MATMECA
Ecole d'Ingénieurs en Modélisation Mathématique et Mécanique

INRIA
SOPHIA ANTIPOLIS

Rapport de projet de fin d´études

Promotion 2003

Partitionnement de maillages sur une grille de calcul

Rodolphe LANRIVAIN

Responsables industriels :

Hervé Guillard (INRIA) et Hugues Digonnet (CEMEF)

Responsable universitaire :

Bruno Dubroca (MAB)

Remerciements

Je tiens à remercier l'ensemble des personnes que j'ai pu rencontrer lors de mon stage au sein du projet SMASH :

Hervé Guillard qui m'a accueilli dans son équipe et accompagné dans mon travail,
Hugues Digonnet qui a encadré une partie du stage avec beaucoup d´application et de disponiblité,
Monserrat Argente pour sa gentillesse et sa bonne humeur,
Les autres stagiaires et ingénieurs experts ...

Table des matières

Introduction

A l'heure actuelle, le calcul parallèle est devenu incontourable en calcul scientifique. A l'origine effectué sur des équipements spécialisés, le calcul parallèle devient de plus en plus présent dans les laboratoires et les industries. D'autre part, le développement d'internet et des multiples réseaux de communication se poursuit à une vitesse importante. Nous sommes donc dans une situation où *à la fois* les ressources en CPU et la vitesse des réseaux s'accroissent très rapidement. Il semble en fait (voir [4]) que les bandes passantes des réseaux augmentent plus vite que le CPU. L'ensemble de ces faits conduit à penser qu'un futur possible du calcul scientifique pourrait consister d'unités de calcul géographiquement distantes connectées par des réseaux très rapides. En conséquence, les calculs parallèles pourrait demain être distribués à travers un réseau très hétérogènes de resources de calcul que l'on appelle une "grille".

Utiliser efficacement une telle grille, demande de résoudre des problèmes d'allocations de ressources difficile. La formulation théorique habituelle de ce problème consiste en ce que les informaticiens appellent un problème de *mapping* : le programme parallèle à exécuter est assimilé à un graphe pondéré (le graphe source ou le graphe d'exécution) où l'on associe à chaque sommet du graphe un poids de calcul et à chaque arête un certain coût de communication. L'ensemble des ressources de calcul est aussi modélisée par un "graphe de ressources ou graphe de l'architecture". A chaque unité de calcul (sommet du graphe) est associée un poids correspondant à la puissance de calcul du processeur et à chaque arête un poids correspondant à la vitesse de la communication. Un "mapping" du graphe source sur le graphe de l'architecture consiste en une application de l'ensemble des sommets du graphe source sur l'ensemble des sommets du graphe de l'architecture et le problème consiste à trouver le meilleur mapping, c'est à dire celui qui minimise le temps d'exécution du programme.

Pour les applications de type élément ou volumes finis qui demandent l'utilisation d'un maillage, le graphe d'exécution peut souvent être confondu avec le maillage lui même (voir le paragraphe 1.2.2). Le problème de mapping devient alors un problème de *partitionnement* de maillage. Pour les calculs intensifs s'effectuant sur des machines parallèles, ce problème a déjà fait l'objet de très nombreuses études. Cependant, le plus souvent il a été abordé dans un contexte homogène où les unités de calcul sont non différenciées et où le coût des communications est considéré comme uniforme à travers le réseau. L'introduction des problématiques liées aux grilles de calcul oblige à revoir ces approches et à distribuer les sous maillages sur des processeurs de vitesses très différentes de façon à équilibrer la charge et à tenir compte de débits de communications très disparates.

Dans une première partie, nous présentons le calcul sur grille, ainsi que le problème du partitionnement de maillage. Ensuite, un outil développé afin de mesurer les caractéristiques de l´architecture machine est présenté. *Meshmigration*, code parallèle (MPI) developpé en C++ au CEMEF , permet le repartitionnement parallèle de maillages non structurés. L'objectif du stage est d'adapter *Meshmigration*, de manière à prendre en compte les deux principales contraintes liées à l'architecture des grilles de calcul, à savoir : les processeurs ont des vitesses de calcul hétérogènes et les débits des communications entre ces processeurs sont également hétérogènes. La troisième partie est consacrée à ce repartitionneur : on y présente la méthode ainsi que les modifications effectuées pour tenir compte d´un graphe des ressources non homogène. Enfin, nous présentons un ensemble de tests, effectués avec des configurations représentatives du calcul sur grille (architecture machine et code de calcul), et validant les améliorations apportées à *Meshmigration*.

Ce stage à été effectué dans le cadre du projet MECAGRID, qui est un projet de l'ACI-GRID 2003 du ministère de la recherche, et qui réunit l'INRIA Sophia-Antipolis, le CEMEF et l'IUSTI. Il a pour but la réalisation d'une grille de calcul régionale à partir de grappes de PC localisées en région PACA. Cette grille sera dédiée à des applications de calcul massivement parallèle en mécanique des fluides multimatériaux.

Chapitre 1

Contexte

Ce chapitre présente le contexte dans lequel se placent l'étude et les développements du stage. Dans un premier temps, le calcul sur grille est introduit en précisant les applications et le modèle de développement parallèle concernés. Nous présentons ensuite le problème du partitionnement de maillage.

1.1 Parallélisme et grille de calcul

"Réaliser des prévisions météorologiques ou climatiques, calculer le comportement aérodynamique d'un nouveau modèle d'avion, décrypter le génome d'un organisme vivant, détecter des particules élémentaires produites par un accélérateur, etc. : de telles tâches, qui nécessitent d'énormes calculs ou le traitement de quantités colossales de données, sont aujourd'hui nombreuses et variées. Elles sont aussi de plus en plus ambitieuses, donc de plus en plus exigeantes en puissance de calcul, en flux de données ou en capacité de mémoiré".
"Aussi, depuis six ou sept ans, un concept potentiellement révolutionnaire se développe : relier entre eux -via Internet en particulier - des équipements géographiquement dispersés et constituer un réseau qui cumule les capacités de calcul, de stockage, etc., de tous ses membres. Chacun de ceux-ci pourra alors disposer de la somme des ressources - puissance de calcul, mémoire, logiciels, données - apportées par tous les autres participants au réseau. Telle est l'idée de base des " grilles informatiques ", idée qui signifie une globalisation et une dématérialisation des ressources informatiques."
Cette introduction, issue d'une présentation du "GRID Computing" à l'INRIA[1], situe parfaitement les motivations qui ont amené au développement actuel des grilles de calcul. Le projet MECAGRID met actuellement en place un grille sur la région PACA, reposant sur une collaboration entre l´INRIA (Sophia et Grenoble), le CEMEF (Sophia) et l´IUSTI (Marseille). Les applications seront dédiées à la mécanique des fluides et la simulation numérique multi-matériaux.

Dans ce cadre, le modèle de développement parallèle le plus adapté est celui par échange de messages (Message Passing). Les deux grands représentants de cette famille sont les librairies PVM (Parallel Virtual Machine, [5]) et MPI (Message Passing Interface, [8]), et ce pour des raisons de portabilité et d´adaptation aux architectures à mémoire distribuée (une des caractéristique des grilles !). Notre étude se place donc dans un contexte de simulation numérique fluide/multi-matériaux, caractérisée par les méthodes éléments ou volumes finis reposant sur un maillage ; mais également dans celui d´architectures "type grille" donc hétérogènes et parfaitement adaptées à une programmation SPMD (Simple Programm Multiple Data).

[1] http ://www.inria.fr/presse/themes/gridcomputing/grid1.fr.html

1.2 Partitionnement de maillages

1.2.1 Notions générales sur les maillages

Le contexte est celui d'application type éléments ou volumes finis reposant sur un maillage. On se donne donc un maillage du domaine de calcul selon la définition suivante issue de [7].

◇ **Définition 1.1 (MAILLAGE)** *Un maillage est une discrétisation d'une partie finie de l'espace* $\mathbb{R}^d$ *(d : dimension de l'espace). Il est défini par un couple de deux ensembles finis* $(\mathcal{N}, \mathcal{E})$*. L'ensemble* $\mathcal{N}$ *est l'ensemble des noeuds. L'ensemble* $\mathcal{E}$ *est l'ensemble des éléments.*

Un noeud $n \in \mathcal{N}$ est un point de l'espace donc défini par ces coordonnées $(x_1,...,x_d)$.
Un élément $e \in \mathcal{E}$ est un ensemble fini de noeuds.
Il existe plusieurs types d´éléments pour un même maillage. Nous avons choisi de considérer les *maillages simplexes* pour lesquels la dimension des éléments est fixée par celle de l´espace. Le tableau 1.1 présente les différents éléments d´un maillage simplexe en fonction de la dimension de l´espace d.

	1D	**2D**	**3D**	**Terminologie**
d + 1	segment	triangle	tétrahèdre	élément
d	×	segment	triangle	face
d − 1	×	×	segment	arête

TAB. 1.1 – Caractérisation des éléments d´un maillage simplexe

Selon l´application, il est nécessaire de connaitre l´ensemble de ces éléments ou seulement une partie. Nous restreignons cet ensemble des éléments à l´ensemble des éléments de dimension $d+1$: l´utilisation de la terminologie "éléments du maillage" concerne donc les segments en 1D, les triangles en 2D et les tétrahèdres en 3D.
Un élément $e \in \mathcal{E}$ est alors défini par la connectivité du maillage :

$$\begin{array}{rccl} T & : & \mathcal{E} & \rightarrow & \mathcal{P}(\mathcal{N}) \\ & & e & \mapsto & T(e) = \{n_1, n_2, ..., n_{d+1}\} \end{array} \tag{1.1}$$

De même, on définit l'inverse de la connectivité :

$$\begin{array}{rccl} T^{-1} & : & \mathcal{N} & \rightarrow & \mathcal{P}(\mathcal{E}) \\ & & n & \mapsto & T^{-1}(n) = \{e_1, e_2, ...\} \end{array} \tag{1.2}$$

$T^{-1}(n)$ est ainsi l´ensemble des éléments auxquels appartient le noeud n.
Nous présentons figure 1.1 l´exemple d´un maillage 2D simplexe. Il comprend des noeuds appartenant à 1, 2, 3, 4, 5 et même 6 éléments, et est donc non-structuré.

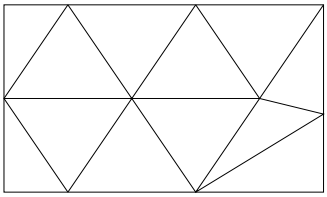


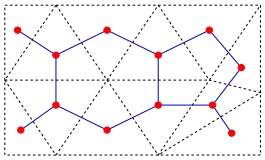
FIG. 1.1 – Maillage 2D simplexe

1.2.2 Graphes et maillages

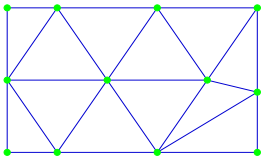
Il est possible de disposer des propriétes topologiques d´un maillage en définissant un graphe associé.

◇ **Définition 1.2 (GRAPHE)** *Un graphe G est un couple (V, Σ) de deux ensembles finis et disjoints. Les éléments de l'ensemble $V = \{v_1, v_2, ..., v_n; n \in \mathbb{N}\}$ sont appelés les sommets de G (dont la cardinalité n s'appelle l'ordre de G). Les éléments de l'ensemble $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_m; m \in \mathbb{N}\} \subset (V \times V)$ sont les arêtes de G.*

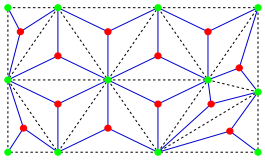
A partir d´un même maillage, il est possible de définir plusieurs graphes, caractéristiques d´une certaine topologie du maillage. Nous présentons, figure 1.2, trois exemples de graphes associés au maillage de la figure 1.1.



(a) Graphe dual



(b) Graphe nodal



(c) Graphe combiné

FIG. 1.2 – Trois graphes issus d'un même maillage (tiré de **??**)

On note :
- $G_{\text{dual}} = (\mathcal{E}, \Sigma_{ee})$, le graphe dual où les sommets du graphe sont les éléments du maillage, et chaque arête relie deux éléments ayant une face comune ;
- $G_{\text{nodal}} = (\mathcal{N}, \Sigma_{nn})$, le graphe nodal où les sommets du graphe sont les noeuds du maillage, et chaque arête relie deux noeuds appartenant à un même élément ;
- $G_{\text{combiné}} = (\mathcal{N} \cup \mathcal{E}, \Sigma_{en})$, le graphe combiné où les sommets du graphe sont les noeuds et éléments du maillage, et chaque arête relie un élément et un de ses noeuds.

Le choix d´un graphe associé dépend des propriétés du maillage dont on souhaite disposer. Comme justifié en annexe B.3, le graphe le plus représentatif du maillage, dans le cadre d´une application éléments finis, est le graphe combiné. Nous choisissons donc ce graphe comme représentation du maillage.

1.2.3 Partitionnement d´un graphe

On définit $\mathcal{P}r$ comme un ensemble fini de processeurs de cardinalité p :

$$\mathcal{P}r = \{0, 1, ..., p-1\} \tag{1.3}$$

Soit donc, un ensemble de p processeurs $\mathcal{P}r$ et un maillage du domaine de calcul.
Le partitionnement d´un maillage consiste, sous un certain nombre de contraintes, à assigner à chaque processeur un sous-maillage du maillage considéré. L´ensemble des sous-maillages définit la partition. Les contraintes se retrouvent quelle que soit la méthode de partitionnement choisie.

Citons principalement les contraintes sur :
- l'équilibrage de la charge de travail correspondant à l'étape de résolution ;
- la taille des interfaces, qui déterminera le volume des données à communiquer entre sous-domaines ;
- la régularité des sous-maillages, nécessaire pour assurer la convergence globale de certaines méthodes de résolution liées aux applications.

Compte tenu du choix d´un graphe comme représentation du maillage, partitionner le maillage revient à partitionner le graphe. On note $G = (V, \Sigma)$ le graphe issu du maillage du domaine de calcul.

◇ **Définition 1.3 (PARTITION)** *La partition d'un graphe $G = (V, \Sigma)$ sur un ensemble fini de cardinalité p est définie par l'application suivante :*

$$\pi : V \longrightarrow \{0, 1, ..., p-1\} \tag{1.4}$$

qui à un chaque sommet du graphe associe un sous-graphe "hôte".

En fait, le problème de partitionner un graphe sur p processeurs se décompose en deux étapes :

1. définition des p sous-graphes
2. assignation de chaque sous-graphe à un processeur (Mapping)

Lorsque l´architecture machine est hétérogène, l´étape de mapping est incluse dans l´étape de définition des sous-graphes, puisqu´il est nécessaire que les sous-graphes soient adaptés à l´architecture : plus un processeur est rapide, plus le sous-graphe qui lui est attribué doit être grand, plus une connexion entre deux processeurs est lente, plus l´interface entre les sous-graphes correspondants doit être réduite.
L´*architecture machine* sur laquelle l´application parallèle sera exécutée, conditionne donc le partitionnnement et l´on consacrera le chapitre 2 à son étude.

Partitionnement et repartitionnement

Dans certains codes de simulation, il est nécessaire d´effectuer un remaillage du domaine de calcul ce qui entraine une redéfinition des sous-maillages.
Le problème du repartionnement d´un graphe consiste, à partir d´une première partition π_1 du graphe, à définir une autre partition π_2. On parlera de méthodes de repartitionnement lorsque la partition initiale est prise en compte pour définir la deuxième partition.

1.3 Conclusion

Le problème de partitionnement étant connu pour être NP-complet, différentes heuristiques ont été developpées. On peut distinguer les méthodes basées directement sur des partitions de maillages de celles basées sur des partitions de graphes. On renvoie à l´annexe A pour une présentation de quelques méthodes de partitionnement.
Nous avons choisi de poursuivre le travail de Hugues Digonnet ([3]), *Meshmigration*, qui est un repartitionneur parallèle de maillages non-structurés. L´étude de la méthode sur laquelle repose ce repartitionneur, ainsi que la présentation des modifications effectuées pour l´adapter au calcul sur grille seront menées au chapitre 3.

Chapitre 2

Architecture de calcul

Comme nous l´avons introduit en présentant le partitionnement d´un graphe (§1.2.3), les caractéristiques de l´architecture de calcul conditionne la partition du graphe et sont donc des paramètres de la méthode de partitionnement.
Dans le cadre du calcul sur grille, lors d´une requête d´un utilisateur pour disposer d´un ensemble de processeurs, la répartition de ces processeurs sur les différents centres de la grille, et à l´intérieur de chaque centre, n´est pas, à priori, connue. De plus, l´utilisation d´une partie des ressources par un autre utilisateur, ce qui altèrerait les performances, est possible.
Il est donc nécessaire de disposer d´un outil de mesure de ces caractéristiques, qui soit représentatif des performances et qui n´engendre pas un surcoût du temps de calcul de la partition. Nous présentons et validons ici les méthodes mises en place dans cet outil, pour mesurer la vitesse des processeurs et les débits du réseau de communication.

Graphe de l´architecture

L´architecture de calcul peut être représentée par un graphe complet pondéré : les sommets sont les processeurs et les arêtes représentent la communication existante entre deux processeurs. On associe à chaque sommet la vitesse normalisée du processeur correspondant et à chaque arête le débit normalisé du réseau correspondant. L´utilisation de la bibliothèque de communication MPI ([8]) assure que tous les processeurs peuvent communiquer entre eux, ce qui fait du graphe un graphe complet que l´on pourra représenter par une matrice des communications.

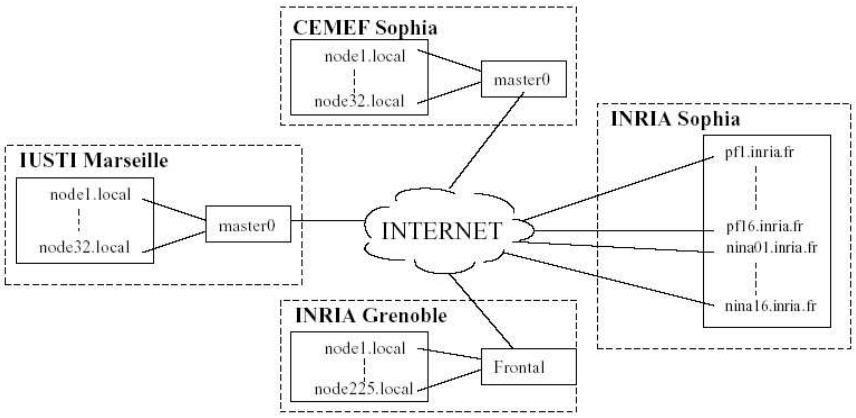


FIG. 2.1 – Schéma de la grille de calcul du projet MECAGRID

Exemples d´architecture

La figure 2.1 présente un schéma de la grille de calcul. De façon plus détaillée, la figure 2.2 donne une représentation interne du cluster de l´INRIA Sophia et l´équation 2.1 montre comment on représente le graphe des communications correspondant à la figure 2.2.
Nous renvoyons au site internet du projet MECAGRID pour toutes informations sur les caractéristiques internes (machines et réseaux) des autres centres de la grille.

2.1 Vitesse des processeurs

L'idée, ici, est de mesurer le temps nécessaire pour effectuer une série d'opérations représentatives d'un code éléments ou volumes finis. L´exécution d´un code éléments finis représenterait donc une excellente mesure. Ainsi, le code *Stokes*, présentée en annexe B, est exécutée sur un maillage carré de 20000 noeuds, et traite 60000 inconnues (2 vitesse, 1 pression). Cependant, le test des vitesses de processeurs avec ce type d´application n´est pas pratique (fichier de données, maillage, temps de calcul) et il est plus simple d´utiliser un test moins élaboré. Nous avons choisi d´effectuer un simple produit scalaire.
Nous présentons deux types de produit scalaire de 100000 réels codés en double précision : l'un effectué dans l'ordre des composantes (mesure 1), l'autre de façon aléatoire (mesure 2). Quatre machines de l´INRIA, pf, tahat, nina et taessa sont utilisées pour valider les tests. Pour chacune de ces machines, on présente, dans le tableau 2.1, quatre informations :

- la fréquence du processeur, informations théoriques des performances ;
- le temps nécessaire pour effectuer la mesure 1 ;
- le temps nécessaire pour effectuer la mesure 2
- le temps d´execution de l´application Stokes.

Ces quatres informations sont complétées par ce que l´on a nommé "fréquence normalisée" : on considère donc les inverses des temps, soit des fréquences, adimensionnées par la plus petite fréquence.

Machine	**pf10**	**tahat**	**nina10**	**taessa**
Fréquence (Mhz)	933	1000	2000	2400
Fréquence normalisée	1	1.07	2.14	2.57
Temps mesure 1 (s)	0.051	0.047	0.013	0.011
Fréquence normalisée	1	1.07	3.97	4.6
Temps mesure 2 (s)	0.176	0.168	0.072	0.066
Fréquence normalisée	1	1.04	2.43	2.61
Exécution Stokes (s)	52.35	52.28	21.61	20.14
Fréquence normalisée	1	1.01	2.42	2.60

TAB. 2.1 – Mesure des performances des processeurs

Ce tableau permet de tirer deux conclusions. La première est que des deux mesures, seule la mesure 2 donne des résultats cohérents avec les données théoriques et avec l´application éléments finis, comme le montrent les fréquences normalisées. La raison provient du fait qu'avec la mesure 1, l'utilisation du cache machine modifie les performances. En effet, lors d'un accés mémoire à une variable, le cache permet de stoker cette variable et un ensemble de variables “voisines”. Le produit scalaire aléatoire évite de prendre en compte des effets de cache trop important, et mesure donc bien les performances du processeur.
La seconde est que les temps nécessaires aux mesures (1 ou 2) sont minimes et nettement inférieurs à ceux de l´exécution de Stokes, ce qui était l´un des objectifs fixés.
La mesure 2 est donc retenue et la fréquence normalisée mesurée sera la vitesse de processeur considérée par la suite.

2.2 Débit du réseau

L'idée conclusion est cette fois de mesurer le temps nécessaire pour envoyer un message de données d'un processeur à un autre, afin de constuire une matrice des communications ($V = \{v_{ij}, \forall (i,j) \in (\mathcal{P}r \times \mathcal{P}r)\}$. Nous avons choisi d'envoyer un paquet de 1000 réels codés en double précision (test de type "ping").
Ainsi par exemple, la figure 2.2 présente les résultats obtenues pour un ensemble de quatre processeurs du cluster de l´INRIA : $\mathcal{P}r$ = {pf1,pf2,nina1,nina2}.

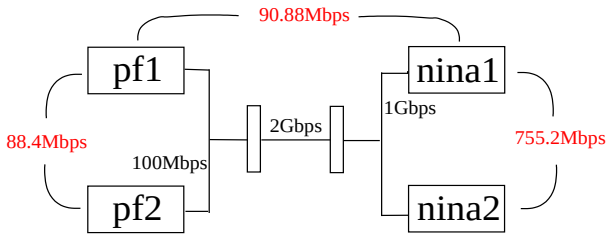


FIG. 2.2 – Mesures (en rouge) et données théoriques (en noir) des débits du réseau type du cluster INRIA

La matrice des communications V étant supposée symétrique, on présente ici les débits normalisés théoriques dans la partie supérieure de la matrice, et ceux mesurés dans la partie inférieure. Les coefficients diagonaux n´ont pas lieux d´être puisqu´il n´existe pas de réseau entre un processeur et lui même.

$$V = \begin{pmatrix} \times & 1 & 1 & 1 \\ 1 & \times & 1 & 1 \\ 1.02 & 1.02 & \times & 10 \\ 1.02 & 1.02 & 8.57 & \times \end{pmatrix} \tag{2.1}$$

Les mesures effectuées sont cohérentes avec les données théoriques.
Le graphe de la figure 2.3 représente l´architecture présentée ci-dessus, les processeurs pf sont en rouge, les nina en bleu. On renvoie au chapitre 4 figure 4.5, pour deux autres exemples de mesures, un sur une configuration avec deux machines de production INRIA, l´autre sur une configuration type cluster mais avec six processeurs.

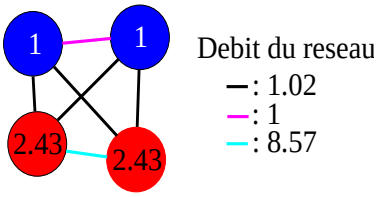


FIG. 2.3 – Graphe de l´architecture d´un sous-ensemble du cluster INRIA

2.3 Conclusion

L´outil developpé permet de disposer des vitesses de processeurs (fréquences normalisées de la mesure 1) et des débits du réseau (matrice des communications) de l´architecture de calcul visée. Ces informations seront les paramètres des modifications que nous apporterons au repartitionneur *Meshmigration* qui fera l´objet du chapitre suivant.

Chapitre 3

Meshmigration

L'objectif de cette partie est de présenter la méthode locale de repartitionnement parallèle utilisée par *Meshmigration* afin de montrer, au mieux, les modifications nécessaires pour une adaptation aux calculs sur grille. En amont de ceci, nous introduisons quelques notations et une notion essentielle, la fonction coût. Cette fonction sera le critère d'évaluation d'une partition.

3.1 Repartitionnement et renumérotation

On rappelle que le graphe associé au maillage est le graphe combiné $G_{\text{combiné}} = (\mathcal{N} \cup \mathcal{E}, \Sigma_{en})$ introduit au §1.2.2, et que l´ensemble $\mathcal{P}r$ est l´ensemble des p processeurs (def. 1.3).
Etant donnée une partition π (def. 1.3) du graphe $G_{\text{combiné}}$ sur l´ensemble $\mathcal{P}r$, on définit l´inverse de π par les deux applications suivantes :

$$\begin{array}{rcccl} \mathcal{N}_1 & : & \mathcal{P}r & \longrightarrow & \mathcal{P}(\mathcal{N}) \\ & & p & \longmapsto & \mathcal{N}_1(p) = \{n \in \mathcal{N} \ / \ \pi(n) = p\} \end{array} \tag{3.1}$$

$$\begin{array}{rcccl} \mathcal{E}_1 & : & \mathcal{P}r & \longrightarrow & \mathcal{P}(\mathcal{E}) \\ & & p & \longmapsto & \mathcal{E}_1(p) = \{e \in \mathcal{E} \ / \ \pi(e) = p\} \end{array} \tag{3.2}$$

$\mathcal{N}_1(p)$ (resp. $\mathcal{E}_1(p)$) est l´ensemble des noeuds (resp. éléments) appartenant au processeur p.
On associe à chaque noeud et élément de ces ensembles un numéro local. On définit ainsi deux bijections $\forall p \in \mathcal{P}r$:

$$\nu_p \ : \ \mathcal{N}_1(p) \longrightarrow \{0, ..., \text{card}(\mathcal{N}_1(\text{p})) - 1\} \tag{3.3}$$

$$\eta_p \ : \ \mathcal{E}_1(p) \longrightarrow \{0, ..., \text{card}(\mathcal{E}_1(\text{p})) - 1\} \tag{3.4}$$

Afin de conserver le maillage dans son ensemble, il est nécessaire d'ajouter à cette numérotation locale un numéro de processeur. On définit ainsi une double numérotation des noeuds et des éléments par les bijections suivantes :

$$\begin{array}{rcccl} g_n & : & \mathcal{N} & \longrightarrow & \mathcal{K}_{\mathcal{N}} = \left\{(i,j) \in \{0, ..., \text{card}(\mathcal{N}_1(\text{j})) - 1\} \times \mathcal{P}\text{r}\right\} \\ & & n & \longmapsto & N = (n_1, n_2) \ / n_2 = \pi(n) \text{ et } n_1 = \nu_{n_2}(n) \end{array} \tag{3.5}$$

$$\begin{array}{rcccl} g_e & : & \mathcal{E} & \longrightarrow & \mathcal{K}_{\mathcal{E}} = \left\{(i,j) \in \{0, ..., \text{card}(\mathcal{E}_1(\text{j})) - 1\} \times \mathcal{P}\text{r}\right\} \\ & & e & \longmapsto & E = (e_1, e_2) \ / e_2 = \pi(e) \text{ et } e_1 = \eta_{e_2}(e) \end{array} \tag{3.6}$$

Une partition d'un graphe combiné est donc entièrement définie par ces deux doubles numérotations. Ainsi, un repartitionnement consiste simplement à renuméroter $\mathcal{N}$ et $\mathcal{E}$.

Remarque 3.1 *Dans cette optique de renumérotation, le partitionnement est simplement un cas particulier du repartitionnement : l'ensemble du maillage est sur un seul processeur. On a donc* $\mathcal{N}_1(i) = \emptyset$ *et* $\mathcal{E}_1(i) = \emptyset$ *pour tous les* $i \in \mathcal{P}r$ *sauf pour* $i = p$*, avec* p *premier processeur hôte.*

3.2 Evaluation d'une partition : fonction coût

Afin de pouvoir améliorer, optimiser une partition, il est nécessaire d'établir un critère de comparaison entre deux partitions. L'idée retenue est de définir une fonction coût permettant de classer les partitions les unes par rapport aux autres.

L'objectif, une fois la partition effectuée, étant l'exécution d'un programme parallèle, une fonction coût naturelle est le temps d'exécution de ce programme. Classiquement, pour le processeur p, on décompose ce temps en trois parties :

t_p, temps de calcul
c_p, temps de communication
a_p, temps d'attente

Nous allons maintenant mettre en place un modèle de représentation de ces différents temps.

Temps de calcul :
On suppose que la charge de travail d'un processeur évolue de façon linéaire en fonction du nombre d'éléments et de noeuds qu'il héberge. Il ne s´agit évidemment pas de dire que le nombre d´itérations nécessaire à la résolution d´un système évolue linéairement avec le nombre d´inconnues. Par contre, le nombre d´opérations nécessaire à chaque itérations évolue lui, linéairement avec le nombre d´inconnues. On considère le cas de vitesses de processeur hétérogènes. On note s_p la vitesse de calcul du processeur p. Il vient :

$$t_p = \frac{d_p}{s_p} \tag{3.7}$$

avec : d_p charge de travail du processeur p
Le terme d_p ne dépend que de l'application et donc du problème à traiter. L'étude de plusieurs modèles pour représenter cette charge de travail a été menée par Hugues Digonnet ([3]) dans sa thèse, et ne sera pas abordée ici. Par contre, le terme s_p est fixé par l'architecture machine sur laquelle sera exécutée l'application. Sa mesure a été présentée au §2.1.

$$d_p = \sum_{E \in \mathcal{E}_1(p)} d(E) + \sum_{N \in \mathcal{N}_1(p)} d(N) \tag{3.8}$$

L'équation (3.8) est un modéle général qui différencie les coûts de calcul des éléments et des noeuds. Les poids asssociés aux éléments d(E) et aux noeuds d(N) peuvent représenter par exemple le nombre de variables aux noeuds ou aux mailles. La détermination de ces poids ne sera pas abordée ici mais fait partie des perspectives possibles.
Dans un premier temps, le modèle retenu est l'équation (3.9) qui recense le nombre d'éléments et de noeuds hébergés par le processeur p.

$$d_p = card(\mathcal{E}_1(p)) + card(\mathcal{N}_1(p)) \tag{3.9}$$

sous l'hypothèse : d(E)=d(N)=1 $\forall$ E $\in \mathcal{E}$, $\forall$ N $\in \mathcal{N}$.

Temps de communication :
On choisit de définir le temps de communication comme la somme de toutes les communications deux à deux. L'utilisation de communications bloquantes (un processeur ne continue sa tache que lorsque le message qu'il a envoyé a été recu) justifie ce choix. De plus, dans les applications éléments ou volumes finis, les communications globales (broadcast) sont réduites devant les communications locales (point à point) et on les néglige donc dans l´expression du temps de communication du processeur p.

$$c_p = \sum_{\substack{i \in \mathcal{P}r \\ i \neq p}} c_{pi} \tag{3.10}$$

avec c_{pi} temps passé dans les communications entre le processeur p et i.
On note v_{pi} la vitesse (de transfert ou le débit) de la communication entre les processeurs p et i et d_{pi} le volume de cette communication. Il vient :

$$c_{pi} = \frac{d_{pi}}{v_{pi}} \tag{3.11}$$

De la même façon que pour le temps de calcul, le volume de communication ne dépend que du problème à traiter alors que la vitesse dépend de l'architecture machine.

Compte tenu du choix fait au §1.2.2 de représenter le maillage par un graphe combiné associé, il existe une communication entre deux processeurs si un élément et au moins l'un des noeuds qui le constituent ne sont pas sur le même processeur. Le volume des communications d_{pi} sera donc évalué par :

$$d_{pi} = card\Big(\mathcal{N}_1(i) \cap T(\mathcal{E}_1(p))\Big) + card\Big(\mathcal{E}_1(i) \cap T^{-1}(\mathcal{N}_1(p))\Big) \tag{3.12}$$

En effet, $\mathcal{N}_1(i)$ (def. 3.1) est l'ensemble des noeuds du processeur i, $\mathcal{E}_1(p)$ (def.3.2) l'ensemble des éléments de p, et donc $T(\mathcal{E}_1(p))$ (def 1.1) l'ensemble des noeuds formants ces éléments. Donc si $\mathcal{N}_1(i) \cap T(\mathcal{E}_1(p)) \neq \emptyset$ il y a au moins une communication. Le raisonnement pour le second terme est le même mais pour les éléments de i.
Le calcul de $card(\mathcal{N}_1(i) \cap T(\mathcal{E}_1(p)))$ et $card(\mathcal{E}_1(i) \cap T^{-1}(\mathcal{N}_1(p)))$ est réalisé en faisant une boucle sur l´ensemble des éléments et des noeuds appartenant au processeur p :

$$d_{pi} = \sum_{E \in \mathcal{E}_1(p)} c(E, p, i, \mathcal{E}, \mathcal{N}) + \sum_{N \in \mathcal{N}_1(p)} c(N, p, i, \mathcal{E}, \mathcal{N}) \tag{3.13}$$

avec un coût de communication local associé aux éléments et aux noeuds noté :

$$\begin{aligned} c(E, i, j, \mathcal{E}, \mathcal{N}) &= card\Big(\mathcal{N}_1(j) \cap T(E)\Big) \qquad E \in \mathcal{E}_1(i) \\ c(N, i, j, \mathcal{E}, \mathcal{N}) &= card\Big(\mathcal{E}_1(j) \cap T^{-1}(N)\Big) \quad N \in \mathcal{N}_1(i) \end{aligned} \tag{3.14}$$

Fonction coût :
Dans une application parallèle, le temps d´exécution est égal au temps que mettra le processeur le plus lent à réaliser sa tâche. Le temps d´attente n´a pas lieu d´être puisque le processeur le plus lent est le seul à ne pas attendre. Ceci, ainsi que les définitions des t_p et c_p, permet de définir une borne supérieure du temps d´exécution parallèle :

$$\max_{p\in\mathcal{P}r}(t_p + c_p + a_p) = \max_{p\in\mathcal{P}r}(t_p + c_p) \leq \max_{p\in\mathcal{P}r}(t_p) + \max_{p\in\mathcal{P}r}(c_p) \tag{3.15}$$

La fonction coût retenue est cette borne supérieure :

$$\phi = \max_{p\in\mathcal{P}r}(t_p) + \max_{p\in\mathcal{P}r}(c_p) \tag{3.16}$$

Il convient de définir deux coefficients α et β qui permettent de pondérer le temps de calcul et le temps de communication :

$$\phi = \alpha \max_{p\in\mathcal{P}r}(t_p) + \beta \max_{p\in\mathcal{P}r}(c_p)$$

Ces coefficients permettent d'adapter la fonction coût à la partition souhaitée.
Dans la suite du document, on fixe ces coefficient à l'unité.

Le paragraphe suivant (3.3), présente la méthode locale de repartitionnement. Très succintement, étant donnée une partition du graphe associé au maillage, la stratégie parallèle consiste à former des paires de processeurs et à optimiser localement les bipartitions locales.
La fonction coût ϕ n'est pas adapter au cas d'un repartitionneur parallèle. En effet, plusieurs partitions peuvent avoir la même valeur de ϕ (exemple figure 3.1). Ainsi une amélioration locale de la partition n'entraine pas forcément une amélioration globale.
Le problème de la fonction ϕ est l'utilisation du *max* qui enlève tout caractère local. La nouvelle relation d'ordre s'applique, non plus dans $\mathbb{R}$, mais sur des vecteurs de données.

◇ **Définition 3.1 (RELATION D'ORDRE θ)** *Soient deux vecteurs V^a,V^b de dimension p. On note $\ddot{V}^a$ (resp. $\ddot{V}^b$) le vecteur obtenu en classant les composantes de V^a (resp.V^b) de la plus grande à la plus petite. On note $\mathcal{D}(\ddot{V}^a, \ddot{V}^b)$ l'ensemble des i tels que $\ddot{v}^a_i \neq \ddot{v}^b_i$. On définit alors une relation d´ordre entre les deux vecteurs V^a,V^b par :*

$$V^a =_\theta V^b \iff \mathcal{D}(\ddot{V}^a, \ddot{V}^b) = \emptyset \tag{3.17}$$

$$V^a >_\theta V^b \; si \max_{i\in\mathcal{D}(\ddot{V}^a,\ddot{V}^b)} \ddot{v}^a_i > \max_{i\in\mathcal{D}(\ddot{V}^a,\ddot{V}^b)} \ddot{v}^b_i \tag{3.18}$$

Cette relation a deux propriétés essentielles :
- une minimisation locale entraine une minimisation globale ;
- la première composante de la relation correspond à la fonction coût ϕ.

Avec un algorithme parallèle tel que celui considéré, la fonction coût la plus adaptée est la suivante :

$$\ddot{\psi} = \ddot{T} + \ddot{C} \tag{3.19}$$

avec : $T = {}^t\left(t_1, \ldots, t_{card(\mathcal{P}r)}\right)$
$C = {}^t\left(c_1, \ldots, c_{card(\mathcal{P}r)}\right)$

L´objectif du partitionnement de maillage est de trouver la partition π du graphe associé qui minimise la fonction coût ψ au sens de la relation d´ordre θ.

Afin d'illustrer certaines propriétés de cette relation d'ordre, on introduit une repésentation simplifiée de la partition. Soit, donc, le graphe $G_p(V_p, E_p)$ définit par :

- V_p : ensemble des sommets du graphe
  Chaque sommet correspond à un processeur.
  Le poids associé à un sommet correspond au temps de travail du processeur : t_p.
- E_p : ensemble des arrêtes du graphe
  Il existe une arrête entre deux sommets si et seulement si
  il existe des communications entre les deux processeurs correspondants.
  Le poids associé à une arrête correspond au temps de communications : c_{ij}.

La figure 3.1 montre trois graphes correspondants à trois partitions différentes. On a mis en évidence, en bleu, une paire de processeurs : la partition (a) n´est pas équilibrée au niveau de la charge (26 et 10) alors que la partition (b) l´est (18,18). Le tableau 3.1 illustre l'un des intérêts majeur de la relation d'ordre θ à savoir qu'elle permet de différencier les partitions grâce à leur propriétés locales. En effet, si l'on reste au niveau global, la fonction ϕ ne fait aucune différence entre les trois partitions ($\phi = 57$). La relation d´ordre θ appliquée au vecteur ψ permet par contre, de les classer ; on obtient ainsi de la meilleure à la moins bonne : (b),(a) et (c).

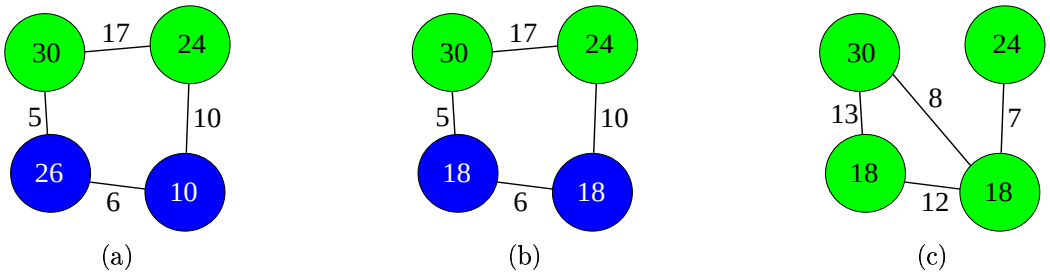


FIG. 3.1 – Exemple de trois partitions sur trois processeurs

	(a)	**(b)**	**(c)**	**Classement**
$\boldsymbol{\phi}$	$57 = 30 + (10 + 17)$	$57 = 30 + (10 + 17)$	$57 = 30 + (12 + 8 + 7)$	$?/?/?$
$\ddot{\mathbf{T}}$	${}^t(30, 26, 24, 10)$	${}^t(30, 24, 18, 18)$	${}^t(30, 24, 18, 18)$	$?/?/a$
$\ddot{\mathbf{C}}$	${}^t(27, 22, 16, 11)$	${}^t(27, 22, 16, 11)$	${}^t(27, 25, 19, 7)$	$?/?/c$
$\ddot{\boldsymbol{\psi}}$	${}^t(57, 48, 40, 21)$	${}^t(57, 46, 34, 29)$	${}^t(57, 49, 37, 25)$	$b/a/c$

TAB. 3.1 – Evaluations de trois partitions (Classement de la meilleure à la moins bonne des partitions selon les critères d'évaluation)

3.3 Méthode locale de repartitionnement

Cette partie a pour but de détailler la méthode de repartitionnement parallèle sur laquelle repose *Meshmigration*. Nous détaillons ensuite les modifications apportées dans le cadre du calcul sur grille.

3.3.1 Algorithme de repartitionnement

L'approche se base sur un couplage des processeurs permettant d'échanger simultanément des entités à l'intérieur des paires formées. L'utilisation du parallélisme se fait donc dès la phase de partitionnement, et s'adapte parfaitement aux repartitionnements durant les phases de calcul parallèle (remaillage ou modification de l'architecture de calcul). La relation d'ordre 3.19 assure que les optimisations locales permettent d'obtenir une partition minimisant la fonction coût choisie.
L'algorithme parallèle est donc le suivant :

Algorithme 3.1 Méthode locale d'optimisation (tiré de [3])

Initialisation :
- Un ensemble de processeurs p avec leur domaine $\left(\mathcal{N}_1(p), \mathcal{E}_1(p)\right)$ (eq. 3.1 et 3.2)
- Une variable logique $optimum = faux$

Tant que non optimum **répète :**

Pour tous les processeurs p **faire :**

Construction de son Voisinage :
- $\mathcal{V}_p$: ensemble des processeurs ayant des communications avec p

Couplage des processeurs :
- recherche d'un processeur « âme soeur » $\overline{p} \in \mathcal{V}_p$
à l'aide d'une fonction *amitié*
- soit formation de la paire $(p, \overline{p})$
- soit p reste isolé.

Une variable locale $NbChangements$

Si il existe $\overline{p}$ **Alors :**

Optimisation locale de la paire formée :
- échanges d'éléments et de nœuds dans la paire $(p, \overline{p})$
suivant une priorité de transfert établie à l'aide de *gains*
- $NbChangements$ = nombre de changements effectués par l'optimisation

Sinon :
- processeur isolé donc $NbChangement = 0$

fin Si

fin Pour

Une communication globale :
- $optimum = vrai$ si aucun changement *réactualisation de optimum.*

fin Tant que

fin algorithme

Pour être en accord avec la fonction coût ψ et prendre en compte l´architecture machine dans la méthode, il est nécessaire de redéfinir les fonctions *amitié* et *gains*. Nous présentons les étapes de *couplages* et d´*optimisation locale* pour ensuite expliciter ces deux fonctions en les décomposant en une partie sur la charge de travail et une sur les communications.

Couplage parallèle des processeurs

L'algorithme de couplage permet de former, en parallèle, le maximum de paires de processeurs ayant le plus fort potentiel à optimiser la partition. Les paires qu´il est possible de former sont les paires de processeurs voisins, ie de processeurs entre lesquels il existe des communications. Précisons qu´il existe des communications entre deux processeurs s´il existe une interface entre les sous-graphes qu´ils hébergent. On note $\mathcal{R}(\mathcal{P}r, \mathcal{P}r)$ l´ensemble des paires de processeurs possibles.
Cette étape de couplage nécessite un critère de comparaison entre les différentes paires possibles. Ce critère doit rendre compte des améliorations possibles dans une paire, tant au niveau de la charge de travail que des communications.
Nous définissons la fonction symétrique *amitié* par la relation suivante :

$$\begin{array}{rccl} amitié & : & \mathcal{R}(\mathcal{P}r, \mathcal{P}r) & \longrightarrow & \mathbb{R} \\ & & (i,j) & \longmapsto & amitié(i,j) = amitié_T(i,j) + amitié_C(i,j) \end{array} \tag{3.20}$$

avec $amitié_T$ et $amitié_C$ deux fonctions symétriques de $\mathcal{R}(\mathcal{P}r, \mathcal{P}r)$ dans $\mathbb{R}$.
Ces deux fonctions, que nous définirons au §3.2.2 et §3.2.3, sont une évaluation des améliorations possibles dans une paire de processeur, au niveau de la charge de travail ($amitié_T$) et au niveau des communications ($amitié_C$).
On note $\mathcal{R}'(\mathcal{P}r, \mathcal{P}r)$ l'ensemble des paires possibles n´ayant pas encore été formées. On a $\mathcal{R}'(\mathcal{P}r, \mathcal{P}r) = \mathcal{R}(\mathcal{P}r, \mathcal{P}r)$ lorsqu´aucune n´a été formée. Ainsi, la paire (p, q) se forme si et seulement si :

$$amitié(p,q) = \max_{(i,j) \in \mathcal{R}'(\mathcal{P}r, \mathcal{P}r)} amitié(i,j) \tag{3.21}$$

Optimisation locale d'une paire de processeurs

On considère une paire de processeurs (p,q). Nous rappelons la définition (3.19) de la fonction coût : $\ddot{\psi} = \ddot{T} + \ddot{C}$.
Pour la charge de travail, une optimisation de la bipartition liée à la paire modifiera uniquement t_p et t_q. Ceci permet d'écrire la restriction de T à la paire (p,q) :

$$T|_{pq} = {}^t(t_p, t_q)$$

Pour les communications, une optimisation de la bipartition liée à la paire se ferait indépendamment des autres processeurs si et seulement si, les communications de la paire avec les autres processeurs restaient identiques :

$$\sum_{\substack{r \in \mathcal{P}r \\ r \neq p,q}} (c_{rp} + c_{rq}) = C^{ste} > 0 \tag{3.22}$$

Cette contrainte est en fait une approximation dans la mesure où le voisinage de la paire peut être modifié. Cependant, les modifications effectuées, pour optimiser les communications, étant très locales, l´approximation de la contrainte 3.22 est acceptable.
Sous la contrainte précédente, nous pouvons réduire le vecteur communication à :

$$C|_{pq} = {}^t(c_p, c_q)$$

Ainsi la restriction de la fonction coût à la paire de processeurs (p,q), s'écrit :

$$\ddot{\psi}|_{pq} = \ddot{T}|_{pq} + \ddot{C}|_{pq} \tag{3.23}$$

L'objectif est alors de touver la meilleure bi-partition de la paire (p,q) par rapport à $\psi|_{pq}$. Ce problème étant NP-complet, une heuristique a été developpée afin d'obtenir un minimum de $\psi|_{pq}$.

Cette heuristique consiste à faire migrer, un par un, les meilleurs éléments et noeuds vis à vis de $\psi|_{pq}$. La sélection s'effectue à l'aide d'une priorité de transfert : le *gain* à la migration. Ce gain est une fonction qui à chaque élément et noeud de la paire associe un réel. On définit ainsi les notations :

$$\forall E \in \mathcal{E}_1(p) \cup \mathcal{E}_1(q) \ : \ g(E) = g_T(E) + g_C(E) \tag{3.24}$$

$$\forall N \in \mathcal{N}_1(p) \cup \mathcal{N}_1(q) \ : \ g(N) = g_T(N) + g_C(N) \tag{3.25}$$

avec g_T (respectivement g_C) gain, sur la charge de travail $T|_{pq}$ (respectivement sur les communications $C|_{pq}$) à faire migrer l'entité (E ou N) d´un processeur à l´autre de la paire.
De la même façon que pour les composantes de la fonction *amitié*, nous définirons les fonctions g_T et g_C au §3.2.2 et §3.2.3

La partie suivante présente les définitions, prenant en compte les caractéristiques de la grille, des composantes charge et communication de la fonction *amitié* et des *gains* à la migration. On considère une paire de processeurs (p,q).

3.3.2 Optimisation sur la charge

On rappelle la définition de la charge de travail d'un processeur p introduite précédemment (3.7), et où l'on suppose que la charge de travail d'un processeur évolue de façon linéaire en fonction du nombre de noeuds et d'éléments qu'il héberge :

$$t_p = \frac{d_p}{s_p} \text{ avec } d_p = card(\mathcal{E}_1(p)) + card(\mathcal{N}_1(p))$$

La restriction $T|_{pq}$ s'écrit alors :

$$T|_{pq} = \ ^t(t_p, t_q) = \ ^t\Big(\frac{d_p}{s_p}, \frac{d_q}{s_q}\Big)$$

Sous la contrainte $d_p + d_q = C^{ste}$, on cherche le plus petit vecteur, $T|^{eq}_{pq}$, parmi les vecteurs $T|_{pq}$ possibles. Les composantes de ce vecteur sont données par :

$$t^{eq}_p = t^{eq}_q \Longleftrightarrow \frac{d^{eq}_p}{s_p} = \frac{d^{eq}_q}{s_q} \tag{3.26}$$

avec

$$d^{eq}_p + d^{eq}_q = d_p + d_q$$

On pose :

$$d^{eq}_p = d_p + a \text{ et } d^{eq}_q = d_q - a$$

D´après l´équation 3.26, il vient :

$$\frac{d_p + a}{s_p} = \frac{d_q - a}{s_q} \Longleftrightarrow a = \frac{d_q s_p - d_p s_q}{s_p + s_q}$$

On obtient donc :

$$t_p = \frac{d_p}{s_p} + \frac{a}{s_p} \text{ et } t_q = \frac{d_q}{s_q} - \frac{a}{s_q}$$

L'amitié du processeur p pour le processeur q est donnée par le gain maximal possible :

$$amitié_T(p,q) = \begin{cases} \dfrac{a}{s_p} & \text{si } a > 0 \\ -\dfrac{a}{s_q} & \text{si } a < 0 \end{cases} \tag{3.27}$$

Ayant considéré en première approximation, que les éléments et les noeuds représentaient la même charge de travail (d(E)=d(N)=1), le gain à migrer une de ces entités du processeur p vers le processeur q est le suivant :

$$\forall X \in \mathcal{E}_1(p) \cup \mathcal{N}_1(p), \ g_T(X) = \max\Big(\frac{d_p}{s_p}, \frac{d_q}{s_q}\Big) - \max\Big(\frac{d_p - 1}{s_p}, \frac{d_q + 1}{s_q}\Big) \tag{3.28}$$

On note $T|'_{pq}$ la restriction de T à la paire (p,q) une fois l'entité déplacée. On a en fait comparé les composantes maximales de $T|_{pq}$ et de $T|'_{pq}$, ce qui permet de les ordonner suivant la relation d'ordre θ.

3.3.3 Optimisation sur les communications

Contrairement à la charge de travail pour laquelle on connait le minimum de $T|_{pq}$, on ne peut estimer à priori le minimum de $C|_{pq}$. Il est cependant nécessaire d'avoir une idée des améliorations possibles dans cette paire.
L'idée retenue consiste à évaluer la qualité des communications entre les processeure p et q. On calcule, pour chaque élément et noeud de la paire ($\mathcal{E}_1(p) \cup \mathcal{N}_1(p)$ pour le processeur p et $\mathcal{E}_1(q) \cup \mathcal{N}_1(q)$ pour le processeur q), le gain, au niveau des communications, à le changer de processeur :

$$amitié_C(p,q) = \sum_{X \in \left(\mathcal{E}_1(p) \cup \mathcal{N}_1(p)\right)} g_C(X,p,q) + \sum_{X \in \left(\mathcal{E}_1(q) \cup \mathcal{N}_1(q)\right)} g_C(X,q,p) \tag{3.29}$$

La fonction $amiti_C$ dépend du modèle de g_C choisi. Nous explicitons donc le modèle retenu.
On considère une entité $X \in \mathcal{E}_1(p) \cup \mathcal{N}_1(p)$, i.e un élément ou un noeud du processeur p. Le calcul du *gain* à la migration de cette entité du processeur p vers le processeur q consiste à évaluer les communications liées à l´entité avant migration ($\pi(X) = p$) et après migration ($\pi(X) = q$).
D´après le choix du graphe combiné comme graphe associé au maillage, on peut définir le voisinage $\mathcal{V}(X)$ de l´entité :

$$\begin{array}{l} \forall X \in \mathcal{E}_1(p),\ \mathcal{V}(X) = T(X) \\ \forall X \in \mathcal{N}_1(p),\ \mathcal{V}(X) = T^{-1}(X) \end{array} \tag{3.30}$$

Le gain à la migration de p vers q est défini par :

$$\forall X \in \mathcal{E}_1(p) \cup \mathcal{N}_1(p),\ g_C(X,p,q) = \gamma \sum_{Y \in \mathcal{V}(X)} g(X,Y,p,q) \tag{3.31}$$

avec le coefficient γ défini par :

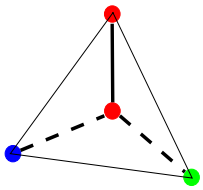
$$\gamma = \left\{ \begin{array}{ll} 1 & \text{si } \exists Y \in \mathcal{V}(X)/\pi(Y) = q \\ 0 & \text{sinon} \end{array} \right. \tag{3.32}$$

et les gains élémentaires par :

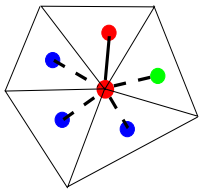
$$g(X,Y,p,q) = \left\{ \begin{array}{ll} -\dfrac{1}{v_{pq}} & \text{si } \pi(Y) = p \\ +\dfrac{1}{v_{pq}} & \text{si } \pi(Y) = q \\ +\dfrac{1}{v_{pr}} - \dfrac{1}{v_{qr}} & \text{si } \pi(Y) = r, r \neq p, q \end{array} \right. \tag{3.33}$$

Le coefficient γ est d'importance : notre objectif étant de comparer les communications avant et après migration d'une entité de p vers q, le test de "q-voisinage" est obligatoire pour ne pas prendre en compte les communications de p avec d´autres processeurs que q.
Une modification essentielle est la prise en compte de l'existance possible d'un, ou plusieurs, processeur $r \neq p, q$ sur le voisinage d'une entité pouvant migrer de p vers q (q-voisinage). Dans le cadre d'un réseau hétérogène, une optimisation possible de $C|_{pq}$ est un transfert des communications de p avec r à q avec r si $v_{pr} < v_{qr}$.

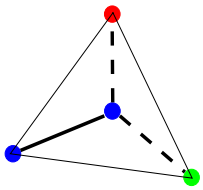
Nous présentons deux exemples simples (2D) où trois processeurs interviennent dans le voisinage de l'entité à évaluer. L'appartenance au processeur p (resp. q et r) est symbolisée par une coloration rouge (resp. bleue et verte). A gauche, le gain indique si la migration de l'élément du processeur p vers le processeur q améliorerait la partition. L'idée est la même à droite avec le noeud du processeur p.



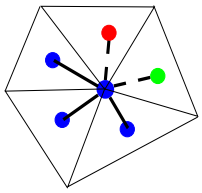
Avant migration



Avant migration



Apres migration



Apres migration

$$gain = \{\frac{1}{v_{pq}} + \frac{1}{v_{pr}}\} - \{\frac{1}{v_{pq}} + \frac{1}{v_{qr}}\} \qquad gain = \{3 * \frac{1}{v_{pq}} + \frac{1}{v_{pr}}\} - \{\frac{1}{v_{pr}} - \frac{1}{v_{qr}}\}$$

FIG. 3.2 – Illustration de l'évaluation de gains "communication" d'un élément et d'un noeud à migrer

3.4 Conclusion

Nous avons montré dans ce chapitre que la méthode locale de repartitionnement utilisée dans *Meshmigration* s´adaptait parfaitement au calcul sur grille. Les étapes principales de la méthode étant le couplages en parallèle des processeurs et l´optimisation locale des paires formées, seuls le critère de formation de ces paires (fonction *amitié*) et la priorité de transfert des éléments et noeuds ont été à redéfinir.
Le chapitre suivant sera consacré à la validation des modifications effectuée. On s´attachera à valider séparement les modificatioins concernant la charge de travail et les communications.

Chapitre 4

Validations

Ce chapitre présente un ensemble de tests réalisés afin de valider les modifications apportées au repartitionneur parallèle *Meshmigration*.
Dans un premier temps, nous vérifions séparément les modifications portant sur la charge de travail et celles portant sur les communications.
Ensuite, l'éxécution d'une application type éléments finis (Solveur de Stokes présenté en annexe B) nous permet de connaitre le gain réalisé en terme de temps de calcul et de contrôler le modèle de fonction coût mis en place.

4.1 Partitionnement de maillage

Cette partie présente les tests nous ayant permis de valider, séparément, les modifications liées à la charge de travail et aux communications.
On utilise deux maillages : un carré (441 noeuds, 813 éléments) et un cube (1443 noeuds, 6758 éléments). Ces maillages sont volontairement simples pour permettre une visualisation efficace.
Pour chaque test, on présente la "partition homogène" obtenue avec *Meshmigration* homogène, i.e avant modification, la "partition optimisée" obtenue avec *Meshmigration* optimisé pour les grilles, ainsi que la partie de la fonction coût correspondante (T ou C).
On renvoie aux définitions 3.7, 3.10 et 3.11 où l´on a introduit les expressions des composantes de T et C.

4.1.1 Résultats sur la charge

L´architecture machine est la suivante : les processeurs numérotés 1, 2, 3 et 4 ont des vitesses de 1, 2, 3 et 4 et les sous-maillages qui leur sont associés sont bleu, vert, rouge et blanc(gris) ; le réseau est homogène.

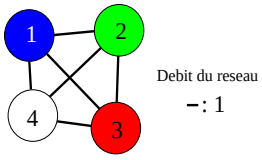
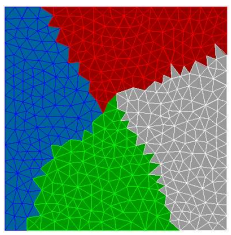
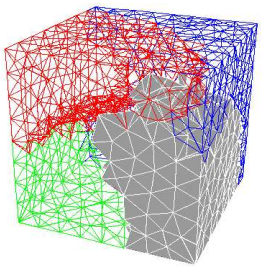


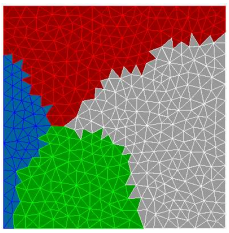
FIG. 4.1 – Graphe de l´architecture machine



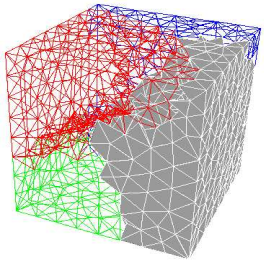
(a_1) Partition homogène



(a_2) Partition homogène



(b_1) Partition optimisée



(b_2) Partition optimisée

Processeur	$T_{\text{homogène}}$	$T_{\text{optimisé}}$
1	315	125
2	156.5	125
3	104.33	125.33
4	78.25	125.75

Processeur	$T_{\text{homogène}}$	$T_{\text{optimisé}}$
1	2013	820
2	1026.5	820
3	686.76	820
4	518.75	820.25

FIG. 4.2 – Validation sur une achitecture hétérogène au niveau des processeurs

On observe que sur la partition homogène, les sous-maillages sont de taille équivalente ce qui fait que le processeur 2 étant deux fois plus rapide que le processeur 1, sa charge de travail, $T_{homogene}$, est deux fois moindre (156,5 pour 315 avec le carré ; 1026.5 pour 2013 avec le cube). Le raisonnement est le même pour les autres processeurs. Par contre, sur la partition optimisée, les charges de travail sont équilibrées puisque la taille des sous-maillages dépend des vitesses des processeurs. Autre résultat essentiel, la partie charge de travail de la fonction coût, à savoir le $\max_{p \in \mathcal{P}r}(t_p)$ passe de 315 à 125 pour le carré, et de 2013 à 820 pour le cube, ce qui montre bien que l´on a amélioré la partition.
L´amélioration est donc notable puisque dans le cadre d´application parallèle, il est nécessaire d´équilibrer la charge de façon à supprimer les temps d´attente.

4.1.2 Résultats sur les communications

L´architecture machine est la suivante. Les processeurs ont la même vitesse, 1. Le réseau de communications est hétérogène : les communications entre un processeur et le processeur bleu sont 1000 fois plus rapides que les autres.

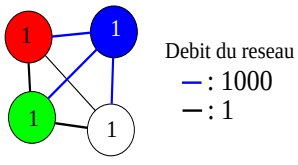
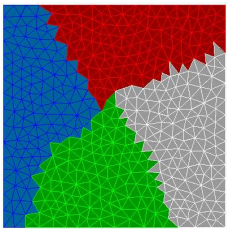
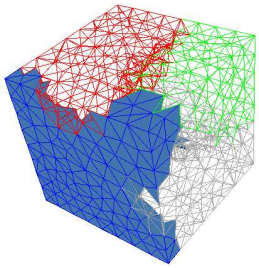


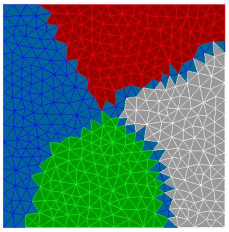
FIG. 4.3 – Graphe de l´architecture machine



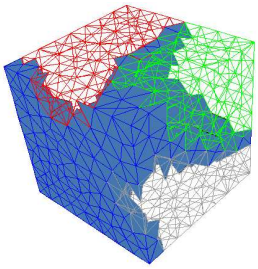
(a_1) Partition homogène



(a_2) Partition homogène



(b_1) Partition optimisée



(b_2) Partition optimisée

Processeur	$C_{\text{homogène}}$	$C_{\text{optimisé}}$
1	0.049	0.153
2	33.027	0.051
3	29.022	0.052
4	54.000	0.050

Processeur	$C_{\text{homogène}}$	$C_{\text{optimisé}}$
1	1.182	3.123
2	683.494	1.040
3	583.512	1.063
4	934.176	1.020

FIG. 4.4 – Validation sur une achitecture hétérogène au niveau du réseau

Sur la partition homogène, les communications des processeurs rouge, vert et blanc sont pénalisantes car le réseau restreint à ces trois processeurs est lent.
Sur la partition optimisée, on observe que ce réseau restreint à été supprimé en créant des communications rapides sur toutes les interfaces. Comme le montre le tableau, le temps de communications du processeur bleu (0.153 pour le carré et 3.191 pour le cube) est égale à la somme des communications des trois autres processeurs, ce qui montre que ces trois processeurs ne communiquent plus qu´avec le processeur bleu.
Les améliorations se traduisent par une baisse de la partie communication de la fonction coût, à savoir le $\max_{p\in\mathcal{P}r}(c_p)$, qui passe de 54 à 0.153 pour le carré, et de 934.176 à 3.123 pour le cube.

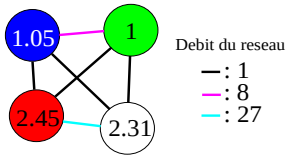
4.2 Performances

Nous présentons figure 4.5, pour les tests 1 et 2, le graphe de l´architecture machine sur laquelle on a exécuté *Meshmigration* et l´application *Stokes*, la partition optimisée du maillage ainsi que la fonction coût de la partition ϕ (définition 3.16) et le temps de résolution de l´application (T_{Stokes}) en seconde.
Les configurations de deux tests, à savoir, l´architecture machine, le maillage du domaine de calcul ainsi que la simulation considérée, sont les suivantes :

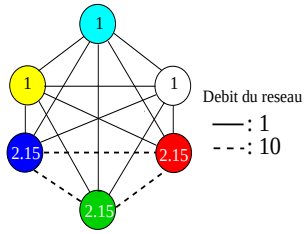
- Configuration test 1
  1. Architecture : 2 machines de production bi-processeur, une à 1Ghz (en bleu et vert), l´autre à 2,4Ghz (en rouge et blanc)
  2. Maillage : un carré composé de 20029 noeuds et 40056 éléments (triangles)
  3. Stokes : résolution en vitesse/pression (3 inconnues par noeuds), soit 60087 inconnues, de l´écoulement de Poiseuille d´un fluide newtonien, créé en imposant une pression sur l´une des face

– Configuration test 2

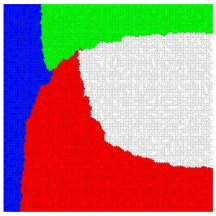
1. Architecture : 6 machines du cluster, 3 à 933Mhz (en jaune, cyan et blanc) et 3 à 2Ghz (en bleu, rouge et vert)
2. Maillage : cylindre extrudé sur une face, composé de 64914 noeuds et 377042 éléments
3. Stokes : résolution en vitesse/pression (4 inconnues par noeuds), soit 259656 inconnues, d´un problème consistant à imposer une pression sur la face d´entrée (non extrudée), et à maintenir fixe l´autre face sauf sur la partie extrudée ; le cylindre étant composé d´un fluide newtonien.



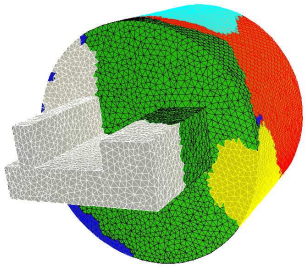
Architecture machine 1



Architecture machine 2



Partition optimisée du maillage 1



Partition optimisée du maillage 2

Partition	ϕ	T_{Stokes} (s)
Homogène	15355	108.2
Optimisée	9011	62.2
Gain (%)	41.3	42.5

Test 1

Partition	ϕ	T_{Stokes} (s)
Homogène	88108	97.9
Optimisée	57261	65.5
Gain (%)	35	33

Test 2

FIG. 4.5 – Validation avec une application éléments finis : solveur de Stokes

Les tableaux permettent de faire plusieurs observations. Tout d´abord, le critère d´évaluation d´une partition, la fonction coût, qui a été construit comme une approximation du temps d´exécution parallèle d´une application type éléments finis, semble tout à fait satisfaisant. En effet, pour le test 1 (resp. 2), on estime avoir réalisé un gain de 41.3% (resp. 35%) et l´exécution de l´application *Stokes* sur les partitions homogène et optimisée montre un gain de temps de 42.5% (resp. 33%).
Ensuite, et c´est le résultat essentiel, ces deux tests montrent que les modifications effectuées sur *Meshmigration* sont efficaces sur différents maillages (2D et 3D), et différentes architectures machines. De plus, un gain de temps de l´ordre de 40% sur l´exécution d´un code éléments finis est une amélioration appréciable.

4.3 Conclusion

L´ensemble des tests effectués nous a permis de valider les modifications apportées au repartitionneur parallèle *Meshmigration* et de montrer que la prise en compte de l´architecture machine était essentielle lorsque celle-ci était hétérogène. L´amélioration constatée des performances d´un code éléments finis justifie la démarche entreprise.

Conclusion générale

Le but de ce travail était d´adapter le repartitionnement parallèle de maillages non structurés, *Meshmigration*, au calcul sur grille. Cette étude est innovante dans la mesure où le problème du partitionnement de maillages a été, le plus souvent, abordé dans un contexte homogène.
Cette adaptation nécessitait une connaissance des caractéristiques de l´architecture machine : vitesse de processeurs et débits du réseau de communication. Nous avons, à cette fin, développé et validé un outil de mesure, représentatif des performances et totalement autonome.
Meshmigration s'appuie sur une méthode locale de repartitionnement parallèle : l'idée consiste à partir d'une partition initiale, à former des paires de processeurs et à échanger, dans ces paires, les éléments et noeuds améliorant la partition. Les modifications prenant en compte les paramètres mesurés ont principalement porté sur le critère d'évaluation d'une partition, sur le critère de formation de ces paires, ainsi que sur la définition des gains donnant une priorité de transfert aux éléments et aux noeuds.
Nous avons montré, par une phase de tests représentatifs du calcul sur grille (architecture machine et code de calcul), que le repartitionneur était maintenant capable de distribuer les sous maillages sur des processeurs de vitesses très différentes, en équilibrant la charge, et de tenir compte de débits de communications très disparates. L´amélioration constatée des performances d´un code éléments finis justifie la démarche entreprise.

Perspectives

Rappelons que le projet MECAGRID, qui est un projet de l'ACI-GRID 2003 du ministère de la recherche, et qui réunit l'INRIA Sophia-Antipolis, le CEMEF et l'IUSTI, a pour but la réalisation d'une grille de calcul régionale à partir de grappes de PC localisées en region PACA. Le déploiement de cette grille est en cours, et dans un futur proche (été 2003), *Meshmigration*, sera un outil essentiel, pour adapter les partitions des maillages, et *tirer ainsi le meilleur profit des ressources disponibles.*
Des *améliorations des performances du repartitioneur*, dans le cas du partitonnement (ie le cas on ne dispose pas d´une première partition) sont possibles par des optimisations "plus globales".

Bibliographie

[1] C. Cross C. Walshaw. *Multilevel mesh partitionning for heterogeneous communication networks.* FGCS 17(2001)601-623.

[2] Thierry Coupez. *Stable stabilised mixed finite element in forming process simulation.* rapport interne CEMEF, en préparation, 2002.

[3] Hugues Digonnet. *Repartionnement dynamique, mailleur parallèle et leurs applications à la simulation numérique en mise en forme des materiaux.* Thèse de l´Ecole Nationnale Supérieure des Mines de Paris, 2001.

[4] I. Foster and C. Kesselman. *Blueprint for a New Computing Infrastructure.* (Eds.) Morgan-Kaufmann, 1998.

[5] Dongarra (Jack J.) Jiang (Weicheng) Manchek (Robert) Geist (Al), Beguelin (Adam). *PVM3 User's guide anf Reference Manual.* Rapport Technique n ORNL/TM-12187, 1994.

[6] Vipin Kumar George Karypis. *Parallel Multilevel k-way Partitionning Scheme for Irregular Graphs.* Technical report 96-036, Department of Computer Science, University of Minnesota, 1998.

[7] Paul-Louis George Pascal Jean Frey. *Maillages, applications aux éléments finis.* Hermès Science Publications, 1999.

[8] Huss-Lederman (Steven) Dongarra (Javk J) Snir (Marc), Otto (Steve W.). *MPI : the complete reference.* MIT Press, xii+336p, 1996.

Annexe A

Quelques méthodes de partitionnement

A.1 Méthodes géométriques

Cette classe de méthodes s'appuie sur la stratégie "divide and conquer". En effet, un multi-partitionnement est effectué par bi-partition recursive : on obtient une 2^k-partition après k itérations. La figure 4.6 montre l'obtention d'une 4-partition après 2 itérations. (nous considérons ici la bi-partition qui est la méthode la plus répendue mais le prinicipe reste valable avec les quadri-partition et octa-partition).

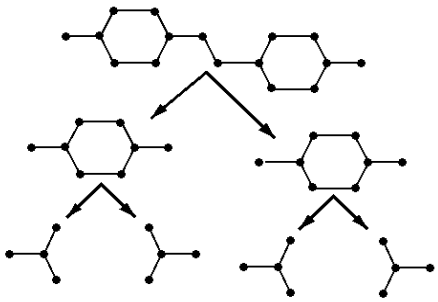


FIG. 4.6 – Partionnement d'un graphe en 4 sous-graphes par bissection récursive

La bi-partition repose ici sur des critères géométriques. Une fois un repère défini (1 origine; 1,2 ou 3 axes), on réduit le maillage à un ensemble de points définis par leurs coordonnées. Ces points peuvent être les noeuds ou les centres de gravités des éléments du maillages. L'idée consiste alors simplement à projeter cet ensemble de points sur un axe donné et à trier ces valeurs. On sépare alors le domaine en deux en assignant la moitié ayant les plus petites valeurs à un sous-domaine, et l'autre moitié à l'autre sous-domaine. Différentes possibilités existent pour le choix de l'axe de projection, nous introduisons les deux principales :
- ORB (Orthogonale Recursive Bissection) : on choisit les axes du repère de façon circulaire
- IRB (Inertial Recursive Bissection) : on choisit les axes principaux d'inertie du domaine maillé
Ces méthodes appartiennent aux méthodes directes dans la mesure où la position finale d'un élément est directement déterminée. Autrement dit, on ne tient pas compte de la partition initiale, et l'assignation des sous-maillages aux processeurs risque de nécessiter un volume de communication important.
Ces méthodes donnent d'assez bons résulats lorsque le domaine est relativement simple et compacte, toutefois la topologie du maillage n'étant pas prise en compte, le volume des communications peut être très important.

A.2 Méthodes multi-niveaux

On considère un graphe issu du maillage à partitionner que l'on note $G = (V, E)$. La structure de base d'un algorithme multi-niveaux est illustrée figure 4.7 et se décompose en trois étapes :
- Contraction :
cette étape permet de réduire la taille du graphe à traiter ; on construit ainsi une série de graphes $(G_0, G_1, ..., G_n)$ tels que $card(G_i) > card(G_j)\ \forall i < j$.
- Partition initiale :
une partition du graphe le plus réduit est réalisée (par bissection recursive par exemple)
- Raffinement et amélioration de la partition :
on projette la partition obtenue au niveau inférieur sur le niveau supérieur et on améliore localement l'interface avec un algorithme de type Kernighan-Lin.

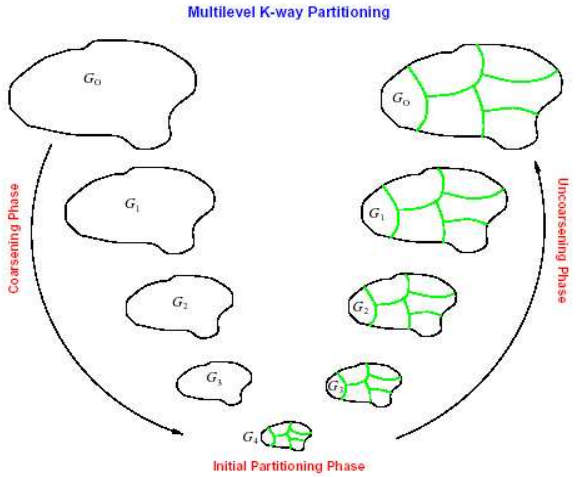


FIG. 4.7 – Schéma d'une approche multi-niveaux (tiré de [6])

Principe des algorithmes de type Kernighan-Lin :
On considére un domaine découpé en deux sous-domaines A et B. L'idée consiste à choisir deux sous-ensembles de A et B que l'on note A' et B'. On échange A' et B' si le nombre d'arrêtes d'interfaces diminue.

Plusieurs bibliothéques de partitionnement s'appuient sur les méthodes multiniveaux. Elles se différencient, principalement, par le choix des méthodes pour obtenir la partition initiale ainsi que pour améliorer localement l'interface lors du raffinement.
Il convient de citer les librairies METIS/ParMETIS[1], qui proposent un ensemble complet de méthodes et algorithmes, Jostle/PJostle[2], sous licence, mais avancé dans l'adaptation des méthodes aux architectures de calcul hétérogènes (cf [1]), ainsi que SCOTCH[3], développé au LaBRI, Laboratoire Bordelais de Recherche en Informatique.

Les versions parallèles de ces librairies restent des méthodes multiniveaux, les trois étapes étant nettement complexifiées.
En résumé, ces méthodes, les plus développées, sont rapides et capables de déterminer une partition de qualité, puisque prenant en compte la topologie du graphe. Par contre, comme les méthodes géométriques, ces méthodes sont des méthodes directes et ne sont pas forcément adaptées au repartitionnent.

[1] L'ensemble de la littérature est présente à l'URL http ://www.cs.umn.edu/ metis.
[2] http ://www.gre.ac.uk/ c.walshaw/jostle
[3] http ://www.labri.fr/Perso/ pelegrin/scotch

A.3 Méthodes locales

On considère un graphe $G = (V, E)$ comme représentation du maillage, ainsi qu'une partition de ce graphe distibuée sur un ensemble de p processeurs. Ces méthodes consistent à utiliser cette partition initiale comme point de départ d'un algorithme de diffusion. Elles sont basées sur la migration des sommets. Les différentes approches permettant de faire migrer les sommets reposent sur le critère introduit par Leiss et Reddy.

On note ω_u la charge de travail et $adj(u)$ le voisinage d'un processeur u. Nous renvoyons au §3.2 où sont définis les notions de charge de travail, et de communications (donc de voisinnage).

◇ **Définition 4.1 (Critère de Leiss/Reddy)** *Soit un processeur u, il choisit son processeur voisin $v_j \in adj(u)$ qui maximise $\omega_{v_i} - \omega_u$ avec $\omega_{v_i} - \omega_u > 0$. S'il n'y a pas unicité, il prend celui ayant le plus petit j. Si aucun de ses voisins ne vérifie ces conditions, aucune requête n'est faite.*

L'utilisation de ce critère permet de former des paires de processeurs tant que la partition peut être améliorée. Une fois ces paires formées, le choix des sommets à migrer se fait en associant à chaque sommet une priorié de transfert, et la migration s'effectue tant que l'équilibre de la charge de travail dans la paire peut être améliorée.
Meshmigration, qui est le repartionneur parallèle que nous allons adapter aux calculs sur grille, repose sur une méthode locale.
Ce type de méthodes est particulièrement efficace dans le cadre du repartitionnement, puisque reposant sur l'utilisation d'une partition initiale. Lorsqu'il n'existe pas de partition initiale, l'algorithme est nécessairement plus long à trouver une partition mais celle-ci est de qualité puisque prenant en compte la topologie du graphe.

Remarque 4.1 *En aucun cas, la liste précédente ne se veut exhaustive, ni ne cherche à établir une quelconque classification des méthodes de partitionnement.*

Annexe B
Solveur de Stokes

Nous présentons succintement, dans cette annexe, un solveur éléments finis permettant de résoudre, en vitesse/pression, un problème de Stokes simplifié. Ce solveur a été developpé au Cemef, et nous renvoyons à la référence [2] pour plus de détails. Dans un premier temps, nous introduisons le problème de Stokes considéré. Ensuite, la formulation éléments finis P1+/P1 est décrite.

B.1 Problème de Stokes

Soit un domaine borné de $\mathbb{R}^3$ noté Ω et de frontière $\Gamma = \Gamma_e + \Gamma_p + \Gamma_s$, avec Γ_e entrée, Γ_p paroies et Γ_s sortie du domaine. Le problème à résoudre est celui de l´écoulement d´un fluide newtonien, régi par la mécanique des milieux continus :

$$\begin{cases} \dfrac{\partial \rho}{\partial t} + \nabla.(\rho v) = 0 \quad (1) \quad \text{équation de continuité} \\ \rho \dfrac{dv}{dt} + \nabla.(\sigma) = \rho f \quad (2) \quad \text{équation de l´équilibre dynamique} \end{cases} \tag{4.1}$$

Tenseur des contraintes :

$$\sigma = s - pI$$

Tenseur des déformations :

$$\epsilon(v) = \frac{1}{2}\big(\nabla v + {}^t\nabla v\big)$$

Relation déviateur des contraintes/déformations linéaire :

$$s = 2\eta\epsilon(v)$$

On considère un écoulement simplifié tel que :
- fluide incompressible ($\rho = C^{ste}$) ;
- forces extérieures de masse et d'inertie négligeables devant les forces de visosité ($f = 0$) ;
- stationnaire ($\frac{\partial .}{\partial t} = 0$)

On aboutit au problème dit de Stokes généralisé linéaire :

$$\begin{cases} 2\eta\nabla.\big(\epsilon(v)\big) - \nabla p = 0 \quad (1') \\ \nabla.(v) = 0 \qquad\qquad\qquad (2') \end{cases} \tag{4.2}$$

Conditions aux limites :
- pression imposée en entrée du domaine : $p = p_0$ sur Γ_e
- contact collant sur les paroies : $v = 0$ sur Γ_p

B.2 Méthode éléments finis : P1+/P1

Nous présentons ici, sans entrer dans les détails, la méthode éléments finis permettant de résoudre en vitesse/pression le système 4.2. Soient $\mathcal{V}$ et $\mathcal{P}$ les espaces fonctionnels associés aux champs de vitesse et pression et définit comme suit :

- $\mathcal{V} = \{u \in (H^1(\Omega))^3 \ / \ u = 0 \text{ sur } \Gamma_p\}$
- $\mathcal{P} = L^2(\Omega)$

Formulation faible :

Trouver (v,p) $\in \left(\mathcal{V} \times \mathcal{P}\right)$ tels que :

$$\begin{cases} \displaystyle\int_\Omega 2\eta\epsilon(v) : \epsilon(u)\mathrm{d}\Omega - \int_\Omega p\nabla.u\mathrm{d}\Omega = -\int_{\Gamma_e} p_0 u.n\mathrm{d}\Gamma, \ \forall u \in \mathcal{V} \\ \displaystyle -\int_\Omega q\nabla.v\mathrm{d}\Omega = 0, \ \forall q \in \mathcal{P} \end{cases} \tag{4.3}$$

On se donne une discrétisation du domaine Ω, notée $(\mathcal{N}, \mathcal{E})$ composée d´un ensemble de tétrahèdres $\{\Omega^e, \forall e \in \mathcal{E}\}$.

On définit les espaces d´approximation comme suit :

- $\mathcal{V}_h = \left\{v_h \in (\mathcal{C}(\Omega))^3 / v_h|_{\Omega^e} \in (P^1(\Omega^e)^3, \ \forall e \in \mathcal{E}\right\}$
- $\mathcal{P}_h = \left\{p_h \in (\mathcal{C}(\Omega))^3 / p_h|_{\Omega^e} \in (P^1(\Omega^e)^3, \ \forall e \in \mathcal{E}\right\}$

Afin de satisfaire la condition INF/SUP, l´interpolation en vitesse est compléter par une fonction bulle. On définit ainsi un troisième espace d´approximation :

- $\mathcal{B}_h = \left\{b_h, \ b_h|_{\Omega^e} \in H_0^1(\Omega) \cap \mathcal{C}^0(\Omega^e), b_h = 0 \text{ sur } \partial\Omega^e, \ \forall e \in \mathcal{E}\right\}$

Problème discret :

Trouver $(v_h, b_h, p_h) \in \left(\mathcal{V}_h \times \mathcal{B}_h \times \mathcal{P}_h\right)$ tels que :

$$\begin{cases} \displaystyle\int_\Omega 2\eta\epsilon(v_h) : \epsilon(v_h^*)\mathrm{d}\Omega - \int_\Omega p_h\nabla.v_h^*\mathrm{d}\Omega = -\int_{\Gamma_e} p_0 v_h^*.n\mathrm{d}\Gamma, \ \forall v_h^* \in \mathcal{V}_h \\ \displaystyle\int_\Omega 2\eta\epsilon(b_h) : \epsilon(b_h^*)\mathrm{d}\Omega - \int_\Omega p_h\nabla.b_h^*\mathrm{d}\Omega = 0, \ \forall b_h^* \in \mathcal{B}_h \\ \displaystyle -\int_\Omega p_h^*\nabla.(v_h + b_h)\mathrm{d}\Omega = 0, \ \forall p_h^* \in \mathcal{P}_h \end{cases} \tag{4.4}$$

Système algébrique équivalent :

$$\begin{bmatrix} A & 0 & B \\ 0 & A_b & B_b \\ {}^tB & {}^tB_b & 0 \end{bmatrix} \begin{pmatrix} V \\ V_b \\ P \end{pmatrix} = \begin{pmatrix} F \\ 0 \\ 0 \end{pmatrix} \tag{4.5}$$

Après simplification, le système équivalent suivant est obtenu :

$$\begin{bmatrix} A & B \\ {}^tB & -C \end{bmatrix} \begin{pmatrix} V \\ P \end{pmatrix} = \begin{pmatrix} F \\ 0 \end{pmatrix}, \text{ avec } C = {}^tB_b(A_b)^{-1}B_b \tag{4.6}$$

Résolution du système :

Le solveur résoud le système précédent, par l´intermédiaire de la librairie PETSC[4], avec une méthode itérative de type résidu minimum (MINRES) avec préconditionnement.

[4] http ://www.mcs.anl.gov/petsc

B.3 Parallélisme

Le solveur est parallisé par une approche SPMD (un seul programme sur de multiples données). Nous présentons les opérations principales d´une résolution parallèle de façon à mettre en avant les caractéristiques attendues d´une partition.
Nous considérons une exécution parallèle sur deux processeurs, P_0 et P_1, pour simplifier les explications.
On considère une partition du maillage sur ces deux processeurs et telle que : un élément appartient à un unique processeur, un noeud peut être partagé par plusieurs processeurs. Chaque processeur effectue l´assemblage des contributions de ses éléments à la matrice locale du système. De la même façon, le second membre est assemblé par sous-domaines. Le nombre d´éléments d´un sous-domaine est donc représentatif d´une certaine charge de travail.
La méthode de résolution du système globale utilisée dans le solveur est la méthode du résidu minimal. Trois types d´opérations font intervenir le parallélisme : les produits matrice-vecteur, les calculs de minimum ou de maximum de vecteurs et les produits scalaires.

Produit matrice-vecteur

On considère le calcul du produit $AX = B$, avec $A = A_{00} + A_{10} + A_{01} + A_{11}$, $X = X_0 + X_1$ et $B = B_0 + B_1$ comme illustrée figure 4.8.

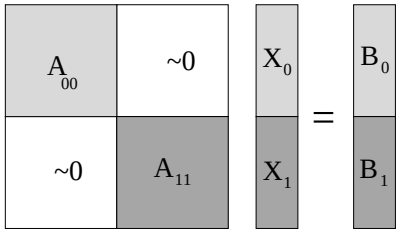


FIG. 4.8 – Distribution des données pour un produit matrice-vecteur par ligne et une matrice quasi diagonale par blocs

La particularité des matrices concernées est d´être "quasi diagonale" par blocs, autrement dit les coefficients des matrices A_{10} et A_{01} sont presque tous nuls. La conséquence sur la distribution des données est qu´il n´est pas nécessaire de distribuer l´ensemble du vecteur X. Les échanges entre processeurs ne concernent donc quelques coefficients. De plus ils sont locaux car l´existence de ces coefficients non nuls est dûe au partage possible de noeuds, donc liée aux interfaces entre sous-maillages.

Calcul de minimum ou de maximum de vecteurs, produits scalaires.

Soit un vecteur $X = X_0 + X_1$ distribué sur les deux processeurs. Chaque processeur peut calculer le minimum du sous-vecteur qu´il possède. Par contre, pour connaitre le minimum de X, une communication globale est nécessaire. L´idée est la même pour le calcul d´un produit scalaire.

L´ensemble des calculs effectués dans ces opérations, pour chaque processeur, est directement lié aux nombres de variables assignés, donc aux nombres de noeuds qu´il héberge.

Cette présentation, très sommaire, des opérations parallèles de la résolution d´un système linéaire, précise un certain nombres de contraintes imposées aux partitionnement. Tout d´abord, **la charge de travail d´un processeur dépend à la fois du nombre d´éléments et de noeuds qu´il héberge**. Le partage de noeuds entre processeurs entraine des étapes de synchronisation (locale et globale) et de communication. **Les communications dites "élément/noeud"**, i.e un élément sur processeur et un de ces noeuds partagé avec un autre, **sont donc représentatives d´un code élément fini**.
Le choix du graphe combiné (présenté au §1.2.2) comme graphe associé au maillage est donc licite lorsque l´application parallèle utilisant le maillage est de type éléments finis.

Résumé

Le projet MECAGRID de l'INRIA, au sein duquel s'est déroulé ce stage, vise à mettre en place une grille de calcul à partir de grappes de PC localisées en région PACA. Du point de vue logiciel, il s'agit d'adapter les applications parallèles concernant la mécanique des fluides multimatériaux à ces nouvelles machines de calcul.

Ces applications de types éléments ou volumes finis, s'appuient sur l'utilisation d'un maillage. Dans le contexte de calculs intensifs s'effectuant sur des machines parallèles, il est nécessaire de partitionner ce maillage et de distribuer les sous-maillages sur les différents processeurs de façon à équilibrer la charge de travail effectué sur chacun des processeurs. *Meshmigration*, code parallèle (MPI) developpé en C++ au Cemef, permet le repartitionnement parallèle de maillages non structurés. L'objectif du stage est d'adapter *Meshmigration*, de manière à prendre en compte, lors du partitionnement, les deux principales contraintes liées à l'architecture des grilles de calcul, à savoir : les processeurs ont des vitesses de calcul hétérogènes et les vitesses de communication entre ces processeurs sont également hétérogènes.

Meshmigration s'appuie sur une méthode locale de repartitionnement parallèle : l'idée consiste à partir d'une partition initiale, à former des paires de processeurs et à échanger, dans ces paires, les éléments et noeuds améliorant la partition. Les modifications ont principalement porté sur le critère d'évaluation d'une partition, sur le critère de formation de ces paires, ainsi que sur la définition des gains donnant une priorité de transfert aux éléments et aux noeuds. L'ensemble des tests effectués, avec des configurations représentatives du calcul sur grille (architecture machine et code de calcul), nous a permis de valider les améliorations apportées à *Meshmigration*