Assemblage parallèle d'une matrice et/ou d'un second membre : Application à la Parallélisation d'un Code de Mécanique des Structures

David Goudin

LaBRI
Université Bordeaux I – ENSERB
CNRS UMR 5800
351, cours de la Libération
33405 Talence Cedex

Résumé

Cet article décrit un algorithme d'assemblage parallèle pouvant être utilisé lors de la phase de résolution de grands systémes linéaires creux. La première partie traite des problémes posès par un code de mécanique des structures nommé OSSAU développé au CEA. La deuxiéme partie est consacrée à la description de notre algorithme et à son intégration dans la chaîne logicielle EMILIO, développée au LaBRI sein du théme ALiENor.

Mots-clés : assemblage parallèle, matrices creuses, éléments finis

1. Introduction

Le code OSSAU du CEA/CESTA est un code industriel vectorisé de mécanique des structures, non linéaire en temps et en deux ou trois dimensions d'espace. C'est un code de calcul de structure par éléments finis qui résout les équations de la mécanique des milieux continus en statique ou en dynamique (quand les effets d'inertie sont négligeables) ; il calcule donc la réponse d'une structure à diverses sollicitations (forces pontuelles, pressions, ...). La réponse de la structure est caractérisée par les déplacements aux nœuds de la discrétisation par éléments finis ; la charge totale appliquée à la structure est divisée en un certain nombre d'incréments appliqués successivement, chaque incrément résout le probléme non linéaire par une technique dite de Newton-Raphson qui suit un enchaînement d'itérations :

$$K(U_i) \cdot \Delta U_i = R(U_i) \ , \tag{1}$$

où ΔU_i représente les déplacements aux nœuds et $K(U_i)$ la matrice globale de rigidité. Cette matrice symétrique définie positive est construite par un assemblage des matrices élémentaires associées aux éléments du maillage:

$$K(U_i) = \sum_{e \in \Omega} Ke(U_i) \ , \tag{2}$$

Les contraintes appliquées à la structure peuvent être telles que d'une itération à l'autre il peut y avoir de grandes différences de complèxité (grands déplacements, plastification, etc...) et donc de temps au niveau des calculs des matrices élémentaires. Dans ce cas, le problème devient non linéaire (on peut, pour simplifier, considérer que la matrice K de l'itération $p+1$ est fonction du vecteur u calculé à l'itération p). Le calcul des contraintes dans la structure permet de prévoir la destruction de celle-ci. Le vecteur second membre est lui aussi calculé par un assemblage de seconds membres élémentaires associés aux éléments du maillage :

$$R(U_i) = \sum_{e \in \Omega} Re(U_i) \ . \tag{3}$$

2. Problèmatique de l'Assemblage

Comme on l'a vu précédemment, l'assemblage consiste à calculer les matrices élémentaires de chaque élément du maillage et additionner ensuite cette matrice élémentaire á la matrice de rigidité globale K. Une matrice élémentaire décrit les propriétés physiques de l'élément et la complèxité de son calcul dépend essentiellement des contraintes appliquées à la structure. Dans certains cas tests que nous avons à traiter (cas du CEA par exemple), le coût d'assemblage de la matrice de rigidité globale K est supérieur à celui de la factorisation de la matrice. Cette disproportion est accentuée lors de l'apparition d'un phénomène de plastifification de la structure, en effet les calculs des matrices élémentaires constituent alors la phase la plus coûteuse de toute la chaîne présentée ci-après.

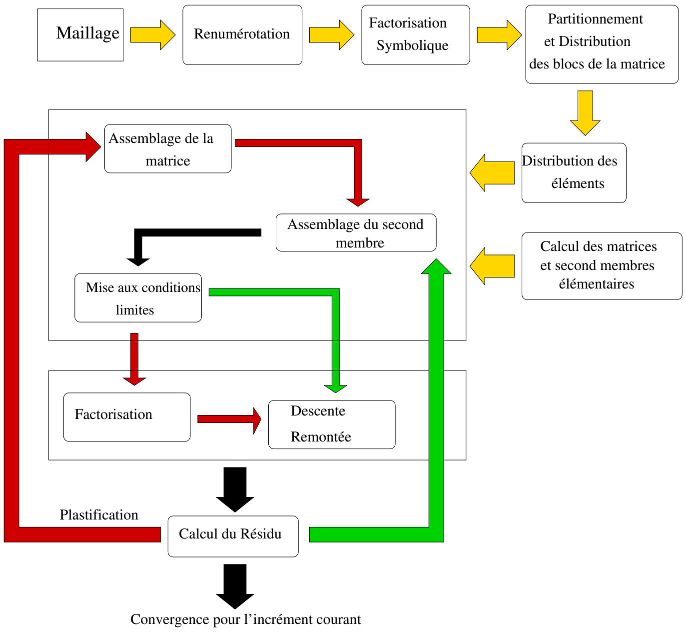

FIG. 1 – *La chaîne EMILIO.*

L'assemblage est une étape de la chaîne logicielle parallèle EMILIO développée au LaBRI par le thème ALiENor et qui apporte différentes solutions aux problèmes de la résolution parallèle de grands systèmes linéaires creux par méthodes directes. EMILIO se décompose en plusieurs étapes :

- la renumérotation de la matrice, qui permet de minimiser le remplissage de celle-ci lors de la factorisation numérique ;
- la factorisation symbolique, qui permet de connaître à l'avance la structure de la matrice factorisée et donc d'estimer les volumes de données et de calculs nécessaires à la factorisation numérique. Afin d'économiser la place mémoire et d'accélérer le calcul de la factorisation symbolique, celle-ci peut s'effectuer par blocs;
- le placement des éléments du maillage sur les processeurs. Afin de maximiser la localité des calculs sur les éléments et de minimiser les communications, il s'agit de placer les éléments sur les processeurs contenant le plus de leurs nœuds ;
- l'assemblage des éléments et du second membre, pour remplir la matrice à factoriser et le ou les vecteurs de second membre.

- le placement des blocs de la matrice sur les processeurs de la machine parallèle, en tenant compte de leurs dépendances mutuelles, afin d'obtenir un parallélisme maximum lors de la factorisation numérique ;
- la factorisation numérique proprement dite, nous utilisons une factorisation LDL^T plutôt que LL^T, afin de pouvoir traiter des matrices à coefficients complexes (cas du CEA) ;
- la résolution du système diagonal ainsi construit (phase de descente-remontée).

Avant de présenter les solutions retenues pour la distribution des éléments, l'assemblage de la matrice de rigidité et l'assemblage du second membre, nous devons tout d'abord présenter les phases de renumérotation et de distribution des blocs de la matrice.

3. Renumérotation

Cette renumérotation est choisie de telle sorte que la mise à jour des termes non-diagonaux dans l'ordre de la matrice permutée obtenue par renumérotation provoque beaucoup moins de remplissage et d'opérations que si la matrice avait été conservée telle quelle (le remplissage est l'ensemble des termes nuls de A qui deviennent non-nuls dans la matrice factorisée L).
Notre approche consiste à coupler les deux techniques de renumérotation les plus connues : *degré minimum* et la *dissection emboîtée* avec quelques modifications pour l'algorithme de degré minimum initial. En effet, nous ajoutons aux sous-graphes passés à l'algorithme de degré minimum une description topologique de leur voisinage immédiat. Cette information supplémentaire, appelée « halo », est alors exploitée par un algorithme de degré minimum approché, modifié pour ne numéroter que les sommets appartenant au graphe. L'arbre d'élimination (c'est-à-dire l'arbre des super-variables) est alors construit. Nous avons appelé cet algorithme de degré minimum modifié Halo-AMD, ou HAMD.
Afin de réaliser la factorisation numérique de façon efficace, il est nécessaire d'utiliser des routines de calcul par blocs de type BLAS 3, sur des blocs de taille suffisante. De fait, il faut amalgamer les colonnes de la matrice renumérotée afin d'obtenir des blocs suffisament larges, sans pour autant pénaliser le remplissage. L'algorithme d'amalgamation que nous utilisons n'opère que sur les sous-graphes passés à l'algorithme de degré minimum, car les séparateurs sont presque toujours de taille suffisante.
La partition du graphe en super-variables est obtenue en fusionnant la partition induite par les séparateurs avec la partition des super-variables amalgamées localement dans chaque sous-graphe.
Notre algorithme de renumérotation définit donc, en complément du calcul d'une permutation des inconnues, une partition des colonnes de la matrice en blocs colonnes. Cette partition comprend les séparateurs calculés par l'algorithme de Dissections Emboîtées, et les super-variables amalgamées dans chaque sous-graphe grâce à l'arbre d'élimination construit par l'algorithme HAMD.

4. Repartitionnement et distribution des blocs

Certains solveurs numériques prennent entièrement à leur charge le séquencement dynamique des calculs de factorisation des blocs sur les processeurs de l'architecture parallèle, en se basant sur le placement des blocs préalablement calculé par un algorithme de pré-traitement de type « *subtree-to-subcube* ».

L'approche que nous avons choisie consiste à décharger entièrement le solveur de cette tâche, et donc à pré-calculer totalement l'ordonnancement des calculs de factorisation des blocs de façon statique, en simulant le plus précisément possible le comportement de l'architecture parallèle utilisée. Notre algorithme comprend deux phases distinctes.

La première est la phase de repartitionnement, dans laquelle les plus gros blocs sont redécoupés afin de pouvoir faire participer plusieurs processeurs à leur traitement. On exhibe ainsi un parallélisme de type « calcul parallèle sur des blocs pleins », qui s'applique principalement aux plus gros séparateurs de l'arbre d'élimination, à la fin de la factorisation. Au cours de cette phase, on parcourt l'arbre d'élimination, et on calcule pour chaque bloc un ensemble de processeurs candidats susceptibles de l'accueillir. Cette notion de processeur candidat est indispensable pour préserver la bonne localité des communications ; celle-ci suit la logique de l'arbre d'élimination, car ne seront candidats pour un nœud de l'arbre d'élimination que les processeurs ayant participé à la factorisation du sous-arbre dont il est la racine.

La deuxième phase est la phase de distribution proprement dite, dans laquelle on attribue à chaque bloc colonne le processeur qui le calculera, parmi ses processeurs candidats. Pour ce faire, on remonte l'arbre d'élimination des feuilles vers la racine et, à mesure que l'on distribue les nœuds de l'arbre sur les processeurs, on calcule le temps (estimé) pris par leur factorisation et l'envoi de leurs contributions. Ainsi, le choix de placement d'un bloc colonne sur l'un de ses processeurs candidats est décidé en connaissant la date à laquelle potentiellement le processeur aura terminé de le traiter. Le processeur élu sera alors celui qui pourra avoir fini de traiter le bloc colonne le plus tôt.

5. Distribution des éléments

La distribution des éléments est directement induite par la distribution des blocs de la matrice. Elle est en 1D car, pour l'instant, la phase d'assemblage de la matrice opère en 1D. Cette distribution initiale des éléments est non seulement une « très bonne » condition initiale pour les assemblages parallèles de la boucle non linéaire, mais elle permet aussi de connaître à l'avance les différentes communications à effectuer dans le module d'assemblage. Enfin, cette distribution sera remise en cause lors de la régulation dynamique de charge pour l'assemblage, cette dernière s'opérant par migration d'éléments entre différents processeurs lors des phases de plastification.

Pour effectuer la distribution des éléments, il est nécessaire de connaître non seulement la distribution 1D des blocs colonnes, mais aussi des informations relatives au maillage : géométrie de chaque élément, numéros des nœuds composant cet élément, et nombre de degrés de liberté par nœud. Le traitement effectué se décompose en deux parties :

- l'attribution des éléments totalement locaux à un processeur selon un critère d'équilibrage de charge en termes de nombre d'éléments (par élément local à un processeur, on entend un élément dont tous les nœuds appartiennent à des blocs locaux à ce processeur au sens de la distribution des blocs colonnes) ;
- la répartition des éléments restants.

L'algorithme que nous utilisons équilibre la masse de calculs pour chaque processeur, c'est-à-dire que chaque processeur doit calculer un nombre équivalent de matrices élémentaires. Les communications ne sont pas prises en compte pour l'équilibrage, car elles sont supposées recouvertes par les calculs.

Comme la phase de distribution des nœuds prend en compte les coûts de calcul associés aux nœuds, et que ces coûts sont inégaux suivant la connexité du nœud dans le maillage, il peut en résulter un nombre de colonnes très différent d'un processeur à l'autre, ce qui entraîne un déséquilibre au niveau du nombre d'éléments totalement locaux. Pour s'adapter à ce déséquilibre inévitable, l'algorithme procède en deux étapes :

- dans une première phase, il construit, pour chaque élément, la liste des processeurs sur lesquels ont été placés les nœuds composant cet élément. Si ce nombre est égal à 1 et que le nombre courant d'éléments possédés par le processeur n'est pas supérieur à une certaine fraction du nombre total d'éléments (fixée au préalable), on place l'élément sur ce processeur, sinon on stocke dans un tableau le numéro de l'élément ainsi que le nombre de propriétaires. Il est nécessaire de ne placer qu'une fraction des éléments locaux afin de ne pas déséquilibrer le placement dès cette phase si un processeur possède beaucoup plus de nœuds que les autres, et donc beaucoup plus d'éléments locaux, sans que ce déséquilibre puisse être compensé par le placement judicieux des autres éléments ;
- ensuite, on trie le tableau regroupant les éléments restant à traiter selon le nombre de processeurs propriétaires des nœuds. On détermine alors, pour chaque élément, sur quel processeur il doit être placé, en fonction de la charge courante de chaque processeur candidat, et en privilégiant le processeur qui possède le plus de nœuds appartenant à l'élément considéré. Pour les éléments locaux non placés lors de l'étape précédente, on cherche à minimiser la distance, en terme de numéros logiques de processeurs, entre le processeur possédant les nœuds de l'élément et le processeur sur lequel celui-ci sera placé, afin d'augmenter la localité des communications.

Notre algorithme de placement détermine pour chaque processeur la masse de calculs à effectuer, mais aussi la quantité de communications à recevoir et à envoyer lors de la phase d'assemblage proprement dite.

6. Algorithme d'assemblage

6.1. Assemblage de la matrice de rigidité

Cette phase utilise les informations qui lui sont données par les deux distributions : la distribution des blocs colonnes de la matrice et la distribution des éements du maillage. Cette dernière définit non seulement la liste des éléments dont il faudra calculer la matrice élémentaire mais aussi les tailles des messages à envoyer, le nombre de messages à recevoir, etc...

On commence par traiter les éléments qui ont été attribués au processeur courant mais dont un ou plusieurs nœuds sont dans des blocs appartenant à un autre processeur. Avant de calculer la matrice élémentaire d'un élément non local, on teste si un message n'est pas en attente de réception, c'est-à-dire si un autre processeur a envoyé totalement ou partiellement la matrice élémentaire d'un élément apportant une contribution à des blocs possédés par le processeur courant, celui-ci ne possédant pas l'élément lui-même. Si un ou plusieurs messages sont en attente, on fait une réception bloquante (on n'accepte de traiter qu'un nombre N de messages pour que le processeur ne soit pas "bloqué" en attente) et on effectue, grâce au contenu du message,les additions de contributions dans le ou les blocs de la matrice concernés ; sinon, on calcule la matrice élémentaire d'un élément non local au processeur courant (mais lui ayant été attribué par la distribution des éléments), on prépare les messages à envoyer aux processeurs concernés, et on effectue les additions de contributions locales. Pour les mises à jour locales, on a pris soin de trier au préalable les nœuds de l' élément afin que les accès aux contributions locales se fassent plus rapidement à partir du dernier bloc accédé par la précédente mise à jour. Il faut noter que cette mise à jour des coefficients de la matrice est effectuée seulement sur la partie triangulaire inférieure, car la matrice est symétrique définie positive.

Lorsque le traitement des éléments "non totalement" locaux est terminé, on traite alors les éléments totalement locaux, c'est-à-dire calculer leur matrice élémentaire, puis faire l'addition des contributions dans la partie de matrice correspondante, et qui est donc locale au processeur. Avant de calculer une matrice élémentaire, on teste si un message contenant des mises à jour locales n'est pas en attente de réception, si tel est le cas il est traité en priorité.

Lorsqu'un processeur a fini de traiter tous ses éléments,, li lance une réception bloquante pour attendre, si nécessaire, des messages de contributions envoyés par les autres processeurs, afin de finir de mettre à jour les blocs locaux de la matrice.

6.2. Assemblage du second membre

Les deux phases d'assemblage peuvent être exécutées séparément ou combinées, en particulier il est nécessaire lors de certains tests de ne réassembler que le second membre et non la matrice de rigidité, avant de lancer la résolution.

En ce qui concerne l'assemblage du second membre, nous avons décidé d'attribuer à un processeur p tous les seconds membres correspondants aux blocs colonnes (en 1D) déjà placés sur p par le module de placement des blocs colonnes de la matrice. Cet assemblage du vecteur second membre global R suit les mêmes schémas de communication et de calcul que pour la matrice de rigidité K. Lorsque les deux assemblages parallèles doivent être enchaînés, un algorithme qui "couple" les deux phases est exécuté, c.a.d que l'on a exploité le fait qu'ils ont les mêmes schémas d'exécution pour implémenter un seul algorithme pour les deux assemblages. Grâce à ce couplage, les messages inter-processeurs contiennent non seulement les matrices élémentaires mais aussi les seconds membres élémentaires, ce qui signifie que le nombre de communications sera le même pour un assemblage "simple" (matrice K ou vecteur R) que pour les deux assemblages mixés; dans le deuxième cas, la masse de calculs sera plus importante donc le recouvrement calculs-communications sera d'autant plus facile à exploiter.

7. Tests

Le tableau suivant résume les résultats obtenus pour l'assemblage de la matrice de rigidité K sur une grille 2D carrée de dimension 511 (1 DDL parnœud, 260100 éléments, 261121 colonnes), sur les cubes 3D carrés de dimension 31 (1 DDL par nœud, 27000 éléments, 29791 colonnes) et 39 (1 DDL par nœud, 54872 éléments, 59319 colonnes) et sur plusieurs cas tests fournis par le CEA : COLOGB15 (3 DDL par nœud, 3476 éléments, 10428 colonnes), COLOGB30 (3 DDL par nœud, 1260 éléments, 20373 colonnes) et COLOGB75 (3 DDL par nœud, 3150 éléments, 50208 colonnes). le cas de Cologne. Le calcul d'une matrice élémentaire est simulé avec des temps de 0.01 s, 0.05 s et 0.1 s ; ces temps ont été choisis après l'analyse de différentes exécutions du code OSSAU séquentiel original. Les tests ont été effectués sur l'IBM SP2 du CNUSC (192 nœuds), avec des nœuds fins Power2SC 120MHZ ayant 256 Mega de mémoire.

Nom	Nombre de processeurs				Elt time
	4	8	16	32	
COLOGB15	1.82	0.97	0.63	0.46	0.01 s
COLOGB30	3.72	2.10	0.96	0.52	-
COLOGB75	9.01	4.60	2.56	1.63	-
GRID511	667.44	336.80	167.43	83.88	-
CUBE31	69.83	35.02	17.70	8.93	-
CUBE39	141.86	71.16	35.77	18.08	-
COLOGB15	8.45	4.46	2.50	1.40	0.05 s
COLOGB30	16.56	8.55	4.58	2.31	-
COLOGB75	41.13 41	20.72	10.67	5.88	-
GRID511	3305.24	1655.12	830.22	407.94	-
CUBE31	344.81	172.65	86.36	43.31	-
CUBE39	700.67	350.72	175.40	87.87	-
COLOGB15	16.40	8.55	4.30	2.17	0.1 s
COLOGB30	32.42	16.77	8.43	4.28	-
COLOGB75	80.90	40.60	20.67	10.68	-
GRID511	6560.33	3290.47	1650.49	814.67	-
CUBE31	688.63	342.61	172.50	86.11	-
CUBE39	1394.14	698.16	348.74	175.10	-

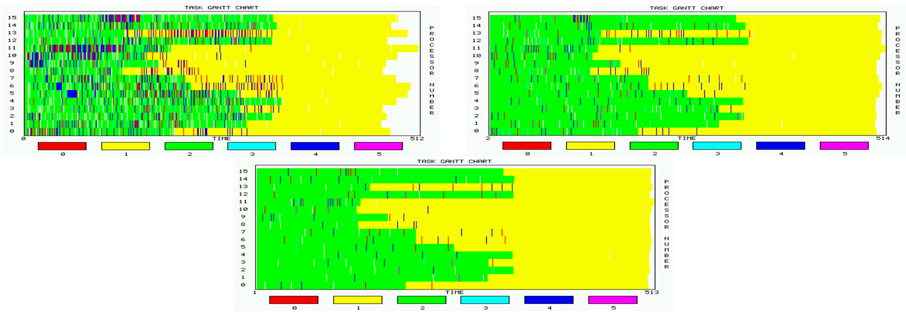

FIG. 2 – *Diagrammes de Gantt pour COLOGB75, temps = 0.01 s, 0.05 s and 0.1 s.*

8. Conclusion et Perspectives