Rapport de fin de thèse

MecaGrid

Olivier BASSET

Responsables :
Hervé Guillard (INRIA) et Thierry Coupez (CEMEF)

1. Introduction

Les machines de calcul basées sur les réseaux sont récemment apparues comme une alternative pour permettre aux utilisateurs d'accéder à des ressources et des puissances de calcul considérables. Des applications parallèles comme SETI@home ont rendues populaire le concept des grilles. De plus, le développement et l'amélioration très rapide des réseaux rendent possible la considération pour des calculs massivement parallèles sur des machines autres que des "superordinateurs". Ces nouveaux outils sont appelés des grilles de calcul, qui ne sont d'autres que des clusters de PC interconnectés. Elles peuvent être vues comme des ordinateurs virtuels distribués et constitués d'un ensemble hétérogène de super nœuds, où chaque super nœud est homogène (ici, un cluster).

L'article [Fos99] est reconnu comme la proposition de développement la plus intéressante pour la nouvelle technologie des grilles de calcul. Le succès de leur développement repose sur plusieurs facteurs : l'amélioration rapide de la puissance des processeurs et du débit des réseaux, la disponibilité des middlewares permettant l'interconnexion de ressources de calcul et de données, l'implémentation (ou l'amélioration) d'algorithmes parallèles hétérogènes, et enfin, la participation et le dévouement des utilisateurs. L'environnement des grilles diffère des machines parallèles standard pour plusieurs aspects :

elles sont hétérogènes dans leur puissance de calcul (processeur et mémoire virtuelle),
elles ont des débits de communication moins performants et plus hétérogènes,
elles sont plus instables de part leur dépendance à des institutions autonomes, et
elles peuvent intégrer des systèmes d'exploitation très différents.

Le développement des grilles de calcul commença dans le milieu des années 1990. Le but est de rendre disponible différentes ressources de calcul et de données sur une grille afin de réaliser des projets scientifiques qui seraient impossibles ou, surtout, trop coûteux dans un site esseulé. Toute une technologie se développe aujourd'hui, et depuis quelques années, sur ce sujet, et ce sont des organisation telles que iGrid [Bro99] ou Globus Alliance [Fos97] qui semblent être les mieux reconnues à l'heure actuelle dans le monde.

Ce chapitre présente différentes problématiques posées lors de la construction d'un tel outil dédié aux calculs en mécaniques des fluides, ainsi que lors de son utilisation avec la librairie MPI. Après avoir montré une vue d'ensemble sur ce qu'il se fait à l'heure actuelle sur les grilles de calcul, les thèmes abordés sont les suivants : le projet MecaGrid et l'ensemble des ressources disponibles pour créer une grille, les principaux intérêts portés sur la grille, les difficultés techniques rencontrées pour connecter des clusters avec adresses IP privées, les technologies employées pour les résoudre, les performances réelles obtenues et leurs conséquences. Puis, plusieurs techniques d'optimisation sont proposées en tenant compte des performances observées. Enfin, des applications sont étudiées : leurs résultats expérimentaux montrent combien il est possible d'améliorer les performances de la grille de calcul avec nos différentes techniques d'optimisation.

2. Etat de l'art des grilles de calcul

2.1. Technologie existante

Bien que chaque grille soit unique, il y a des problèmes de conception récurrents qui apparaissent. C'est justement parce qu'ils sont rencontrés souvent que des organisations développent des solutions les résoudre. Leurs principales préoccupations sont les suivantes :

Sécuriser des données qui, pourtant, doivent être accessibles à un grand nombre de personnes et d'institutions.

Donner une interface facile à utiliser tout en cachant une grande complexité, comme la diversité des ressources et des configurations.

Créer un protocole de transfert de fichiers très rapide, puisqu'il est fort probable que le débit des réseaux soit moins performant qu'avec n'importante quelle autre application.

Intégrer les différentes politiques des institutions.

Même si le concept des grilles de calcul est en développement depuis plus de dix ans, il reste difficile de construire de tels outils. Ainsi, les logiciels *plug-and-play* n'existent, à l'heure actuelle, que pour les *Desktop Computing* et les grilles d'entreprise qui sont simplifiées par le fait qu'elles profitent de leurs propres réseaux et administrateurs. Mais, ce type de logiciel n'existe pas encore pour les grilles de type *Virtual Organizations* (VO). On appelle VO un regroupement d'institutions totalement indépendantes les une des autres. Cependant, d'autres logiciels comme celui de Globus Alliance [Fos97] (http://www.globus.org) permettent de créer des grilles VO dans un laps de temps raisonnable (de l'ordre de quelques mois).

2.2. Ce qu'il se fait sur les grilles de calcul en termes de simulation à grande échelle

Une des plus grandes organisations s'occupant du développement des grilles est l'iGrid [Bro99]. Elle regroupe des centres de recherche du monde entier : on y trouve des participants des USA, d'Australie, du Canada, d'Allemagne, du Japon, de Hollande, de Russie, de Suisse, de Singapour, et de Taiwan. Ils identifient quatre principales utilisations possibles des grilles :

Applications distribuées : calculs parallèles qui sont exécutés sur des machines parallèles, qui font appelle à des librairies (*API*) pour l'allocation des ressources, l'autorisation, authentification et les communications (comme la librairie MPI et le middleware Globus). Comme exemples d'applications à grande échelle, on trouve des calculs en mécanique des fluides, ou des simulations de collisions de trous noirs.

Applications sur demandes (*on demand*) : accès à distance d'instruments tels que des microscopes ou des accélérateurs.

Applications de données intensives : recherche et visualisation basées sur des bases de données.

Applications collaboratives : accès à des calculs ou des bases de données en temps réel, comme dans les diagnostiques médicaux ou la télé-immersion.

Analyse d'images satellites

Parmi les secteurs les plus demandeurs de la technologie des grilles, on trouve l'analyse d'images satellites [Lee96] : des données apportées en très grand nombre par les satellites a généralement besoin d'importantes machines distribuées pour les stocker, les analyser, les visualiser, les partager…

Modélisation climatique

La recherche dans le domaine de la modélisation climatique [Allc01] retrouve environ les mêmes besoins : les ingénieurs font face à des besoins grandissants en ressources. Des simulations de problèmes complexes ont une durée très grande, et une énorme quantité de données résultent de ces calculs sous formes de fichiers qui doivent être analysés. Ils sont donc très intéressés à la création d'une grille qui puissent analyser, transporter et distribuer des quantités importantes de données et gérer un très grand nombre d'utilisateurs. En effets, dans ce domaine, des équipes de chercheurs se trouvent très éloignés les uns des autres tout autour de la planète et doivent se communiquer des données sur les différents climats terrestres.

Etude des tremblements de terre

Dans [Pea04], les auteurs mettent en évidence les progrès considérables que les gilles apportent à l'ingénierie des tremblements de terre. Les ingénieurs investiguent le comportement des structures terrestres avec à la fois des simulations numériques et des expériences physiques. Récemment, une nouvelle approche hybride demande une analyse distribuée des données expérimentales et calculées : ils font donc appel à la technologie des grilles pour construire NEESGrid (*Network for Earthquake Engineering Simulation*). NEESgrid permet de rassembler et accéder à des données géologiques récoltées dans différentes parties du monde par des ingénieurs dispersés, puis de les analyser avec des calculs intensifs et massivement parallèles. Le résultat est l'apparition de toutes nouvelles méthodes de travail bien plus efficaces.

Simulations numériques de la relativité

Un record de taille pour une simulation en relativité numérique a été réalisé sur grille de calcul par les auteurs de [Alle01]. Le logiciel nommé Cactus [Alle99] résous les équations d'Einstein pour simuler numériquement l'évolution d'une vague gravitationnelle. Leur modèle s'appuis sur une méthode de différence finie et une fonction discrétisée sur une grille régulière. En plus de la relativité numérique, Cactus est de plus en plus utilisé à des fins différentes comme en astrophysique, chimie, propagation de fissures, et modélisation des climats. Pour les calculs scientifiques parallèles, il se base sur les librairies et middleware MPI, PETSc et Globus. Une application avec un maillage à environ 720 millions de nœuds a été exécutée avec succès sur les 1500 processeurs d'une grille de calcul. Ils proposent de quantifier la puissance de leur machine en Flops/s :

$$Flops/s = \frac{NbFlops_{total}}{Temps_{total}} \qquad (1)$$

Le nombre de flops total est le nombre d'opérations effectué sur tous les processeurs, le temps total est la durée de l'exécution. Cette définition intéressante nous servira par la suite pour évaluer la puissance de nos outils de calcul. Sur leur grille de calcul, ils obtiennent une

puissance de 42 Gigaflops par seconde, mais, avec plusieurs techniques d'optimisation, ils arrivent à atteindre 249 Gigaflops par seconde. Ces techniques améliorent la répartition de données par partitionnement et diminuent les messages de communication par compression. Cet exemple impressionnant par sa taille montre à quel point les grilles de calculs peuvent fournir des calculateurs très puissants.

Recherche de la Nasa

L'article [Bar99] décrit une expérience menée par la Nasa, dans laquelle une application en calculs de dynamique des fluides à grande échelle est adaptée pour être exécutée de façon performante sur un environnement distribué. Cette grille de calcul est créée grâce au middleware Globus [Fos97]. Un logiciel de simulation numérique basé sur la librairie de MPI [Gro96] et appelé OVERFLOW [Bun95] est utilisé pour expérimenter la grille de calcul.
La Nasa veut développer une nouvelle génération d'outil de design (*next generation design tool*) pour améliorer la précision des simulations et diminuer de moitié les cycles de développement des avions. Afin de réussir un tel challenge, il est nécessaire d'apporter une amélioration conséquente dans la façon de créer, calculer, comprendre, stocker et communiquer les données. Il n'est cependant pas probable que tout ceci puisse passer par l'utilisation de machines parallèles conventionnelles, même très performantes. La Nasa est donc en train de construire une infrastructure à l'échelle des Etats-Unis appelée *Information Power Grid* (IPG). L'IPG a pour but de fournir un accès uniforme à travers une interface pratique, à des ressources très importantes de calcul, de communication, d'analyse et de stockage des données. Pour créer un environnement de calcul transparent avec une bonne scalabilité et adaptabilité, d'importantes ressources hétérogènes et géographiquement dispersées aux Etats-Unis sont reliées. L'utilisation de cette machine parallèle virtuelle unifiée passe par une interface qui ne présente pas de complications par rapport à celle d'une machine plus conventionnelle (comme un superordinateur ou un cluster).

Les deux principales utilisations de cet outil sont les suivantes : la première est la mise à disposition de technologies communes à des collaborateurs géographiquement dispersés et indépendants ; la seconde est de permettre de résoudre des applications extrêmement importantes et massivement parallèles, comme le font les clusters.

Cette application est très importante pour la Nasa qui bientôt, aura besoin de dépasser les ressources habituellement disponibles sur les machines parallèles de leurs sites ou celles de leurs partenaires. Une mauvaise répartition des données, ainsi que des temps de communication trop élevés, sont identifiés comme leurs principales sources de mauvais résultats. L'hétérogénéité de leur grille IPG et les distances qui séparent leurs différentes ressources de calculs sont donc à la base de leurs problèmes rencontrés au sujet de la performance. Des techniques d'optimisation de répartition de la masse de calcul sont appliquées avec un certain succès, mais ils estiment que plusieurs efforts doivent encore être faits pour que IPG soit un environnement utile et accessible aux calculs à très grande échelle. Avec les applications d'un véhicule de retour d'équipage et d'un hélicoptère de l'armée, l'utilisation de la grille de calcul est comparée à celle d'un cluster. Après avoir obtenu de mauvais résultats sur grille avec un maillage à 2.5 millions de nœuds, ils proposent des techniques d'optimisation telles qu'un schémas en temps décalé (*deferred scheme*) et une répartition adaptée de la masse de calcul et des communications. C'est avec le logiciel MeTiS [Kar95], plus quelques améliorations qu'un partitionnement optimisé est effectué. Les résultats qu'ils obtiennent sont assez encouragent, mais ils considèrent que beaucoup de

travail et de réflexions doivent encore être menés pour pouvoir utiliser la grille dans des buts de production.

3. Le projet MecaGrid

3.1. Description du projet

Le projet MecaGrid est soutenu par l'ACI GRID – Action Concertée Incitative (ACI) Globalisation de Ressources Informatiques et des Données (GRID) – du ministère de la recherche depuis octobre 2002. L'ACI GRID a pour objectif de renforcer les recherches françaises menées autour des grilles de calcul ou de données. Le projet MecaGrid vise à bâtir une grille de calcul régionale (en PACA) à partir de grappes de PC (clusters) pour des applications de calculs intensifs parallèles en mécanique des fluides hétérogènes. Plus précisément, la grille de calcul sera dédiée à la simulation numérique multi matériaux, dont des écoulements multi fluides avec capture d'interfaces. Pour cela, des codes de calcul utilisant la librairie MPI doivent pouvoir être exécutés sur la grille sans pour autant avoir à en changer l'implémentation parallèle. Les clusters situés sur plusieurs sites différents dans la région PACA qui forment MecaGrid sont des clusters de production. C'est-à-dire qu'ils sont utilisés constamment et individuellement par les centres de recherche auxquels ils sont rattachés. Ceci est un aspect très important dans l'approche que l'on aura vis-à-vis de cet outil de calcul. Dans sa phase initiale, MecaGrid regroupe les clusters du CEMEF (Centre de Mise en Forme des Matériaux de l'Ecole des Mines de Paris à Sophia-Antipolis), de l'INRIA Sophia-Antipolis, et de l'IUSTI (Institut Universitaire des Systèmes Thermiques et Industriels) à Marseille.

Le but final du projet est de mettre en valeur le concept des grilles de calcul dans le cadre de simulation à grande échelle en mécanique des fluides hétérogènes. Il s'agit de montrer ce qu'il est possible de faire à l'heure actuelle avec les moyens et les ressources disponibles. De plus, cette étude permet de mieux comprendre le fonctionnement très particulier des grilles de calculs et d'identifier les configurations et les utilisations à améliorer dans l'avenir. Les méthodes numériques présentées dans cette thèse ont pour but de permettre la réalisation de simulation à grande échelle sur MecaGrid. C'est pourquoi, elles sont toutes choisies et développées de sorte à garder au maximum une simplicité de résolution et de stockage, dans l'optique de permettre des applications de grande taille.

3.2. Difficultés administratives

Les grilles de calcul de type *Virtual Organizations* sont particulièrement délicates à mettre en place pour plusieurs raisons. Les ressources de calcul sont situées à différents endroits géographiques, et les institutions indépendantes auxquelles elles appartiennent ont leurs propres priorités, administrateurs, gestionnaires de batch, systèmes de queues, procédures de sécurité, matériels, et utilisateurs. Tout ceci représente autant de conflits potentiels lorsque l'on parle de coordination : les règles locales peuvent contredire les règles communément décidées pour la globalité de la grille. De plus, les différents clusters sont des

structures « vivantes », qui évoluent constamment et indépendamment des autres clusters de la grille (exemple : changement de configuration ou de version de librairie, évolution des machines…). Il apparaît donc évident qu'un administrateur « global » doit coordonner tous les changements de configuration qui interviennent localement, et veiller au bon fonctionnement global de la grille de calcul.

3.3. Ressources disponibles

Les ressources de calcul qu'utilise MecaGrid sont des clusters situés sur les différents sites des membres du projet. Ces ressources sont assemblées pour donner naissance à un outil de calcul parallèle de plus grande taille que ceux qui sont disponibles localement.
Le site de l'INRIA Sophia a 2 clusters différents reliés à MecaGrid. Le premier (appelé *nina*) contient 16 biprocesseurs Xeon à 2 GHz. Ils sont reliés entre eux par un réseau Ethernet à 1 Gigabit/s (1 Gb/s). Le deuxième (appelé *pf*) contient 19 biprocesseurs Pentium III à 933 MHz. Ils sont eux reliés par un réseau Ethernet à 100 Mégabit/s (100 Mb/s). Le cluster de l'IUSTI à Marseille est constitué de 32 monoprocesseurs Pentium IV à 2 GHz avec un réseau Fast-Ethernet à 100 Mb/s. Enfin, le cluster situé sur le site du CEMEF contient 32 biprocesseurs Pentium III à 1 GHz reliés entre eux par un réseau Fast-Ethernet à 100 Mb/s.
Ce tableau résume toutes les caractéristiques de l'architecture théorique de MecaGrid :

Site	cluster	Adresse IP	CPUs	GHz	RAM/nœud	CPUs/nœud	LAN
INRIA Sophia	nina	publique	32	2.0	1 Go	2	1000 Mb/s
INRIA Sophia	pf	publique	38	1.0	512 Mo	2	100 Mb/s
IUSTI	iusti	privée	30	2.0	512 Mo	1	100 Mb/s
CEMEF	cemef	privée	64	1.0	512 Mo	2	100 Mb/s

Tableau 1 – Ressources de calculs disponibles

Chaque cluster a sa propre adresse IP, soit privée, soit publique, et il y a un total de 164 processeurs disponibles. Leur puissance de calcul est quantifiée en GHz, et on observe un facteur deux entre la fréquence des plus lents et des plus rapides. RAM/nœud montre la quantité de mémoire vive disponible sur chaque nœud (en Gigaoctet et Mégaoctet). CPUs/nœud est le nombre de processeurs que contient chaque nœud : certain sont des monoprocesseurs (un processeur par nœud), et d'autres sont des biprocesseurs (deux processeurs par nœuds). Enfin, le débit des réseaux locaux (LAN ou *Local Area Network*) est mesuré en Mégabit par seconde (Mb/s).

Une des informations les plus importantes à retenir du tableau *(1)* est la présence d'adresses IP publiques pour certains clusters et privées pour d'autres. Nous verrons par la suite les difficultés techniques que cela entraîne sur la construction de la grille, et les solutions que différentes technologies peuvent apporter pour les résoudre.

La colonne RAM/nœud du tableau *(1)* montre la quantité de mémoire virtuelle disponible sur chaque nœud. Or, les clusters à biprocesseurs (ceux de l'INRIA et celui du CEMEF) ont deux processeurs par nœud qui doivent se partager, non seulement la quantité de RAM contenue sur le nœud, mais aussi l'accès à cette mémoire. En effet, il n'existe qu'un seul bus d'accès à la mémoire pour deux processeurs d'un même nœud. Nous verrons par la suite que ce dernier point est loin d'être négligeable lorsque l'on parle du niveau de performance. Une distinction importante doit être dressée au sujet des deux utilisations possibles de ces biprocesseurs (emploi de un ou deux processeurs par nœud). Par exemple, si une simulation est lancée avec

deux processus par nœud sur le cluster du CEMEF, la mémoire disponible pour chaque processeur n'est plus de 512 Mo, mais de 256 Mo, si l'on considère que les données sont parfaitement distribuées. Une étude plus approfondie suivra au sujet des performances qu'offrent ces deux utilisations des biprocesseurs.

3.4. Intérêt d'une grille de calcul

L'intérêt que l'on porte sur les grilles de calcul à l'heure actuelle porte sur plusieurs points. Tout d'abord, avoir la possibilité d'effectuer des calculs plus important que sur un cluster isolé peut constituer un premier objectif. La quantité de RAM que contiennent les nœuds représente une limite dans l'utilisation des processeurs ; et de ce fait, plus la simulation est de taille importante, plus le nombre de processeurs nécessaires à l'exécution est grand. Il est donc appréciable d'avoir accès à des ressources de calcul supérieures à celles que l'on peut généralement avoir dans les centres de recherche ou de production avec un financement qui reste raisonnable. Relier des clusters de production indépendants entres eux semble largement plus avantageux que d'acquérir une machine contenant la puissance réunie de cette manière.

Ensuite, le contexte des simulations à grandes échelles entraîne des problèmes qui interviennent lorsque l'on n'a accès qu'à un cluster local. Plus particulièrement, ce sont le nombre des processeurs réservés et les temps de calcul qui peuvent créer des obstacles sur un cluster de production isolé. En effet, l'exécution de gros calculs demande souvent une réservation d'un grand nombre de processeurs sur une longue durée, ce qui a pour conséquence d'entraîner deux inconvénients majeurs. Premièrement, de telles machines étant utilisées en permanence et par plusieurs personnes, il devient parfois difficile de trouver un nombre suffisant de processeurs disponibles afin d'effectuer la réservation et de commencer la simulation. De cette manière, une requête peut rester bloquée dans la file d'attente pendant une période relativement longue avant de pouvoir être exécutée. Deuxièmement, il est quasiment inenvisageable de réserver ne serait-ce que la moitié ou le trois-quarts des ressources de calcul d'un cluster pendant une période dépassant les quelques jours, ou la semaine. Ceci entraînerait une saturation, et de longues files d'attentes se formeraient pour les autres utilisateurs.

MecaGrid est conçue pour donner des solutions à ces différentes préoccupations. En reliant plusieurs clusters de production, les ressources de calculs sont considérablement augmentées, et des calculs de grande taille dépassant les possibilités d'un cluster isolé deviennent potentiellement réalisable. De plus, les processeurs disponibles étant largement plus abondants sur la grille, en réserver un grand nombre sur une longue période ne représente plus un problème, ni pour l'intéressé (temps d'attente diminué), ni pour les autres utilisateurs (pas de saturation de la file d'attente). Par exemple, il est clair que de réserver 24 nina parmi les 32 disponibles (ce qui correspond à 75% des ressources) est bien plus problématique que de réserver 24 processeurs sur les 164 que contient la grille (ce qui représente à peine 15% des ressources).

Pour conclure, MecaGrid n'a pas la prétention de fournir un outil de calcul plus performant qu'un cluster, mais de rendre disponible des ressources bien plus importantes, tout en optimisant au maximum ses performances. Reste à déterminer la technologie à utiliser, puis à étudier les caractéristiques et l'utilisation d'un tel outil de calcul.

3.5. Technologie de MecaGrid

Difficultés techniques

Les administrateurs des centres de recherche de l'INRIA, du CEMEF et de l'IUSTI ont participé activement à l'élaboration de cette grille de calcul sur laquelle des applications implémentées avec la librairie MPI doivent pouvoir être exécutées sans en changer le code. Ils ont collaborés ensemble pour mettre en place l'architecture et déterminer les technologies nécessaires pour y parvenir en tenant compte des différentes politiques (voir l'introduction). La principale difficulté technique réside dans la connexion de plusieurs clusters à adresses IP publiques et privées. En effet, la plupart des middlewares pour grilles n'acceptent pas les adresses privées.

Le fait qu'il y ait des clusters à adresse IP privée et d'autres à adresse publique pose un problème technique pour bâtir les liaisons. La communication entre deux nœuds arbitraires de sites différents, comme le nécessite la librairie MPI, sous entent que les LAN de tous les clusters soit « visibles » ou accessibles par internet. Or, les paquets TCP ne peuvent pas circuler dans un réseau global si une des parties (source ou destination) a une adresse IP privée. Il faut donc trouver un moyen pour permettre la communication inter processeur dans la globalité de la grille.

Solutions envisageables

Une solution serait de concevoir une grille qu'à partir de cluster à adresse IP publique, mais pour des raisons administratives, et pour rassembler le maximum de ressources un autre moyen doit être employé.

La deuxième possibilité envisageable est de faire tourner MPICH avec des connexions distantes sécurisées de type SSH. MPICH est une implémentation spéciale de la librairie MPI [Gro96] v1.1 standard pour les grilles de calcul. Les problèmes engendrés par cette technique se situent au niveau de la communication inter clusters et sont liés à la façon dont elle est gérée par MPICH. Lors du lancement des processus MPI, la connexion directe sur chaque nœud, n'est pas permise. A l'exécution, et suivant les communications requises, le service CH_P4 de MPICH est amené à ouvrir des connexions supplémentaires entre les noeuds sur des ports dont nous ne connaissons pas à l'avance le numéro. Cela pose un gros problème de sécurité car il faudrait autoriser des connexions sur n'importe quel port, ce qui est incompatible avec la politique de sécurité des réseaux des différents sites. Une telle solution (MPICH/SSH) a donc vite été écartée à cause des modifications qu'elle oblige à opérer sur les architectures réseaux locales.

Une troisième solution pour relier des clusters à adresse privée consiste à créer un VPN (*Virtual Private Network*) qui, lui, est entièrement compatible avec MPI. Ce type de réseau fourni un outil totalement transparent pour gérer les envois de messages. Avec l'installation du VPN, il n'est pas nécessaire d'avoir une adresse IP publique par nœud, mais seulement une par site, qui est abritée par une machine « frontale ». Cette frontale n'est pas un nœud, mais une machine dédiée au réseau VPN. Bien que les clusters de l'INRIA ai des adresses publiques, ils sont considérés comme aillant des adresses privées pour qu'ils puissent intégrer ce VPN qui établi les communications au sein de MecaGrid. Plus précisément, le VPN crée des tunnels entre les toutes les machines frontales qui elles ont des adresses IP publiques. C'est ainsi que les communications entre les différentes ressources de calcul

géographiquement dispersées vont s'opérer : chaque paire de frontale est connectée entre elles par un tunnel qui crypte les données, encapsule et compresse les paquets, puis les transmet. Le cryptage assure une certaine sécurité, et la compression permet d'améliorer le débit à travers les tunnels. Pour cela, on utilise CIPE, un logiciel simple qui gère les tunnels en transmettant les paquets TCP sous forme de paquets UDP encryptés sur des ports définis par l'utilisateur. Le VPN fait en sorte que chaque processeur peut communiquer avec n'importe quel autre processeur de la grille, et ce, qu'ils fassent parti du même site ou pas. Les communications locales (entre deux processeurs du même site) se font par le réseau local (LAN), tandis que pour les autres, c'est la frontale qui est considérée comme le lieu de passage obligatoire. Ainsi, un message lancé depuis un nœud du site de l'INRIA jusqu'à autre nœud situé à l'IUSTI passe en premier dans la frontale de l'INRIA. Celle-ci a été configuré pour lancer le message dans le tunnel approprié, en direction de la frontale de l'IUSTI, qui elle, décrypte et décompresse les données avant de les rediriger vers le nœud de destination par le LAN. En fait, le VPN fonctionne comme si tous les processeurs faisaient parti d'un WAN (Wide Area Network).

Schémas conceptuel de MecaGrid

La figure *(1)* représente de façon schématique l'architecture théorique de MecaGrid. Les machines frontales par lesquelles passent les communications inter sites sont représentées par des carrés. Les nœuds reliés entre eux en forme d'étoile représentent les différents clusters. Enfin, les connexions inter sites qui transitent par internet sont en noir : ce sont les tunnels.

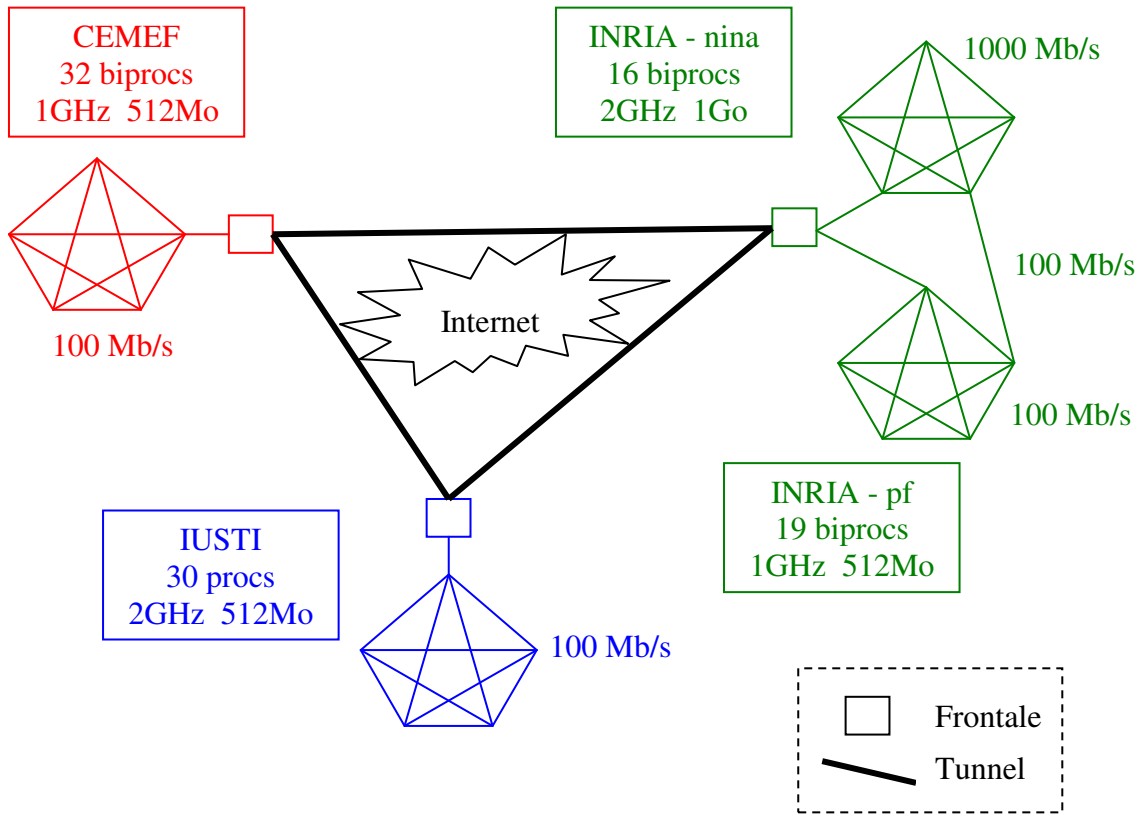

Figure 1 – Architecture théorique de MecaGrid

Ce VPN peut être mis en place avec le middleware de Globus Alliance [Fos97]. Le prochain paragraphe détail l'utilisation des services inclus dans Globus2 qui nous apportent les technologies nécessaires à la construction du VPN.

Principes et fonctionnement de Globus2

Essentiellement pour des raisons de sécurité et d'administration, ce sont Globus 2.2.4 et MPICH-G2 [Fos98] qui ont été choisis pour construire l'infrastructure de MecaGrid et atteindre l'objectif final : fédérer un ensemble de ressources et fournir à l'utilisateur une grille de calcul dont l'utilisation soit aussi simple qu'une machine parallèle classique.
MPICH-G2 est une implémentation spéciale de la librairie MPICH en utilisant exclusivement la version 2 de l'outil Globus.

Globus 2.2.4 est une boite à outils composée d'un ensemble de services dédiés au calcul sur grille. Ce middleware ne résous pas tous les problèmes que l'on peut rencontrer mais fourni plusieurs solutions intéressantes. Ses services interviennent dans les domaines de la sécurité (authentification, identification, confidentialité), de la gestion des ressources, de la communication (avec notamment un support concernant la librairie MPI), et de la gestion des requêtes. Pour l'interrogation et l'allocation des ressources, la communication entre ces services se fait grâce au langage RSL. Les principaux services dont nous avons besoin sont : GRAM, MDS et GridFTP. Voici brièvement le rôle de chacun d'eux :

GRAM regroupe le *gatekeeper* et le *jobmanager*. Le *gatekeeper* sert à authentifier les requêtes RSL qui arrivent et détermine quelle est l'identité locale qui correspond au propriétaire de cette requête. Pour cela, il existe un fichier (grid-mapfile) associant un ensemble de certificats à un ensemble d'identités locales. La connexion au *gatekeeper* se fait sur le port 2119 en TCP et en provenance d'un port éphémère (supérieur à 1023) de la machine cliente. La requête est ensuite passée au *jobmanager* qui traduit la requête RSL dans le langage du gestionnaire de soumission local (LSF, PBS, ...).

MDS constitue le service d'information de la grille et est basé sur OpenLDAP 2.0. Cet annuaire contient des informations sur tous les nœuds de la grille. MDS comprend le GRIS et le GIIS. Le GRIS fournit les informations sur une ressource spécifique (ex : un cluster). Le GIIS rassemble l'ensemble des informations mises à disposition par les GRIS et permet de sélectionner les machines de part leurs caractéristiques. Les connexions éventuelles se font à partir d'un port éphémère vers le port 2135 en TCP.

GridFTP est l'équivalent sécurisé de FTP. Ce protocole fonctionne avec deux canaux : un canal de contrôle et un canal de transfert de données. La connexion de contrôle se fait à partir d'un port éphémère contrôlé par le client vers le port 2811 du serveur en TCP, et la connexion de transfert se fait d'un port éphémère du client vers un port éphémère du serveur.

L'ensemble des communications générées doit, évidemment, être authentifié et pour cela Globus utilise les certificats X.509 dont le principe est détaillé ici. Globus assure la sécurité de la grille grâce à GSI. Le GSI vérifie la provenance d'une requête et l'authenticité d'un service, à l'aide de certificats électroniques cryptés par un algorithme asymétrique.
Le principe du cryptage asymétrique est le suivant. Chaque utilisateur génère son couple de clé privée/publique qui lui servira à être authentifié par le module de sécurité de la grille. La clef privée est gardée par l'utilisateur alors que la clef publique est distribuée à tous. Si l'utilisateur crypte un message avec ça clef privée, tout le monde pourra le décrypter avec la clef publique et l'origine du message (utilisateur) sera garantie car seul l'utilisateur peut crypter un message avec sa clef privée. Si une autre personne crypte un message avec la clef publique de l'utilisateur, seul celui-ci sera capable de le décrypter et donc de connaître le

contenu. Un double cryptage (clef privé de A/clef publique de B) permet alors de certifier l'origine du message (venant de A) et que seul B peut le lire.
En résumé, dans Globus :

- l'utilisateur génère son couple de clefs
- garde sa clef privée dans un endroit sécurisé
- envoie sa clef publique à l'Autorité de Certification (AC)
- l'AC fabrique alors un certificat en cryptant avec sa propre clef privée :
  - le nom de l'AC
  - le sujet (nom de l'utilisateur, organisme, ...)
  - la clef publique de l'utilisateur
  - les dates de validité
- l'AC renvoie le certificat à l'utilisateur

A l'utilisation, le service de la grille (ex : *gatekeeper*) qui possède la clef publique de l'AC en question pourra décrypter le certificat de l'utilisateur et être sûr que la clef publique associé à l'utilisateur est la bonne. Le service demandera sous forme de challenge, à l'utilisateur de crypter un message pour voir si la clef privée détenue par l'utilisateur correspond à la clef publique certifiée par l'AC. Si tout se passe bien, l'utilisateur est considéré comme identifié de façon sûre. Les machines constituant la grille utilisent le même système de certificat pour passer les challenges d'authentification.

3.6. Conclusion

A ce stade, la grille de calcul est élaborée avec succès. Les différents problèmes rencontrés ont tous été résolus grâce aux technologies disponibles à l'heure actuelle. MecaGrid, qui relie les quatre clusters des centres de recherche participant au projet, est prête à être utilisée. Cependant, pour plusieurs raisons évoquées précédemment, il existe des moments où de nouveaux problèmes apparaissent, mais l'administrateur global est là pour restabiliser la grille. Les premières utilisations de MecaGrid sont consacrées aux tests de performance afin de mieux comprendre les caractéristiques de ce nouvel outil de calcul. Ensuite, une recherche importante sur l'optimisation d'utilisation sera menée pour contrebalancer les inconvénients de cette machine parallèle très particulière. Enfin, tout sera réuni pour permettre à l'utilisateur d'exécuter des applications à grande échelle dans le cadre de simulations numérique d'écoulement de fluides hétérogènes.

4. Performances réelles de la grille

4.1. Outil de mesure des performances

La partie précédente présente l'architecture théorique de MecaGrid, mais nous étudions ici la réalité, en contraste avec la théorie. Afin de connaître l'architecture réelle de la grille sous Globus, nous avons développé un outil, nommé Performance, qui mesure à la fois la vitesse des processeurs, et le débit de tous les réseaux que contient la grille de calcul. La durée d'exécution d'un test est d'environ 30 secondes. Voici comment s'opère la prise des mesures qui caractérisent la machine de calcul parallèle.

Vitesse des processeurs

Afin de mesurer la vitesse des processeurs, Performance mesure le temps que met chaque processeur pour calculer un produit scalaire de 100 000 réels codés en double précision. Ainsi, nous obtenons une mesure de la vitesse des processeurs en Mégaflops par seconde. Le flop correspond au nombre d'opérations effectuées (addition, soustraction, multiplication, division) sur des réels. Plus la mesure en Mégaflops par seconde (Mflops/s) est élevée, plus le processeur est puissant.

Débit des réseaux

L'idée est cette fois-ci de mesurer le temps nécessaire pour envoyer un message de données d'un processeur à un autre. Notre outil Performance mesure ces temps de communication en envoyant un paquet de 1000 réels en double précision (test de type « ping ») entre chaque pair de processeurs. Nous pouvons ainsi quantifier la vitesse des communications, ou le débit des réseaux, en Mégaoctets (Mo) par seconde.

4.2. Performances de la grille

Nous utilisons ici l'outil présenté précédemment pour évaluer les performances de MecaGrid. Pendant toute la durée du test, le programme est lancé plusieurs fois à intervalles de temps réguliers afin de mettre en évidence des éventuelles évolutions du débit des réseaux et de la vitesse des processeurs au sein de la grille de calcul.

Le premier enjeu est de mesurer les performances réelles de la grille. Notamment, c'est le débit du réseau reliant les trois sites qui n'est pas a priori connus avec précision, et qui a besoin d'être mesuré. Ainsi, nous allons pouvoir estimer le degré d'hétérogénéité de la grille, ce qui est très important lorsqu'il s'agit de calcul parallèle. En effet, la masse de calcul et les données doivent être réparties équitablement entre tous les processeurs en tenant compte de l'architecture de la machine parallèle. Dans le cas d'une machine parallèle homogène (comme un cluster), il s'agit simplement d'une division par le nombre de processeurs utilisés pour effectuer le calcul : c'est un partitionnement homogène (même masse de calcul et même quantité de donnée sur tous les processeurs). Comme le montre l'architecture théorique de MecaGrid, nous nous attendons à faire face à une machine de calcul hétérogène de part ses ressources. Les quantités mesurées dans cette partie ont pour but de déterminer l'hétérogénéité de la grille, pour ensuite pouvoir les étudier et optimiser l'utilisation d'un tel outil. Suivant les résultats obtenus ici, il faudra notamment, ne plus partitionner de façon homogène, mais d'une manière à tenir compte de la puissance de chaque processeur et de la vitesse des différents réseaux.

Le deuxième enjeu consiste à savoir s'il est nécessaire de réajuster la répartition de la masse de calcul pendant la simulation numérique. La partition effectuée grâce aux caractéristiques mesurées vise à répartir au mieux la masse de calcul entre tous les processeurs disponibles. Cependant, si les caractéristiques de la grille changent au cours du temps, il est possible que la partition faite au début du calcul ne soit plus optimisée et ralentisse trop l'exécution du programme. On serait dans ce cas amené à repartitionner, puis continuer le calcul avec une nouvelle partition optimisée. Il faut donc être très attentif à la façon dont les caractéristiques de la grille évoluent au cours du temps.

Vitesse des processeurs

Le graphe suivant montre la vitesse des processeurs en Mflops/s en fonction du temps lors d'un test de performance avec un processeur sur chaque cluster : nina, pf, cemef et iusti. Un test de Performance a été lancé toutes les 30 min pendant 24h.

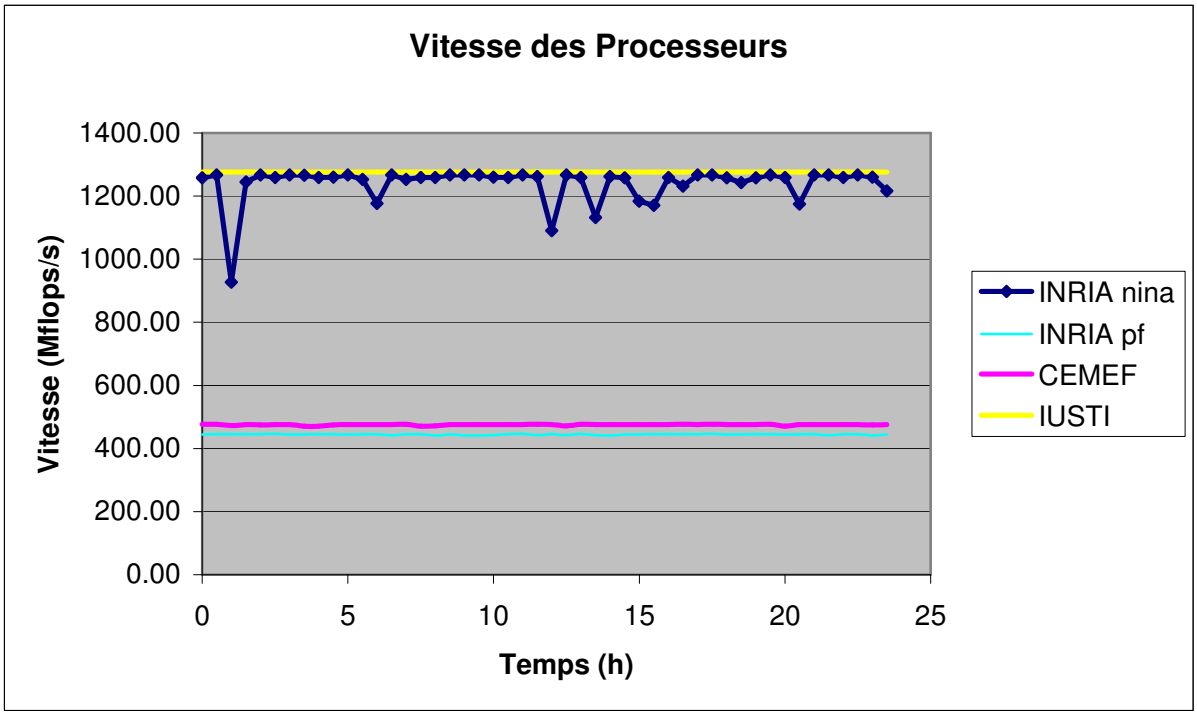

Figure 2 – Puissance des processeurs au cours du temps

D'autres tests ont été exécutés de la même façon en utilisant cette fois-ci 2 puis 4 processeurs sur chaque cluster. Aucune différence notable n'y apparaît en comparaison avec les résultats précédents. Ceci montre que les performances mesurées ici sont indépendantes du nombre de processeurs utilisés. Voici les moyennes des vitesses et des écarts types observées dans tous les tests effectués :

Processeurs	nina	pf	cemef	iusti
Mflops/s	125.71	44.45	47.59	127.52
Ecart Type %	3.03	0.33	0.11	0.19

Tableau 2 – Puissance moyenne des processeurs

Ces résultats sont conformes à l'architecture théorique de la grille. Par contre, la vitesse des processeurs nina de l'INRIA varie beaucoup plus que les autres : leur écart type moyen est de 3%, tandis que les autres vitesses ont un écart type moyen inférieur à 0.33%. Ces variations peuvent être dues au fait que ce sont des biprocesseurs. Mais, puisque les vitesses des autres biprocesseurs de la grille n'ont pas une telle variation, ceci ne représente pas une explication suffisante. Cependant, en observant la courbe de la puissance des nina, les variations aléatoires ne paraissent pas alarmantes, car, ce n'est qu'à certains moments bien précis et pendant une durée très courte que des bouleversements apparaissent. Le reste du temps, la vitesse parait constante.

Débit du réseau intra site (interne à chaque cluster)

La figure *(3)* montre le débit du réseau interne à chaque site en Mo/s en fonction du temps. Ces résultats ont été obtenus par des tests de Performance toutes les 30 min pendant 24h avec 2 processeurs sur chaque cluster.

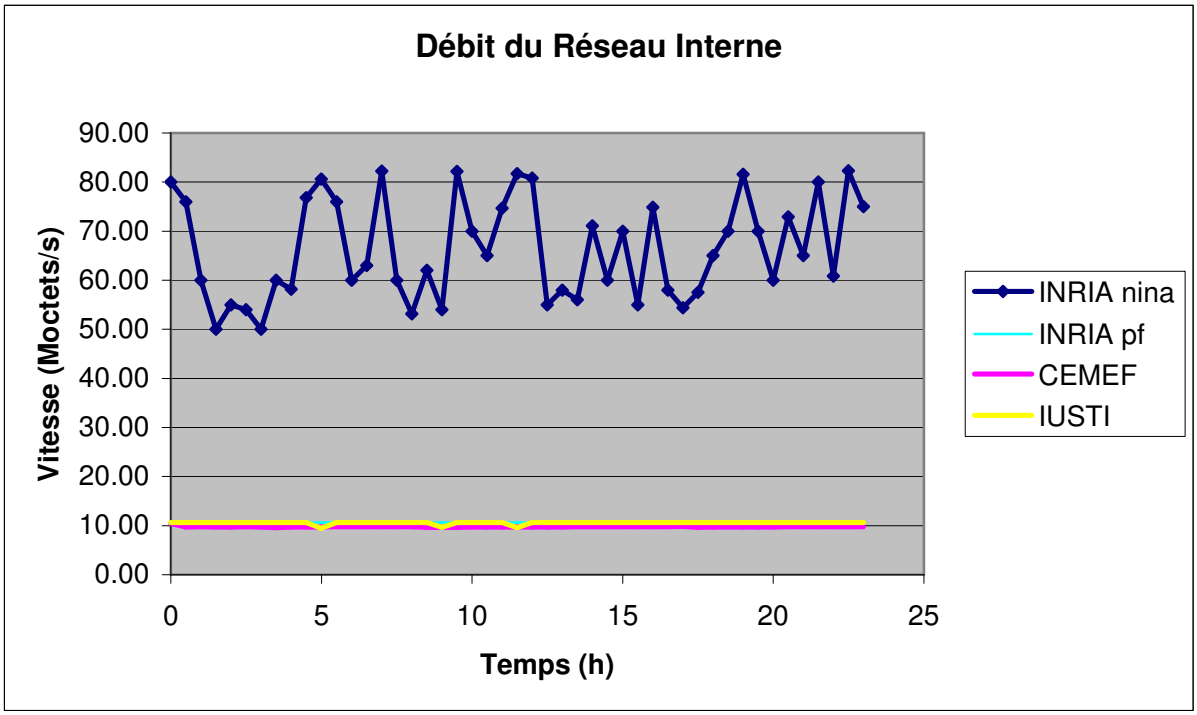

Figure 3 – débit des réseaux internes à chaque cluster en fonction du temps

D'autres tests ont été exécutés de la même façon en utilisant cette fois-ci 4 processeurs sur chaque cluster. Aucune différence notable n'y apparaît en comparaison avec les résultats précédents. Ceci montre que les débits des réseaux internes mesurés ici sont indépendants du nombre de processeurs utilisé. Voici les moyennes des débits (converties en Mégabits/s pour pouvoir les comparer au mesures théoriques) et des écarts types observées dans tous les tests effectués :

LAN	nina	pf	pf – nina	cemef	iusti
Mb/s Théorique	1000.00	100.00	100.00	100.00	100.00
Mb/s Réel	509.69	86.32	89.25	84.05	86.65
Ecart Type %	28.01	0.26	0.25	0.71	0.59

Tableau 3 – Débit moyen des réseaux internes

En comparant les débits théoriques et les débits réels observés, on s'aperçoit que le réseau interne au cluster INRIA-nina est assez loin de ce qu'il peut être (presque 50%), mais que les autres sont plus proches de leur valeur maximale (environ 90%). Nous remarquons aussi que le débit du réseau entre les processeurs nina varie beaucoup : son écart type moyen est de 28%, alors que les autres débits ont un écart type moyen de 0.5%.
Il est assez difficile d'expliquer les mauvais résultats du réseau entre les nina. Le fait que le cluster de l'INRIA ait une adresse IP publique pourrait éventuellement expliquer cela. Mais, si c'était le cas, le réseau entre les processeurs pf (qui ont eux aussi une adresse IP publique), serait alors dans le même cas. Cette explication ne parait donc pas satisfaisante. L'usage du

réseau par les autres utilisateurs du cluster INRIA-nina semble alors la raison la plus probable qui puisse influencer de la sorte l'état du réseau interne.

Débit du réseau inter site (entre chaque site)

Sur la figure suivante, nous avons le débit du réseau externe entre chaque site en Mo/s en fonction du temps. Ces résultats ont été obtenus par des tests de Performance toutes les 30 min pendant 24h avec 1 processeur sur chaque site (3x1).

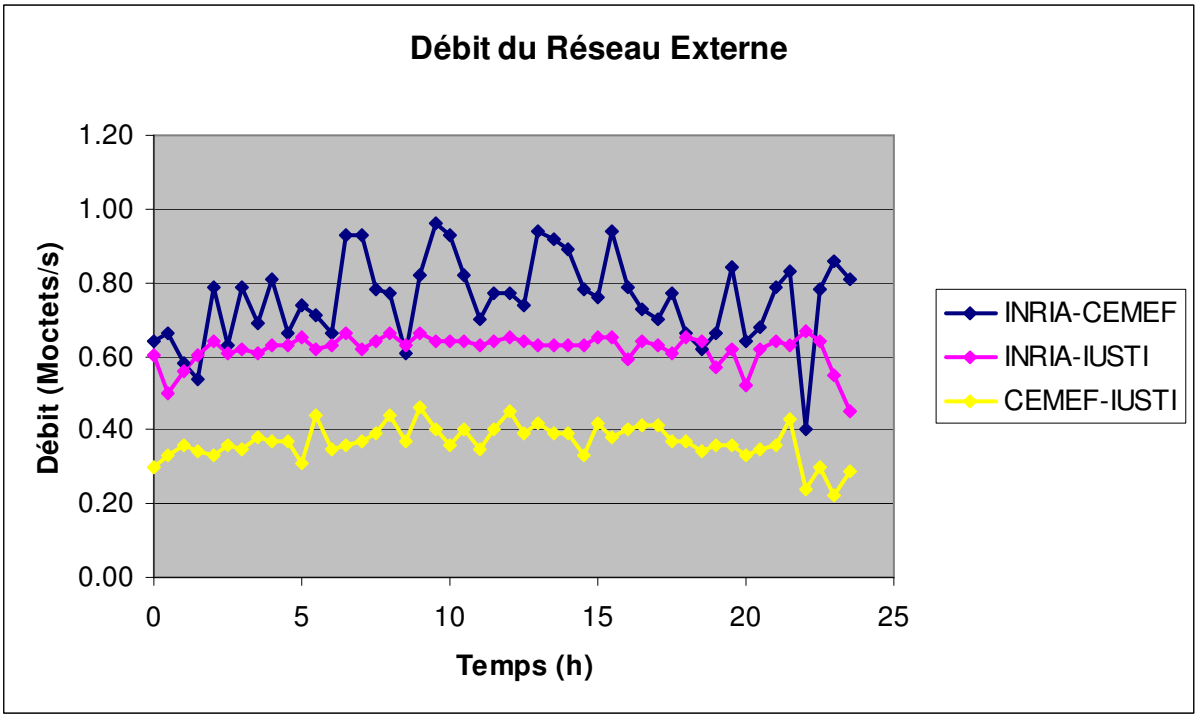

Figure 4 – Débit moyen des réseaux inter sites

Il nous faut maintenant vérifier si le fait d'augmenter le nombre de processeurs influe sur la vitesse du réseau inter site : avoir plus de processeurs augmente les communications et pourrait saturer les réseaux de la grille. Ainsi, nous avons fait d'autres mesures du débit inter site avec 2 puis 4 processeurs par site (3x2 et 3x4). Les résultats obtenus ne présentent pas de différence significative par rapport au cas 3x1. En moyenne, nous avons :

Réseau Externe	nina – cemef	nina – iusti	cemef – iusti
Mb/s	7.21	5.03	3.69
Ecart Type %	15.00	7.91	13.98

Tableau 4 – Débit moyen des réseaux inter sites

Les variations de débit sont assez importantes : leur écart type moyen peut aller jusqu'à 15%. Ce n'est pas étonnant, vu que les trois sites de la grille sont reliés par une connexion internet, dont le débit dépend forcément de nombreux paramètres aléatoires. En plus d'avoir des variations importantes, cette connexion à en moyenne une performance très faible par rapport aux débits constatés au sein des différents clusters. Ceci a pour conséquence d'avoir une grille de calcul très hétérogène dans ses performances de réseau, puisque le rapport entre le débit le plus rapide (509.69 Mb/s à l'intérieur du cluster INIRA-nina) et le plus faible (3.69 Mb/s entre le CEMEF et l'IUSTI) est d'environ 130.

Evaluation des fréquences de variation des performances

Dans tous les tests effectués, la vitesse des processeurs – autre que nina – ne varie que très peu (écart type de moins de 0.5%). Mais les nina varient un peu plus (environ 3%) : leurs puissances ont tendance à s'écrouler pendant une période très courte (inférieure à 5 minutes). Quant au débit des réseaux, leurs variations semblent complètement aléatoires. Du moins, si l'on devait leur donner une période caractéristique, elle serait de l'ordre de la dizaine de minutes.

Caractère hétérogène de la grille

Les résultats présentés ci-dessus montrent que la grille de calcul a des ressources de calcul très hétérogène : à la fois pour le débit de réseau et pour la vitesse des processeurs. Donc la répartition des tâches doit en tenir compte. Les réseaux sont séparés en 3 grands groupes :

le réseau interne au cluster INRIA-nina à environ 510 Mb/s,
les autres réseaux internes à environ 90 Mb/s, et
les réseaux inter site à environ 5 Mb/s.

Les processeurs sont eux divisés en 2 groupes :

INRIA-nina et IUSTI à plus de 125 Mflops/s, et
INRIA-pf et CEMEF à plus de 44 Mflops/s.

La figure *(5)* résume l'architecture réelle de la grille. Puisque certaines des quantités mesurées ne sont pas tout à fait constantes au cours du temps, ce sont les valeurs moyennes qui sont affichées ici. Les frontales par lesquelles passent les communications inter sites sont représentées par des carrés. Les nœuds reliés entre eux en forme d'étoile représentent les différents clusters. Enfin, les connexions Internet inter sites sont en noir. Une comparaison peut être facilement établie avec le même schéma montrant l'architecture théorique de la grille (figure *1*).

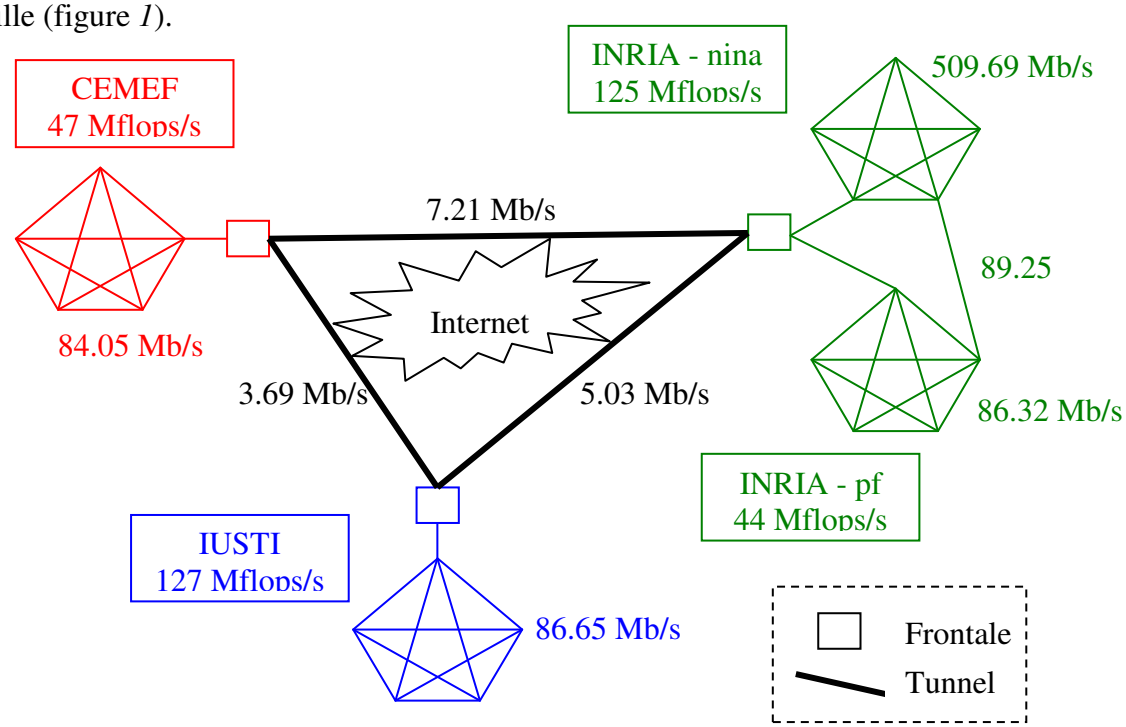

Figure 5 – Architecture réelle de MecaGrid

Globus vs Non-Globus

Il est légitime de se demander si l'ajout de middleware Globus joue un rôle dans les performances des codes de calculs. Afin de mettre en évidence une éventuelle influence, un calcul est exécuté plusieurs fois sur les mêmes ressources, avec, puis sans Globus. Pour cela, il faut non seulement utiliser un seul et même cluster, mais aussi avoir des conditions identiques, c'est-à-dire le même nombre de nœud et de processeurs par nœud. Nous avons choisi d'utiliser le cluster INRIA-nina et 16 nœuds (un processeur par nœud). Le programme doit être compilé deux fois pour créer un exécutable qui s'appuis sur Globus, et un autre qui ne l'utilise pas ; tout en faisant bien attention à appliquer les mêmes options de compilation. Dans la pratique, comme le montre le tableau *(5)*, les temps d'exécution des programmes avec et sans Globus ne montrent pas de différences notables.

1x3	Globus	Non Globus
Temps (s)	455.57	461.07

Tableau 5 – Temps d'exécution avec et sans Globus

Cette application est réalisée avec notre code de calcul et les méthodes discutées dans cette thèse. Pour cela, trois mêmes processeurs du cluster cemef ont été choisis pour les deux exécutions – avec Globus et MPICH-G2, puis sans Globus, mais avec MPI. Sans répéter ce test sur tous les différents clusters, cette hypothèse est supposée vraie pour MecaGrid dans sa globalité : Globus et MPICH-G2 ne détériorent pas les performances de calculs.

Importance des communications par rapport aux calculs

Un sujet sur lequel nous n'avons pas encore d'indications bien précises est l'importance que peuvent avoir les communications par passage de message par rapport au travail de calcul des processeurs. Malheureusement, notre logiciel ne contient pas à l'heure actuelle d'outil pour mesurer le temps passé dans les communications afin de le comparer à celui passé dans les calculs. Cela permettrai d'acquérir une meilleure compréhension sur la nature des calculs, et de mieux comprendre quelle est l'utilisation la plus propice de MecaGrid. Cependant, dans le cadre de sa recherche au sein de l'INRIA, Sophia Steven Wornom trouve des résultats très intéressants avec un autre code de calcul. Ce logiciel, nommé AERO, simule des écoulements monophasiques et biphasiques d'aérodynamiques externe et interne avec des méthodes de type Volumes Finis. Il est développé par l'université du Colorado en collaboration avec l'INRIA [Gui94]. A titre indicatif, voici ses résultats sur le rapport de temps entre les communications et les calculs lors d'un exécution de AERO sur 8 processeurs.

	nina	pf	iusti	cemef
Calculs (s)	331.4	564.7	341.2	759.6
Commnications (s)	10.6	25.7	16.7	47.9
Com/Calcul	0.03	0.05	0.05	0.06

Tableau 6 – Temps de calcul et de communication sur clusters isolés

Le tableau *(6)* montre les temps de calcul et de communication sur chacun des quatre clusters de MecaGrid individuellement. Pour des clusters isolés, les rapports entre les communications et les calculs sont très petits, ce qui montre que le travail des processeurs est largement supérieur aux communications inter processeurs.

	nina-pf	nina-cemef	nina-iust	iusti-cemef
Calculs (s)	401.8	550.8	640.7	970.6
Commnications (s)	25.6	173.1	253.5	315.0
Com/Calcul	0.06	0.31	0.40	0.32

Tableau 7 – Temps de calcul et de communication sur MecaGrid

Après avoir vu le cas des clusters individuels, ce sont différentes paires de clusters qui sont étudiées sur le tableau *(7)*. L'hétérogénéité de MecaGrid fait que les résultats sont très différents de ceux du tableau *(6)* : les rapports communications/calculs sont à la fois plus importants, mais aussi plus diverses entre eux. Cela montre que, dans certains cas, les communications peuvent avoir une très grande importance vis-à-vis des calculs (jusqu'à 40%). Le meilleur rapport correspond à la paire nina-pf, ce qui est tout fait normal, vu qu'ils font partie tous les deux du site de l'INRIA Sophia. Ainsi, leurs communications passent par un réseau local (LAN) et non par le VPN et les frontales de MecaGrid. Le rapport le plus médiocre est pour nina-iusti bien qu'ils ne fassent pas partie des clusters les moins performants, bien au contraire : ce sont eux qui montrent les meilleurs performances individuelles. C'est justement parce que leurs processeurs sont puissants qu'ils passent plus de temps que les autres dans les communications par rapport aux calculs.

5. Optimisations

Actuellement, une des principales déceptions survenues après les premières utilisations de la grille porte sur les performances obtenues par les applications parallèles. Ces résultats décevant proviennent de deux raisons : premièrement, les ressources de calcul étant très hétérogènes, la masse de calcul attribuée à chaque processeur peut être sévèrement déséquilibrée ; et deuxièmement, le débit des réseaux inter sites (qui forment les connections inter clusters) est beaucoup plus lent que celui des réseaux interne à chaque cluster (LAN). L'analyse des performances réelles de MecaGrid montre qu'il y a un facteur 2.87 entre les processeurs les plus puissants et les moins puissants, et un facteur de 138 entre les débits les plus lents et les plus rapides. Cette hétérogénéité extrême pose donc des problèmes de performance, et nous cherchons ici des techniques d'optimisation qui permettent de contre balancer ces inconvénients d'utilisation. De fait, cette problématique est beaucoup plus complexe que dans le cas d'une machine parallèle standard : pour obtenir des résultats satisfaisant sur une grille, il faut non seulement répartir au mieux la masse de calcul, mais aussi diminuer les communications sur réseaux pénalisants au profit des meilleurs réseaux disponibles.

5.1. Instrumentation de PETSc

La librairie PETSc utilisée par la librairie CimLib sur la grille de calcul possède un outil d'instrumentation intéressant sur les performances parallèles. Quelques informations importantes fournies par cet outil -log_summary sont présentées ici. Par la suite, nous l'utiliserons pour évaluer une méthode pouvant améliorer les performances de la grille.

PETSc crée un fichier log qui présente un résumé des performances de temps (en seconde), de vitesse de calcul (en Flops/sec) et d'activité de communication entre processeurs (telle que le

nombre et la taille des messages lancés). Nous pouvons comparer le travail fait par chaque processeur afin, par exemple, de vérifier la répartition de la masse de calcul, ou de quantifié les communications.
Ensuite, des indications sur la mémoire utilisée, et sur la création / destruction des différents objets sont présentées.
Enfin, le fichier log donne le temps moyen pour envoyer un message de taille zéro (appelé temps de latence), ainsi que le temps moyen passé dans MPI_Barrier(). Cette fonction MPI ordonne la synchronisation de tous les processeurs ; c'est-à-dire que plus la charge de calcul est bien répartie, plus ce temps est faible.

Pour conclure, cette instrumentation offerte par PETSc donne plusieurs informations très importantes sur les performances parallèles du code, mais ne donne pas le temps réels passé dans les communications. Donc, le seul moyen que nous avons en ce moment pour mesurer les temps de communications reste l'outil Performance présenté précédemment.

5.2. Analyse des biprocesseurs

Les clusters de l'INRIA et du CEMEF sont constitués de biprocesseurs, c'est-à-dire que l'on a la possibilité de placer un ou deux processus sur chaque nœud lors de l'exécution. Ces deux utilisations sont très différentes, et le but est ici de déterminer laquelle est la plus judicieuse pour exécuter des codes éléments finis sur la grille de calcul.

Débit du réseau entre processeurs du même nœud

Tout d'abord, on remarque que le débit du réseau entre 2 processus d'un même nœud est bien supérieur au débit du réseau entre 2 nœuds distincts :

INRIA - nina :
nina07-nina10 : 509.69 Mb/s
nina10-nina10 : 2014.27 Mb/s

INRIA - pf :
pf12-pf10 : 86.32 Mb/s
pf10-pf10 : 616.65 Mb/s

CEMEF :
node25-node26 : 84.05 Mb/s
node25-node25 : 744.32 Mb/s

La vitesse des communications constitue un point positif à l'utilisation des biprocesseurs.

Vitesse des processeurs d'un même nœud

Dans la pratique, on observe que la vitesse des processeurs est moins importante lorsque les nœuds sont partagés (c'est-à-dire lorsqu'il y a 2 processus par nœud). Cependant, l'outil Performance ne montre pas une telle différence de vitesse, mais c'est lors d'une exécution d'un code éléments finis que l'on peut l'observer. En effet, la résolution d'un système linéaire sur un nœud partagé se fait 223.42 secondes, alors que cette même résolution se fait en 176.34 secondes sur un nœud libre. La vitesse des biprocesseurs serait donc plus

lente lorsqu'ils sont partagés (environ 79% de leur vitesse normale). Ceci pourrait être expliqué par le fait que les 2 processeurs d'un nœud utilisent le même bus : l'utilisation intensive des 2 processeurs peut donc saturer leur bus commun d'accès à la mémoire. Après avoir vu dans a) le côté positif des biprocesseurs, b) nous montre un désavantage important de cette utilisation. Mais alors, quelle est la meilleure solution ? Est-il plus avantageux d'utiliser les 2 processeurs d'un nœud en même temps ou pas ?

Conclusion sur l'utilisation des biprocesseurs

Afin de trancher entre les 2 utilisations possibles des biprocesseurs (1 processeur par nœud, ou 2 processeurs par nœud), nous exécutons un code éléments finis sur la grille des 2 manières différentes. Celle qui aura pris le moins de temps – en conciliant la vitesse des processeurs et la vitesse des communications – sera retenue.
En utilisant 2 processeurs sur chaque site contenant des biprocesseurs (2 nina, 2 pf et 2 CEMEF), nous notons les temps de résolution d'un système linéaire :

Avec 1 processeur par nœud : 664.19 secondes

Avec 2 processeurs par nœud : 715.82 secondes

La solution optimale est donc de ne pas utiliser les 2 processeurs d'un biprocesseur : il est plus avantageux de lancer l'exécution avec un seul processeur par nœud.

5.3. Etudes des préconditionneurs

Une idée pour améliorer les performances sur la grille de calcul consiste à faire varier le degré k du préconditionneur ILU(k) pour le système linéaire avec PETSc. Jusqu'à présent, nous utilisions le préconditionneur ILU(k) avec un degré k = 0. Le nombre permis d'entrées non nulles dans la matrice dépend directement de ce degré k : plus k est élevé, plus le nombre d'entrées non nulles est élevé. Avoir un k > 0 permet d'avoir une masse de calcul supérieure sur chaque processeur, tout en diminuant les communications entre les processeurs et le nombre itérations lors de la résolution du système linéaire. Compte tenus de l'architecture de la grille et du coût élevé des communications, cette méthode parait être adaptée à notre problème. Les résultats suivants ont été obtenus sur la grille de calcul 3x2 (avec 2 processeurs sur chaque site) avec un maillage à 65000 nœuds, puis sur un cluster isolé (avec 6 processeurs) :

Nombre d'itérations	ILU (0)	ILU (1)	ILU (2)	ILU (3)
MecaGrid	100	60	46	38
Cluster	100	61	44	37

Temps de résolution	ILU (0)	ILU (1)	ILU (2)	ILU (3)
MecaGrid	100.00	60.31	65.44	86.95
Cluster	100.00	97.21	126.27	203.53

Tableau 8 – Temps d'exécution relatifs en fonction du préconditionneur

Les figures suivantes montrent les résultats du tableau *(8)* sur des graphes pour bien se rendre compte que le nombre d'itérations nécessaire diminue quand k augmente, et que les temps de résolution évoluent différemment en fonction de k sur une grille de calcul et sur un cluster isolé. Ainsi, il est clair que d'utiliser ILU(1), voire même ILU(2) et ILU(3), fait gagner du temps sur la grille de calcul en rabaissant les communications ; alors que sur un cluster seul,

cela n'apporte pas d'améliorations notables. Nous concluons donc que les communications entres processeurs n'est que très peu pénalisante sur un cluster seul, comparé à MecaGrid.

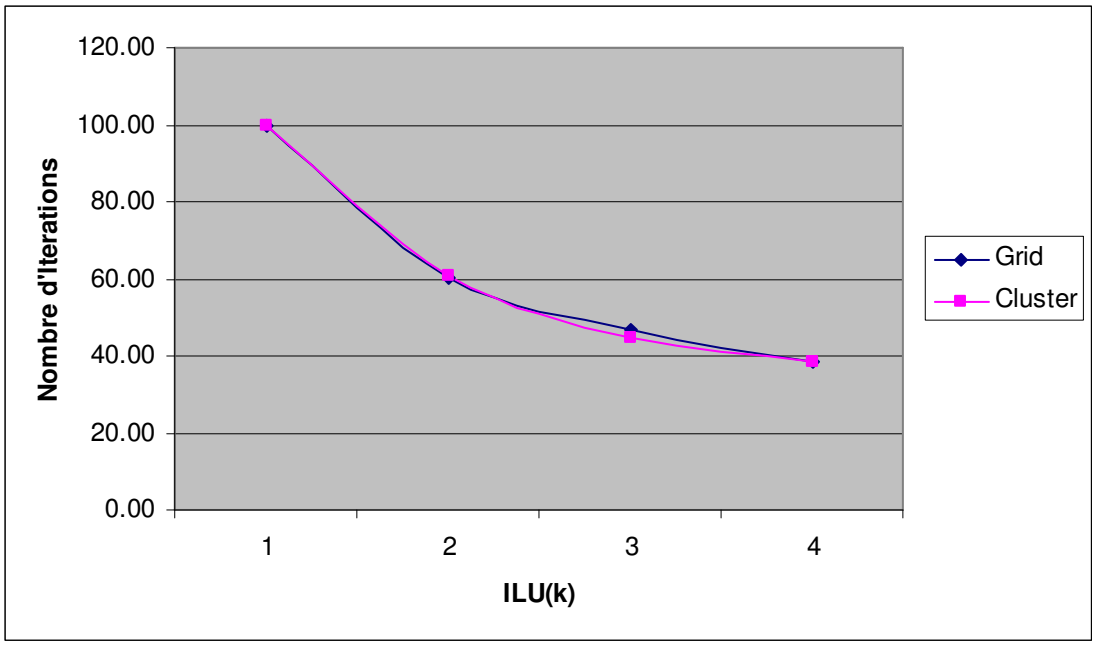

Figure 6 – Nombre d'itérations nécessaires à la convergence en fonction du préconditionneur

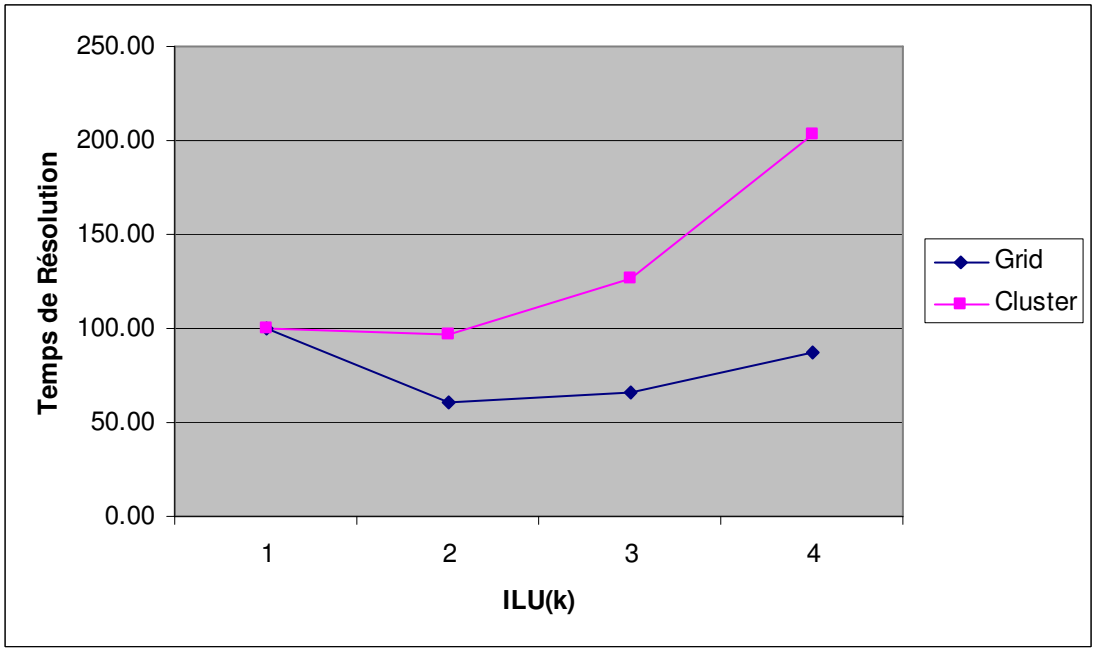

Figure 7 – Temps d'exécution en fonction du préconditionneur

Avec l'instrumentation de PETSc présentée précédemment, nous pouvons mesurer le nombre de messages MPI lancés, la taille maximale de ces messages MPI, le nombre de réductions globales MPI, et la mémoire utilisée par les processeurs :

ILU(k)	k = 0	k = 1	k = 2	k = 3
Nb MPI messages	1.936E+03	1.186E+03	9.000E+02	7.810E+02
Taille MPI messages	5.720E+07	3.940E+07	3.300E+07	2.978E+07
Nb MPI réductions	2.027E+02	1.277E+02	1.000E+02	8.717E+01
Mémoire (octets)	4.310E+07	7.639E+07	1.350E+08	2.119E+08

Tableau 9 – Instrumentation de PETSc en fonction du préconditionneur

Le nombre d'itérations lors de la résolution du système linéaire diminue fortement en fonction de k. Par contre, le temps de résolution est minimum pour k = 1, mais remonte quand k > 1. ILU(1) permet donc d'avoir un temps de résolution d'environ 60% du temps obtenus avec ILU(0), ce qui est loin d'être négligeable. Nous voyons d'ailleurs que, comme nous le souhaitions, les communications entre processeurs diminuent quand k augmente.

Cependant, cette méthode entraîne automatiquement une augmentation considérable de la place mémoire utilisée sur chaque processeur. En effet, entre ILU(0) et ILU(1), elle augmente d'un facteur 1.77. Ceci pourrait poser un problème important si le maillage contient beaucoup de nœuds, car la place mémoire utilisée pourrait alors devoir sortir de la mémoire cache, ce qui ralenti terriblement la vitesse de calcul du programme.

En résumé, entre ILU(0) et ILU(1), nous avons :

un facteur 0.61 sur le nombre de messages MPI
un facteur 0.69 sur la taille maximale des messages MPI
un facteur 0.63 sur le nombre de réductions globales MPI
un facteur 1.77 sur la quantité de mémoire utilisée
et un facteur 0.60 sur le temps de résolution.

5.4. Particularités du processus maître

Lorsque l'on parle d'exécution parallèle avec la librairie MPI de passage de messages, l'idée de processeur maître est très importante. C'est lui qui lit et écrit les données sur le disque dur, qui initialise les communications globales, et qui réalise les tâches qui ne peuvent pas être parallélisées. De ce fait, il doit souvent accéder à une mémoire virtuelle plus importante que les autres. Un exemple simple est celui du partitionneur non parallélisé : le processeur maître (dit le processeur zéro) lit est store la totalité maillage, affiche les informations à l'écran, communique avec les autres processeurs, puis échange des données pour créer une partition par processus. Dans ce cas, il a beaucoup plus de travail à faire, de messages à lancer et recevoir, et de données à stocker que les autres. C'est donc sans surprise que, dans la pratique, le choix du processeur maître montre une influence non négligeable sur les performances. Nous étudions ici les meilleures manières qui optimisent l'utilisation de la grille de calcul.

Puisque le processeur zéro est susceptible d'effectuer une quantité de travail supérieure, le but est de diminuer le temps pendant lequel il est le seul calculer, car pendant ce temps, les autres peuvent attendre qu'il finisse avant de pouvoir continuer. C'est pourquoi, il est judicieux de placer le processeur maître sur un nœud performant où la puissance du processeur est grande par rapport aux autres au sein de la grille. Ensuite, il faut que ce processeur maître ait un accès suffisant à la mémoire virtuelle car c'est lui qui comporte le grand risque de swap (dépassement de la RAM). Pour cela, les nœuds qui contiennent le plus de RAM sont privilégiés ; mais aussi, dans le cas de cluster à biprocesseurs, il est mieux d'avoir le

processeur maître seul sur un nœud pour qu'il n'ait pas à partager la mémoire disponible avec un autre processus.

Sur le point de vue des communications, il y a encore plus de paramètres à considérer. Tout d'abord, les messages pouvant s'effectuer au sein d'un cluster (à travers un LAN, avec un nœud du même cluster) ou par internet (suivant un tunnel du VPN, avec un nœud d'un autre site), il faut non seulement choisir un cluster au réseau interne performant, mais aussi un site aux connexions internet rapide. Ensuite, dans le VPN mis en place, tous les messages sont pris en charge par les machines frontales : elles encapsulent les messages, les cryptent, les compressent, les envoient, les reçoivent, les décompressent, les décryptent. Il est donc important que le processeur maître soit placé sur un cluster dont le site contient une frontale performante (de part son processeur, sa mémoire, sa connexion…). En conséquence, les communications opérées par le nœud maître en grand nombre seront les moins pénalisantes possible.

Dans la pratique, des différences de performance considérables sont observables en fonction de la position du nœud maître. Ils sont montrés ici les temps obtenus lors de la résolution d'un système linéaire sur deux clusters et quatre processeurs par cluster. Les paires de cluster nina-pf, nina-cemef et iusti-cemef sont testés en changeant la localisation du processus zéro. Le temps d'exécution sur pf-nina (avec le nœud maître sur le cluster pf) est 140% celui obtenu sur nina-pf (avec le nœud maître sur le cluster pf). Dans ce cas, c'est la puissance du processeur maître et la vitesse de son LAN qui cause ce résultat puisque aucune frontale n'est utilisée. De la même manière, le temps d'exécution sur cemef-nina est 270% celui obtenu sur nina-cemef. Ici, tous les paramètres discutés précédemment influent, et les résultats sont spectaculaires. Enfin, le temps d'exécution sur cemef-iusti est 115% celui obtenu sur iusti-cemef.

5.5. Méthode de partitionnement optimisé

Enjeux

Les applications en calculs de mécanique des fluides s'appuient généralement sur des données de grande taille correspondant au maillage spatial de la cavité (domaine de calcul). Lors d'une exécution en parallèle, ce sont ces données qui doivent être partitionnées sur tous les processeurs pour répartir la masse de calcul. Pour chaque itération (ou incrément en temps, ou sous-itération), des opérations indépendantes sont effectuées en parallèle, et donc, des communications entre les données distribuées sur toutes les machines doivent s'opérer. La problématique de la répartition équitable de la masse de calcul et de la quantité des données est très importante dans le contexte du calcul parallèle, mais est primordial lorsque l'on parle de grille de calcul. C'est essentiellement le caractère hétérogène des grilles qui augmente l'importance que l'on attache à ce sujet. L'enjeu principal est de minimiser les temps d'attentes que peuvent avoir les processeurs les uns envers les autres, et les temps passés dans les communications inter processeurs tout au long de la simulation. Dans le cas des applications basées sur des maillages, c'est le partitionnement du maillage lui-même qui accomplit la répartition des données et de la masse de calcul.

Le code de calcul CimLib que l'on utilise pour simuler des écoulements de fluides hétérogènes sur MecaGrid est basé sur des méthodes de type éléments finis qui s'appuient sur des maillages de tétraèdres. Le nombre d'inconnues à résoudre dans les systèmes linéaires est

proportionnel au nombre de nœuds inclus dans le maillage, puisqu'elles ont toute une interpolation continue P1. Plus précisément, et pour chaque incrément de temps, 4 inconnues par nœud sont résolues pour la mécanique (v_x, v_y, v_z, p), et 1 inconnue par nœud est résolue pour le transport de la matière (α). Cela montre bien que répartir la masse de calcul (et donc le nombre d'inconnues à résoudre) correspond parfaitement au nombre de nœuds que contient la partition (partie du maillage) envoyée sur chaque processeur. Chaque processeur calcule l'écoulement dans sa portion du maillage, tandis que les communications sont effectuées lors du produit matrice - vecteur est dépendent du nombre de nœuds à l'interface des partitions.

Le fait de vouloir un partitionnement très optimisé est d'autant plus important que les méthodes numériques choisies dans CimLib ont une approche Eulérienne : elles ne nécessitent pas de remaillage pendant la simulation. De ce fait, c'est le même maillage qui est utilisé tout au long du calcul. De plus, l'étude de performance réelle de la grille montre que celle-ci évolue d'une façon non négligeable en fonction du temps. Une question très importante se pose donc ici : faut-il repartitionner le maillage pendant la simulation pour tenir compte des évolutions des performances ? En effet, si une masse de calcul bien précise a été attribuée à un processeur x en tenant compte de sa puissance calculée avant l'exécution du programme, mais que celle-ci se met à changer de façon importante, il est possible qu'il faille modifier la répartition des tâches initialement prévue. Et il en est de même pour le débit des réseaux. De cette façon, un partitionnement très bien optimisé au début de la simulation peut éventuellement se transformer en un partitionnement de mauvaise qualité qui ralentit les calculs, dès lors que les performances de la grille évoluent d'une trop grande manière. Afin de donner une réponse satisfaisante à cette question primordiale, il faut regarder de près la façon dont varient la puissance de calcul des processeurs et le débit des réseaux de MecaGrid. Les résultats observés précédemment à ce sujet montrent que la vitesse des processeurs ne varient pas d'une manière très importante : il y a un écart type d'environ 1% autour d'une valeur moyenne. Par contre, les variations aléatoires qui se produisent dans le débit des réseaux (à la fois internes et externes aux clusters) peuvent avoir jusqu'à un écart type de 30% autour d'une valeur moyenne, ce qui n'est absolument pas négligeable. Seulement, il semble impossible de pouvoir attribuer une période d'oscillation, tellement les évolutions sembles aléatoires. Sinon, elle serait sûrement de l'ordre de quelques dizaines de minutes, ou de une heure. Maintenant, il faut comparer cette durée avec celle que prend un repartitionnement. Pour en avoir une idée, cette opération prend environ 10 minutes sur la grille avec 2 processeurs sur chaque site pour un maillage à 65000 nœuds ; et cela peut être bien plus long lorsque le maillage est de grande taille, et qu'un grand nombre de processeurs est utilisé. Ainsi, la durée du repartitionnement n'est pas être significativement inférieure aux périodes de variation de la puissance de calcul. Autrement dit, si on avait recours au repartitionnement, on serait amené à répéter cette opération très souvent, et le gain de temps sur les calculs serait bien inférieur au temps passé dans les recherches d'optimisation. En conclusion, il n'est pas nécessaire de repartitionner le maillage pendant la simulation. De plus, l'ensemble des quantités mesurées reste sensiblement dans la même zone autour de valeurs moyennes, qui elles, restent inchangées au cours du temps. Ces moyennes peuvent donc servir de référence à l'exécution du partitionnement, avant le début des calculs.

Méthode de partitionnement

La méthode de partitionnement de maillage présentée ici constitue une amélioration à celle établie lors du projet DRAMA [Bas00]. Plus précisément, elle apporte une généralisation au contexte hétérogène que l'on a sur une grille de calcul. L'algorithme d'optimisation des partitions est brièvement présenté ; puis, plusieurs techniques d'utilisations

sont passées en revue, incluant des perspectives pouvant améliorer le lien entre le partitionneur et les solveurs numériques.

Pour partitionner un maillage destiné à une exécution parallèle sur une machine homogène, il suffit de diviser tout simplement le maillage de façon à ce que chaque processeur contienne le même nombre de nœud. Il faut aussi faire en sorte que les communications inter processeurs – qui correspondent directement à l'interface entre les partitions du maillage – soient minimisées. Mais, dans le contexte des grilles de calcul, le partitionnement représente une problématique bien plus complexe. En effet, les grandes différences qu'il existe dans les vitesses des processeurs, les débits des réseaux et les quantités de mémoire disponibles, engendrent une importante hétérogénéité des ressources de calcul difficile à négocier. Pour obtenir des résultats satisfaisant sur une grille, il faut donner plus de travail à un processeur rapide qu'à un autre plus lent, mais aussi diminuer les communications sur réseaux pénalisants au profit des meilleurs réseaux disponibles.

Il faut faire attention a ce que le partitionnement ne crée pas de partitions trop grosses pour la mémoire des processeurs qui les utilise. En effet, un dépassement de la mémoire entraîne un swap de la RAM, ce qui ralenti extrêmement les calculs avec des accès au disque trop fréquents. C'est une raison supplémentaire pour n'utiliser qu'un seul proc par nœud dans les clusters à biprocesseur et ainsi disposer d'une quantité de RAM supérieure. Par exemple, nous considérons que dans la pratique, la limite supérieure que peut avoir une partition de maillage serait autour de 50000 nœuds pour 256 Mo de RAM. Cette estimation est extrêmement dépendante des types de solveur appelés pendant la simulation. Dans notre cas, seuls des solveurs de type P1 (5 inconnues par nœuds du maillage au total et par incrément de temps) sont utilisés. Avec d'autres solveurs de type P0 (où les inconnues sont sur les éléments), il y a bien plus d'inconnues à résoudre, et donc, des partitions de maillage plus petites doivent être produites.

Notations et définitions

Soit $G = G(V, E)$ un graphe non orienté avec des sommets *V* et des arrêtes *E*. Les arrêtes représentent des dépendance dans les données du maillage. Dans ce modèle, on considère que le graphe représente le maillage : les sommets représentent les nœuds et les éléments du maillage.
Ainsi, partitionner le maillage revient à partitionner son graphe correspondant. Dans la littérature, [Shi95] traite du cas sur des machines parallèles à architecture homogène. Le problème du partitionnement homogène sur *n* processeurs est divisé en deux parties : définition des *n* sous graphes, et association de chaque sous graphe avec un processeur.
Sur grille de calcul, l'architecture est hétérogène pour ce qui est du réseau et des processeurs. En conséquence, il nous faut considérer ces paramètres pour définir les sous graphes, et la deuxième étapes doit être fusionnée avec la première pour associer correctement les sous graphes avec leur processeur.
Pour évaluer la qualité d'une partition π, il est nécessaire de définir deux modèles qui servent à estimer un temps caractéristique induit par une simulation parallèle. Pour tous les *n* processeurs, le temps de calcul est défini par :

$$t_p \equiv \frac{d_p}{s_p} \qquad (2)$$

où d_p est la masse de calcul du processeur *p* (exemple : nombre de nœuds du maillage associés au processeur *p*, car les inconnues à résoudre sont situées aux nœuds) et s_p est la vitesse mesurée du processeur *p*. Le temps passé dans les communications est définie par :

$$c_p \equiv \sum_{i \neq p} \frac{d_{pi}}{v_{pi}} \qquad (3)$$

où d_{pi} est le volume des communications entre les processeurs *p* et *i* (exemple : nombre de faces dans le maillage coupées par la partition) et v_{pi} est le débit mesuré du réseau reliant *p* et *i*.

Avec ces deux modèles, une fonction coût de la partition est définie par :

$$\Phi = \beta T + (1-\beta) C \qquad (4)$$

où $T = (t_0, t_1, ..., t_{n-1})^T$ et $C = (c_0, c_1, ..., c_{n-1})^T$. Quant à β, c'est un paramètre appartenant à $[0,1]$ qui sert de poids entre *T* et *C*. Autrement dit, il détermine l'importance que l'on attache à la puissance de calcul des processeurs vis-à-vis du débit des communications inter processeurs. Aucune technique n'est aujourd'hui disponible pour choisir au mieux la valeur de β, mais c'est seulement dans la pratique, avec des résultats expérimentaux, qu'on y parvient. De cette manière, toute partition peut être évaluée en calcul cette fonction coût. Maintenant, le problème du partitionnement revient trouver la partition associée au graphe qui minimise la fonction coût Φ.

Méthode locale du repartitionnement parallèle

Dans cette partie, la stratégie de minimisation de la fonction coût est brièvement décrite. La méthode est itérative, locale et parallèle.
On introduit un nouveau graphe qui lui représente l'architecture de la machine de calcul appelé le graphe d'architecture. Ses sommets représentent les processeurs, et les arrêtes représentent les connexions réseaux entre les nœuds.
Dans le cadre de cette méthode d'optimisation de partition, il faut obligatoirement commencer par se munir d'une partition valide avec une qualité quelconque. C'est pourquoi, avant l'exécution de cette méthode, on crée une partition homogène sans tenir compte de l'architecture de la machine. En considérant donc une partition initiale π_0, la première étape consiste à déterminer la meilleures paires de processeurs qui va optimiser la partition. Pour cela, nous développons un critère, fonction d'une partition π_i et des caractéristiques architecturales (s_p et v_{pi}), qui permet de comparer toutes les paires possibles. Une fois que toutes les paires séparées sont formées, la partition, limitée à chaque paire, est optimisée localement : les sommets (nœuds et éléments) sont transférés d'un processeur à un autre, tant que la partition limitée est meilleure. Un autre critère a été développé pour déterminer une priorité de transfert entre les sommets. Ainsi, les deux dernières étapes sont répétées tant que l'on a la possibilité d'améliorée de façon globale la partition.

5.6. Utilisations du partitionneur

A l'heure actuelle, il n'y a d'autre solution que de déterminer à l'avance – avant de commencer une simulation – les ressources sur lesquelles le calcul va être lancé. Cela correspond à choisir les clusters, puis le nombre de processeurs par cluster. Une fois ce choix fait, le partitionneur doit répartir le maillage en tenant compte des caractéristiques moyennes de ces ressources. Enfin, le calcul peut être exécuté en appliquant bien les correspondances entre les partitions et les processeurs. Cette technique « manuelle » a plusieurs inconvénients : elle implique au moins deux requêtes (une pour partitionner, une pour lancer la simulation), elle oblige à connaître les caractéristiques moyennes de toutes les ressources, à devoir choisir le nombre de processeurs par cluster au risque de ne plus en avoir l'accès au moment de l'exécution… Quelques perspectives peuvent donc être dressées à ce sujet.

La deuxième technique envisageable est de coupler l'outil de mesure de performance, le partitionneur, et le code de calcul (CimLib). Cela permettrai d'exécuter une seule requête, et sur des ressources quelconques. Après avoir fait un test de performance, le partitionneur pourrait répartir le maillage en considérant les performances des nœuds et des réseaux utilisés. Enfin, la simulation serait prête pour commencer directement après et sur les mêmes nœuds, sans qu'une nouvelle requête ne soit nécessaire. Cette méthode « automatique » ne demande pas une connaissance au préalable des performances des ressources, mais elle a le désavantage de présenter le choix entre mesurer rapidement des performances approximatives, ou rallonger la durée de ces mesures pour pouvoir connaître des performances plus précises (plus proches des valeurs moyennes réelles). Ce dernier point implique une quantité de calcul importante et superflue, dans le sens où elle ne participe pas à la simulation numérique. De plus, un outil de performance comme celui là peut ne pas être parfaitement adapté aux méthodes élément fini, puisque la nature des calculs est différente, et cela n'est pas optimal. C'est cette réflexion qui introduit une troisième technique qui serait idéale.

Cette dernière méthode, dite idéale, serait d'instrumenter le code de calcul pour que des données de performance puissent être enregistrées pendant les calculs d'éléments finis de la simulation. Ainsi, l'utilisation de l'outil de performance serait évitée au profit d'un outil bien plus adaptée (sur la nature des calculs), et puis surtout, les opérations effectuées pour les mesures de performance ne serait pas inutiles pour la simulation numérique. Pour cela, les noeuds à utiliser sont quelconque à priori, et la première étape représente la création d'une partition homogène, vu qu'à ce moment là, les performances réelles des ressources utilisées ne sont pas censées être connues. Ensuite, le calcul peut commencer tout en mesurant la puissance des processeurs et le débit des réseaux. Pour une simulation non stationnaire sur plusieurs centaines ou milliers d'incréments de temps, quelques incréments suffisent, pour que la durée des mesures soit assez importante pour avoir une moyenne convenable, mais qui reste négligeable par rapport à la durée totale de l'exécution. Ces calculs préliminaires sont peut être plus lents qu'avec une partition optimisée, mais ils servent en même temps à récolter les données utiles au partitionneur pour réaliser une nouvelle partition optimisée. Une fois cette deuxième étape accomplie, la simulation peut continuer, mais cette fois ci, avec une répartition des calculs et des données hétérogène pour s'adapter aux ressources. Cette méthode « automatique » demande quelques développements au sein du code de calcul (ici CimLib) pour enregistrer des mesures de performance pendant l'assemblage des matrices et la résolution des systèmes linéaires, mais représente sûrement une (évolution) qui améliore l'utilisation de la grille de calcul.

5.7. Conclusions

Les performances de MecaGrid varient considérablement au cours du temps ; mais l'idée de repartitionner pendant l'exécution du programme parait difficile compte tenu, à la fois, du temps nécessaire à repartitionner, et de la rapidité à laquelle changent les performances. Le repartitionnement aurait seulement été utile dans le cas où les variations des performances avaient une période significative bien plus importante (de l'ordre de la dizaine d'heure).
Dans le paragraphe suivant, diverses applications sont étudiées en utilisant les techniques d'optimisations présentées ici. Il est notamment montré l'influence que peuvent avoir les partitions sur la résolution des systèmes linéaires. Plus précisément, ce sont les méthodes de préconditionnement qui semblent dépendre de la façon dont le partitionnement a été effectué.

6. Exemples d'applications sur la grille

6.1. CimLib

Après avoir analysé les caractéristiques de l'architecture théorique puis réelle de la grille de calcul, le but est de l'utiliser pour effectuer des calculs à grande échelle qui serviront à la fois à tester nos techniques d'amélioration de performance sur cet outil, et à montrer des applications dans le cadre de la mécanique des fluides hétérogènes. C'est avec notre code de calcul implémenté en C++ que les simulations numériques sont exécutées. La librairie CimLib développée au CEMEF contient un ensemble de solveurs numériques qui s'appuient sur des méthodes de type éléments finis. Elle a été parallélisée en utilisant la librairie *MPI* (*Message Passing Library*) largement reconnue lorsque l'on parle de programmation parallèle par envois de messages. Les matrices locales qui sont calculées dans CimLib sont ensuite prisent en charge par PETSc. Cette librairie (*Portable and Extensible Toolkit for Scientific Computation*) permet de résoudre en parallèle et de façon itérative des systèmes d'équations linéaires et non linéaires impliquant des matrices creuses. Les matrices locales que PETSc reçoit, sont assemblées en une matrice globale. Plusieurs méthodes de préconditionnement sont à disposition pour permettre une meilleure résolution : selon la nature du système à résoudre, des préconditionneurs de type ILU(k) ou Jacobi par blocs sont souvent choisis. Enfin, le système linéaire global est résolu en utilisant une méthode du résidu conjugué, soit MINRES, soit GMRES, selon les propriétés du système.

Les méthodes d'éléments finis qu'utilisent les solveurs de CimLib s'appuient sur un maillage : la cavité qui correspond au domaine de calcul est maillée avec des tétraèdres (ou des triangles en 2D). Pour les calculs parallèles, ce maillage est partitionné ; puis, chaque processeur calcule l'écoulement dans sa portion du maillage. Les communications inter processeurs effectuées lors du produit matrice - vecteur dépendent du nombre de nœuds à l'interface des partitions.

Conformément aux méthodes choisies et développées tout au long de cette thèse, les inconnues de tous nos systèmes à résoudre sont exclusivement situées sur les nœuds du maillage. Ainsi, dans le problème de Navier Stokes (solveur P1+/P1 pour les écoulements), il

y a 4 inconnues par nœud en 3D (3 pour la vitesse, et 1 pour la pression) et 3 inconnues par noeud en 2D (2 pour la vitesse, et 1 pour la pression). Le transport, lui (solveur pour l'évolution des phases), résulte en un système à 1 inconnue par nœud. En effet, la fonction de phase transportée est une fonction Level Set qui est à la fois scalaire et continue (P1). Ces deux systèmes sont résolus l'un après l'autre, et ce, à chaque incrément de temps, tout au long de la simulation. Par conséquent, il y 5 inconnues à trouver par incrément de temps. Le nombre d'incréments dépend de la durée totale de la simulation, et du pas de temps utilisé.

6.2. Cas d'extrusion à 65000 nœuds

Ce premier exemple est un cas d'extrusion en 3D avec un maillage à environ 65000 nœuds. Il correspond à la simulation stationnaire de l'écoulement d'un fluide newtonien incompressible en négligeant les termes d'inertie. C'est donc avec les équations de Stokes qu'une telle simulation est calculée. Le maillage de la cavité 3D qui correspond au domine de calcul est représenté dans la figure *(8)*. La partie cylindrique de la pièce fait 80 cm de diamètre. Les différentes couleurs indiquent le processeur qui contient chaque élément : il y a ici six partitions (et donc six couleurs) pour un calcul sur six processeurs.

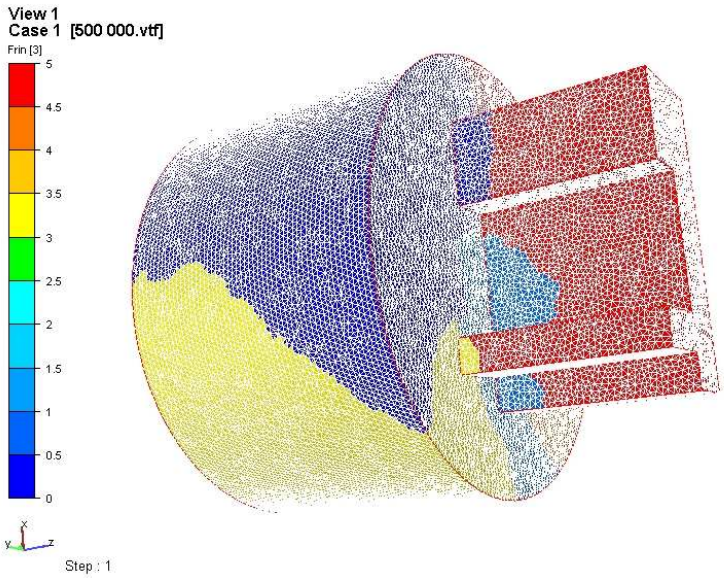

Figure 8 – Maillage et partitions de la cavité

Les conditions aux limites sont les suivantes pour faire passer le fluide dans la pièce de la gauche vers la droite. Une pression est imposée (condition limite de type Neumann) à 100000 Pa sur le plan d'entrée dans la pièce ($z = 0$, à gauche sur la figure), tant dis que sur le plan de sortie ($z = 85\,\text{cm}$, à droite sur la figure), et sur tous les autres nœuds surfaciques, la pression est laissée libre. Un contact collant (condition limite de type Dirichlet avec $v = 0$) est appliqué sur toute la surface de la cavité, excepté sur les plans d'entrée et de sortie où seule la composante en *x* est laissée libre ($v_x = v_y = 0$). Cela force le fluide à rentrée dans la pièce par la gauche avec une direction dans le sens de l'axe *z*, et d'en sortir par la droite, toujours dans le direction des *z*. Les figures *(9)* et *(10)* montrent les résultats de cette simulation stationnaire

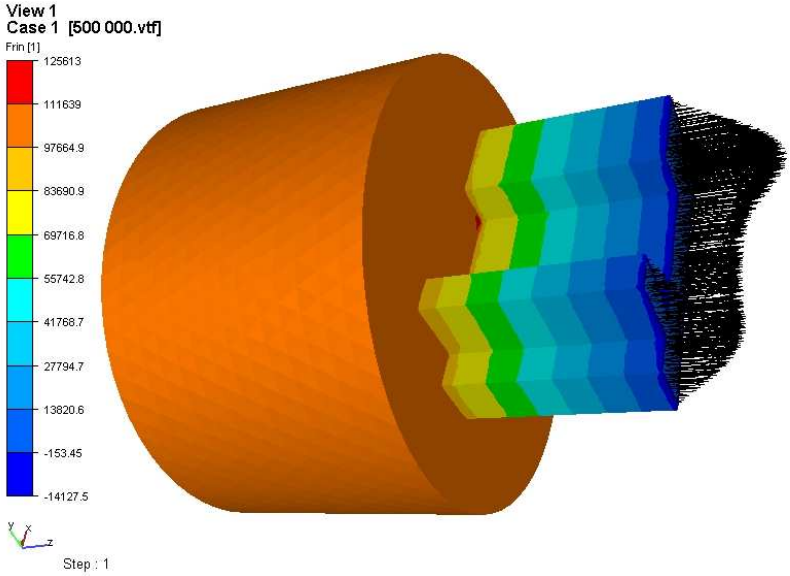

Figure 9 – Profil de pression et champ de vitesse

La pression dans le fluide newtonien contenu dans la cavité est visualisée avec les franges de couleurs. On observe un gradient de pression régulier dans la partie étroite, et elle s'annule quasiment sur le plan de sortie. Les vecteurs de couleur noirs représentent la vitesse du fluide lorsque celui-ci sort de la pièce sous l'effet de la pression imposée en entrée. Comme prévu par les conditions limites prédéfinies, cette vitesse est en direction de l'axe *z*, mais son intensité varie en fonction des coordonnées *x* et *y*. La figure *(10)* en montre la norme.

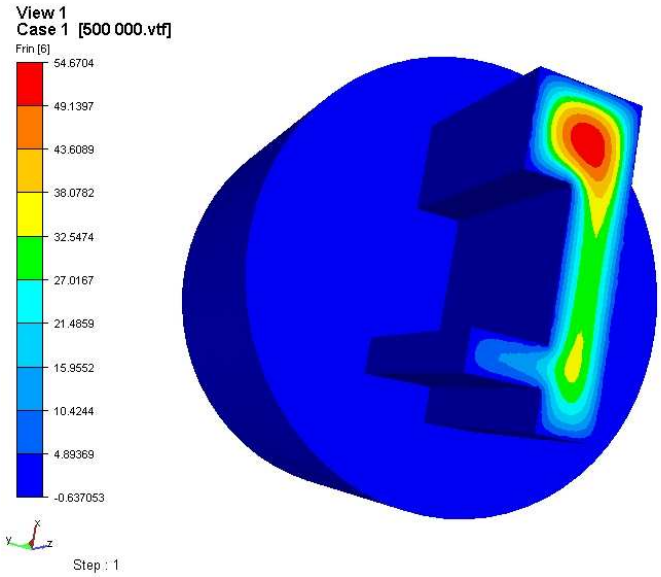

Figure 10 – Norme de la vitesse

La norme de la vitesse est nulle sur toute la surface de la pièce, sauf sur les plans d'entrée et de sortie où la plus forte valeur est de 55 cm/s (en rouge sur la figure *10*).

Voici maintenant les temps de résolution sur chaque site de MecaGrid séparément avec un (1x1) puis trois (1x3) processeurs. Les temps affichés ici englobent uniquement l'assemblage des matrices et la résolution d'un système linéaire et symétrique résultant des équations de Stokes. Après avoir appliqué un préconditionnement par ILU(0), PETSc résous le système linéaire par une méthode de résolution itérative de type résidu conjugué MINRES. Le nombre

d'itérations nécessaires pour converger vers la solution avec une précision à 10^{-7} est indiqué pour que l'on puisse comparer la performance des différents clusters indépendamment du nombre d'itérations exécuté (avec le temps passé par itération).

1x1	INRIA-nina	CEMEF	IUSTI
Temps (s)	210.91	1038.46	207.01
Nb itérations	521	521	521
Temps/iter (s)	0.40	1.99	0.40

1x3	INRIA-nina	CEMEF	IUSTI
Temps (s)	87.56	455.57	173.66
Nb itérations	653	653	650
Temps/iter (s)	0.13	0.70	0.27

Tableau 10 – Temps d'exécution et nombre d'itérations sur un cluster

Ces résultats sont en accord avec l'architecture des clusters étudiés : le cluster de l'INRIA-nina a les meilleures performances, et le cluster du CEMEF a les moins bonnes. La différence qu'il existe dans les temps d'exécution montre bien la diversité des ressources de calcul et de l'hétérogénéité globale de MecaGrid. Ainsi, le même calcul est 5 fois plus lent en série (sur un processeur) au CEMEF que sur INRIA-nina, et 5.4 fois plus lent en parallèle (sur trois processeurs). En série, INRIA-nina et IUSTI ont des performances similaires, mais en parallèle (sur trois processeurs), INRIA-nina se montre plus rapide (2 fois plus rapide). C'est sans doute la qualité du réseau interne (LAN) qui crée cette différence : il est de 1000 Mb/s sur INRIA-nina, et de 100 Mb/s sur IUSTI, soit 10 fois plus performant. En effet, en série, aucune communication n'est effectuée, donc le débit du LAN n'influe absolument pas sur les temps de calcul ; mais en parallèle, le réseau est utilisé pour échanger des messages entre les nœuds, et donc, les performances dépendent débit du LAN. C'est aussi sûrement ce qui explique qu'il y ai encore plus de différence entre le CEMEF et INRIA-nina en parallèle qu'en série. Enfin, les quelques itérations que l'IUSTI fait en moins viennent du fait que les versions des librairies et compilateurs ne sont pas tout à fait les mêmes sur tous les clusters de la grille. Car, étant dans un cas homogène, les partitions et les calculs sont censés être les mêmes sur tous les clusters.

Dans un deuxième temps, nous notons les temps de résolution sur la grille avec 1 processeur sur chaque site (3x1), 2 processeurs sur chaque site (3x2), puis 3 processeurs sur chaque site (3x3). Pour chaque cas, deux sortes de partitions ont été utilisées ici : une partition homogène, puis une partition optimisée en tenant compte des caractéristiques de la grille mesurées par notre outil de performance.

3x1	Homogène	Optimisée
Temps (s)	649.52	493.89
Nb itérations	651	634
Temps/iter (s)	1.00	0.78

3x2	Homogène	Optimisée
Temps (s)	458.86	436.05
Nb itérations	643	765
Temps/iter (s)	0.71	0.57

3x3	Homogène	Optimisée
Temps (s)	495.37	498.93
Nb itérations	657	780
Temps/iter (s)	0.75	0.64

Tableau 11 – Temps d'exécution et nombre d'itérations sur trois clusters

Sur trois processeurs, les temps d'exécution sont plus importants sur la grille (3x1) que sur chaque site séparément (1x3). Plusieurs raisons différentes viennent s'ajouter pour arriver à ce résultat décevant à première vue. Premièrement, tous les clusters étant concernés, les moins performants ralentissent les plus puissants. Il est donc normal qu'une résolution sur trois processeurs sur la grille soit inférieure à celle exécutée sur INRIA-nina. Cependant, les résultats ne sont pas catastrophiques car les temps du cas 3x1 sont bien inférieures au cas 1x1 sur le CEMEF, et du même ordre que le cas 1x3 sur le CEMEF. Deuxièmement, les communications très lentes entre les sites importent une pénalisation supplémentaire qui n'existe pas au sein d'un cluster isolé. Et troisièmement, l'hétérogène des ressources de calculs entraîne une difficulté dans la répartition des tâches. Une amélioration importante de cette problématique est apportée par l'optimisation des partitions : le gain de temps est dans ce cas d'environ 20% par itération, quelque soit le nombre de processeurs.
Nous remarquons que dans certains cas (ici pour 3x2 et 3x3), le nombre d'itérations nécessaires à la convergence du système linéaire varie de façon importante entre les partitions homogènes et optimisées. Cela pourrait paraître surprenant, mais en fait, c'est une mise en évidence de l'influence que peut avoir le partitionnement sur la résolution. Plus particulièrement, ce sont les méthodes de préconditionnement qui dépendent des partitions du maillage. Ce résultat est important car cela oblige à faire attention à ce que les partitions créées pour répartir la masse de calcul ne gênent pas la méthode de résolution pour converger vers la solution.

Enfin, la dernière observation que l'on peut tirer du tableau *(11)* est la baisse d'efficacité parallèle lorsque le nombre de processeurs augmente trop. La résolution d'un système linéaire sur six processeurs de la grille (3x2) est meilleur que sur trois (3x1) ; mais sur neuf processeurs (3x3), les temps de résolution sont supérieurs au cas 3x2. Si le cas 3x1 est considéré comme le temps de référence, l'efficacité parallèle est de 0.71 sur six processeurs, et de 0.44 sur neuf processeurs. Pour un « petit » maillage à 65000 nœuds comme celui utilisé ici, exécuter le programme sur neuf processeurs de MecaGrid est moins intéressant que sur six. Cela vient du fait que la masse de calcul répartie sur chaque processus devient trop petite, et le temps passé dans les communications prend le dessus pour ralentir les calculs. Plus le nombre de processeur est grand, moins le nombre d'inconnues à résoudre par chaque processeur est petit, et plus il est nécessaire de communiquer entre les nœuds et les clusters. Ce phénomène est déjà observé sur des clusters isolés, mais est largement amplifié lorsqu'il s'agit de machine parallèle hétérogène et de grille de calcul.

Avec l'instrumentation de PETSc, il est possible de connaître la puissance en flops/s de la machine parallèle utilisée pour l'exécution. Cela correspond à la définition de la puissance *(1)*. Dans le cas 3x2 du tableau *(11)*, la puissance de MecaGrid sur 6 processeurs est de 89 Mflops/s pour une partition homogène, et de 93 Mflops/s pour une partition optimisée.

6.4. Ecroulement du barrage

L'application de l'écroulement d'un barrage en 3D est un cas souvent utilisé pour tester des codes de simulation pour écoulement de fluides et de surfaces libres. C'est en fait une colonne d'eau qui s'effondre sous son propre poids dans une cavité en forme de parallélépipède rectangle. Il s'agit donc d'une simulation d'écoulement de fluide newtonien hétérogène et instationnaire. Notre code de calcul CimLib et les méthodes numériques développées dans cette thèse sont utilisés ici. Brièvement, les équations de Navier Stokes qui tiennent compte des termes d'inertie sont généralement utilisées dans ce cas. L'évolution des deux phases en présence (l'eau et l'air) est assurée par le transport d'une fonction Level Set qui permet de capturer les interfaces.

Les résultats du paragraphe qui traite de l'optimisation de l'utilisation de MecaGrid montrent que la méthode de préconditionnement optimale pour résoudre les équations de Navier Stokes en 3 dimensions est ILU par bloc de 1. C'est donc celui-là qui est utilisé ici.

La cavité de dimensions 1.5 x 1.0 x 2.0 mètres est maillée avec 445 000 nœuds et 2 560 000 éléments. La taille de maille correspondante est de 2 centimètres. Le temps total de la simulation est 3 secondes, et le pas de temps utilisé est 0.005 seconde. 600 incréments en temps sont donc nécessaires. Sur la figure *(11)*, l'interface entre le fluide et l'air est montrée aux moments $t = 0s$, $t = 0.6s$, $t = 1.2s$, $t = 1.8s$, $t = 2.4s$, et $t = 3s$. Les paramètres utilisés sont les suivants : le fluide a une viscosité de 10 Pa.s (à peu près celle d'un miel liquide) et une masse volumique de 1000 kg/m^3, l'air a une viscosité de 1 Pa.s et une masse volumique de 1 kg/m^3. Les conditions aux limites sont de type glissant, c'est-à-dire que, sur toutes les surfaces de la géométrie, seule la composante tangentielle de la vitesse est imposée à zéro.

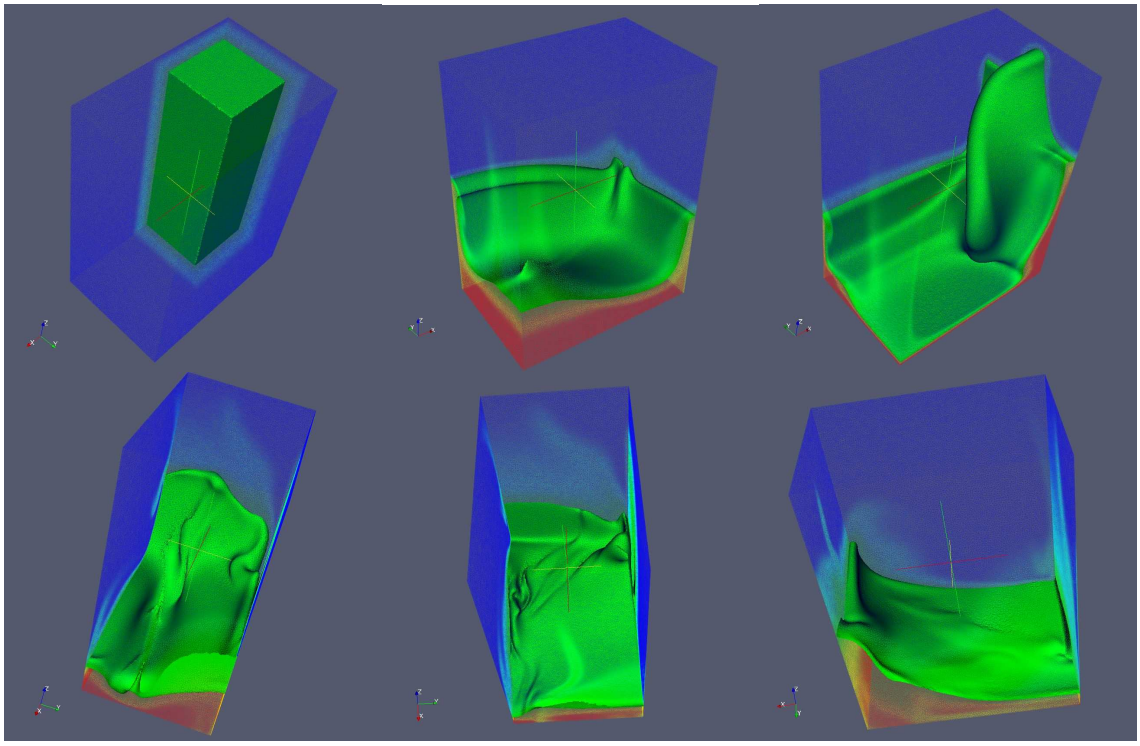

Figure 11 – Surface libre lors de l'écroulement d'un barrage

Un nombre de 12 processeurs est choisi pour exécuter cette application. Dans un premier temps, seul le cluster nina de l'INRIA est utilisé (1x12). Dans un deuxième temps, c'est avec 3 clusters différents de MecaGrid (nina, pf et iusti) que le cas est exécuté (3x4), avec un partitionnement homogène, puis, avec un partitionnement optimisé. Pour indication, chaque partition de maillage homogène contient environ le même nombre de nœuds, soit 35100. Par contre, ce n'est pas de tout le cas pour la partition optimisée : le nombre moyenne de nœuds par processeur dans les clusters nina, pf et iusti sont respectivement de 45500, 18150, et 52600. Ces valeurs sont en accord avec la puissance des processeurs mesurée (respectivement 125, 44, et 127 Mégaflops par seconde).

Le tableau *(12)* montre les temps d'exécution de l'assemblage des matrices locales pour le solveur Navier Stokes (NS) et du transport des interfaces (Alpha), ainsi que les temps de résolution des systèmes.

Lieu	nina	3x4	3x4
Nb processeurs	12	12	12
Partition	Homogène	Homogène	Optimisée
NS assemblage	12 648	24 156	19 800
NS résolution	80 028	108 907	89 268
Alpha assemblage	11 472	28 841	23 640
Alpha résolution	564	996	816
Total (s)	104 712	162 899	133 524

Tableau 12 – Temps d'exécution sur un cluster et sur la grille

Premièrement, la partie qui prend le plus temps pendant la simulation est la résolution du solveur de Navier Stokes pour les écoulements. Le système non linéaire est résolu avec plusieurs itérations de Newton qui elles, sont des résolutions de systèmes linéaires. En moyenne, 4 à 5 itérations de Newton sont nécessaires à la convergence du système non

linéaire. Ce solveur est donc le plus complexe, et consomme plus de temps que le solveur pour le transport des interfaces.

Deuxièmement, l'exécution sur le cluster nina isolé est la plus rapide (environ 29 heures), tandis que sur MecaGrid, la durée est bien plus longue (45 heures, soit 155% du temps d'exécution sur nina). Les raisons principales sont les suivantes : le cluster pf ralenti les calculs de part ses performances propres, et des communications inter sites très pénalisantes sont introduites.

Troisièmement, un partitionnement optimisé permet d'améliorer ce résultat : un gain de 18% est apporté au temps d'exécution total. Cela traduit une meilleure répartition de la masse de calcul : une quantité le travail supérieure est donnée aux processeurs les plus puissants (ici sur iusti) pour que ceux-ci terminent leur calcul approximativement en même temps. Cependant, la géométrie de la cavité (de forme parallélépipède rectangle) ne permet pas de diminuer les communications pénalisantes de manière significative. Il est en effet très difficile de diviser un objet cubique en plusieurs parties dont les interfaces soient petites. Tout de même, la contribution d'une partition optimisée est loin d'être négligeable sur des simulations à grande échelle.

Exactement la même application est présentée ici, avec, cette fois-ci, un maillage plus fin à 1 500 000 nœuds et 8 625 000 éléments. 11 processeurs sont utilisés sur les 3 mêmes clusters (nina, pf et iusti) pour avoir un total de 33.

Lieu	3x11	3x11
Nb processeurs	33	33
Partition	Homogène	Optimisée
NS assemblage	60 847	48 792
NS résolution	351 407	281 784
Alpha assemblage	36 316	29 121
Alpha résolution	2 235	1 792
Total (s)	450 805	361 489

Tableau 13 – Temps d'exécution sur la grille avec des partitions homogène et hétérogène

Les durées totales d'exécution sont d'environ 5 jours et 6 heures avec des partitions optimisées, et de 4 jours et 5 heures avec des partitions optimisées. Le gain apporté par cette technique est donc de 20% : valeur similaire à celle obtenue avec un maillage 3 fois moins fin. On peut ainsi conclure que ce résultat ne dépend pas de la taille du problème, mais plutôt de la forme de la géométrie. Afin de vérifier cela, une autre application susceptible d'engendrer de meilleurs résultats est étudiée dans le paragraphe suivant.

6.5. Déverse d'un jerricane

Cette application montre l'écoulement d'un fluide par le goulot d'un jerricane. Au début de la simulation, le fluide se trouve totalement dans le jerricane qui se trouve renversé : son goulot ouvert est dirigé vers le bas. Le fluide commence alors à se déverser par l'ouverture, tandis que l'air s'engouffre à son tour dans le jerricane sous forme de bulles. Ainsi, la cavité se vide peu à peu de son contenu initial et se rempli d'air, jusqu'à ce que le fluide soit totalement déversé. Le même code de calcul et les mêmes méthodes numériques que dans l'application précédentes de l'écroulement du barrage sont utilisés ici.

La totalité du domaine est maillé avec 500 000 nœuds et 2 750 000 éléments tétraédriques. Les dimensions du goulot sont de (20 x 20 x 10) centimètres. Les cavités inférieures et supérieures sont de même taille, soit (1 x 1 x 0.40) mètres. Le solveur pour l'écoulement ne permet pas un pas de temps supérieure à 0.01 seconde. Un nombre très important d'incrément de temps et donc nécessaire (1000 incréments pour 10 secondes de simulation). Les paramètres utilisés sont les suivants : le fluide a une viscosité de 10 Pa.s (à peu près celle d'un miel liquide) et une masse volumique de 1000 kg/m^3, l'air a une viscosité de 1 Pa.s et une masse volumique de 1 kg/m^3. Les conditions aux limites sont de type collant, c'est-à-dire qu'une vitesse nulle est imposée sur toutes les surfaces de la géométrie.
Sur la figure *(12)* est représentée l'interface entre le fluide et l'air aux temps $t = 0s$, $t = 1s$, $t = 3s$, $t = 6s$, $t = 11s$, et $t = 18s$. Alors que des bulles d'air montent dans le jerricane (la cavité du haut), le fluide coule dans une cuvette (la cavité du bas) à travers le goulot du jerricane. Le niveau du fluide baisse donc dans le jerricane et monte dans la cuvette.

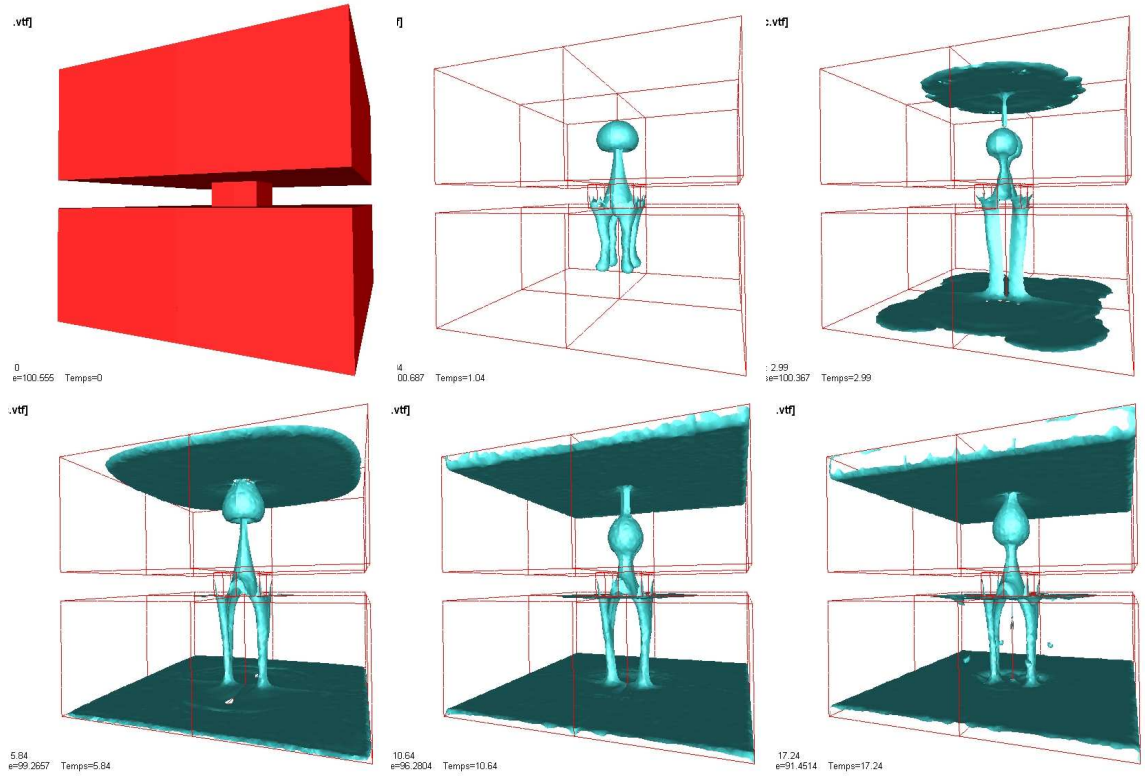

Figure 12 – Surface libre lors du déversement d'un jerricane

Cette application est intéressante car, premièrement, elle induit des mouvements de surface libre complexes, et deuxièmement, sa géométrie facilite un partitionnement adaptée à la grille de calcul. En effet, l'existence du goulot (partie étroite de la cavité) permet – si le partitionnement est bien réalisé – de diminuer considérablement les communications inter sites. Pour cela, il suffit d'attribuer la cavité du haut à un site, et celle du bas à un autre site. De cette manière, c'est seulement à l'endroit de la partie étroite que les communications inter sites s'opèrent, et elles deviennent donc bien moins importantes que les autres communications plus performantes.

Un total de 15 processeurs, et les 3 clusters nina, pf et iusti de MecaGrid sont choisis pour réaliser cette application. Il reste à déterminer combien de processeurs sont attribués sur

chaque cluster. Chacun des 2 sites s'occupe d'une partie de la cavité afin de pouvoir placer l'interface entre les deux parties du maillage au niveau de la partie étroite : à l'INRIA, 4 nina et 5 pf ont la partie supérieure du maillage ; à l'IUSTI, 6 processeurs ont la partie inférieure. Ce choix se justifie par la puissance calculée des processeurs : pour les nina et les iusti, la puissance est d'environ 125 Mflops/s, et pour les pf, elle est de 44 Mflops/s, soit 2.5 fois inférieure. Sur le site de l'INRIA, 2 nina sont remplacés par 5 pf. Ainsi, avec 4 nina, 5 pf et 6 iusti, la puissance de calcul distribuée sur les deux sites sont quasiment les mêmes.

Les mesures de temps suivantes sont relevées après 100 incréments de temps, avec des partitions homogènes, puis optimisées. Dans le premier cas (tableau *14*), le préconditionnement ILU(0) est utilisé par le solveur des écoulements, mais dans le deuxième cas (tableau *15*), c'est avec ILU par blocs de 1 : ILU(1).

ILU(0)	4 5 6	4 5 6
Nb processeurs	15	15
Partition	Homogène	Optimisée
Nb itérations	10 702	10 847
NS assemblage	2 926	1 743
NS résolution	526 659	212 962
Alpha assemblage	1 846	1 808
Alpha résolution	231	62
Total (s)	531 662	216 575

Tableau 14 – Temps d'exécution sur la grille avec le préconditionneur ILU(0)

Avec ILU(0), le gain de temps apporté par le partitionnement optimisé est de 60%, ce qui est un excellent résultat. Le temps d'exécution est passé de 6 jours et 4 heures à 2 jours et 12 heures. Le progrès apporté par l'optimisation des partitions est spectaculaire. Mais puisque les résultats du paragraphe précédent avec l'écroulement du barrage sont obtenus avec le préconditionneur ILU(1), aucune comparaison peut être tenue pour l'instant.

ILU(1)	4 5 6	4 5 6
Nb processeurs	15	15
Partition	Homogène	Optimisée
Nb itérations	2 210	1 904
NS assemblage	3 680	2 455
NS résolution	235 504	107 899
Alpha assemblage	2 357	2 896
Alpha résolution	258	113
Total (s)	241 799	113 364

Tableau 15 – Temps d'exécution sur la grille avec le préconditionneur ILU(1)

La première observation est qu'avec le préconditionneur ILU(1), les résultats sont bien meilleurs que dans le tableau *(14)* : le temps d'exécution (avec un partitionnement homogène) correspond à 45% de celui obtenu avec ILU(0). Cette méthode fait diminuer le nombre d'itérations nécessaires à la convergence des systèmes linéaires de presque 5 fois, mais le coût de celles-ci augmente (106 secondes par itération contre 49 secondes). Au finale, utiliser ILU(1) est quand même bien plus avantageux que ILU(0).

Deuxièmement, avec ILU(1), le gain de temps apporté par le partitionnement optimisé est légèrement inférieure qu'avec ILU(0). Le temps d'exécution est passé de 2 jours et 19 heures à 1 jour et 8 heures, ce qui fait un gain de 53%, contre 60% avec ILU(0).

Dans l'application de l'écroulement du barrage, ILU(1) est utilisé et seulement 20% de temps est gagné par l'optimisation des partitions. Pourquoi donc, dans les mêmes conditions, cette technique fait-elle gagner 20% dans l'écroulement du barrage, et 53% dans la déverse d'un jerricane ? C'est la géométrie de la cavité et la façon dont on la partitionne qui est crucial. Avec une géométrie purement cubique, l'apport d'une partition optimisée ne joue quasiment que sur une meilleure répartition de la masse de calcul, car les communications pénalisantes à l'interface des partitions ne peut pas être fortement diminuées. Or, en plus de répartir équitablement la masse de calcul, une géométrie telle que celle utilisée ici permet justement de diminuer considérablement les communications inter sites très pénalisantes en « coupant » le maillage dans un endroit où il y a peu de nœuds. Au total, l'addition des deux techniques d'optimisation (partitionnement et ILU(1)) fait gagner presque 80% sur les temps de calcul de MecaGrid.

L'instrumentation de PETSc monte les mesures de puissance suivantes pour ces exécutions sur 15 processeurs de MecaGrid : 915 Mflops/s avec un partitionnement homogène et ILU(0) ; et 1.96 Gflops/s avec un partitionnement optimisé et ILU(1).

6.6. Conclusion

Compte tenu des caractéristiques de la grille de calcul, il y a des applications qui offrent plus de possibilités d'optimisations que d'autres. Lorsqu'il la cavité est de forme cubique, la seule optimisation possible sur les partitions se situe au niveau de la répartition des tâches. Mais, avec d'autres géométries plus adaptées, non seulement la masse de calcul est mieux répartie entre les processeurs suivant leur puissance, mais aussi, les communications pénalisantes sont minimisées au profit des réseaux les plus rapides.

7. Etude parallèle de la grille

7.1. Définition de l'efficacité parallèle

L'accélération (*speed up*) et l'efficacité parallèle sont des grandeurs habituellement utilisées pour évaluer les performances d'une application parallèle sur machine homogène. L'accélération correspond au gain de temps apporté par la parallélisation de l'exécution. C'est, en fait, le rapport entre les temps d'exécution en série (sur un seul processeur) et en parallèle (sur p processeurs) d'un même problème. Les temps d'exécutions sont notés ainsi :

$$t(n, p) \qquad (5)$$

où n est la taille du problème à résoudre et p est le nombre de processeurs utilisées. L'évolution de l'efficacité en fonction du nombre de processeurs permet d'évaluer la scalabilité d'un code. Une efficacité de 1 représente une parallélisation parfaite, c'est-à-dire que le temps de résolution d'un problème est divisé exactement par le nombre de processeurs. L'accélération et l'efficacité sont calculées comme suit :

$$Acc = \frac{t(n,1)}{t(n,p)} \qquad (6)$$

$$Eff = \frac{Acc}{p} \qquad (7)$$

Cependant, un problème se pose lorsque l'on parle de machine parallèle hétérogène et de grille de calcul. La définition de l'accélération (6) n'est plus consistante car une dépendance se crée pour le choix du processeur utilisé dans la mesure du temps d'exécution $t(n,1)$. Si, par exemple, on veut calculer l'accélération obtenue sur MecaGrid avec les clusters nina et cemef, le résultat sera très différent si l'on utilise un nina ou un cemef pour mesurer $t(n,1)$. Une solution intuitive pour adapter la définition (6) pour les grilles de calcul est de ne plus prendre $t(n,1)$ comme temps de référence, mais $t(n,r)$ qui correspond au temps d'exécution, non pas en série, mais sur un nombre *r* de processeurs spécifiques :

$$AccGrille = r \times \frac{t(n,r)}{t(n,p)} \qquad (8)$$

$$EffGrille = \frac{AccGrille}{p} \qquad (9)$$

Le choix des *r* processeurs dépend des ressources de calcul (ici des clusters) utilisées pour mesurer $t(n,p)$: ils doivent inclure un processeur par cluster employé. Si les *p* processeurs sont tous situés sur 2 clusters différents (par exemple : nina et cemef), les *r* processeurs seront d'un nombre de 2, avec un sur nina et un autre sur cemef. $t(n,r)$ peut ainsi être mesuré en concordance avec le choix au préalable de *p*. Pour que cela reste utile, il faut que *r* puisse être suffisamment petit par rapport à *p*. Entre autre, il faut que les ressources de calcul – parmi lesquelles les *p* processeurs sont choisis – puissent être regroupées entre elles en terme de performance. Sinon, le calcul de telles grandeurs risque de porter un intérêt minime. Même si cette nouvelle définition pour l'accélération peut comporter quelques inconvénients, c'est celle-là que l'on utilisera par la suite pour étudier les codes de calculs sur MecaGrid.

7.2. Essais parallèles

Les tests effectués pour étudier l'accélération et l'efficacité parallèle de MecaGrid sont fait à partir d'une application réelle de simulation numérique. C'est le cas du déversement d'un jerricane similaire à celui présenté dans le paragraphe précèdent qui est choisi. Seulement, un maillage plus grossier à 120 000 nœuds a été créé afin de pouvoir passer l'exécution sur un nombre minimal de processeur, compte tenu de la quantité de mémoire maximale que peut contenir la RAM disponible. Avec un pas de temps fixé à 0.01 seconde, 100 incréments sont effectués pour une simulation numérique d'un total de 1 seconde. Les modèles et les méthodes numériques sont identiques à celles présentées dans cette thèse et utilisées dans les applications ci-dessus. Les systèmes non linéaires résultant des équation de Navier Stokes sont préconditionnés par ILU(0), tandis que pour la capture des interfaces par Level Set, les systèmes linéaires sont préconditionnés par une méthode Jacobi par blocks. Toutes les résolutions sont faites par la méthode itérative du résidu général minimal GEMRES de PETSc.

Cette même application est lancée plusieurs fois sur MecaGrid avec des clusters prédéfinis : nina, pf et iusti. C'est donc en prenant 1 processeur sur chacun de ces 3 clusters (3x1) que le cas de référence correspondant à $t(n,r)$ est exécuté. Ensuite, c'est avec 6 (3x2), 12 (3x4), puis 24 (3x8) processeurs que l'exécution est répétée. Les partitions utilisées étant ici homogènes, le nombre de nœuds contenus dans chaque partition de maillage est sensiblement le même sur tous les processeurs, et dépend du nombre total de processeurs, vu que le maillage reste le même.

Dans le tableau *(16)*, les temps d'exécution sont affichés dans chacun des cas. De plus, le nombre d'itérations total nécessaire pour faire converger les systèmes linéaires permet de calculer le temps passé en moyenne pour une itération.

Exécution	3x1	3x2	3x4	3x8
Nb processeurs	3	6	12	24
Nœuds/proc	39 843	19 921	9 961	4 980
Nb itération	365 000	447 900	533 600	543 600
Temps total	241 285	151 582	97 558	162 878
Temps/itér	0.66	0.34	0.18	0.30

Tableau 16 – Temps d'exécution sur la grille de calcul

Augmenter le nombre de processeurs fait baisser la durée d'exécution jusqu'à 12 processeurs. Au-delà de cette limite, elle se met au augmenter, comme le montre la figure *(13)*.

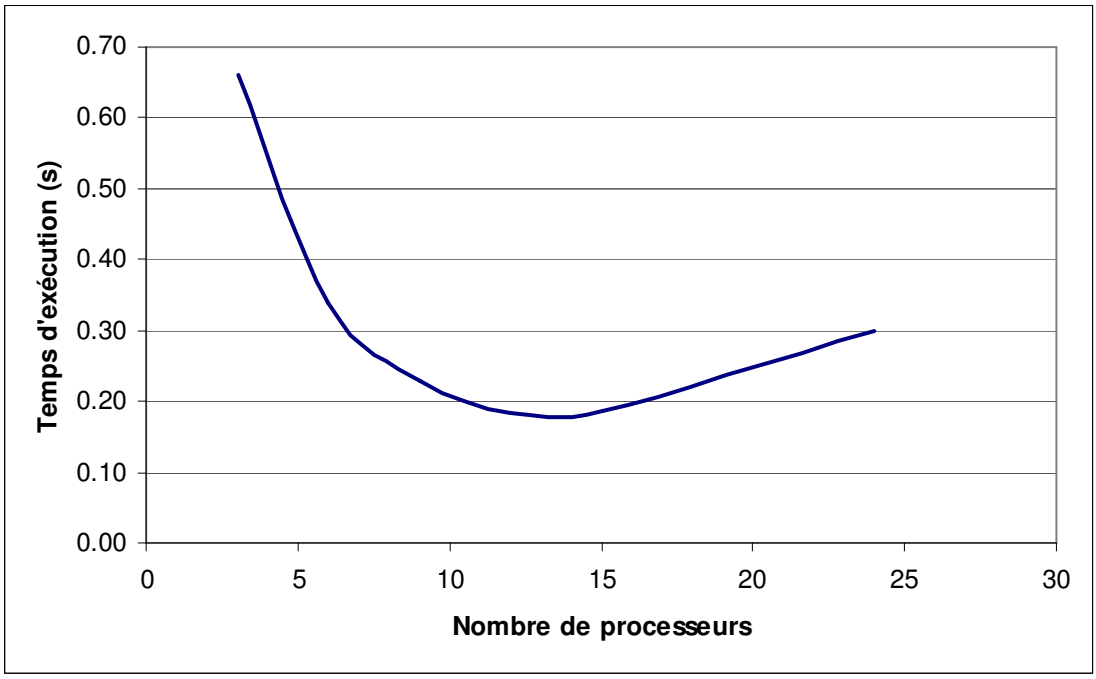

Figure 13 – Temps d'exécution en fonction du nombre de processeurs

Ce résultat pourrait paraître surprenant, mais, le fait de diviser la masse de calcul de façon trop importante entraîne une augmentation du rôle que jouent les communications – inter processeurs et inter clusters – par rapport à celui des calculs eux-mêmes. Ce premier constat amène à penser que la quantité de travail assignée aux processeurs doit être maximale pour obtenir de bonnes performances avec MecaGrid.

On remarque aussi que le nombre de d'itérations nécessaire à la convergence des systèmes augmente avec le nombre de processeurs. La figure *(14)* montre l'évolution du nombre d'itération total en fonction du nombre de processeurs. Cette augmentation est conséquente, et ne peut être négligée. Elle est due à l'influence qu'ont les partitions sur les méthodes de préconditionnement : plus le nombre de partitions est grand, plus les préconditionneurs ont du mal à conditionner les systèmes. C'est ainsi que la convergence vers les solutions est plus lente.

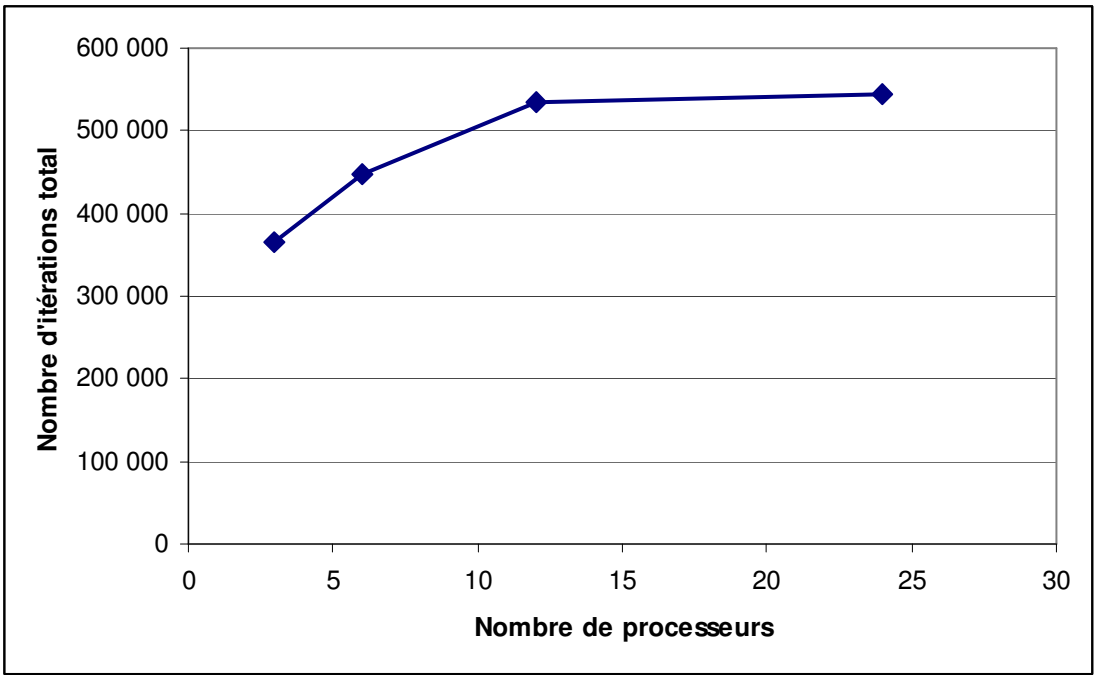

Figure 14 – Nombre d'itérations en fonction du nombre de processeurs

7.3 Efficacité et scalabitité sur MecaGrid

Avec les temps d'exécution du tableau *(16)*, l'accélération parallèle et l'efficacité peuvent être étudiées. Dans un premier, c'est avec les temps passés en moyenne pour une itération (temps total divisé par le nombre total d'itérations effectuées), et dans un deuxième temps avec les temps d'exécution totaux.

	3x2	3x4	3x8
Accélération	5.86	10.85	6.62
Efficacité	0.98	0.90	0.28

Tableau 17 – Efficacité parallèles (par itération)

Les résultats de ce premier tableau *(17)* sont indépendants du nombre d'itérations : le fait que celui-ci varie n'apparaît pas. Ce sont donc l'accélération et l'efficacité des calculs eux-mêmes, indépendamment de l'influence que peuvent avoir les partitions sur la résolution des systèmes. Ici, les efficacités parallèles pour 6 et 12 processeurs sont bonnes, car supérieure à 0.90 ; mais elle est médiocre pour 24 processeurs. Cela se traduit par une mauvaise scalabilité pour le code de calcul sur MecaGrid. La scalabilité d'un code est sa faculté de garder une bonne accélération, quelque soit le nombre de processeurs choisis pour le calcul parallèle. La figure *(15)* montre la scalabité lorsque l'on ne prend pas en compte le nombre d'itération. Elle

est plutôt bonne lorsque peu de processeurs sont utilisés (et donc lorsque la masse de calcul sur chaque processeur est grande), mais devient médiocre dans le cas contraire.

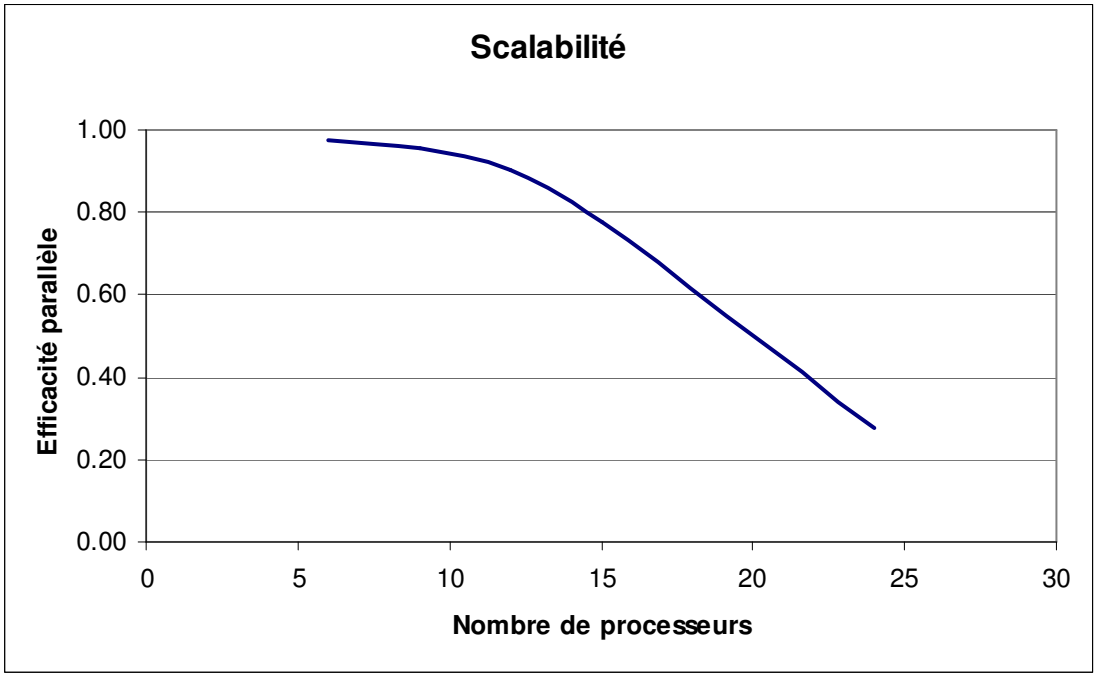

Figure 15 – Scalabilité (par itération)

Le tableau *(18)* est similaire au *(17)*, mais cette fois le nombre d'itérations est pris en compte. Puisque celui-ci augmente avec le nombre de processeur, il est logique que l'efficacité soit moins bonne que dans le cas précédent (tableau *17*), surtout sur 12 et 24 processeurs.

	3x2	3x4	3x8
Accélération	4.78	7.42	4.44
Efficacité	0.80	0.62	0.19

Tableau 18 – Efficacité parallèles (en totalité)

En plus d'une mauvaise scalabilité naturelle sur grille de calcul, il s'ajoute ici le fait la convergence des systèmes à résoudre devient plus difficile lorsque le nombre partitions de maillage augmente. La figure *(16)* illustre ce cas, en comparaison avec la figure *(15)* précédente : la scalabilité totale est encore plus dégradée.

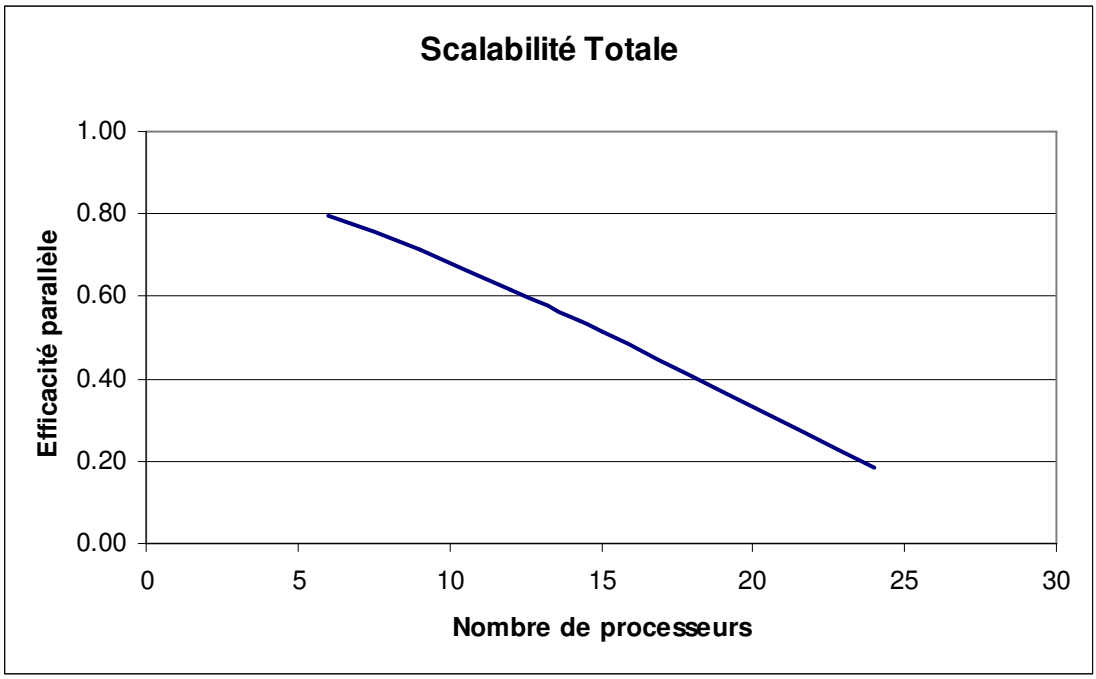

Figure 16 – Scalabilité (en totalité)

7.4. Conclusion

En général, les mauvaises communications inter cluster dues aux connexions internet font qu'il est préférable d'utiliser la grille pour effectuer des simulations à grandes échelles (et donc avec de gros maillages). Les petits maillages ne sont pas favorables à la grille car la masse de calcul des processeurs n'est pas assez importante vis-à-vis des communications pénalisantes. Et même, pendant une petite simulation, il est possible de passer plus de temps dans les communications que dans le calcul lui-même. Contrairement, avec des gros maillages, les tâches effectuées par les CPU prennent le dessus sur les échanges de messages (de données).

Ces derniers résultats montrent qu'une utilisation performante de MecaGrid passe par une répartition des tâches qui donne aux processeurs une masse de calcul conséquente pour que celle-ci garde le dessus sur les communications qui, nous l'avons vu, peuvent être médiocres dans MecaGrid. Dans l'idéale, il faudrait donc donner aux processeurs des partitions de maillage contenant le maximum de nœuds possible, tout en restant vigilant à la quantité de RAM disponible, car la dépasser amènerait des swaps de mémoire qui ralentissent les calculs. Ici, on observe pour avoir une efficacité proche de 1.0, il faut que la partition de maillage contienne un nombre de nœuds d'au moins 50 000. C'est dans la pratique, la valeur maximale permise par des processeurs aillant 256 Mo de RAM disponible.

8. Conclusion

Une grille de calcul a été élaborée avec succès en reliant quatre clusters de trois centres de recherche indépendants et géographiquement dispersés. Les nombreux problèmes rencontrés ont été résolus après avoir considéré les technologies disponibles à l'heure actuelle. Notamment, le middleware Globus apporte beaucoup lorsqu'il s'agit de coordonner différentes ressources indépendantes qui ont leurs propres politiques de sécurité. Le résultat est une grille de calcul très hétérogène dans sa conception, avec des processeurs et des réseaux très différents. Cet outil est donc logiquement plus moins performant que d'autres machines plus conventionnelles, mais il a l'avantage d'offrir une grande quantité de ressources de calculs à moindre coût. Les différentes études ont permis d'identifier plusieurs problématiques qui peuvent être améliorées avec des techniques d'optimisation adaptées. Ainsi, des utilisations plus performantes de la grille ont été définies avec succès. Enfin, une connaissance approfondie de cette infrastructure permet de la mettre en valeur avec des applications en mécanique des fluides hétérogènes. Tout est réuni pour réaliser des simulations à grande échelle. Et c'est dans cette optique que la suite de cette thèse consiste à rechercher des méthodes numériques adaptées aux gros calculs, dans le cadre de la simulation d'écoulements multi fluides. Cela veut dire que l'on privilégiera des techniques permettant des résolutions simples et des utilisations minimales de mémoire virtuelle. Enfin, des simulations à grande échelle de surfaces libres pourront être exécutées sur la grille de calcul.

Références

[Osh88] S. Osher and J. A. Sethian, “Fronts propagating with curvature dependent speed: Algorithms based on Hamilton-Jacobi formulation”, *J. Comput. Phys.* 79, 12 (1988).

[Eva91] L. C. Evans and J. Spruck, “Motion of level set via mean curvature I”, *J. Diff. Geom.* 33, 635 (1991).

[Che91] Y. G. Chen, Y. Giga and S. Goto, “Uniqueness and existence of viscosity solutions of generalized mean curvature flow equations”, *J. Diff. Geom.* 33, 749 (1991).

[Mer94] B. Merriman, J. Bence, and S. Osher, “Motion of multiple junctions: A Level Set approach”, *J. Comput. Phys.* 112, 334 (1994).

[Gui94] H. Guillard and B. N’Konda, “Godunov type method on non-structured meshes for three dimensional moving boundary problems”, *Comput. Methods Appl. Mech. Eng.* 113, 183 (1994).

[Ada95] D. Adalsteinsson and J. A. Sethian, “A fast level set method for propagating interfaces”, *J. Comput. Phys.* 118, 269 (1995).

[Shi95] B. A. Shirazi, A. R. Hurson and K. M. Kavi, “Scheduling and load balancing in parallel and distributed systems”, *IEEE Computer Science Press* (1995).

[Bun95] P. Buning,W. Chan, K. Renze, D. Sondak, I.-T. Chiu, J. Slotnick, R. Gomez, and D. Jespersen, “Overflow user’s manual, version 1.6”, *NASA Ames Research Center* (1995).

[Kar95] G. Karypis and V. Kumar, “A fast and high quality multilevel scheme for partitioning irregular graphs”, *Department of Computer Science Tech. Rep. 95-035, University of Minnesota* (1995).

[Sus96] M. Sussman, P. Smereka, and S. Osher, “A level set method for computing solutions to incompressible two-phase flow”, *J. Comput. Phys.* 122, 179 (1996).

[Zha96] H. K. Zhao, T. Chan, B. Merriman and S. Osher, “A variational level set approach to multi-phase motion”, *J. Comput. Phys.* 122, 179 (1996).

[Lee96] C. Lee, C. Kesselman, S. Schwab, “Near-real-time Satellite Image Processing: Metacomputing in CC++”, *Computer Graphics and Applications*, 16(4):79-84 (1996).

[Har96] E. Harabetian, S. Osher and C. W. Shu, “An Eulerian approach for vortex motion using a level set regularization procedure”, *J. Comput. Phys.* 127, 15 (1996).

[Gro96] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, “A highperformance, portable implementation of the MPI Message Passing Interface Standard”, *Parallel Computing*, 22, 789 (1996).

[Fos97] I. Foster and C. Kesselman "Globus : A metacomputing infrastructure toolkit", *The International Journal of Supercomputer Applications and High Performance Computing*, 11, 2 (1997).

[Fos97] I. Foster and C. Kesselman, "Globus: A metacomputing infrastructure toolkit", *International Journal of Supercomputer Applications*, 11, 115 (1997).

[Zha98] H. K. Zhao, B. Merriman, S. Osher and L. Wang, "Capturing the behaviour of bubbles and drops using the variational level set approach", *J. Comput. Phys.* 143, 495 (1998).

[Wis98] A. Wissink and R. Meakin, "Computational fluid dynamics with adaptive overset grids on parallel and distributed computer platforms", *In Intl. Conf. on Parallel and Distributed Processing Techniques and Applications*, 1628 (1998).

[Fos98] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke, "Wide-area implementation of the Message Passing Interface", *Parallel Computing*, 24, 1735 (1998).

[Pen99] D. Peng, B. Merriman, S. Osher, H. Zhao, and M. Kang, "A PDE-Based Fast Local Level Set Method", *J. Comput. Phys.* 155, 410 (1999).

[Fos99] I. Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", *Morgan Kaufmann* (1999).

[Alle99] G. Allen, W. Benger, C. Hege, J. Mass´o, A. Merzky, T. Radke, E. Seidel, and J. Shalf, "Solving Einstein's equations on supercomputers", *IEEE Computer*, 32, 12 (1999).

[Bar99] S. Barnard, R. Biswas, S. Saini, R. Van der Wijngaart, M. Yarrow, L. Zechter, I. Foster and O. Larsson, "Large-Scale Distributed Computational Fluid Dynamics on the Information Power Grid using Globus", *Proc. of Frontiers '99* (1999).

[Bro99] M. D. Brown, T. DeFanti, M. A. McRobbie, A. Verlo, D. Plepys, D. F. McMullen, K. Adams, J. Leigh, A. E. Johnson, I. Foster, C. Kesselman, A. Schmidt, S. N. Goldstein, "The International Grid (iGrid): Empowering Global Research Community Networking Using High Performance International Internet Services", *iGrid* (1999).

[Bas00] A. Basermann, J. Clinckemaillie, T. Coupez, J. Fingberg, H. Digonnet, R. Ducloux, J. M. Gratien, U. Hartmann, G. Lonsdale, B. Maerten, D. Roose and C. Walshaw, "Dynamic load balancing of finite element applications with the DRAMA library", *Appl. Math. Modelling*, 25 (2), 83 (2000).

[Rip01] M. Ripeanu, A. Iamnitchi, and I. Foster, "Performance Predictions for a Numerical Relativity Package in Grid Environments", *International Journal of High-Performance Computing Applications*, 15, 4 (2001).

[Alle01] G. Allen, T. Dramlitsch, I. Foster, N. Karonis, M. Ripeanu, E. Seidel and B. Toonen, “Supporting Efficient Execution in Heterogeneous Distributed Computing Environments with Cactus and Globus”, *Proceedings of SC 2001*, 10 (2001).

[Allc01] B. Allcock, I. Foster, V. Nefedova, A. Chervenak, E. Deelman, C. Kesselman, J. Leigh, A. Sim, A. Shoshani, B. Drach, D. Williams, “High-Performance Remote Access to Climate Simulation Data: A Challenge Problem for Data Grid Technologies”, *Proceedings of SC 2001* (2001).

[Bal02] S. Balay, K. Buschelman,W. D. Gropp, D. Kaushik, M. Knepley, L. C. McInnes, B. F. Smith and H. Zhang, “PETSc : Users Manual ANL-95/11 - Revision 2.1.5.”, *Argonne National Laboratory* (2002).

[Pea04] L. Pearlman, C. Kesselman, S. Gullapalli, B.F. Spencer, Jr., J. Futrelle, K. Ricker, I. Foster, P. Hubbard and C. Severance, “Distributed Hybrid Earthquake Engineering Experiments: Experiences with a Ground-Shaking Grid Application”, *Proceedings of the 13th IEEE Symposium on High Performance Distributed Computing* (2004).