-\sum_{k=1}^{k_i}\left(R_k^i\right)^T\lambda\right)=\left(B_i\right)^T K_i u_i \tag{4.2-18}$$

4.3 Problème d'interface FETI

4.3.1 Problème de Stokes

En substituant (4.2-14) dans la contrainte de compatibilité (4.2-10b) et en prenant en compte (4.2-17), $\left(P_4\right)$ devient le problème d'interface FETI qui réalise ainsi une condensation du problème de raccord sur les multiplicateurs de Lagrange:

$$\left(P_5\right)\quad\begin{bmatrix}F_I & G_I\\ \left(G_I\right)^t & 0\end{bmatrix}\begin{bmatrix}\lambda\\ \alpha\end{bmatrix}=\begin{bmatrix}d\\ e\end{bmatrix}\begin{matrix}\updownarrow\ n_I\\ \updownarrow\ Qn_r^i\end{matrix} \tag{4.3-1}$$

avec

- la matrice carré FETI d'interface[37] (de taille n_I) $F_I:=\sum_{i=1}^{P}R_i\left(K_i\right)^+\left(R_i\right)^T$ en notant $R_i:=\sum_{k=1}^{k_i}R_k^i$,
- la matrice $G_I:=\begin{bmatrix}R_1B_1 & \ldots & R_QB_Q\end{bmatrix}$ qui stocke la trace des modes de corps rigides sur l'interface Γ_I,
- les vecteurs solutions $\lambda:=\begin{bmatrix}\lambda_1 & \ldots & \lambda_{n_I}\end{bmatrix}^T$ et $\alpha:=\begin{bmatrix}\left(\alpha_1\right)^T & \ldots & \left(\alpha_Q\right)^T\end{bmatrix}^T$,
- les seconds membres $d:=\sum_{i=1}^{P}R_i\left(K_i\right)^+f_i$ et $e:=\begin{bmatrix}\left(f_1{}^TB_1\right) & \ldots & \left(f_Q{}^TB_Q\right)\end{bmatrix}^T$,
- $\left(K_i\right)^+=\begin{cases}\left(K_i\right)^{-1}\text{ si }\Omega^i\text{ non flottant}\\ \text{sinon une pseudo-inverse}\end{cases}$

La matrice du problème d'interface FETI est appelée matrice de Lagrange, notée $\boldsymbol{L}$, on l'a d'ailleurs symétrisée sciemment afin d'en rendre la résolution plus simple. Pour traiter des problèmes de grandes tailles en parallèle, il n'est évidemment pas souhaitable de l'assembler explicitement, d'où le recourt à un GC. Ainsi, comme dans l'algorithme de Neumann-Neumann, une fois les matrices $\left(K_i\right)^+$ obtenues, les produits matrices vecteurs bold $F_I\boldsymbol{v}$, avec $\boldsymbol{v}$ une vecteur d'interface de taille n_I, pourront s'effectuer par des descentes-remontées distribuées et parallèles.

Remarques:

- *La taille des problèmes $\left(P_4\right)$ et $\left(P_5\right)$ est similaire, le second comportant juste $Q_r:=\sum_{i=1}^{Q}n_r^i$ équations supplémentaires.*

37 Appelée parfois opérateur de flexibilité d'interface ('interface flexibility operator').

- (P_5) *est un problème de Stokes, comme en CFD, où* λ *joue le rôle de la vitesse et* α *celui de la pression.*
- *Lorsque la décomposition du problème initial ne comporte pas de points de croisement, i.e. les nœuds d'interface n'appartiennent qu'à deux sous-domaines, l'opérateur de FETI est SPD et* (P_5) *est donc régulier. Ce cas particulier est rare : par exemple, une structure élancée coupée en tranches orthogonalement à son axe principal. En pratique, tous les partitionneurs produisent ce type de points qui ont pour particularité de fixer des équations de continuité redondantes. Une stratégie possible consisterait à développer des heuristiques ad hoc pour les trier automatiquement, mais cela complique sensiblement l'implantation et la maintenance de l'algorithme.*

En fait, il est plus facile d'inclure toutes les contraintes (même les redondantes) et d'admettre que, bien que $\boldsymbol{L}$ *soit singulière, le problème* (P_5) *est consistant et admet au moins une solution : toute combinaison linéaire de* λ *et* α *satisfaisant l'équilibre nous convient, elles ne sont que des intermédiaires pour atteindre les* $\boldsymbol{u}_i$ *via (4.2-13/14). De plus, tant que* $\boldsymbol{L}$ *reste positif, un GC est censé fonctionner. Il est de toute façon le seul recours possible car un solveur direct achoppe sur un système singulier.*

Cependant, bien que $\boldsymbol{L}$ soit symétrique, cette matrice est qu'indéfinie et on ne peut donc lui appliquer tel quel un gradient conjugué.

4.3.2 Problème linéaire mixte

On reformule donc le problème d'interface (P_5) comme le problème (P_6) de minimisation d'une fonctionnelle quadratique sous contraintes linéaires

$$(P_6)\begin{cases}\min_\lambda \Phi(\lambda)=\frac{1}{2}\lambda^T F_I \lambda-\lambda^T\left(G_I \alpha+d\right)\\ \text{sous la contrainte } \left(G_I\right)^T\lambda=e\end{cases} \tag{4.3-2}$$

Cette équivalence entre un problème de type point-selle (P_3), un système matriciel (P_5) et cette minimisation décrit en fait une classe de problèmes linéaires mixtes très répandue (mécanique des fluides dans *Code_Saturne,* méthode de contact unilatéral dans *Code_Aster*…). Il se résout numériquement par une méthode de type Uzawa ou un Lagrangien augmenté *via* un projecteur orthogonal sur l'espace des contraintes admissibles.

La matrice de FETI, $\boldsymbol{F_I}$, étant positive et semi-définie[38], on peut résoudre ce problème avec un Gradient Conjugué Projeté (GCP) d'où la terminologie anglo-saxonne de 'substructure based CPG algorithm'. Cet algorithme est similaire à un GCPC standard dont le préconditionnement est enrichi d'un opérateur de projection supplémentaire qui assure la satisfaction de la contrainte (4.3-2b).
Ce processus itératif débute par le choix d'une valeur de λ qui satisfasse ladite contrainte

$$\lambda=\lambda^0:=G_I\left(\left(G_I\right)^T G_I\right)^{-1} e \tag{4.3-3}$$

Remarques:

- $\boldsymbol{G_I}$ *est une matrice régulière, de taille* $n_I\times Q_r$ *, dès que le domaine initial n'est pas flottant… ce qui, en statique, est la moindre des choses ! Ainsi* $(\boldsymbol{G_I})^T\boldsymbol{G_I}$ *devient une matrice carrée inversible d'ordre* Q_r *.*
- *En séquentiel, le stockage de cette matrice carrée et de* $\boldsymbol{G_I}$ *peut devenir problématique sur des machines disposant de peu de mémoire et avec un partitionnement élevé. Aussi l'implantation FETI de Code_Aster propose un mot-clé (*`STOCKAGE_GI`*) pour décider si elles sont calculées et stockées une fois pour toute, recalculées à chaque itération ou si cette décision dépend des proportions des données et en particulier de l'encombrement mémoire de ces objets par rapport aux plus gros objets JEVEUX contiguës en mémoire. Dans tous les cas* $\left(\left(G_I\right)^T G_I\right)^{-1}$ *est calculée et stockée une seule fois et bien sûr, dans le cas de problèmes avec multiples seconds*

38 En présence de points de jonction ([FR94] Th5.3) sinon elle est SPD.

membres, ces opérations ne sont effectuées qu'au premier pas de temps : la matrice et l'interface ne changeant pas, il en est de même des restrictions des modes rigides.

Et, pour s'assurer que cette contrainte reste vérifiée au cours du processus, on va enlever des itérés λ^n les composantes qui ne la satisfont pas. Pour ce faire, on va mettre à jour, itération après itération, les λ^n en adjoignant à la valeur initiale, une composante $P\tilde{\lambda}^n$ vérifiant la contrainte homogène

$$\begin{aligned} &\lambda^n := \lambda^0 + P\tilde{\lambda}^n \\ &\text{avec } \left(G_I\right)^T P\tilde{\lambda}^n = 0 \end{aligned} \tag{4.3-4}$$

Cette composante «homogène» est filtrée en la projetant sur $(\boldsymbol{G_I})^T$ *via*

$$\begin{aligned} &\mathbf{P:} = I - G_I\left(\left(G_I\right)^T G_I\right)^{-1}\left(G_I\right)^T \\ &\text{ainsi } \left(G_I\right)^T P = 0 \end{aligned} \tag{4.3-5}$$

A chaque itération n, en prenant les notations de l'algorithme (4.4-1) du paragraphe suivant

$$\left(G_I\right)^T\tilde{\lambda}^n := \left(G_I\right)^T\left(\lambda^{n+1}-\lambda^n\right) = \alpha^n\left(G_I\right)^T p_\Gamma^n = \alpha^n\left(G_I\right)^T \mathbf{Pr}_\Gamma^n = \alpha^n\left(\mathbf{PG_I}\right)^T r_\Gamma^n = 0 \tag{4.3-6}$$

Remarques:

- *Cette approche est à rapprocher des techniques de Krylov-augmenté[Saa03] qui sont largement utilisées pour adjoindre des contraintes lors de résolutions de systèmes linéaires. On choisit généralement de les mettre en œuvre par un projecteur.*
- *Cette projection requiert seulement la factorisation de la matrice carrée symétrique $(\boldsymbol{G_I})^T\boldsymbol{G_I}$. Elle intervient dans des résolutions du type $\left[\left(G_I\right)^T G_I\right]x = y$ qui modélisent en fait un problème grossier de taille $Q_r << n_I << N$.*
- *L'opérateur de projection $\boldsymbol{P}$ est symétrique. Cette propriété sera mise à profit pour la double projection du GCPPC.*

4.3.3 Problème projeté

Le problème mixte (P_6) devient alors équivalent au problème projeté en $\tilde{\lambda}$

$$\left(P_7\right) \quad \left(P^T F_I P\right)\tilde{\lambda} = P^T\left(d - F_I\lambda^0\right) \tag{4.3-7}$$

On peut résoudre ce nouveau *via* un GCP car son opérateur est symétrique semi-définie positive.

Remarque:

- *En fait, toute une famille de projecteurs est licite, chacun étant lié à une décomposition particulière de $\Re^{n_I}$, avec $\boldsymbol{Q}$ une matrice SPD,*

$$P(Q) := I - \mathbf{QG_I}\left(\left(G_I\right)^T\mathbf{QG_I}\right)^{-1}\left(G_I\right)^T \tag{4.3-8}$$

*Puisque λ et $\boldsymbol{G_I}$ sont, respectivement, homogènes à des inter-efforts à l'interface et à un déplacement, il s'ensuit que $\boldsymbol{Q}$ doit l'être avec un champ de contrainte. Pour simplifier l'algorithmique, on choisit bien souvent: $\boldsymbol{Q}=\boldsymbol{I}$. C'est la solution retenue dans Code_Aster. Lorsque la structure est «légèrement» hétérogène on préconise plutôt $Q = M_L^{-1}$ **ou** M_{SL}^{-1} (préconditionneur lumped ou super-lumped cf §4.5) voire en cas d'hétérogénéités plus forte $Q = M_D^{-1}$ (préconditionneur de Dirichlet).*

4.4 Algorithme simplifié

4.4.1 Gradient conjugué projeté

En récapitulant les éléments des paragraphes précédents, l'algorithme de FETI-1 simplifié s'écrit donc autour d'un GCP du problème d'interface:

Initialisation λ^0 donné (4.3-3), $r_\Gamma^0 = F_I \lambda^0 - d$, $g_\Gamma^0 = \mathbf{Pr}_\Gamma^0$, $p_\Gamma^0 = g_\Gamma^0$

Boucle en n

(1) $z_\Gamma^n = F_I p_\Gamma^n$

(2) $\alpha^n = \dfrac{\langle g_\Gamma^n, g_\Gamma^n \rangle}{\langle z_\Gamma^n, p_\Gamma^n \rangle}$

(3) $\lambda^{n+1} = \lambda^n + \alpha^n p_\Gamma^n$ (nouvel itéré)

(4) $r_\Gamma^{n+1} = r_\Gamma^n - \alpha^n z_\Gamma^n$ (nouveau résidu)

(5) $g_\Gamma^{n+1} = \mathbf{Pr}_\Gamma^{n+1}$

(6) Test d'arrêt **via** $\|g_\Gamma^{n+1}\| < \varepsilon \|g_\Gamma^0\|$

(7) $\beta^{n+1} = \dfrac{\langle g_\Gamma^{n+1}, g_\Gamma^{n+1} \rangle}{\langle g_\Gamma^n, g_\Gamma^n \rangle}$

(8) $p_\Gamma^{n+1} = g_\Gamma^{n+1} + \beta^{n+1} p_\Gamma^n$ (nouvelle direction de descente)

Algorithme 4.4-1._ FETI-1 simplifié.

Remarque:

•Dans Code_Aster le critère d'arrêt est piloté par le paramètre `RESI_RELA` *du mot-clé* `SOLVEUR`*. Ce test d'arrêt s'effectue aussi dans la phase d'initialisation pour ne pas débuter une séquence avec un second membre quasi-nul. On teste en particulier si* $\|g_\Gamma^0\| < \varepsilon \|f_i\|$ et $\|f_i\| <$ prec_machine .

Ce cas de figure peut être fréquent lors des problème avec multiples seconds membres (thermo-mécanique, mécanique linéaire sans recalcul de la matrice tangente…) lorsqu'on utilise les techniques d'accélérations de FETI. Le second membre dual corrigé tenant alors compte de l'information spectrale déjà contenue dans les solutions précédentes.

Le nombre maximun d'itérations de l'algorithme est borné par la valeur renseignée dans le mot-clé `NMAX_ITER` *. Une valeur pré-calculée est fournie par défaut.*

Au bout d'un certain nombre d'itérations, la mise à jour du résidu d'interface par la relation de récurrence (4) n'est plus pertinente du fait des erreurs d'arrondis. Elle diverge par rapport à sa valeur réelle et risque de fausser le comportement de l'algorithme. Il faut alors le recalculer explicitement via la formule $r_\Gamma^{n+1} = F_I \lambda^{n+1} - d$ *. Le mot-clé* `REAC_RESI` *permet de fixer cette fréquence de réactualisation : typiquement une réactualisation toutes les 10 ou 20 itérations.*

•Pour pallier les mêmes problèmes, on peut arrêter l'algorithme et le redémarrer en partant de la dernière valeur du vecteur de Lagrange. Le déclenchement de cette procédure étant piloté par une stagnation ou des oscillations de décroissance du critère d'arrêt. Ce type de critère pilotable n'a pas été implanté pour l'instant dans Code_Aster.

•Tant qu'il n'y a pas d'ambivalence, on simplifie les notations en notant dorénavant λ^n *au lieu de* $\tilde{\lambda}^n$ *la partie homogène du vecteur de Lagrange (4.3-4) qui est la solution recherchée du problème d'interface (4.3-7). A convergence, il faut penser à reconstruire le véritable Lagrange solution en lui rajoutant sa composante initiale vérifiant la contrainte d'admissibilité.*

Mécaniquement, la méthode FETI peut être interprétée comme suit. Chaque itération démarre avec une étape de projection qui évalue la position globale des sous-domaines flottants en analysant leurs modes de corps rigides. Ensuite, les inter-efforts sont évalués à l'interface *via* le produit matrice-vecteur (1) qui s'écrit sous forme distribuée

$$\mathbf{F_I p}_\Gamma^n := \sum_{i=1}^{P} \mathbf{R}_i \left(\mathbf{K}_i\right)^+ \underbrace{\left(\mathbf{R}_i\right)^T \mathbf{p}_\Gamma^n}_{\mathbf{p}_\Gamma^{n,i}} \tag{4.4-1}$$

D'où la nécessité d'estimer, à chaque itération n, pour chaque sous-structure i et éventuellement concurremment, le produit $\left(K_i\right)^+ \left(R_i\right)^T p_\Gamma^n$ *via* le problème de Neumann habituel

$$\begin{bmatrix} K_{ii} & K_{i\Gamma} \\ K_{\Gamma i} & K^{i}_{\Gamma\Gamma} \end{bmatrix} \begin{Bmatrix} \bar{p}^{n,i}_{i} \\ \bar{p}^{n,i}_{\Gamma} \end{Bmatrix} = \begin{Bmatrix} 0 \\ p^{n,i}_{\Gamma} \end{Bmatrix} \tag{4.4-2}$$

où l'on fournit les inter-efforts d'interface, $p^{n,i}_{\Gamma}$, restriction du vecteur p^{n}_{Γ} aux degrés de liberté d'interface de la $i^{ème}$ sous-structure, pour en déduire le saut de déplacement local à cette interface. Leur sommation permet d'évaluer le produit recherché

$$F_I\, p^{n}_{\Gamma} := \sum_{i=1}^{P} R_i \begin{bmatrix} \bar{p}^{n,i}_{i} \\ \bar{p}^{n,i}_{\Gamma} \end{bmatrix} = \sum_{i=1}^{P} \bar{p}^{n,i}_{\Gamma} \tag{4.4-3}$$

Si ce saut de déplacement à l'interface est nul la convergence est atteinte, sinon, on met à jour le vecteur d'inter-efforts à l'interface p^{n+1}_{Γ} avant de réitérer le processus.

Remarques:

•Contrairement au méthode de Schur primale (sans préconditionneur avec espace grossier), FETI propage l'erreur globalement à chaque itération du GCP. Le recourt à la projection peut être vu comme un moyen élégant d'empêcher un sous-domaine flottant de déterminer ses modes de corps rigides indépendamment des contributions des autres sous-domaines.

•La propagation de l'erreur s'achève via un petit problème grossier représenté par $(\boldsymbol{G_I})^T \boldsymbol{G_I}$ qui intervient dans la phase de projection (5). Ainsi, lorsque le nombre de sous-domaines augmente (à taille de maille fixée), le niveau de propagation d'informations inter-domaines fait de même, ce qui laisse bien présager la scalabilité numérique de la méthode.

4.4.2 Critère d'arrêt

Le critère d'arrêt de la méthode repose sur la constatation, qu'à chaque itération, les champs de déplacements locaux et les multiplicateurs de Lagrange vérifient la relation

$$u^{n}_{i} = \left(K_i\right)^{+} \left\{ f_i - \sum_{k=1}^{k_i} \left(R^{i}_{k}\right)^T \lambda^n \right\} + B_i \alpha^{n}_{i} \tag{4.4-4}$$

d'où

$$\sum_{i=1}^{P} \mathbf{R}_i\, \mathbf{u}^{n}_{i} = \underbrace{\mathbf{P}\left(\mathbf{d} - \mathbf{F}_I\, \boldsymbol{\lambda}^n\right)}_{\mathbf{g}^{n}_{\Gamma}} \tag{4.4-5}$$

En d'autre terme, le saut de déplacement à travers l'interface est égal au résidu projeté du GCP. En surveillant ce résidu on peut donc arrêter l'algorithme sur un critère mécanique.

4.4.3 Philosophie du découpage des solutions

Le découpage (4.2-14) des solutions de chaque sous-domaine est dicté par la présence de modes de corps rigides dans des sous-structures flottantes. Sa première composante

$$\left(K_i\right)^{+} \left\{ f_i - \sum_{k=1}^{k_i} \left(R^{i}_{k}\right)^T \lambda \right\} \tag{4.4-6}$$

prend en compte la partie non-rigide qui peut être traitée sans encombre par le GCP, tandis que la seconde $B_i \alpha^{n}_{i}$ (très basse fréquence) regroupe lesdits modes de corps rigides pour lesquels les méthodes de Krylov éprouvent des difficultés.

C'est pour cette raison qu'un projecteur est mis en place, pour éliminer directement ces «trouble fêtes». Si on résume, la philosophie générale qui préside à cette décomposition est donc: la solution par sous-domaine comporte deux types de composantes, une «sympathique» et l'autre «embêtante». On va donc construire un opérateur de projection *ad hoc* pour se débarrasser, autant que faire ce peut, de la seconde catégorie.

4.4.4 Dualité avec la méthode de Schur primale

On montre que l'opérateur FETI peut se reformuler comme suit

$$F_I := \sum_{i=1}^{P} R_i \left(K_i\right)^+ \left(R_i\right)^T = \sum_{i=1}^{P} R_i \begin{bmatrix} 0_{ii} & 0_{i\Gamma} \\ 0_{\Gamma i} & \left(S^i\right)^+ \end{bmatrix} \left(R_i\right)^T \quad (4.4\text{-}7)$$

où $\left(S^i\right)^+$ est l'inverse généralisé de la $i^{ème}$ composante du complément de Schur. Ce qui illustre clairement la dualité entre FETI et les méthodes de Schur primale.

Remarque:

- *Sans point de jonction, on retrouve même le préconditionnement Neumann-Neumann des méthodes primales (cf. §3.2) généralisé aux sous-domaines flottants*

$$F_I = \sum_{i=1}^{P} \left(S_i\right)^+ = \left(M_{NN}^{-1}\right)_{\text{flottant}} \quad (4.4\text{-}8)$$

4.4.5 Convergence

On a vu que FETI-1 pouvait s'interpréter comme un GC sur l'opérateur projeté $P^T F_I P$. On a alors le résultat de convergence suivant (avec $\|\|_{F_I}$ la norme matricielle associée à F_I, appelée aussi norme en énergie dans l'espace des multiplicateurs de Lagrange)

$$\|\tilde{\lambda}^n - \tilde{\lambda}\|_{F_I} \leq 2\left(\frac{\sqrt{\eta\left(P^T F_I P\right)}-1}{\sqrt{\eta\left(P^T F_I P\right)}+1}\right)^n \|\tilde{\lambda}^0 - \tilde{\lambda}\|_{F_I} \quad (4.4\text{-}9)$$

On montre que dans certains cas de figure et notamment pour des problèmes élastiques, le conditionnement de FETI est régi par

$$\eta\left(P^T F_I P\right) = \vartheta\left(\frac{H}{h}\right) \quad (4.4\text{-}10)$$

Cette méthode DD est donc numériquement scalable car son conditionnement théorique diminue avec le nombre de sous-domaines. Par contre, elle n'est pas optimale. Si on raffine localement le maillage, le conditionnement n'est pas détérioré sauf si on augmente parallèlement le nombre de sous-domaines.
Cette propriété très encourageante pour traiter des calculs massivement parallèles n'est pas présente en méthode itérative de Schur primale non préconditionnée (cf. (3.2.6)). En général, le conditionnement des autres méthodes DD non préconditionnées augmente avec le nombre de sous-domaines. Il faut leur adjoindre des préconditionneurs sophistiqués, du type «Neumann-Neumann avec espace grossier», «balancing» ou «wire-basket», pour retrouver une telle propriété.

4.4.6 Modes de corps rigides et pseudo-inverses

On a vu que le point d'orgue de FETI est la construction des pseudo-inverses des sous-structures flottantes et des modes de corps rigides associés. De leur bonne détermination dépend la validité du processus, puisqu'ils rentrent à la fois dans la construction de l'opérateur FETI et dans celle du projecteur.

D'un point de vue mathématique, différentes techniques de résolution de systèmes singuliers existent (décomposition SVD, Q-méthode, équations normales…). Le dynamicien des structures a développé parallèlement des techniques particulières (cf. détection automatique, définition explicite ou approchée…) pour s'affranchir de cette difficulté. Les modes de corps rigides lui sont en effet précieux pour effectuer l'analyse modale de structures flottantes (avions, navires…) ou dans des conditions d'essais très particulières (suspensions). On peut ainsi déterminer les caractéristiques modales intrinsèques de ces structures non perturbées par la modélisation de conditions limites délicates. Ils servent aussi pour certaines méthodes de synthèse modale (Mac Neal/Craig-Bampton [R4.06.02/03]) ou d'analyse modale (Lanczos avec prétraitement «modes de corps rigides» [R5.01.01] §5.5.4).

On ne s'intéressera ici qu'à une méthode de détection automatique qui ne nécessite aucune information supplémentaire de la part de l'utilisateur mais qui, *a contrario*, est aveugle aux caractéristiques mécaniques du problème: nombre réel de modes de corps rigides, liaisons particulières, Dirichlet généralisé sur ces modes… Elle ne peut se fier qu'au modèle numérique final qui amalgame bien des imperfections: modes de corps

rigides, degrés de liberté non alimentés en raideur, modes cinématiques parasites, modes rigides internes à un mécanisme, hétérogénéité des coefficients matériels et/ou des tailles de mailles etc.

Supposons tout d'abord que l'on ait pu renuméroter la matrice de raideur du $i^{ème}$ sous-domaine sous la forme

$$K_i := \begin{bmatrix} K^i_{pp} & K^i_{pr} \\ K^i_{rp} & K^i_{rr} \end{bmatrix} \tag{4.4-11}$$

où l'indice p signifie «principal», c'est-à-dire suffisant pour que le système mécanique soit iso-statique, et r «redondant», les termes engendrés par les modes de corps rigides. Ces fameux modes que l'on recherche et qui rendent $\boldsymbol{K}_i$ singulière. Par ce subterfuge de renumérotation apparaît ainsi une sous-matrice K^i_{pp} régulière d'ordre $n^i_i + n^i_\Gamma - n^i_r$ et un bloc K^i_{rr} de taille n^i_r, le nombre de modes de corps rigides recherchés.

Un bon candidat pour l'ensemble des modes de corps rigides est

$$B_i := \begin{bmatrix} u^1_p & \dots & u^{n^i_r}_p \\ u^1_r & \dots & u^{n^i_r}_r \end{bmatrix} = \begin{bmatrix} -\left(K^i_{pp}\right)^{-1} K^i_{pr} \\ I_{rr} \end{bmatrix} \tag{4.4-12}$$

car il vérifie la relation

$$K_i B_i = 0 \tag{4.4-13}$$

Ils sont donc solution des n^i_r systèmes linéaires

$$\begin{bmatrix} K^i_{pp} & 0_{pr} \\ 0_{rp} & I_{rr} \end{bmatrix} \begin{Bmatrix} u^j_p \\ u^j_r \end{Bmatrix} = \begin{bmatrix} -K^i_{pr} \\ I_{rr} \end{bmatrix}_j \quad \left(j = 1 \dots n^i_r\right) \tag{4.4-14}$$

où l'on note $\mathbf{0}_{pr}$ (resp. bold $\mathbf{0}_{rp}$) la matrice nulle de taille $\left(n^i_i + n^i_\Gamma - n^i_r\right) \times n^i_r$ (resp. $n^i_r \times \left(n^i_i + n^i_\Gamma - n^i_r\right)$), $\boldsymbol{I}_{rr}$ la matrice identité d'ordre n^i_r et $[]_j$ pour la $j^{ème}$ colonne de la matrice.

Tout le problème est donc de trouver la renumérotation adéquate permettant de formuler ces systèmes linéaires qu'il suffira par la suite d'inverser. Heureusement, les méthodes de factorisation de type Gauss permettent, en se référant à la valeur absolue des pivots, de sélectionner et trier les composantes redondantes qui nous intéressent.
On va donc automatiquement détecter les colonnes correspondant à des pivots très petits, annuler les lignes (pour constituer le bloc $\mathbf{0}_{rp}$) et les colonnes correspondantes (resp. $\mathbf{0}_{pr}$), fixer à l'unité la valeur de ces pivots (resp. $\boldsymbol{I}_{rr}$), constituer un vecteur second membre, initialisé à zéro, dont on fixera à l'unité les composantes associées aux pivots et où l'on recopiera l'opposé des colonnes précédemment omises (resp. $-K^i_{pr}$).

La factorisée de la matrice (4.4-14) sert aussi à déterminer la pseudo-inverse de $\boldsymbol{K}_i$, car on montre qu'elle s'écrit

$$\left(K_i\right)^+ := \begin{bmatrix} \left(K^i_{pp}\right)^{-1} & 0_{pr} \\ 0_{rp} & 0_{rr} \end{bmatrix} \tag{4.4-15}$$

vérifiant ainsi les relations de Moore-Penrose

$$\begin{aligned} K_i \left(K_i\right)^+ K_i &= K_i & \left(K_i \left(K_i\right)^+\right)^T &= K_i \left(K_i\right)^+ \\ \left(K_i\right)^+ K_i \left(K_i\right)^+ &= \left(K_i\right)^+ & \left(\left(K_i\right)^+ K_i\right)^T &= \left(K_i\right)^+ K_i \end{aligned} \tag{4.4-16}$$

Dans l'algorithme de FETI, on s'intéresse en fait juste à son action sur un vecteur quelconque $v^i := \left\{ v^i_p \quad v^i_r \right\}^T$

$$\left(K_i\right)^+ \begin{Bmatrix} v^i_p \\ v^i_r \end{Bmatrix} := \begin{Bmatrix} \left(K^i_{pp}\right)^{-1} v^i_p \\ 0_{rr} \end{Bmatrix} \tag{4.4-17}$$

que l'on obtiendra en multipliant la factorisée résultant du processus précédent par les composantes de ce vecteur. Multiplication particulière qui prendra soin d'omettre les DDLs correspondants aux pivots ($I_{rr} \times v_r^i$).

Remarque:

•En arithmétique finie un réel est rarement identiquement nul, il faut donc déterminer un seuil relatif, et non absolu, en deçà duquel la valeur absolue du pivot sera considérée comme nulle. Ce choix est d'ailleurs crucial pour le bon déroulement de FETI car il permet de bien distinguer les modes de corps rigides des autres sources potentielles de pivots nuls. Parmi tous les choix possibles on retient souvent le test sur le rapport des termes diagonaux (cf. Manuel PERMAS, MSC NASTRAN)

$$\frac{\left(D_i\right)_{kk}}{\left(K_i\right)_{kk}} \leq C \text{ prec\_machine} \quad (4.4\text{-}18)$$

où D_i représente la matrice diagonale de la factorisée LDL^T de K_i , C le paramètre utilisateur d'arrêt (typiquement 10^{-6}*) et prec_machine la précision machine. Dans Code_Aster, on utilise ce critère avec le paramètre $C=10^{-n}$ fixé par l'utilisateur via le paramètre $NPREC=n$ du mot-clé* `SOLVEUR`*. Par défaut il est renseigné à 8. On traque donc les pertes de plus de la moitié des décimales dans les ordres de grandeurs d'un terme diagonal.*

Remarquons, pour finir, que cette approche pour calculer les modes de corps rigides et les pseudo-inverses associées n'implique pas de gros surcoûts et qu'une collaboration est en cours avec l'équipe de développement du solveur MUMPS pour affiner cette recherche (*via* l'ANR SOLSTICE[Boi7c]).

4.4.7 Analyse spectrale et réorthogonalisation

Lors d'un calcul, on est d'abord intéressé par les grandes tendances de la solution liées aux propriétés intrinsèques du problème continu, par opposition à ses détails, qui eux fluctuent avec les paramètres de discrétisation (finesse du maillage, pas de temps…). Numériquement, l'information des premières est contenue dans les parties proches de zéro du spectre de $\boldsymbol{K}$ alors que les détails sont liés à sa partie haute. Or les solveurs itératifs de type Krylov débusquent prioritairement les extrémités du spectre. Un Schur itératif primal ou un GC sur le problème global, exhument donc lors des premières itérations, les détails de la solution. Puis son résidu plonge avec la capture de la structure principale de la solution. On ne peut donc interrompre le processus «en cours de route» et on a tendance à stocker toute l'information, en particulier toutes les directions de descentes $\left(p^n\right)_n$ si on veut le synthétiser.

Alors qu'en dual, c'est exactement l'inverse ! FETI débusque prioritairement ses valeurs propres hautes mais celles-ci correspondent à la partie basse du spectre du problème primal. D'où de meilleures propriétés spectrales et des facilités algorithmiques. En particulier pour les procédures de réorthogonalisation et de rédémarrage du GC qui sont plus faciles à mettre en œuvre et moins coûteuses que si on appliquait un GC sur le problème global ou si on utilisait un Schur itératif primal:

•Les vecteurs directions de descentes $\left(p_\Gamma^n\right)_n$ sont dimensionnés à la longueur de l'interface au lieu de la taille du problème global,
•L'information principale est contenue dans les premiers vecteurs.

En calcul de structure, le problème d'interface a des composantes spectrales bien sériées : un paquet de valeurs propres de grande amplitude étant bien séparé des plus petites. D'autre part, l'opérateur FETI $\boldsymbol{F_I}$ étant compact, son spectre comporte un point d'accumulation près de zéro (Th. de Riesz-Fredholm) et sa partie haute est moins fournie que sa partie basse. Donc rapidement, dès que la partie haute du spectre est capturée, le conditionnement effectif du problème d'interface s'améliore, d'où des propriétés de superconvergence de l'algorithme.

Cependant en arithmétique finie cette bonne séparation spectrale conduit à une perte d'orthogonalité des directions de descente, et l'on sait l'importance cruciale que revêt cette propriété pour les méthodes de type Krylov: la convergence du GC s'en trouve amoindrie. Pour y remédier, on impose donc lors de la construction de la nouvelle direction de descente (cf. algorithme n°4.4-1, étape (8)), une phase de réorthogonalisation explicite des directions de descente entre elles.

Cette pratique très répandue en analyse modale (cf. [R5.05.01] §5.3.1 et annexe 2) peut se décliner sous différentes variantes: réorthogonalisation totale, partielle, sélective … *via* toute une panoplie de procédures

d'orthogonalisation: GS, GSM, IGSM, Househölder, Givens… Pour les solveurs itératifs standards cette technique est souvent trop dispendieuse en mémoire et en CPU. Mais pour FETI on a vu qu'elle restait très abordable car on ne stocke que les premiers vecteurs et leur taille est dimensionnée à l'interface.

A chaque itération k , on ne se fie pas à la propagation théorique de l'orthogonalité des directions de descente et on l'impose *via* un algorithme de type Gram-Schmidt (GS). Ces réorthogonalisations systématiques requièrent le stockage des N_{orth} premiers vecteurs $p_\Gamma^n \ \left(n=1\dots N_{orth}\right)$ et de leur produit par l'opérateur d'interface $z_\Gamma^n = F_I\, p_\Gamma^n \ \left(n=1\dots N_{orth}\right)$. Les deux dernières étapes de l'algorithmes 4.4-1 se réécrivent donc

$$p_\Gamma^{n+1} = g_\Gamma^{n+1} - \sum_{i=0}^{N_{orth}} \frac{\langle g_\Gamma^{n+1}, F_I\, p_\Gamma^i \rangle}{\langle p_\Gamma^i, F_I\, p_\Gamma^i \rangle}\ p_\Gamma^i \quad (\text{nouvelle dd } F_I - \text{orthogonalisée}) \qquad (4.4\text{-}19)$$

Comme le problème d'interface est modeste, le surcoût mémoire est négligeable. Tout comme le surcoût calcul si on prend bien soin de choisir un algorithme de réorthogonalisation fiable (orthogonalité à la précision machine près), robuste et efficace.

Remarque:

- *Dans l'algorithme FETI implanté dans Code_Aster, l'utilisateur a accès à différents paramètres des procédures de réorthogonalisation : le type de méthode (mot-clé* `TYPE_REORTHO_DD`*; pas de réorthogonalisation, GS, IGS et IGSM) et le nombre de premières dd stockées (* `NB_REORTHO_DD` *). Des valeurs par défaut ou pré-calculées en fonction des autres données sont proposées.*

4.5 Préconditionnement du problème d'interface

4.5.1 Gradient conjugué projeté préconditionné

Comme pour la méthode itérative de Schur primale, on va se focaliser uniquement sur les préconditionneurs matriciels découpés et SPD, $\boldsymbol{M}$, permettant de diminuer le conditionnement de l'opérateur intervenant dans le GCP de l'algorithme 4.4-1. Car même si le conditionnement intrinsèque de l'opérateur FETI semble meilleur que celui du complément de Schur, un préconditionneur efficace et peu coûteux est toujours le bienvenu pour accélérer la convergence. Surtout en présence d'hétérogénéités et de points de jonction. On parlera alors de gradient conjugué projeté préconditionné[39] (GCPPC).

Son déroulement complété par la procédure de réorthogonalisation partielle préconisée précédemment prend alors la forme suivante :

39 En anglais 'Preconditioned Conjugate Projected Gradient' (PCPG).

Initialisation λ^0 donné (4.3-3), $r_\Gamma^0 = F_I \lambda^0 - d$, $g_\Gamma^0 = \mathbf{Pr}_\Gamma^0$, $h_\Gamma^0 = \mathbf{PM^{-1}} g_\Gamma^0$
$p_\Gamma^0 = g_\Gamma^0$
Boucle en n
(1) $z_\Gamma^n = F_I p_\Gamma^n$
(2) $\alpha^n = \dfrac{\langle g_\Gamma^n, p_\Gamma^n \rangle}{\langle z_\Gamma^n, p_\Gamma^n \rangle}$
(3) $\lambda^{n+1} = \lambda^n + \alpha^n p_\Gamma^n$ (nouvel itéré)
(4) $r_\Gamma^{n+1} = r_\Gamma^n - \alpha^n z_\Gamma^n$ (nouveau résidu)
(5) $g_\Gamma^{n+1} = \mathbf{Pr}_\Gamma^{\mathbf{n+1}}$ (projection 1)
(6) Test d'arrêt **via** $\|g_\Gamma^{n+1}\| < \varepsilon \|g_\Gamma^0\|$
(7) $\tilde{h}_\Gamma^{n+1} = M^{-1} g_\Gamma^{n+1}$ (préconditionnement)
(8) $h_\Gamma^{n+1} = P \tilde{h}_\Gamma^{n+1}$ (projection 2)
(9) $p_\Gamma^{n+1} = h_\Gamma^{n+1} - \sum_{i=0}^{N_{orth}} \dfrac{\langle h_\Gamma^{n+1}, F_I p_\Gamma^i \rangle}{\langle p_\Gamma^i, F_I p_\Gamma^i \rangle} p_\Gamma^i$ (nouvelle dd orthogonalisée)

Algorithme 4.5-1._ FETI-1 version intermédiaire.

On remarque la double projection des étapes (5)(8) et la réorthogonalisation partielle de l'étape (9). Dans la littérature on rencontre principalement deux types de préconditionneurs adaptés pour FETI : le préconditionneur de Dirichlet et celui dit «lumped».

4.5.2 Préconditionneur de Dirichlet

Ce préconditionneur est basé sur l'interprétation mécanique de FETI qui consiste, à chaque itération n et pour chaque sous-domaine i, à prescrire des inter-efforts à l'interface et à en déduire le saut de déplacement d'interface résultant (*via* le problème de Neumann). Donc on construit le problème inverse en imposant un saut de déplacement à l'interface et en calculant les inter-efforts associés. La démarche étant complètement « duale » de celle employée par le préconditionnement dit de «Neumann-Neumann» pour la méthode itérative de Schur primale, il n'est pas étonnant d'aboutir à un préconditionneur du type

$$M_D^{-1} := \sum_{i=1}^{P} R_i \begin{bmatrix} 0_{ii} & 0_{i\Gamma} \\ 0_{\Gamma i} & S^i \end{bmatrix} \left(R_i\right)^T \qquad (4.5\text{-}1)$$

où l'on reconnaît les composantes locales du complément de Schur.
Muni de ce préconditionneur, l'algorithme FETI devient une méthode DD non seulement numériquement scalable mais aussi optimale, puisque son conditionnement répond asymptotiquement à la loi

$$\eta\left(M_D^{-1}\right) = \vartheta\left(1 + \log^2\left(\frac{H}{h}\right)\right) \qquad (4.5\text{-}2)$$

Le préconditionnement apporte bien une réelle plus-value, car ce résultat est bien plus intéressant que celui de FETI non préconditionné (4.4-10). Si la parallélisation de la méthode est bien menée, ce résultat lui confère des bonnes propriétés de scalabilité forte et faible. Ce résultat numérique assure donc que de bonnes progressions du speed-up et du scale-up sont atteignables (§2.4).

Cependant la construction du complément de Schur peut s'avérer coûteuse en capacité mémoire et en temps calcul ce qui ampute d'autant les gains de convergence escomptés du GCPPC. Pour cette raison on lui préfère souvent un préconditionneur plus fruste, plus allégé dit « lumpé » ou lumped.

Remarque:

- *Ce préconditionneur de Dirichlet pour FETI présente les mêmes problèmes de stockage que celui de Neumann-Neumann pour la sous-structuration primale: même si certains blocs matriciels indispensables au préconditionneur sont déjà présents dans l'opérateur FETI, on est obligé de les*

stocker et de les factoriser deux fois pour des contingences de renumérotation. De plus, il requiert une descente-remontée sur les degrés de liberté internes qui double la complexité calcul du GCP.

4.5.3 Préconditionneur lumped

On le prénomme de cette façon car, d'un point de vue mécanique, il correspond à la fourniture de tractions d'interfaces grossières («lumped») permettant de reproduire des déplacements bords quand seuls les degrés de liberté d'interface sont autorisés à bouger («peau» liée à un massif rigide)

$$M_L^{-1} := \sum_{i=1}^{P} R_i \begin{bmatrix} 0_{ii} & 0_{i\Gamma} \\ 0_{\Gamma i} & K_{\Gamma\Gamma}^i \end{bmatrix} \left(R_i\right)^T \tag{4.5-3}$$

Il ne conduit pas à une méthode DD optimale mais il ne requiert aucun stockage supplémentaire et n'a recours qu'à des opérations simples facilement parallélisables: produit matrice-vecteur, restriction-extrapolation. Encore plus allégé est le préconditionneur «superlumped» qui, non seulement néglige l'influence des points intérieurs du sous-domaine dans le comportement de l'interface, mais en plus considère chaque degré de liberté de l'interface comme indépendant (ressort lié à un bâti rigide)

$$M_{SL}^{-1} := \sum_{i=1}^{P} R_i \begin{bmatrix} 0_{ii} & 0_{i\Gamma} \\ 0_{\Gamma i} & \mathrm{diag}\left(K_{\Gamma\Gamma}^i\right) \end{bmatrix} \left(R_i\right)^T \tag{4.5-4}$$

Son efficacité semble cependant limité au faible partitionnement (quelque sous-domaines) de problèmes relativement homogènes.

Remarque:

- *C'est le préconditionneur «économique» lumped qui est pour l'instant utilisable dans Code_Aster (mot-clé `PRE_COND`). Même si il semble qu'il ne soit véritablement comparable au Dirichlet que jusqu'à une vingtaine de sous-domaines. Au delà, ce dernier est conseillé pour assurer un meilleur compromis convergence/coût calcul.*

4.5.4 Points de jonction et d'hétérogénéités

Comme pour les méthodes itératives primales, la présence de points de jonction détériore notablement l'efficacité des préconditionneurs FETI. Le phénomène s'amplifie lorsque des propriétés matériaux ou des tailles de mailles très différentes conduisent à des coefficients de raideur très hétérogènes aux degrés de liberté d'interface.
Pour y pallier on enrichit les préconditionneurs précédents de matrices effectuant une mise à l'échelle de leurs coefficients ('scaling') en tenant compte:
•de la multiplicité des points de jonction,
•des coefficients de raideur variés qui s'y confrontent.

D'où les préconditionneurs «équilibrés» par des matrices de pondération

$$M_{DS}^{-1} := \sum_{i=1}^{P} A_i R_i \begin{bmatrix} 0_{ii} & 0_{i\Gamma} \\ 0_{\Gamma i} & S^i \end{bmatrix} \left(R_i\right)^T A_i \tag{4.5-5}$$

$$M_{LS}^{-1} := \sum_{i=1}^{P} A_i R_i \begin{bmatrix} 0_{ii} & 0_{i\Gamma} \\ 0_{\Gamma i} & K_{\Gamma\Gamma}^i \end{bmatrix} \left(R_i\right)^T A_i \tag{4.5-6}$$

avec $\mathbf{A}_i$ la matrice diagonale dont les termes sont, soit la multiplicité des degrés de liberté de l'interface concernée (pour tenir compte de (1)), soit le barycentre des raideurs à ces mêmes degrés de liberté (pour tenir compte, à la fois, de (1) et (2)). Notons que pour certaines combinaisons d'hétérogénéités et de découpages en sous-domaine et pour certains problèmes multiphysique une telle approche se révèle insuffisante[Gos03].

Remarque:

- *Dans Code_Aster, cet enrichissement du préconditionneur est pour l'instant basé sur les multiplicités géométriques des points d'interface (mot-clé `SCALING`). Toutefois, on a remarqué que dans certains cas, sans qu'on puisse se l'expliquer, il pouvait être contre-productif !*

4.5.5 Algorithme FETI-1 complet

En tenant compte de cette phase de pondération et en enrichissant l'algorithme de la reconstruction du champ de déplacement global qui seul intéresse l'utilisateur, on obtient :

Initialisation λ^0 donné (4.3-3), $r_\Gamma^0 = F_I \lambda^0 - d$, $g_\Gamma^0 = \mathbf{Pr}_\Gamma^0$, $h_\Gamma^0 = \mathbf{PAM^{-1}Ag}_\Gamma^0$

$p_\Gamma^0 = g_\Gamma^0$

Boucle en n

(1) $z_\Gamma^n = F_I p_\Gamma^n$

(2) $\alpha^n = \dfrac{\langle g_\Gamma^n, p_\Gamma^n \rangle}{\langle z_\Gamma^n, p_\Gamma^n \rangle}$

(3) $\lambda^{n+1} = \lambda^n + \alpha^n p_\Gamma^n$ (nouvel itéré)

(4) $r_\Gamma^{n+1} = r_\Gamma^n - \alpha^n z_\Gamma^n$ (nouveau résidu)

(5) $g_\Gamma^{n+1} = \mathbf{Pr}_\Gamma^{n+1}$ (projection 1)

(6) Test d'arrêt **via** $\|g_\Gamma^{n+1}\| < \varepsilon \|g_\Gamma^0\|$

(7) $\bar{g}_\Gamma^{n+1} = \mathbf{Ag}_\Gamma^{n+1}$ (scale 1)

(8) $\tilde{g}_\Gamma^{n+1} = M^{-1} \bar{g}_\Gamma^{n+1}$ (préconditionnement)

(9) $\tilde{h}_\Gamma^{n+1} = A \tilde{g}_\Gamma^{n+1}$ (scale 2)

(10) $h_\Gamma^{n+1} = P \tilde{h}_\Gamma^{n+1}$ (projection 2)

(11) $p_\Gamma^{n+1} = h_\Gamma^{n+1} - \sum_{i=0}^{N_{orth}} \dfrac{\langle h_\Gamma^{n+1}, F_I p_\Gamma^i \rangle}{\langle p_\Gamma^i, F_I p_\Gamma^i \rangle} p_\Gamma^i$ (nouvelle dd orthogonalisée)

Finalisation

$= ¿ \lambda^0 + \mathbf{P}\boldsymbol{\lambda}^n$, $r_\Gamma^{sol} = F_I \lambda^{sol} - d$, $\alpha^{sol} = \left(\left(G_I \right)^T G_I \right)^{-1} \left(G_I \right)^T r_\Gamma^{sol}$

λ^{sol} ¿ u_i^{sol} **via** (4.2−14) puis u^{sol}

Algorithme 4.5-2._ FETI-1 complet.

Nous allons maintenant aborder les grands principes qui ont guidé l'implémentation séquentielle et parallèle de l'algorithme dans le code.

5 Méthode FETI: parallélisme et compléments

5.1 Parallélisation de la méthode

5.1.1 Principe

L'efficacité de la DD en mécanique des structures provient en grande partie du fait qu'elle permet de propager le parallélisme bien en amont du solveur linéaire. D'où la dénomination parfois usitée de «parallélisme mécanique» par référence aux parallélismes informatique et numérique déjà évoqués (cf. §2.5/2.6).
De nombreuses stratégies sont envisageables pour mettre en œuvre l'organisation des flots de données et d'instructions qui sous-tendent ce parallélisme. Nous détaillerons plus particulièrement l'implantation actuelle de FETI-1 dans *Code_Aster*. Mais tout d'abord quelques mots sur le modèle de parallélisme qui a été retenu pour cette implantation. Compte-tenu de tout ce que l'on a déjà écrit sur le sujet dans les paragraphes précédents, c'est tout naturellement celui des communications *via* MPI qui va s'imposer. De part sa proximité conceptuelle avec la philosophie de la DD. Cependant l'utilisateur doit pouvoir utiliser cet algorithme en parallèle comme en séquentiel et, dans ce dernier cas de figure, sans être obligé d'installer sur sa machine d'exécution la librairie MPI *ad hoc*. Il faut donc adopter une architecture de calcul et de communication et une organisation du modèle de données qui ne perturbent pas celles de l'existant, tout en permettant la cohabitation des ces différentes versions au sein d'un même code source.
Au lancement du calcul, chaque processeur va lire le maillage, affecter un modèle aux mailles retenues, construire les chargements globaux indiqués dans le fichier de commande, bref effectuer tous les pré-traitements globaux requis par le calcul mécanique (calcul thermique…). Cette partie séquentielle pourrait d'ailleurs bénéficier, à terme, du schéma parallèle remontant du solveur linéaire.
Puis chaque processeur va partitionner le maillage en plusieurs sous-domaines (*via* les opérateurs DEFI_PART_FETI/OPS[U4.23.05] répertoriés comme autant de groupes de mailles GROUP_MA, cf. §6.4). En même temps, certains chargements[40] associés à des entités géométriques dites tardives[41] vont être projetés sous-domaine par sous-domaine. A gros traits, chaque processeur n'est donc appelé à manipuler et à stocker que les données liées aux sous-domaines dont il a la charge. Cela permet de limiter les occupations mémoires et les coûts calculs, empiriquement de les équilibrer et d'éviter les redondances de données et d'instructions : c'est la phase de partitionnement de la méthode de Foster (cf. §2.2).
Puis chaque processeur calcule les termes élémentaires des mailles de son périmètre, les assemble en autant de matrices de rigidité locales $\boldsymbol{K}_i$ et de seconds membres locaux $\boldsymbol{f}_i$. Puis suivent les phases de factorisations symboliques/numériques et les détections conjointes de modes rigides (cf. §4.4) pour constituer localement les $(\boldsymbol{Ki})^+$ et les $\boldsymbol{B}_i$.
Ensuite au moins deux approches sont possibles: «maître-esclave» et «esclave-esclave». Dans la première, le processeur maître joue un rôle particulier en centralisant les contributions des autres processeurs après les traitements distribués (produit matrice-vecteur (1) et préconditionnement (7)(8)(9)) de loin les plus gourmands en ressources. Il gère seul les tâches mineures d'ordonnancement de l'algorithme (produits scalaires (2), test d'arrêt (6), daxpy (3)(4)). C'est le choix sous-optimal mais plus souple et plus simple à mettre en œuvre qui a été retenu pour l'implantation de FETI dans *Code_Aster*.
Dans la seconde approche, chaque processeur ne communique que les données d'interface aux processeurs «voisins» concernés (ceux dont les sous-domaines partagent une interface avec le périmètre du processeur donné) et conduit lui-même les portions d'opérations mineures liées à son périmètre. Après chacune d'elles, une communication globale rassemble l'information et construit la donnée pertinente; produit scalaire, norme…

Les phases de projection (étapes (5) et (10)) et de réorthogonalisation (étape (11)) jouissent d'un statut particulier car elles manipulent des grandeurs globales communes à tous les sous-domaines: l'ensemble des restrictions des modes rigides à l'interface pour l'une, les directions de descentes pour l'autre. Certains auteurs proposent des algorithmes pour les résoudre en parallèle, mais ce scénario n'a pas été retenu dans *Code_Aster*. Pour l'instant, les données *ad hoc* sont cooptées et orchestrées par le processeur maître qui réalise seul ces opérations et distribue, si nécessaire, le résultat. Cela limite d'autant les gains de speed-up prévus par la loi d'Amdhal mais, *a contrario*, ces opérations sont peu coûteuses tant que la taille de l'interface et

40 Blocage de Dirichlet (DDL_IMPO, FACE_IMPO etc.), force nodale (FORCE_NODALE), contact-frottement méthode continue.
41 On projette les chargements associés ou qui vont être associés (contact continue), au cours du calcul, à des mailles tardives (LIGREL tardif en langage développeur). Ces dernières étant liées (Dirichlet) ou non (force nodale/contact continue) à des nœuds tardifs. Toutes ces entités géométriques sont dites tardives car elles n'existent pas dans le maillage initial et sont rajoutées au cours du processus de modélisation pour faciliter la mise en données de l'utilisateur. Il faut donc les répartir par sous-domaine «à la volée».

le nombre de modes rigides reste faible: quelques pourcents de la taille du problème global pour l'une, quelques dizaines de modes pour l'autre.

Pour finir, chaque processeur reconstruit sa solution locale $\mathbf{u}_i^{sol}$ et sa projection dans le champ solution global $\boldsymbol{u}^{sol}$. Puis le processeur maître «ramasse les copies» en cumulant ces contributions (nonobstant les problèmes de multiplicités aux interfaces et de ddls tardifs), avant de les distribuer à tous les processeurs. Car ces derniers doivent reconstruire correctement les nouveaux systèmes linéaires ou les post-traitements qui vont suivre. En non-linéaire, il en va se même pour différents champs (stockés dans des CHAM_ELEM) notamment ceux calculés lors de l'intégration des lois de comportement. A chaque archivage, il faut les rassembler globalement et les diffuser à tous les processeurs.

Remarques:

- *En terme de stockage, le surcoût imposé au processeur maître peut par contre être moins négligeable: il faut stocker les matrices pleines* $\boldsymbol{G}_I$ *et* $\left(\left(\boldsymbol{G}_I\right)^T \boldsymbol{G}_I\right)^{-1}$ *et les directions de descentes* $p_\Gamma^n \ \left(n=1\ldots N_{orth}\right)$ *, au sein d'un même pas de temps, voire entre plusieurs pas de temps, en cas de problèmes «accélérés» du type multiples seconds membres. C'est pour cette raison qu'a été mis en place, en parallèle, le paramètre* NB_SD_PROC0 *qui permet de soulager le processeur maître (de numéro 0 par défaut) en lui attribuant moins de sous-domaines que prévu par l'ordonnancement automatique du code. Ce paramètre est le pendant parallèle du paramètre séquentiel* STOCKAGE_GI *(cf. §4.3).*
- *L'algorithme débute par une procédure automatique d'assignation des sous-domaines par processeurs (cf. méthodes de Foster § 2.2) qui essaye de soulager le processeur maître et qui attribue un nombre égal de sous-domaines par processeur. Les numéros de ces sous-domaines sont contigus par processeur et sont issus de la numérotation du partitionneur. Informatiquement, compte-tenu de la localisation de cet algorithme de tri et des structures de données qu'il gère, il serait assez aisé d'introduire d'autres stratégies de répartitions pour équilibrer la charge suivant d'autres critères.*
- *Bien sûr, suivant la configuration du calcul et le paramétrage fourni par l'utilisateur, de nombreuses structures de données sont conservées pour limiter la complexité calcul: modes rigides, matrices d'interconnexion, opérateur de reconstruction de la solution globale etc. Il faut veiller à ce que ce mode de fonctionnement ne soit pas perturbé par le parallélisme, ni n'engendre de surcoûts notables.*

5.1.2 Détails sur l'implantation dans Code_Aster

Tout d'abord, l'algorithme FETI a été codé en séquentiel puis cette implantation a été adaptée pour supporter un parallélisme par envoi de message en MPI-1 (sur machine 32 et 64-bits). En effet, la priorité était de mesurer l'impact d'un tel solveur multidomaine sur l'architecture et les structures de données du code, d'en limiter les conséquences (lisibilité, efficacité, maintenabilité) et de s'assurer de son bon fonctionnement sur des cas standards. D'ailleurs, pour de nombreux auteurs, un tel solveur se révèle souvent très efficace (en CPU et en occupation mémoire), même en séquentiel, lorsque l'on monte en degrés de liberté.
Ensuite, la stratégie de parallélisation a été la suivante:

1.La souplesse d'assignation sous-domaine/processeur est assez peu pratiquée dans les autres codes, car elle complique la programmation «esclave-esclave». Dans notre vision «maître-esclave», elle semble la plus naturelle pour faire cohabiter dans un seul code, une version purement séquentielle (sans même un link à une librairie MPI) et une parallèle. Le séquentiel devient alors un cas particulier: tous les sous-domaines se retrouvent sur le même processeur. En plus, pour opérer un équilibrage «empirique» des calculs, tous les auteurs proposent (et on l'a vérifié lors des tests) de joindre plusieurs sous-domaines par processeur.

•Jusqu'à l'opérateur appelant le solveur FETI (MECA_STATIQUE ou STAT_NON_LINE), tous les processeurs exécutent la même séquence d'opérations et connaissent donc les mêmes objets JEVEUX: maillage, matériaux, champs de prétraitements, SD_FETI… C'est relativement sous-optimal, mais les prétraitements sont souvent, comparativement au solveur, peu gourmands en CPU et en mémoire (compte tenu de la cible parallèle retenue, quelque dizaines de processeurs).

•Une fois dans l'opérateur principal, on oriente les opérations effectuées concurremment par les processeurs en tarissant le volume de données qui leur est affecté. Et ce, de la préparation des données solveur, aux factorisations symbolique et numérique, en passant par les phases d'assemblage, de calculs élémentaires et bien sûr l'algorithme de résolution proprement dit. Cela se fait très simplement, sans envoi de message particulier, *via* l'objet '&&FETI.LISTE.SD.MPI' qui filtre les boucles sur les sous-domaines:

```
CALL JEVEUO('FETI.LISTE.SD.MPI','L',ILIMPI)
DO 50 I=1,NBSD   <boucle sur les sous-domaines>
IF (ZI(ILIMPI+I).EQ.1) THEN
<on n'effectue la suite d'instructions prévues que si le
sous-domaine est contenu dans le périmètre du proc. courant>
ENDIF
50 CONTINUE
```

Concernant les gros objets JEVEUX usuels, chaque processeur ne construit que les données dont il a besoin: SD SOLVEUR maître et celles esclaves dépendant du périmètre du processeur courant et la même chose pour les NUME_DDL, les MATR_ASSE et les CHAM_NO. Par contre, les données de petit volume sont elles calculées par tous les processeurs car elles permettent souvent d'orienter le calcul et il est bien sûr important que tous les processeurs fassent le même cheminement logiciel.

•Pour que la vision multidomaine parallélisée ne casse pas la gestion des données et reste lisible, évolutive et maintenable, on encapsule les structures de données principales (matrice MATR_ASSE, vecteur CHAM_NO etc.). *A contrario*, il peut être contre-productif de vouloir dimensionner des objets annexes (CHAM_ELEM, RESU_ELEM…) suivant la taille des données qu'ils ont réellement à stocker. Cela complique les échanges de messages, pollue la programmation et fragilise des zones de codes déjà complexes. Il est plus intéressant, tous critères confondus, d'initialiser à zéro l'objet et de ne remplir que les zones qui concernent le processeur. Ce constat a été fait, par exemple, lors de la parallélisation des calculs élémentaires (où l'on tarit juste l'écriture des flux de données par un boucle conditionnelle basée sur un vecteur de booléen *ad hoc* similaire à celui de l'exemple ci-dessus) ou lors de la reconstruction du champ solution global (à convergence de FETI), où pour ne pas communiquer les «mappings» entre les NUME_DDL locaux et le global, les processeurs dialoguent dans une «base» surdimensionnée, celle du vecteur d'interface global.

•Plutôt que des boucles de communications points à points entre les processeurs esclaves et le maître (MPI_SEND/RECV), on a retenu dans une première approche des communications collectives (MPI_REDUCE…) qui encapsulent les premières et gèrent de manière transparente les problèmes de synchronisation et de buffering. Cela assure une meilleure lisibilité, maintenabilité et portabilité mais, *a contrario*, on ne peut les optimiser en chevauchant les calculs et les communications, en limitant les temps de latence ou le buffering. Pour limiter l'impact dans le code, les requêtes MPI sont centralisées dans une seule routine à laquelle on accède par compilation conditionnelle.

•En statique non linéaire, aux contingences du multiples seconds membres dont le déroulement n'est pas prévisible *a priori* et à d'autres aspects liés à la complexité et à la richesse de l'opérateur, se cumule une stratégie particulière des communications:

1.A la fin de chaque résolution de système linéaire, il faut communiquer la solution globale à tous les processeurs pour reconstruire correctement les nouveaux systèmes linéaires et certains critères.

2.A chaque estimation du résidu (critère d'arrêt de Newton, pilotage, recherche linéaire…), il faut sommer puis diffuser à tous les processeurs, 5 à 7 vecteurs (CHAM_NO) eux aussi dimensionnés à la taille du problème global.

3.A chaque pas de temps archivé, il faut sommer puis diffuser plusieurs champs globaux (CHAM_ELEM) équivalents à plusieurs fois la taille du problème. Et ce, afin de pouvoir enchaîner en parallèle plusieurs opérateurs, éventuellement pas encore parallélisés (post-traitement), sans avoir à retoucher leurs routines de démarrage. Par précaution, le cahier des charges des opérateurs parallélisés (MECA_STATIQUE et STAT_NON_LINE) est donc de produire, en fin d'opérateur, des bases de données JEVEUX globales identiques sur tous les processeurs.

Informatiquement, aucune de ces contingences n'est irrémédiable. On peut ainsi retailler certains de ces échanges pour qu'ils ne portent que sur les seules données à mettre à jour, celles des interfaces (du moins tant que le calcul reste local). On peut utiliser des techniques de «mailles fantômes», dites encore de «halo», pour

automatiser ces mises à jour. On peut aussi réfléchir à des critères globaux qui se construisent à l'aide de critères locaux. Au risque toutefois, de voir diverger les cheminements non linéaires séquentiel et parallèle du même algorithme. Dans un premier temps, tous ces scénarios nous ont semblé trop invasifs et nous leur avons préféré des communications, coûteuses mais conservatives.

Le lecteur désirant plus de détails sur l'implantation de l'algorithme et les difficultés de ce chantier logiciel pourra consulter les notes [Boi05][Boi07][Ass07] et les documentations informatiques du code [D9.03.01] et de ses structures de données [D4.06.05/07/10/11/21].

5.2 Compléments sur FETI

5.2.1 Contrôle de la convergence: résidu d'interface versus résidu global

On a vu au §4.4 qu'un critère d'arrêt de FETI pouvait être le résidu projeté du GCP. Cependant c'est un résidu homogène à un déplacement $\boldsymbol{u^n}$ et non à un effort, tel le résidu global $\boldsymbol{K\,u^n - f}$. De plus, le conditionnement de $\boldsymbol{K}$ est en $\vartheta\left(1/h^2\right)$, alors que celui de $\boldsymbol{F_I}$ varie asymptotiquement plutôt comme $\vartheta\left(H/h\right)$, donc un critère d'arrêt sur le résidu projeté

$$\left\|P\left(F_I\lambda^n - d\right)\right\| \leq \varepsilon \left\|P\left(F_I\lambda^0 - d\right)\right\| \tag{5.2-1}$$

ne pourra malheureusement pas garantir un résidu global du même ordre

$$\left\|\mathbf{K}\mathbf{u}^{\boldsymbol{n}} - f\right\| \leq \varepsilon \left\|\mathbf{K}\mathbf{u}^{\mathbf{0}} - f\right\| \tag{5.2-2}$$

Typiquement, pour beaucoup de problèmes de structures des écarts relatifs d'ordre 10^2 ou 10^3 ont été observés entre ces deux critères en défaveur du second qui est pourtant, au final, le seul «juge de paix» de l'algorithme.
On ne peut évidemment pas, à chaque itération de FETI estimer le résidu global, car cela ferait exploser les coûts calcul et les communication inter-processeurs. Des tests d'arrêts mixtes «résidu projeté/champ de déplacement sur l'interface» paraissent plus pertinents mais ils s'appuient sur le préconditionneur de Dirichlet. Ils n'ont donc pas encore été testés dans *Code_Aster*.

5.2.2 Réduction de modèle

En utilisant des bases de polynômes ou de splines, on peut discrétiser plus efficacement les multiplicateurs de Lagrange, c'est-à-dire, réduire la taille du problème d'interface (n_I). Et ce, tout en préservant la précision des calculs et en ne dégradant pas le conditionnement de $\boldsymbol{F_I}$. On affaiblit la condition de continuité des champs de déplacement aux interfaces *via* une discrétisation des multiplicateurs du type (B -spline d'ordre s pour le $m^{ème}$ degré de liberté)

$$\lambda_i^m(\xi) := c_{1\mathrm{i}}^m + c_{2\mathrm{i}}^m\left(\xi - \xi_i\right) + \ldots + c_{2\mathrm{i}}^m\left(\xi - \xi_i\right)^s \tag{5.2-3}$$

sur chaque sous-intervalle $\Gamma_I^i := \left[\xi_i, \xi_{i+1}\right]$ de l'interface. Avec des bases particulières[42] de ce type, on peut mettre au point des procédures de raffinements adaptatifs ce qui amplifie les gains potentiels (de l'ordre de 40% cf. [FR94] §5.5).

Remarques:

- *Cette technique, dite «de réduction de modèle», impose un calcul effectif des matrices d'interconnexion qui ne répondent alors plus à la définition (4.2-11). Elles doivent mixer la discrétisation éléments finis du champ de déplacement (des fonction de forme N_i) et celle des multiplicateurs, via des termes du genre*

$$\int_{\Gamma_I^i} N_i \xi^j \, d\Gamma \tag{5.2-4}$$

- *Toutefois, lorsque les maillages sont compatibles aux interfaces, il faut avoir recourt à des procédures de lissage pour post-traiter le champ de déplacement dans ces zones, ce qui alourdit et complexifie le processus.*

5.2.3 Maillages non conformes et éléments finis incompatibles

42 Notamment orthogonale pour ne pas dégrader le conditionnement de l'opérateur FETI.

La technique précédente peut être déployée pour combiner des sous-domaines maillés par des éléments incompatibles ou non conformes. C'est une voie concurrentes de celle des éléments joints (avec des ingrédients numériques assez proches) qui a été testées en statique, en modal, en analyse vibratoire et en contact-frottement.

5.2.4 Projecteurs généralisés

Pour diminuer la complexité calcul de l'algorithme, nous avons jusqu'à présent utilisé un projecteur orthogonal standard du type (4.3-5) avec $Q=I$

$$P(Q):=I-\mathbf{QG_I}\left(\left(G_I\right)^T\mathbf{QG_I}\right)^{-1}\left(G_I\right)^T \tag{5.2-5}$$

Cette stratégie est efficace pour bon nombre de calculs de structures, surtout lorsque la DD répond à des impératifs de type HPC et partitionne le domaine initial indépendamment de toutes considérations géométriques et physiques. Par contre, si la DD sert à mettre en place un travail collaboratif entre différents contractants, ce découpage a toute les chances de séparer les matériaux et de respecter les interfaces naturelles.

Dans ce dernier cas de figure, il est plus efficace de sophistiquer l'étape de projection en généralisant le projecteur (5.2-5). Tout comme pour les préconditionneurs FETI (et ce n'est pas un hasard comme on va le voir), deux candidats sont souvent proposés

$$\begin{aligned}P\left(M_L^{-1}\right):&=I-M_L^{-1}G_I\left(\left(G_I\right)^TM_L^{-1}G_I\right)^{-1}\left(G_I\right)^T\\ P\left(M_D^{-1}\right):&=I-M_D^{-1}G_I\left(\left(G_I\right)^TM_D^{-1}G_I\right)^{-1}\left(G_I\right)^T\end{aligned} \tag{5.2-6}$$

Cette projection induit un surcoût moindre que ce que l'on pourrait escompter de prime abord car elle permet de supprimer la seconde étape de projection de l'algorithme 4.5-2.

Remarque:

- *Comme pour les préconditionneurs, la version lumped semble la plus intéressante pour réaliser un bon compromis coût/efficacité.*

5.2.5 Initialisation des multiplicateurs de Lagrange

Pour certains cas pathologiques, avec de fortes distorsions de maillage et/ou des hétérogénéités matériaux, FETI-1 peut afficher une convergence plus chahutée et un résidu projeté initial important par rapport à un Schur itératif primal de type BDD. Cette dégradation est principalement due à ce résidu initial exagéré.

En proposant, récemment, une nouvelle stratégie pour décomposer les forces appliquées entre les sous-domaines, P.Gosselet et al[Gos03] ont résolu ce problème. Ils décomposent les forces condensées sur l'interface sur une diagonalisée de la matrice de raideur d'interface. Ce qui se traduit par l'application d'un des projecteurs sophistiqués évoqués ci-dessus et, surtout, par un nouveau multiplicateur de Lagrange initial. Au lieu de la formule (4.3-3), on remplace l'étape (0) des algorithmes précédents par

$$\lambda^0:=P(Q)\lambda^{00}+\mathbf{QG_I}\left(\left(G_I\right)^T\mathbf{QG_I}\right)^{-1}e \tag{5.2-7}$$

où

$$\lambda^{00}:=\left(M_L^{-1}\right)\sum_{i=1}^{P}R_i\begin{bmatrix}0_{ii} & 0_{i\Gamma}\\ 0_{\Gamma i} & K_{\Gamma\Gamma}^i\end{bmatrix}\left(f_\Gamma^i-K_{\Gamma i}K_{ii}^{-1}f_i\right) \tag{5.2-8}$$

et $P(Q)$ est un des projecteurs orthogonaux de (5.2-6).

Remarques:

- *Comme pour les préconditionneurs, il est conseillé d'équilibrer cette phase d'initialisation.*
- *C'est la seule étape de l'algorithme, si l'on utilise un préconditionnement lumped, où l'on a besoin de distinguer, comme pour les méthodes primales, les variables internes et celles d'interface.*
- *La construction de λ^{00} a exactement la même interprétation mécanique que celle du préconditionneur de Dirichlet équilibré. Son coût est identique à ce dernier, c'est-à-dire égal à la moitié celui d'une itération de FETI-1.*
- *Cette initialisation est plutôt conseillée pour un FETI préconditionné par Dirichlet, $Q=M_D^{-1}$ **ou** M_{DS}^{-1} , car la factorisée des matrices de rigidité internes est déjà disponible.*

Sachant que pour les cas non pathologiques, un lumped suffit et ce problème d'initialisation n'est souvent pas critique.

- *En posant* $\lambda^{00}=0$ *et* $\boldsymbol{Q}=\boldsymbol{I}$ *on retrouve l'initialisation standard de FETI.*

5.2.6 Multiples seconds membres

Lors d'analyses thermo-mécaniques et/ou des calculs mécaniques non linéaires, la problématique des multiples seconds membres[43] se posent souvent: réactualisation non systématique de la matrice tangente, caractéristiques matériaux ne dépendant pas de la température… On a alors à résoudre une succession de problèmes du type $\boldsymbol{K}\,\boldsymbol{u}=\boldsymbol{f}$ dont seuls les seconds membres changent.

Cette configuration est idéale pour les solveurs directs qui, une fois la factorisée construite, n'ont plus qu'à effectuer les descente-remontées *ad hoc*. *A contrario*, c'est le cauchemar des solveurs itératifs qui démarre sans connaissance *a priori* de la solution et recommence, à chaque résolution, tout le processus. Dans le cas des solveurs itératifs de type Krylov (tel le GCPC), on réutilise parfois les directions de descente des systèmes linéaires précédents pour tenir compte de l'information spectrale déjà exhumée et ainsi «booster» le démarrage de l'algorithme. On parle alors de techniques d'accélération.

Avec cependant toutes les contingences de manipulations de directions de descente déjà évoquées pour les procédures de réorthogonalisations (cf §4.4). Dans le cas du solveur d'interface FETI, on a vu que ce type de technique était plus pertinente du fait de la meilleure distribution spectrale de l'information (les premières directions de descente de chaque pas de temps sont les plus importantes) et de la plus faible taille de l'information à stocker (taille de l'interface).
Supposons que l'on cherche à résoudre le système $\mathbf{K}\mathbf{u}^{\mathbf{2}}=f^2$ après avoir déjà résolu $\mathbf{K}\mathbf{u}^{\mathbf{1}}=f^1$ et stocké ses N_{orth} premières directions de descentes $p_\Gamma^{n,1}$. Ces directions de descentes sont supposées orthogonalisées à la précision machine près, donc en les rangeant dans une matrice carré $V_1:=\left[p_\Gamma^{1,1} \quad \ldots \quad p_\Gamma^{N_{orth},1}\right]\in\Re^{n_I\times N_{orth}}$, elles vérifient

$$\left(V_1\right)^T\left(P^T F_I P\right)V_1=H_1 \text{ avec } H_1:=\operatorname{diag}\left(h_1^1 \quad \ldots \quad h_1^{N_{orth}}\right) \tag{5.2-9}$$

La procédure d'accélération va donc consister à rajouter, dans la phase d'initialisation de FETI, une composante sur l'espace engendré par ces directions de descente (en reprenant les notations de l'algorithme 4.5-2)

$$\begin{aligned}\tilde{\lambda}^0 &:= G_I\left(\left(G_I\right)^T G_I\right)^{-1} e\\ \lambda^0 &:= V_1\left(H_1\right)^{-1}\left(V_1\right)^T P^T\left(d_2-F_I\tilde{\lambda}^0\right)+\tilde{\lambda}^0\end{aligned} \tag{5.2-10}$$

et à réorthogonaliser les nouvelles directions de descente $p_\Gamma^{n,2}$ par rapport à celles du système précédent V_1. Et ce, aussi bien dans la phase d'initialisation que dans l'étape (11). Il faut donc organiser un processus de réorthogonalisation a géométrie variable: au sein d'une même résolution entre les directions de descentes nouvellement exhumées et entre les différentes résolutions, avec les N_{orth} directions déjà stockées.
Ce procédé se généralise à plusieurs résolutions successives, et permet, à peu de frais de booster la convergence des résolutions ultérieures. Pour plus de détails on pourra consulter le papier de C.Farhat[Far00].

Remarques:

- *Cette technique d'accélération est activable dans Code_Aster via le mot-clé* `ACCELERATION_SM`. *Elle procure des gains en nombre d'itérations notables: si les seconds membres sont «proches», le second FETI peut converger en moitié moins d'itérations et ainsi de suite.*
- *L'équivalent de cette technique d'accélération existe aussi pour les problèmes à multiples premiers membres ('near-by problems') dont les matrices et les seconds membres ont peu bougés entre deux résolutions (grandes déformations, contact-frottement). Elles ont été testées mais pas restituées dans Code_Aster.*

5.2.7 Conditions de Dirichlet

Dans *Code_Aster*, la prise en compte de condition de Dirichlet s'effectue *via* des multiplicateurs de Lagrange *ad hoc*, qu'ils ne faut donc pas confondre avec ceux du problème d'interface FETI. Ces DDLs tardifs sont

43 'Multiple right hand side' dans la littérature.

attachés aux DDLs physiques dont ils imposent les valeurs et numériquement et informatiquement, ils évoluent conjointement[44]. Lorsqu'il s'agit de ménager un flot de données indépendants, sous-domaine par sous-domaine, il faut bien veiller à les attacher au bon processeur. Les choses se compliquent lorsque ces Lagranges impliquent des nœuds de l'interface. Pour respecter certaines contraintes de la mécanique non linéaire et en particulier le fait que les valeurs de ces Lagranges participe à l'estimation des résidus mécaniques, il a été fait le choix de les attribuer à un des sous-domaines[45].

Cette modélisation des Dirichlets par des Lagranges est très riche car elle permet la prise en compte de condition de Dirichlet généralisée du type (mot-clé LIAISON_…)

$$u_x^7 - 3u_y^{12} = 5 \quad (5.2\text{-}11)$$

Certains auteurs[Rix02] proposent des stratégies (second problème grossier, préconditionneur) pour compléter FETI afin qu'il puisse aider un code ne disposant par de ces «Lagranges de Dirichlet» à prendre en compte ces conditions. On parle de problème à multiples contraintes linéaires (LMPCs pour 'Linear MultiPoint Constraints'). C'est un des attraits de la méthode FETI. Mais manque de chance, Code_Aster disposant déjà de cette faculté, ces deux fonctionnalités se telescopent plutôt ! En pratique, dans *Code_Aster*, lorsque le partitionneur détecte un Dirichlet généralisé entre des DDLs strictement contenus au sein de sous-domaines distincts (c'est-à-dire non portés par un nœud d'interface), le calcul s'arrête sur un message d'erreur signifiant l'imcompatibilité. Tous les autres cas de figures sont normalement autorisés, testés dans la base de cas-tests officielle et correctement pris en compte.

5.2.8 FETI & Co

Depuis 1991, du fait des nombreuses connections entre les méthodes de Schur primales et duales, de nombreuses variantes ont vues le jour:

- FETI-2 qui introduit un second niveau de problème grossier pour mieux prendre en compte les problèmes de type plaque et coque,
- Les méthodes hybrides (FETI-DP, GIRKS…) qui cherchent à garantir la continuité des déplacement sur une partie de l'interface et l'équilibre des efforts sur son complémentaire,
- Les méthodes mixtes (LaTIn, FETI-2-champs…) qui remplacent les conditions d'interface usuelles par une combinaison linéaire de type Robin. D'où une plus grande richesse de comportements mécaniques (contact-frottement…).
- De nombreuses autres plus ou moins pérennes, liées à des aspects déjà évoqués (interface duale, projecteur, LMPC…): P-FETI1, P-FETI2, FETI-T, A-FETI etc.

Pour de plus amples informations, on pourra consulter les synthèses récentes de P.Gosselet & C.Rey[Gre06] et de Y.Fragakis & M.Papadrakakis[Fpa03].

5.3 FETI versus Neumann-Neumann

5.3.1 Récapitulatif

D'après ses pères et les différentes équipes qui ont, par la suite, implanté et utilisé la méthode FETI, elle présente de nombreux avantages. On a déjà insisté sur les points suivants:

- Meilleures performances que beaucoup de solveurs mono-domaines et de méthodes DD,
- Sa scalabilité numérique intrinsèque indépendamment de tout préconditionneur,
- La possibilité de lui adjoindre un préconditionneur efficace et peu coûteux (lumped),
- Méthode d'appariement de calculs et de raccord de maillage/modèle,
- Aspects solveur de «recherche» ouvert à de nombreuses adaptations à la carte contrairement aux solveurs directs «boîtes noires»,
- Le traitement des Dirichlets généralisés,
- Large périmètre d'utilisation,
- Foisonnement de travaux académiques et implantation dans des codes industriels (ZEBULON, SAMCEF),
- «Relative» facilité d'implantation dans un code existant,
- Ses faibles complexité calcul et mémoire.

Sachant que, pour modérer cet enthousiasme,

44 Par exemple, ils ne forme qu'une seule entité aux yeux des renuméroteurs de matrices.
45 Par convention, celui dont le numéro est le plus faible.

- Sa grande difficulté est la détermination des modes de corps rigides, tandis que celle des méthodes itératives primales est la mise en place d'un problème grossier efficace et général.
- Sa sensibilité aux «ratio d'aspect» ('ratio aspect') des sous-domaines. Plus leurs dimensions sont équilibrées meilleure est la convergence de l'algorithme.
- Sa sensibilité aux hétérogénéités de matériaux et à la distance[46] moyenne entre sous-domaines. Plus cette dernière est faible, plus grande est la vitesse de propagation de l'information sur toute la structure au cours de l'algorithme.
- Dans les deux cas, FETI ou Neumann-Neumann, des développements complémentaires sont souvent souhaitables: réorthogonalisation des directions de descentes, équilibrage, accélérations…

D'autres items méritent d'être précisés et c'est ce que nous allons brièvement aborder dans les paragraphes suivants.

5.3.2 Superconvergence

On a vu que les valeurs propres (cf. §4.4) des opérateurs d'interfaces sont bien sériées: celles de grande amplitude sont bien séparées des autres, ce qui améliore le conditionnement effectif des opérateurs (rapport des valeurs propres extrémales non encore capturées) au cours d'un algorithme de type GC. D'où la superconvergence «constatée» du processus[47].

L'opérateur FETI étant compact, son spectre comporte un point d'accumulation près de zéro (Th. de Riesz-Fredholm) et sa partie haute est moins fournie que sa partie basse. Bref, même si les opérateurs S et F_I ont des conditionnement initiaux identiques, la convergence du second l'emporte sur le premier (cf. [FR94] §6.1). D'ailleurs, le préconditionneur lumped de FETI amplifie ce phénomène en tassant les plus petites valeurs propres quasiment de la même manière que celui de Dirichlet, et pour un coût bien moindre.

Dans tous les cas, Neumann-Neumann ou FETI, une procédure automatique de réorthogonalisation doit être rajoutée dans la phase de construction de la nouvelle direction de descente. Car il est avéré que ce type de situation accélèrent la perte d'orthogonalité[48] «congénitale» des méthodes de Krylov.

5.3.3 Interface de mesure nulle

Contrairement aux méthodes primales, FETI ne nécessite pas le calcul d'intégrales sur des domaines de mesure nulle: point en 2D/3D et segment en 3D. Par exemple, pour un partitionnement en sous-domaines cubiques et avec une discrétisation par éléments finis nodaux, FETI doit prévoir les communications d'un sous-domaine avec au maximum 6 voisins, pour les méthodes primales ce chiffre grimpe à 26 !

Cela provient de la formulation faible de la contrainte de continuité qui s'impose *via*

$$\left(u_i - v_j, \mu_{ij}\right)_{\Gamma_I} := \int_{\Gamma_{i,j}} \mu_{ij} Tr_{\Gamma_{ij}}\left(v_i - v_j\right) d\Gamma \qquad (5.3\text{-}1)$$

Donc évidemment si $mes\left(\Gamma_{i,j}\right)=0$, ce produit scalaire est nul et aucune communication n'est requise entre les deux sous-domaines.

Les méthodes primales n'ont pas cette chance car elles reposent sur une décomposition des variables du problème, en degrés de liberté internes et degrés de liberté d'interface, toutes les contributions des uns et des autres doivent donc s'agglomérer pour constituer les composantes locales du complément de Schur.

En plus des gains en complexité calcul, occupation mémoire, implantation et maintenance informatique de l'algorithme, cela réduit d'autant les communications inter-processeurs lorsqu'on est en parallèle et facilite donc la scalabilité du processus. Bien sûr, en primal, des techniques peuvent être déployées pour grouper les messages de différentes entités géométriques, mais elles compliquent sérieusement l'implantation de l'algorithme et bride la généralité des interconnexions entre sous-domaines.

46 La distance entre deux sous-domaines (au sens de la théorie des graphes) est le nombre minimal d'interfaces à franchir pour passer de l'un à l'autre.

47 Cf. A.Van der Sluis & A.Van der Vorst. *The rate of convergence of conjugate gradient*. Numerische Mathematik, **48** (1986), pp543-560.

48 Cf. B.N.Parlett. *The symmetric eigenvalue problem*. Ed. Prentice Hall (1980).

5.3.4 Assemblage des matrices de rigidité

Dans le même ordre d'idée, FETI sans préconditionneur de Dirichlet n'a pas besoin de distinguer les variables internes et externes sous-tendant tels ou tels blocs matriciels ou vectoriels. Cela simplifie d'autant l'implantation de l'algorithme dans un code existant dans les phases assemblages matricielles et vectorielles.

5.3.5 Périmètre d'utilisation

Depuis l'arrivée sur «le marché de la DD» de FETI, les méthodes de Schur duales font l'objet de recherches théoriques et appliquées constantes et donc leur périmètre d'utilisation n'a cessé de s'étoffer. De manière non exhaustive on a pu recenser:

- Mécanique non-linéaire avec hétérogénéités sévères,
- Analyse modale et vibratoire,
- Problèmes d'évolution,
- Problèmes de contact-frottement,
- Acoustique,
- Modélisation de type plaque ou coque,
- Multiphysique: milieux poreux, fluide incompressible,
- CFD et électromagnétisme,
- Composites,
- Dynamique rapide.

Pour chacune de ces applications, des aménagements ont vu le jour et ont donc fait de FETI, la méthode DD «tout terrain» de la mécanique des structures.

5.3.6 Speed-up et scale-up

La scalabilité numérique de FETI muni d'un préconditionneur de Dirichlet a déjà été mentionnée. En résumé, d'un point de vue théorique:

- Si on augmente la finesse de discrétisation tout en maintenant le nombre de sous-domaines, la méthode converge en un nombre d'itérations similaire (d'où un bon scale-up),
- Si au contraire, on fixe la taille du problème global tout en augmentant le nombre de sous-domaines (pour par exemple profiter de plus de processeurs disponibles), le nombre d'itérations décroît (d'où un bon speed-up),
- Si on maintient un nombre de degrés de liberté constant par sous-domaine, tout en augmentant la taille du problème global *via* le nombre de sous-domaines, le nombre d'itérations demeure inchangé.

Sur le papier donc, FETI résout un problème n fois plus grand dans le même temps CPU, pourvu que le nombre de processeurs connaisse la même croissance. Cette extensibilité faible et forte a été vérifiée en pratique sur des cas d'écoles et des études industrielles de grandes tailles : on parle alors de scalabilité parallèle. Des analyses mécaniques de structures comportant des millions d'inconnues, distribuées sur des milliers de processeurs, ont montrées l'extensibilité de la méthode[Far02].

6 Le partitionnement de maillage

6.1 Généralités

Pour des structures industrielles souvent complexes et de grandes tailles, il n'est pas raisonnable de demander systématiquement aux utilisateurs qui ont déjà passé de longues semaines à construire un maillage, de le partitionner manuellement. D'autant plus que les critères d'élaboration d'un maillage n'ont rien à voir avec ceux qui président à son découpage en vue d'un calcul par DD. Il est plus avisé de faire appel à des outils automatiques ou semi-automatique[49] pour répondre à ce problème multi-critères sous contraintes:

- Réduire les interfaces pour minimiser la taille du problème traité et le volume des communications.
- Equilibrer la taille des sous-domaines,
- *A contrario*, étirer les sous-domaines afin de réduire la largeur de bande des K_i,
- Faire tendre vers l'unité les ratios d'aspect pour améliorer le conditionnement du problème d'interface,
- Réduire la distance, au sens graphe entre sous-domaines, pour augmenter l'efficacité du problème grossier,
- Limiter le nombre de structures flottantes afin de minimiser la taille du problème grossier,
- Limiter le nombre de points de jonction qui dégradent les conditionnements et compliquent l'algorithmique,

La plupart de ces *items* ont déjà été abordés tout au long de ce document et certains sont clairement antagonistes: par exemple, «augmenter les connexions inter-domaines /réduction de la largeur de bande» *versus* «réduction des interfaces». Suivant les cas de figures et les besoins de l'utilisateur, on attribut le statut de critère ou celui de contrainte, aux uns ou aux autres.

Et ce d'autant plus que d'autres aspects, plus métiers ou organisationnels, peuvent rentrer en ligne de compte:

- Découpage respectant l'intégrité des pièces mécaniques représentées,
- Répartition du travail d'étude entre contractants,
- Hétérogénéités de maillage, de caractéristiques matériaux ou de chargements,
- Limitations CPU et/ou mémoire,
- Bibliothèque de composants,
- Reprise d'anciens calculs et/ou de portions de maillage,
- Contingenter à l'intérieurs des sous-domaines les zones particulières : Dirichlet généralisé, contact-frottement …
- Prise en compte de maillages ou d'éléments finis incompatibles,
- Compromis avec les algorithmes de renumérotation et/ou d'équilibrage de charge dynamique,
- Préparer des stratégies complémentaires: maillage adaptatif, superposition de modèles…

Parmi tous ces critères, des expériences numériques[FR94][Gos03] ont montré que les ratios d'aspect et les hétérogénéités jouent un rôle déterminant. De leurs bonnes valeurs et de leurs répartitions dépendent grandement les performances numériques de l'algorithme. Même en séquentiel, un bon partitionnement est souhaitable car il induit une bonne localisation des données et assure ainsi une meilleure efficacité à l'algorithme.

6.2 Algorithmes de partitionnement

La grande variété des algorithmes de partitionnement n'a d'égale que le nombre de chercheurs travaillant dans le domaine ! Au delà de cette boutade et même si la situation s'est éclaircie par rapport au début des années 90, une kyrielle de méthodes restent en lice et un flot de nouvelles, issues de domaines aussi variés que les algorithmes génétiques et stochastiques, les réseaux neuronaux ou le traitement du signal, continu à l'alimenter.
Toute tentative de classement est dès lors périlleuse. Il semble tout de même qu'elles peuvent se décomposer en deux grandes familles: celles réalisant des partitionnements dit

- **EOD ('Element Oriented Decomposition'),** plutôt utilisées en éléments finis, pour lesquelles l'unité de découpe et d'équilibrage est l'élément. Les interfaces sont alors constituées d'arêtes ou de facettes.
- **VOD ('Vertex Oriented Decomposition')**, plutôt utilisées en volumes finis, pour lesquelles l'unité est le sommet. Dans ce cas, des éléments peuvent être partagés entre plusieurs sous-domaines.

49 L'utilisateur peut ainsi mieux orienter la stratégie de découpe pour des raisons liées à l'étude, à ses différents contractants ou aux limitations de la méthode DD qui va ensuite utiliser ce partitionnement.

Cette arborescence se subdivise ensuite en au moins trois catégories:

- Les méthodes d'ingénieurs,
- Celles issues de l'optimisation,
- Les heuristiques basées sur la théorie des graphes.

Parmi toutes ces techniques, il est difficile de choisir *a priori*: ce choix est «maillage, méthode DD et études dépendants». Les outils de partitionnement clé en main (METIS, SCOTCH…) en proposent souvent plusieurs. Ils sont structurés en deux étapes. Une première phase choisit de manière déterministe un ensemble de sous-domaines $\left\{\Omega_i\right\}_{i=1}^{P}$ et leurs topologies. La seconde optimise ce résultat suivant certains critères et sous contraintes.

Les algorithmes de la première phase peuvent être de type glouton (GR pour 'Greedy algorithm' ou FGR pour 'Fast GReedy'), basées sur la théorie des graphes et les renumeroteurs (Cuthill-Mc Kee standard ou récursif, par avancée de fronts), la théorie spectrale (bissection spectrale récursive RSB), les coordonnées des points du maillage (décomposition par la médiane, par secteur angulaire), par projection sur un axe (algorithmes de type moments d'inertie, standard ou récursif). Dans ceux de la second phase, on retrouve plutôt les méthodes tabou, les recuits simulés, les algorithmes génétiques ou probabilistes.
Les plus populaires et les plus efficaces, en moyenne, semblent être les algorithmes de type glouton : ils sont très rapides et produisent une décomposition qui est souvent presque optimale. Pour plus renseignements on peut consulter [FR94], [LeA05] et les sites des logiciels «majors» dans ce domaine: CHACO, JOSTLE, METIS et SCOTCH.

6.3 Outils de découpage de graphes

Parmi les nombreux outils disponibles «sur étagère», certains sont plus cités que d'autres dans la littérature DD. Ainsi, de manière non exhaustive[50] , on retrouve:

- **SplitMesh**: développé et utilisé à l'ONERA pour le code ZEBULON, ainsi que chez HUTCHINSON, DASSAULT-AVIATION et à la SNPE. Il découpe un maillage en se basant uniquement sur la topologie des éléments car il n'utilise qu'un seul type d'algorithme: des bissections emboîtées. Une particularité intéressante dans le cadre d'une DD multiniveau est sa capacité à maintenir une information hiérarchique du découpage (sous-domaines «emboîtés»).

- **METIS**[Met]: développé par G.Karypis de l'université du Minnesota, cette librairie est surtout connue pour l'algorithme de renumérotation de matrices creuses qui porte son nom. Ce dernier est d'ailleurs déjà présent dans *Code_Aster* avec la multifrontale, MUMPS et FETI (mot-clé `RENUM='METIS'`).
  Des codes comme SIC (UTC) et ARC3D (HUTCHINSON) l'ont choisi car il intègre les meilleurs algorithmes du moment, à la fois efficaces et robustes. Des versions séquentielles et parallèles sont disponibles sur le site officiel. Entre autre particularité, il permet de coupler renumérotation, découpage et d'améliorer le produit d'un autre partitionneur.
  En tant que partitionneur séquentiel de maillage, il est intégré à l'opérateur de partitionnement (semi)automatique de *Code_Aster*, `DEFI_PART_FETI`[U4.23.05]. Deux variantes sont possibles: la PMETIS (mot-clé `METHODE='PMETIS'`) basée sur une méthode récursive de bissection et la KMETIS (`METHODE='KMETIS'`) qui s'appuie sur une méthode dite «k-way». Toutes les deux sont appelées en tant qu'outils externes.

- **SCOTCH**[Sco]: développé par F.Pellegrini de l'INRIA-Bordeaux, ce partitionneur s'appuie sur des heuristiques d'optimisation locale de type Fiduccia-Mattheyses très performantes. Il manipule exclusivement des graphes de communication associés aux maillages. Cela oblige donc l'utilisateur à convertir au préalable son maillage sous forme de graphe. Mais en même temps cette contrainte peut se révéler très utile pour imposer des stratégies particulières (cf. le groupage maille volumique/maille de peau associée [LeA05]). Une version séquentielle et une parallèle sont disponibles, ainsi qu'un interpréteur de stratégies permettant d'adapter plus finement les critères et les contraintes de l'outil aux cas de figure de l'utilisateur.

50 Il y' a aussi MS3D, JOSTLE, CHACO, CHAOS, PARTY…

Lui aussi est appelable dans DEFI_PART_FETI (mot-clé METHODE='SCOTCH'), en séquentiel, mais cette fois en tant que bibliothèque (pour plus d'efficacité). Il est utilisé en tant que renumeroteur cette fois via la solveur linéaire MUMPS (mot-clé RENUM='SCOTCH').

6.4 Implantation dans *Code_Aster*: *DEFI_PART_FETI/OPS*

6.4.1 Principe

Pour partitionner un maillage, *Code_Aster* propose deux alternatives:

- On peut se contenter d'un partitionnement automatique plus ou moins piloté, on fait alors appel à DEFI_PART_FETI. L'opérateur utilise METIS ou SCOTCH pour produire un ensemble de groupes de mailles (volumiques et de bords) correspondant à autant de sous-domaines.
- On veut ou on doit utiliser un partitionnement manuel (cf. les contingences énoncées au §6.1), c'est le rôle de DEFI_PART_OPS. L'utilisateur rempli lui même la liste des groupes de mailles volumiques et surfaciques correspondant aux sous-domaines. L'opérateur opère des vérifications de cohérences (notamment vis-à-vis des mailles associées aux charges) et produit la structure de données *Aster* décrivant le partitionnement pour FETI: la *SD_FETI*[D4.06.21].

En pratique, DEFI_PART_OPS est appelé en fin de processus par DEFI_PART_FETI. Le fait de pouvoir l'utiliser directement est une facilité laissée à l'utilisateur averti.

6.4.2 Périmètre d'utilisation et difficultés

Les mailles de peau doivent être identifiées et traitées de manière adéquate pour se retrouver dans le bon sous-domaine (celui de leur élément de volume associé) et ne pas risquer de constituer un sous-domaine à part entière. Ce dernier serait alors sans rigidité ! Le repérage des mailles de peau est un problème en soit, car il ne peut pas être simplement appréhendé par des notions de dimension: un calcul 3D peut comporter des éléments 2D voire 1D qui alimentent toutes la matrice de raideur. De même, dans la profusion des typologies de calculs *Code_Aster* , la simple définition des interfaces est parfois problématique. Deux mailles superposées participant au calcul de la même zone géométrique peuvent se trouver séparées dans des sous-domaines distincts et prendre donc une portion d'interface en sandwich. *A contrario* , le repérage d'une interface dans une zone de contact, de fissuration ou d'échange paroi en thermique, dépasse largement la notion usuelle de «zone partagée entre plusieurs sous-domaines».
Une option de l'opérateur DEFI_PART_FETI peut pallier certaines de ces difficultés (TRAITER_BORD) en retirant avant le partitionnement les mailles de bords repérées géométriquement: celles qui sont incluses dans d'autres mailles au sens strict. Une fois le partitionnement des mailles restantes effectué, on les ré-injecte dans le sous-domaine de leur maille père.
En fin de chaîne, l'opérateur de transcription du partitionnement en objets *Code_Aster*, DEFI_PART_OPS, reçoit bien séparément les parties volumique (GROUP_MA) et surfacique (GROUP_MA_BORD , si elle existe) de chaque sous-domaine.
Autre classe de difficulté, il peut arriver que le partitionneur génère des sous-domaines non connexes. Ainsi dans l'exemple ci-dessous (cf. figure 6.4.1) d'une ailette de turbine, la plupart des sous-domaines sont en plusieurs morceaux. La définition des interfaces peut s'en trouver perturbée et la résolution du problème d'interface achoppe. Une option de DEFI_PART_FETI permet de détecter (mot-clé CORRECTION_CONNEX) ce cas de figure et d'arrêter le calcul si nécessaire. Plus tardivement dans la chaîne de calcul, une option avancée de FETI teste la validité de l'interface (INFO_FETI) en simulant un calcul trivial sur le partitionnement généré.

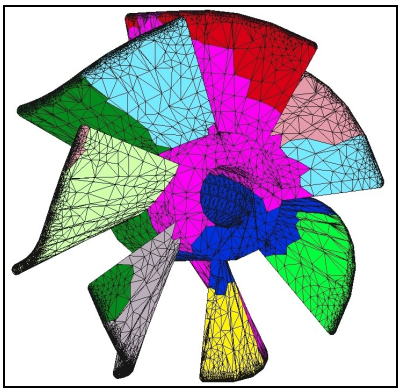


Figure 6.4.1._ Partitionnement non connexe d'une ailette de turbine.

Cependant, la non connexité ou le caractère étoilé[51] d'un sous-domaine si ils ne perturbent pas la définition des interfaces (un point d'interface est un point commun à plusieurs sous-domaines) ou ne la rend pas indécidable, n'est pas rédhibitoire pour FETI. En attestent les cas-tests FETI008b ([V1.04.108], plaque partitionnée en 2 SD étoilés) et FETI010a ([V1.04.110], contact-frottement entre deux parois) de la base officielle du code.

Pour finir, des traitements particuliers doivent être effectués dès la phase de partitionnement pour juger de l'adéquation entre le découpage et les entités géométriques des chargements. Et le cas échéant, soit prévenir et arrêter le calcul, soit adopter des stratégies correctives. Car ménager un flot de données indépendant, sous-domaine par sous-domaine, entre les mailles initiales partitionnées (volumiques et surfaciques) et les mailles tardives induites par certains chargement, n'est pas toujours une sinécure !

On a déjà vu au paragraphe §5.2 que les conditions de Dirichlet simples requièrent des traitements particuliers pour projeter leurs LIGRELs de mailles tardives associées. En particulier lorsqu'elles concernent, et c'est souvent le cas en partitionnement automatique, des points situés sur l'interface. Le cas des Dirichlet généralisés est plus problématique car ils induisent une dépendance entre sous-domaines. Lorsque le partitionneur détecte un Dirichlet généralisé entre des DDLs strictement contenus au sein de sous-domaines distincts (c'est-à-dire non portés par un nœud d'interface), le calcul s'arrête sur un message d'erreur signifiant l'incompatibilité. Tous les autres cas de figures sont normalement autorisés, testés dans la base de cas-tests (FETI001 à 06[V1.04.101/106]) et correctement pris en compte. De même pour les zones de contact-frottement (avec la méthode continue uniquement) qui doivent être complètement insérées au sein d'un sous-domaine, sans intersecter l'interface (cf. FETI008 à 10[V1.04.108/10]).

Pour finir, certaines difficultés pour générer un partitionnement licite vis-à-vis de calcul multi-domaines visés peuvent être contournées *via* les mot-clés GROUPAGE et POIDS_MAILLE de DEFI_PART_FETI (uniquement avec METHODE='SCOTCH'). En activant le premier, on peut forcer le partitionneur à maintenir dans un seul sous-domaine un ensemble de mailles (par exemple, pour prendre en compte une zone de contact). Quant au second, il permet de construire des sous-domaines de tailles variables (par exemple, pour ménager un équilibrage de charge particulier: grosse zone élastique/petite zone plastique) en attribuant des poids à des groupes de mailles.

6.5 Exemples d'utilisation et paramétrages

Récapitulons le paramétrage principal de ces opérateurs de partitionnement et illustrons leurs utilisations *via* des cas-tests officiels. Pour plus d'informations, on pourra consulter la documentation utilisateur associée[U4.23.05] et les cas-tests officiels déjà cités.

Opérateurs	*Mot-clé*	*Valeur par défaut*	*Détails/Conseils*	*Réf.*
DEFI_PART_FETI	CORRECTION_ CONNEX	'NON'	'OUI'/'NON' coûteuse sur gros cas 3D.	§6.4
	EVALUATION	SO [52]	Équilibrage de charge.	§6.4
	EXCIT	SO	Mêmes chargements que le calcul mécanique.	SO
	GROUPAGE	SO	Grouper des mailles.	§6.4
	METHODE	*'SCOTCH'*	'KMETIS', 'PMETIS' et 'SCOTCH'. PMETIS préconisé pour un nombre de sous-domaines < 20, KMETIS au delà. SCOTCH est plus efficace mais semble moins robuste.	§6.3
	MODELE	SO	Même modèle que le calcul mécanique.	SO
	NB_PART	SO	Nombre de sous-domaines.	SO

51 En dimension n, sous-domaine dont les parties se joignent par des entités de taille n-2 (point en 2D, point/segment en 3D).
52 Sans Objet.

Opérateurs	Mot-clé	Valeur par défaut	Détails/Conseils	Réf.
	TRAITER_BORDS	'OUI'	'OUI'/'NON'.	§ 6.4
DEFI_PART_OPS	GROUP_MA	SO	Mailles volumiques participant au calcul des matrices et à la détermination de l'interface.	§ 6.4
	GROUP_MA_BORD	SO	Mailles de peau.	§6.4
	MODELE	SO	Même modèle que le calcul mécanique.	SO
	EXCIT	SO	Mêmes chargements que le calcul mécanique.	SO

Tableau 6.5-1._ Récapitulatif du paramétrage principal des opérateurs de partitionnement.

```
SDFETI1= DEFI_PART_OPS(NOM='SD',
                       MODELE=MODM,
                       DEFI=(_F(GROUP_MA='FETI1',),
                             _F(GROUP_MA='FETI2',),
                             _F(GROUP_MA='FETI3',GROUP_MA_BORD='B5',),
                             _F(GROUP_MA='FETI4',GROUP_MA_BORD='B6',),),
                       EXCIT=(_F(CHARGE=CH1),
                              _F(CHARGE=CH2),
                              _F(CHARGE=CH3),
                              _F(CHARGE=CH4),),);
SDFETI2 = DEFI_PART_FETI (
                  MODELE=MODM,
                  NB_PART=16,
                  EXCIT=(_F(CHARGE=CH1),
                         _F(CHARGE=CH2),),
                  METHODE='SCOTCH' ,
                  NOM_GROUP_MA='SD',
                  TRAITER_BORDS='OUI',
                  CORRECTION_CONNEX='NON',) ;
```

Exemple 6.5-1._ SDFETI1, partitionnement manuel en 4 SD (FETI005a).
SDFETI2, automatique en 16 SD (FETI007a).

7 Implantation dans *Code_Aster*

7.1 Difficultés particulières

Les développements autour de FETI ont «re-défriché» des terrains logiciels qui avaient déjà fait l'objet de travaux par le passé: le multidomaine et le parallélisme. Un des leitmotivs de ce chantier logiciel a été justement de réutiliser, autant que faire se peut, l'expérience acquise sur ces thématiques. Un autre souci, a été de s'insérer dans l'architecture du code sans briser sa cohérence et en essayant de respecter ses exigences en termes d'utilisation (modularité, paramétrisation, robustesse, complétude) et de qualité logicielle (validation, documentation). Cette volonté de ne pas révolutionner la structure du code et son flot de données et de gérer, dans une seule version l'*Aster* standard séquentiel et le multidomaine parallèle, en cherchant plus la complétude et la robustesse que les performances, a conduit à des contorsions logicielles et algorithmiques pénalisantes.
Dès l'étude bibliographique[Boi03] certaines difficultés avaient été évaluées:
•S'insérer dans l'architecture du code sans briser sa cohérence d'ensemble, ni brider les potentialités de la DD.
•Respecter les exigences du code en terme d'utilisation et de qualité logicielle: modularité, flexibilité, robustesse, paramétrisation, portabilité, complétude et évolutivité.
•Faire cohabiter la prise en compte, quasi-systématique dans *Code_Aster*, des conditions limites par double dualisation et la méthode FETI d'où la gestion simultanée de deux familles de Lagrange.
•Encapsuler les structures de données principales du code (matrice, vecteur…) pour «bluffer» l'architecture actuelle et émuler la notion d'héritage entre structure de données mère (domaine global) et filles (liées aux sous-domaines).
•Obtenir automatiquement et avec un degré de robustesse suffisant, la «pierre angulaire de l'édifice FETI-1»: modes de corps rigides et pseudo-inverses des sous-domaines flottants.

Mais bien sûr, au cours des développements, d'autres difficultés sont apparues:

•Le traitement des mailles de peau et des non connexités de sous-domaine,
•La projection des chargement par sous-domaine et en particulier celle des mailles tardives,
•La problématique des multiples seconds membres,
•La mécanique non linéaire et la prise en compte de zones de contact-frottement,
•Le parallélisme et la qualification MPI du code,
•La gestion de la mémoire JEVEUX .

Le lecteur désirant plus de détails sur l'implantation de l'algorithme et les difficultés de ce chantier logiciel pourra consulter les notes [Boi05][Boi07] et les documentations informatiques du code [D9.03.01] et de ses structures de données [D4.06.05/07/10/11/21].

7.2 Tests et validations

D'autre part, la validation du processus complet de décomposition de domaine FETI fait d'une attention particulière: une vingtaine de cas-tests officiels ont déjà été restitués(cf. FETI001a à FETI010a [V1.04.101 à 110]) et deux campagnes de tests sur des cas semi-industriels ont été menées[Boi05][Ass07].
Lors du développement logiciel, un effort constant a été produit pour permettre au développeur et à l'«utilisateur averti» de tester et de tracer le déroulement de l'algorithme FETI: de la factorisation symbolique aux différentes étapes du solveur d'interface, en passant par les calculs élémentaires. En jouant sur les options du paramètre INFO_FETI , on peut par exemple tester la validité des modes rigides, les orthogonalités du GCPPC, calculer les premiers modes de l'opérateur d'interface, tracer et profiler les messages MPI, récapituler les tailles des sous-domaines, de leurs matrices de rigidité et de leurs factorisées (pour évaluer l'équilibrage), les temps passés par processeur à chaque étape du calcul…

Une option (cf. § 7.4, INFO_FETI(12:12)='T') permet d'ailleurs de tester la validité de l'interface et de la SD_FETI en vérifiant l'inversion des systèmes canoniques sur le découpage réel produit par DEFI_PART_*** ainsi que des règles de cohérences des différents objets JEVEUX .

Certains de ces tests et affichages restent valides en séquentiel comme en parallèle (certaines études ne peuvent fonctionner que dans ce mode). Leur prolixité dépend du paramétrage (INFO_FETI et INFO) et de l'opérateur.

7.3 Périmètre d'utilisation

Pour l'instant et compte-tenu de certaines difficultés ou de choix du cahier des charges, le périmètre d'utilisation de FETI, séquentiel et parallèle, est restreint:

- A la **mécanique statique linéaire et non linéaire** (opérateurs MECA_STATIQUE et STAT_NON_LINE) générant des systèmes linéaires symétriques réels (donc sans la THM par exemple; signalé par une erreur fatale <F>).
- Pas de Dirichlet simple imposé *via* AFFE_CHAR_CINE (sinon <F>).
- Pas de macro-élément (<F>).
- Pas d'éléments finis étendus (X-FEM).
- Pas de Dirichlet généralisé (AFFE_CHAR_MECA/LIAISON_***) entre sous domaines (<F>).
- **Restriction du non linéaire**: pas de de force fluide (<F>); uniquement méthode de contact-frottement continue (<F>) avec chaque zone de contact insérée dans un sous-domaine, sans intersecter l'interface (<F>). On émet une alarme (<A>) lorsque des modélisations peu orthodoxes sont débusquées: point commun à plusieurs surfaces de contact…

Peu ou pas testé avec des éléments finis de structure, du pilotage, la recherche linéaire, critère de convergence en contrainte généralisée, poutre en grande rotation.

- Des réserves sur les modélisations ou les algorithmes susceptibles d'impacter les interfaces (sauf les chargements par AFFE_CHAR_MECA pour lesquels normalement ce cas de figure est prévu): éléments joints, fissure etc. Il vaut mieux (essayer) de contingenter ces zones à l'intérieur de sous-domaines.
- Solveurs locaux à chaque sous-domaine tous identiques, basés sur la multifrontale, et à paramétrage homogène (même renuméroteur…).

7.4 Exemples d'utilisation et paramétrages

Récapitulons le paramétrage principal de l'algorithme FETI-1 et illustrons son utilisation *via* des cas-tests officiels. Pour plus d'informations, on pourra consulter la documentation utilisateur associée[U4.50.01] et les cas-tests déjà cités.

Opérande	***Mot-clé***	***Valeur par défaut***	***Détails/Conseils***	***Réf.***
SOLVEUR/ METHODE='FETI'				
Paramètres fonctionnels	PARTITION	SO [53]	Nom de la structure de données SD_FETI générée par les op. de partition.	§6.5
	NMAX_ITER	0	Nombre d'itérations maximum du GCPPC. Si 0, NMAX_ITER[54]= $max(n_l/100,10)$.	§4.4
	REAC_RESI	0	Fréquence de réactualisation du calcul de résidu. Valeur conseillée= 10 ou 20.	§4.4
	RESI_RELA	10^{-6}	Critère de convergence. En non-linéaire, surtout si contact-frottement ou gros partitionnement, ne pas hésiter à durcir ce critère RESI_RELA<10^{-8}.	§4.4
	STOCKAGE_GI	'OUI'	'OUI', 'NON' et 'CAL'. Paramètre valide uniquement en séquentiel. Valeurs 'CAL' ou 'NON' conseillées si beaucoup de SD et peu de mémoire.	§4.3
Préconditionneur	PRE_COND	*'LUMPE'*	'SANS' et 'LUMPE'. Valeur par défaut optimale.	§4.5

53 Sans Objet.

54 n_l taille du problème d'interface.

Opérande	Mot-clé	Valeur par défaut	Détails/Conseils	Réf.
	SCALING	'MULT'	'SANS' et 'MULT'. Valeur par défaut optimale. Activé que si préconditionnement.	§4.5
Réorthogonalisation	NB_REORTHO_DD	0	Nombre de directions de descente initiales stockées par pas de temps. Si valeur nulle, NB_REORTHO_DD= max(NMAX_ITER/10,5). Plus ce chiffre est grand meilleure est la convergence mais c'est au détriment de la mémoire.	§ 4.4
	TYPE_REORTHO_DD	'GSM'	'SANS', 'GS', 'GSM' et 'IGSM'. Valeur par défaut optimale.	§4.4
Accélération	ACCELERATION_SM	'OUI'	'OUI' et 'NON'. Activée que si réorthogonalisation. Gros gains en convergence mais coûts mémoires.	§ 5.2
	NB_REORTHO_INST	0	Nombre de pas de temps précédents retenus pour la phase d'accélération. Plus ce chiffre est grand meilleure est la convergence mais c'est au détriment de la mémoire. Valeur conseillée: 10.	§5.2
Divers	NB_SD_PROC0	0	Pour décharger le processeur maître. Licite qu'en mode parallèle.	§5.1
	RENUM	'METIS'	'MD', 'MDA' et 'METIS'. Renuméroteur de la multifrontale pour les inversions des K_i.	§2.6
	NPREC	8	Seuil de détection des modes rigides	§4.4
Tests, monitoring et sorties fichiers	INFO_FETI	'F' x 24	INFO_FETI(1:5)='T' affichages «niveau développeurs». INFO_FETI(6:8)='T' test de la validité des modes rigides, calcul d'une partie du spectre de F_I et des orthogonalités du GCPC. INFO_FETI(12:12)='T' test de la validité de l'interface et de la SD_FETI. INFO_FETI(9:11)='T' profiling INFO_FETI(13:15) sorties fichiers de données et de résultats.	

Tableau 7.4-1._ Récapitulatif du paramétrage principal de FETI-1

```
RESU= MECA_STATIQUE (MODELE=MODM,
                     CHAM_MATER=CHMAT,
                     SOLVEUR=_F(
                              METHODE='FETI' ,
                              PARTITION=SDFETI2,
                              RESI_RELA=1.E-8,
                              NMAX_ITER=100,
                              NB_REORTHO_DD=100,
                              NPREC=5,
                             INFO_FETI='FFFFFFFFFFT',)
                     EXCIT=(_F(CHARGE=CH1),
                            _F(CHARGE=CH2),),),
```

Exemple 7.4-1._ Suite de l'exemple 6.5-1,
calcul FETI sur un cube partitionné en 16 SD avec affichage standard (FETI007a).

```
***********************************************
```

```
<FETI/ALFETI    0>                    Numéro du processeur
NUMERO D'INCREMENT     1              Numéro du pas de temps
NB SOUS-DOMAINES      16              Nombre de SD
NB DE MODES RIGIDES   88              Nombre total de modes rigides capturés
 POINTS INTERFACE / MAILLAGE / RAPPORT  Taille de l'interface
        5691        27107        0.21D+02 %
TAILLE (GI + GIT*GI)/MATRICE MOYENNE : 0.1263D+04 %  Tailles de G_I et de (G_I)^T G_I

 SOUS-DOMAINE   / MATRICE   / FACTORISEE   / NOEUDS   Tailles des K_i , K_i^-1
 N    1     :      121956      1380960         2041
 N    2     :      120192      1282914         2026
 N    3     :      120744      1403775         2031
 N    4     :      119658      1416582         2012
 N    5     :      121446      1418445         2046
….
N   16     :      118818      1336083         2010
 -------------------------------------------------
TOTAL       :        1913073        21479058           32245
MOYENNE     :      119567      1342441         2015

 SOUS-DOMAINE   / CPU FACSYM / CPU ASSE / CPU FACNUM   Profiling des étapes par SD
 N    0     :      0.4385D+00  0.1953D-02  0.0000D+00
 N    1     :      0.2412D+00  0.9863D-01  0.5869D+00
 N    2     :      0.2334D+00  0.9863D-01  0.5615D+00
 N    3     :      0.2324D+00  0.9863D-01  0.6094D+00
 N    4     :      0.2471D+00  0.9961D-01  0.6162D+00
 N    5     :      0.2412D+00  0.9863D-01  0.6172D+00
 …
 N   16     :      0.2422D+00  0.9570D-01  0.5684D+00
 -------------------------------------------------
CPU + SYS TOTAL  : 0.4202D+01  0.1554D+01  0.9476D+01
CPU + SYS LE PIRE: 0.4385D+00  0.9961D-01  0.6699D+00

 PROCESSEUR     / CPU CALCUL_ELEM  Profiling des calculs élémentaires par processeur
 N    1     :      0.1644D+01       (ici en séquentiel)
 ----------------------------------
CPU + SYS TOTAL  : 0.1644D+01
CPU + SYS LE PIRE: 0.1644D+01
 ***********************************************
 ***********************************************       Critère d'arrêt
 * FETI: NORME DU SECOND MEMBRE/RESIDU INITIAL = 0.1286D+00 0.4766D-01
 *       NORME DU RESIDU A ATTEINDRE EN ABS/RELA = 0.4766D-09 0.1000D-07
 ***************************************
 ***************************************       Evolution du critère
 * ITER    4 NORME DU RESIDU (RECURRENCE) EN ABS/RELA = 0.4207D-02 0.8827D-01
 ***************************************
 ***************************************
 * ITER   13 NORME DU RESIDU (RECURRENCE) EN ABS/RELA = 0.3116D-03 0.6538D-02
 ***************************************
 ***************************************
 * ITER   22 NORME DU RESIDU (RECURRENCE) EN ABS/RELA = 0.2835D-04 0.5948D-03
 ***************************************
 …
 ***************************************
 * ITER   55 NORME DU RESIDU (RECURRENCE) EN ABS/RELA = 0.1256D-08 0.2634D-07
 ***************************************
 * FETI: NORME DU RESIDU INITIAL/FINAL/RELATIF= 0.4766D-01 0.4588D-09 0.9626D-08

  *********************************      Récapitulatif des informations à convergence
  * CONVERGENCE EN   58 ITERATION(S)
  *********************************
```

Exemple 7.4-2._ Affichages standards de l'exemple précédent (feti007a.mess).

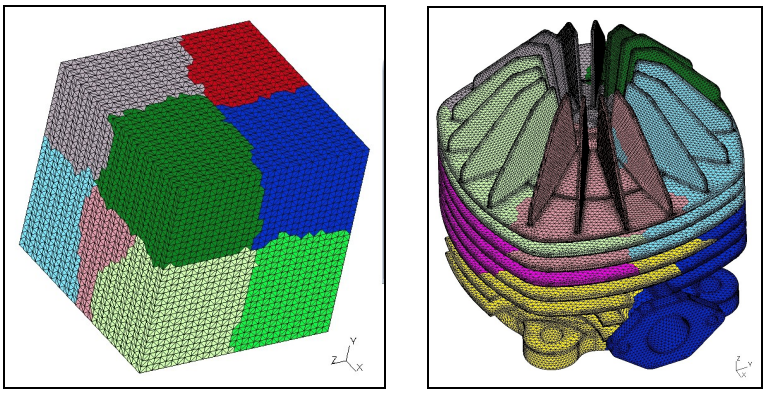
```
RESU= STAT_NON_LINE (MODELE=MODM,
                     CHAM_MATER=CHMAT,
                  SOLVEUR=_F( METHODE= ' FETI ',
                              PARTITION=SDFETI,
                              RESI_RELA=1.E-8,
                              NMAX_ITER=30,
                              NB_REORTHO_DD=30,
                              NB_REORTHO_INST =10,
                            ),
```

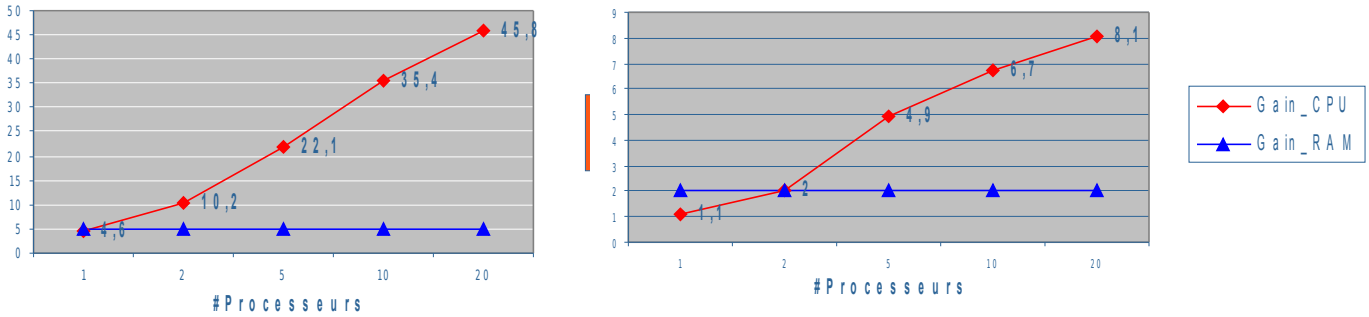
```
....

INSTANT DE CALCUL :  1.000000000E+00
-------------------------------------------------------------------------------------------
|   ITERATIONS    |     RESIDU      |     RESIDU      |     OPTION      |   ITERATIONS    |
|     NEWTON      |     RELATIF     |     ABSOLU      |   ASSEMBLAGE    |     FETI        |
|                 | RESI_GLOB_RELA | RESI_GLOB_MAXI |                 |                 |
-------------------------------------------------------------------------------------------
|     0        X | 2.59146E-01  X | 7.85020E+00    |TANGENTE         |    11           |
|     1        X | 6.28310E-02  X | 1.40760E+00    |                 |     1           |
|     2        X | 2.45893E-02  X | 5.36154E-01    |                 |     1           |
|     3        X | 9.24273E-03  X | 2.00245E-01    |                 |     1           |
|     4        X | 3.87289E-03  X | 8.37624E-02    |                 |     1           |
```

Exemple 7.4-3._ Extraits de fichier de commande et message sur un problème non linéaire avec multiples second membres (feti005b.comm/mess).

Concluons ce chapitre par deux séries de tests issues de[Boi05] qui illustrent bien les écarts de comportement de l'algorithme et de performances de son implantation dans *Code_Aster*.

Avec le cube «universitaire», FETI a du «grain à moudre» car les étapes de factorisation sont prédominantes. Les matrices de rigidité «remplissent[55]» beaucoup (facteur de remplissage de 90) du fait du caractère massif du maillage. C'est l'inverse avec le carter de moteur qui comporte beaucoup de zones minces dont les nœuds comportent peu de voisins (facteur de remplissage de14). Donc FETI ne gagne pas assez sur l'étape de factorisation pour compenser les pertes des descentes-remontées (en creux *via* la multifrontale) à chaque itération. La multifrontale sur tout le domaine n'effectue, elle, qu'une descente-remontée !





Figures 7.4-1._ Calculs mécaniques linéaires sur les cas-tests du cube (0.6 M degrés de liberté) et du carter (0.5 M degrés de liberté). Gains que procure FETI parallèle (20 sous-domaines) par rapport à la méthode par défaut du code (multifrontale séquentielle). Courbes de gains en temps «elapsed» et en mémoire RAM par processeur, suivant le nombre de processeurs appelés. Tests effectués sur le cluster du LaMSID.

55 La factorisée globale est 90 fois plus imposante que la matrice de rigidité initiale.

8 Conclusions et perspectives

Dans le cadre des simulations thermo-mécaniques avec *Code_Aster* , l'essentiel des coûts calcul provient souvent des systèmes linéaires. Pour y pallier, il peut alors être judicieux de scinder le maillage initial en plusieurs sous-domaines sur lesquels vont être conduits des calculs élémentaires, des assemblages et des résolutions parallèles plus efficaces: c'est la **Décomposition de Domaine** (**DD**). Parmi les méthodes DD envisageables, *Code_Aster* a choisi d'intégrer la méthode **FETI** .

Ce document essaye de donner au lecteur les **limitations théoriques et pratiques de l'algorithme FETI-1 et d'autres approches concurrentes en calcul hautes performances** pour la mécanique des structures. L'accent a aussi été mis sur leurs **déclinaisons dans *Code_Aster*** (opérateurs `MACR_ELEM_STAT`, `DEFI_PART_FETI` et `SOLVEUR/METHODE='MULT_FRONT'/'MUMPS'/'PETSC'`) et les perspectives fonctionnelles et logicielles qu'elles ouvrent. Cependant il faut garder à l'esprit que **FETI est un solveur de recherche**, donc encore **en phase de validation et de fiabilisation**.

Car les développements autour de FETI ont «re-défriché» des terrains logiciels qui avaient déjà fait l'objet de travaux: le multidomaine et le parallélisme. Un des leitmotivs de ce chantier logiciel a été justement de réutiliser, autant que faire se peut, l'expérience acquise sur ces thématiques. Un autre souci, a été de s'insérer dans l'architecture du code sans briser sa cohérence et en essayant de respecter ses exigences en termes d'utilisation (modularité, paramétrisation, robustesse, complétude) et de qualité logicielle (validation, documentation). **Cette volonté de ne pas révolutionner la structure du code et son flot de données et de gérer, dans une seule version l'*Aster* standard séquentiel et le multidomaine parallèle, en cherchant plus la complétude et la robustesse que les performances, a conduit à des contorsions logicielles et algorithmiques pénalisantes.**

Le solveur FETI a été éprouvé sur différentes géométries en changeant de plate-forme et de contexte mécanique. En linéaire, les gains (par rapport à la méthode par défaut du code, la multifrontale) en mémoire et en temps sont souvent intéressants, de l'ordre de plusieurs dizaines de pourcents suivant le nombre de processeurs (cf. figure 8.1). En non-linéaire, suivant le nombre de pas de temps, la fréquence de réactualisation de la matrice tangente, les archivages et le nombre de variables internes, les gains sont moins assurés.

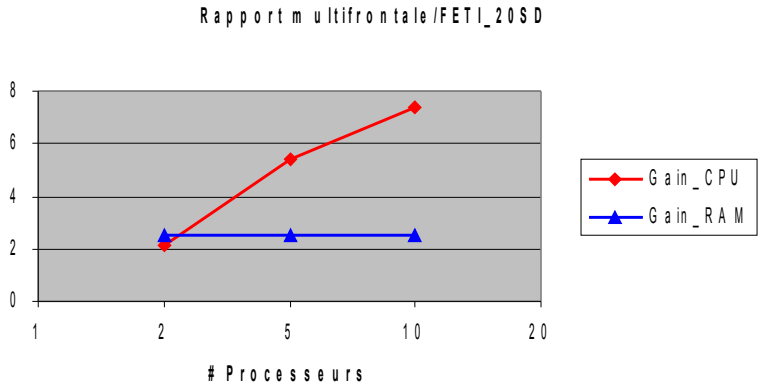


Figure 8.1._ Calcul mécanique linéaire sur le cas-test de tuyauterie de la figure 1.2.-1 (1.7 M degrés de liberté). Gains que procure FETI parallèle (20 sous-domaines) par rapport à la méthode par défaut du code (multifrontale séquentielle). Courbes de gains en temps «elapsed» et en mémoire RAM par processeur, suivant le nombre de processeurs appelés. Tests effectués sur le cluster du LaMSID.

Fort de cette expérience de développement, on peut donc apporter quelques «bémols» à l'argumentaire qui a présidé au choix de la méthode (cf. §5.3):

Argument	Constat
Meilleures performances que les solveurs mono-domaine et les autres méthodes DD.	Les benchmarks de la littérature ont été conduits souvent sur des solveurs mono-domaine SKYLINE moins optimisés que la multifrontale de *Code_Aster*. Celle-ci est particulièrement adaptée

Argument	Constat
	pour traiter les Lagranges et profiter de la gestion «out-of-core» de la mémoire.
Scalabilité.	Scalabilité parallèle difficile à réaliser avec une vision «maître-esclave» et les actuels traitements séquentiels (problème grossier, réorthogonalisation, redémarrages).
Préconditionneur efficace et peu coûteux.	Parfois décevant suivant la qualité du découpage
Méthode d'appariemment de calculs, solveur de recherche.	Pas encore exploité avec FETI. Exploité en linéaire avec Schur direct primal (MACR_ELEM_STAT).
Non prise en compte des interfaces de mesure nulle.	Pas exploité (vision «maître-esclave»).
Raccord de maillages/modèles.	Pas exploité.
Traitement des Lagranges généralisés.	En conflit et redondant avec les traitements déjà prévus par le code.
Implantation dans des grands codes de mécaniques de structures.	A part ZEBULON et SALINAS[Dur08], peu de communications sur le sujet. On ne sait pas toujours discerner si FETI est employé en «boîte noire» sur la plupart des études ou si il s'agit toujours d'un solveur de recherche en développement.
Foisonnement de travaux académiques de la communauté «computational mechanics».	Sessions dédiées de conférences et numéros spéciaux de revues.
Large périmètre d'utilisation.	En mécanique, en électromagnétisme, en neutronique…
«Relative» facilité d'implantation dans un code existant.	Jugement plus nuancé de personnes rencontrées ayant déjà travaillé sur le sujet. Toujours pas de librairie publique permettant de «pluger» rapidement FETI dans un code. Difficultés pour générer un flot de données indépendantes, qui soit pérenne, efficace, robuste et parallèle dans un code de l'ampleur de *Code_Aster* (cf. §7.1).

Tableau 8.1._ Argumentaire revu et corrigé du choix de l'implantation de FETI dans Code_Aster.

D'un point de vue purement logiciel, ce chantier aura bousculé *Code_Aster* en le faisant rentrer de plein pied dans les problématiques que pose le HPC pour les codes nativement séquentiels: distribution des données et des traitements, parallélisme algorithmique par envoi de message, adhérence aux librairies MPI.

Le **solveur FETI a donc encore des progrès à faire pour être compétitif et fiable**. Cependant, bon **nombre de ses chemins logiciels et de son retour d'expérience sont réexploités pour tirer partie des librairies externes MUMPS et PETSc.** Avec ce parallélisme numérique moins ambitieux que le mécanique, mais plus robuste et plus générique, on bénéficie ainsi de l'expertise et du retour d'expérience de ces produits et de leurs équipes. Ce parallélisme peut d'ailleurs profiter, surtout en non linéaire, d'un cumul avec le parallélisme informatique déployé en amont du solveur sur les calculs élémentaires et les assemblages. C'est l'approche «MUMPS»[U2.08.03/U4.50.01][R6.02.03] conseillée en cas de problème de robustesse et/ou de limitation de périmètre d'utilisation de FETI.

Cependant les gros besoins en mémoire sont souvent le point faible de ces librairies. Pour exhiber un parallélisme important (>100 processeurs), les approches hybrides semblent donc, pour l'instant, la seule solution viable en mécanique des structures. Les travaux en cours sur le sujet (produits HIPS et MaPHyS, maquette CAAY, préconditionneur multigrille) devraient permettre d'alimenter la réflexion et d'orienter les futurs développements.

9 Bibliographie

9.1 Livres/articles/proceedings/thèses...

[Che05] K.Chen. *Matrix preconditioning techniques and applications* . Ed. Cambridge University Press (2005).
[DV99] I.S.Duff & H.A.Van Der Vorst. *Developments and trends in the parallel solution of linear systems* . Rapport interne Rutherford Appleton Lab RAL-TR-1999-027 (1999).
[Duf06] I.S.Duff et al. *Direct methods for sparse matrices* . Ed. Clarendon Press (2006).
[Dur08] C.Durand et al. *Calcul haute performance avec Code_Aster : état des lieux et perspectives* . Note interne EDF R&D HT-62/08/01339 (2008).
[Esc92] Y.Escaig. *Décomposition de domaine multiniveau et traitements distribués pour la résolution de problèmes de grandes tailles* . Thèse de Doctorat, UTC (1992).
[Far00] C.Farhat, K.Pierson & M.Lesoinne. *The second generation FETI methods and their application to the parallel solution of large-scale linear and geometrically non-linear structural analysis problems* . Comput. Methods Appl. Mech. Engrg., 184 pp333-374 (2000).
[Far02] C.Farhat et al. *Salinas, a scalable software for high-performance structural and solid mechanics simulations* . Preprint du Gordon Bell Award (2002).
[FR91] C.Farhat & F.X.Roux. *A method of finite element tearing and interconnecting and its parallel solution algorithm* . Internat. J. Numer. Methods Engrg., 32-6 pp1205-1227 (1991).
[FR94] C.Farhat & F.X.Roux. *Implicit parallel processing in structural mechanics* . Comp. Mech. Adv., 2 pp1-124 (1994).
[Fos95] I.Foster. *Designing and building parallel programs : concepts and tools for parallel software engineering* . Ed. Addison-Wesley (1995).
[FPa03] Y.Fragakis & M.Papadrakakis. *The mosaic of high performance domain decomposition methods for structural mechanics: Part I & II* . Comput. Methods Appl. Mech. Engrg. 192 pp3799-3830 (2003) et 193 pp4611-4662 (2004).
[Gol96] G.Golub & C.Van Loan. *Matrix computations* . Ed. Johns Hopkins University Press (1996).
[Gos03] P.Gosselet. *Méthodes de décomposition de domaine et méthodes d'accélération pour les problèmes multichamps en mécanique non linéaire* . Thèse de Paris VI (2003).
[GRe06] P.Gosselet & C.Rey. *Non overlapping domain decomposition methods in structural mechanics* . Arch. Comput. Meth. Engrng., 13-4 pp515-572 (2006).
[Kru06] J.Kruis. *Domain decomposition methods for distributed computing* . Ed. Saxe-Coburg (2006).
[Las98] P.Lascaux & R.Théodor. *Analyse numérique matricielle appliquée à l'art de l'ingénieur* . Ed. Masson (1998).
[LeT91] P.Le Tallec, Y.H. De Roeck & M.Vidrascu. *Domain decomposition methods for large linearly elliptic 3D problems* . J.Comp. Appl. Math., 34 pp93-117 (1991).
[LeT94] P.Le Tallec. *Domain decomposition methods in computational mechanics* . Comp. Mech. Adv., 1 pp121-220 (1994).
[Mag07] F.Magoulès. *Mesh partitioning techniques and domain decomposition methods* . Ed. Saxe-Coburg (2007).
[Meu99] G.Meurant. *Computer solution of large linear systems* . Ed. Elsevier (1999).
[Qui03] M.J.Quinn. *Parallel programming in C with MPI and OpenMP* . Ed. Mac Graw Hill (2003).
[Rix02] D.J.Rixen. *Extended preconditionneurs for the FETI method applied to constrained problems.* Int. J. Numer. Meth. Engng., 54 pp1-26 (2002).
[Rou89] F.X.Roux. *Méthode de decomposition de domaine à l'aide de multiplicateurs de Lagrange et application à la résolution en parallèle des équations de l'élasticité linéaire* . Thèse de Paris VI (1989).
[Saa03] Y.Saad. *Iterative methods for sparse matrices* . Ed. PWS (2003).
[Smi96] B.Smith, P.Bjortad & W.Gropp. *Domain decomposition, parallel multilevel methods for elliptic PDE* . Ed. Cambridge University Press (1996).
[TWi05] A.Toselli & O.Widlund. *Domain decomposition methods – Algorithms and theory* . Ed. Springer (2005).

9.2 Rapports/compte-rendus EDF

[Anf03] N.Anfaoui. *Une étude des performances de Code_Aster: proposition d'optimisation*. Stage de DESS de mathématiques appliquées de PARIS VI (2003).
[Ass07] A.Assire. *Bilan du chantier logiciel FETI dans Code_Aster: calculs sur cas industriels et comparaison aux autres solveurs* . Note interne EDF R&D HT-62-2007-00330 (2007).
[Ber97] J.Y.Berthou. *Parallélisation du calcul des termes élémentaires dans Code_Aster version NEW4* . Note interne EDF R&D HI-76/97/027 (1997).
[Boi03] O.Boiteau. *Décomposition de domaine et parallélisme en mécanique des structures: état de l'art et benchmark pour une implantation raisonnée dans Code_Aster* . Note interne EDF R&D HI-23/03/009 (2003).
[Boi05] O.Boiteau. *Quelques tests de FETI parallèle dans Code_Aster* . Note interne EDF R&D HI-23/05/044 (2005).
[Boi07] O.Boiteau. *Bilan du chantier logiciel FETI dans Code_Aster* . Note interne EDF R&D HI-23/07/018 (2007).
[Boi07b] O.Boiteau. Intégration de MUMPS parallèle «distribué» dans *Code_Aster* . Note interne EDF R&D HI-23/07/03167 (2007).
[Boi07c] O.Boiteau. *Réunions et livrables de l'ANR SOLSTICE* . Compte-rendu interne EDF R&D CR-I23/2007/05 et 035, CR-I23/2008/012 + onglet dédié dans la base Notes SINETICS.
[Boi08] O.Boiteau. *Activation des fonctionnalités «Out-Of-Core» de MUMPS dans Code_Aster* . Compte-rendu interne EDF R&D CR-I23/2008/047 (2008).
[Des07] T.DeSoza. Evaluation et développement du parallélisme dans *Code_Aster* . Rapport de Master ENPC (2007).
[Gom96] A.Gomez, C.Caremoli & P.Schoenberger. *Etude de faisabilité de la parallélisation du Code_Aster sur CRAY C98* . Note interne EDF R&D HI-76/96/023 (1996).
[LeA05] P.Legallou et A.Assire. *Intégration du logiciel de partitionnement SCOTCH dans Code_Aster* . Rapport de Master CSA Bordeaux I (2005).
[Tar04] N.Tardieu. *Parallélisation de MEF++, principes et performances* . Rapport interne du GIREF-Université de Laval (2004).

9.3 Ressources Internet

[Ddm] Portail web international en décomposition de domaine: http://www.ddm.org .
[Don] Site web de Jack Dongarra: http://www.netlib.org/utk/people/JackDongarra.
[Met] Site web de l'outil METIS: http://glaros.dtc.umn.edu/gkhome/views/metis.
[Mpi] Site web de l'implémentation MPICH de MPI: http://www-unix.mcs.anl.gov/mpi.
[Omp] Site web d'OpenMP: http://www.openmp.org.
[Orap] Site web de l'association ORAP (ORganisation Associative du Parallélisme commune au CNRS, CEA et INRIA): http://www.irisa.fr/orap.
[Sco] Site web de l'outil SCOTCH: . http://www.labri.fr/perso/pelegrin/scotch/scotch_fr.html.

10 Description des versions du document

Version Aster	Auteur(s) Organisme(s)	Description des modifications
V7.4	O.BOITEAU EDF/R&D/SINETICS	Texte initial
v10.4	O.BOITEAU EDF/R&D/SINETICS	**Beaucoup de corrections formelles dues aux portages .doc/.odt**; Mise à jour sur le parallélisme, sur MUMPS et sur SCOTCH.