Vers un paradigme de programmation orienté workflow pour la résolution de méthodes d'algèbre linéaire sur des plateformes de calcul global à faible consommation énergétique

par Laurent Choy

Thèse présentée à

l'Université des Sciences et Technologies de Lille

pour obtenir le titre de

Docteur en Informatique

Thèse soutenue le 28 septembre 2007 devant la Commission d'Examen:

Président:	Bernard	TOURSEL	Professeur LIFL, USTL, France
Rapporteurs:	Michel	DAYDE	Professeur ENSEEIHT-IRIT, France
	Brigitte	PLATEAU	Professeur INPG, France
Examinateurs:	Franck	CAPPELLO	Directeur de Recherche INRIA Futurs, France
	Jack	DONGARRA	Professeur ORNL, Université du Tennessee, USA
	Mitsuhisa	SATO	Professeur HPCS Lab., Université de Tsukuba, Japon
Directeur de thèse:	Serge	PETITON	Professeur LIFL, USTL, France

Toward a workflow programming paradigm for linear algebra on power-aware Global Computing platforms

by Laurent Choy

A Dissertation Submitted in
Partial Fulfilment of the
Requirements for the Degree of

Doctor of Philosophy
in Computer Science

at
the University of Science and Technology of Lille

Dissertation presented on September 28^{th} 2007 to the Committee:

Head of Committee:	Bernard	TOURSEL	Professor LIFL, USTL, France
Dissertation Committee:	Michel	DAYDE	Professor ENSEEIHT-IRIT, France
	Brigitte	PLATEAU	Professor INPG, France
Defense Committee:	Franck	CAPPELLO	Senior research scientist INRIA Futurs, France
	Jack	DONGARRA	Professor ORNL, University of Tennessee, USA
	Mitsuhisa	SATO	Professor HPCS Lab., University of Tsukuba, Japan
PhD Advisor	Serge:	PETITON	Professor LIFL, USTL, France

Préambule Grâce au soutien du Collège Doctoral Franco-Japonais et à l'équipe associée FJ-Grid de l'INRIA Futurs, j'ai eu la chance de passer un tiers de ma thèse au laboratoire HPCS de l'Université de Tsukuba au Japon. Les recherches que nous présentons dans ce mémoire sont le fruit d'une collaboration étroite avec nos partenaires japonais. Ces derniers ont montré un intérêt certain à la rédaction de ce document. Ainsi, après un résumé étendu de nos travaux en langue française, nous développons notre étude en langue anglaise.

A though to all members of my family

Acknowledgments

Since it is time to look backwards, I realize the great opportunity I had to do this PhD. It was of course a rich period from a scientific point of view. I feel also very lucky for having combined this work with a great experience in Japan.
Therefore, I would like to thank my PhD adviser, Serge Petiton. I really appreciated working with him since his leadership was a perfect balance between guidance and liberty. I am also grateful for his proposition to work with the Professor Sato's team. He trusted me and I hope I have not deceived him.
I acknowledge Mistuhisa Sato who received me in its laboratory for 2 months in 2005 and then 8 months in 2006-2007. I thank him for his help in order to get several funding and for having taken care of my integration in Japan. He gave me much valuable advice and he was very patient while I wrote this dissertation. I hope we will work again together in the future.

I express my gratitude to Brigitte Plateau and Michel Dayde, the reviewers of my PhD dissertation. I thank also the members of the jury, Franck Cappello, Jack Dongarra, Mitsuhisa Sato and Bernard Toursel who accepted to preside my defense. I am greatly honoured by their presence.

Next, I especially ackowledge Olivier Delannoy. Although we were not in the same laboratory, we always worked together. It was a profitable and enjoyable collaboration. While I was in Japan, he was one of the rare constant link to France. Therefore, I cannot thank him enough for his indirect support.
The following greetings are dedicated to Amine Aouad, Haiwu He, Saad Amrani, Toussaint Guglielmi, Olivier Soyez, Jie Pan, Ye Zhang and Guy Bergère. The MAP team laboratory was a pleasant place to do research: relaxed and friendly but always serious. All of them have contributed to build this good atmosphere. I particularly enjoyed working with Amine and Haiwu, and I felt bad when they left the office. I hope we will meet again.
I thank all members of the HPCS laboratory for their welcome. They tried and succeeded to make my integration as easy as possible. The task was pretty hard due to the gaps of language and culture between us. I am very grateful to Yoshihiro Nakajima and Yoshihiko Hotta for their constant help and their advice.

I have enjoyed working as a member of the Grand Large project of the INRIA Futurs and, more generally, at the INRIA. I have met lots of kind and obliging people during my interships and my PhD. I particularly thank Philippe Augerat who was the first person to guide my researches. I do not forget his support in order to enter the ENST Paris and then to get a PhD funding. I also thank again, Franck and Serge, for having been my advisors of Master internship and for having supported and encouraged my collaboration with the HPCS laboratory through the FJ-Grid associated team. Last but not least, I sincerely acknowledge Gina Grisvard for her patience and her kindness.

Contents

List of Figures

List of Tables

List of Algorithms

Première partie

Résumé étendu - Extended abstract in French

Chapitre 1

Introduction

1.1 Contexte large

Le calcul parallèle, à l'origine domaine privilégié des super-calculateurs, est de plus en plus réparti. Cette tendance est accompagnée d'une hétérogénéité grandissante des ressources de stockage et calcul, mais également d'une grande variété de réseaux parmi lesquels nous comptons désormais Internet.

Les intergiciels sont confrontés à de nouvelles contraintes inhérentes à l'ouverture sur Internet telles la volatilité des machines, les problèmes de confidentialité. Ils doivent en outre masquer l'hétérogénéité des ressources de calcul et des réseaux afin que les utilisateurs se concentrent sur les aspects algorithmiques de leurs applications. Malgré les efforts réalisés au niveau des intergiciels de calcul parallèle et réparti, un travail significatif doit être fait au niveau applicatif pour répartir efficacement des applications à l'échelle d'Internet.

1.2 Contexte de l'étude

Les recherches menées par le projet Grand Large de l'INRIA Futurs couvrent un vaste spectre de l'informatique parallèle et répartie à grande échelle et se caractérisent par une approche très expérimentale. Au sein de l'équipe MAP, nous étudions le domaine applicatif et proposons des solutions méthodologiques et algorithmiques pour adapter efficacement des applications au contexte de calcul global. Les intergiciels actuels n'offrant pas encore une totale abstraction de la complexité des plateformes de calcul, nous devons aussi étudier des techniques de programmation spécifiques à des configurations logicielles et matérielles données.

Nous ne considérons pas les plateformes de calcul pair-à-pair ou les grilles de calcul comme des outils concurrents des super-calculateurs. Nous les voyons plutôt comme des outils complémentaires s'adressant généralement à des catégories d'utilisateurs différentes. En particulier, le calcul sur grille ou pair-à-pair ne prétend pas atteindre les plus hautes performances, en terme de puissance de calcul, mais il dispose d'atouts nombreux expliquant l'intérêt que nous lui portons.

Dans le cadre de l'équipe associée FJ-Grid du projet Grand Large de l'INRIA Futurs, nous nous sommes rendus à plusieurs reprises au laboratoire HPCS de l'Université de Tsukuba, Japon. Nous avons été sensibilisés à la problématique du calcul haute performance à faible consommation d'énergie, domaine dans lequel le laboratoire HPCS possède une expertise certaine. De plus, les recherches visant une réduction de la consommation énergétique des super-calculateurs sont de plus en plus nombreuses à l'image des récents projets de machines pétaflopiques. Par exemple, le RIKEN disposera en 2012 d'une machine atteignant les 10 pétaflops et dont les spécifications imposent une réduction de 10% de la consommation énergétique comparée à une machine de puissance identique mais composée d'éléments standards. Nous avons souhaité introduire dans notre étude sur le calcul global des considérations propres à la réduction de la consommation énergétique. A l'instar du calcul haute performance, qui est souvent technologiquement en avance, nous pressentons que cette thématique de recherche sera bientôt dynamique pour le calcul global.

1.3 Motivations

Principales motivations pour le calcul global

Bien que non quantifiées, les machines connectées à Internet nous laissent espérer répartir à très grande échelle des calculs nécessitant beaucoup de ressources telles l'espace disque, la mémoire. Ces machines peuvent être des ordinateurs standards appartenant à des particuliers et directement reliés à Internet, des grappes de PCs ou des réseaux de stations de travail appartenant à des institutions ou des entreprises, etc. Les progrès matériels, les gains de bande passante et le taux croissant de foyers équipés reliés à Internet sont des facteurs accroissant notre intérêt. Le principe du volontariat consistant à offrir ses ressources de calcul, lorsqu'elles sont inutilisées ou sous-exploitées, est une condition nécessaire à la viabilité du calcul global. Ce modèle est prometteur pour les personnes n'ayant pas accès à des ressources de calcul conséquentes. Ce manque est généralement le fait d'un coût élevé des machines performantes mais aussi des accès extrêmement restreints aux ressources existantes. En théorie, le calcul global offre une disponibilité constante de ressources en quantité suffisante. Ainsi, il est possible de résoudre une majorité de problèmes dès que le besoin se fait sentir. Occasionnellement, nous pensons que le calcul global peut compléter les ressources d'un super-calculateur lorsque ce dernier doit répondre à une charge imprévue de calcul ou de données à stocker. Cela évite un achat précipité voire inutile de machine.

Motivations pour le calcul global à faible consommation d'énergie

En règle générale, le client d'une plateforme de calcul global ne fait pas du temps d'exécution de son application un critère de performance majeur. Il peut accepter une légère hausse de ce temps de calcul afin qu'un mécanisme de réduction de la consommation énergétique soit mis-en-œuvre. En retour, suivant le modèle économique de la plateforme de calcul, le client peut bénéficier d'une compensation (ex. nombre de nœuds de calcul plus élevé). La mise-en-œuvre d'un système de réduction de la consommation énergétique est également avantageuse pour les autres parties d'une plateforme de calcul global. Les propriétaires des nœuds de calcul font des économies d'énergie (et donc financières). Ainsi, parmi l'ensemble des plateformes de calcul global, ces derniers seront tentés de rejoindre en priorité celles offrant de tels mécanismes. Nous voyons donc que le concepteur d'un intergiciel de calcul global accroît les chances de succès de son logiciel si celui-ci permet de réduire la consommation d'énergie.

La réduction de la consommation énergétique des machines n'est pas le seul bénéfice que nous pouvons obtenir. En effet, celles-ci ont bien souvent une dissipation thermique inférieure aux systèmes traditionnels. Il est possible de ne pas installer de mécanisme de refroidissement et donc de réaliser des économies. La faible dissipation thermique permet de réduire le risque de panne des machines et donc d'assurer une continuité de service. Ce point est essentiel pour les acteurs de l'e-économie. Enfin, la faible dissipation thermique permet la mise-en-œuvre de machines à forte densité en terme de $FLOPs/m^3, GB/m^3$. Cette diminution d'espace occupé, permise également par l'absence (ou le peu) de système de refroidissement, engendre une réduction non négligeable du coût d'exploitation.

Motivations pour l'étude de méthodes d'algèbre linéaire

Les méthodes d'algèbre linéaire requièrent généralement beaucoup de ressources de tous types : cycles CPU, mémoire, espace disque. De plus, la distribution de ces méthodes génère souvent beaucoup de communications. Les ressources réseaux à l'échelle d'Internet, tout comme les ressources CPU, mémoire et disque des machines volontaires, sont des données critiques car non dédiées et partagées. Il semble pertinent d'étudier comment adapter efficacement ces méthodes numériques au contexte du calcul global. En outre, les problèmes d'algèbre linéaire sont souvent le fondement d'applications plus complexes, utiles au quotidien, voire dans l'industrie. Nous étudions plus précisément le problème de la recherche des valeurs propres d'une matrice réelle symétrique. Il concentre toutes les motivations citées ci-dessus. De plus, il se décompose en plusieurs sous-problèmes que l'on peut retrouver au sein d'autres applications et il permet de manipuler plusieurs paradigmes de programmation.

1.4 Contributions

Nous avons étudié le paradigme du parallélisme paramétrique qui se distingue par une quasi-absence de communications. Il semble le candidat idéal pour la répartition d'applications numériques sur Internet. Nous avons confronté au sein de la même application ce paradigme avec un parallélisme plus classique engendrant communications et points de synchronisation. Ensuite, nous avons étudié le problème de la gestion des données : partage et placement au sein des ressources de calcul et stockage. Nous avons repoussé le plus longtemps possible les considérations relatives aux langages de programmation et aux intergiciels de calcul. En effet, nous souhaitons apporter une méthodologie et des solutions aux contraintes du calcul global qui soient aisément reproductibles avec d'autres problèmes. De plus, notre algorithme peut ainsi être porté sur de nombreuses plateformes au moyen de changements mineurs. Nous avons montré que le produit matrice-vecteur est un calcul très intéressant à exploiter sur des plateformes à grande échelle et pour des matrices de grandes dimensions. En effet, le mode d'accès aux données permet de multiples répartitions possibles et évite le chargement total des données en mémoire. En revanche, appliqué sans amélioration, le produit matrice-vecteur apporte de nombreuses et volumineuses communications qui sont problématiques sur Internet. Nous proposons donc un mécanisme de persistance de données qui réduit significativement le volume des communications. Nous proposons également des solutions réduisant l'usage de mémoire principale afin d'exploiter un spectre plus large de machines sur Internet. Nous proposons de faire de "l'out-of-core" et d'employer une version redémarrée de notre méthode numérique. Enfin, une contribution importante est de proposer une mise-en-œuvre de nos propositions, sans simulation et dans des conditions expérimentales réelles, c'est-à-dire mettant en jeu des ressources de calcul hétérogènes et des réseaux incluant Internet. L'objectif est double. Nous montrons la viabilité du calcul global pour résoudre de tels problèmes à condition que la rapidité de calcul ne soit pas un critère de performance requis (sans quoi, il faut se tourner vers le calcul haute performance). De plus, nous souhaitons encourager d'autres chercheurs à mener des recherches similaires dans le domaine des méthodes numériques sur les plateformes de calcul à grande échelle.

La programmation parallèle et répartie à grande échelle est une tâche complexe. De plus, la profusion d'intergiciels de calcul réparti tend à annihiler les espoirs fondés sur le nombre de machines potentiellement exploitables car il est impossible de développer une version de son application pour chaque interface de programmation. Ainsi, nous contribuons au développement d'YML. YML offre un environnement de programmation parallèle et répartie intuitif et masquant l'hétérogénéité des plateformes de calcul global. De cette participation, nous distinguons deux aspects. Dans un premier temps, nous avons développé plusieurs modules logiciels d'YML. Par exemple, nous avons mis-en-œuvre le module nécessaire à YML pour exploiter les plateformes de calcul reposant sur l'intergiciel OmniRPC. Le second aspect de notre travail concerne l'adaptation du problème de recherche des éléments propres d'une matrice. Nous mettons en valeur les différences méthodologiques et d'implémentation dues à un plus haut niveau d'abstraction par rapport à la précédente étude. Nous montrons comment exploiter certaines possibilités offertes par le

langage de graphe YvetteML et comment tirer profit de l'approche orientée composant d'YML.

Enfin, nous apportons une première contribution à un thème de recherche nouveau qu'est le calcul global à faible consommation énergétique. Nous avons montré comment tirer profit de l'hétérogénéité des processeurs des nœuds de calcul afin de réduire les consommations globale et locale sur chaque nœud. Nous avons évalué ces gains au moyen d'une application complexe, recherchant les éléments propres d'une matrice réelle symétrique, afin de donner une crédibilité à cette contribution et pour montrer que de tels gains sont possibles pour des applications utiles au quotidien.

Chapitre 2

Recherche des éléments propres d'une matrice réelle symétrique sur des plateformes de calcul global

2.1 Le paradigme idéal : le parallélisme paramétrique

Nous étudions en premier lieu le parallélisme paramétrique, très souvent appelé "task-farming". Ce paradigme de programmation parallèle consiste à soumettre plusieurs fois le même programme mais avec différents jeux de paramètres. L'intérêt majeur du parallélisme paramétrique pour le calcul global est le faible nombre de communications car les tâches sont indépendantes.

2.1.1 La recherche des éléments propres d'une matrice réelle symétrique tridiagonale

Soit T une matrice réelle symétrique tridiagonale, $T \in M_N(\mathbb{R})$. Nous cherchons les couples (λ,v) avec $v \in \mathbb{R}^N$ et $\lambda \in \mathbb{R}$ tels que $Tv = \lambda v$. Nous choisissons la méthode de la Bisection présentée en Partie II Section 3.2.6. Son intérêt majeur est de permettre la mise-en-œuvre d'un parallélisme paramétrique. Cette méthode est également intéressante car elle repose sur une récurrence et non sur de coûteuses opérations matricielles. Plus précisément, nous utilisons la méthode de la Bisection Parallèle, et non la méthode de Multi-section, car de précédents travaux dans le domaine du calcul haute performance l'estiment meilleure dans le cas de systèmes à mémoire répartie (voir en Partie II Section 3.3.2). Le principe de la Bisection Parallèle est simple. Nous faisons une partition de l'intervalle réel de recherche des valeurs propres (notons le I). Ensuite, nous soumettons à plusieurs nœuds de calcul le même programme réalisant une Bisection séquentielle avec différents sous-intervalles issus

de la partition de I.

Il est important d'équilibrer au mieux le nombre de valeurs et vecteurs propres à calculer parmi les nœuds de calcul. En particulier, nous devons prendre garde aux agglomérats de valeurs propres. Nous proposons de tenir compte de la répartition des valeurs propres dans I afin de calculer les sous-intervalles. Ainsi, nous considérons un paramètre seuil s indiquant le nombre maximum de valeurs propres par sous-intervalle. Nous appliquons l'algorithme séquentiel de la Bisection sur I et nous le stoppons dès que les sous-intervalles trouvés respectent le seuil donné. Les Figures 2.1 et 2.2 illustrent, pour un intervalle contenant deux amas de valeurs propres, une partition classique en sous-intervalles de même diamètre et notre proposition de partition. Actuellement, le seuil s est choisi de façon empirique. Il a une forte influence sur l'efficacité de notre méthode parallèle. Une trop grosse valeur peut ne pas éclater un agglomérat de valeurs propres. Une trop faible valeur génère beaucoup trop de sous-intervalles. Il n'est pas conseillé de soumettre 1 sous-intervalle par nœud de calcul car le ratio temps de calcul par temps de communication (soumission et récupération des résultats) serait mauvais. Nous proposons de constituer des paquets de sous-intervalles et de soumettre 1 paquet par nœud. Afin d'équilibrer la charge de travail, les paquets sont constitués au moyen d'une distribution cyclique des sous-intervalles calculés.

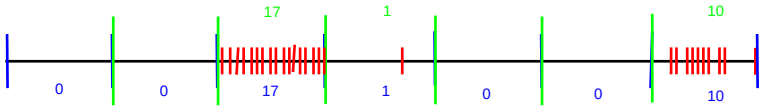

Fig. 2.1 – Exemple d'une partition classique d'un intervalle contenant 28 valeurs propres

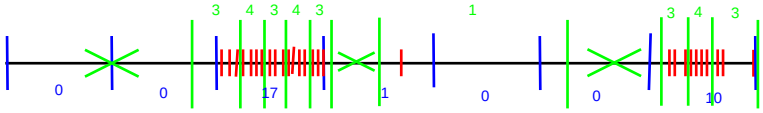

Fig. 2.2 – Exemple d'une partition équilibrée d'un intervalle contenant 28 valeurs propres (seuil=4)

2.1.2 Conditions expérimentales

Nous utilisons les logiciels OmniRPC et XtremWeb évoqués dans la Partie II Chapitre 2. Chacun représente un modèle dominant du calcul global. OmniRPC est un logiciel de programmation par RPC dans le domaine du calcul sur grille d'ordinateurs. XtremWeb symbolise le calcul pair-à-pair en exploitant les ressources inutilisées des ordinateurs connectés à Internet au moyen du principe de vol de temps de cycle.

Les ressources de calcul utilisées sont un réseau de stations de travail à l'Université de Lille 1 (USTL), France, ainsi que deux grappes de PCs à l'Université de Tsukuba, Japon. Dans le premier cas, les machines sont hétérogènes et non dédiées à nos expérimentations. Dans

le second cas, les PCs sont homogènes, dédiés et plus puissants. Les réseaux empruntés sont également hétérogènes. Que ce soit au Japon ou en France, la bande passante varie de 10 à 100 MBits tandis que celle-ci est en moyenne de 1.74MBits sur Internet entre les deux sites (au-dessus de TCP). La conjonction des ressources matérielles, réseaux et des deux logiciels de calcul permet de construire les 6 plateformes données dans la Table 2.1.

	Config. 1	Config. 2	Config. 3
Intergiciel	XtremWeb		
Emplacement du client	USTL		
Emplacement du dispatcher	USTL		
Nombre et emplacements des workers	50 USTL	50 Université de Tsukuba	24 USTL 26 Université de Tsukuba

	Config. 4	Config. 5	Config. 6
Intergiciel	OmniRPC		
Emplacement du client	USTL		
Nombre et emplacements des serveurs OmniRPC	2 USTL	2 à Tsukuba	1 USTL 1 à Tsukuba
Nombre de workers par serveur	25		
Nombre et emplacements des workers	50 USTL	50 Université de Tsukuba	25 USTL 25 Université de Tsukuba

Tab. 2.1 – Platformes expérimentales pour l'étude du parallélisme paramétrique

Nous utilisons une matrice réelle tridiagonale dont les composantes de la diagonale valent $a \in \mathbb{R}$ et les composantes des sous- et sur-diagonales valent $b \in \mathbb{R}$. Nous connaissons à l'avance les valeurs propres pour la vérification des résultats : $\lambda_j = a+2b\cos(\frac{j\pi}{N+1})$ avec $j = 0, \ldots, N-1$. Nous prenons $a = 2$, $b = 1$, $N = 10^5$, et un seuil de $s = 200$ valeurs propres maximum par sous-intervalle. L'intervalle initial est le domaine de Gerschgorin afin de calculer toutes les valeurs propres. Nous construisons 100 paquets de sous-intervalles et soumettons donc 100 tâches indépendantes.

Dans cette première partie de l'étude, nous formulons deux hypothèses majeures qui sont traitées dans la Partie I Section 2.2 : nous ne transférons pas la matrice qui est stockée au préalable par les nœuds de calcul et nous calculons les vecteurs propres sans les stocker sur le disque dur. Les besoins en mémoire sont modérés, au plus $162\,MO$ $((96+8*s)*N)$ et l'occupation d'espace disque est négligeable en raison de la deuxième hypothèse.

Les temps obtenus lors des expérimentations sont exprimés en secondes. Chaque temps horloge du client, Table 2.2, est une moyenne de 10 valeurs. La Table 2.3 présente les temps moyens de calcul d'un nœud (moyennes de 100 valeurs) pour obtenir ses couples de valeurs et vecteurs propres.

	Config. 1	Config. 2	Config. 3
Temps de soumission	1	1	1
Temps d'attente	4074	2739	2735
Temps de retrait des résultats	2	2	1

	Config. 4	Config. 5	Config. 6
Temps de soumission	1384	1607	1392
Temps d'attente	1165	1809	1463
et de retrait des résultats			

Rappel des config.	XtremWeb : 1-2-3,OmniRPC : 4-5-6 Lille : 1-4, Tsukuba : 2-5, Lille & Tsukuba : 3-6

Tab. 2.2 – Temps horloge du client pour obtenir les valeurs et vecteurs propres

	Config. 1	Config. 2	Config. 3
Moyenne des temps	(213, 928)	(112, 755)	(198, 980)
Temps minimum	(142, 467)	(63, 250)	(107, 491)

	Config. 4	Config. 5	Config. 6
Moyenne des temps	(192, 728)	(138, 930)	(180, 801)
Temps minimum	(140, 429)	(79, 325)	(95, 311)

Rappel config.	XtremWeb : 1-2-3,OmniRPC : 4-5-6 Lille : 1-4, Tsukuba : 2-5, Lille & Tsukuba : 3-6
	(a, b) = temps de calcul pour obtenir les (valeurs propres, vecteurs propres)

Tab. 2.3 – Temps de calcul des nœuds pour trouver leurs éléments propres

2.1.3 Résumé des analyses

Les tests expriment tout l'intérêt que nous devons porter au paradigme du parallélisme paramétrique dans un contexte de calcul global. Nous pouvons exploiter des machines connectées à Internet, quelque soit leur localisation, sans affecter de façon significative les temps horloge du client. En effet, nous ne comptons que deux communications par soumission. De plus, le volume de ces communications est modéré en raison de nos hypothèses pour les expérimentations. En travaillant à l'échelle d'Internet, et donc à l'échelle de la planète, nous pouvons en permanence tirer profit du décalage horaire afin d'exploiter des machines oisives durant la nuit. Nos expérimentations ont également révélé l'importance d'une configuration optimale du logiciel de calcul global. En effet, XtremWeb lance par défaut 1 processus par processeur d'un multi-processeur. Si l'application requiert beaucoup de mémoire, alors les temps de calcul peuvent être décevants. Les valeurs propres de la matrice utilisée pour les tests présentés dans ce document sont réparties uniformément. Nous avons testé d'autres matrices ayant des agglomérats de valeurs propres telle la matrice tridiagonale obtenue par tridiagonalisation de la matrice réelle symétrique Bcsstk25 issue de la collection Harwell-Boeing de "Matrix Market". Nous avons constaté des temps de calcul équilibrés entre les nœuds.

Dans le cadre du calcul global, les résultats obtenus montrent que notre problème d'algèbre linéaire est résolu efficacement au moyen de la Bisection Parallèle. L'usage du parallélisme paramétrique en est la principale explication. Ainsi, afin de résoudre des problèmes plus complexe, une démarche intéressante consiste en la recherche de sous-problèmes permettant la mise-en-œuvre de ce paradigme. Dans la section suivante, nous présentons comment paralléliser et répartir la recherche des valeurs et vecteurs propres d'une matrice réelle symétrique. En particulier, nous exploitons le travail de cette section en incluant la méthode de la Bisection Parallèle.

2.2 Recherche des éléments propres d'une matrice réelle symétrique dans un contexte de calcul global

Nous trouvons dans l'étude de ce problème numérique de multiples intérêts. Il est souvent utilisé au sein d'applications plus complexes. Sa distribution et sa parallélisation dans un contexte de calcul global représentent un véritable challenge. En effet, il requiert beaucoup de ressources (CPU, mémoire, espace disque, bande passante, etc). Il est également fortement communicant et possède plusieurs points de synchronisation. Enfin, il est possible de décomposer ce problème en sous-problèmes et d'utiliser pour l'un d'eux le paradigme du parallélisme paramétrique.

2.2.1 Méthode numérique

Soit une matrice réelle symétrique A. Nous cherchons les couples (λ,u) tels que $Au = \lambda u$ avec $dim(A) = N \times N$, $u \in \mathbb{R}^N$ et $\lambda \in \mathbb{R}$. Dans un premier temps, nous tridiagonalisons la matrice A au moyen de la méthode de Lanczos décrite Partie II Section 3.2.5. Nous obtenons alors une matrice tridiagonale symétrique T pour laquelle nous pouvons appliquer la méthode de la Bisection afin de trouver ses valeurs et vecteurs propres. Les valeurs propres de T telles que $Tv = \lambda v$ sont également celles de A, dîtes valeurs propres de Ritz. Enfin, nous calculons les vecteurs propres de Ritz $u(= Qv)$ et vérifions l'exactitude des résultats avec les normes euclidiennes des résidus $\|Au - \lambda u\|_2 (< \varepsilon)$.

Algorithm 1 Méthode de Lanczos pour la tridiagonalisation d'une matrice réelle symétrique

Require: A, q_0 (vecteur initial)

$q_{candidate} \leftarrow q_0$

$e_0 \leftarrow \|q_{candidate}\|_2$

for $k = 1, \ldots, N$, pas=1 **do**

1) Nouvelle colonne de la base Q

$q_k \leftarrow \frac{q_{candidate}}{e_{k-1}}$

2) Colonne candidate pour la $k+1^{\text{ème}}$ itération

$q_{candidate} \leftarrow Aq_k$

$q_{candidate} \leftarrow q_{candidate} - q_{k-1}e_{k-1}$

3) Nouvel élément de la diagonale

$d_k \leftarrow q_k^t q_{candidate}$

$q_{candidate} \leftarrow q_{candidate} - q_k d_k$

4) Reorthogalisation (ex. reorthogalisation totale)

$q_{candidate} \leftarrow q_{candidate} - Q^t Q q_{candidate}$

5) Nouvel élément de la sous-diagonale (fait si $k \neq N$)

$e_k = \|q_{candidate}\|_2$

end for

Notre motivation pour la méthode de Lanczos est principalement due à l'accès aux données de A et de Q au moyen de produits matrice-vecteur (Q est matrice formée par les vecteurs de la base vectorielle du sous-espace de Krylov). En conséquence, A et Q ne sont pas chargées intégralement en mémoire, ni même les blocs de données lors de la parallélisation. La méthode de Lanczos est constituée de nombreuses opérations linéaires très simples permettant l'emploi d'outils existant optimisés tels les BLAS. Enfin, il est intéressant de constater que le nombre d'itérations de la méthode de Lanczos est connu.

2.2.2 Parallélisation et répartition

Au sein de la tridiagonalisation de Lanczos, le travail de parallélisation concerne essentiellement les produits matrice-vecteur impliquant A et Q. Nous proposons un découpage

de A par blocs de lignes parmi les nœuds. Comme nous calculons progressivement Q au rythme d'une colonne par itération, nous faisons une distribution cyclique des colonnes. Sachant que nous travaillons à l'échelle d'Internet et que les performances de ce support sont faibles, nous ne pouvons envoyer à chaque produit matrice-vecteur les blocs de données. Nous proposons donc de faire de la persistance de données afin que les nœuds conservent les données qu'ils utilisent. Ainsi, A n'est distribuée qu'une seule fois à l'initialisation du problème et Q se construit peu à peu au fil des itérations. La répartition des produits matrice-vecteur génère beaucoup de communications. De plus, les résultats des produits étant exploités immédiatement, nous devons ajouter des points de synchronisation. Ces deux caractéristiques sont défavorables dans un contexte de calcul global. Il est conseillé de choisir un parallélisme à gros grain. Toutefois, nous devons prendre garde aux ressources limitées des nœuds de calcul. En choisissant un grain trop important, nous risquons de saturer la mémoire ou l'espace disque des nœuds. Nous proposons de résoudre le problème de limitation de mémoire en adoptant le principe de programmation "out-of-core". Les données sont stockées sur le disque et sont chargées en mémoire uniquement lorsqu'un calcul doit les utiliser. Nous faisons une programmation "out-of-core" à gros grain, de l'ordre de N éléments (une ligne de matrice). Pour la recherche des valeurs et vecteurs propres de la matrice tridiagonale T, nous exploitons le parallélisme paramétrique de la Bisection Parallèle. Si la matrice tridiagonale tient en mémoire sans trop de difficulté, les vecteurs propres ne le peuvent. Ainsi, nous utilisons également une technique "out-of-core" pour toute manipulation de ces vecteurs. Le calcul des vecteurs propres de Ritz, puis le calcul des résidus, consistent en la parallélisation de produits matrice-vecteur. Nous faisons donc usage de "l'out-of-core" lors des accès à A, Q et aux vecteurs de Ritz. Nous bénéficions à nouveau de la persistance de données afin de limiter le volume de données transférées. Ces deux phases terminales sont également coûteuses en terme de nombre de communications et sont suivies de points de synchronisation.

Les applications recherchant des couples de Ritz n'exploitent généralement qu'un nombre très limité $k(\ll N)$ de ces couples. Il n'est donc pas nécessaire de calculer les N valeurs et vecteurs de Ritz. Ainsi, nous proposons d'utiliser un schéma redémarré de la méthode numérique. Le Graphe 2.3 illustre son fonctionnement. Un nouveau paramètre est la dimension du sous-espace de travail m lors de la tridiagonalisation de Lanczos ($k < m \ll N$). Celle-ci ne possède plus que m itérations. Nous avons également $T \in M_m(\mathbb{R})$ et $Q \in M_{m,N}(\mathbb{R})$. La Bisection Parallèle permet alors de trouver m valeurs propres de Ritz "candidates" parmi lesquelles nous sélectionnons les k valeurs de plus grands modules. Nous pouvons ensuite calculer les k vecteurs propres de Ritz candidats et vérifier la convergence au moyen des normes euclidiennes des résidus. Dès qu'un couple ne vérifie pas la condition du test, nous redémarrons l'algorithme pour calculer d'autres couples de Ritz candidats. Au moment du redémarrage, nous devons tenir compte des calculs déjà réalisés, sans quoi il est impossible de converger. Pour y parvenir, nous modifions le vecteur initial de la tridiagonalisation de Lanczos : il est égal à la combinaison linéaire des vecteurs de Ritz candidats dont les coefficients sont les valeurs propres de Ritz candidates. L'intérêt majeur du schéma redémarré est de limiter considérablement le besoin en mémoire. La tridiagonalisation de Lanczos est nettement moins pénalisée par la phase de ré-orthogonalisation totale de Q (car la dimension de Q est réduite). La réduction de

la dimension de Q répond en partie à nos craintes sur la saturation de l'espace disque des nœuds de calcul. En outre, comme la phase de Bisection Parallèle est appliquée à une matrice de dimension m, les données et résultats du problème peuvent tenir en mémoire (si m est modéré). Le schéma redémarré ne garantie pas systématiquement un nombre d'opérations inférieur au schéma classique étant donné que le nombre de redémarrage peut être important ; ce dernier dépendant de k, de A, du vecteur initial, etc.

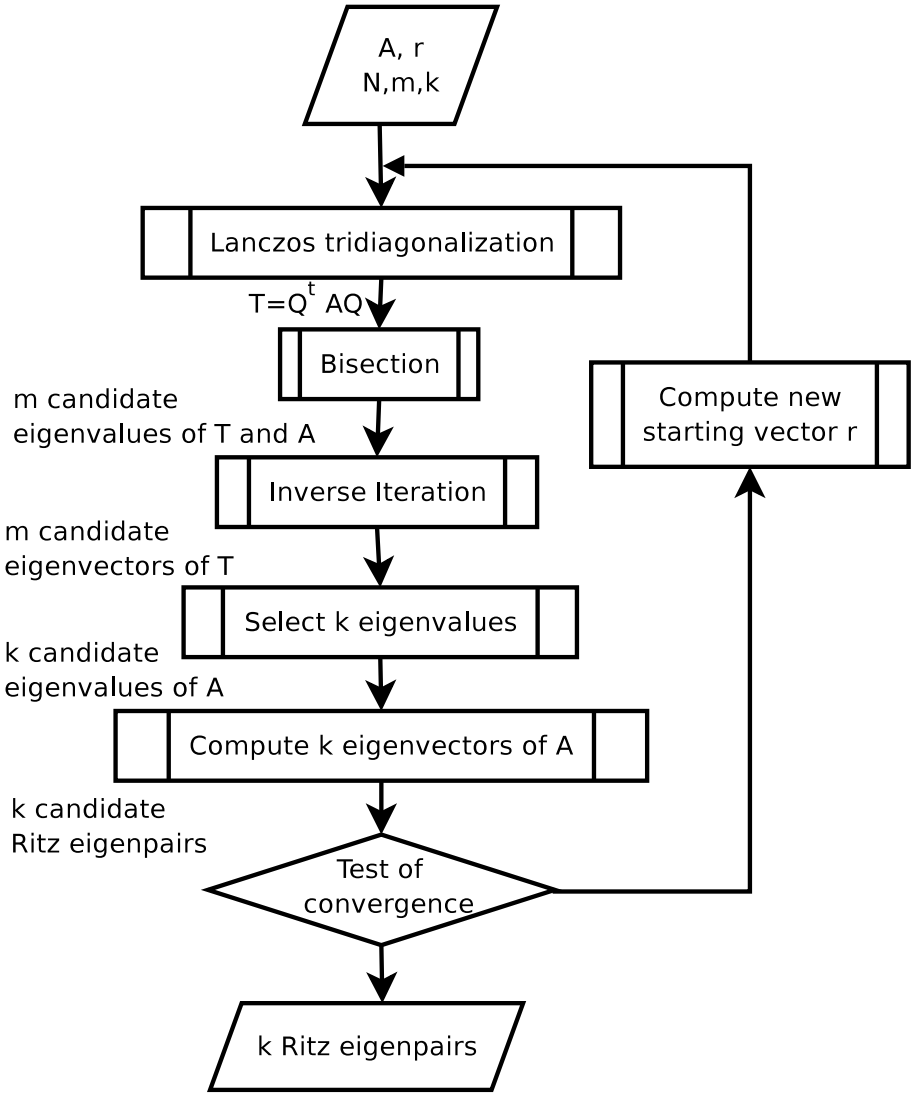

Fig. 2.3 – Graphe de tâches illustrant l'algorithme redémarré pour la recherche des éléments propres d'une matrice réelle symétrique

2.2.3 Conditions expérimentales

Actuellement, les logiciels de calcul global pair-à-pair ne permettent pas de répartir efficacement des méthodes d'algèbre linéaire très communicantes. Ainsi, nous utilisons uniquement le logiciel de calcul sur grille OmniRPC. Celui-ci permet de mettre en œuvre

aisément le mécanisme de persistance de données. Notre solveur requiert $8 * m * N$ octets en mémoire sur un nœud de calcul. La Table 2.4 donne le nombre de communications et la Table 2.5 présente leur volume. Les paramètres sont expliqués dans la Table 2.6.

	Nombre d'appels RPC asynchrones	Nombre de synchronisations
Distribution de A	p	$\lceil \frac{p}{f} \rceil$
Tridiagonalisation de Lanczos	si $m \leq p$, $i(m + mp + \frac{m(m-1)}{2})$ si $p < m$, $i(m + 2mp - \frac{p(p+1)}{2})$	$i(2m-1)$ $i(2m-1)$
Bisection et Itérations Inverses	is	i
Calculs des vecteurs de Ritz	$ik * max(m,p)$	i
Tests de convergence	si convergence, ikp si échec du test aux j^{th} éléments propres, ijp	i i

Tab. 2.4 – Nombre d'appels RPC asynchrones et de synchronisations du schéma redémarré

	Volume de données transferées (octets)
Distribution de A	$16pX$
Lanczos	si $m \leq p$, $8iN(m + 2m + \frac{p^2-1}{2})$ si $p < m$, $8iN(2m + 3mp - \frac{(p+1)^2}{2})$
Bisection et Itérations Inverses	$8m^2 + 24m$ (avec $S = 1$)
Calculs des vecteurs de Ritz	$16ik * max(m,p) * N$
Tests de convergence	si convergence, $16ikpN$ si échec du test aux j^{th} éléments propres, $16ijpN$

Tab. 2.5 – Volume des communications du schéma redémarré

Nous procédons à deux phases expérimentales bien distinctes. Une première série se fait dans un contexte de calcul global à l'échelle d'Internet. Nous étudions l'influence de la variation de certains paramètres tels m, k. Nous pouvons aussi évaluer en conditions réelles le coût des communications et de certaines phases de calcul de notre méthode. Les ressources de calcul et les réseaux sont inchangés par rapport à la section précédente. La seconde phase expérimentale exploite les ressources de Grille'5000. Grille'5000 [1] est un outil de calcul à grande échelle comprenant plusieurs grappes inter-connectées par un réseau rapide dédié. Nous utilisons les machines de quatre centres de calcul à Orsay, Lille, Sophia-Antipolis et Rennes. Ces machines sont des mono- ou bi-processeurs de la famille AMD Opteron (de 2.0 à 2.6GHz) et ont de 2GO à 4GO de mémoire. Notre motivation pour l'usage de Grille'5000 est double. Dans un premier temps, nous souhaitons isoler notre application des perturbations extérieures pour affiner l'analyse faite lors des tests sur des plateformes de calcul global. Dans un second temps, nous souhaitons accroître la dimension de la matrice et le nombre de nœuds exploités afin de tester, voire améliorer le passage à l'échelle de notre solveur.

[1] http ://www.grid5000.fr

N	Ordre de A
m	Dimension du sous-espace de Krylov
k	Nombre de couples de Ritz calculés
p	Nombre de nœuds de calcul
i	Nombre d'itérations du schéma redémarré
X	Nombre de composantes non nulles de A
f	Nombre maximum de fichiers ouverts en même temps lors de la distribution de A
s	Nombre de paquets de sous-intervalles

Tab. 2.6 – Rappel de paramètres

Pour les deux phases de tests nous utilisons la même matrice réelle symétrique de dimension $N = 47792$. Le vecteur initial est $r = (\frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}})$ avec $dim(r) = N$, $\|r\|_2 = 1$. Nous faisons varier les paramètres m et k. Pour la première phase, dans un contexte de calcul global, nous construisons quatre plateformes réparties sur l'USTL et l'Université Tsukuba et décrites dans la Table 2.7. Sur Grille'5000, nous créons également quatre plateformes similaires, ainsi qu'une cinquième pour affiner notre analyse (voir Table 2.8). Pour tester le passage à l'échelle de notre solveur sur Grille'5000, nous avons utilisé deux autres matrices réelles symétriques d'ordres $N = 203116$ et $N = 430128$ (respectivement 43 millions et 193 millions d'éléments non nuls). Les plateformes exploitées, présentées Tables 2.9 et 2.10, possèdent 206 et 412 nœuds de calcul.

	Config. 1	Config. 2	Config. 3	Config. 4
Emplacement du client	USTL			
Nombre et emplacements des serveurs OmniRPC	1 à Lille	2 à Lille	1 à Lille 1 à Tsukuba	2 à Lille 2 à Tsukuba
Nombre de workers par serveur	29 nœuds par serveur			
Nombre total de workers	29	58	58	116
Emplacements des workers	USTL		USTL et Université de Tsukuba	

Tab. 2.7 – Plateformes internationales pour la matrice d'ordre N=47792

Les temps obtenus sont exprimés en secondes. Ces temps, relevés chez le client, sont des temps horloge : ils incluent les temps de communication et les temps de calcul sur les nœuds distants. En ce qui concerne la première phase dans un contexte de calcul global, la Table 2.11 et la Figure 2.4 montrent les temps horloge pour trouver les k valeurs et vecteurs de Ritz. La Table 2.12 donne le nombre de redémarrages avant de converger. La Table 2.13 donne les ratios moyens de temps passé dans la phase de tridiagonalisation, dans la phase de calcul des vecteurs de Ritz et dans les tests de convergence par rapport aux temps totaux d'exécution. La Table 2.14 détaille les ratios moyens de temps passés au sein de la tridiagonalisation de Lanczos. Au sujet des tests utilisant Grille'5000, la Table 2.15 et la Figure 2.5 présentent les temps horloge pour résoudre le problème avec la matrice de dimension $N = 47792$. Les Tables 2.16, 2.17, 2.18 et 2.19 concernent les temps

	Config. 1	Config. 2	Config. 3	Config. 4	Config. 5
Emplacement du client	Orsay				
Nombre et emplacements des serveurs OmniRPC	1 à Orsay	1 à Orsay	1 à Orsay 1 à Lille	2 à Orsay 1 à Sophia	2 à Orsay 1 à Sophia 1 à Lille
Nombre de workers par server	29				
Nombre total de workers	29	58	58	116	116
Emplacements des workers	Orsay		Orsay Lille	Orsay Sophia	Orsay Sophia Lille

Tab. 2.8 – Plateformes experimentales sur Grille'5000 pour la matrice d'ordre N=47792

	Config. pour $N = 203116$
Emplacement du client	Orsay
Nombre et emplacements des serveurs OmniRPC	4 à Orsay 2 à Sophia 1 à Lille
Nombre total de workers	206

Tab. 2.9 – Plateforme experimentale sur Grille'5000 pour la matrice d'ordre N=203116

	Config. pour $N = 430128$	
Emplacement du client	Orsay	
Nombre et emplacements des serveurs OmniRPC	5 à Orsay 1 à Sophia 1 à Lille	8 à Orsay 1 à Sophia 1 à Lille 1 à Rennes
Nombre total de workers	206	412

Tab. 2.10 – Plateformes experimentales sur Grille'5000 pour la matrice d'ordre N=430128

horloge des expérimentations à plus grande échelle. Nous évaluons enfin le temps requit pour résoudre un unique produit matrice-vecteur dans les Tables 2.20 et 2.21.

m, k	Config. 1	Config. 2	Config. 3	Config. 4
10, 1	1358	1289	2895	6824
10, 2	2624	2515	5566	13045
10, 3	5051	4847	11067	24957
10, 4	10356	9853	23533	52993
15, 1	1248	1264	2795	6269
15, 2	2140	2155	4798	11137
15, 3	3068	2850	5866	13196
15, 4	8322	7765	17987	40558
20, 1	1190	1232	2556	5549
20, 2	1822	2112	4072	8529
20, 3	3065	2645	5557	11789
20, 4	6316	5875	13894	38339
25, 1	1765	1711	3369	7068
25, 2	2008	1582	3413	7673
25, 3	3341	2617	5343	11221
25, 4	4831	4563	11915	23524
Rappel des config.	OmniRPC : config. 1-2-3-4 Lille 29 workers : config. 1 Lille 58 workers : config. 2 Lille & Tsukuba 58 workers : config. 3 Lille & Tsukuba 116 workers : config. 4			

Tab. 2.11 – Temps horloge pour calculer k couples de Ritz (en secondes) sur les plateformes internationales - N=47792

2.2.4 Résumé des analyses

Nous soulignons l'importance du choix de l'algorithme et du paradigme de programmation parallèle employés pour résoudre notre problème d'algèbre linéaire. Les expérimentations confrontent au sein de la même application un modèle de parallélisme paramétrique et un autre type de parallélisme classique très communiquant avec des points de synchronisation. Ce dernier modèle souffre de l'impact des communications sur Internet, voire au sein d'un réseau local peu performant tel que celui de nos plateformes à l'USTL. La mise-en-œuvre d'un mécanisme de persistance de données pour réduire le volume des communications ne suffit pas à y remédier. Nous notons également que l'algorithme redémarré tend à synchroniser d'avantage l'exécution du programme. Le mécanisme de redémarrage est toutefois précieux si nous considérons le critère de passage à l'échelle car il réduit significativement les besoins en mémoire vive. Les expérimentations sur Grille'5000 montrent que nous pouvons traiter des problèmes d'ordres importants. Ce passage à l'échelle est également permis

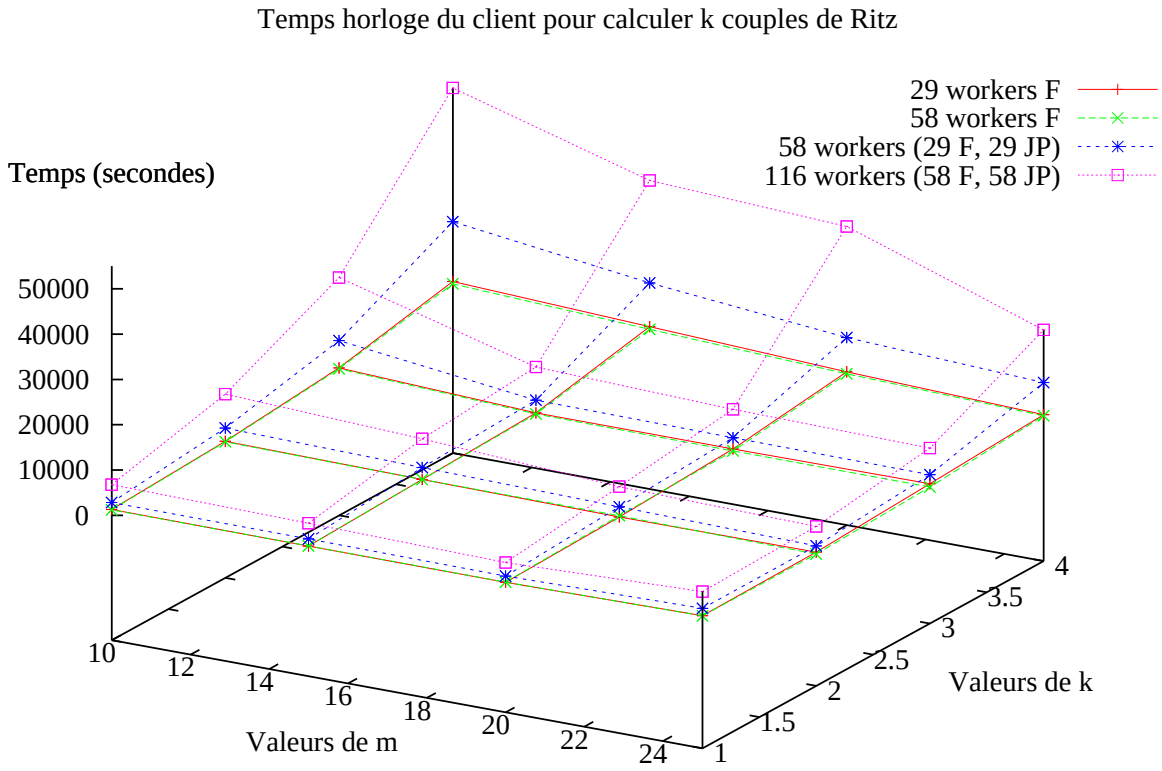

FIG. 2.4 – Temps horloge pour calculer k couples de Ritz (en secondes) sur les plateformes internationales - N=47792

m, k	nombre de redémarrages	m, k	nombre de redémarrages
25, 1	2	15, 1	3
25, 2	2	15, 2	5
25, 3	3	15, 3	6
25, 4	6	15, 4	18
20, 1	2	10, 1	5
20, 2	3	10, 2	9
20, 3	4	10, 3	17
20, 4	10	10, 4	36

TAB. 2.12 – Nombre de redémarrages pour calculer k couples de Ritz sur les plateformes internationales - N=47792

	Config. 2 Recherche des k couples de Ritz = 100%		
m, k	Tridiagonalisation de Lanczos	Recherche des vecteurs de Ritz	Tests de convergence
10, 1	90%	1%	7%
10, 2	87%	2%	9%
10, 3	86%	3%	10%
10, 4	86%	4%	8%
15, 1	93%	1%	5%
15, 2	90%	2%	7%
15, 3	87%	2%	9%
15, 4	87%	4%	8%
20, 1	95%	1%	3%
20, 2	92%	1%	5%
20, 3	89%	2%	7%
20, 4	88%	4%	8%
25, 1	95%	1%	2%
25, 2	93%	2%	4%
25, 3	91%	3%	5%
25, 4	88%	4%	7%

Tab. 2.13 – Proportion de temps écoulé (en %) dans la tridiagonalisation de Lanczos, le calcul des vecteurs propres de Ritz et les tests de convergence par rapport au problème complet de recherche des couples de Ritz

	Config. 2 Tridiagonalisation de Lanczos = 100%	
m, k	Produits Matrice-Vecteur	Réorthogonolisations
10, 1	88%	10%
10, 2	88%	10%
10, 3	90%	8%
10, 4	88%	9%
15, 1	86%	12%
15, 2	86%	12%
15, 3	86%	12%
15, 4	85%	13%
20, 1	80%	18%
20, 2	86%	12%
20, 3	83%	15%
20, 4	79%	18%
25, 1	78%	20%
25, 2	77%	20%
25, 3	76%	22%
25, 4	77%	22%

Tab. 2.14 – Proportion de temps écoulé (en %) dans les produits Matrice-Vecteur impliquant A et les ré-orthogonalisations par rapport à la tridiagonalisation de Lanczos

m, k	Config. 1	Config. 2	Config. 3	Config. 4	Config. 5
10, 1	190	93	147	431	162
10, 2	320	167	241	706	266
10, 3	616	309	465	1376	512
10, 4	1477	761	1136	3501	1258
15, 1	173	90	135	387	149
15, 2	298	148	233	661	260
15, 3	307	163	239	675	266
15, 4	1072	542	826	2500	922
20, 1	157	81	122	344	142
20, 2	247	124	186	524	217
20, 3	253	125	188	538	217
20, 4	778	392	594	1770	672
25, 1	196	102	157	437	181
25, 2	317	160	241	669	282
25, 3	320	161	244	677	283
25, 4	664	329	506	1465	580

Tab. 2.15 – Temps horloge pour calculer k couples de Ritz (en secondes) sur les plateformes de Grille'5000 - N=47792

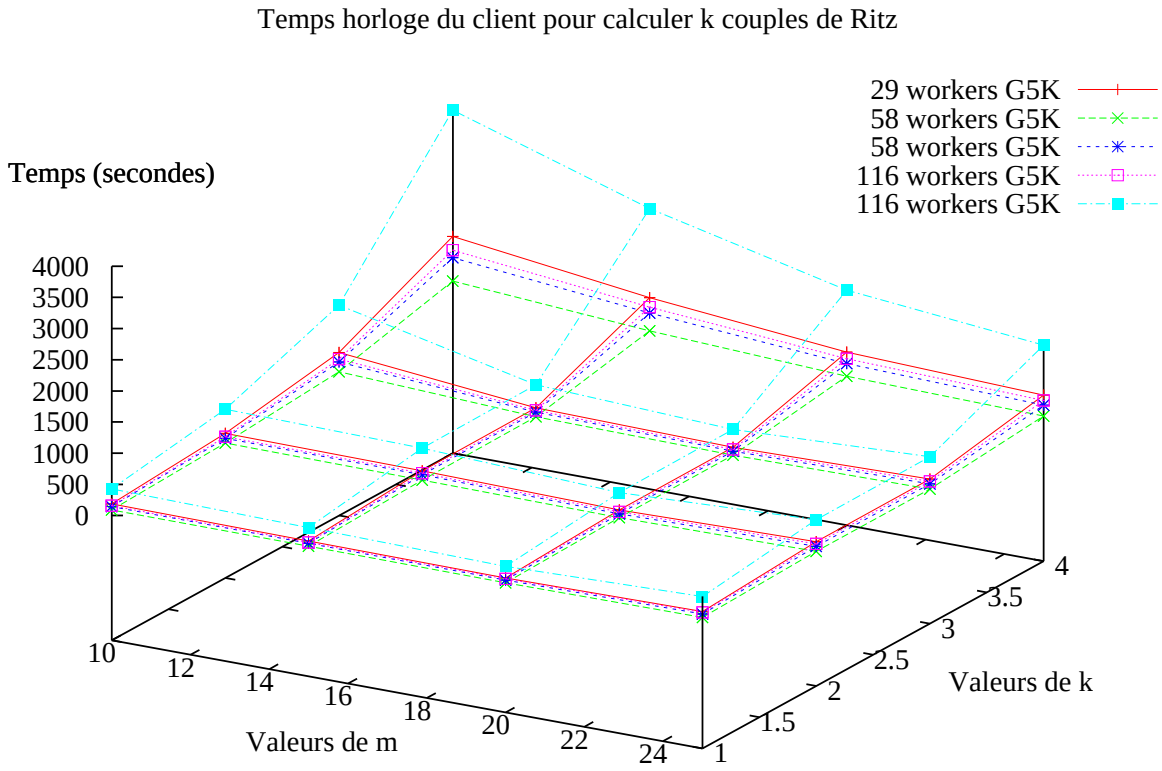

Fig. 2.5 – Temps horloge pour calculer k couples de Ritz (en secondes) sur les plateformes de Grille'5000 - N=47792

(m,k)	Temps horloge avec 206 workers (3 sites)
10, 1	2809
10, 2	2768
10, 3	4878
10, 4	6103

Tab. 2.16 – Temps horloge pour calculer k couples de Ritz (en secondes) sur Grille'5000 - N=203116

Nombre de nœuds	Temps horloge pour (m,k)=(15,1)
206 (3 sites)	10962
412 (4 sites)	13150

Tab. 2.17 – Temps horloge pour calculer k couples de Ritz (en secondes) sur Grille'5000 - N=430128

Détails des temps horloge des tests avec 206 nœuds sur 3 sites	
Total de la recherche des éléments propres	10962
Lanczos - distribution de la nouvelle colonne de Q	22
Lanczos - Produit Matrice-Vecteur impliquant A	10106
Lanczos - Ré-orthogonalisation	129
Bisection et Itérations Inverses	1
Calculs des vecteurs de Ritz	9
Contrôles des normes des résidus	691

Tab. 2.18 – Détails des temps horloge pour les expérimentations sur Grille'5000 - N=430128 et 206 nœuds

Détails des temps horloge des tests avec 412 nœuds	
Total de la recherche des éléments propres	13150
Lanczos - distribution de la nouvelle colonne de Q	20
Lanczos - Produit Matrice-Vecteur impliquant A	12311
Lanczos - Ré-orthogonalisation	159
Bisection et Itérations Inverses	9
Calculs des vecteurs de Ritz	11
Contrôles des normes des résidus	810

Tab. 2.19 – Détails des temps horloge pour les expérimentations sur Grille'5000 - N=430128 et 412 nœuds

	Nombre de Produits Matrice-Vecteur avec A	Temps horloge par produit
Lanczos - Produits Matrice-Vecteur	75	134
Contrôles des normes des résidus	5	138

Tab. 2.20 – Temps horloge (en secondes) pour un Produits Matrice-Vecteur impliquant A - N=430128, 206 workers

	Nombre de Produits Matrice-Vecteur avec A	Temps horloge par produit
Lanczos - Produits Matrice-Vecteur	75	164
Contrôles des normes des résidus	5	162

Tab. 2.21 – Temps horloge (en secondes) pour un Produits Matrice-Vecteur impliquant A - N=430128, 412 workers

en faisant de la programmation "out-of-core". Nous constatons la nécessité de cette technique lors des produits matrice-vecteur. A l'instar du schéma redémarré, "l'out-of-core" améliore le passage à l'échelle mais pénalise fortement l'application en terme de rapidité de calcul en raison des multiples accès aux disques. Par exemple, cette technique explique en partie les temps de calcul relativement long des produits matrice-vecteur impliquant A.

Selon la dimension de la matrice A et le nombre k de valeurs et vecteurs de Ritz recherchés, nous soulignons l'importance du choix de m (dimension du sous-espace de Krylov et de la matrice T calculés par la tridiagonalisation de Lanczos). Une trop faible valeur de m par rapport à k n'apporte pas assez d'informations et engendre de nombreux redémarrages. Une trop forte valeur ajoute des opérations inutiles dans la phase de tridiagonalisation. Les expérimentations révèlent aussi l'impact du grain de parallélisme choisit et du choix de la plateforme exploitée. En effet, en raison des faibles performances d'Internet, voire des réseaux locaux, nous ne bénéficions pas automatiquement d'une réduction du temps total d'exécution en diminuant le grain de parallélisme (et en partageant calculs et données entre un plus grand nombre de nœuds). Même en utilisant un réseau dédié relativement rapide tel celui de Grille'5000, nous constatons qu'un grain de parallélisme trop fin engendre trop de communications par rapport à des temps de calcul très faibles. Enfin l'impact de la configuration du logiciel de calcul global pour les ressources de la plateforme peut être considérable. Les tests soulignent en particulier les performances décevantes obtenues avec les configurations utilisées lorsque OmniRPC utilise tous les processeurs d'un multiprocesseur.

Au terme de notre étude, nous constatons que le prochain facteur limitant, quand au passage à l'échelle de notre solveur, est l'espace disque occupé par les fichiers de données (principalement pour A). Nous pourrions y remédier en utilisant un format compact d'écriture de données dans les fichiers mais cela risquerait d'accroître significativement les temps de chargement de données. Un second point à améliorer serait la distribution initiale de la matrice A parmi les nœuds de calcul.

Chapitre 3

Vers la mise-en-œuvre de plateformes hétérogènes à faible consommation d'énergie

3.1 Proposition

Dans le domaine du calcul haute performance sur grappe de PCs, [104] propose d'exploiter le "slack-time" de chaque nœud généré par un déséquilibre de la charge de travail en faisant varier dynamiquement la fréquence des processeurs des nœuds. La variation de la fréquence d'un processeur se fait en modifiant la tension d'alimentation de celui-ci, d'où la dénomination DVS (Dynamic Voltage Scaling). Le calcul global est également sujet à bien d'autres sources de "slack-time" telles l'hétérogénéité des nœuds de calcul et les communications au moyen de réseaux lents et non dédiés. Nous proposons d'évaluer les bénéfices possibles si nous exploitons l'hétérogénéité des fréquences CPU des nœuds de calcul au moyen d'un mécanisme de DVS. Comme support de notre étude, nous adaptons notre application de recherche des éléments propres d'une matrice réelle symétrique afin que nous puissions faire varier dynamiquement la fréquence des nœuds en divers points de l'algorithme.

3.2 Plateforme expérimentale

Bien que nous visions l'usage de ces mécanismes à faible consommation d'énergie sur des plateformes de calcul global, nous devons effectuer notre étude sur une grappe de PCs au laboratoire HPCS de l'Université de Tsukuba. Il n'existe pas à notre connaissance de plateforme à très grande échelle permettant de faire du DVS et de mesurer avec précision la

consommation énergétique de chaque nœud. Toutefois, il est concevable de se servir d'une grappe de taille réduite pour notre étude. En effet, nous étudions uniquement le "slack-time" résultant de l'hétérogénéité des nœuds, et non celui dû aux communications sur les réseaux peu performants. De plus notre application recherchant les éléments propres d'une matrice réelle symétrique a été adaptée dans l'optique d'un déploiement sur grille de grappes.

Chaque machine de la grappe du laboratoire HPCS de l'Université de Tsukuba utilise des composants [114] permettant de relever la puissance consommée sans intrusion dans l'architecture matérielle de la machine. Ainsi aucune influence extérieure ne perturbe les mesures des puissances consommées. La grappe possède 16 nœuds dont les détails sont donnés dans la Table 3.1. La modification de la fréquence d'un processeur est faite au moyen des instructions *PowerNow!*. Les fréquences disponibles sont données Table 3.2.

CPU	AMD Opteron
Horloge (max.)	2200MHz
Cache L1/L2	128KB/1MB
Memoire	1GB (DDR)
Réseau	1 GB Ethernet
OS Linux	Red Hat, kernel 2.6.11
gcc	4.1.2
MPI	LAM/MPI v7.1.3

Tab. 3.1 – Caractéristiques des nœuds de la grappe

Ratio (%)	Fréquence (MHz)
100	2200
90	2000
80	1800
60	1600
40	1400
20	1200
0	1000

Tab. 3.2 – Table des fréquences

3.3 Un solveur pour la recherche des éléments propres d'une matrice intégrant des mécanismes de DVS

Nous avons adapté le solveur précédemment développé pour le déploiement sur des plateformes de calcul global. Il n'y a pas de modification notable d'un point de vue algorithmique, ni pour le graphe d'exécution. La distribution des données est inchangée. Sur

les 16 nœuds de la grappe, nous considérons 1 client et 15 nœuds de calcul pour lesquels nous appliquons le DVS. A l'initialisation du solveur, tous les nœuds initialisent la mesure de puissance électrique et choisissent une fréquence "défaut". Ensuite, à divers points du graphe d'exécution, nous disposons des appels de fonctions *PowerNow!* pour modifier la fréquence CPU. Ces appels sont placés en début et fin du solveur, en amont et en aval des routines d'attente et de communications bloquantes (MPI_Wait, MPI_Gather, MPI_Bcast). Les valeurs des fréquences que nous passons en paramètres des fonctions varient suivant les objectifs des recherches menées.

3.4 Expérimentations et analyses

3.4.1 Données numériques du problème

Nous utilisons une matrice de dimension $N = 32490$ qui est un pavage de la matrice *bcsstk*09 issue de la collection de matrices Harwell-Boeing du site Internet Matrix-Market[1]. Elle a 16.5 millions de composantes non nulles, soit une moyenne de 510 éléments non nuls par ligne. Le but de cette étude ne concernant pas l'impact des paramètres du solveur, nous utilisons un unique couple $(m, k) = (20, 2)$. Autrement dit, nous calculons 2 couples de Ritz et nous utilisons un sous-espace de Krylov de dimension 20 dans la phase de tridiagonalisation de Lanczos.

3.4.2 Utilisation de mécanismes de DVS durant les communications bloquantes et les temps d'attente des processus

Nous étudions l'usage de mécanismes de DVS lors des communications et des temps d'attente des processus. La Table 3.3 montre les fréquences initiales des processeurs pour trois configurations d'hétérogénéité. Quand nous faisons appel aux routines de DVS, nous abaissons la fréquence des processeurs des nœuds de calcul au niveau de la fréquence du processeur le plus lent. (0% pour la configuration B, 40% pour la configuration C).

	Fréq. CPU (%) du client	Fréq. CPU par défaut (%) des 15 workers
Configuration A	100	15 à 100
Configuration B	100	5 à 100, 1 à 80, 2 à 60, 2 à 40, 1 à 20, 4 à 0
Configuration C	100	5 à 100, 1 à 80, 2 à 60, 7 à 40

TAB. 3.3 – Configurations des plateformes hétérogènes

[1] http ://math.nist.gov/MatrixMarket/

Les résultats sont donnés Table 3.4. Nous présentons les temps horloge et les consommations globales d'énergie (exprimés respectivement en secondes et joules). Nous donnons aussi plusieurs ratios : comparaisons des données des configurations B puis C avec la configuration A pour une stratégie de DVS donnée, comparaisons des données des tests utilisant une stratégie de DVS avec les tests sans appel de routine DVS pour la même configuration. En observant les 5 premières colonnes, nous constatons que le temps horloge d'exécution augmente significativement lorsque l'hétérogénéité des ressources de calcul augmente. En effet, notre solveur est très sensible à l'hétérogénéité en raison du grand nombre de communications et points de synchronisation. Ainsi, le temps horloge d'exécution est directement lié aux performances du nœud de calcul le plus lent. Plus le temps de calcul est important, plus la consommation énergétique est grande. Les 2 dernières colonnes de la Table 3.4 montre que l'usage de mécanisme DVS uniquement lors des communications et temps d'attente n'engendre pas une hausse significative du temps horloge d'exécution mais n'entraîne pas non plus une réduction de la consommation énergétique. Une première explication concerne le choix de notre application. En effet, en raison de la persistance de données sur les nœuds, le volume des communications est faible. De plus le réseau local est dédié (bande passante et latence constantes) et notre application est fortement synchronisée. Ainsi, il y a beaucoup de brèves communications entre des processus relativement synchronisés et ayant peu de temps d'attente. De plus, le changement de fréquence d'un processeur n'est pas instantané mais requiert par exemple $50\mu s$ pour passer du niveau 100% au niveau 10 % (cf. Table 3.2).

Configuration - Stratégie de DVS	Temps (sec)	Consommation énergétique (W.s)	Temps (%) comparé à config. A	Energie (%) comparé à config. A	Temps (%) comparé au non usage de DVS	Energie (%) comparé au non usage de DVS
A - pas de DVS	490	773041	-	-	-	-
B - pas de DVS	897	1176348	+83	+52	-	-
C - pas de DVS	684	943448	+39	+22	-	-
A - DVS et bcast	491	771671	-	-	$+ < 1$	$- < 1$
B - DVS et bcast	897	1169854	+82	+51	$+ < 1$	$- < 1$
C - DVS et bcast	682	936067	+38	+21	$- < 1$	$- < 1$
A - DVS et gather	492	775315	-	-	$+ < 1$	$+ < 1$
B - DVS et gather	897	1174100	+82	+51	$+ < 1$	$- < 1$
C - DVS et gather	681	936363	+38	+20	$- < 1$	$- < 1$
A - DVS et wait	491	772396	-	-	$+ < 1$	$- < 1$
B - DVS et wait	903	1173586	+83	+51	$+ < 1$	$- < 1$
C - DVS et wait	688	942467	+40	+22	$+ < 1$	$- < 1$

Tab. 3.4 – Plateformes hétérogènes et DVS durant les communications bloquantes et les temps d'attente

3.4.3 Impact de l'ajustement permanent des fréquences de tous les processeurs à la fréquence du nœud le plus lent

Dans cette section, nous considérons que le client et 14 nœuds de calcul utilisent 100% de leur fréquence CPU. Un seul nœud voit sa fréquence initiale abaissée. Nous avons donc six configurations suivant que ce nœud ait sa fréquence initiale aux niveaux 0, 20, 40, 60, 80 ou 90%. Les Tables 3.5 et 3.6 présentent nos résultats. La première table donne les temps horloge (secondes), la consommation globale d'énergie (Joules) et la puissance moyenne consommée par nœud (Watts). La seconde table présente divers ratios : comparaison du temps horloge, de la consommation globale d'énergie et de la puissance moyenne consommée par nœud, avec la configuration où tous les nœuds utilisent 100% du CPU (première ligne) ; comparaison de ces mêmes données entre les configurations ayant un seul nœud à fréquence réduite et les configurations correspondantes où tous les nœuds ont une fréquence ajustée au niveau du nœud le plus lent.

Fréquences CPU (%) des 15 workers	Temps (sec)	Consommation énergétique (W.s)	Puissance consommée par nœud (W)
15 à 100	490	773041	98
1 nœud lent à 90, 14 à 100	514	802524	97
15 à 90	528	796492	93
1 nœud lent à 80, 14 à 100	552	843519	95
15 à 80	568	818277	89
1 nœud lent à 60, 14 à 100	601	899867	93
15 à 60	619	853515	85
1 nœud lent à 40, 14 à 100	680	1000250	91
15 à 40	689	914133	82
1 nœud lent à 20, 14 à 100	758	1085626	89
15 à 20	776	992403	79
1 nœud lent à 0, 14 à 100	882	1227917	86
15 à 0	903	1114730	76

Tab. 3.5 – Plateformes hétérogènes et DVS durant toute l'exécution

Tout d'abord, nous confirmons l'analyse précédente car un seul nœud hétérogène suffit à accroître considérablement le temps horloge et donc la consommation énergétique (respectivement jusqu'à +80% et +58% par rapport à la configuration homogène où tous les nœuds sont à 100%).

Lorsque nous nivelons toutes les fréquences des processeurs des nœuds de calcul au niveau du nœud le plus lent dès le début de l'exécution et pour toute la durée des calculs, le temps horloge ne croît que de 1 à 2% alors que la consommation globale d'énergie diminue jusqu'à 9%. Les économies locales sont également très intéressantes. En effet, les puissances consommées par nœud indiquées dans la colonne 4 de la Table 3.5 sont trompeuses car elles sont influencées par la consommation du nœud le plus lent. Ainsi,

Freq. CPU (%) des 15 workers	Variations par rapport à config. homogène à 100%			Variations par rapport à config. avec un nœud lent		
	temps horloge (%)	conso. énergie (%)	puissance consommée par nœud (%)	temps horloge (%)	conso. énergie (%)	puissance consommée par nœud (%)
15 à 100	-	-	-	-	-	-
1 à 90, 14 à 100	+4	+3	< -1	-	-	-
15 à 90	+7	+3	−5	+2	< 1	−4
1 à 80, 14 à 100	+12	+9	−3	-	-	-
15 à 80	+15	+5	−9	+2	−3	−6
1 à 60, 14 à 100	+22	+16	−5	-	-	-
15 à 60	+26	+10	−13	+2	−5	−8
1 à 40, 14 à 100	+38	+29	−7	-	-	-
15 à 40	+40	+18	−16	+1	−8	−9
1 à 20, 14 à 100	+54	+40	−9	-	-	-
15 à 20	+58	+28	−19	+2	−8	−11
1 à 0, 14 à 100	+80	+58	−12	-	-	-
15 à 0	+84	+44	−22	+2	−9	−11

Tab. 3.6 – Plateformes hétérogènes et DVS durant toute l'exécution

Baisse de la fréquence de 100% à	Réduction de la consommation énergétique locale
90	2
80	6
60	10
40	15
20	17
0	20

Tab. 3.7 – Réduction de la consommation énergétique locale en nivelant les fréquences CPU

bien plus qu'une réduction de la puissance consommée allant de 4% à 11% à première vue, nous obtenons en réalité une réduction variant de 5% à 22%. Puisque le temps horloge est seulement augmenté de 1 ou 2%, la consommation énergétique locale des 14 nœuds est fortement réduite. Par exemple, pour une exécution du solveur recherchant 2 couples de Ritz, un nœud fonctionnant initialement à 100% de sa fréquence CPU économise 17% d'énergie si l'on ajuste sa fréquence au niveau du nœud le plus lent fonctionnant à 20%. La Table 3.7 présente les économies d'énergie qu'un nœud peut réaliser dans le cadre d'une exécution du solveur avec les paramètres indiqués précédemment.

3.5 Conclusion et perspectives

Cette étude présente les bénéfices en terme d'économie d'énergie que l'on peut tirer si l'on exploite l'hétérogénéité des fréquences CPU des nœuds de calcul. Nos résultats sont fortement liés à l'application choisie et concernent en général les applications fortement communicantes et synchronisées. Nous avons montré que l'usage de DVS uniquement pendant les périodes de "slack-time" n'apporte pas de bénéfices significatifs. En revanche, si l'on ajuste la fréquence des processeurs convenablement sur l'ensemble de l'exécution, nous obtenons une réduction de 9% de la consommation énergétique globale et une réduction jusqu'à 20% localement sur un nœud sans augmentation significative du temps de résolution du problème.

Il serait ensuite des plus intéressants d'étudier les gains énergétiques si nous exploitions l'hétérogénéité d'autres composants telle la vitesse d'accès disque. Comme nous utilisons un mécanisme "out-of-core", ce facteur peut fortement influer. Nous souhaiterions étendre ce travail en évaluant les possibles réductions de consommation si nous exploitions aussi le "slack-time" résultant des faibles performances réseaux. L'usage d'un émulateur tel Modelnet permettrait de réaliser cette étude sur la même grappe d'ordinateur.

Chapitre 4

YML, un environnement d'aide au développement d'applications à très grande échelle

4.1 Présentation et objectifs d'YML

La conception d'une application parallèle et distribuée est une tâche difficile qui est souvent rendue plus ardue en raison de complications extérieures propres aux intergiciels, aux ressources de calcul, aux réseaux, etc. L'objectif d'YML est de masquer toutes ces difficultés additionnelles. De plus, de nombreux intergiciels ont été développés et se partagent les ressources de calcul disponibles. Si le client d'une application souhaite exploiter un nombre maximum de nœuds, il doit donc développer son application pour chacune des interfaces de programmation des intergiciels. Ce travail est fastidieux, voire impossible. YML fournit au client une unique interface YvetteML et utilise un mécanisme de "back-end" pour exploiter différents intergiciels. Le langage YvetteML permet au client de construire intuitivement un graphe de tâches de calcul. Chaque tâche de calcul est représentée sous forme d'un composant facilement réutilisable. L'approche composant aide le client à clairement distinguer les communications des phases de calcul autonomes.

4.2 Contributions

YML est principalement développé au laboratoire PRiSM de l'Université de Versailles Saint-Quentin-en-Yvelines. Ma contribution à YML est double. En premier lieu, je développe des modules du logiciel permettant d'améliorer ou étendre les possibilités d'YML. Dans un second temps, j'adapte le problème de recherche des valeurs et vecteurs propres

pour YvetteML. Ce dernier point permet aussi de tester les dernières versions d'YML et d'apporter un avis d'utilisateur indispensable afin de signaler les problèmes rencontrés ou de proposer des fonctionnalités utiles. L'étude de ce problème d'algèbre linéaire permet également d'unifier mon travail de développement de modules d'YML avec celui des sections précédentes.

4.3 Développement de modules d'YML

Notre travail a débuté avec le développement d'un "back-end" permettant à YML d'exploiter les plateformes gérées par l'intergiciel OmniRPC. Ensuite, nous avons mis-en-œuvre le "Worker YML" et le serveur de données. Ce dernier conserve les paramètres initiaux, les exécutables et les résultats. Le "Worker YML" gère l'exécution du programme sur le nœud distant : téléchargement des paramètres et de l'exécutable auprès du serveur de données, lancement de l'exécutable, dépôt des résultats. La Figure 4.1 montre les principales interactions entre ces trois modules. En réalité, le serveur de données gère un ensemble de processus légers de manière à supporter des requêtes simultanées provenant de "Worker YML". Enfin, nous avons participé au développement du support multi "back-ends" qui devra prochainement permettre à YML d'utiliser dynamiquement plusieurs "back-ends".

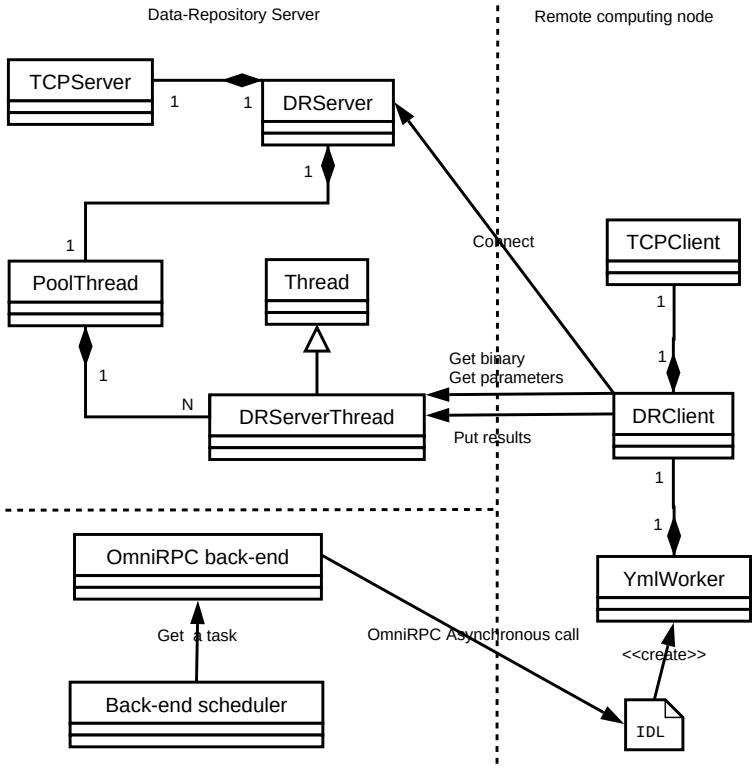

FIG. 4.1 – Intéractions entre le back-end OmniRPC, le worker YML et le DR Serveur

4.4 Etude d'un problème d'algèbre linéaire pour YML

Le développement d'un premier algorithme relativement complexe pour YML a révélé le manque de certaines fonctionnalités qui ont alors été apportées par Olivier Delannoy. En particulier, YML accepte désormais des paramètres en entrée et en sortie sous forme de fichiers. Cela nous est indispensable pour des données comme les matrices et vecteurs. En outre, YML supporte maintenant les collections de données d'un même type, puis la manipulation et l'indexation des éléments d'une collection à l'image des langages traditionnels de programmation.

L'adaptation pour YvetteML du problème de recherche des valeurs et vecteurs propres d'une matrice réelle symétrique ne nécessite pas de changement majeur d'un point de vue algorithmique par rapport à notre travail directement sur OmniRPC. En revanche, nous notons des différences significatives d'implémentations. En effet, la persistance de données n'est pas encore gérée par le "back-end" OmniRPC, ni même YvetteML. Il faut donc à chaque fois envoyer l'intégralité des données nécessaires aux calculs. Ainsi, bien que les hautes performances ne soient pas un objectif prioritaire d'YML, la distribution d'un calcul sur Internet doit être sérieusement étudiée en raison du coût des communications. Si un nœud dispose des ressources matérielles et peut recevoir un volume important de données pour traiter lui-même séquentiellement une tâche, alors il est sûrement préférable de lui attribuer plutôt que de la distribuer. Cette tâche ne peut malheureusement pas être faite par le client car YvetteML ne supporte pas la moindre opération numérique. Cette dernière limitation est également l'objet d'une réflexion. D'une part, nous pouvons agglomérer un maximum d'opérations au sein d'un même composant afin de limiter le nombre de composants et donc le nombre de communications. Mais ceci ce fait au détriment du principe de ré-utilisabilité qui est un point essentiel d'YML. D'autre part, nous pouvons attribuer un composant à chaque étape de calcul mais cela engendre un nombre important de communications. Nous avons choisi cette dernière option afin de favoriser la ré-utilisabilité de nos composants (ex. pour le développement de la méthode d'Arnoldi) bien qu'elle ne soit pas optimale en terme de nombre et volume de communications sur Internet.

Nous proposons deux graphes de tâches orchestrant nos composants illustrés par les Figures 6.3 et 6.4 aux pages 138 et 139.

4.4.1 Conditions expérimentales

Les toutes premières expérimentations que nous avons menées, ont pour objectif d'observer le surcoût de l'usage combiné d'YML et du "back-end" OmniRPC par rapport à une simple exécution au moyen d'OmniRPC. Nous présentons les tests faits avec le graphe de tâches de la Figure 6.3 qui permet une comparaison avec les tests de la Partie I Section 2.2.

Nous exploitons les ressources de calcul et les réseaux locaux de l'Université de Lille 1 au moyen d'OmniRPC et du "back-end" d'YML associé. L'ordonnanceur d'YML et le serveur de données sont localisés au sein du même LAN. Les détails sont donnés dans la Table 4.1. La matrice d'ordre $N = 47792$ est identique à celle utilisée précédemment. Les paramètres variables sont à nouveau m et k (respectivement la dimension du sous-espace de Krylov et le nombre de couples de Ritz calculés).

	Configuration 1	Configuration 2
Intergiciel	OmniRPC	
Emplacement du client Emplacement des serveurs OmniRPC Emplacement des workers	Lille Lille Lille	
Nombre de serveurs OmniRPC	1	2
Nombre de workers par serveur	29	
Nombre total de workers	29	58
Nombre de blocs de A	29	58

TAB. 4.1 – Platformes des tests préliminaires d'YML

4.4.2 Résumé des analyses

La Table 4.2 présente les temps horloges relevés chez le client pour résoudre le problème. La Table 4.3 est un rappel des valeurs obtenues avec des expérimentations similaires en utilisant directement OmniRPC. Enfin, la Table 4.4 explicite les rapports des temps entre les deux précédentes tables.

m, k	Configuration 1	Configuration 2
15, 1	221	209
15, 2	323	236
15, 3	462	296
20, 1	197	146
20, 2	321	232
20, 3	368	297
25, 1	257	176
25, 2	267	186
25, 3	281	195

TAB. 4.2 – Temps horloge du client pour obtenir k couples de Ritz avec YML (en minutes)

Les temps horloge relevés chez le client sont de 4 à 11 fois plus long en utilisant YML. Les ressources de calcul et réseau sont relativement comparables (et les tests sont faits la nuit dans les deux cas). De plus, le graphe YvetteML est très proche du schéma d'exécution directement avec OmniRPC. Nous distinguons plusieurs explications à un tel surcoût. Tout d'abord, le client YML ne peut pas réaliser de simples calculs au sein du code

m, k	29 workers à Lille	58 workers à Lille
15, 1	20	21
15, 2	35	35
15, 3	51	47
20, 1	19	20
20, 2	30	35
20, 3	51	44
25, 1	29	28
25, 2	33	26
25, 3	55	43

Tab. 4.3 – Temps horloge du client pour obtenir k couples de Ritz directement avec OmniRPC (en minutes)

m, k	29 workers à Lille	58 workers à Lille
15, 1	11	9
15, 2	9	6
15, 3	9	6
20, 1	10	7
20, 2	10	6
20, 3	7	6
25, 1	8	6
25, 2	8	7
25, 3	5	4

Tab. 4.4 – Surcoût d'YML (avec le back-end OmniRPC) par rapport à un usage direct d'OmniRPC

YvetteML mais il doit les faire dans des composants qui sont répartis. Si le calcul est très rapide, le gain en temps de calcul est négligeable, voire nul, alors que le surcoût en temps de communication est évident. Ensuite, nous considérons le surcoût directement imputable au logiciel YML qui est la conjonction de plusieurs points. L'ordonnanceur d'YML met un temps non négligeable pour sélectionner la tâche suivante à exécuter. Le procédé d'empaquetage/dépaquetage des paramètres est également très coûteux. Enfin, la raison principale est le transfert systématique d'un binaire de composant vers un nœud de calcul (i.e. à chaque invocation dans le graphe YvetteML). Au contraire, lors d'une exécution directe sur une plateforme OmniRPC, les squelettes (RPC) sont pré-enregistrés et plus aucun transfert n'est réalisé.

4.5 Conclusion et perspectives

YML est un logiciel encore jeune et développé par une équipe très réduite. Il possède donc les défauts de sa jeunesse à savoir qu'il ne masque pas encore totalement l'hétérogénéité des plateformes de calcul car les "back-ends" ne peuvent pas encore être tous exploités dynamiquement ensemble. De plus, afin de réduire le surcoût d'YML, de nombreuses optimisations restent à apporter tel un mécanisme de cache pour les binaires de composants sur les nœuds de calcul. Néanmoins, nous pouvons déjà apprécier le langage de haut niveau intuitif qu'est YvetteML afin de composer un graphe d'exécution en se concentrant essentiellement sur des détails algorithmiques et de répartition. Nous avons également bénéficié du principe de programmation par composants en réutilisant plusieurs fois certains d'entre eux.

A court terme, nous devrons réaliser de nombreux tests pour tester le passage à l'échelle d'YML. Nous augmenterons à la fois le nombre de nœuds exploités et la dimension de la matrice. Nous devrons aussi tester ce même graphe YML avec le "back-end" XtremWeb puis réaliser des expérimentations similaires avec le graphe de la Figure 6.4.

Nous souhaiterions poursuivre notre contribution au développement d'YML, en particulier pour la mise-en-œuvre du support dynamique "multi back-ends". Cela concerne l'ajout d'un système de collecte et exploitation d'informations sur les ressources de calcul, les réseaux et les exécutions passées, ainsi qu'un nouvel ordonnanceur capable d'exploiter ces informations pour attribuer une tâche au "back-end" adéquat.

Chapitre 5

Conclusion générale

L'axe principal de cette thèse de doctorat est l'étude de paradigmes et techniques de programmation pour la distribution et la parallélisation de méthodes numériques sur les plateformes de calcul global. Nous insistons sur l'importance des choix numériques permettant de choisir un paradigme de programmation optimal pour le support de calcul. Par exemple, nous avons choisi la méthode de la Bissection permettant un parallélisme paramétrique. Nous soulignons aussi l'impact de ces choix numériques sur le partage, le placement et l'accès aux données qui sont des points cruciaux lorsque des communications sur Internet entrent en jeu. L'utilisation de la méthode de Lanczos est un exemple de choix numérique favorable au contexte de calcul global. Nous avons ensuite exposé les contraintes dont nous devons tenir compte lorsque nous adaptons un problème sur une plateforme de calcul global. Pour certaines d'entre-elles, pour lesquelles le client peut agir au niveau applicatif, nous avons proposé des solutions comme "l'out-of-core", la persistance de données, les méthodes redémarrées. Dans l'esprit du projet Grand Large de l'INRIA Futurs, nous accordons une part primordiale à l'aspect expérimental, donnant ainsi une réelle crédibilité à notre mise-en-œuvre. Ainsi, nous avons réalisé des expérimentations sur plusieurs plateformes regroupant des ressources de calcul et des réseaux très hétérogènes à l'USTL, France, et à Tsukuba, Japon. L'usage de communications sur Internet est notamment un facteur essentiel pour des évaluations pertinentes. Enfin, nous bénéficions également de l'outil Grille'5000. Même si un tel environnement n'est pas représentatif du calcul global, nous montrons son utilité en testant le passage à l'échelle de notre application et en affinant les analyses faites dans un contexte de calcul global avec les plateformes France-Japon.

A ce thème de recherche principal, nous avons ajouté une considération relative aux mécanismes de réduction de consommation d'énergie. Ce thème de recherche est très dynamique dans le domaine du calcul haute performance. Le succès des super-calculateurs de la famille Blue Gene et les spécifications de la future machine pétaflopique du RIKEN en sont une parfaite illustration. De telles considérations énergétiques seront probablement bientôt grandement étudiées pour le calcul distribué à très grande échelle. Nous souhaitons

que les réductions de consommation énergétique obtenues lors de nos expérimentations encouragent d'autres chercheurs de la communauté du calcul global à persévérer dans cette thématique. En effet, nous avons obtenu des baisses de la consommation énergétique globale de 9% et locales de 20% pour une hausse du temps de calcul de seulement 2% au maximum. Il existe de nombreux autres facteurs laissant espérer des gains énergétiques. De telles recherches sont donc prometteuses.

Durant nos recherches nous avons constaté que la programmation et le déploiement d'applications à l'échelle de l'Internet est une tâche très complexe. Aux difficultés algorithmiques s'ajoutent d'autres contraintes telle l'hétérogénéité des intergiciels de calcul global, la complexité des interfaces de communications, etc. Nous sommes donc impliqués dans la conception du logiciel YML qui vise à masquer un grand nombre de difficultés au développeur. Nous avons présenté les différents modules logiciels que nous avons développés et nous avons expliqué comment adapter notre application numérique à YML. Un accent particulier a été mis sur les bénéfices d'une programmation orientée "workflow" (i.e. par graphe de tâches) et en utilisant une approche composant. Nous avons discuté des avantages et inconvénients de différentes orchestrations possibles de composants en mettant en avant la distinction entre parallélisme et distribution lorsque nous travaillons avec des logiciels de hauts niveaux, et en discutant de l'intérêt de la distribution de certaines tâches en raison du coût des communications.

Part II

Dissertation

Chapter 1

Introduction

1.1 Context

Main context related to Global Computing

Parallel computing, including High Performance Computing, is more and more distributed. Significant work has been done on network and computers in order to build efficient dedicated clusters and clusters of clusters. The hardware improvement has been done in conjunction with the development of computing software to deploy applications and reach very high performance. The current trend is exploring very large Grid Computing and Peer-to-Peer Computing by harnessing available computing and storage resources distributed over the Internet. This new track of distributed computing has required the implementation of software dealing with new constraints such as safety, volatility of volunteer peers and confidentiality of data. Moving from dedicated platforms to very large non-dedicated platforms requires also to modify the programming methodology of numerical applications. We must adapt the distribution of those applications by taking into account the new constraints. For instance, we choose the grain of parallelism of an application (size and number of tasks) depending on the bandwidth of the network and the limited storage space of the peers.

The INRIA Futurs Grand-Large project is a pioneer in very large heterogeneous computing, globalization of data and resources distributed over the Internet. This project particularly focuses on three research topics. It favours an experimental approach in order to handle them. The first topic concerns the development and the experimentation of a Peer-to-Peer Computing environment. It targets to hide to the end-users, the complexity and the heterogeneity of networks and computing resources. The second one deals with specific issues such as volatility and reliability. The last one focuses on the application level. It aims at proposing a programming methodology suitable for very large scale

numerical applications on Global Computing platforms. Global Computing is a generic name for several models such as large Grid Computing, Peer-to-Peer Computing, Remote Computing and so on. This topic of research requires a significant algorithmic work but also an analysis of available tools, programming languages and software in order to adapt efficiently the numerical applications to Global Computing.

The MAP team of the LIFL/CNRS laboratory of the USTL is focusing on the third point. Nevertheless, its work is deeply linked to the first one. In fact, our research experiments often use the tools developed by the Grand-Large project. Thus, The MAP team plays a major role in the necessary testing step of computing software and, it provides a precious user feed-back.

Context related to power-aware parallel computing

Power and energy consumption has become a tremendous topic in the communities of micro-architecture and computer science. This first interest on low-power technologies was mainly due to the development of embedded systems such as cellular phones which generally run on batteries. The commercial success of those devices increased this trend. In this case, improving the autonomy was the major requirement.

Energy conservation is now a major requirement in server-class systems and High Performance Computing (HPC). The leading motivation is to save money. Indeed, low-power technologies allow reducing the consumption of massively parallel supercomputers, high performance clusters and web-server farms. Besides, the number of cooling systems can be decreased (and their consumption too). As an example, the RIKEN has recently delivered the specifications of its future 10-petaflop system in Japan (built by 2012). It specifies that the energy consumption of this new system have to be only $\frac{1}{10}$ of the energy consumption of a supercomputer built with common components and giving similar performances. The cost of electricity is not the only target. We must consider the Total Cost of Ownership (TCO) composed of the cost of acquisition and the cost of operation. As explained in [97], the latter cost is deeply linked with the reliability of the system. In fact, low-power architectures have a bigger Mean Time Between Failure (MTBF) than classical systems. So, the cost of maintenance is smaller and the availability is much better. In the case of e-business, a poor availability means huge losses of money. Moreover, such architectures also allow designing high-density packaging. It avoids using machine rooms which have an expensive cost of acquisition and, after, an expensive cost of operation. Finally, we must consider other motivations like environmental protection.

1.2 Motivations

Motivations for research on Global Computing

In this section, we browse the main reasons which motivate the parallel and distributed computing community to focus on Global Computing.

First, huge quantities of computing resources are connected to Internet. They can be Personal Computers (PCs), Networks of Workstations (NOWs) of schools or universities, clusters, and so on. Most of the time, those devices are switched on but remain idle. Thus it is very attractive to harness their unused resources. On a computing device, there are many kinds of resources and they can be used by a wide range of applications. For instance, the storage Peer-to-Peer applications are precursor by sharing files among peers. The memory and CPU cycle resources of peers can also motivate the distribution of applications. Few years ago, this concept of very large computing was not conceivable especially because of very slow Internet connections. Nowadays, more and more computers are connected to the Internet by means of xDSL technologies. Besides, the volume of disk space of desktop PCs is increasing tremendously. Their main memory and the CPU frequency are rising too. Thus, PCs become attractive candidates for distributed computing.

Second, we consider that Global Computing can provide a solution to people that have no access to significant computing tools. It concerns individual end-users, small companies such as start-up, schools, etc. The problem of the access is generally explained by the cost of efficient computing devices. Besides, the access to those machines is often restricted to few users because of safety and confidentiality considerations.

Third, we point out the continuous availability of computing devices. It allows providing an "on demand" access scheme to a large amount of resources. In other words, in the ideal case, we can find immediately enough suitable resources to solve an application without buying any expensive hardware. For instance, it can interest a common person in order to perform multimedia applications which are generally greedy in term of CPU, memory and disk space. Usual clients of supercomputers may also be attracted by this facility. Indeed, in case of a sudden increase of computational need, the capabilities of their supercomputer may be overcome. In this specific situation, Global Computing resources can supplement supercomputers. It avoids buying in a hurry a new expensive computing device. Besides, this kind of computing peak is supposed to be rare, thus a new bought machine would be seldom used later. We can even imagine a national requisition scheme of Global Computing resources for urgent and critical applications such as prediction of typhoon routes, real-time tsunami prediction, etc.

Motivations for research on power-aware Global Computing

It would be very interesting to use power-aware technologies in heterogeneous Grid Computing, even in Peer-to-Peer Computing. In HPC, many argue that power-aware computing sacrifices performance too much. We must reconsider this argument in the context of heterogeneous world-wide Grid Computing because the end-user does not, and cannot, target the highest performance any more. Indeed the computing and network resources are not dedicated. The client may agree to run his programs slower but cheaper provided that the performance remains acceptable.

Both the end-user and the volunteer peer can take advantage of power-aware computing platforms. The volunteer peer may be interested in saving energy while he is offering his resources. Thus the developers of platforms of heterogeneous Grid Computing should take into account power and energy consumption in order to attract volunteer peers and to guaranty the success of their software. The client not only performs his computations cheaper (or free of charge depending on the "economic model"), but he submits his tasks to a power-aware platform which probably attracts more peers than the other traditional platforms.

More and more private companies harness their own resources and perform Grid Computing. They can be interested in power-aware technologies. First, Grid Computing is only a tool and not an objective. So, those companies do not want to spend too much money on it. Reducing the cost of electricity is a first good point. Second, they often have a small staff in charge of resources. Thus, the good reliability of power-aware platform is a great benefit. Third, machines must fit in existing buildings and the space for cooling devices is limited. Therefore, it is almost necessary to use high-density packaging providing high values for metrics such as $FLOPs/m^3, GB/m^3$. Finally, employees may work in the same building as PCs and they may appreciate cool and quiet devices.

1.3 Scope of the study

Scope of the study on Global Computing

Supercomputers and clusters are traditionally used for the sake of scientific numerical problems because those applications generally need a lot of resources. For instance, a lot of applications are based on linear algebra methods using expensive matrix computations. Therefore, the study of those methods in order to adapt them on Global Computing platforms is a relevant topic of research. In particular, we focus on the real symmetric eigenproblem. This linear algebra problem is a good candidate for our study. Indeed, it is the base of many scientific applications. Besides, it gathers most of difficulties an end-user may face while distributing his application. An eigenproblem of large order

requires a lot of disk space, many CPU cycles and a lot of main memory. It generates also a lot of communications with significant volume of data. It poses the problem of data distribution, and so on.

Actually, our main goal is not only to adapt a specific linear algebra method. The study of the real symmetric eigenproblem is a pretext to focus on parallel and distributed programming paradigms and to propose helpful solutions to adapt many linear algebra problems to the context of Global Computing. We underline that we do not study Global Computing in order to compete with High Performance Computing. In fact, large scale computing over the Internet cannot reach the highest performances in term of speed of computations. We consider Global Computing as a complementary computing tool as explained in the motivations section. This study is necessary in order to show the viability of Global Computing, in particular for linear algebra methods.

Developing an efficient parallel and distributed application on a wide heterogeneous platform is a difficult task. Besides, there exists much Global Computing software using different interfaces. Thus, the end-user has to master the usage of several tools if he targets to deploy widely his application. YML is a framework which intends to hide the heterogeneity of hardware, software and provides a uniform access to the Global Computing resources. As a user of Global Computing, we use several tools and we face the difficulties that YML aims to hide. Therefore, we feel deeply concerned by the YML project and we participate to its development.

Scope of the study on power-aware Global Computing

We can exploit many characteristics of heterogeneous world-wide Grid Computing in order to save energy. For instance, on the volunteer peer's side, we can often decrease the CPU frequency without affecting the global performance because of the saturation of the memory (or another component). In fact, a commercial trend is to sell desktop PCs with a huge peak frequency whereas the other components are not scaled to the processor. The exploitation of the memory saturation has already been studied in the context of High Performance Computing and can be applied to heterogeneous Grid Computing. The CPU frequency is modified by means of Dynamic Voltage Scaling called DVS.

In Cluster Computing and High Performance Computing, we can also take advantage of the slack-time caused by an unbalanced distribution of work. The slack-time is the idle time of some computing resources while the others are still working. Those waiting resources can be run slower in order to save energy without increasing the execution wall-clock time. At the same time, the power/time efficiency (in $W.s^{-1}$) is improved!

In the context of heterogeneous world-wide Grid Computing, we consider two other sources of slack-time. The first one is due slow communication. In fact, contrary to the computations on supercomputers, dedicated clusters or dedicated Grids such as Grid5000, hetero-

geneous Grid Computing makes use of the Internet which has a low bandwidth and which is composed of many heterogeneous networks. The second source of slack-time concerns the heterogeneity of the computing nodes. Contrary to High Performance Computing and Cluster Computing which are using homogeneous resources in order to reach the highest performance, world-wide Grid Computing harnesses a wide range of heterogeneous volunteer devices. For instance, a powerful desktop PC may have twice more memory and a twice faster CPU than a laptop. We expect such an heterogeneity of the resources of the workers can generate a significant slack-time. To our knowledge, no previous work clearly evaluates the potential energy savings we can get by using DVS for this kind of slack-time. More generally, power-aware considerations have not yet been explored for heterogeneous large Grid Computing.

1.4 Contributions

Contributions for Global Computing

We first focus on parallel and distributed programming paradigms on world-wide heterogeneous platforms. This computing environment is characterized by the impact of communication over the Internet or low-bandwidth LAN. We stress the interest of the parametric parallel paradigm (also called task-farming paradigm) in order to minimize the number of communications. We oppose this paradigm to a classical parallelism using many communications and synchronization points. In this last case, we show the cost of communication and propose some solutions to minimize the volume of transferred data. So, depending on the characteristics of the computing environment (HPC, cluster inside a high-bandwidth LAN, Global Computing, etc.), the choice of the appropriate parallel programming paradigm is essential. This choice is deeply linked with the numerical method adopted to solve the problem. By means of the real symmetric eigenproblem as an example, we underline that we must choose the good paradigms at the very beginning of the application study. Besides, when the numerical method allows it, we underline that it is conceivable to combine an optimal paradigm, to a non-optimal paradigm if we do not find better solutions. In the particular case of the real symmetric eigenproblem, the parametric parallelism of the Bisection and Inverse Iteration methods is wrapped into a communicating algorithm based on the Lanczos tridiagonalization and a restarted strategy. With the restarted scheme, we confirm the importance of the choice of the numerical methods depending on the context of computations. Indeed, by means of this strategy, we can solve a very large problem while working in a little subspace. It is relevant since resources of peers may be limited.

The methodology we use to develop a program suitable for the constraints of Global Computing can be employed for many numerical applications. We delay as far as possible the language and software considerations. For instance, after the choice of the programming paradigms, linked with the choice of the numerical method, we propose to deal with data

issues. Depending on the operations on data (i.e. the data access pattern), we can choose the data structures, the data sharing and the data mapping. It appears that matrix-vector products are very convenient operations due to their data-access scheme allowing an easy distribution of data and of computations. The drawback is the number of communications and synchronizations. Then we show the interest of techniques such as the out-of-core and the data-persistence which can be used by most linear algebra problems. By using out-of-core, we can solve very large problems and use a little amount of memory on the remote computing nodes. Indeed, the resources of Global Computing can be very heterogeneous and some nodes may have few memory. A simple evaluation of the memory needs show the necessity to use it. The data persistence reduces significantly the volume of transferred data. Although we can decide to use those 2 techniques before the choice of a Global Computing software, their implementation deeply depends on software choices. Next, we show the importance of a good usage of the Global Computing software in order to build the experimental platform. Indeed, we give an example of bad configuration which burdens drastically the performances: the wall-clock times to solve the problem are doubled.

In addition to our contribution related to a Global Computing programming methodology, an important contribution is to implement all propositions (no simulations) and to test the programs on realistic world-wide heterogeneous platforms. The concrete results show the feasibility and the viability of the Global Computing model for linear algebra problem as long as the speed of the computations is not an important criterion of the client. In this last case, High Performance Computing devices are the most appropriate tools.

We contribute to the development of the YML framework as explained in previous section. Our contribution covers 2 points. The first one concerns the development of modules. In particular, we have developed a back-end for the OmniRPC RPC programming software. Thus, a client can currently submit the same YvetteML program to an OmniRPC platform or an XtremWeb one. We have also focused on the Data Repository Server/Client module and on the YML Worker module. The client does not interact directly with them but they play a major role in YML. The first one manages the parameters and the binaries upload/download requests. The second one executes the binary on the remote computing nodes for any kind of back-end. The second kind of contribution related to YML concerns the development of a real symmetric eigensolver using the same numerical method as previously. It is not a simple work of implementation. We must reconsider our way to develop the eigensolver by taking into account the component programming model of YML and some constraints of the YvetteML graph language. Besides, as YML hides all middleware considerations, some issues like data-persistence are problematic. We propose to focus more on distribution than parallelization of tasks. We have proposed many re-usable components and a workflow of execution in order to orchestrate them. By means of the back-end mechanisms, the same components and graph can be deployed on several computing platforms managed by different middleware. Besides, during the experimental step, we have proposed to test the latest versions of YML. On the one hand, we have contributed to improve the stability of YML. On the other hand, we have provided a necessary user feedback and we have proposed new features in order to extend

the capabilities of YML.

Contributions for power-aware Global Computing

In this work, a first contribution is to evaluate the impact on the energy consumption of the heterogeneity of a cluster/Grid platform (from a CPU frequency point of view). Then, we show how to take advantage of the slack-time caused by this heterogeneity in order to save energy by using DVS, with no significant loss of performance. This work may also contribute to motivate researchers of the community of heterogeneous large Grid Computing to start working on this new promising topic. In order to give credibility to our work, we have modified the eigensolver developed for our study of the real symmetric eigenproblem and we get a DVS-capable eigensolver. In fact, the eigenproblem appears in many daily and industrial applications. Therefore, this study shows that many potential users are concerned by our work.

1.5 Outlines

An essential characteristic of our field of research is the interdisciplinarity which combines computer science and linear algebra. We have to review related work in both domains. Regarding linear algebra, we can limit this study to the real symmetric eigenproblem. The computer science domain is much larger since it concerns parallel and distributed computing. Chapter 2 presents a wide overview of Global Computing tools. Then, Chapter 3 gives a presentation of fundamental linear algebra methods for the real eigenproblem and presents related works for its parallelization in the domain of High Performance Computing. The remainder of the dissertation is organized as follows. In Chapter 4, we introduce the parametric parallelism and study it by means of the Bisection and Inverse Iteration methods in order to solve the real symmetric tridiagonal eigenproblem. Then, in the same chapter, we deal with a more general parallel and distributed paradigm and we handle more constraints of Global Computing. Our case study is the real symmetric eigenproblem. In particular, we use a restarted Lanczos tridiagonalization. Chapter 5 focuses on low-power technologies for Global Computing. It is a new promising field of research in our context although it is already a mature topic in High Performance Computing. For a better comprehensibility of this dissertation, we intentionally provide a short related work on low-power computing in this chapter instead of doing it with Chapters 2 and 3. Next, in Chapter 6, we present our contributions to the development of the YML Global Computing framework. We unify this work with the study of Chapter 4 by adapting the real symmetric eigenproblem on YML. Finally, we conclude and present some perspectives in Chapter 7.

Chapter 2

Tools for Global Computing

We adopt a user approach and consider Global Computing software as tools. Indeed, our main goal is not to create software but we use existing ones in order to propose some solutions to solve large scale real symmetric eigenproblems. In order to find the relevant tools for our study, we made a wide survey of existing software of Global Computing. The most intuitive way to present them is to start with low-level ones and to finish with high-level software. In fact high-level tools rely on low-level ones. This wide survey does not target at providing an exhaustive list of tools. It intends to represent our point of view of the distributed computing domain and it gathers the main software we met while searching the suitable tools for our problem.

2.1 Communication models and tools for the Grid

The basis of parallel and distributed computing is the communication layer between the computing units which compose a complex system. We generally consider distributed- and shared-memory systems and each one has specific models and tools of communication.

2.1.1 Communication models and tools for distributed-memory systems

2.1.1.1 Remote Procedure Calls

The Sun RPC implementation is widely-known. However, there exist other RPC libraries dedicated to large scale distributed computing. We cite DCE-RPC [1], DFN RPC [2], Peregrine RPC [3], MRPC [4] and RPC-V. DCE-RPC targets the interoperability of

applications in an heterogeneous environment but it only provides synchronous RPC over UDP. Nevertheless, we can emulate asynchronous RPC and call-back mechanisms by means of a multi-threading capability. In order to transfer large volume of data, DCE-RPC allows using non-blocking unidirectional data-pipes. DFN-RPC has 4 kinds of RPC over TCP: synchronous RPC, asynchronous and buffered asynchronous RPC, parallel RPC. Besides, it provides a call-back mechanism and a message-passing-like mode based on RPC. In [1], DFN-RPC has better transfer rate and less delay than DCE-RPC, SUN-RPC and PVM. The overhead over TCP is very low. Peregrine RPC optimizes this transfer rate by using its own protocol over IP. MRPC has been integrated into Compositional C++. It relies on the Active Message model and support MPMD architectures. Active Messages provide an efficient data control and data transfer. Finally, RPC-V intends to be fault tolerant by means of several techniques of request replication.

2.1.1.2 Remote Method Invocations

The Java Sun RMI implementation is not adapted to large distributed computing. In [5], its overhead is twice larger than the overhead of some RPC and MPI implementations. The Manta [6] compiler builds a binary whose Sun RMI methods are replaced by optimized routines based on an efficient communication protocol called PANDA. Manta is able to deliver standard JAVA bytecode to remote computers that do not have a Manta environment. Inversely, the Manta environment is able to handle standard Java bytecode. A similar effort targets to optimize object serialization. Ibis [5] is a kind of middleware dedicated to RMI. Indeed, the front-end supports the API of the Sun RMI, GMI [7], RepMI and Satin. RepMI and Satin are software of the Albatross project [9]. GMI adds collective communication to the RMI and RepMI focuses on object replication. Satin enables parallel RMI and a replication mechanism by using slave threads. Finally, the back-end of Ibis has many interfaces for communication protocols like TCP, UDP and PANDA.

2.1.1.3 Message-passing communication

This paragraph refers to PVM [11], MPI and IceT [12]. PVM virtualiazes the resources of many heterogeneous computers connected to the same network. So, it provides to the user the image of an unique supercomputer. PVM supports a lot of architectures, from the simple desktop PC to the powerful vectorial machine. The messages are exchanged between computing tasks that are distributed among computers depending on criteria related to the architectures or to the requests of the user.

MPI-1 and MPI-2 are the specifications of a message-passing library and not an implementation. We notice the following implementations: MPICH, MPICH-G2 [10], PACX-MPI, ScaMPI, MPI-Connect and LAM. MPICH-G2 links MPICH to the Globus toolkit (Subsection 2.4). PACX-MPI implements MPI-1 but only a part of MPI-2. It aims to optimize

collective communication on large scale Grids and supports several protocols like TCP, ATM, SSL. ScaMPI implements most of MPI-1 specifications and focuses on dynamic process management and the MPI-IO (Input/Output) part of MPI-2. MPI-Connect provides the interoperability of many MPI implementations. Finally, MagPIe [8] is based on MPICH but it improves collective communication.

As PVM, the IceT message-passing library virtualizes the underlying network of computers. The IceT routines can be integrated into a standard Java code in order to benefit from the bytecode portability on the Sun JVM. On the contrary, the user of PVM has to provide as many binaries as harnessed architectures.

2.1.2 Communication models and tools for shared-memory systems

OpenMP is a specification of library routines, environment variables and directives that are introduced in a sequential program in order to perform a shared memory parallel execution. During the execution, each parallel section is performed by concurrent shared-memory threads.

2.1.3 Tools suitable for shared-memory or distributed-memory systems

We present High Performance Fortran (HPF) and P4. HPF is a programming language suitable for SPMD parallelism. Data are distributed and mapped on computing nodes depending on directives inserted into the program. The HPF compiler provides a dedicated binary and optimizes communication for each architecture (shared- or distributed-memory). The P4 library works on shared- or distributed-memory MIMD machines or clusters of PCs. If the memory is shared, the user can define its own lock mechanisms. In case of distributed-memory, the message-passing communication model offers point-to-point and collective communication.

2.2 Workflow languages and tools

2.2.1 Web-services and workflow languages

Due to the emergence of the web-services, many dedicated languages have been created. With WSFL [62] based on the XML language, IBM wants to compose web-services within

an oriented graph. Microsoft has proposed XLANG [63] which uses the description language WSDL. The web-services are autonomous agents which interact between each other. BPEL4WS [64] uses the structured language XLANG and the oriented graph language WSFL. It considers two kinds of process. The first one defines the exchanges between the web-services and the second one specifies the execution graph. Finally, Sun, Intalio, SAP and BEA propose WSCI. It has a similar approach as XLANG. In fact, WSCI is a description language comparable to WSDL. It defines message flows between the web-services.

2.2.2 Workflow languages for Grid services

Global Computing faces similar problems as web-services. So, the Open Grid Services Architecture (OGSA) describes an architecture for a service-oriented Grid Computing environment. It defines the interfaces of the main Grid services by means of the description language WSDL. Then, the GSFL [66] language, based on XML Schema, allows specifying interactions between those Grid services so as to build meta-services.

2.2.3 Workflow tools for the Grid

2.2.3.1 Software-aided workflow design

Software like PYRROS [67] and CASH [68] aim to generate a parallel code from a sequential one by using annotations written by the client. For instance, such annotations may specify how to build independent tasks, how to distribute data and so on. OREGAMI [69] does not parallelize a sequential program but it handles the mapping of data on the computing resources. The user inserts into the parallel program instructions by means of the LaRCS description language and the compiler generates the execution graph. The 3 previous tools provide a graphical representation of the execution graph.

2.2.3.2 User-designed workflows

Most of tools let the user make the execution graph. He has to manage data and tasks dependences and sometimes the distribution of data on nodes. With the DAGman, GridAnt [70] and YvetteML (cf. YML, Section 2.7), a text file defines the workflow. Many tools provide a GUI in order to compose the workflow: FhRG, CODE 2.0 [71], HeNCE [72], Paralex [73], Symphony [74], TENT [75], TME, WebFlow [78], Triana [76], a component of UNICORE [35], GridFlow and Ptolemy II. GridAnt, HeNCE and UNICORE offer another GUI showing dynamically the execution of the parallel program.

2.2.3.3 Focus on workflow models

[77] presents some workflow patterns used by the previous software. DAGman, UNICORE, YvetteML, PYRROS and CASH rely on a direct acyclic graph (DAG). HeNCE uses a modified DAG allowing making programs with unknown number of iterations. OREGAMI proposes direct, acyclic and process-oriented graphs by means of the TCG (Temporal Communication Graph) representation. [77] allows using many patterns like DAG, TCG and ITG (Iterative Task Graph). An ITG is a non-oriented graph allowing iterative programs with unknown number of iterations. FrHG chooses the Petri net representation. PtolemyII and Triana propose the most exhaustive lists of workflow patterns. PtolemyII offers 12 proprietary models. Triana has a proprietary or several usual models like DAG, BPEL4WS pattern. Other software does not use a specific pattern. The vertices of the graph often represent the tasks whereas the edges concern data dependences. It is called data-flow diagram and is used by TME, WebFlow, Symphony and Paralex. For TENT the edges only refer to task organization without any data specification: it is called control-flow. Phred and GSFL combine data-flow and control-flow because they consider data dependences and task organization may have different flows. Finally, we notice that CODE 2.0, Paralex, BPEL4WS, WSFL and WSCI allow making hierarchic graphs. It consists in encapsulating and re-using existing graphs in order to compose a new workflow.

2.3 Large scale resource management and scheduling

There exist a lot of scheduling and resource management local tools for computing platforms (DQS, NQE, PBS, LoadLeveler, and so on). We here present similar software for large scale computing which often forwards instructions to the local tools. First, some tools focus on only one objective. They belong to a toolkit or a bigger project. For instance, GRAM and ARMS are responsible for the resource management and AppLeS is dedicated to global scheduling. Second, there are more complex tools like EZ-Grid [27], Nimrod [25], Gallop [28] and SCIRun. They collect information related to the computing and network resources and perform the scheduling. SCIRun is almost an autonomous Global Computing software. It is interesting to focus on their computing paradigm. In fact, Nimrod, SCIRun and APST [24] (extension of AppLeS) aim to do task-farming (cf. Section 4.2.1). Gallop, with its local scheduler Prophet, performs Remote Computing.

2.4 Global Computing toolkit

Globus [19] is a toolkit for Global Computing software. It provides many tools like GRAM for the resource management, Nexus in order to configure the network layer, MDS and

HBM for the information services, GSI and GSS for security considerations, GEM, GASS, RIO, GridFTP and so on. Globus is becoming a standard de-facto. Indeed, a lot of Global Computing software solutions target the interoperability with Globus-based platforms

2.5 Grid Computing software

2.5.1 Grid RPC programming software

Low level RPC programming is not user-friendly for most of scientists. So, we present software whose API hides the complexity of the RPC. Ninf [13][14] offers synchronous and asynchronous calls. It can emulate parallel RPC and perform call-backs. OmniRPC [15] is a thread-safe evolution of Ninf by means of OpenMP. Netsolve [30][31] is merely the most known Grid RPC software. It allows synchronous, asynchronous and parallel RPC. It manages data dependences by making a DAG and it has simple fault-tolerant mechanisms. Instead of proposing another independent software, GridRPC is a specification which aims to reach interoperability among existing RPC software. There are several available GridRPC implementations: NinfG, Netsolve and DIET (we consider DIET as a higher level tool).

2.5.2 High-level Grid Computing software

In this paragraph, we deal with MW [36], Bond, EveryWare [37], Cactus [38], DIET and GrADS [34]. They have often more features than previous RPC programming software and they target to hide the complexity of many problems related to the Grid. MW uses PVM. So it virtualizes the resources of the Grid. The scheduling is very simple because it relies on static information and a FIFO queue. MW offers also a basic fault-tolerance. Bond is an agent system based on the Java JADE framework and plans to use Globus. The agents are able to communicate between each other. They solve the tasks assigned by a scheduler which represents the application by means of a DAG. EveryWare [37] contains APST (Section 2.3) and is based on the middleware Legion. It can also rely on a Globus-based middleware or on Condor for lower scale platforms. It proposes an interesting Gossip mechanism, coupled with NWS, in order to gather information and maintain a global state of the platform. Cactus [38] has a highly modular and configurable architecture. It contains a minimal core (the trunk) and the client can plug many modules (the thorns) depending on his needs. Cactus provides several implementations of thorns. For instance, regarding the communication layer, it offers thorns for PVM, OpenMP, MPICH-G. DIET is built upon CORBA communication protocols and has chosen the GridRPC API. It uses LDAP and some tools presented in this document like NWS, FAST, etc. Since the scalability is a major consideration of DIET, the scheduling is distributed. Finally, we consider GrADS which is built with many Globus tools (e.g.

GRAM, NWS). It has an original approach. The client builds his distributed application by means of configurable components and annotations. The annotations define the needs of the application. The GrADS compiler configures the components depending on the available computing resources and the requirements of the annotations.

2.5.3 Object-oriented Grid middleware

The CORBA standard is based on the Object Management Architecture defined by the Object Management Group. We notice 5 main components: the Object Request Broker in charge of the transport of requests, the object services (system-oriented services), the common facilities (user-oriented services), the application objects (domain-specific objects) and the user application objects. The interface of each object is defined by a description language (IDL) and is clearly separated from the implementation of the object. We notice that CORBA is deployable on the Internet with the IIOP implementation of the communication protocol CORBA GIOP. CCM, also called CORBA3, is an evolution of CORBA using the component programming paradigm. The Legion [22] middleware is currently called Avaki. It considers the resources of the Grid as a unique virtual machine that we can widen by adding objects. An object is a resource or a service. The objects are independent from each other in order to guaranty a good modularity. The interface definition of an object by means of the CORBA IDL or the MPL language is also clearly separated from the implementation. In Globe [18], all processes communicate through shared distributed objects. Those objects are transparently physically distributed following two techniques: shared or duplicated. The user-defined objects must contain 4 sub-objects: interface definition, replication policy, communication policy, replication coherence control.

2.6 Remote Computing and Peer-to-Peer Computing software

In this section, we distinguish 3 kinds of software targeting computations over the Internet. First, there are stand-alone applications developed for a specific need of computations. Second, we present tools which help scientists to build and deploy their own stand-alone applications. Finally, we consider higher level software able to deploy (almost) any kind of applications.

2.6.1 Stand-alone applications

Those applications use the principle of "cycle stealing". It consists in harnessing volunteer resources connected to the Internet when they are idle. The parallel paradigm is the parametric parallelism, also called task-farming (cf. Section 4.2.1). Famous applications are Distributed.net, Seti@home and GIMPS. In bio-informatics, scientists seldom have computing resources, so they currently make use of this tremendous computing facility for projects like Genome@home, Folding@home.

2.6.2 Tools to develop and deploy stand-alone applications

Developing such stand-alone applications requires a significant knowledge in computer science. Therefore, we present two tools that help people to develop them. The first is Cosm. It provides a programming API called Mithral CS-SDK in order to parallelize a sequential program. It offers also a communication library CPU/OS Layer hiding the heterogeneity of volunteer nodes on the Internet. Cosm has been used for the EON project and Genome@home. The second tool is BOINC [40]. It allows each volunteer node hosting several applications and specifying how to share its resources. BOINC considers also fault tolerance and security. For instance, we notice a redundancy mechanism and then a consensus technique for multiple results. Currently, BOINC is the base of new versions of Seti@home, Folding@home or new projects like Predictor@home, Einstein@home and Climateprediction.net.

2.6.3 Polyvalent Remote and Peer-to-Peer Computing Software

Snipe [23] is restricted to the Remote Computing model. It is based on PVM and RDCS. RDCS is in charge of data replication and replication coherence control, fault detection, check-pointing and restarting. The following software solutions use the Peer-to-Peer Computing model although un-symmetric relationships generally remain between the entities. We cite XtremWeb [16][17], Javelin [41] and Javelin++ [42], Charlotte [33], Bayanihan [44], Entropia [43] and finally ATLAS. The Java language is generally used because of its portability since many platforms can run a standard JVM. Besides, for safety considerations, Javelin, Bayanihan and Charlotte rely on the isolation of Java applets in the web-browser. Javelin++ and XtremWeb rely on the JVM sandboxing capability since the Sun JDK 1.2. Entropia uses a proprietary sandboxing mechanism. Another interest of Java is the wide range of communication tools. Javelin++ is based on RMI and Bayanihan uses the ORB HORB implemented in Java. The dominant programming paradigm is the parametric parallelism. Nevertheless, Entropia aims to handle data-parallelism and Charlotte distributes parallel sections of a program. The scheduling policy is usually simple but we notice the interesting "work stealing" mechanism used by ATLAS and Javelin++ which has been introduced by the multi-threaded language Cilk [45]. We finally focus on

fault tolerant mechanisms. They are essential in such large and volatile environments. By allowing only task-farming, XtremWeb can implement a simple mechanism of fault detection by means of heart-beat stimulations and re-scheduling of the entire lost task. As Charlotte, Javelin++ and Bayanihan tackle data dependence, and the fault tolerance is more complex. They use the "distributed eager scheduling" mechanism explained in [42]. Bayanihan adds also the task replication, the consensus of multiple results and uses a blacklisting policy. Javelin++ intends to use a consensus in order to distinguish node failures from network ones.

2.7 Global Computing framework

In Section 2.6.3, we have presented software solutions that enable deploying an application over the Internet. Global Computing frameworks target the same goal but they provide more features like workflow design module, monitoring and profiling, debugging and so on. We here consider UNICORE [35], YML [48], Gateway [39] and P-GRADE (MTA SZTAKI company). We first focus on the front-end properties. UNICORE, Gateway and P-GRADE provide a GUI with a workflow composition tool and a monitoring environment. Actually, P-GRADE seems to be dedicated to Cluster Computing and it offers many additional features like debugging, post-mortem analysis. YML does not propose a GUI and the workflow design is done by means of the workflow language YvetteML in a text file. The 4 frameworks require the client to divide his application into independent tasks. Each YML task is an XML component: it makes easy the re-use and the composition of tasks. The UNICORE and Gateway tasks have embedded information like security requirements. Then we look at the back-ends of those frameworks. Gateway is based on the Globus toolkit. UNICORE relies on local resource management tools and local schedulers like PBS, LoadLeveler. Actually, UNICORE is a framework dedicated to Remote Computing. The GRIP project intends to extend the capabilities of UNICORE by using Globus and the web-services. YML allows using several back-ends. There currently exist two back-ends for OmniRPC and XtremWeb. Finally, P-GRADE relies on MPICH-G2 or Condor-G.

2.8 Grid Portals

In [79], a portal is defined as a web-server application able to communicate with a Grid software. [80] considers 3 kinds of portals. We first have the "user portals". They are simple web-server applications accessing to Global Computing platforms in order to submit and monitor some tasks. Sometimes, it is also possible to select the computing resources. Ninf [79], Legion [85], Nimrod-G [84] and UNICORE have their own portal. NPACI HotPage [81] and GRB [83] offer a solution to Grid built with the Globus toolkit. WebSubmit [82] only gives access to local schedulers (LoadLeveler, NQS, LSF). Then, we consider "Science Portals". They are dedicated to specific and complex applications

that can hardly be deployed through a conventional "user portal". [80] presents XCAT-SP and gives 2 examples of applications. Finally, the "portal construction kits" like GridPort [86] and GPDK allow building portals. For instance, Visportal [87] is built by means of GPDK. The Computing Portals Organization[1] maintains a list of existing projects related to portals.

2.9 Profiling, prediction, simulation and emulation tools for parallel and distributed applications

2.9.1 Profiling tools

We present AIMS [88], VAMPIR & VAMPIRtrace [89] (Intel), Pablo [90], VT (IBM), MPE & Nupshot and Paradyn [91]. [92] compares some of those tools. Except for Paradyn, they use collected data in order to perform a post-mortem analysis. Paradyn is limited to a runtime analysis. So, it is adapted for long applications. VT also allows dynamic profiling of IBM AIX kernels. It is interesting to see how intrusive the tool is. Paradyn does not need any modification of the program. The default profiling modes of VAMPIR, MPE and VT do not need any modification too, but we can improve/extend collected data by adding routines to the source code of the application. With AIMS, we have to add routines through a text editor or a GUI. Pablo also requires some routines and, sometimes, we have to substitute standard functions by the Pablo ones (e.g. profiling of I/O). We focus now on the communication layers handled by those profiling tools. All of them support MPI. AIMS and Paradyn also handle PVM whereas NX is tackled by AIMS and Pablo. Regarding the programming API, C and Fortran are commonly accepted. VT adds C++. HPF is handled by VT, AIMS and VAMPIR. We notice that MPE is language-independent. Finally, we look at supported platforms. AIMS, VAMPIR, Pablo and Paradyn can work on many configurations. MPE & Nupshot only requires TCL/TK libraries. On the contrary, VT exclusively works on IBM RS6000 and IBM SP.

2.9.2 Prediction tools

NWS [20][21] performs measurements related to the different entities of the Grid (e.g. network bandwidth and latency, disk space). The collected information is then gathered in order to perform short-term predictions that can be used for the scheduling. PACE [94] aims to predict the computing and communication times of a distributed application. The user has to model his application and to provide information about the computing resources and the network. PACE also estimates the variation of those times by modifying some parameters of the application or the configuration of the Grid. Then, in order to

[1] http://www.computingportals.org

predict the performances of distributed applications, FAST [93] uses NWS and evaluates the needs of the routines (memory and computing times). This last point relies on tests done during the initial installation of FAST on the computing nodes.

2.9.3 Simulation and emulation tools

A simulation consists in reproducing the behaviour of a system by means of an analytic model (e.g. execution of an application on a Global Computing platform). The emulation allows testing in real time the behaviour of the developed system. It uses an experimental environment reproducing the targeted environment. An interesting review is done in [96]. It presents some simulation tools like Brick , SimGrid, SimGrid2, GridSim, GangSim and OptorSim. It also compares grid emulation tools such as MicroGrid and Grid eXplorer. Emulators like ModelNet and NetBed are dedicated to the reproduction of communication over the Internet.

2.10 Chosen tools

We choose two Global Computing software in order to build several experimental platforms and deploy a parallel and distributed eigensolver. On the one hand, we use the Grid Computing software OmniRPC based on the RPC programming paradigm. On the other hand, we choose the XtremWeb Peer-to-Peer Computing software. We motivate those choices in Chapter 4 Section 4.2.2.3. XtremWeb proposes a web portal but its usage is strictly limited to pure task-farming. So we prefer using the Java API of XtremWeb. Then, we adapt the same eigensolver to the YvetteML workflow language of the YML framework which has 2 back-ends for OmniRPC and XtremWeb. The motivations for using the YML framework are given in Chapter 6 Section 6.1.1. Next, we also use some MPI implementations such as MPICH or LAM. The main motivation is that Dynamic Voltage Scaling routines of the experimental low-power cluster are available only for an MPI API (see details in Chapter 5).

Chapter 3

Overview of numerical methods for the real eigenproblem

In this section, we first focus on important numerical methods allowing computing the real eigenproblem. Then, we present interesting works on parallel real symmetric eigensolver.

3.1 Short definitions reminder

$A \in M_N(\mathbb{R})$	$A \in M_N(\mathbb{C})$
A invertible $\leftrightarrow \exists A^{-1}$, $AA^{-1} = A^{-1}A = I_n$	A invertible $\leftrightarrow \exists A^{-1}$, $AA^{-1} = A^{-1}A = I_n$
A symmetric $\leftrightarrow A^t = A$	A hermitian $\leftrightarrow A^* = A$
A normal $\leftrightarrow AA^t = A^tA$	A normal $\leftrightarrow AA^* = A^*A$
A orthogonal $\leftrightarrow AA^t = A^tA = I_n$	A unitary $\leftrightarrow AA^* = A^*A = I_n$

The real eigenproblem of a real matrix A consists in finding some eigenpairs (λ,u) such that $Au = \lambda u$ where $A \in M_N(\mathbb{R})$, $u \in \mathbb{R}^N$ and $\lambda \in \mathbb{R}$.

Two matrices A and A' are similar if there exists an invertible matrix P such that $A' = P^{-1}AP$. A is diagonalisable is A if similar to a diagonal matrix. Two similar matrices have the same eigenvalues.

3.2 Important numerical methods for the real eigenproblem

3.2.1 Power Iteration method

Hypothesis: A is a real diagonalisable matrix, $A \in M_N(\mathbb{R})$. We note $\lambda_1, \dots, \lambda_N$ the eigenvalues of A with the following condition: $|\lambda_N| > |\lambda_{N-1}| \geq |\lambda_{N-2}| \geq \cdots \geq |\lambda_1|$. $u_1, u_2, \dots, u_N$ are the associated eigenvectors.

The Power Iteration method computes λ_N, the largest eigenvalue of A in magnitude. We consider an initial vector x_0 such that $x_0 = \alpha_1 u_1 + \alpha_2 u_2 + \dots + \alpha_N u_N$ and the sequence $x_{n+1} = Ax_n$. We get:

$$x_k = {\lambda_1}^k \alpha_1 u_1 + {\lambda_2}^k \alpha_2 u_2 + \dots + {\lambda_N}^k \alpha_N u_N$$

$$x_k = {\lambda_N}^k [\alpha_N u_N + \dots + (\frac{\lambda_2}{\lambda_N})^k \alpha_2 u_2 + (\frac{\lambda_1}{\lambda_N})^k \alpha_1 u_1]$$

Besides:

$$\text{for } i < N, \lim_{k \to +\infty} (\frac{\lambda_i}{\lambda_N})^k = 0$$

So when k is large: $x_k \sim \lambda_N^k \alpha_N u_N$ and $|\lambda_N| \sim \frac{\|x_{k+1}\|}{\|x_k\|}$, $u_n \sim x_k$.

Algorithm 2 Algorithm of the Power Iteration method

Require: A, x_0, ϵ
Ensure: λ_N
  $x_{cour} \leftarrow x_0$
  $\lambda_{prec} \leftarrow 1$
  $\lambda_{cour} \leftarrow 0$
  while $|\lambda_{cour} - \lambda_{prec}| > \epsilon$ **do**
    $\lambda_{prec} \leftarrow \lambda_{cour}$
    $x_{normal} \leftarrow \frac{x_{cour}}{\|x_{cour}\|}$
    $x_{cour} \leftarrow A x_{normal}$
    $\lambda_{cour} \leftarrow x_{normal}^t x_{cour}$
  end while

If A is invertible we can find the lowest eigenvalue (in magnitude) by means of the Inverse Iteration method. Indeed, the eigenvalues of A^{-1} are the invert on those of A. The condition on the eigenvalues becomes $|\lambda_N| \geq |\lambda_{N-1}| \geq |\lambda_{N-2}| \geq \cdots \geq |\lambda_2| > |\lambda_1|$. This method does not require to know A^{-1} but requires to solve linear systems $Ax = B$.

3.2.2 Deflation method

Hypothesis: A is a real diagonalisable matrix, $A \in M_N(\mathbb{R})$. Its eigenvalues are ordered as follows: $|\lambda_N| > |\lambda_{N-1}| > |\lambda_{N-2}| \geq \cdots \geq |\lambda_1|$. We note $u_1, u_2, \ldots, u_N$ the associated eigenvectors.

If we want to compute more than one eigenvalue, the Power Method is not sufficient. The Deflation method gets the second largest eigenvalue (in magnitude). If the method is repeated and if all the eigenvalues are distinct (i.e. in the previous condition, all the $\geq$ become $>$), then we can find all the eigenvalues. However, the accumulation of rounding-off errors and approximations leads to a degradation of the results.

The Deflation method consists in finding a matrix B whose eigenvalues are 0, λ_{N-1}, λ_{N-2}, $\ldots$, λ_1. Then, the Power Method is applied to B. There exists a basis $(v_1, v_2, \ldots, v_N)$ such that $\forall (i,j) v_i^t u_j = \delta_{ij}$ (Kronecker's delta). If we note $B = A - \lambda_N u_N v_N^t$ then $\forall i = 1, \ldots, N$: $Bu_i = Au_i - \lambda_N u_N (v_N^t u_i) = \lambda_i u_i - \lambda_N u_N \delta_{Ni}$. We see that the eigenvalues of B are 0, λ_{N-1}, λ_{N-2}, $\ldots$, λ_1.

3.2.3 Jacobi method and Givens rotations

Hypothesis:A is a real symmetric matrix, $A \in M_N(\mathbb{R})$.

The Jacobi technique is also an iterative method. At each iteration k, we compute a new matrix A_k similar to A such that $A_k = Q_k^{-1} A Q_k$ where Q_k is a real orthogonal matrix ($Q^t = Q^{-1}$). If A is diagonally dominant ($\forall i, \sum_{j \neq i} |a_{ij}| < |a_i|$), A_k converges to a real diagonal matrix D which has the same eigenvalues as A. If the condition is not respected, A usually converges but not always. In case of convergence, the eigenvalues of A are the elements of D and its eigenvectors are the columns of the matrix $Q = Q_1 Q_2 \ldots Q_k$. The Q_k matrices are called Givens rotation matrices (see Figure 3.1). The elements of $A_{k+1} = Q_k^{-1} A_k Q_k$ are:

$$\begin{cases} \text{if } i \neq p,q \text{ and } j \neq p,q & a_{ij}^{k+1} = aij^k \\ \text{if } i = p \text{ and } j \neq p,q & a_{pj}^{k+1} = \cos\theta a_{pj}^k - \sin\theta a_{pj}^k \\ \text{if } i = q \text{ and } j \neq p & a_{qj}^{k+1} = \sin\theta a_{pj}^k + \cos\theta a_{qj}^k \\ \text{if } i = j = p & a_{pp}^{k+1} = \cos^2\theta a_{pp}^k + \sin^2\theta a_{qq}^k - \sin 2\theta a_{pq}^k \\ \text{if } i = j = q & a_{qq}^{k+1} = \sin^2\theta a_{pp}^k + \cos^2\theta a_{qq}^k + \sin 2\theta a_{pq}^k \\ \text{if } i = p, j = q & a_{pq}^{k+1} = \cos 2\theta a_{pq}^k + \frac{\sin 2\theta}{2}(a_{pp}^k - a_{qq}^k) \\ \text{the other components are found by symmetry} \end{cases}$$

$$\mathcal{Q}_k = \begin{pmatrix}
1 & & 0 & \vdots & & & & \vdots & & & & \\
 & \ddots & & \vdots & & 0 & & \vdots & & 0 & & \\
0 & & 1 & \vdots & & & & \vdots & & & & \\
\cdots & \cdots & \cdots & \cos\theta & \cdots & \cdots & \cdots & \sin\theta & \cdots & \cdots & \cdots & p \\
 & & & \vdots & 1 & & 0 & \vdots & & & & \\
 & 0 & & \vdots & & \ddots & & \vdots & & 0 & & \\
 & & & \vdots & 0 & & 1 & \vdots & & & & \\
\cdots & \cdots & \cdots & -\sin\theta & \cdots & \cdots & \cdots & \cos\theta & \cdots & \cdots & \cdots & q \\
 & & & \vdots & & & & \vdots & 1 & & 0 & \\
 & 0 & & \vdots & & 0 & & \vdots & & & & \\
 & & & \vdots & & & & \vdots & 0 & & 1 & \\
 & & & p & & & & q & & & &
\end{pmatrix}$$

Figure 3.1: Givens rotation matrix

In order to converge to a diagonal matrix, for each iteration k, we must nullify an element (not null) a_{pq}^k so that $a_{pq}^{k+1} = 0$. To reach this goal, we can choose θ such that $\cos 2\theta = \frac{a_{qq}^k - a_{pp}^k}{2a_{pq}^k}$, $-\frac{\pi}{4} \leq \theta \leq \frac{\pi}{4}$. The algorithm is given in Algorithm 3.

Algorithm 3 Algorithm of the Classical Jacobi method

Require: ϵ, A
Ensure: D (in case of convergence)
  $A_0 \leftarrow A$
  while $\sum_{i \neq j}^{n} {a_{ij}^{k+1}}^2 \leq \epsilon$ **do**
    Choose the a_{pq}^k having the highest magnitude, $a_{pq}^k \neq 0, p \neq q$
    Compute Q_k according a_{pq}^k
    Compute $A_{k+1} \leftarrow Q_k^t A_k Q_k$
  end while

3.2.4 QR method and Householder reflections

Hypothesis: A is a real matrix, $A \in M_N(\mathbb{R})$. We note $\lambda_1, \ldots, \lambda_N$ its eigenvalues such that $|\lambda_N| > |\lambda_{N-1}| > |\lambda_{N-2}| > \cdots > |\lambda_1|$.

QR method by means of Householder reflections

Let us consider a vector $v \in \mathbb{R}^N$, v not nul. The Householder matrix associated to v is $H_v = I - 2\frac{vv^t}{v^t v}$. This matrix is symmetric and orthogonal.

Now we consider a vector $x \in \mathbb{R}^N$, $x = \delta_0 v + \delta_1 a_1 + \delta_2 a_2 + \cdots + \delta_{N-1} a_{N-1}$ where

$(a_1, \ldots, a_{N-1})$ is an orthogonal basis of the hyperplane $span\{v\}^\perp$ and $\delta_i \in \mathbb{R}^* \forall i = 1, \ldots, N$. We have: $H_v x = -\delta_0 v + \delta_1 a_1 + \delta_2 a_2 + \cdots + \delta_{N-1} a_{N-1}$. It means that x and $H_v x$ have the same projection on $span\{v\}^\perp$ but opposite projections on v. Therefore, by choosing $v = x \pm \|x\|_2 e_1$ where e_1 is the unit vector $(1, 0, \ldots, 0)$, we get a method to nullify all the elements of x except the first one (i.e. $H_v x = \gamma e_1$, $\gamma \in R^*$).

Algorithm 4 Algorithm of the QR decomposition method

Require: A

Ensure: QR decomposition of A

$A_0 \leftarrow A$

for $i = 1, \ldots, N-1$, step=1 **do**

1) Compute H_i with $v = c_i \pm \|c_i\|_2 e_1$

c_i is the vector composed of the $N - i + 1$ last elements of the i^{th} column of A_{i-1}

2) Compute $A_i \leftarrow H_i A_{i-1}$

end for

$$\mathcal{A}_{i-1} = \begin{pmatrix} \bullet & \bullet & \cdots & & & \cdots & \bullet & \\ 0 & \bullet & \bullet & \cdots & & \cdots & \bullet & \\ 0 & 0 & \bullet & \bullet & & & \vdots & \\ \vdots & 0 & 0 & \bullet & \cdots & \cdots & & i \\ & \vdots & 0 & \bullet & \cdots & & & \\ & & \vdots & \vdots & \cdots & & \vdots & \\ & & 0 & \bullet & \cdots & \cdots & \bullet & \\ & & & i & & & & \end{pmatrix}$$

$$\mathcal{H}_i = \begin{pmatrix} I_{i-1} & 0 \\ 0 & H_i \end{pmatrix}$$

At the end of the algorithm described in Algorithm 4, $(H_{N-1} \ldots H_2 H_1)$ is orthogonal (and thus invertible) because it is the product of orthogonal matrices. We get a QR decomposition by choosing $R = A_{N-1}$ and $Q = (H_{N-1} \ldots H_2 H_1)^{-1}$. We have also $Q = (H_{N-1} \ldots H_2 H_1)^t$ because Q is orthogonal. Then, since each H_i is symmetric, we get $Q = H_1^t H_2^t \ldots H_{N-1}^t = H_1 H_2 \ldots H_{N-1}$. The QR decomposition requires $\frac{4}{3}N^3$ elementary operations. The computations are much faster if we previously transform A into an Hessenberg matrix.

Finally, it is proved that the sequence $k = 1, \ldots, N-1, A_{k+1} = {Q_k}^t A_k Q_k$ converges to an upper triangular matrix T whose eigenvalues are the diagonal elements. As Q is orthogonal, A and T are similar matrices and have the same eigenvalues.

QR method by means of Givens rotations

The Householder reflexions allow nullifying all the components under the diagonal. When A is a sparse matrix, it is much more interesting to nullify only non-nul elements by means of the Givens rotations (explained for the Jacobi method).

3.2.5 Tridiagonalization of a real symmetric matrix

Many methods aim to compute the eigenvalues of a real symmetric tridiagonal matrix T. The tridiagonalization can be performed by means of the Householder reflections or the Givens rotations. It consists in nullifying all the components of the upper/lower triangular sub-matrix except the diagonal and the first sub-diagonal. Then we get a symmetric Hessenberg matrix. It generally needs $\frac{4}{3}N^3$ floating point operations in order to tridiagonalize a dense matrix.

Hypothesis: A is a real symmetric matrix, $A \in M_N(\mathbb{R})$.

We can use also the Lanczos tridiagonalization method explained in Algorithm 6. It is the symmetric case of the Arnoldi method. We first have to choose an initial vector q_0. The method contains N iterations. It builds an orthogonal basis Q of the Krylov subspace $K^N(q_0, A) = \{q_0, Aq_0, A^2q_0, \ldots, A^{N-1}q_0\}$ such that $Q^tAQ \sim T$. Let us note d and e the diagonal and sub-diagonal of T. For $k = 1, \ldots, N$, each iteration k computes a new vector q_k of Q, d_k and e_k the new elements of T (if $k = N$ e_N is not computed). The Lanczos method suffers from a loss of orthogonality among the vectors q_k due to the rounding-off errors of the floating-point arithmetic. Therefore, we have to reorthogonalize the basis Q. We notice three techniques. The first one performs a full reorthogonalization of Q at each iteration. The second one makes a full-reorthogonalization only if a test is validated. It maintains a sufficient semi-orthogonality among the q_k (orthogonal to half the machine precision). The third one does a local reorthogonalization at each iteration but it generates spurious eigenvalues.

3.2.6 Bisection method

Hypothesis: T is a real symmetric tridiagonal matrix, $T \in M_N(\mathbb{R})$.

The Bisection method is also called Givens method or Sturm sequence method. It computes all the eigenvalues of a real symmetric tridiagonal matrix T which are contained in a given interval. If this interval is the Gerschgorin domain, then all the eigenvalues are computed. The Gerschgorin theorem says that each eigenvalue of a matrix $A \in M_N(\mathbb{R})$ respects the criterion $|\lambda_i - a_{ii}| \leq \sum_{j=1, j\neq i}^{N} |a_{ij}|$, for $1 \leq i \leq N$. Thus all the eigenvalues are in the interval $[\min_{1\leq i\leq N}(a_{ii} - r_i),\ \max_{1\leq i\leq N}(a_{ii} + r_i)]$.

Algorithm 5 Algorithm of the Lanczos tridiagonalization method

Require: A, q_0 (initial vector)

$q_{candidate} \leftarrow q_0$
$e_0 \leftarrow \|q_{candidate}\|_2$
for $k = 1, \ldots, N$, step=1 **do**
1) New column of the basis Q
$q_k \leftarrow \frac{q_{candidate}}{e_{k-1}}$
2) Candidate column for the $k+1^{th}$ iteration
$q_{candidate} \leftarrow Aq_k$
$q_{candidate} \leftarrow q_{candidate} - q_{k-1}e_{k-1}$
3) New element of the diagonal
$d_k \leftarrow q_k^t q_{candidate}$
$q_{candidate} \leftarrow q_{candidate} - q_k d_k$
4) Reorthogalization (e.g. full)
$q_{candidate} \leftarrow q_{candidate} - Q^t Q q_{candidate}$
5) New element of the sub-diagonal (done if $k \neq N$)
$e_k = \|q_{candidate}\|_2$
end for

The inertia of a real matrix A is the triplet of integers $(\nu(A), \zeta(A), \pi(A))$ which depicts: the number of negative eigenvalues, the number eigenvalues that are zero and the number of positive eigenvalues. Let $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N$ be the eigenvalues of A. $\forall \sigma \in \mathbb{R}$, $\pi(A - \sigma I)$ represents the number of eigenvalues of A upper than σ. Thus, $\pi(A - \sigma I) = i$ means that $\lambda_1 \geq \cdots \geq \lambda_i \geq \sigma \geq \lambda_{i+1} \geq \cdots \geq \lambda_n$.

By choosing suitable values of σ, we can gradually share the starting interval in order to isolate each eigenvalue into a sub-interval whose diameter is the precision of the bisection algorithm (so, we compute approximated eigenvalues).

In order to compute $\pi(A - \sigma I)$, we use the inertia low of Sylvester: if A is a real symmetric matrix and X a real non-singular matrix, then A and XAX^t have the same inertia. In other words, $\nu(A) = \nu(XAX^t)$, $\zeta(A) = \zeta(XAX^t)$, $\pi(A) = \pi(XAX^t)$. If A is real symmetric, $(A - \sigma I)$ too. Besides a real symmetric matrix is orthogonally diagonalizable. It means that we can do the decomposition $A = PDP^t$ where D is a diagonal matrix and P an orthogonal lower triangular matrix. Therefore, there exist D and P such that $(A - \sigma I) = PDP^t$. $(A - \sigma I)$ and D have the same eigenvalues and due to the Sylvester law, $\pi(A - \sigma I)$ is exactly the number of positive eigenvalues. Those eigenvalues are the elements of D. If we consider that A is actually the real tridiagonal symmetric matrix T, we get:

$$\mathcal{T} - \sigma I = \begin{pmatrix} \alpha_1 - \sigma & \beta_1 & 0 & \cdots & 0 \\ \beta_1 & \alpha_2 - \sigma & \beta_2 & \ddots & \vdots \\ 0 & \beta_2 & \alpha_3 - \sigma & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \beta_{n-1} \\ 0 & \cdots & 0 & \beta_{n-1} & \alpha_n - \sigma \end{pmatrix}$$

The numerical translation of $(T - \sigma I) = PDP^t$ gives the following recurrence:

$$\begin{cases} d_1 = \alpha_1 - \sigma \\ \text{for } i = 1, \ldots, n-1 \\ \qquad d_{i+1} = (\alpha_{i+1} - \sigma) - \frac{{\beta_i}^2}{d_i} \end{cases}$$

The Bisection method requires $O(Nk)$ floating point operations with k the number of eigenvalues in the starting interval. If we want to compute the eigenvectors, we have to use the Inverse Iteration method and it also adds $O(Nk)$ floating point operations.

3.2.7 Cuppens or Divide & Conquer method

Hypothesis: T is a real symmetric tridiagonal matrix, $T \in M_N(\mathbb{R})$.

This method consists in finding the decomposition $T = QDQ^t$ where D is a diagonal matrix and Q an orthogonal matrix. This method differs from the others since it is recursive. The "divide" step recursively shares the decomposition problem into 2 sub-problems of smaller orders. The "conquer" step gathers the two sub-results and computes the result of the full decomposition. In [58], the authors explain it in details. It is one of the fastest method. It needs $O(N^2)$ floating point operations ($\frac{4}{3}N^3$ are also required for the tridiagonalization).

3.3 Parallelization of the real eigenproblem

The parallelization of the real eigenproblem has generally been studied for High Performance Computing on supercomputers and dedicated clusters. We first focus on the Householder reductions and the Givens rotations which are used in many situations (QR factorization, Jacobi method, tridiagonalization, etc). Thus, there is a significant related work on those methods. Then, we look at the parallel tridiagonal eigenproblem. It has been also heavily studied, in particular the Bisection method. Thirdly, we notice a recent dynamic topic of research consisting of parallelizing the Krylov subspace methods

for large scale eigenproblems. We conclude this section with work on the real eigensolver for platforms of Global Computing.

3.3.1 Parallelization of the Householder reflections and the Givens rotations

Parallel QR factorization by means of the Givens rotations

In [55], the authors parallelize the QR method by means of the Givens rotations. They target sparse matrices and design their algorithm so as to minimize the storage space needs. The algorithm uses the PVM message passing model on the distributed memory Cray T3D supercomputer. [56] also focuses on the QR parallelization. Instead of the traditional Givens method, the authors use the fast-Givens one. It consists in keeping the current matrix in a factored form DA where D is a diagonal matrix. The targeted experimental environment is also dedicated to high performance since it uses PVM and the BLACS (communication library for the BLAS) on a dedicated cluster of workstations. Experimentations were planed on a supercomputer too (Meiki CS-2).

Parallel tridiagonalization by means of Householder reductions

By using the Householder reflections, [54] aims to parallelize the tridiagonalization of dense matrices. The main effort concerns the data mapping on the processors following a square-torus topology. The targeted platform is composed of an Intel Paragorn supercomputer with the SUNMOS operating system (dedicated to massively parallel-distributed memory systems). In [53], the authors also perform a parallel tridiagonalization by means of the Householder reflections. Actually, the goal of the authors is to do a QL diagonalization in order to find all the eigenvectors (we have previously presented the Bisection and the Divide & Conquer methods for the tridiagonal eigenproblem). The MPI message passing model is used on the IBM SP2 supercomputer.

3.3.2 Parallelization of the tridiagonal eigenproblem

Parallelizing the Bisection method

[52] handles the tridiagonal eigenproblem by means of the Bisection method. The authors choose to use the multi-section strategy in order to isolate each eigenvalue and then their extraction is done using the parallel bisection. The Inverse Iteration method allows computing the eigenvectors. For the experimentations, we notice the usage of two supercomputers: Alliant FX/8 and Cray X-MP/48. In [51], the authors study in details the parallel bisection and the multi-section performances for the hypercube multiprocessors Intel iPSC-2. They show that choosing between one of those strategies depends on

some parameters: order of the problem, number of processors, ratio of arithmetic costs to communication costs.

Parallel Divide & Conquer methods

[58] presents the parallelization of the Divide & Conquer Cuppens method. Parallelizing this recursive method is much more challenging for distributed than shared memory systems. The authors performed a two dimensional block cyclic distribution of data. A drawback of their implementation is the high memory usage ($2n^2$). The experimentations are done on an IBM SP2 and also a dedicated cluster of workstations.

3.3.3 Parallelization of the Krylov subspace methods

In [57], the authors present the PLANSO package which is based on the sequential LANSO implementation. It uses the MPI message passing model and is tested on a supercomputer Cray T3E or on a cluster of multiprocessors (28 UltraSPARC workstations). We notice also the Parallel ARPACK package. It is the parallel version of ARPACK[1] which implements the implicit restarted LANCZOS and ARNOLDI method (IRAM). It uses the BLASC and MPI and, works on many parallel-vectorial supercomputers (Cray-C90, Cray T3D, Intel Delta, CM-200 and CM-5) and clusters of workstations. [59] presents the Multiple Explicitly Restarted Arnoldi Method (MERAM) which is based on the Explicitly Restarted Arnoldi Method (ERAM). It proposes also an asynchronous version of MERAM. A comparison is done with other methods like IRAM. The experimentations combine a cluster of workstations and a CM5 parallel machine.

3.4 Parallel eigensolver for Global Computing

In [59], the authors underline that the asynchronous MERAM is a good candidate method for Global Computing. In [60], this method is distributed by means of the RPC programming software Netsolve on two geographic sites interconnected by Internet. To our knowledge it is one of the rare studies of the eigenproblem on a Global Computing platform. However, we benefit from a significant work on the matrix-vector product (MVP) in [61]. The authors focus on several block partitioning strategies. They simulate data persistence and study the benefit of an out-of-core mechanism. In this work, the matrices are not transferred but generated on each computing node. It is a strong hypothesis because the cost of communication over the Internet is a major issue in Global Computing.

[1] http://www.caam.rice.edu/software/ARPACK/

3.5 Chosen numerical methods

In order to solve the real symmetric eigenproblem, we choose a numerical method composed of two main steps. We first tridiagonalize the real symmetric matrix A by means of the Lanczos tridiagonalization method. Secondly, we solve the tridiagonal symmetric eigenproblem of the matrix T with the Bisection method. The Inverse Iteration method is used to compute the eigenvectors of T which are necessary in order to find those of A. Our motivations for those numerical methods are given in Chapter 4.

In order to distribute the Lanczos method, we take into account the study of the large scale distributed MVP in [61]. For the distribution of the Bisection method, followed by the Inverse Iteration, we focus on the parallel Bisection. In addition, we use as far as possible the optimized BLAS and LAPACK libraries.

Chapter 4

The Real Symmetric Eigenproblem on Global Computing Platforms

4.1 Scope of the study

Global Computing does not compete High Performance Computing. We must consider it as a complementary scientific tool. The choice between Global and High Performance Computing depends on several factors and also on the targeted criterion of performance. Currently, supercomputers and powerful clusters are much better in order to reach high Flop/s. They are also relevant for all applications requiring confidentiality, reliability or real-time constraints.

Considering the huge number of computers connected to Internet, Global Computing may provide a solution to the applications needing a large amount of storage space. This point can be a critical factor for High Performance Computing devices. Next, very large numerical problems are often memory- or CPU-intensive. They can also be distributed among volunteer machines of a Global Computing platform. Nevertheless, the computing time must not be a criterion of the user. The most promising aspect of Global Computing is the availability of the volunteer machines. This availability provides several advantages. First, it offers a significant amount of resources to people who have no access to high performance devices. This problem of access is explained by the expensive cost of dedicated machines and also simply by the limited and restricted usage of supercomputers. Second, Global Computing provides a continuous availability of computing resources. It is an "on demand" access scheme. In other words, people use volunteer machines only when needed. Thus, for punctual use, it avoids buying some devices that would not be used again later. Even owners of high performance computers may be concerned by this last point.

In this thesis, our precise goal regarding Global Computing is to solve very large numerical problems. In particular we focus on the real symmetric eigenproblem. This choice is mainly due to the importance of this numerical problem in many industrial applications. The page ranking method of the Google web-search engine is a famous example of eigenproblem usage. The eigenproblem is also a resource-demanding problem. It requires CPU cycles, main memory, disk space and it generally generates much communication. Thus, it is a good choice of study because it emphasizes the main challenges that we must handle in order to use Global Computing for large computations. We propose some solutions to most of those challenges. We underline also few problems that should be optimized. The overall contribution is to propose a Global Computing programming methodology for complex linear algebra problems, not restricted to the real symmetric eigenproblem. In short term, we would like to encourage researchers to perform similar studies. In long term, many companies may use Global Computing platforms for solving their large applications.

4.2 Parametric-parallel applications

We first present the parametric-parallel paradigm which is the simplest one for Global Computing. This paradigm is usually called task-farming. It can be advantageously used in order to solve the real symmetric tridiagonal eigenproblem.

4.2.1 Definition and interests of the parametric-parallelism

A parametric-parallel application is composed of a single sequential source code and several sets of parameters. Distributing such an application consists in sending to each volunteer node the same program but a different input set of parameters. Each volunteer node computes a specific output result without any communication with the other nodes. By gathering all the output results, we get the global solution.

By nature, the parametric-parallelism respects a distributing computing model. The distributed model allows sharing the CPU and memory workload and also the disk space usage. The parametric-parallelism uses a parallel computing model only if several tasks are sent to volunteer nodes and then solved in parallel. This point must be clearly underlined although it seems obvious afterwards. The parallel model provides faster computations. We notice some specific points of interest of the task-farming model for Global Computing. First, the absence of communication between the tasks is fundamental because world-wide computing uses the Internet network which has poor performances and is not reliable. The less we use it, the better. Second, the volunteer peers do not need to know each other. It greatly simplifies the scheduling management of the deployment Global Computing tool. The data management is also very easy since pieces of data associated to the tasks are

independent. Finally, from the user point of view, developing this kind of application is relatively easy because of the absence of task- and data-flow.

Many parametric-parallel and distributed applications are famous because they were precursor in Global Computing and related to dynamic or popular topics of research (such as bio-informatics). Section 2.6.1 gives some examples.

4.2.2 The real tridiagonal symmetric eigenproblem

This linear algebra problem consists in computing the eigenpairs (λ,v) of a real symmetric tridiagonal matrix T such that $Tv = \lambda v$ where $T \in M_N(\mathbb{R})$, $v \in \mathbb{R}^N$ and $\lambda \in \mathbb{R}$.

4.2.2.1 The Bisection method

In Section 3.2.6, we have presented the sequential Bisection method. Many points motivate the choice of the Bisection method among all the existing techniques. First, it is a fast method. It needs $O(Nk)$ floating point operations in order to compute k eigenvalues in the starting interval. Besides, we must add $O(Nk)$ floating point operations for the Inverse Iteration method in order to compute the eigenvectors. The Bisection method differs from the other ones since it relies on a recurrence and not on expensive matrix computations such as matrix-vector products. Then, the Bisection is mainly a CPU-intensive method and is not very memory-intensive. This property is very interesting in the context of Global Computing using standard PCs. In fact, the commercial trend of the last decade associates fast CPU with small amount of memory. Last but not least, the Bisection is naturally a good candidate to task-farming since it shares the starting interval into subintervals and each subinterval can be handled independently.

4.2.2.2 Propositions to adapt the Bisection method to Global Computing

The natural parametric-parallelism of the Bisection

In Section 3.3.2, we cite some work done in High Performance Computing in order to parallelize the Bisection. They have shown the effectiveness of the Parallel Bisection on a distributed-memory system whereas multi-section is better for shared-memory systems. Thus we target to adapt Parallel Bisection to Global Computing.

The Parallel Bisection is a typical parametric-parallel application. Indeed, let us consider the sequential Bisection and Inverse Iteration programs, and I, a starting interval of research containing the targeted eigenvalues. Then, let us make a strict partition of I such as $I = \cup I_j$, $\emptyset = \cap I_j$, $j = 0 \ldots p-1$. We can build p independent tasks composed

of the sequential program and one of the subinterval I_j. Each task can be handled by a computing node which computes without any communication the eigenvalues (and the associated eigenvectors) contained in its assigned subinterval.

A solution to optimize the load balancing

Once we propose using the Parallel Bisection, the following important issue is how to share the starting interval. In fact, it has a great impact on the workload balancing among the computing nodes. In particular, if a matrix has a cluster of eigenvalues in one interval of the partition, only one computing node has a significant work to perform. Figure 4.1 gives an example of a naive partition ($I_j = [c_j, c_j + \frac{b-a}{p}[$ where $I = [a, b[$, $c_0 = a$ and $c_{p-1} + \frac{b-a}{p} = b$) in presence of clustered eigenvalues.

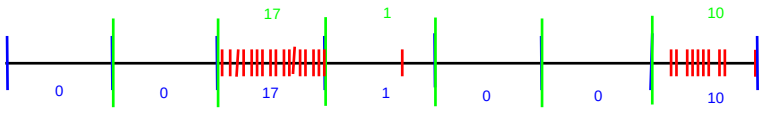

Figure 4.1: Naive partition of an interval with 28 eigenvalues

Therefore, we propose to take into account the distribution of the eigenvalues in order to build a suitable partition. It is done by means of the Bisection recurrence itself combined with a given threshold. This threshold specifies the maximum number of eigenvalues per subinterval. We apply the sequential Bisection algorithm on the starting interval but stop it as soon as each subinterval contains less eigenvalues than the threshold. Figure 4.2 illustrates this process in presence of the same clustered eigenvalues as Figure 4.1.

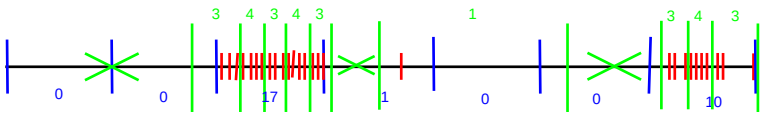

Figure 4.2: Balanced partition of an interval with 28 eigenvalues, threshold=4

If the partition does not respect the mathematical definition, we can loose eigenvalues or compute some eigenvalues several times. In [50], the authors stress the problem of the non-monotonicity of the function which computes the Bisection recurrence. It motivated us to centralize the computation of the partition on the client node.

The threshold is an important parameter and it could be an additional topic of study. A too big value does not divide clusters of eigenvalues. A too tiny value generates a huge number of subintervals and then too many tasks. We currently make an empirical choice and typical values are between 200 and 500. We do not recommend assigning one subinterval per computing node. Indeed, if the number of subintervals is large, it may give poor ratios between the computing times and the communication times (in order to submit the tasks and get the results). Thus, we consider a second parameter which is the number of requests sent to the nodes. We perform a cyclic distribution of the subintervals

among the requests in order to balance the workload (giving connected subintervals to the same request would annihilate our effort to divide clusters of eigenvalues).

4.2.2.3 Experimentations on world-wide grids with OmniRPC and XtremWeb

Software of Global Computing

We present in Chapter 2 many tools designed to distribute and parallelize applications. Among all of them, we focus on OmniRPC and XtremWeb. Several reasons motivate those choices. XtremWeb targets Peer-to-Peer Computing by means of the work-stealing model and OmniRPC focuses on Grid Computing with the RPC programming model. The two computing models are dominant in Global Computing and it is interesting to deploy the same application on them. XtremWeb and OmniRPC greatly differ and each one proposes interesting features:

- According to the peer-to-peer principle, the relationship between a client and all the volunteers is anonymous and can be reversed. Nevertheless, it remains a central authority with a third entity: the dispatcher. The dispatcher collects requests from the client, schedules the requests when a volunteer asks for a job and it gets the results from the volunteers. With OmniRPC, there is an obvious unidirectional relationship between the client and the workers. OmniRPC proposes two configurations: "SSH" and "Cluster". The "SSH" configuration requires the client to know each worker.
- XtremWeb uses connection-less protocols, is fault-tolerant and uses a sandboxing technique whereas OmniRPC relies on connected communication and is not fault-tolerant.
- OmniRPC offers data persistence, proposes to choose between several kinds of schedulers (e.g. round robin, PBS) and several invocation methods (e.g. RSH, SSH). XtremWeb lacks data persistence and the scheduling is very simple.
- A major difference is the way to handle the parameters at the client API level. Due to the low level RPC programming mechanism, the client of OmniRPC loads the input and output parameters in main memory. On the contrary with XtremWeb, the client stores the parameters in files. The files are zipped, serialized (XtremWeb uses Java RMI) and sent.
- The way to reach parallelism is very different between OmniRPC and XtremWeb. OmniRPC enables parallel computing by means of direct asynchronous RPC calls on computing nodes. The XtremWeb client also performs asynchronous submissions of tasks but the requests are kept by the dispatcher. Parallelism is reached only if several volunteers ask for a job in parallel. If they ask in a sequential way, the application is only distributed. We underline that OmniRPC proposes a second kind of parallelism because it is thread-safe and allows running OpenMP jobs.

In spite of all those differences, we notice a similar interesting property. XtremWeb handles firewalls by using the "pull model". It means that communication is initiated by the volunteer nodes and the client, and never by the dispatcher. OmniRPC can also harness nodes on a protected site as long as SSH connections are allowed on a relay server ("cluster configuration").

Finally, our choice for these software solutions is motivated by their support in case of technical questions or requests for improvement. Indeed XtremWeb is developed by a team of the INRIA Grand Large project. OmniRPC is designed at the HPCS laboratory of the University of Tsukuba where I stayed 10 months thanks to the French-Japanese Doctoral Consortium (CDFJ) and the FJ-Grid associated team of the Grand Large project.

Computing and network resources

We harness heterogeneous computational nodes on two different geographic sites at the USTL, France, and at the University of Tsukuba, Japan. Details are given in Table 4.1.

Number of PCs	CPU	2^{nd}cache/proc (KB)	Memory (MB)
Network of workstations at the USTL			
14	Pentium 3, 450MHz	512	128
14	Pentium 4, 3.2GHz	1024	1024
22	Celeron, 2.4GHz	128	512
14	Celeron, 2.4GHz	128	1024
28	Celeron, 2.2GHz	128	512
8	Celeron, 2GHz	128	512
8	Celeron, 1.4GHz	256	256
OS Debian 4.0.2/3.3.6, kernel 2.6.13/2.6.11, gcc 3.3.5/3.3.6			
Clusters at the University of Tsukuba			
16	Dual Xeon, 3GHz	512	1024
8	Dual Xeon, 3GHz	2048	1024
2	Dual Xeon, 3.6GHz	1024	4096
OS Linux, kernel 2.6.9, gcc 3.3.5			

Table 4.1: Computational resources at the USTL and at Tsukuba

Our experimental platforms use different kinds of networks like Local Area Networks (LAN) and the Internet. At the USTL, we have a Network Of Workstations (NOW). The bandwidth varies from 10 to 100 MBits in our laboratory and it is 1GBits in the USTL. At the University of Tsukuba, the nodes are organized within two clusters. The bandwidth is 100 MBits or 1 GBits between the nodes and the switches. It is 1 GBits between the local switches and the Tsukuba WAN. Table 4.2 shows the average of 50 measurements of bandwidth over TCP and UDP with the tool Netperf between Lille and Tsukuba. The measurements are done at different times of the day. The two targeted Global Computing middleware for the experimentations use communication over TCP.

	Over TCP	Over UDP
Average of 50 measurements (MBits)	1.74	8.77

Table 4.2: Bandwidth between the USTL and Tsukuba

The experimental computing and network resources are realistic. In other words, they are representative of Global Computing resources. In fact, the computational resources are greatly heterogeneous. The architectures of the two LAN are also different and we use communication through the Internet. Third, the conditions of usage differ between the two sites. At Tsukuba, a reservation policy is set up and the nodes are dedicated to computational science. It guaranties a constant amount of resources, availability and stability of nodes. On the other hand, the USTL does not use a reservation policy and the machines are shared with students. So, network bandwidth, available memory and CPU cycles may vary greatly. Machines are also often switched off.

Experimental platforms

Table 4.3 presents the six experimental platforms that we build by means of OmniRPC or XtremWeb and, the computing and network resources. We use the "cluster configuration" of OmniRPC. Each OmniRPC relay server uses a simple round robin scheduler, SSH invocations and multiplexes all communications into only one.

	Config. 1	Config. 2	Config. 3
Middleware	XtremWeb		
Location of client	Lille		
Location of dispatcher	Lille		
Nb and location of workers	50 USTL	50 University of Tsukuba	24 USTL 26 University of Tsukuba

	Config. 4	Config. 5	Config. 6
Middleware	OmniRPC		
Location of client	Lille		
Nb and location of OmniRPC servers	2 at Lille	2 at Tsukuba	1 at Lille 1 at Tsukuba
Nb workers per server	25		
Nb and location of workers	50 USTL	50 University of Tsukuba	25 USTL 25 University of Tsukuba

Table 4.3: Experimental platforms for the study of the parametric-parallelism

Parameters and hypothesis

We use the tridiagonal matrix T described in Figure 4.3 whose eigenvalues are $\lambda_j = a + 2b\cos(\frac{j\pi}{N+1})$ where $j = 0, \ldots, N-1$. We choose $a = 2$, $b = 1$, $N = 10^5$, a threshold of 200 eigenvalues maximum per subinterval. We build 100 independent tasks (i.e. 100

sets of subintervals). With OmniRPC, we directly submit one task per worker. With XtremWeb, all the tasks are given to the dispatcher and the volunteer nodes have to collect them (in parallel or not).

$$\mathcal{T} = \begin{pmatrix} a & b & 0 & \cdots & \cdots & 0 \\ b & a & b & 0 & \cdots & \vdots \\ 0 & b & a & b & \ddots & \vdots \\ \vdots & 0 & b & \ddots & \ddots & 0 \\ \vdots & \vdots & \ddots & \ddots & \ddots & b \\ 0 & \cdots & \cdots & 0 & b & a \end{pmatrix}$$

Figure 4.3: Matrix for the study of the parametric-parallelism - N=100000, a=2, b=1

For this study on the parametric-parallelism paradigm, we have two major assumptions. First, the matrix is centralized on each computational node. We do not yet deal with matrix transfers. They are handled in Section 4.3. Second, the eigenvectors are computed but not yet retrieved by the client. This problem is also handled in Section 4.3. The computational nodes can use the eigenvectors to perform additional computations and then to return results monopolizing limited network bandwidth. For instance, we currently compute the Euclidean norm of the residuals.

Evaluation of the memory and disk needs

Our program requires at the most $162\,MB$ $((96+8*threshold)*N)$ of memory in order to solve one task. The storage need is not significant in this section. In fact, the tridiagonal matrix and the N eigenvalues easily fit in memory and, we do not store the eigenvectors (see the previous hypothesis).

4.2.2.4 Results and analysis

Results

For each configuration, the times are given in seconds. Each time is computed by means of 10 runs. So, each wall-clock time of the client in Table 4.5 is an average of 10 measurements. Each computing time of the remote nodes in Table 4.4 is the average of 1000 measurements (10 runs of 100 tasks).

In Table 4.4, we give couples of computing times. A couple (a, b) means that a (resp. b) is the computing time for the eigenvalues (resp. eigenvectors) by means of the Bisection method (resp. Inverse Iteration). We also provide the minimal remote computing times of each configuration for the eigenvalues and eigenvectors computations. It emphasizes the heterogeneity of the nodes.

	Config. 1	Config. 2	Config. 3
Average time	(213, 928)	(112, 755)	(198, 980)
Minimal time	(142, 467)	(63, 250)	(107, 491)

	Config. 4	Config. 5	Config. 6
Average time	(192, 728)	(138, 930)	(180, 801)
Minimal time	(140, 429)	(79, 325)	(95, 311)

Config. reminder	XtremWeb: 1-2-3,OmniRPC: 4-5-6 Lille: 1-4, Tsukuba: 2-5, Lille & Tsukuba: 3-6
	(a, b) = (time for eigenvalues, time for eigenvectors)

Table 4.4: Computing times of the remote nodes to find their eigenpairs (in seconds)

	Config. 1	Config. 2	Config. 3
Times of submission	1	1	1
Times of wait	4074	2739	2735
Times to retrieve results	2	2	1

	Config. 4	Config. 5	Config. 6
Times of submission	1384	1607	1392
Times of wait and to get results	1165	1809	1463

Config. reminder	XtremWeb: 1-2-3,OmniRPC: 4-5-6 Lille: 1-4, Tsukuba: 2-5, Lille & Tsukuba: 3-6

Table 4.5: Wall-clock times of the client to get the eigenvalues (in seconds)

Analysis

First, we focus on the computing times of the remote nodes depending on the Global Computing software. With XtremWeb, the eigenvalues and the eigenvectors are computed faster when we increase the number of workers at the University of Tsukuba and decrease this number at the same time at Lille (on dual-processor computers at Tsukuba, there is one XtremWeb worker per processor). In fact, the computers at Tsukuba are more powerful than at Lille. With OmniRPC, this conclusion is also true for the computations of the eigenvalues. On the contrary, the computations of the eigenvectors with OmniRPC take much more time at Tsukuba than at Lille. We harness dual-processor computers at Tsukuba and each one has two OmniRPC workers (one per processor). Thus, the main memory has to be shared. On those machines at Tsukuba, may be the Java Virtual Machine manages well memory for two XtremWeb workers, whereas the two OmniRPC workers compete in order to get some memory. The other shared resources like the hard disk or the bandwidth cannot explain this behaviour for our program. As the Bisection is mainly a CPU-intensive application, it does not suffer from this problem.

Second, we still consider the computing times of the remote nodes but we fix the location of the workers. At Lille, the OmniRPC nodes compute faster than the XtremWeb workers. We explain this difference of performance by the overhead of the Java-based workers of XtremWeb whereas the OmniRPC nodes are simply invoked by RPC calls. At the University of Tsukuba, we observe the opposite situation: the XtremWeb workers perform better than the OmniRPC ones. As previously, on a dual-processor system, the management of memory among two workers is probably more efficient by means of the JVM.

Third, we focus on the times of the client. They are wall-clock times because they include communication times and remote computing times. We observe a major difference of behaviour between OmniRPC and XtremWeb which is due to the connected/disconnected strategies. On the one hand, XtremWeb uses a disconnected model. The client submits its 100 tasks to the dispatcher and it immediately starts waiting for the results. On the other hand, OmniRPC relies on a connected model. The client submits as many tasks as there are available nodes (asynchronous RPC calls) and then it starts an active wait. The remaining tasks are gradually submitted as soon as some workers are available.

Finally, if we do not consider the heterogeneity of computational nodes, the location of workers has not a significant impact on the results since we implement parametric-parallelism. We only need to send input parameters and get the results at the end. Besides, the volume of communication is low. Nevertheless, without the two assumptions formulated in Section 4.2.2.3, the client would have to send the matrix ($1.5\,MB$) and, above all, it would have to get the eigenvectors (a total of $74.5\,GB$). Such data transfers would have a considerable impact on tests.

4.2.2.5 Conclusion and perspectives

In this first part of the chapter, we show the interest of the parametric-parallelism. It is a suitable paradigm for Global Computing involving computing nodes connected by the Internet. In fact, there are only two communications per task: the volunteer node gets its task and then it sends the results. So, if the volume of data for those two communications is moderate, the location of the computing nodes almost does not matter. The client can harness all over the world the most suitable nodes for his application. He can also take advantage of the time difference between the countries in order to find more idle computers. Depending on the Global Computing software and its configuration, some volunteer nodes are not suitable. For instance, two OmniRPC workers do not give optimal performances on a dual-processor computer. However, the performances remain acceptable for our experimental application because it does not require a lot of memory. The client can avoid this problem because it configures himself the OmniRPC platform. By default, XtremWeb launches one worker per processor. The results are good on a dual-processor system for the Bisection and the Inverse Iteration method. Nevertheless, with a memory intensive application, we have to be cautious for multiprocessors whose

main memory is not scaled to the number of processors. Besides, we have no reference on the behaviour of that Global Computing software on shared-cache multi-core systems.

From the eigenproblem point of view, this section shows that Parallel Bisection can be efficiently used for Global Computing as it was previously done for High Performance Computing. A natural perspective is to use the Parallel Bisection as a sub-problem of the real symmetric eigenproblem. It consists in dividing the main problem in two steps: tridiagonalization and tridiagonal symmetric eigenproblem. Work of this section is useful for the second step. We have also stressed the existence of clusters of eigenvalues and we propose a solution which divides them and balances the workload among the workers.

From a methodological point of view, in order to distribute and parallelize an application over the Internet, a good approach would consist in searching task-farming capabilities into sub-problems in order to get a parallel application with a minimum of communication. This preliminary analysis does not depend on any Global Computing software since most of them can perform parametric-parallelism. Unfortunately, very few linear algebra methods allow using this parallel paradigm. Second, a parametric-parallel sub-problem is interesting only if the client designs workload-balanced tasks (not necessary true if the workers are too heterogeneous). Otherwise, the parametric-parallel program lasts as long as the sequential one and it adds communication. Then, once the client has designed the parametric-parallel subprograms, it is important to evaluate the memory requirements of the independent tasks and the volume of data for the input and output communications. Indeed, if the Global Computing software can take into account such information (or if the client builds his platform), it can select the most suitable volunteer workers (architecture, location).

4.3 The real symmetric eigenproblem on Global Computing platforms

The parametric-parallelism is a very specific paradigm which can be applied to very few linear algebra problems. Most of complicated applications have data dependencies which generate much communication and synchronization points. Moreover, the previous Bisection and Inverse Iteration methods only concern the real symmetric tridiagonal eigenproblem. Therefore, in this section we present our study on the real symmetric eigenproblem of a real symmetric matrix A. It consists in finding some eigenpairs (λ,u) such that $Au = \lambda u$ where $A \in M_N(\mathbb{R})$, $u \in \mathbb{R}^N$ and $\lambda \in \mathbb{R}$. We proceed in two steps. We first tridiagonalize the matrix A and then we take advantage of the task-farming capabilities of the Bisection in order to find the eigenvalues of A. This study is challenging for Global Computing because several sub-problems are CPU and memory intensive and sometimes, they bring up the question of data storage. They also generate communication and synchronizations.

4.3.1 Numerical method

4.3.1.1 Tridiagonalization with the Lanczos method

In Section 3.2.5, we explained the principle of the Lanczos tridiagonalization. It computes an orthonormal basis whose vectors compose a matrix Q, and a real symmetric tridiagonal matrix T such that $T = Q^t A Q$, where A is a real symmetric matrix, $A \in M_N(\mathbb{R})$.

Algorithm 6 Reminder of the algorithm of the Lanczos tridiagonalization method

Require: A, q_0 (initial vector)
$q_{candidate} \leftarrow q_0$
$e_0 \leftarrow \|q_{candidate}\|_2$
for $k = 1, \ldots, N$, step=1 **do**
1) New column of the basis Q
$q_k \leftarrow \frac{q_{candidate}}{e_{k-1}}$
2) Candidate column for the $k+1^{th}$ iteration
$q_{candidate} \leftarrow A q_k$
$q_{candidate} \leftarrow q_{candidate} - q_{k-1} e_{k-1}$
3) New element of the diagonal
$d_k \leftarrow q_k^t q_{candidate}$
$q_{candidate} \leftarrow q_{candidate} - q_k d_k$
4) Reorthogalization (e.g. full-reorthogalization)
$q_{candidate} \leftarrow q_{candidate} - Q^t Q q_{candidate}$
5) New element of the sub-diagonal (done if $k \neq N$)
$e_k = \|q_{candidate}\|_2$
end for

The main motivation for Lanczos concerns the data access pattern. In fact, the matrix A is accessed through matrix-vector products (MVP). Q is also accessed by means of MVP during the full-reorthogonalization. Thus, we do not have to load the full matrices or block of matrices in memory and we can load the components by using many simple strategies. Besides, the other operations of the Lanczos method are basic and we can use existing optimized libraries like BLAS. Last but not least, the Lanczos tridiagonalization has a known by advance number of iterations. Among all the reorthogonalization methods, we choose the full-reorthogonalization, at each Lanczos iteration, in order to maintain a full orthogonality among the vectors of the Krylov basis. It adds many floating-point operations but it is very important to maintain a good stability of the numerical results and the systematic full-reorthogonalization is the best way to ensure it. Besides, the workload and the data storage can be distributed among volunteer nodes.

4.3.1.2 Numerical method of the real symmetric eigensolver

The sequential numerical method that we study in order to solve the real symmetric eigenproblem actually contains five steps. We first tridiagonalize A as explained in Section 4.3.1.1. Once we have computed the tridiagonal matrix T, we can apply the Bisection on it in order to find the eigenvalues of T (and A). The eigenvalues of A are often called the Ritz eigenvalues. Then, we use the Inverse Iteration method to compute the eigenvectors of T. Each Ritz eigenvector u of A is computed with the product $u = Qv$, where Q is the orthogonal Krylov basis found by Lanczos, v an eigenvector of T such that $Au = \lambda u$ and $Tv = \lambda v$. Finally, we check the accuracy of the computations by means of the Euclidean norm of the residuals $\|Au - \lambda u\|_2 (< \varepsilon)$.

4.3.2 Proposed distributed and parallel real symmetric eigensolver

In this section and the following ones, we present the distributed and parallel real symmetric eigensolver based on the previous numerical method. We first make global propositions which are middleware- and programming language-independent: they mainly concern the parallel paradigms, the data structures and the distribution of data. Second, we consider some constraints of Global Computing like volume of communication, nodes with limited resources, matrix distribution and so on. We first find some middleware-independent solutions. Then, we also glance at useful features of the Global Computing software that we can use.

Step 1: distributing the Lanczos tridiagonalization

We have previously explained that a motivation for the Lanczos tridiagonalization is that the matrices are accessed through matrix-vector products (MVP). Each one of the N Lanczos iterations performs one MVP involving A and a full-reorthogonalization composed of MVP involving Q. The distribution of the computations of the MVP depends on the matrix storage management. At the beginning, A must be stored by the client in a compact format since it is impossible for him to store $N * N$ elements of A. Q is gradually built during the tridiagonalization: one column per iteration.

We propose to send A only one time at the beginning of the problem. Indeed, at each Lanczos iteration, the same matrix A is used and sending such data is a very expensive task. Thus, the workers must keep data of A. As A is not necessary a sparse matrix, using a compact format during the computations would give high access time. So for the computations, the workers save data of A in a file with a non-compact format. We choose the binary format of file, instead of text, in order to save disk space (statement true for this kind of data, not in general) and to load quickly in memory matrix lines. We give details in part 4.3.3.3. We do not send the full matrix A to each volunteer worker

involved in the MVP. Indeed, each worker only needs selected data of A. Besides, a worker can hardly store the full matrix A in a non-compact format when N is large. Therefore, data of A is shared by blocks of contiguous rows among the computing nodes. Regarding the matrix Q, the client can also hardly store it and it is not acceptable to send Q to the computational nodes at each iteration. So, we propose a cyclic distribution of the columns of Q among the nodes as they are gradually computed and each worker has to keep its columns. Distributing the data storage of A and Q stresses the problem of data persistence that we handle in part 4.3.3.3.

For each MVP, the client only sends to each node the vector that is multiplied to the matrix. Each node performs the computations related to the data of its block (of A or Q) and returns a piece of the result. As the result of the MVP is always used immediately (for scalar-vector or vector-vector operations), the client must wait that all the nodes have completed their task. As a consequence, the parallelism scheme of the Lanczos tridiagonalization is highly-communicating and very synchronized. This kind of parallelism is challenging for a world-wide computing environment. It means that we have to choose a coarse-grain parallelism by increasing the block size of matrices and by decreasing the number of nodes. Nevertheless, we must not increase the grain of parallelism too much because the computational nodes have limited resources (CPU, memory, disk space, etc). We propose some solutions later.

Steps 2 and 3: Distributing the tridiagonal symmetric eigenproblem

The second and third steps of the method concern the Bisection and Inverse Iteration methods. We employ the distribution and parallelization proposed in Section 4.2. The matrix T consists of $2N-1$ elements. It fits in memory and can be sent to the remote nodes. However, if N becomes very large, it may generate significant communication times. A restarted scheme is proposed in Section 4.3.3.2 and it annihilates most of memory limitations during the Bisection and Inverse Iteration methods. In fact, the order of the working subspace is much smaller than N. Each node computes some Ritz eigenvalues and some eigenvectors. The client is able to get in memory the Ritz eigenvalues. For the associated eigenvectors, it can hardly get them in memory at the same time (N vectors). We cannot assume it will get them by means of a file because such hypothesis is middleware-dependant. Therefore, we introduce a static parameter which indicates the authorized memory usage. The client allocates this amount of memory and gradually performs the Bisection and Inverse Iteration submissions by sets of tasks depending on the remaining free memory. We recall that each set contains several tasks, and each task contains an interval of research (cf. Section 4.2.2.2). When a set of tasks has been solved, the client writes in a binary file the current eigenvectors and frees the memory for the following set of tasks. Even though this process seems complicated, it is necessary if we target to solve the full eigenproblem (i.e. in the space of order N). The restarted scheme proposed in Section 4.3.3.2 generally reduces this process to only one set of tasks.

Steps 4 and 5: distributed computations of the Ritz eigenvectors and the residuals

In the fourth step and the fifth step, we must distribute the computations of the Ritz eigenvectors and we compute the Euclidean norm of the residuals. It consists in parallelizing the MVP $u = Qv$ for each eigenvector v found by the Inverse Iteration. Those MVP are performed one after the other. The matrix Q is already distributed on the computational nodes. An optimization would consist in duplicating Q on other nodes and performing all the MVP in parallel. However, this solution requires much voluminous communication for the duplication. It is not suitable to Global Computing environment. Besides, depending on the middleware, it can cause memory problems for the client in order to get all the Ritz eigenvector at the same time. After having chosen a restarted scheme in 4.3.3.2, the duplication of Q is definitely not interesting because Q changes at each restart. Thus, it would generate much voluminous communication.

Before computing all the residuals $Au-\lambda u$ (or less if one residual fails) and their Euclidean norm, the client must wait for the completion of the previous distributed computations. It adds a synchronization point. Like for the computations of the Ritz eigenvectors, the norms of the residuals are sequentially computed. We only parallelize the computation of each residual. Contrary to the computations of the Ritz eigenvectors which are using Q, we here use A which does not change in case of restarted scheme (the duplication would only be done at the very beginning of the eigenproblem). However A is large and it would be too expensive to duplicate it. Besides, with the restarted scheme, we compute only few eigenpairs, the cost of duplication would be probably high compared to sequential computations. Finally, all the norms of the residuals are computed only for the last loop when we converge (see Figure 4.4). Otherwise, we stop computing the norms as soon as one does not respect the test of convergence. For each computation of a residual, we have to distribute the MVP Au. The client must respect a synchronization point before the remaining vector-vector and norm computations. He performs himself those calculations because distributing them would probably generates expensive communication regarding to the sequential execution time (one can use efficient BLAS routines).

Summary

In this general proposition for the distribution of the real symmetric eigenproblem, it is interesting to underline the combination of two paradigms. On the one hand, we have an usual distributed and parallel paradigm which generates communication and needs synchronization points. On the other hand, we employ the communication-less parametric-parallel paradigm. In the context of Global Computing, the data management is very important. It must be done so as to minimize the volume of communication (and the number of communications if possible). We must also weight the pros and cons before distributing a computation because the communication costs may annihilate the benefits of the parallelism (i.e. they hide the reduction of the remote computing times). This general proposition reveals some constraints of Global Computing that we detail and solve in the following sections.

4.3.3 Constraints of Global Computing and propositions

4.3.3.1 Main constraints of Global Computing

The computing nodes of a Global Computing platform can be very heterogeneous. It concerns mainly the peak CPU frequency, the total amount of main memory and the volume of disk space. Besides, those nodes may be shared with other users without any sharing policy. Thus, those three resources are not constant. A solution can be provided by the middleware of Global Computing. It requires a scheduling policy and a data distribution policy which use collected pieces of information that are regularly updated. Data distribution can be achieved by means of a dedicated peer-to-peer storage platform. At the application level, we can handle some constraints but it seems difficult to deal with the variation of the resources. Distributing a program is by nature a solution in order to share the CPU, memory and disk usage. Harnessing more computational nodes may be not sufficient. In Section 4.3.3.2, we propose solutions to limit the memory usage.

Network resources are also a critical aspect of Global Computing. They can be heterogeneous and not constant. In particular, the cost of communication over Internet has a considerable impact on the performances of the applications. At the application level, we can act upon the number of communications and the volume of transferred data as shown in Section 4.3.3.3.

Node and network resources are common constraints in most of parallel and distributed computing environments. Global Computing emphasizes them and adds communication over the Internet. This last factor generates many other constraints that are specific to Global Computing. First, we must consider the volatility of some volunteer nodes. It is due either to a network failure, a node failure or a voluntary behaviour (the machine is switched off or disconnected from the Internet). Solutions can be found at the middleware level with fault tolerance mechanisms. Some systems try to detect a failure and submit again the same task (e.g. XtremWeb). It is also possible to duplicate the same task and to submit it to many peers (e.g. Seti@home). A good solution would be to detect the failure and to recover the task but it is very challenging to implement such mechanisms for Global Computing (it is much easier for Cluster Computing or dedicated grid platforms). Another simple track, which is not fault tolerant, is to create a black-list of unstable nodes. We consider that the application level does not have to tackle this problem. Second, we can face malicious nodes. We estimate it has to be handled at the middleware level. It concerns the integrity of the computations because a node may return bad results. Also, the client can be malicious by submitting malware (malicious software) to the volunteer nodes. To avoid such problems, Global Computing software can propose authentication protocols and sandboxing techniques. Similar points inherent to activities over Internet are the confidentiality of data and the denial of service. While the first can be solved by encryption, the second one can hardly be solved.

4.3.3.2 Reduction of the memory and disk needs

Out-of-core

As we target to solve large eigenproblems, we obviously cannot load in memory the full matrices A and Q ($\in M_N(\mathbb{R})$). If p is the number of computing nodes, each one stores $\frac{N*N}{p}$ elements of A (A is not necessary sparse) and the same quantity for Q. It still represents a significant volume of data and p must be rather high if no other solution is proposed. Therefore, for A and Q, we propose an out-of-core technique by loading data gradually when they are needed for computations. We do not use a compact format because we want to do fast accesses to data in files. Since the matrices are accessed by means of MVP and, as we do not choose a compact format of storage, the accesses to the relevant data are easy and not too long. We choose a coarse grain out-of-core by loading data row-by-row (obviously, it does not concern the tridiagonal matrix T). We do not target to optimize our out-of-core solution at the cache level.

Restarted scheme

Loading in memory the N Ritz eigenvalues is rather easy but doing so with the Ritz eigenvectors is impossible for the client (there are also the eigenvectors of T which are necessary in order to compute the Ritz eigenvectors). For small values of N, the client can store them on the hard-disk but if N is large it can hardly do it. Of course, it is possible to share such data on the hard-disk of the volunteer nodes but it is not really convenient for the analysis of results. Besides, it would add much communication. This is something to avoid in the context of Global Computing.

The applications using the eigenproblem usually need few eigenpairs. They typically ask for the eigenpairs whose eigenvalues have the highest norms (or lowest). Therefore, it is not useful to save the N eigenpairs and we can limit their number to $k \ll N$. Storing k Ritz eigenvectors in memory can still be difficult but the client is able to store them on his own hard-disk. It gets data of the eigenvectors gradually by means of out-of-core.

Since we do not keep the N eigenpairs, we can choose a restarted scheme. It has been proposed for eigensolvers on High Performance Computing platforms (see Section 3.3.3). Figure 4.4 presents the flowchart of the restarted scheme. It consists of only m iterations in the Lanczos tridiagonalization instead of N, where $k < m \ll N$. Thus, we get a matrix T of order m and a matrix Q whose dimension is $m * N$. The reduction of the dimension of Q is a major advantage of the restarted scheme. It provides an answer to the problem of disk space but it does not solve it entirely for very large matrices. It also reduces significantly the cost of the full-reorthogonalization at each iteration of the Lanczos tridiagonalization. The Bisection and the Inverse Iteration methods compute m approximated candidate Ritz eigenvalues of A. Then, we select the k eigenvalues that have the highest norms and compute the k approximated candidate Ritz eigenvectors. The following step consists in checking the accuracy of the candidate Ritz eigenpairs with

the euclidean norms of the residuals. If all eigenpairs successfully pass the test, we have found the right Ritz eigenpairs. Otherwise, as soon as one candidate eigenpair fails to the convergence test, we restart the whole process and compute new candidate eigenpairs. In order to converge, we must take into account information given by the previous wrong candidate eigenpairs. It is done by computing a new initial vector which is a linear combination of the wrong candidate eigenvectors (the coefficient of an eigenvector is the associated eigenvalue).

We underline that choosing a restarted scheme does not necessary reduce the global number of operations because we can restart many times. The main interest is to use much less memory and save a significant amount of disk space.

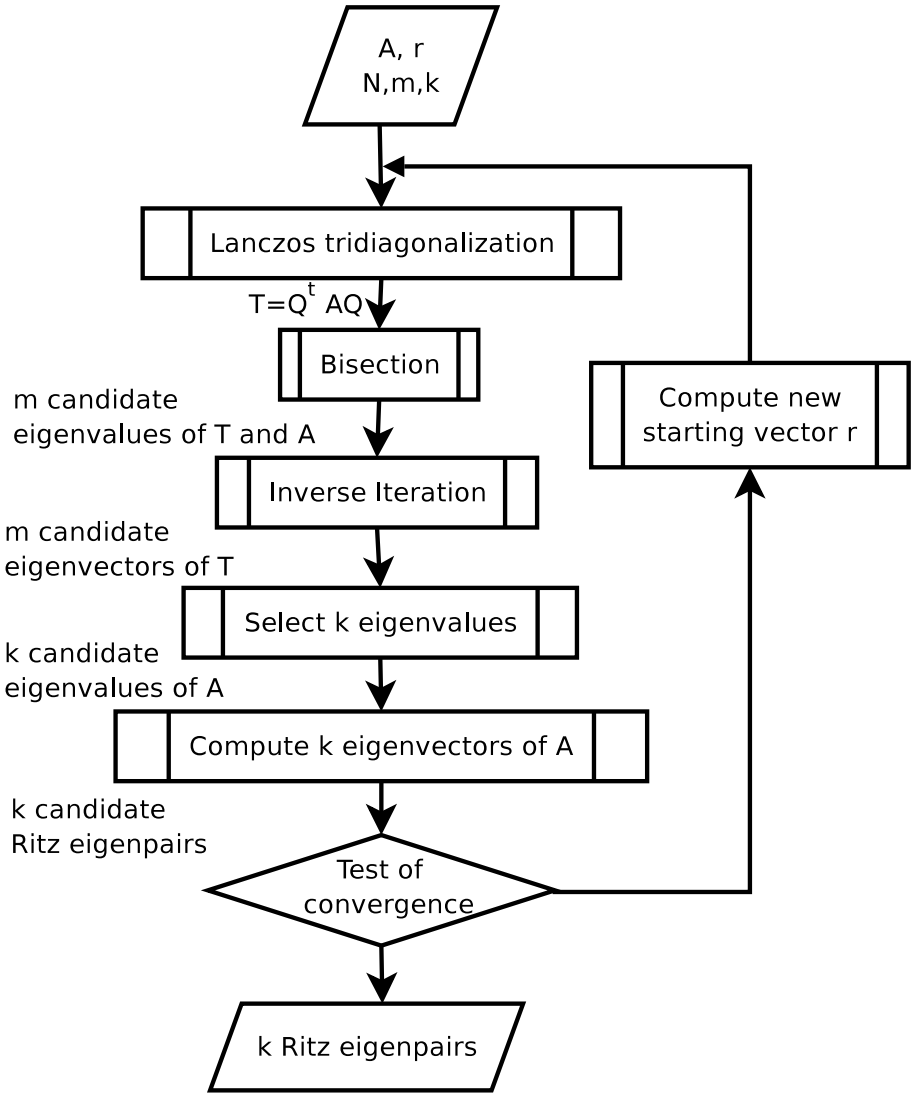

Figure 4.4: Flowchart of the restarted real symmetric eigensolver

4.3.3.3 Reduction of the communication costs

The network bandwidth is a critical resource of Global Computing and we must reduce as much as possible the volume and the number of communications.

Distribution of the matrix A

Distributing the blocks of the matrix A among the computing nodes is a difficult issue. We assume data of the matrix are originally stored by the client in a compact format into a text or binary file (for instance, we are used to handle the Matrix Market format in a text file). We assume also that this compact file provides half of the non-zero matrix components because A is symmetric (actually, a bit more than the half since some elements are on the diagonal). Each component is given with the associated column and row numbers. If A is large, it is impossible for a common desktop computer to store in memory X components and $2X$ numbers of row and column. For instance if we consider that $X = 200.10^6$, a component uses $8B$ and a number of row/column takes $4B$, it represents $3GB$ of data. So we must load and send data of A by steps. Even if the client could load in memory all data, it would saturate the network bandwidth to send it entirely at the same time.

Then, we must choose between sending the whole matrix to all the nodes or to send only the relevant pieces of data to each node. With the first possibility, the client performs several sending of data of A. Each node gradually gets data, analyses it and keeps the relevant components. It is interesting to distribute the work of the data selection. Regarding the communication point of view, the first interest of this first possibility deeply depends on the Global Computing middleware. Indeed, if it does not provide a broadcast routine, it would give very poor performances: in the case of a distribution among p nodes, it would consist in sending p times X components and $2X$ numbers of rows and columns. Even if the software provides a broadcast routine, we have to consider its implementation. There exist efficient implementations of broadcast routines based on a tree structure but they concern Cluster Computing. In the context of Peer-to-Peer Computing, the volunteer nodes are usually anonymous. Besides, in the context of large Grid Computing, the nodes seldom communicate between each other. Thus, a broadcast routine may simply hide a loop with a send routine. We do not choose this solution because it depends too much on the implementation of the broadcasting. With the second solution, the analysis of data of A is done by the client. It parses the data file, selects relevant data and sends it to the good node. Each node gets relevant data and stores it immediately. Contrary to the previous solution, the work of the data selection is not distributed. We have seen that data can be too voluminous to fit in the memory of the client. Thus, the client cannot parse the file only one time so as to load all data in memory in order to do the selection by means of memory accesses. However, the client must parse the data file only one time or very few times because it may contain a lot of components and the text file accesses are expensive. An easy solution would consist in parsing the original file only one time, and to send progressively the components to the relevant peers. However, it would generate X communications of very small volume (X components and $2X$ numbers of

rows and columns). This number of communications is too large if we consider the cost of communication and also the overhead of most of middleware at the communication layer.

Our proposition consists in breaking up the original file into p smaller files (i.e. one per node). In other words, the client parses the original file once, analyzes each piece of data and saves it in the files of the good nodes (generally two nodes or one if the component is on the diagonal). Then each file is loaded in memory and sent to the associated node. We recommend binary file for faster memory loading. The total number of communications is $p << X$ and, in total, we send X components and $2X$ numbers of rows and columns. Generally, an operating system allows opening a limited number of files, so, if the number of nodes is rather important, it can be necessary to do the same work a small number of times for different groups of nodes. The client can choose a parameter f which is the maximum number of files opened at the same time (i.e. maximum number of nodes which get data of A at the same time). The total volume of data and number of communications are unchanged; the original file is just parsed a few times more.

The distribution of A is a challenging problem. We have proposed a solution for large matrices using few memory, with few communication and the matrix file is parsed once (or very few times). However, we acknowledge it should be improved in future work.

Data-persistence

The numerical method contains several MVP involving A and Q. Those MVP are distributed among the computing nodes. Voluminous data of A and Q must not be transferred on the network for each MVP. The transferred data are only the two vectors of the MVP (result and multiplied vectors). So, each node gets its block of A at the very beginning of the execution as explained previously. It gets data of Q progressively during the Lanczos tridiagonalization. At each Lanczos iteration, one column of Q is given to a computing node (cyclic distribution). Each node keeps an handle on its blocks of Q and A so as to find them when needed. This feature is called data-persistence and must be implemented at the Global Computing software level rather than at the application level. In the context of Peer-to-Peer Computing, a node owning a block of matrix may withdraw from the platform. A solution is to perform a redundant data-persistence but it generates more communication. With the Grid Computing model, the computing nodes are generally not so volatile and there exists software providing data-persistence. We underline that data-persistence reduces the volume but not necessary the number of communications (if we use the RPC programming paradigm, it is still necessary to make a remote procedure call).

4.3.4 Experimentations on world-wide grids with OmniRPC

Software of Global Computing

Currently, software of Peer-to-Peer Computing does not allow distributing efficiently numerical methods requiring much communication and using voluminous data. Thus, we use the Grid Computing software OmniRPC that we have presented in Chapter 2 and Section 4.2.2.3. In particular, we take great advantage of its data-persistence feature.

Once OmniRPC is chosen and according to all previous propositions for our eigensolver, we can now evaluate the number and the volume of communications. Table 4.6 presents the number of communications and synchronization points. Table 4.7 shows the volume of data sent and received for each OmniRPC call. Table 4.8 explains the meaning of variables given in Tables 4.6 and 4.7.

	Number of asynchronous RPC calls	Number of synchronizations
Distributing A	p	$\lceil \frac{p}{f} \rceil$
Lanczos tridiagonalization	if $m \leq p$, $i(m + mp + \frac{m(m-1)}{2})$ if $p < m$, $i(m + 2mp - \frac{p(p+1)}{2})$	$i(2m-1)$ $i(2m-1)$
Bisection and Inverse Iteration	is	i
Computing Ritz eigenvectors	$ik * max(m,p)$	i
Tests of convergence	if convergence, ikp if test fails at the j^{th} eigenpairs, ijp	i i

Table 4.6: Number of asynchronous RPC calls and synchronizations of the restarted algorithm

	Volume of sent and received data (Bytes)
Distributing A	$16pX$
Lanczos	if $m \leq p$, $8iN(m + 2m + \frac{p^2-1}{2})$ if $p < m$, $8iN(2m + 3mp - \frac{(p+1)^2}{2})$
Bisection and Inverse Iteration	$8m^2 + 24m$ (with $S = 1$)
Computing Ritz eigenvectors	$16ik * max(m,p) * N$
Tests of convergence	if convergence, $16ikpN$ if test fails at the j^{th} eigenpairs, $16ijpN$

Table 4.7: Approximated volume of communications of the restarted algorithm

The memory usage depends a lot on platform-dependent propositions and it also deeply

N	Order of A
m	Size of the Krylov subspace
k	Number of computed Ritz eigenpairs
p	Number of computing nodes
i	Number of iterations of the restarted scheme
X	Number of non-zero components of A
f	Maximum number of files opened at the same time for the distribution of A
s	Number of sets of subintervals

Table 4.8: Parameters reminder of the real symmetric eigensolver

depends on programming choices done with OmniRPC. Contrary to platform-independent solutions, those programming choices can be easily modified. Generally, the modifications improving the scalability of the application leads to poorer performances (e.g. a smaller f for the distribution of A saves memory but generates more parsing of the data file of A). Our implementation monopolizes $8 * m * N$ Bytes on each computing node, and $max(8 * p * N, 16 * e)$ Bytes on the client. During the distribution of A, e is the maximum number of components of A sent to a group of nodes: it depends on the choice of f.

Computing and network resources

The characteristics of the computing and network resources are the same as in Section 4.2.2.3.

Experimental platforms

Table 4.9 presents the four experimental platforms built with OmniRPC and the computing resources at the USTL and at Tsukuba. We use the "cluster configuration" of OmniRPC. Each relay server uses a simple round robin scheduler with SSH invocations. Communication is multiplexed.

	Config. 1	Config. 2	Config. 3	Config. 4
Location of client	USTL			
Nb and location of OmniRPC servers	1 at Lille	2 at Lille	1 at Lille 1 at Tsukuba	2 at Lille 2 at Tsukuba
Nb workers per server	29 nodes/server			
Total nb workers	29	58	58	116
Location of workers	USTL		USTL and University of Tsukuba	

Table 4.9: Experimental world-wide platforms for the real symmetric eigensolver

Parameters and hypothesis

The real symmetric eigensolver has many parameters, such as m and k, and we use many configurations of platforms involving several networks and different kinds of computing resources. Each execution depends on many factors and we cannot monitor some of those factors (e.g. other communication over the Internet, concurrent processes on machines). Therefore, the analysis of each execution is a difficult task. Although we target solving very large eigenproblems on large Global Computing platforms, we first propose using a smaller matrix in order to study the impact of the numerical parameters and the configurations of platforms on the execution wall-clock times. Once this work of analysis will be done, we will test much larger matrices and platforms (see Section 4.3.5).

So, we first use a real symmetric matrix of order $N = 47792$. It is built by means of a tiling whose pattern is the sparse real symmetric matrix Bcsstk18 from the Harwell-Boeing matrix collection of the Matrix-Market website[1]. This last matrix has the following characteristics: $N = 11948$, $nnz = 149090$ (non-zero elements), the average nnz per row or column is 12 and the condition number is 65. $N = 47792$ is a balanced choice in order to study programming paradigms and the impact of the parameters on the convergence of the restarted scheme. It is not necessary to use much larger orders for this kind of study. With this size of matrix, the computing times are not too long. So, we can perform a lot of experiments with different platforms and several parameters. In this study, we do not focus on the impact of the matrix pattern on the convergence. Thus, we only test intensively this matrix of order $N = 47792$. The starting vector is $r = (\frac{1}{\sqrt{N}}, \ldots, \frac{1}{\sqrt{N}})$, $r \in \mathbb{R}^N$. This choice is motivated by the Euclidean norm of r which is 1.

We consider two input parameters. The size of the Krylov subspace m which is either 10, 15, 20 or 25, and the number of required Ritz eigenpairs k which varies from 1 to 4. With the four platforms, it represents 64 different combinations.

For the Bisection and Inverse Iteration steps, only one computing nodes is involved because m is rather small. So, the maximum threshold of eigenvalues per subinterval (see our proposition in Section 4.2.2.2) is upper than m in order to start the computation with the Gerschgorin domain which contains all the eigenvalues.

Results

All times are in seconds. Most of them are the average of five runs. They are measured on the client's side. They are wall-clock times which include communication and remote computing times. Table 4.10 and the associated Figure 4.5 show the wall-clock times needed to compute the k Ritz eigenpairs. For each couple (m, k), the number of restarts is given in Table 4.11.

Table 4.12 shows the average percentages of time spent in the Lanczos tridiagonalization,

[1] http://math.nist.gov/MatrixMarket/

the Ritz eigenvectors computations and the tests of convergence (compared to the total wall-clock time). The wall-clock times of the Bisection and Inverse Iteration are so small compared to the others that we do not show the related percentages. Since the four configurations have similar percentages, we only show data of the second one. Table 4.13 gives the percentages of time spent in the MVP involving A (in the Lanczos tridiagonalization) and the reorthogonalization compared to the whole Lanczos tridiagonalization.

m, k	Config. 1	Config. 2	Config. 3	Config. 4
10, 1	1358	1289	2895	6824
10, 2	2624	2515	5566	13045
10, 3	5051	4847	11067	24957
10, 4	10356	9853	23533	52993
15, 1	1248	1264	2795	6269
15, 2	2140	2155	4798	11137
15, 3	3068	2850	5866	13196
15, 4	8322	7765	17987	40558
20, 1	1190	1232	2556	5549
20, 2	1822	2112	4072	8529
20, 3	3065	2645	5557	11789
20, 4	6316	5875	13894	38339
25, 1	1765	1711	3369	7068
25, 2	2008	1582	3413	7673
25, 3	3341	2617	5343	11221
25, 4	4831	4563	11915	23524
Config. reminder	OmniRPC: config. 1-2-3-4 Lille 29 workers: config. 1 Lille 58 workers: config. 2 Lille & Tsukuba 58 workers : config. 3 Lille & Tsukuba 116 workers : config. 4			

Table 4.10: Wall-clock times to compute the k Ritz eigenpairs (in seconds) on the world-wide platforms - N=47792

4.3.5 Experimentations on Grid5000 with OmniRPC

Grid5000[49] is a large scale computing tool composed of many clusters distributed in several computing centers in France. Those clusters are interconnected by a fast dedicated network. The Grid5000 usage is based on a reservation policy and a deployment mechanism allowing people configuring their own environment. Details can be found on the Grid5000 website [2].

[2]http://www.grid5000.fr

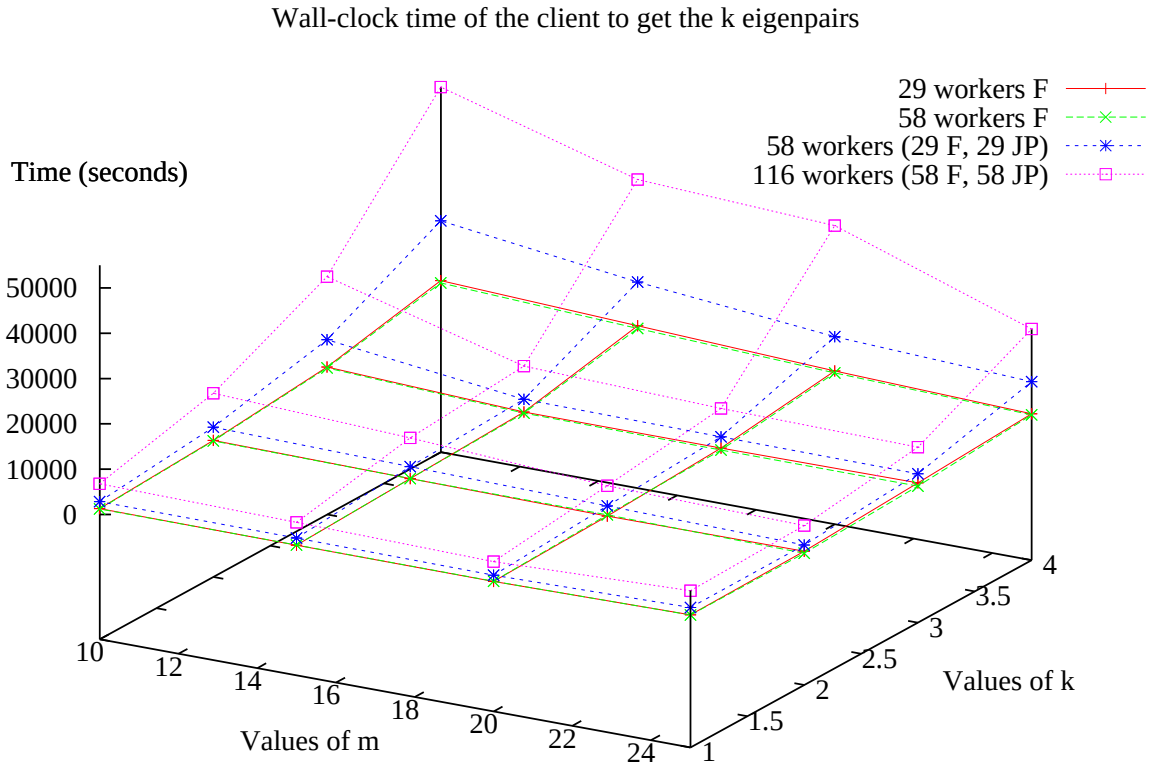

Figure 4.5: Wall-clock times to compute k Ritz eigenpairs (in seconds) on the world-wide platforms - N=47792

m, k	number of restarts	m, k	number of restarts
25, 1	2	15, 1	3
25, 2	2	15, 2	5
25, 3	3	15, 3	6
25, 4	6	15, 4	18
20, 1	2	10, 1	5
20, 2	3	10, 2	9
20, 3	4	10, 3	17
20, 4	10	10, 4	36

Table 4.11: Number of restarts to compute the k Ritz eigenpairs on the world-wide platforms

	Config. 2 Whole eigenproblem = 100%		
m, k	Lanczos tridiagonalization	Ritz eigenvectors	Tests of convergence
10, 1	90%	1%	7%
10, 2	87%	2%	9%
10, 3	86%	3%	10%
10, 4	86%	4%	8%
15, 1	93%	1%	5%
15, 2	90%	2%	7%
15, 3	87%	2%	9%
15, 4	87%	4%	8%
20, 1	95%	1%	3%
20, 2	92%	1%	5%
20, 3	89%	2%	7%
20, 4	88%	4%	8%
25, 1	95%	1%	2%
25, 2	93%	2%	4%
25, 3	91%	3%	5%
25, 4	88%	4%	7%

Table 4.12: Time spent (in%) by the Lanczos tridiagonalization, the computations of the Ritz eigenvectors and the tests of convergence compared with the whole eigenproblem

	Config. 2 Lanczos tridiagonalization = 100%	
m, k	Matrix-Vector Products	Reorthogonolizations
10, 1	88%	10%
10, 2	88%	10%
10, 3	90%	8%
10, 4	88%	9%
15, 1	86%	12%
15, 2	86%	12%
15, 3	86%	12%
15, 4	85%	13%
20, 1	80%	18%
20, 2	86%	12%
20, 3	83%	15%
20, 4	79%	18%
25, 1	78%	20%
25, 2	77%	20%
25, 3	76%	22%
25, 4	77%	22%

Table 4.13: Time spent (in%) by the matrix-vector products involving A and the re-orthogonalizations compared with the whole Lanczos tridiagonalization

Motivations for the usage of Grid5000

First, the results of the previous experiments on world-wide platforms do not depend only on the algorithmic solutions, the effectiveness of our implementation but they also depend on outside events. Those perturbations are due to communication over the Internet and concurrent processes of other users on computing devices at the USTL. It would be very interesting to isolate from outside perturbations the same experiments. It would allow us improving our analysis of previous tests done on world-wide platforms. Of course, it is not relevant to compare the computing times since Grid5000 is a high performance and dedicated computing tool whereas the previous tests are done with realistic Grid platforms.

Second, Grid5000 is a good tool in order to test the scalability of our implementation with much bigger matrices and more computing nodes.

Software of Global Computing

Thanks to the deployment mechanism of Grid5000, we can configure exactly the same software environment as for previous tests. However, we must test and use a new version of OmniRPC for the 64Bits architectures.

Computing and network resources

We use the resources of four computing centers: Orsay, Lille, Sophia and Rennes. The harnessed nodes are single- or dual-processor computers (AMD Opteron family, from 2.0 to 2.6GHz of peak frequency). There is one core per CPU. The main memory is either 2GB or 4GB. The intra-site network is Gigabit Ethernet and the sites are inter-connected by the RENATER network (1Gbit/s in practice).

Parameters and experimental platforms

We first use the same matrix of order $N = 47792$ as in Section 4.3.4. We consider the same input parameters m and k. As shown in Table 4.14, the configuration 1, 2, 3, 4 are similar to the previous ones built at Lille and Tsukuba. In particular, for the configuration 4, we use 29 dual-processor computers at Sophia in order to run 58 OmniRPC workers. A fifth configuration uses only one OmniRPC worker per dual-processor system and the 29 remaining workers are launched on computing nodes at Lille.

In order to test the scalability of the application, we use two other matrices of order $N = 203116$ and $N = 430128$ which are also a tiling of the matrix Bcsstk18 from the Harwell-Boeing matrix collection. The first one has 43 million elements and the second one 193 million elements. 10% of the components of the matrices are non-zeros. We underline that the configurations of Tables 4.15 and 4.16 use only one OmniRPC worker per node even in the case of multiprocessors. The main reason is explained in Section 4.3.6 which presents the analysis of the results. A second reason is the small disk space

	Config. 1	Config. 2	Config. 3	Config. 4	Config. 5
Location of client	Orsay				
Nb and location of OmniRPC servers	1 at Orsay	1 at Orsay	1 at Orsay 1 at Lille	2 at Orsay 1 at Sophia	2 at Orsay 1 at Sophia 1 at Lille
Nb workers per server	29				
Total nb workers	29	58	58	116	116
Location of workers	Orsay		Orsay Lille	Orsay Sophia	Orsay Sophia Lille

Table 4.14: Experimental platforms on Grid5000 for N=47792

$(4.2GB)$ at Sophia: it does not allow storing two blocks of data of A.

	Config. for $N = 203116$
Location of client	Orsay
Nb and location of OmniRPC servers	4 at Orsay 2 at Sophia 1 at Lille
Total nb workers	206

Table 4.15: Experimental platform on Grid5000 for N=203116

	Config. for $N = 430128$	
Location of client	Orsay	
Nb and location of OmniRPC servers	5 at Orsay 1 at Sophia 1 at Lille	8 at Orsay 1 at Sophia 1 at Lille 1 at Rennes
Total nb workers	206	412

Table 4.16: Experimental platform on Grid5000 for N=430128

Results

All times are in seconds and are measured on the client's side (wall-clock times). Table 4.17 and Figure 4.6 show the wall-clock times needed to compute the Ritz eigenpairs for the matrix of order 47792.

In Tables 4.19, 4.20, 4.21 and 4.22, we give the wall-clock times picked up for the two large matrices. For the matrix of order $n = 203116$, we fix the number of nodes and modify the values of the couple (m, k). On the contrary, for the matrix of order $N = 430128$, we fix a value for (m, k) and change the number of nodes. We present an evaluation of the computing time of one MVP in Tables 4.23 and 4.24.

m, k	Config. 1	Config. 2	Config. 3	Config. 4	Config. 5
10, 1	190	93	147	431	162
10, 2	320	167	241	706	266
10, 3	616	309	465	1376	512
10, 4	1477	761	1136	3501	1258
15, 1	173	90	135	387	149
15, 2	298	148	233	661	260
15, 3	307	163	239	675	266
15, 4	1072	542	826	2500	922
20, 1	157	81	122	344	142
20, 2	247	124	186	524	217
20, 3	253	125	188	538	217
20, 4	778	392	594	1770	672
25, 1	196	102	157	437	181
25, 2	317	160	241	669	282
25, 3	320	161	244	677	283
25, 4	664	329	506	1465	580

Table 4.17: Wall-clock times to compute the k Ritz eigenpairs (in seconds) on Grid5000 - N=47792

m, k	number of restarts	m, k	number of restarts
25, 1	2	15, 1	3
25, 2	3	15, 2	5
25, 3	3	15, 3	5
25, 4	6	15, 4	17
20, 1	2	10, 1	5
20, 2	3	10, 2	8
20, 3	3	10, 3	15
20, 4	9	10, 4	36

Table 4.18: Number of restarts to compute the k Ritz eigenpairs on Grid5000 - N=47792

(m,k)	Wall-clock times with 206 computing nodes on 3 sites
10, 1	2809
10, 2	2768
10, 3	4878
10, 4	6103

Table 4.19: Wall-clock times to compute the k Ritz eigenpairs (in seconds) on Grid5000 - N=203116

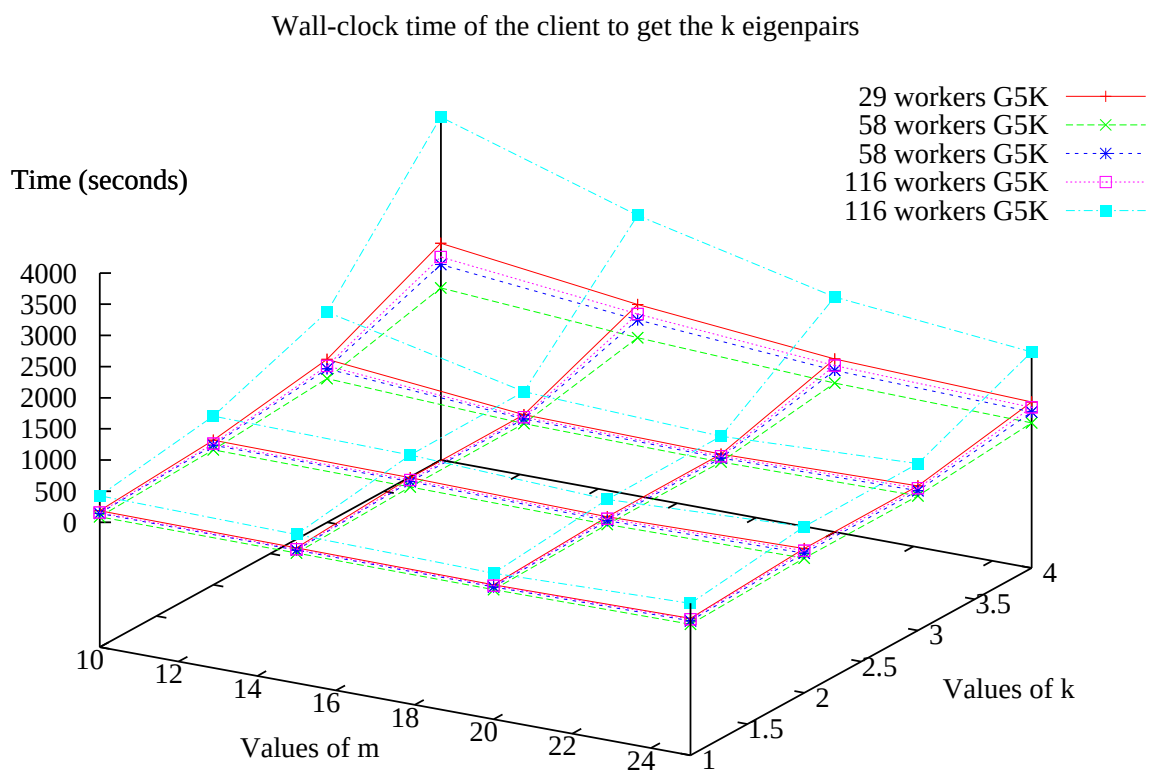

Figure 4.6: Average wall-clock times to compute k Ritz eigenpairs (in seconds) on Grid5000 - N=47792

Nb nodes	Wall-clock times with (m,k)=(15,1)
206 (3 sites)	10962
412 (4 sites)	13150

Table 4.20: Wall-clock times to compute the k Ritz eigenpairs (in seconds) on Grid5000 - N=430128

Detailed wall-clock times with 206 nodes on 3 sites	
Whole eigenproglem	10962
Lanczos - distribute new column of Q	22
Lanczos - MVP with A	10106
Lanczos - Reorthogonalization	129
Bisection and Inverse Iteration	1
Compute the Ritz eigenvectors	9
Check the norm of the residuals	691

Table 4.21: Detailed wall-clock times of the experiments on Grid5000 with N=430128 and 206 computing nodes

Detailed wall-clock times with 412 nodes	
Whole eigenproglem	13150
Lanczos - distribute new column of Q	20
Lanczos - MVP with A	12311
Lanczos - Reorthogonalization	159
Bisection and Inverse Iteration	9
Compute the Ritz eigenvectors	11
Check the norm of the residuals	810

Table 4.22: Detailed wall-clock times of the experiments on Grid5000 with N=430128 and 412 computing nodes

	Number of MVP with A	Wall-clock time/MVP
Lanczos - MVP with A	75	134
Check the norm of the residuals	5	138

Table 4.23: Wall-clock time of one MVP involving the matrix A - N=430128, 206 workers

4.3.6 Analysis and perspectives

4.3.6.1 Analysis of tests on the world-wide platforms - N=47792

We first analyze the wall-clock times in Table 4.10 and the corresponding Figure 4.5.

First, let us fix the parameter k. Generally, for all platform configurations, the larger m, the more the wall-clock time is decreased. In fact, the larger the difference between m and k, the more information to compute the eigenpairs we get and the less restarts we do. We notice six exceptions to this statement. Four exceptions concern the value $k = 1$ when m becomes large (from 20 to 25). In those cases, it is not useful to increase m more than 20 because it adds useless computations. Table 4.11 confirms this point because there is the same number of restarts for $m = 20$ and $m = 25$ when $k = 1$. The two last exceptions concern the configuration 1 for $k = 2$ and $k = 3$ when m is going from 20 to 25. In those cases, decreasing m from 25 to 20 only adds one restart. As there is no communication over the Internet, doing three restarts and working on a smaller subspace is less expensive than performing two restarts but running the Lanczos tridiagonalization on a larger subspace.

Then, let us fix m. For all configurations, when we increase k, the wall-clock time is

	Number of MVP with A	Wall-clock time/MVP
Lanczos - MVP with A	75	164
Check the norm of the residuals	5	162

Table 4.24: Wall-clock time of one MVP involving the matrix A - N=430128, 412 workers

also increased because the problem requires more information but the subspace does not provide more information. So, the number of restarts is increased. There is one exception for $m = 25$ when k goes from 1 to 2. In this case, the number of restarts is the same and there is no communication over the Internet. As the computational nodes at the USTL may change and do not have the same characteristics or may be shared with people, we suppose such events have penalized the tests with $k = 1$ (compared to tests where $k = 2$).

Next, we compare the four configurations of Table 4.10. On the same local network at Lille, we do not notice significant differences between configurations 1 and 2. It means that 29 workers would deal quite well with the CPU and memory requirements of those eigenproblems. Thus, people could judge unnecessary to involve 58 workers. However, from a data storage point of view, each one of the 29 nodes needs $600\,MB$. In the context of Global Computing, we keep in mind that a node may have limited disk space and can be shared with several users. So we would better use the configuration 2 using $300\,MB$ of disk space per worker. Besides, it would be interesting to save CPU cycles too. In fact, when m reaches the upper values ($m = 20, 25$), the configuration 1 becomes a bit slower. Each node handles a twice bigger workload compared to the other configuration. This additional workload is not balanced by the smaller number of communications. In general, the owners of computing resources may appreciate a moderate use of their machines. If this usage is moderate, we think that a larger number of volunteers may be willing to adhere to the Global Computing platform and offer their resources. Configurations 2 and 3 have the same number of workers but the second one requires much communication over the Internet. Although the impact of communication is important, it cannot explain entirely such an increase of the wall-clock time. There is an average increase of 119 %! By analyzing the tests on Grid5000 with the same matrix and similar configurations, we will point out a second cause of low performance. Indeed, we get large wall-clock times when each processor of a dual-processor system handles an OmniRPC worker. In previous Section 4.2.2.4 related to the analysis of the tridiagonal eigenproblem with XtremWeb and OmniRPC (using similar computing resources), we have already pointed out this problem during the Inverse Iteration step. However the impact was less critical, probably because less main memory was required. In addition to the memory usage by two OmniRPC workers on each dual-processor system, may be there is a bad configuration of the cluster mode of OmniRPC (location of the 2 OmniRPC servers). This point is currently studied. Then, we focus on configurations 3 and 4. The last one doubles the numbers of workers. On the one hand, it means a decrease of the workload on each node by a factor 2 but, on the other hand, it doubles the number of RPC calls. The cost of communication over the Internet may not explain entirely the average increase of 123 % of the wall-clock time. In this case too, the usage of all processors of the computers at Tsukuba is a fundamental cause of bad performance as the tests on Grid5000 will show in Section 4.3.6.2. We also think that the configuration 4 reaches a too fine-grain parallelism for this size of matrix.

Now, we study the Tables 4.12 and 4.13. Table 4.12 shows the average percentages of time spent in the Lanczos tridiagonalization, the Ritz eigenvectors computations and the tests of convergence. The wall-clock times of the Bisection and Inverse Iteration methods are so small compared to the others that we do not show the related percentages. Since

the four configurations have similar percentages, we only give data of the configuration 2. Table 4.13 gives the percentages of time spent in the MVP involving A and the reorthogonalization compared to the whole Lanczos tridiagonalization.

First of all, in Table 4.12, the Lanczos tridiagonalization is clearly the most time consuming step of the eigenproblem. It represents between 86% and 95% of the total wall-clock time needed to find the Ritz eigenpairs.

Second, let us fix k. When m is increasing, the percentage of the Lanczos tridiagonalization in Table 4.12 is increasing too. Indeed, the size of the Krylov subspace is larger. In other words, we compute more vectors of the Krylov basis and the number of Lanczos iterations grows (m iterations). Besides, inside a Lanczos iteration, the larger the Krylov basis is, the more a reorthogonalization has floating-point operations as shown in Table 4.13. In fact, when the size if the Krylov basis is increasing, there are more vectors to reorthogonalize. The MVP involving A does not depend on the parameter m, so it does not contribute to the increase of the percentage of the Lanczos tridiagonalization. We stress that the decrease of the percentage of this MVP (Table 4.13) is not due to less computations but it is explained by the increase of the percentage of the reorthogonalization. The rise of the percentage of the tridiagonalization (Table 4.12) is also augmented by the decrease of the time spent in the tests of convergence. In fact, the number of restarts decreases while m increases and k does not change. We do not notice significant variations of the percentages in the computations of the Ritz eigenvectors because this step mainly depends on k (k distributed matrix-vector products Qv with $Q \in M_{m,N}(\mathbb{R})$, $m \ll N$).

Finally, let us fix m. As we increase k, we notice in Table 4.12 a decrease of the percentage of the Lanczos tridiagonalization. It is not due to less computations. Indeed, as illustrated in Table 4.13, this step does not depend on k. On the contrary, when k is increasing and m is fixed, the number of restarts is increasing, so we would have expected higher percentages of Lanczos. We explain this behaviour by the increase of the two other percentages of Table 4.12. In fact, a larger k generates more Ritz eigenvectors to compute, more Ritz eigenpairs to check, and more restarts if m does not change.

Summary

Compared to the study of the parametric parallelism in Section 4.2, the experimentations of this section have underlined the impact of communication over the Internet, coupled with many synchronization points. In a context of Global Computing, we have to choose parallel and distributed paradigms which minimize communication and synchronizations. The Bisection and Inverse Iteration methods are particularly well suited. On the contrary, the Lanczos tridiagonalization and the restarted scheme are not optimal in term of communication and synchronizations. However, they are very interesting in term of scalability. As computing time is not the main criterion of performance of Global Computing, they remain relevant. The scalability is much more important. Next, the tests have shown how the wall-clock times greatly depend on numerical parameters, especially

on the order m of the subspace. Depending on the number of computed Ritz eigenpairs k, the choice of m acts upon the number of restarts (and then on the number of communications) but it also affects the CPU and memory usage on each worker. Then, although the computers at the University of Tsukuba are powerful and dedicated to research, our client at Lille does not get any benefit from harnessing them. An easy explanation blames communication over the Internet but we will show in Section 4.3.6.2 that middleware and platform configurations are also a significant performance burden.

4.3.6.2 Analysis of tests on Grid5000 - N=47792

We analyze Table 4.17 and the associated Figure 4.6. First, we fix the parameter k. Like for the results on Global Computing platforms, the larger m, the more the wall-clock time is decreased. The explanation is the same. There are also some exceptions. For $k = 1, 2$ and 3, increasing m more than 20 adds useless computations. The number of iterations is the same for $m = 20$ and $m = 25$ (see Table 4.18). On the contrary, if $k = 4$, it is interesting to increase m up to 25 in order to get more information: it reduces the number of iterations and thus the wall-clock-times. We underline that using a 64 bits architecture and the new version of OmniRPC has slightly modified the convergence behaviour of our program. The numbers of restarts are modified.

Then, we fix m. For all configurations, when we increase k, the wall-clock time is increased too. So, the behaviour is the same as on a Global Computing platform. However there is no exception. When the number of restarts does not change (from $k = 2$ to $k = 3$ for $m = 15, 20, 25$), the increase of the wall-clock times is tiny but logical due to the larger Krylov subspace. It does not decrease contrary to the only one exception in the tests on a Global Computing platform. This regularity is due to the homogeneity of the computing resources and to the constancy of the network bandwidth (tests mainly done during the night in France).

Next, we compare the five configurations. On the same LAN (configurations 1 and 2), we take great advantage to increase the number of computing nodes from 29 to 58 for all couples (m, k). With the previous tests done on the network of workstations at Lille (Section 4.3.6.1), dividing the workload by increasing the number of nodes was interesting only for $k = 4$ because the workload becomes very significant. For smaller values of k, we have first concluded that 29 nodes deal quite well with this eigenproblem and that dividing the workload does not provide any benefit. Actually, the main cause is the poor network bandwidth of the network of workstations at Lille. Thus, the communication times hide the real reduction of remote computing times. With Grid5000, the network is rather efficient and we can observe the benefits of the larger distribution of work among 58 nodes. Then we observe the configurations 2 and 3. They have the same number of workers (58) but the second configuration uses two sites. With the previous tests on a world-wide platform, using two sites gives very poor performances since the wall-clock time is increased by an average of 119%. We have cited communication on the Internet

as a performance burden. In the case of Grid5000, the wall-clock time is also increase by an average of 50%. We do not use the Internet and the Renater network which is linking the sites of Grid5000 is quite fast. Besides, there is a reservation policy and we perform the experiments during the night. We also do several executions in order to compute average times. Thus, the communication costs cannot be the main explanation of this high percentage (rather good performances, no network congestion, etc). We suppose that using one OmniRPC worker per processor of the multiprocessors at Sophia explains the poor performances. We first think that the processes compete for main memory. However, there is a large amount of memory on the nodes and separated caches. Perhaps the processes compete to access to the network resources. We are currently studying a problem of configuration of the cluster mode of OmniRPC (location of OmniRPC servers on multiprocessors) The same problem is observed if we compare the configurations 3 and 4. Indeed, when we divide the workload by using 128 workers instead of 58, the wall-clock times are increased by an average of 187%. With the tests on a world-wide platform, this increase is 123% and, this problem of the OmniRPC configuration on multiprocessors is a second explanation in addition to slow communication over the Internet.

In order to confirm this hypothesis regarding the usage of multiprocessors, we compare the configurations 4 and 5. The two configurations use 116 workers but the configuration 5 harnesses some nodes in a third site in order to use only one OmniRPC worker per node (in particular on multiprocessors at Sophia). We observe that the average wall-clock time is 60% lower for the configuration 5 (although there is communication on a third site).

By comparing the configurations 3 and 5, we confirm also the previous analysis done for the world-wide platforms. It is not interesting to harness 116 workers because the wall-clock-time is increased by an average of 12%. This increase is no longer due to the competition of OmniRPC workers on multiprocessors but only to the overhead of the communication layer. For this size of matrix and those couples (m, k), choosing a number of workers upper than 58 implies a too fine grain of parallelism. There is too much communication compared to the workload of the workers.

Summary

By means of those tests on Grid5000, we have shown that the main performance burden of the experiments on world-wide platforms is the usage of all the CPU of multiprocessors by the OmniRPC workers. The comparison between configurations 1 and 2 mainly emphasizes the impact of network performances, even inside a LAN. Indeed, in the similar tests on world-wide platforms, slow communication hides the benefits of the distribution of work (i.e. the decrease of the remote computing times of the workers). Finally, we have confirmed that distributing the eigenproblem among 116 workers leads to a too fine grain of parallelism.

4.3.6.3 Analysis of tests with larger matrices

We first look at Tables 4.19 and 4.20. With the matrix of order $N = 203116$, the best value of m is 15. Choosing $m = 20$ and $m = 25$ adds too much computations compared to a smaller Krylov subspace like $m = 15$ which provides enough information in order to compute only one Ritz eigenpair. On the contrary, the value $m = 10$ is too small. Thus, the resolution needs to restart one more time and the wall-clock time is increased.

Then, we consider the largest matrix of order $N = 430128$. With the largest configuration (416 workers), we reach a too fine grain of parallelism. The increase of the wall-clock time is mainly due to the cost of communication and also probably to the overhead of the middleware. Indeed, we use only one CPU of the multiprocessors, the number of restarts obviously does not change, and we have done the experimentations during the night and the early morning while no or few other users were working in the sites of Rennes, Lille, Orsay and Sophia. Next, as the computing times with this last matrix are particularly long, we focus on the detailed computing times.

The MVP involving A of the Lanczos tridiagonalization is clearly the most time consuming step since it represents 92 (resp. 93%) of the wall-clock time with 206 workers (resp. 412). So, it is interesting to evaluate the cost of a single MVP involving A. Such a MVP occurs in the Lanczos tridiagonalization but also in the tests of convergence in order to compute the residuals. As there are 5 main loops, 15 iterations per Lanczos tridiagonalization and 1 MVP with A per Lanczos iteration, we have 75 MVP during the 5 tridiagonalizations. Besides, as there are 5 main loops and 1 computed eigenpair, the computations of the residuals generate 5 MVP with A. Then, with the times of Tables 4.21 and 4.22, we can estimate the average time of one MVP in the tridiagonalizations and in the residual computations. Tables 4.23 and 4.24 give those times which include the communication times. During the tridiagonalizations (resp. residual computations), the average time of one MVP is 30 seconds longer (resp. 26) if it is shared among 412 workers instead of 206. It confirms the too fine grain of parallelism of the largest configuration in the context of Grid5000 and by using the RPC programming paradigm. On a supercomputer, this threshold of grain of parallelism would probably differ. Indeed, a supercomputer would not use communication through the same kind of LAN (and a WAN), it would use a different programming paradigm (not RPC calls), and so on.

Summary

The experimentations with larger matrices and a higher number of nodes show the scalability of our implementation. However, it requires a significant amount of time in order to solve the eigenproblems although we ask for only one eigenpair. On a world-wide platform, or even on an efficient platform built on Grid5000, the cost of communication is probably too high to reach an acceptable level of performance in term of computation speed. We can also consider the middleware overhead. A solution would consist in using a larger grain of parallelism but the volunteer workers would not be able to handle such

very large tasks.

4.3.6.4 Synthesis of the analysis and perspectives

The resolution of the real symmetric eigenproblem requires communication and synchronization points. Thus, when we use a Global Computing platform to solve this problem, we get large wall-clock times. This statement can probably be generalized to most of distributed linear algebra problems that require much communication and many synchronization points. At first sight, if the main criterion of performance chosen by the client is the wall-clock time, it seems that High Performance Computing systems are necessary for this kind of problems. However, we must not eliminate the Global Computing model because of many reasons, such as the restricted access to HPC devices, given in Section 1.2 related to our motivations. If the wall-clock time is not a criterion of performance, then this study shows the viability of the Global Computing model for the resolution of complex problems. We must not limit our analysis to this simplest conclusion. By observing in more details the results we can make some useful comments for the parallelization and the distribution of many linear algebra problems.

We have underlined the importance of the choice of the parallel paradigm. The Bisection and Inverse Iteration methods based on the parametric-parallel paradigm do not suffer from the poor network performances. This task-farming step is wrapped into a classical parallel model based on communication and synchronization points which penalizes the global performance. For all numerical problems, the choice of the suitable parallel paradigm has to be done at the very beginning of the study while searching for the numerical algorithms to employ. The experimentations have emphasized the impact of communication over the Internet and even within a unique LAN. This impact is striking for the Lanczos tridiagonalization and its MVP involving the full matrix A compared to the other steps of the algorithm, particularly compared to the Bisection and Inverse Iteration. This contrast is increased by the restarted scheme. It synchronizes more the execution of the eigensolver. However, it reduces significantly the dimension of the working space. Thus, it reduces the amount of required main memory in order to solve the eigenproblem. It saves disk space too. It contributes to the scalability of our implementation as shown by the experiments on Grid5000. This scalability is also possible by means of other strategies like data-persistence and out-of-core. This choice of using or not those strategies must be settled just after the choices of the algorithms and the parallel paradigms. Indeed, once the algorithms and the parallel paradigms are fixed, we can estimate the memory and disk requirements and we can have a rather good idea of the number of communications. So, depending on the targeted model of parallel computing (High Performance Computing, Global Computing, etc.), we can decide the interest to use such strategies without having already any consideration for middleware and programming details. Of course, their implementation depends on the middleware features and its API.

We have shown the major role of the configuration of the experimental platforms. We have pointed out the configuration of the computing software and also the size of the platform. In the first case, the computing software cannot handle well all CPU of multiprocessors. In the second case, a too large platform gives a too fine grain of parallelism because the computing time on each remote node is too small compared to the communication and invocation times. Contrary to the previous choices (task-farming, data-persistence, etc.), we can hardly forecast the best configuration and we find it empirically.

At the end of our study, some issues remain opened. In particular, the distribution of A must be improved. Our strategy is scalable but, in the worst experimental case (N=430128), it has taken 40 minutes. The new data layer management of OmniRPC, called OmniStorage, may provide an efficient solution. With this new software, each relay node acts as a proxy. The gain would be particularly visible for experiments harnessing some clusters of resources on the Internet. The second opened issue concerns the disk storage space. By increasing more the order of the matrix, the disk requirements may bound the scalability of our eigensolver. Each block of matrix A needs $\frac{8N^2}{p}$ Bytes. We have not chosen a compact format of storage in order to do fast accesses to data. Indeed, due to the out-of-core strategy, the number of file accesses is very important. A simple solution is to increase the value of p i.e. the number of workers but we have seen that a too fine grain of parallelism gives poor performances.

Chapter 5

Toward Power-Aware Computing with Dynamic Voltage Scaling for Heterogeneous Grid

5.1 Scope of the study

In Cluster Computing and High Performance Computing, we can take advantage of the slack-time caused by an unbalanced distribution of work in order to save energy. The slack-time is the idle time of some computing resources while the others are still working. In the context of heterogeneous world-wide Grid Computing, we consider two other sources of slack-time. The first one is due slow communication. The second source of slack-time concerns the heterogeneity of the computing nodes. Contrary to High Performance Computing and Cluster Computing which are using homogeneous resources in order to reach the highest performance, world-wide Grid Computing harnesses a wide range of heterogeneous volunteer devices. To our knowledge, power-aware considerations have not yet been explored for heterogeneous large Grid Computing.

This Chapter is structured as follows. In Section 5.2, we present related work on low-power technologies and power-aware computing for parallel and distributed computing. For the clarity of the dissertation, we deliberately talk about related work in this Chapter instead of with the State-of-the-Art Chapters 2 and 3. Then, Section 5.3 deals with the characteristics of the experimental power-aware cluster. Next, we describe in Section 5.4 the DVS-capable real symmetric eigensolver which is used for the experiments. Section 5.5 gives the results of the experiments and presents our analysis. Finally, in Section 5.6, we conclude the Chapter and present ideas for future work.

5.2 Related work

A great amount of work on power and energy conservation has already been performed. Many approaches enable to save energy. At the pure hardware level, more precisely at the chip level, we can cite asynchronous clocking techniques such as IPCMOS [98]. Besides, always in [98], we see that tuning the L1 data cache size may affect the power consumption. At the component level, the architecture of Transmeta processors shows that we can get significant energy savings on the CPU. Actually, in this case, it is more a hardware-software approach than a pure hardware approach because the hardware core of the processors has been reduced by means of a software layer called "Code Morphing".

The development of processor frequency and voltage scaling is an effective way to decrease consumption. In fact, the CPU is a major power consumer. Besides, as the L2 cache and the main memory are not affected by this frequency and voltage scaling, increasing the performance of the CPU may not provide any benefit due to their "performance saturation". To sum up, Dynamic Voltage Scaling (DVS) relies on the two following principles: the peak frequency of the processor is proportional to the supply voltage and the energy is proportional to the square of this supply voltage. DVS can be operated at several levels. At the processor level, Intel has proposed the SpeedStep technology and Transmeta's Crusoe processors offer the LongRun 2 power management. At the OS layer, [99] describes a power management patch of Linux called Vertigo. At the scheduling policy level, in [100] and [101], researchers focus on "frequency voltage scheduling" instead of the classical "task scheduling". Finally, energy consumption widely depends on the application characteristics. It is possible to implement DVS at the programming level or to tune a frequency scheduling according to a class of applications. In the first case, we can cite [102] which divides an MPI program into several blocks running at different gears. The second case deals with parallel sparse applications [103] and consists in decreasing the CPU performance of nodes which are not in the critical path. In [104], the authors propose a more general algorithm exploiting the slack-time due to the unbalanced workload of nodes.

Then, a lot of work aim to optimize the disk energy consumption. For instance, in [105], the speed of the disk is modulated. In [106], cache optimizations allow putting the disk in stand-by mode until a cache-miss occurs. For disk array-based servers, an interesting track consists of moving popular data on a subset of disks in order to put the others in low-power mode [107]. Next, [108] presents an orthogonal approach by switching on/off PCs of a cluster depending on performance and consumption requirements.

A global approach is to build a whole supercomputer/cluster with power efficient architecture and using low-power components. We can cite Green Destiny [109], Mega-proto [110] and BlueGene/L [111].

Finally, we underline that all previous pieces of work deeply rely on profiling and predicting tools. [112] presents a framework measuring the power-performance efficiency of the

NAS parallel benchmarks. The profiling is done for the CPU, the memory, the disk and the network interface. [113] focuses on the Intel P4 processor. It proposes real power measurements and above all, an interesting power estimation of the processor using counter-based measurements on 22 physical sub-components.

5.3 A DVS-aware experimental platform

Our study on Dynamic Voltage Scaling in order to take advantage of the slack-time of nodes, targets world-wide Grid platforms. Unfortunately, this topic of research is new and it does not exist yet such large platforms allowing performing DVS and measuring the power consumption on each node. We must use a smaller dedicated cluster described in this Section. There is no communication on the Internet. Thus, we cannot focus on the slack-time due to the high latency and the low bandwidth of the Internet. As we focus only on the slack-time due to the heterogeneity of the nodes from a CPU point of view, it is conceivable to base our study on a small DVS-aware cluster because our experimental application is designed for large Grid Computing and it can be applied without or with few modifications on such Grid.

Researchers of the HPCS laboratory of Tsukuba have built a system using Hall elements [114] which enables to measure dynamically the power consumption without modifying the hardware of machines. The Hall device has A/D converters to digitize each instant measured power and to transfer it to a dedicated PC. Measurements can be done with a very high time resolution (up to a few micro-seconds). The experimental cluster is composed of 16 nodes whose specifications are given in Table 5.1.

CPU	AMD Opteron
Clock	2200MHz
L1/L2 cache	128KB/1MB
Memory	1GB (DDR)
Network	1 GB Ethernet
OS Linux	Red Hat, kernel 2.6.11
gcc	4.1.2
MPI	LAM/MPI v7.1.3

Table 5.1: Cluster node characteristics

The Dynamic Voltage Scaling is done with functions wrapping *PowerNow!* instructions. For instance, the function *powernowset(int x)* makes the CPU working at the frequency given by the ratio $x\%$ in Table 5.2. In the remainder of the Chapter, when we say that a node is running at $x\%$ of the CPU frequency, we actually refer to the frequency of the ratio $x\%$ in Table 5.2 (which is not exactly $x\%$ of $2200MHz$). We recall that the power is a function of the frequency and of the voltage: $P = CV^2{}_{dd}f + \beta V^2{}_{dd}$ where C is the

capacitance, $V^2{}_{dd}$ is the supply voltage, f is the frequency and β is temperature- and process-dependent.

Ratio (%)	Frequency (MHz)
100	2200
90	2000
80	1800
60	1600
40	1400
20	1200
0	1000

Table 5.2: Table of available frequencies

5.4 A DVS-capable real symmetric eigensolver

The API which is wrapping *PowerNow!* instructions, is available for MPI implementations. Therefore, we have just modified our real symmetric eigensolver presented in Chapter 4 at the communication programming model level. There is no modification from a workflow point of view. As a consequence, among the 16 nodes of the experimental cluster, we consider 1 client and 15 remote computing nodes. Besides, the distribution of data and the distribution of the workload have not been modified.

At the beginning of the eigensolver, we assign a default CPU frequency to the 15 computing nodes. Those nodes do not have necessary the same default frequencies. We choose one of the values available in Table 5.2. Then, we add DVS routines before and after all MPI wait and MPI blocking communication routines (MPI_Wait, MPI_Gather, MPI_Bcast). We can modify the frequencies during those MPI calls depending on the purpose of our research. On each node, the modified frequency must remain lower or equal to the default frequency.

5.5 Experiments

The heterogeneity is a major difference between world-wide Grid Computing and Cluster Computing or High Performance Computing. We study the usage of DVS in order to exploit the slack-time due to the heterogeneity between the nodes from a CPU frequency point of view.

5.5.1 Experimental numerical settings

We use a sparse real symmetric matrix of order $N = 32490$. It is a tiling of the matrix *bcsstk*09 from the Harwell-Boeing matrix collection of the Matrix-Market website[1]. It has 16.5 million elements. The average non-zero elements per row is 510. Regarding the parameters of our eigensolver, we have chosen $m = 20$ and $k = 2$. In other words, we compute two Ritz eigenpairs and the order of the Krylov subspace is 20. We use only one couple (m, k) because the impact of those parameters on the execution wall-clock time has already been studied in Chapter 4 and it does not enter in the scope of this Chapter.

5.5.2 Interest of DVS during communication and idle times

Goal We first evaluate the impact of the heterogeneity of the nodes from a CPU frequency point of view on the global energy consumption. Then, we evaluate how much we can save energy by means of DVS only during MPI wait and MPI blocking communication routines. Indeed, in order to save energy in presence of heterogeneous resources, the first idea is to reduce the frequency of the fastest nodes when they have finished their computations or when they are waiting for data from other nodes.

Protocol of experiments Before MPI wait and MPI blocking communication routines, we decrease the CPU frequency with the function *powernowset(low_frequency)* and after, we recover the default frequency with the same function *powernowset(default_frequency)*. Table 5.3 shows the default frequencies (set at the beginning of the eigensolver) of the nodes for three experimental configurations: one is homogeneous, two are heterogeneous. When DVS is used, the frequency is lowered to the frequency of the burden nodes (0% for configuration B, 40% for configuration C). The client always stays at 100% of the CPU frequency.

	CPU freq. (%) of the client	Default CPU freq. (%) of the 15 workers
Configuration A	100	15 at 100
Configuration B	100	5 at 100, 1 at 80, 2 at 60, 2 at 40, 1 at 20, 4 at 0
Configuration C	100	5 at 100, 1 at 80, 2 at 60, 7 at 40

Table 5.3: Configurations of the heterogeneous platforms

[1] http://math.nist.gov/MatrixMarket/

Results Table 5.4 presents the wall-clock times (seconds), the energy consumptions ($J = W.s$) and four ratios:

- for each DVS strategy, columns 4 and 5 show the increases of the wall-clock times and the global energy consumptions for the configurations B and C, compared to the configuration A. For each kind of DVS strategy, those ratios allow evaluating the impact of the heterogeneity of the platforms (from a CPU frequency point of view) on the wall-clock times and on the energy consumptions.
- for each one of the three configurations, the two last columns (6 and 7) present the variations of the wall-clock times and the energy consumptions when we apply a DVS strategy compared to the same configuration but without DVS strategy. Those ratios aim to evaluate the impact of DVS during MPI wait and blocking communication routines for each kind of heterogeneous platform.

Each entry, in the second and third columns of the table, represents an average of 10 measurements.

Configuration - DVS strategy	Time (sec)	Energy consumption (W.s)	Time (%) compared with A	Energy (%) compared with A	Time (%) compared with no DVS	Energy (%) compared with no DVS
A - No DVS	490	773041	-	-	-	-
B - No DVS	897	1176348	+83	+52	-	-
C - No DVS	684	943448	+39	+22	-	-
A - DVS during bcast	491	771671	-	-	$+ < 1$	$- < 1$
B - DVS during bcast	897	1169854	+82	+51	$+ < 1$	$- < 1$
C - DVS during bcast	682	936067	+38	+21	$- < 1$	$- < 1$
A - DVS during gather	492	775315	-	-	$+ < 1$	$+ < 1$
B - DVS during gather	897	1174100	+82	+51	$+ < 1$	$- < 1$
C - DVS during gather	681	936363	+38	+20	$- < 1$	$- < 1$
A - DVS during wait	491	772396	-	-	$+ < 1$	$- < 1$
B - DVS during wait	903	1173586	+83	+51	$+ < 1$	$- < 1$
C - DVS during wait	688	942467	+40	+22	$+ < 1$	$- < 1$

Table 5.4: DVS on heterogeneous platforms during blocking communication routines and idle times of nodes

Analysis of results First, we focus on the five first columns of Table 5.4 and consider separately the four DVS strategies. We observe the great impact of the heterogeneity of the resources. For each strategy, when the heterogeneity between the slower and the faster nodes is increased, both the rise of the wall-clock times and the energy consumptions are increased. The ratios are similar for the four DVS strategies. Between the configurations

A and C (resp. A and B) the gap of frequency is 100 (resp. 60). The increases of the wall-clock times are 82% or 83% (resp. 38% , 39% or 40%) and the increases of the energy consumptions are 50% or 51 % (resp. 20%, 21% or 22%). Indeed, our eigensolver is very sensitive to the heterogeneity since it contains much communication followed by global synchronizations. So, the wall-clock time is linked to the performances of the burden nodes. As the wall-clock times are much larger, the global energy consumptions are higher.

Let us look at the two last columns of Table 5.4 in order to observe the impact of DVS for each kind of heterogeneous platform. The reduction of the frequency during those periods does not impact the wall-clock times but it does not provide any interesting energy saving. Several factors explain this absence of significant energy savings. The volume of communication is small because of our data persistence strategy and the LAN has a high and constant bandwidth, a low latency. Besides, our eigensolver has many global synchronizations. So, there are a many short waiting times and the processes are rather synchronized. Finally, changing the frequency of a processor is not instantaneous. For instance, it takes $50\mu s$ in order to change from 100% to 10 %. Thus, we cannot take advantage of very short communication times and idle times.

In [104], the authors have studied the slack-time due to an unbalanced distribution of work in the context of Cluster Computing. They also do not take advantage of DVS when it is applied only during idle times. They apply DVS during all the computations.

5.5.3 Levelling off the default frequency of the nodes in order to save energy

Goal We change our strategy. Instead of using DVS during short and multiple periods of slack-times, we try to avoid slack-times by levelling off the CPU frequencies of the fastest nodes to the CPU frequency of the slowest node (it is done during all the execution). In other words, by means of DVS, we make homogeneous, an heterogeneous platform from a CPU frequency point of view.

Protocol of experiments We consider six heterogeneous configurations of platforms. Each platform has one burden node running at a low frequency whereas the other nodes (14 workers and the client) are at 100% of the CPU frequency. It is probably the worst case of heterogeneity if we have energy consumption considerations. The burden node is running either at 0, 20, 40, 60, 80 or 90% of the highest CPU frequency.

For each one of the six configurations, we first perform 10 executions without using DVS and we present average values of data in Table 5.5. Then, we perform 10 experiments by levelling off the CPU frequencies by means of DVS. We also present average values of data in Table 5.5.

Results Table 5.5 shows the wall-clock times (in seconds), the global energy consumptions (in $J = W.s$) and the average power consumptions per node (in W). Table 5.6 presents ratios computed by means of data of Table 5.5. Columns 2, 3 and 4 compare data of the six heterogeneous configurations with the homogeneous configuration where all nodes are at 100% of the CPU frequency. Those ratios are variations of the wall-clock times, variations of the global energy consumptions and variations of the power consumptions per node. In columns 5, 6 and 7 of Table 5.6, for each configuration, similar ratios compare executions without DVS and with DVS (i.e. all the workers have their CPU frequency lowered to the same level as the burden node).

CPU frequencies (%) of the 15 workers	Time (sec)	Energy consumption (W.s)	Average power consumption per node (W)
15 at 100	490	773041	98
1 bottleneck at 90, 14 at 100	514	802524	97
15 at 90	528	796492	93
1 bottleneck at 80, 14 at 100	552	843519	95
15 at 80	568	818277	89
1 bottleneck at 60, 14 at 100	601	899867	93
15 at 60	619	853515	85
1 bottleneck 40, 14 at 100	680	1000250	91
15 at 40	689	914133	82
1 bottleneck 20, 14 at 100	758	1085626	89
15 at 20	776	992403	79
1 bottleneck at 0, 14 at 100	882	1227917	86
15 at 0	903	1114730	76

Table 5.5: DVS on heterogeneous platforms during all the execution - results of tests

Analysis of results First of all, the results confirm the great impact of the heterogeneity (from a CPU frequency point of view) of a distributed platform on the global energy consumption. Even if there is only one heterogeneous node, it drastically increases the wall-clock times (up to +80% in the worst case) compared to the fastest homogeneous configuration (all nodes at 100% of the CPU frequency). As a consequence, we observe a significant rise of the global energy consumptions (up to 58%). The wall-clock times and the energy consumptions depend on the performance of the unique burden node: it confirms the ratios of columns 4 and 5 of Table 5.4. This conclusion is acceptable for applications having much communication and many synchronization points such as our eigensolver.

If we level off all the nodes to the same CPU frequency as the slowest node one, it increases the wall-clock time only by 1 or 2% but it saves up to 9% of the global energy consumption. We underline that the increase of the wall-clock time is stable for any kind

CPU freq. (%) of the 15 workers	Time (%)	Ratio of energy consumption (%)	Average ratio of power consumption per node (%)	Time (%)	Ratio of energy consumption (%)	Average ratio of power consumption per node (%)
	compared with all nodes at 100%			compared with a configuration with only one bottleneck		
15 at 100	-	-	-	-	-	-
1 at 90, 14 at 100	+4	+3	< -1	-	-	-
15 at 90	+7	+3	-5	+2	< 1	-4
1 at 80, 14 at 100	+12	+9	-3	-	-	-
15 at 80	+15	+5	-9	+2	-3	-6
1 at 60, 14 at 100	+22	+16	-5	-	-	-
15 at 60	+26	+10	-13	+2	-5	-8
1 at 40, 14 at 100	+38	+29	-7	-	-	-
15 at 40	+40	+18	-16	+1	-8	-9
1 at 20, 14 at 100	+54	+40	-9	-	-	-
15 at 20	+58	+28	-19	+2	-8	-11
1 at 0, 14 at 100	+80	+58	-12	-	-	-
15 at 0	+84	+44	-22	+2	-9	-11

Table 5.6: DVS on heterogeneous platforms during all the execution - ratios of results

of heterogeneity whereas the global energy saving is increasing with the heterogeneity. It is particularly interesting for experimentations on a real heterogeneous world-wide Grid because we can harness resources with a higher heterogeneity. We may forecast similar increases of wall-clock times (+1 or +2 %) but higher energy savings by adjusting all the CPU frequencies to the frequency of the burden node.

Next, the local power consumption reductions are very interesting too. Indeed, at first sight with ratios of column 7 in Table 5.6, we see that the average power consumptions per node are decreased by a factor from 4 to 11% when we use DVS. Actually, the power consumption reductions are much more interesting. In fact, data of column 4 of Table 5.5 is influenced by the consumption of the burden node and is used to compute ratios of column 7 in Table 5.6. It does not show the real power consumption reductions for the 14 fastest nodes (originally at 100% of the CPU). Their power consumptions are decreased by a factor from -5% to -22% as shown by column 4 in Table 5.6. For instance, let us consider the configuration with one bottleneck at 20% of the CPU. The average power consumption per node is $89W$. Actually, 14 nodes run at 100% of the CPU and consume $98W$ whereas the burden node uses $79W$. Thus, if we level off the CPU frequency, the power consumption of 14 nodes changes from $98W$ to $79W$ (-19%).

Finally, since the wall-clock time is only increased by 1 or 2%, it means that 14 nodes save locally a lot of energy. For instance, if we consider again the configuration with one bottleneck at 20% of the CPU, a node (initially running at 100%) saves 17% of energy

because its energy consumption varies from $74284J$ $(= 758 * 98)$ to $61304J$ $(= 776 * 79)$. Table 5.7 gives the ratios of the local energy savings of the fastest nodes, initially at 100% of the CPU.

CPU decreased from 100% to	Energy savings (%)
90	2
80	6
60	10
40	15
20	17
0	20

Table 5.7: Local energy savings of the fastest nodes by levelling off their CPU frequencies

5.6 Conclusion and perspective

When the resources have heterogeneous CPU frequencies, using DVS only during the slack-times does not penalize the wall-clock times but it does not provide significant reductions of energy consumption. This conclusion is acceptable for applications having much communication with synchronization points such as our real symmetric eigensolver. If DVS is applied during all the execution, it becomes very worthwhile. In fact, we get important global energy savings without a significant rise of the wall-clock times. For instance, if we decrease the CPU frequency of the nodes from 2200MHz to 1400MHz, we save 8% of energy consumption whereas the wall-clock time is increased by only 1%. With all CPUs set to 1000MHz the energy saving is 9% and the increase of time 2%. The local energy savings of the fastest nodes are even more interesting when we adjust their frequencies. In fact, their power consumptions are decreased from 5 to 22%. As the wall-clock times remain stable (+1 or +2%), their energy consumptions are significantly reduced (up to 20%).

It would be very interesting to study the slack-time due to the heterogeneity of other components of the workers such as the memory bandwidth, the speed of disk access (particularly as we do out-of-core). Finally, by using an emulator such as Modelnet, we would like to evaluate the energy savings if we apply DVS in order to take advantage of the slack-time due to communication on the Internet. In fact, the Internet is composed of many heterogeneous sub-networks with low bandwidths and high latencies and we expect significant slack-times. As the experimental cluster does not use the Internet, we propose to add a 17^{th} node in charge of emulating a large-scale network.

We think that power conservation will be soon a hot topic of research on Global Computing. Indeed, the community of Global Computing will be progressively interested in work

done on low-power devices and power-aware mechanisms for High Performance Computing. Actually, new technologies often appear with HPC before being used at a large scale. The current race for petaflopic systems does not ignore the quest to energy conservative systems. As an example, we can cite the 10-petaflops system of the RIKEN. Its specifications target a reduction by a factor of 10% compared to a 10-petaflops machine built with standard components.

Chapter 6

YML, a Global Computing framework for the conception of very large applications

6.1 Introduction to YML

The YML project has been launched in 2002 at the ASCI CNRS laboratory in the frame of the ACI GRID CGP2P. YML is mainly developed at the PRiSM laboratory at the University of Versailles Saint-Quentin-en-Yvelines by Olivier Delannoy. The MAP team of the LIFL at the USTL (University of Lille 1) is also involved in this project.

6.1.1 Motivations of YML

Designing an efficient parallel and distributed implementation of a sequential problem is a difficult task. In addition, we often face many difficulties which are not linked to the application. The goal of the YML framework is to tackle and hide most of those additional constraints so that we only focus on algorithmic details.

The Global Computing model is promising because a huge number of machines are connected to the Internet and they are potentially available for computations. However, those machines are managed by a wide range of different middleware which have different API. If the user wants to harness many computing resources, he has to know many programming interfaces and he must implement several times the same application with different API. This is hardly conceivable. Even if the client was able to do it, the execution of an implementation using a given API would only harness the nodes managed by the corresponding middleware. Thus, the first goal of YML is to propose a unique API to

the client allowing using the nodes of any kind of middleware of Global Computing. The second service enables to schedule the execution of the tasks of a program on different middleware. YML becomes at the runtime a bridge on top of many middleware. By means of those two services, the heterogeneity of the Global Computing middleware is hidden to the client (at a lowest level, the heterogeneity of the computing resources is hidden by the computing middleware, not by YML). Distributed and parallel programming is a difficult task. YML tries to help the client as much as possible. For instance, YML adopts a component programming model and uses a catalogue of components. It makes easier the re-usability of existing, efficient and bug-free components.

6.1.2 Main concepts of YML

6.1.2.1 Workflow representation and graph description language

YML proposes an intuitive representation of a distributed and parallel application by means of a workflow. An YML workflow consists of a graph whose vertices are independent and communication-less computing tasks. The edges of the graph represent the precedence relationships between the tasks. YML provides a graph description language called YvetteML. With this language, we can build the DAG traducing the precedences between the tasks. This DAG represents a control flow, and not a data flow, since a precedence does not necessary concern a data dependency. The main structures of YvetteML are: the component call, the parallel sections, the sequential loops, the parallelized loops, the conditional branch and the event notification/reception.

6.1.2.2 Component model

YML has chosen the component model in order to represent each task. The re-usability is the main motivation. Besides, this kind of representation forces the client to design a well distributed application by clearly separating the computational blocks, communication and by finding the dependences. Each computational task is described by an "abstract component" and its implementation is written in an "implementation component". The principle is similar to CORBA which separates the IDL description of the object from its implementation. Those middleware-independent components are stored in the Development Catalogue. The components are written by means of an XML syntax.

6.1.3 Architecture of YML

Figure 6.1 presents the architecture of YML. We notice three entities. First, there is the client who is providing the computational components (abstract and implementation

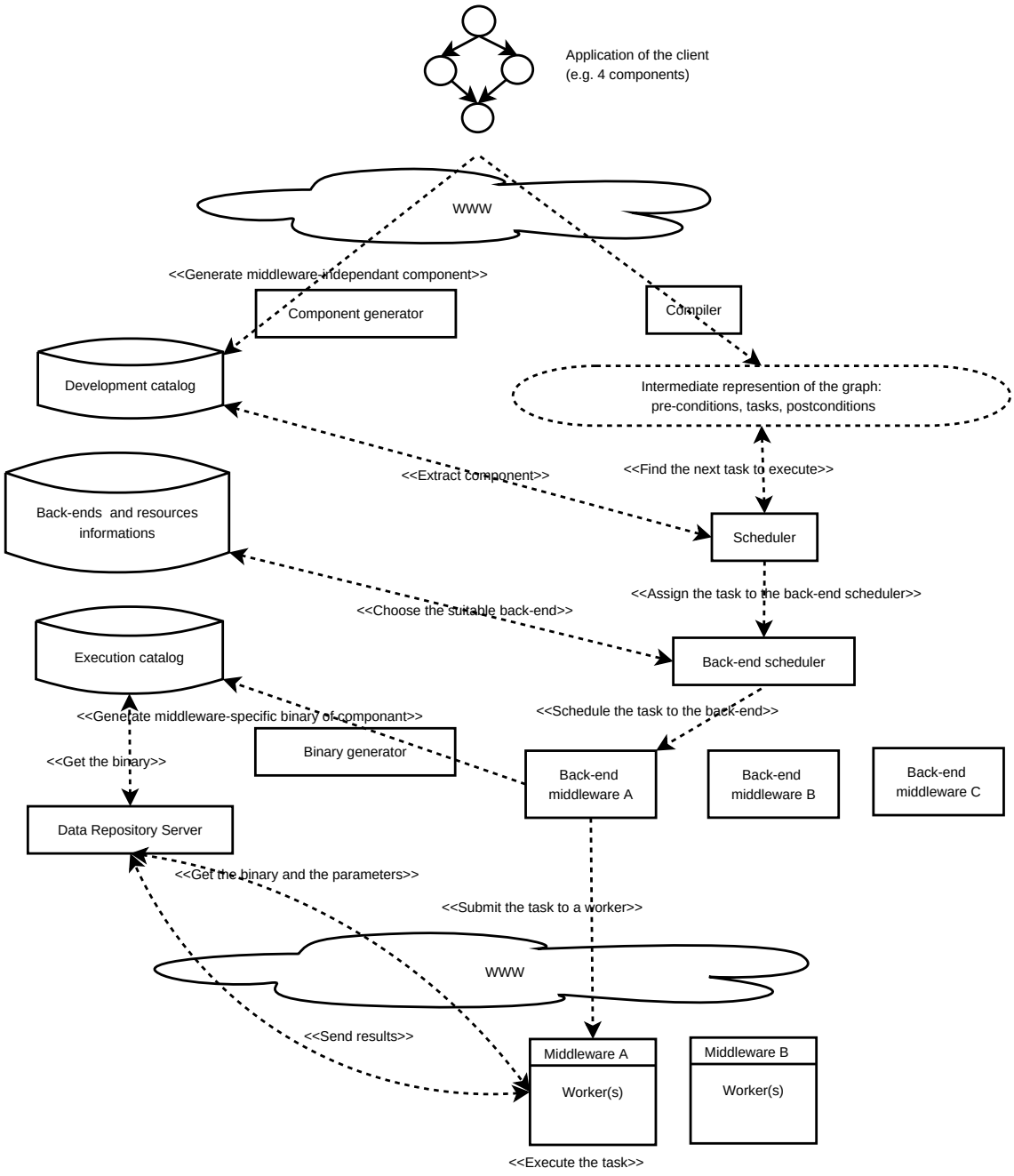

Figure 6.1: Architecture of YML

ones) and the application graph expressing the control workflow. Second, there are the volunteer workers which are using any kind of computing middleware as long as an YML back-end enables to handle it. Finally, we consider YML which is hiding to the client the complexity and the heterogeneity of the Global Computing platforms. Inside YML, we can consider a front-end layer and a back-end layer.

The component generator of YML checks the semantic validity of the tasks (vertices of the application graph) and stores them in the Development Catalogue. There is no middleware consideration at this stage. The YML compiler checks the validity of the application graph and generates an intermediate representation which is handled by the scheduler. The intermediate representation of YML is based on the event mechanism. An event can be assimilated as a boolean. A combination of events allows making pre- and post-conditions in order to formulate the dependencies between the tasks. By means of those pre- and post-conditions, the real-time scheduler manages the execution of the intermediate representation by choosing the next task to run.

This elected task (whose pre-conditions are validated) is given to the back-end layer. The back-end layer is roughly divided in two parts. First, there is the Back-end Scheduler which schedules the current task to the appropriate back-end (in other words, to the appropriate computing and network resources). The scheduling decision relies on static and dynamic information extracted from a database. Such information concerns the middleware, the computing and network characteristics of resources, some statistics related to past executions, etc. Second, the middleware-specific back-end is in charge of submitting the task to an YML worker which is running on a computing resource handled by the computing middleware associated to the back-end. At this stage, the targeted middleware is known. YML can generate the suitable "binary component" to execute. This binary is stored in the Execution Catalogue. Then, the YML worker asks the data-repository server (DR Server) for the binary and the parameters and then, it downloads them. It also manages the execution of the binary on the computing node. At the end of the computations, it sends the results to the DR server. We underline that we can run the DR server and store data on a different location from the YML compiling and scheduling units. It limits the risk of bottleneck.

6.2 Main contributions to YML

In addition to my work on linear algebra problems on Global Computing platforms, I contribute to the development of YML. On the one hand, I develop some modules of this framework and, on the other hand, I test YML and provide a user feedback before releasing new versions. In particular, I unify my work on linear algebra problems on Global Computing platforms and the implementation of some YML modules by adapting the real symmetric eigenproblem to the YvetteML language. Therefore, I deliver an eigensolver which can be run on all Global Computing platforms supported by YML.

6.2.1 Development of YML modules

6.2.1.1 OmniRPC back-end

As my previous work on Global Computing mainly uses the OmniRPC Grid middleware, I have first developed its back-end for YML. Actually, it basically consists in forwarding the requests of the YML scheduler to the OmniRPC platform by means of asynchronous RPC calls. The OmniRPC back-end manages a queue of YML tasks and probes their completion. It returns to the YML scheduler the completed tasks. The current back-end lets the client choose between the "SSH configuration" and the "cluster configuration". It currently does not handle the data-persistence feature offered by OmniRPC.

We consider implementing the support of data-persistence. In fact, YML will soon have an intermediate level of scheduling coupled with an efficient information service between the YML scheduler and the back-ends (we have described it in Figure 6.1 and Section 6.1.3). So, the user will be able to select a specific back-end by providing some criteria such as the data persistence. Such scheduler and information services are planed in a future release. We started designing the architecture and implementing it.

6.2.1.2 YML Worker and Data Repository modules

In parallel, I have focused on the YML Worker module. I underline that the YML Worker is used by any kind of computing middleware. The YML Worker is deeply linked with the DR Server. Therefore, I developed this module too. Figure 6.2 shows a detailed architecture of the three modules and their main interactions. The RPC stub on the computing node only gets from the OmniRPC back-end the identifier of the component binary and the identifiers of the input parameters. Then the stub runs the YML Worker. In the case of XtremWeb, the XtremWeb back-end submits to the dispatcher a binary of the YML Worker and the parameters which are the identifier of the binary (of the component) and the identifiers of the related parameters. Thus, an XtremWeb worker can download the YML Worker. Once the YML Worker is executed, it creates a data-repository client (DR Client) in order to interface with the DR Server unit. Actually, the DR Client is in contact with the DR Server only for the initial connection. Indeed, the DR Server manages a pool of threads and each thread handles the requests of a DR Client. This mechanism allows the data-repository server unit accepting simultaneous requests. Communication is over TCP. Once the DR Client has downloaded the binary and the parameters, the YML Worker executes this binary. At the end, the DR Client sends the results to its thread of the data-repository unit. Before completing its execution, the YML Worker cleans all the traces of the execution.

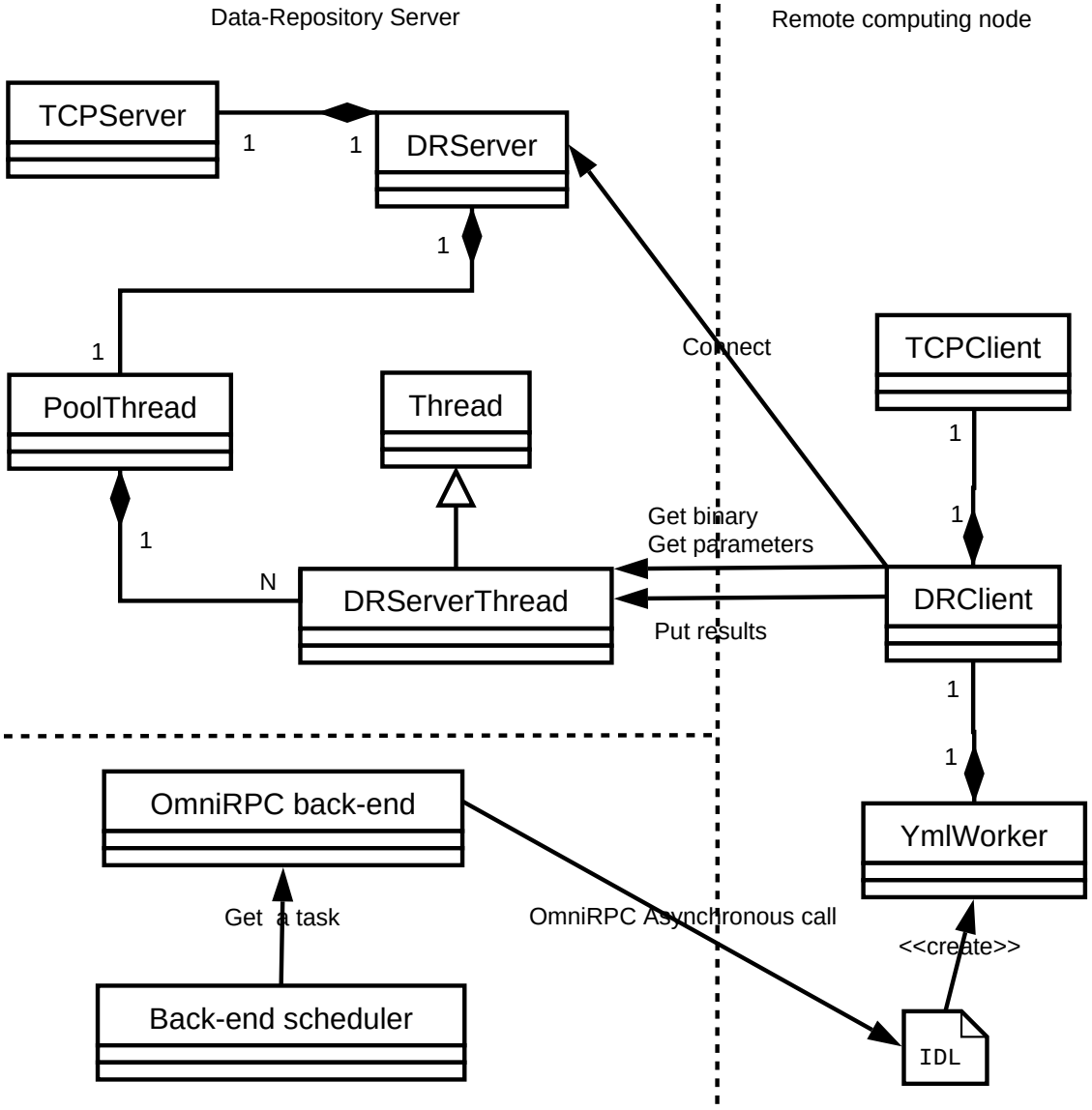

Figure 6.2: Interactions between the OmniRPC back-end, the YML Worker and the DR Server

6.2.2 Adapting the real symmetric eigenproblem to YvetteML

6.2.2.1 A need for new features of YML

At the first stage of the adaptation of our eigensolver to YvetteML, we have expressed our needs to the responsible of YML. Thus, he has developed the following new functionalities and we have tested them. First, an YvetteML graph is now able to take some input parameters, like files or constant values, and to return output parameters. Previously, data were given as constants inside the YvetteML program and the output results were written in a log file. In our case, we target to use several large vectors and matrices. It is not conceivable to write each matrix as a constant. The unique conceivable solution is to give a file name to the YvetteML graph.

Second, adapting the eigenproblem on a computing device requires manipulating many different kinds of data structures like blocks of a matrix, vectors. Even inside a data structure, it can be necessary to gather some elements by family depending on their role in the algorithm. For instance, if we consider a type "vector", it can be interesting to distinguish the vectors of the basis of the Krylov subspace and the Ritz eigenvectors. Thus, we have designed with the developer of YML a structure called "Collection" in order to gather in a common container many elements of the same family. Inside a collection, the elements are ordered. So, it is possible to build associations between the elements of different collections. For instance, from a collection of Ritz eigenvalues and a collection of Ritz eigenvectors, we can easily and quickly get the Ritz eigenpairs. Finally, the YML Collection API also allows doing out-of-core. In fact, each element of a collection is stored in a file and is loaded only when required.

6.2.2.2 Main differences between an implementation with YML and with an RPC programming Grid software

From an algorithmic point of view, there are no modification compared to the work done in Chapter 4. From the parallel and distributed implementation point of view, we notice also few modifications. Indeed, in Chapter 4, we avoided as far as possible making, middleware-dependant assumptions. In this section, we present some modifications in the implementation of the eigensolver and we comment the main differences of methodology in order to do such an implementation.

In previous Chapter, we decided to use data persistence. Its implementation was middleware-dependant since we used a feature of OmniRPC. The current status of the YML does not provide data-persistence. Thus, we must expect larger volume of communication. We must carefully weight the pros and cons before distributing and parallelizing a computation because the communication times may hide the reductions of the remote computing times (it can be even longer). With YML, data are stored by the data-repository unit

by means of files and then, the files are downloaded by the YML Workers. It is a major difference with the implementation on a RPC computing middleware such as OmniRPC. In order to manage data, the client of YML has to create its own data types with import and export functionalities which transfer data from a file to the memory (and reversely). For instance, regarding the matrix A of the eigenproblem, we have built a type using two compact formats: a Matrix Market compact format for the file storage and a personal compact format for the memory storage. A significant advantage of this mechanism is the reduction of the memory constraint.

In a context of world-wide Grid Computing (as in Chapter 4), the client uses a given number of workers. He tries to reach a full parallelism for his application. With a high-level framework such as YML, like for a Peer-to-Peer Computing middleware such as XtremWeb, the client has no idea of the identity and the number of the workers. The resource level is entirely hidden by the computing software. Thus, the client must focus more on the distribution of work than the parallelization. Indeed, even if the client submits at the same time several independent tasks, nothing ensures a fully parallel execution at the resource level. It is another reason explaining that high performance is difficult to reach in our context.

In previous chapter, the data persistence of the blocks of the matrices A et Q was one reason for not fully parallelizing the outer loops of the computations of the Ritz eigenvectors and then, the computations of the residuals. In fact, in order to parallelize those steps, we would have to duplicate the blocks of A and Q and it would have generated much communication (particularly for Q). In the case of YML, there is no data persistence. The blocks of A and Q are always transferred from the data repository unit to the YML Worker. Therefore, if we suppose that a fully parallelism can be reached at the computing middleware level, it can be interesting to parallelize those two outer loops (both have k loops). Of course, as said in previous paragraph, a full parallelism in not ensured. We have implemented parallel and sequential versions of those outer loops (see Figures 6.3 and 6.4).

The YvetteML compiler only handles static graphs. In other word, the structure of the graph must not depend on variables whose values are not known at the compilation time. With a traditional programming tool, we can generally create loops, branch conditions relying on variables such as m, k. In the case of YvetteML, we must fix the values of those variables playing a role in loops or parallel sections. It is not a strong constraint but it is not convenient to recompile the program for each configuration. Indeed the compilation is particularly long since the YvetteML compiler unrolls entirely the execution graph. It is more problematic for branch conditions. For instance, in case of convergence of the eigensolver, we must throw an exception in order to stop the execution.

In addition, in the YvetteML program, it is impossible to make any computation. Therefore, even for a basic operation such as an addition we must use a component. There are two approaches. On the one hand, we had better aggregate as much as possible some operations in the same components so as to limit the number of communications. On

the other hand, if we target to favour the re-usability of components, it is much better to clearly make one component per kind of computation. We have chosen the second approach in order to re-use a maximum of component in the implementation of other YvetteML programs such as one solving the sparse real eigenproblem by means of the Arnoldi method. The drawback is that we can expect high communication times.

6.2.2.3 Main components of the eigensolver

The YvetteML real symmetric eigensolver uses 19 components. We present only the most important ones.

Sharing the matrix The first concerns the partitioning step of the real symmetric matrix. We propose to send this work to only one node instead of parallelizing this component. Thus, we send the matrix A and we get some blocks of rows of A. Data are transferred in a compact format because we use our own "matrix" data type which is closed to the Matrix Market compact format (blocks of A are matrices too). Import and export functions of this data type are responsible for importing data from text file to main memory and reversely (always in compact format). Regarding the cost of communication over the Internet, the size of data, the overhead of YML and the "centralized" architecture of the YML DR Server, we think this solution is better than a parallel matrix partitioning. In other words, by parallelizing this step, we estimate the gain of remote computing times would be hidden by much larger communication times so that the wall-clock time would be larger.

Let us note d the number of components on the diagonal of A and e the number of components under this diagonal. With our choice, a unique YML Worker gets A only one time in a compact format ($d+e$ doubles and $2(d+e)$ integers). Then, it returns x blocks of rows also in a compact format ($x(d+2e)$ doubles and $2x(d+2e)$ integers). With a parallel and distributed partitioning, the matrix A would be transferred x times in a compact format ($x(d+e)$ doubles and $2x(d+e)$ integers). The number of blocks returned by the x YML workers would be the same. Thus, our solution saves $(x-1)$ transfers of A: $(x-1)(d+e)$ doubles and $2(x-1)(d+e)$ integers. Besides, YML is a good tool for the distribution of work, but it does not ensure the parallelization of the tasks at the middleware computing level. Thus, even if we implement a parallel version of the matrix partitioning step, the tasks would probably not be executed fully in parallel. However, it would be interesting to test the parallel version with a broadcasting method.

The matrix-vector product The second important component deals with the matrix-vector product. When we perform this linear algebra operation, we execute x calls of this component in parallel, where x is the number of blocks of rows of the matrix. Only the input block of rows and the output vector differ from one call to another. In Chapter

4, we have stressed the cost of the MVP step. Thus, it is interesting to parallelize and distribute this step among x workers. Besides, for a parallel and distributed MVP, the total volume of transferred data is just a bit larger than for a unique call performing the whole MVP on one YML worker. Let us note again d the number of components on the diagonal of A and e the number of components under this diagonal. With a distributed MVP, we send $(d + 2e)$ doubles and $2(d + 2e)$ integers instead of $(d + e)$ doubles and $2(d + e)$ integers for a sequential MVP on one YML worker. Thus, the transmission and propagation times do not differ a lot compared to the large computing times.

Another motivation for distributing the MVP is the memory usage. Our current implementation of the "matrix" data type does not allow performing out-of-core (a block of rows is considered as a matrix). Although, data are first loaded in memory using a compact format, a very large matrix can hardly fit in memory. It is necessary to distribute A by blocks of rows among more workers. We currently share A by means of an YML Collection of blocks of rows. Thus the grain for the out-of-core mechanism is a block (of rows). When we get a huge block of rows, it can be difficult to load it in memory too. So, we currently choose the number and size of blocks depending on the memory of nodes. This point has to be improved in order to get a scalable eigensolver. We must propose a finest grain at the row level. For instance, we will propose to represent each block of rows by an YML Collection of rows instead of using the "matrix type". In other words, A will be a collection of collection of rows.

Let us consider that an YML Worker can get the full YML Collection of blocks of rows and has enough memory to load a full block (stored in a compact format). It is able to perform alone the entire MVP by doing out-of-core because each element of the YML Collection is gradually loaded when needed. It would be interesting to experiment and compare it to the current distributed version.

Reorthogonalization The full-reorthogonalization is currently performed by only one component and by only one node. As we use a restarted scheme, the reorthogonalization is done with very few vectors of the basis of the Krylov subspace. Besides, the basis of the Krylov subspace is stored by using an YML Collection of rows so as to do out-of-core. Thus, a unique worker can handle those computations. However, we plan to propose a parallel reorthogonalization in order to solve very large eigenproblems with larger values of m. In Chapter 4, this step was parallelized. The total volume of transferred data does not differ significantly between a parallel and a non-parallel reorthogonalization. The first one adds some overheads.

Other components Then, we consider three components. The first performs the Bisection and the Inverse Iteration methods. We do only one call since we work in a small subspace and we need only to compute few eigenpairs. The second component deals with the selection of the candidate Ritz eigenvalues and the associated eigenvectors of the tridiagonal matrix T. As in Chapter 4, we select the highest (in module) eigenvalues. Finally,

we consider the computation on the new starting vector when a candidate Ritz eigenpair has not converged. Like in Chapter 4, this step is not parallelized. We perform a linear combination of the candidate Ritz eigenvectors (the Ritz eigenvalues are the factors).

The other components actually reproduce basic operations on numbers and vectors, or simply wrap BLAS calls (e.g. norm and scaling of a vector, dot product of two vectors, gather, reduction, etc.).

6.2.2.4 Workflow of the eigensolver

According to the principle of re-usability, many components are called several times. For instance, the MVP component is used for the Lanczos tridiagonalization and for the test of convergence $\|AU - \lambda U\| < \epsilon$. Some components are also used by other programs like an Arnoldi implementation on YvetteML. As shown in the Graphs 6.3 and 6.4, we designed two workflows which contain the following parallel sections/loops:

- the initialization of the eigensolver (partitioning of the matrix, initialization of vectors, etc). It is done only one time at the very beginning of the execution.
- the MVP of the Lanczos tridiagonalizations. It occurs $r * m$ times where m is the number of iterations of the Lanczos tridiagonalization (size of the Krylov subspace) and r the number of loops of the restart scheme.
- the computations of each Ritz eigenvector. It occurs $r * k$ times where k is the number of computed Ritz eigenpairs.
- the MVP of the convergence tests. It occurs at the most $r * k$ times.

The second graph proposes two additional parallel loops by parallelizing the outer loops containing the computations of all Ritz eigenvectors and the computations of all the residuals (and their norm).

The execution of the graph is distributed among the workers by using one (or several in a close future) back-end. In parallel sections or parallel loops, the YML Scheduler submits obviously parallel tasks. However, we underline that the tasks are not necessary solved in parallel depending on the back-end and the computing middleware which are used.

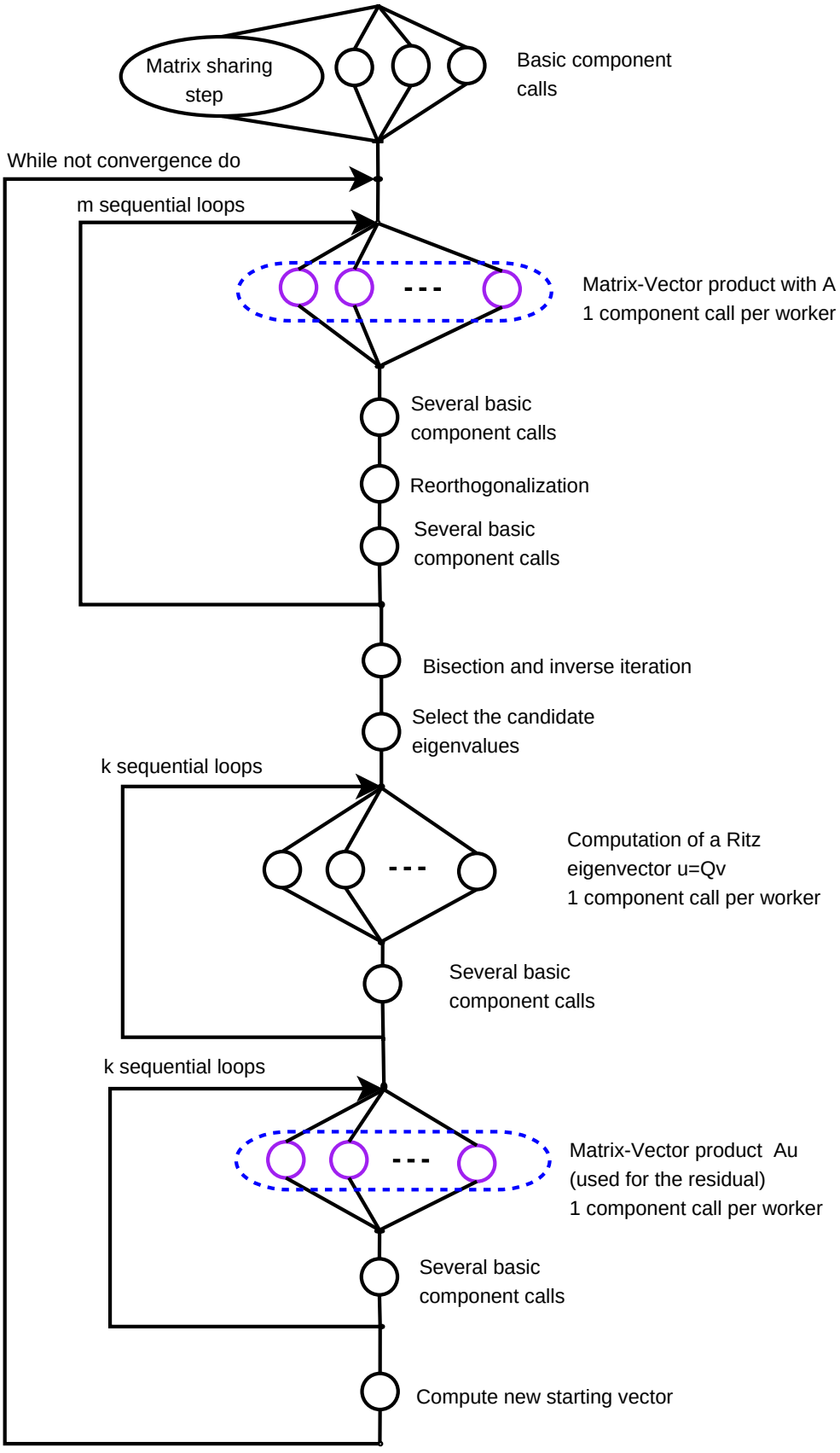

Figure 6.3: First YvetteML workflow of the real symmetric eigensolver

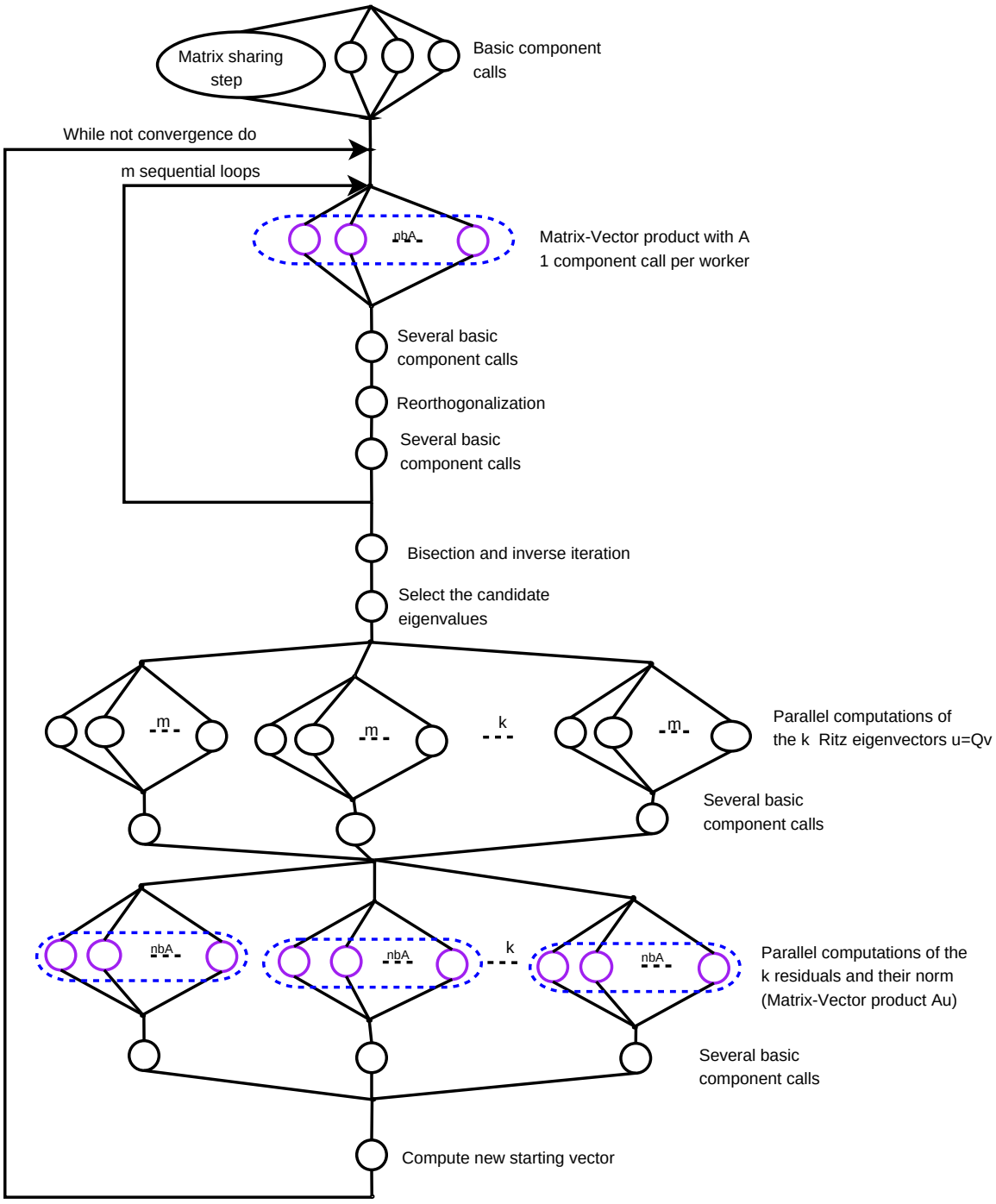

Figure 6.4: Optimized YvetteML workflow of the real symmetric eigensolver

6.2.3 Experimentations on Grid platforms

6.2.3.1 Objective of the experimentations

We use the OmniRPC back-end and the YvetteML workflow related to Figure 6.3. The main objective of those preliminary experimentations is to observe the overhead of the YML platform compared to similar tests done directly on top of the OmniRPC Global Computing middleware (the two execution workflows are similar).

Due to a lack of time, the number of workers is low and the size of the matrix is not very large. Thus, those tests do not emphasize neither the scalability of YML nor the scalability of our YvetteML eigensolver.

6.2.3.2 Platform and numerical settings

For those experiments, we harness the network of workstations at the University of Lille 1. The client, the YML Scheduler and the DR Server are located on a single powerful workstation, also at the University of Lille 1. YML uses the OmniRPC back-end with the "cluster configuration". The OmniRPC relay nodes are run at the University of Lille too. Communication does not go through the Internet but stays in the LAN of the University whose theoretic bandwidth varies from 10 to 100 Mbits. Table 6.1 presents the details of the platform settings.

	Configuration 1	Configuration 2
Middleware	OmniRPC	
Location of the client Location of the OmniRPC relay nodes Location of the workers	Lille Lille Lille	
Nb of OmniRPC servers	1	2
Nb workers per server	29	
Total nb workers	29	58
Number of blocks of A	29	58

Table 6.1: Experimental platforms of the preliminary tests on YML

Regarding the numerical settings, we use the same real symmetric matrix of order $N = 47792$ than in Chapter 4, Section 4.3.4. The dimension of the Krylov subspace is 15, 20 or 25. The number of computed eigenpairs is 1, 2 or 3.

m, k	Configuration 1	Configuration 2
15, 1	221	209
15, 2	323	236
15, 3	462	296
20, 1	197	146
20, 2	321	232
20, 3	368	297
25, 1	257	176
25, 2	267	186
25, 3	281	195

Table 6.2: Wall-clock times of the client to get the k Ritz eigenpairs (in minutes)

m, k	29 workers at Lille	58 workers at Lille
15, 1	20	21
15, 2	35	35
15, 3	51	47
20, 1	19	20
20, 2	30	35
20, 3	51	44
25, 1	29	28
25, 2	33	26
25, 3	55	43

Table 6.3: Wall-clock times of tests done directly with OmniRPC on platforms at Lille (in minutes)

m, k	29 workers at Lille	58 workers at Lille
15, 1	11	9
15, 2	9	6
15, 3	9	6
20, 1	10	7
20, 2	10	6
20, 3	7	6
25, 1	8	6
25, 2	8	7
25, 3	5	4

Table 6.4: Overhead of YML (used with the OmniRPC back-end) compared to a direct usage of OmniRPC (ratio)

6.2.3.3 Results of experiments

The wall-clock times to solve the eigenproblem are between 4 and 11 times longer than the times picked up with a direct execution on OmniRPC. The platform configurations, the input parameters and the numerical algorithms are similar. We point out two causes explaining such differences of computing times.

First, we point out the overhead of YML. There are many reasons. The scheduler of YML takes time to select the remaining tasks to run. It is also quite long to pack and unpack the parameters in order to transfer them between the data repository and the YML Worker. Besides, for each component call assigned to a given node, we must transfer the binary of the component even if the same component has already been run by the same node. We suppose this communication cost for the binary transfer is the main cause of overhead. With YML and the OmniRPC back-end, the remote stub wraps the YML Worker and this YML Worker must download the binary. On the contrary, a direct execution with OmniRPC only transfers the parameters and performs procedure calls on the remote stubs. Those stubs are already registered on the remote nodes.

Second, it concerns the implementation point of view. With an usual OmniRPC program, the client can perform simple computations instead of doing remote procedure calls whose communication times would be long compared to the remote computation times. On the contrary, YvetteML does not support any computation. The client must do many component calls even for very simple tasks. It adds much communication. Besides, we have seen that the volume of communication is significant because YML transfers the executable for each component call.

6.3 Ongoing work on YML

We consider two kinds of ongoing work related to the experiments and the development of YML.

First, we have to experiment the scalability of YML by means of tests using much larger matrices and harnessing more workers. A Grid5000 environment has been set up by Olivier Delannoy in order to perform this work. It would be also interesting to test the second YvetteML workflow, shown by Figure 6.4. We would like to experiment some modifications of the workflow by distributing some computations. The objective would be to confirm that communication really penalizes the distribution of certain tasks such as the partitioning of A, the reorthogonalization. We have also to experiment our eigensolver with the XtremWeb back-end.

Second, we have to finish ongoing implementations on YML related to the multi back-end support. It concerns the Back-end Scheduler module and the information service module.

The information service consists in storing and managing static and dynamic information about the network resources, the computing nodes and their middleware, and also some statistics of the past executions. It is used by the Back-end Scheduler in order to schedule at the runtime a task to the most appropriate back-end. There are many other ongoing work on YML which are handled only by Olivier Delannoy, its main developer. For instance, he plans to optimize the compilation process and returns a more friendly time evaluation. He also wants to make a distributed version of the data repository service and a binary cache on the workers in order to reduce the cost of communications. Indeed, even if a worker handles several times the same component, the same binary is downloaded by this worker at each time. Thus, by temporary caching component binaries, we can reduce significantly the volume of transferred data and we can limit the risk of bottleneck of the DR Server.

6.4 Conclusion

The YML framework targets to hide the complexity and heterogeneity of the Global Computing middleware so that the client only focuses on parallel and distributed considerations for his application. YML provides a user-friendly workflow language called YvetteML and uses the component programming model so as to favour the re-usability and to clearly separate tasks and communication.

Although we appreciate harnessing any kind of computing middleware with a unique YvetteML program, the current status of YML does not hide entirely the heterogeneity of the middleware level. In fact, at the runtime, we can only use one back-end since the multi back-ends support (at the runtime) is still an ongoing work. Our analysis of the eigenproblem has underlined that we can easily discuss about the different possible distributed schemes without any (or very few) middleware considerations. Then, we apply easily our choices with the workflow language of YML. Of course, YvetteML has its own limitations but they are not problematic as soon as we have understood some details of the YvetteML API. In particular, we can take a great advantage by using the YML Collection API. It allows managing data almost as well as lower level programming languages. Besides, with the YML Collection API, the client can easily do out-of-core without worrying on file management.

Our experiments have underlined a significant overhead. We must tackle it quickly if we want to perform larger computations and attract users. In fact, although we do not target very high performances, it is quite long to solve a small real eigenproblem ($N = 47792$) which is not using the Internet network. By using Internet, it is even supposed to be longer. As explained in the ongoing work section, there are many possibilities to improve the effectiveness of YML. It will be very interesting to perform the same experiments with a cache mechanism for the binaries and a more efficient packing and unpacking process.

YML is proposing a novel and very high level approach of Global Computing. As YML is young software developed by a very small team, it remains many things to improve but we are convinced to go on the right way.

Chapter 7

Evolution of large scale computing

7.1 Systems for large scale computing

The Global Computing model is relevant for the solution of large applications only if the end-user cannot access to HPC devices and if he does not take care to the time-to-solution. The cost of communication for the Global Computing model is the main explanation of poor performances. It makes applications giving very long time-to-solutions. This last point eliminates many existing computational problems such as weather simulations, most of industrial and real-time applications. Besides, industrial companies may also worry about the integrity and the confidentiality of the results. Finally, the computational nodes of Global Computing platforms have limited resources. So, for solving disk- and memory-intensive applications, we must divide a lot the workload among many nodes. It generates a fine grain of parallelism which is not suitable for this kind of large and heterogeneous network.

Therefore, High Performance Computing systems are indispensable for most of computational applications. They usually provide large amounts of memory and disk space, large bandwidth and low latency networks. They also guaranty confidentiality and integrity requirements. As they are dedicated to computations (with reservation policies), the experiments are not disturbed by other computations and communication. The HPC community generally consider three kinds of HPC systems. First, there are the commodity systems. There are made of commodity COTS (commercial off-the-shelf) microprocessors and COTS interconnections. Second, we have hybrid machines composed of COTS microprocessors, or customized COTS microprocessors, and custom interconnections (e.g. TSUBAME at TITech, IBM Blue Gene/Light, ASCI White, T2K Open SuperComputer at the Universities of Tokyo, Tsukuba and Kyoto). Third, we consider the custom systems which are entirely made of custom microprocessors and custom interconnections (e.g. Earth Simulator, Cray X1 supercomputer).

7.2 Current trend of HPC

Custom HPC systems are entirely developed for the sake of computational applications. Thus, they are merely the most suitable tools. However, they have a high $/Flops ratio because of high costs of designing and manufacturing, power consumption, etc. A motivation for the development of hybrid machines was to decrease the costs. For instance by using COTS microprocessors, we get economies of scale on their design and manufacture. This trend has carried on by a decline of hybrid machines and an increase of commodity systems. We underline several reasons for the success of commodity systems. First, they have a lower $/Flops ratio. Then, by buying a commodity system, we can enter the HPC community with reasonable expenses. With commodity systems, we do not only save money at the hardware level, we can also save money in order to get software. In fact, commodity and custom systems differ by their software approach. The former adopts an horizontal approach by using much independent software. They are generally open source software. Custom systems often have a vertical approach. The constructor of the supercomputer develops a fully integrated software environment and sells it with its machines. Hybrid systems seem to adopt an horizontal approach. In order to improve the performance, the reliability and the support of software, they can choose products of private companies (instead of open source). Last reason for the success of commodity systems but not least, they can be extended easily and with reasonable costs.

Nevertheless, commodity systems have serious drawbacks. By using commodity networks, we face a high latency and a low bandwidth. Moreover, the failure rate is larger than for custom systems. Besides, the natural trend is the increase of the number of processors. Thus, the probability of failure increases and the parallel efficiency of the computations decreases. In fact, in the time-to-solution, the part of the remote computing times is decreasing while the part of the communication times is increasing. Then, the commodity systems do not deal well with most of applications, such as sparse linear algebra, which have a low spatial and temporal locality of data. In fact, a commodity system has generally a low memory bandwidth and high memory latency. Commodity systems are not suitable for many other kinds of applications such as data flow applications (data parallelism) and problems having strong time-to-solution requirements (mainly because of the network layer). We do not expect that current constructors of commodity processors will perform the necessary research in order to reduce significantly the gap between CPU speed and memory bandwidth (and memory latency). In fact, since a decade, constructors have used the CPU clock rate as a commercial argument and this frequency has greatly risen without a scaling of the other components. Besides, the current market of individual PCs and laptops does not ask for making up this gap: the recent development of multi-core technologies seems to satisfy the needs of consumers for innovation.

The problem of memory (and network) low bandwidth and high latency makes most of applications more scalable on hybrid and custom systems than on commodity systems. Therefore, in spite of the increase of the number of commodity clusters in the TOP500 ranking, we expect the highest performances will be reached always by hybrid and custom

(vectorial) systems. If we review current projects of supercomputers, we notice a significant part of hybrid systems. For instance, several projects study how to integrate into supercomputers the specific IBM Cell chip or GPU. Some projects of hybrid systems, such as the Roadrunner project at the LANL and the Keisoku project of the Riken, target the Petaflops threshold. Actually, the Riken's MDGrape-3 has already reached and overcome this threshold but this supercomputer is so specific to one application that it cannot give good results for the Linpack benchmark of the TOP500 ranking. This point underlines the need for a new ranking process by using a tool such as the "HPC Challenge benchmark suite" for future High Performance Computing systems.

7.3 Current issues of HPC and comparison with Global Computing

7.3.1 Bandwidth and latency of the memory

We have previously said that arithmetic performance of the processor is increasing faster than the increase of local bandwidth and faster than the decrease of the local latency. This phenomenon is more striking for commodity processors than custom vectorial processors. Vectorial processors are less affected because the CPU clock rate is often lower, the memory bandwidth higher and the memory latency lower. Besides, the vectorial computation model naturally hides the memory latency.

High Performance Computing has strong performance requirements such as the time-to-solution. Therefore, many mechanisms intend to hide the memory latency and its low bandwidth in order to maintain a constant maximal usage of the processor. Some examples are given in next paragraph. On the contrary, Global Computing has lower performance requirements because it suffers from very slow network communication. It suffers also from the heterogeneity of the networks and the heterogeneity of the computing nodes. Global Computing systems are weakly-coupled and we must do a coarse grain parallelism. Optimizations at the CPU cache/memory level are difficult to perform on the remote volunteer computing nodes (e.g. we may not have the right). Besides, even if it was possible to do such optimizations, it would probably give no benefit. In fact, those volunteer computing nodes are potentially shared with other users whose processes would annihilate any optimization efforts.

In this paragraph, we gather some mechanisms used by hybrid and commodity systems in order to sustain a peak usage of the processor. We first look at solutions dealing with the low memory bandwidth. An idea is to reduce the memory accesses by increasing the number of cache levels and the size of caches. This solution is suitable for applications with high spatial and temporal localities of data. If the application has not such characteristics (irregular data accesses, little data reuse), the solution is not efficient. Another

solution would be to increase the number of memory chips but it would be very expensive. Some researchers have proposed using sectored cache lines in order to do sector-cache blocking. It would avoid removing potential useful data (i.e. soon reused) while getting new data in cache from the memory. Second, we focus on memory latency hiding mechanisms. Constructors have proposed using multi-threading, cache pre-fetching, and branch prediction. We have also noticed processor-in-memory architectures and a new trend of stream-processing architectures.

7.3.2 Bandwidth and latency of the network

HPC commodity systems are using COTS network components. Therefore, they suffer from much lower network bandwidth and much higher network latency than custom and hybrid systems. When a commodity system has a distributed memory model, it uses a communication library such as MPI. This kind of tool adds some overhead that we must take into account when we target the highest performances. If the system has a distributed shared memory model, it is possible to make direct "processor-remote memory" accesses but it generates significant communication costs.

In Global Computing, the impact of network latency and network bandwidth is considerable. The usage of Internet or slow Ethernet networks is a first explanation. We can also point out the congestion due to communication generated by other users. Much Global Computing software uses the TCP protocol over the Internet and then, performances are burdened by the TCP congestion control. In Chapter 4, we presented some solutions in order to limit the impact of communication. For instance, we have underlined the interest of the parametric parallel paradigm (i.e. task-farming). We can find some analogies between solutions dealing with memory bandwidth (and latency) for HPC and solutions dealing with communication on the Internet for Global Computing. For instance, on the one hand, a cache-blocking technique makes the reuse of data faster by decreasing the number of transfers from the memory. On the other hand, data persistence also makes the reuse of data faster by decreasing the size (and the number, depending on the software) of communication over the Internet. Another example concerns data pre-fetching and stream processing in HPC. The first mechanism pre-loads data in cache from the memory and the second handles a continuous flow of data. The translation of those mechanisms to world-wide data-flow applications, such as video streaming, leads to the usage of the RTP protocol and buffers for received data (the Real-time Transfert Protocol adds real-time constraints).

7.3.3 Disk storage issue

High Performance Computing systems seldom lack storage space because the computing centers take care to buy enough storage devices and supplement them if necessary. By

storage devices, we denote a wide range of items. It can be hard disk of commodity PCs with high transfer rates, dedicated Mass Data Storage Systems, etc. On the contrary, with Global Computing, the remote computing nodes have limited resources. In Section 4.3, we face disk space limitations. We think a solution is to use a complementary distributed storage layer. For instance, if the Grid Computing middleware is OmniRPC, we can use OmniStorage associated with GFarm or BitTorrent. Of course, it adds some overhead but we tolerate it in the context of Global Computing as long as we can solve much larger problems.

7.3.4 Programming models and software issues

Building High Performance Computing and Global Computing systems, is only worth if we study how to adapt efficiently applications to the targeted architectures and if there are efficient software products to develop and deploy applications. Programming models, languages and software of high performance custom systems have reached a certain maturity. Besides new software solutions are generally efficient and have a good support since constructors of custom systems often propose a vertical collection of software (previously explained in Section 7.2). For high performance hybrid and commodity systems, a significant work must be done as well on programming paradigms as on operating systems, libraries and languages. The more the systems are distributed, heterogeneous and weakly-coupled, the more we need to do such studies. Therefore, this need is much more important for Global Computing. We also estimate this need is increased by the youth of this domain and also because industrial companies have not yet really focused on Global Computing.

As the size and the heterogeneity of platforms is increasing, it is more and more difficult to express and perform an efficient distribution of an application. Thus, we need high level programming tools and operating systems able to manage such environments. The development of such software and the study of programming models is greatly challenging because of the very large number of processors (> 100000). Besides, we have to deal with several kinds of parallelism: vectorial parallelism on one CPU, thread parallelism across one CPU (context switching) or within an SMP node, inter-nodes parallelism. A computing system may also have several communication models depending on the memory management: shared memory, distributed memory and shared distributed memory. For Global Computing, software must hide as much as possible the computing and network resources. The YML framework is a good example. The NAREGI project proposes a middleware harnessing many High Performance Computing Centers spread all over Japan. High-level operating systems and programming tools generally have some overhead. We can tolerate it for Global Computing (if it is not too large), but we hardly accept such overhead for HPC. We must find a balanced trade-off between the targeted effectiveness and the programming abstraction level. Besides, in order to reach the highest performance, the end-user must keep control of the process concurrency, he must ensure an efficient data distribution, he has to optimize communication, etc. Performance hardly

fits well with abstraction.

It becomes more and more difficult and expensive to improve the floating point computations performance of one CPU because we have, or will soon, reached a maximum threshold of ILP (Instruction-Level Parallelism).Thus, industrials of commodity processors propose multi-core processors. We now find a growing number of commodity and hybrid systems, as well as Global Computing platforms, composed of computers having multi-core commodity processors. It becomes an urgent issue to study programming models for platforms using multi-core systems in order to exploit all the levels of parallelism. In particular, we add an inter-cores parallelism with shared-memory communication by using the shared L2-cache and the direct cores synchronizations. Actually, the current multi-core approach does not seem an ideal solution because we have previously said that CPUs (and the current cores) are not adapted to most of computational applications which are potentially used by HPC and Global Computing. Those CPU are not also suitable for data flow applications such as most of multimedia applications. Thus, some researchers and industrials are studying the next generation of multi-core CPU with heterogeneous cores. As an example of the challenge of multi-core programming, we can cite that Google has recently acquired the software company PeakStream, which specializes in software to assist developers in programming for multi-core processors. In particular, this start-up was focusing on the optimization of parallel processes for data flow applications.

Finally, the HPC community has been greatly interested in the IBM Cell architecture and its single precision performance. Unfortunately, the IBM Cell has average double precision performance. In the frame of the Roadrunner project, IBM will deliver soon enhanced double-precision Cell chips. Hybrid systems composed of IBM cell have already been built and it is now necessary to learn how to adapt efficiently scientific application to this architecture such as in [115].

In this Chapter, we have mostly used information from [116] and several issues of the "SciDAC Review" magazine. I have also used talks of the 21^{th} ORAP[1] Forum and a status report [117] of HPC in Japan done in Juin 2007 by French researchers and the French Ambassy at Tokyo.

[1] http://www.irisa.fr/orap/

Acronym	Translation
AIMS	Automated Instrumentation and Monitoring System
API	Application Program Interface
AppLeS	ApplicationLevel Scheduling
APST	AppLeS Parameter Sweep Template
ATM	Asynchronous Transfer Mode
BLACS	Basic Linear Algebra Communication Subprograms
BLAS	Basic Linear Algebra Subprograms
BPEL4WS	Business Process Execution Language for Web Services
CASH	Computer Aided SCHeduling
CCM	Corba Component Model
COTS	Commercial Off-The-Shelf
DAG	Directed Acyclic Graph
DCE	Distributed Computing Environment
DQS	Distributed Queuing System
FhRG	Fraunhofer Resource Grid
GASS	Globus Global Access to the Secondrary Storage
GEM	Globus Executable Management
GIOP	General Inter-ORB Protocol
GIS	Globus Grid Information Service
GPDK	Grid Development Portal Toolkit
GPU	Graphics Processing Unit
GrADS	Grid Application Development Software
GRAM	Globus Resource Allocation Manager
GRIP	Grid Interoperability Project
GSFL	Grid Services Flow Language
GSI	Globus Grid Security Infrastructure
GUI	Graphical User Interface
HBM	Globus Heartbeat Monitor
HeNCE	Heterogeneous Network Computing Environment
HPC	High Performance Computing
HPF	High Performance Fortran
IDL	Interface Description Language
IIOP	Internet Inter-ORB Protocol
ILP	Instruction-Level Parallelism
IP	Internet Protocol
ITG	Interactive Task Graph
JADE	Java Agent Development Environment
JVM	Java Virtual Machine
LAN	Local Area Network
LaRCS	Language for Regular Communication Structures
LSF	Load Sharing Facility
MIMD	Multiple Instructions Multiple Data
MPI	Message Passing Interface

Continued on next page

Table 7.1 – continued from previous page

Acronym	Translation
MPMD	Multiple Program Multiple Data
MW	Master-Worker
NQE	Network Queueing Environment
NWS	Network Weather Service
OGSA	Open Grid Services Architecture
OMG	Object Management Group
ORB	Object Request Broker
PBS	Portable Batch System
P-GRADE	Parallel Grid Run-time and Application Development Environment
POOMA	Parallel ObjectOriented Methods and Applications
PVM	Parallel Virtual Machine
RCDS	Resource Cataloging and Distribution System
RIO	Globus Remote Input/Output
RMI	Remote Method Invocation
RPC	Remote Procedure Call
RTP	Real-time Transfert Protocol
SCIrun	Scientific Computing and Imaging Research
SPMD	Simple Program Multiple Data
SSL	Secure Socket Layer
TCG	Temporal Communication Graph
TCL	Tool Command Language
TCP	Transfert Control Protocol
TME	Task Mapping Editor
UDP	User Datagram Protocol
UNICORE	Uniform Interface to Computing Resources
VAMPIR	Visualization and Analysis of MPI Resources
VT	Visualization Tool
WSCI	Web Services Choreography Interface
WSDL	Web Services Description Language
WSFL	Web Services Flow Language
XML	eXtensible Markup Language

Abstract

A significant work has been done on Global Computing middleware and networking but it remains a lot to do at the application level. Thus, we study the parallelization and the distribution of the large real symmetric eigenproblem on Global Computing platforms. Our methodology stresses how important the choice of the numerical methods is, in order to use optimal parallel and distributed paradigms. We oppose the parametric parallelism, allowed by the Bisection method, to the highly communicating and synchronized restarted Lanczos method. Then, we discuss the impact of those choices on the data distribution and the data access schemes. Global Computing is characterized by the heterogeneity and the poor performances of the computing and network resources. Therefore, we propose some solutions such as data-persistence and out-of-core. All propositions are implemented and deployed on several world-wide platforms using the Internet and on Grid5000. Thus, this work shows the viability of Global Computing for the resolution of linear algebra problems as long as we do not target the highest performances.

We contribute to the development of the YML framework. This software hides the complexity and the heterogeneity of Global Computing middleware. In particular, the front-end of YML provides a unique API called YvetteML which is a high-level workflow programming language, and it gives a transparent access to several Global Computing middleware by means of a back-ends mechanism. In addition to the development of modules, such as a back-end for the OmniRPC middleware, we discuss how to adapt the real symmetric eigenproblem to YML and we build two workflows in order to orchestrate the computational sub-tasks of the eigenproblem. An evaluation of the overhead of YML stresses some necessary improvements such as a cache mechanism on the remote computing nodes.

Research on low-power devices and power-aware mechanisms is a tremendous topic of research in High-Performance Computing. How about carrying on similar researches for Global Computing? Therefore, we study how to get a power-aware real symmetric eigensolver. We show that significant global and local energy savings can be achieved by exploiting the heterogeneity of the computing nodes.

keywords: real symmetric eigenproblem, Bisection, Lanczos, parametric parallelism, OmniRPC, XtremWeb, YML, Global Computing, Grid, Grid5000, Power-Aware Computing, DVS

Résumé

Le calcul global est un domaine de recherche vaste, dynamique mais qui bénéficie d'un effort de recherche inégal selon la spécialité. Un travail important reste à faire au niveau applicatif. Nous présentons une méthodologie pour la parallélisation et la distribution d'une méthode d'algèbre linéaire pour la recherche des éléments propres d'une matrice réelle symmétrique. Nous discutons de l'impact des choix algorithmiques sur les paradigmes de parallélisme et la répartition des données. En particulier, nous opposons le parallélisme paramétrique à un parallélisme fortement communicant et synchronisé dans un contexte défavorable de plateformes réparties à l'échelle de l'Internet, puis sur Grille'5000. Nous proposons des mécanismes indispensables pour le déploiement d'applications à grande échelle sur des ressources hétérogènes non dédiées tels la persistance des données, la programmation "out-of-core" et un algorithme numérique redémarré.

Nous contribuons, en parallèle, au développement du logiciel YML qui masque la complexité et l'hétérogénéité des logiciels de calcul global afin que les scientifiques ne se soucient que des détails applicatifs de leurs problèmes. A la mise-en-œuvre de modules logiciels d'YML tel un "back-end" pour le logiciel de calcul sur grille OmniRPC, nous ajoutons une réflexion sur la programmation haut-niveau d'applications numériques par orchestration de composants au moyen du langage de "workflow" YvetteML. Dans la continuité du travail précédent, notre étude de cas est la recherche des éléments propres d'une matrice rélle symétrique.

Enfin, nous explorons un domaine de recherche naissant mais prometteur: le calcul sur grille à faible consommation d'énergie. Dans un contexte de grille de calcul très hétérogène et pour des applications communicantes et synchronisées telle notre méthode de recherches des éléments propres d'une matrice, nous montrons comment réaliser des économies d'énergies significatives en faisant varier la fréquence des processeurs et sans affecter de façon importante les temps de calculs.

Mots clefs: valeurs et vecteurs propres, Bisection, Lanczos, parallélisme paramétrique, OmniRPC, XtremWeb, YML, calcul global, grille de calcul, Grille'5000, calcul à faible consommation énergétique, DVS

Bibliography

[1] Rolf Rabenseifner and Armin Schuch. *Comparison of DCE RPC, DFN-RPC, ONS and PVM.* In DCE Workshop. Pages 39–46. 1993.

[2] Rolf Rabenseifner. *The DFN remote procedure call tool for parallel and distributed applications.* In Kommunikation in Verteilten Systemen. Pages 415–429. 1995.

[3] David B. Johnson and Willy Zwaenepoel. *The peregrine high-performance RPC system.* In Software Practice and Experience. Volume 23. Number 2. Pages 201–221. 1993.

[4] Chi-Chao Chang, Grzegorz Czajkowski, and Thorsten Von Eicken. *MRPC : A high performance RPC system for MPMD parallel computing.* In Software Practice and Experience. Volume 29. Numnber 1. Pages 43–66. 1999.

[5] Rob V. van Nieuwpoort, Jason Maassen, Rutger Hofman, Thilo Kielmann, and Henri E. Bal. *Ibis : an efficient java-based grid programming environment.* In Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande. Pages 18–27. ACM Press. 2002.

[6] Jason Maassen, Rob Van Nieuwpoort, Ronald Veldema, Henri E. Bal, Thilo Kielmann, Ceriel J. H. Jacobs, and Rutger F. H. Hofman. *Efficient java RMI for parallel programming.* In Programming Languages and Systems. Volume 23. Number 6. Pages 747–775. 2001.

[7] Jason Maassen, Thilo Kielmann, and Henri E. Bal. *Gmi : Flexible and efficient group method invocation for parallel programming.* 6^{th} Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers. 2002.

[8] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, and Raoul A. F. Bhoedjang. *MagPIe : MPI's collective communication operations for clustered wide area systems.* ACM SIGPLAN Notices. Volume 34. Number 8. Pages 131–140. 1999.

[9] Henri E. Bal, Aske Plaat, Thilo Kielmann, Jason Maassen, Rob van Nieuwpoort, and Ronald Veldema. *Parallel computing on wide-area clusters : the albatross project.* In proceedings of Extreme Linux Workshop. Pages 20-24. Monterey, California. 1999.

[10] Edgar Gabriel, Michael Resch, Thomas Beisel, and Rainer Keller. *Distributed computing in a heterogeneous computing environment.* In Proceedings of the 5th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Pages 180–187. Springer-Verlag. 1998.

[11] Vaidy S. Sunderam. *PVM : a framework for parallel distributed computing.* In Concurrency, Practice and Experience Volume 2. Number 4. Pages 315–340. 1990.

[12] Paul A. Gray and Vaidy S. Sunderam. *Metacomputing with the IceT system.* The International Journal of High Performance Computing Applications Volume 13. Number 3. Pages 241–252. 1999.

[13] Mitsuhisa Sato and Hidemoto Nakada and Satoshi Sekiguchi and Satoshi Matsuoka and Umpei Nagashima and Hiromitsu Takagi. *Ninf: A Network Based Information Library for Global World-Wide Computing Infrastructure.* HPCN Europe '97: Proceedings of the International Conference and Exhibition on High-Performance Computing and Networking. Pages 491–502. Springer-Verlag. 1997.

[14] Hidemoto Nakada and Mitsuhisa Sato and Satoshi Sekiguchi. *Design and implementations of Ninf: towards a global computing infrastructure.* In Future Generation Computer Systems. Volume 15. Number 5-6. Pages 649–658. Elsevier Science Publishers. 1999.

[15] Mitsuhisa Sato and Taisuke Boku and Daisuke Takahashi. *OmniRPC: a Grid RPC system for Parallel Programming in Cluster and Grid Environment.* Proceedings of the 3^{th} International Symposium on Cluster Computing and the Grid (CCGRID'03). Page 206. IEEE Computer Society. 2003.

[16] Cécile Germain and Vincent Néri and Gilles Fedak and Franck Cappello. *XtremWeb: Building an Experimental Platform for Global Computing.* Proceedings of the First IEEE/ACM International Workshop on Grid Computing (GRID'00). Pages 91–101. Springer-Verlag. 2000.

[17] Gilles Fedak and Cécile Germain and Vincent Néri and Franck Cappello. *XtremWeb: A Generic Global Computing System.* Proceedings of the 1^{st} International Symposium on Cluster Computing and the Grid (CCGRID'01). Page 582. IEEE Computer Society. 2001.

[18] Marteen van Steen and Philip Homburg and Andrew S. Tanenbaum. *The Architectural Design of Globe: A Wide-Area Distributed System.* Techreport, IR-422. Netherlands. 1997.

[19] Ian Foster and Carl Kesselman. *Globus: A Metacomputing Infrastructure Toolkit.* The International Journal of Supercomputer Applications and High Performance Computing,. Volume 11. Number 2. Pages 115–128. 1997.

[20] Richard Wolski. *Forecasting network performance to support dynamic scheduling using the network weather service.* The 6^{th} International Symposium on High Performance Distributed Computing (HPDC'97). Pages 316–325. IEEE Conputer Society. Portland, OR, USA. August 1997.

[21] Rich Wolski, Neil T. Spring, and Jim Hayes. *The network weather service : a distributed resource performance forecasting service for metacomputing.* Future Generation Computer Systems. Volume 15. Number 5–6. Pages 757–768. 1999.

[22] Steve J. Chapin, Dimitrios Katramatos, John Karpovich, and Andrew S. Grimshaw. *The Legion resource management system.* In Job Scheduling Strategies for Parallel Processing. Pages 162–178. Dror G. Feitelson and Larry Rudolph editors. Springer Verlag. 1999.

[23] Graham E. Fagg, Keith Moore, and Jack Dongarra. *Scalable Networked Information Processing Environment (SNIPE).* In Future Generation Computer Systems. Volume 15. Number 5–6. Pages 595–605. 1999.

[24] Henri Casanova and Graziano Obertelli and Francine Berman and Rich Wolski. *The AppLeS parameter sweep template: user-level middleware for the grid.* Proceedings of the 2000 ACM/IEEE conference on Supercomputing (SC'00). Page 60. IEEE Computer Society. Dallas, Texas, USA. 2000.

[25] David Abramson and Rok Sosic and Jon Giddy and B. Hall. *Nimrod: A Tool for Performing Parameterised Simulations Using Distributed Workstations.* The 4th International Symposium on High Performance Distributed Computing (HPDC'95). Pages 112–121. Washington DC, USA. 1995.

[26] Steven G. Parker and Christopher R. Johnson. *Scirun : a scientific programming environment for computational steering.* In Proceedings of the 1995 ACM/IEEE conference on Supercomputing (SC'95). Page 52. 1995.

[27] *Barbara M. Chapman, Babu Sundaram, and Kiran K. Thyagaraja.* Ez-grid : Integrated resource brokerage services for computational grids.

[28] Jon B. Weissman. *Gallop : The benefits of wide-area computing for parallel processing.* Journal of Parallel and Distributed Computing. Volume 54. Number 2. Pages 183–205. 1998.

[29] Junwei Cao, Stephen A. Jarvis, Subhash Saini, Darren J. Kerbyson, Graham R. Nudd. *ARMS: an agent-based resource management system for grid computing.* Scientitic Programming (Special Issue on Grid Computing). Volume 10. Number 2. Pages 135-148. 2002.

[30] Henri Casanova and Jack Dongarra. *NetSolve: a network server for solving computational science problems.* Proceedings of the 1996 ACM/IEEE conference on Supercomputing (SC'96). Page 40. Pittsburgh, PA, USA. 1996.

[31] Henri Casanova and Jack Dongarra. *NetSolve: A Network-Enabled Server for Solving Computational Science Problems.* The International Journal of Supercomputer Applications and High Performance Computing. Volume 11. Number 3. Pages 212–223. 1997.

[32] Keith Seymour and Hidemoto Nakada and Satoshi Matsuoka and Jack Dongarra and Craig Lee and Henri Casanova. *Overview of GridRPC: A Remote Procedure Call API for Grid Computing.* Proceedings of the Third International Workshop on Grid Computing (GRID'02). Pages 274–278. Springer-Verlag. 2002.

[33] Arash Baratloo and Mehmet Karaul and Zvi M. Kedem and Peter Wyckoff. *Charlotte : Metacomputing on the web.* In Proceedings of the 9^{th} International Conference on Parallel and Distributed Computing Systems (PDCS-96). 1996.

[34] Francine Berman and Andrew Chien and Keith Cooper and Jack Dongarra and Ian Foster and Dennis Gannon and Lennart Johnsson and Ken Kennedy and Carl Kesselman and John Mellor-Crummey and Dan Reed and Linda Torczon and and Rich Wolski. *The GrADS Project : Software support for high-level Grid application development.* The International Journal of High Performance Computing Applications. Volume 15. Number 4. Page 327–344. 2001.

[35] Mathilde Romberg. *The UNICORE Architecture: Seamless Access to Distributed Resources.* Proceedings of the 8^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'99). Page 44. IEEE Computer Society. Washington, DC, USA. 1999,

[36] Jean-Pierre Goux and Sanjeev Kulkarni and Jeff Linderoth and Michael Yoder. *An enabling framework for master-worker applications on the computational grid.* 9^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'00). Pages 43–50. IEEE Conputer Society. Pittsburgh, Pennsylvania, USA. August 2000.

[37] Rich Wolski and John Brevik and Chandra Krintz and Graziano Obertelli and Neil Spring and Alan Su. *Running EveryWare on the computational grid.* Proceedings of the 1999 ACM/IEEE conference on Supercomputing (SC'99). Portland, Oregon, United States. 1999.

[38] Gabrielle Allen and Werner Benger and Tom Goodale and Hans-Christian Hege and Gerd Lanfermann and Andre Merzky and Thomas Radke and Edward Seidel and John Shalf. *The cactus code : A problem solving environment for the grid.* 9^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'00). Pages 253. IEEE Conputer Society. August 2000. Pittsburgh, Pennsylvania, USA.

[39] Geoffrey Fox and Tomasz Haupt and Erol Akarsu and Alexey Kalinichenko and KangSeok Kim and Praveen Sheethalnath and Choon-Han Youn. *The gateway system : Uniform web based access to remote resources.* Proceedings of the ACM 1999 Conference on Java Grande (JAVA'99). Pages 1–7. San Francisco, CA, USA. June 1999.

[40] David P. Anderson. *BOINC: A System for Public-Resource Computing and Storage.* Proceedings of the 5^{th} IEEE/ACM International Workshop on Grid Computing (GRID'04). Pages 4–10. Pittsburgh, USA. November 2004.

[41] Bernd O. Christiansen and Peter Cappello and Mihai F. and Ionescu and Michael O. Neary and Klaus E. Schauser and Daniel Wu. *Javelin: Internet-based parallel computing using Java.* In Concurrency: Practice and Experience. Volume 9. Number 11. Pages 1139–1160. 1997.

[42] Michael O. Neary and Sean P. Brydon and Paul Kmiec and Sami Rollins and Peter Cappello. *Javelin++: scalability issues in global computing.* JAVA '99: Proceedings of the ACM 1999 conference on Java Grande. Pages 171–180. ACM Press. 1999.

[43] Andrew Chien. *Architecture of the Entropia Distributed Computing System.* Proceedings of the 16th International Symposium on Parallel and Distributed Processing (IPDPS'02). IEEE Computer Society. 2002.

[44] Luis F. G. Sarmenta. *Bayanihan: Web-Based Volunteer Computing Using Java.* Proceedings of the Second International Conference on Worldwide Computing and Its Applications (WWCA'98). Pages 444–461. Springer-Verlag. 1998.

[45] Robert D. Blumofe and Christopher F. Joerg and Bradley C. Kuszmaul and Charles E. Leiserson and Keith H. Randall and Yuli Zhou. *Cilk: an efficient multithreaded runtime system.* Proceedings of the fifth ACM SIGPLAN symposium on Principles and practice of parallel programming (PPOPP'95). Pages 207–216. Santa Barbara, California, United States. 1995.

[46] Eric J. Baldeschwieler and Robert D. Blumofe and Eris A. Brewer. *ATLAS: An Infrastructure for Global Computing.* Proceedings of the Seventh ACM SIGOPS European Workshop on System Support for Worldwide Applications 1996.

[47] Noam Nisan and Shmulik London and Ori Regev and Noam Camiel. *Globally Distributed Computation over the Internet - The POPCORN Project.* Proceedings of the 18^{th} International Conference on Distributed Computing Systems (ICDCS'98). Pages 592. IEEE Computer Society. 1998.

[48] Olivier Delannoy and Nahid Emad and Serge Petiton. *Workflow Global Computing with YML.* The 7^{th} IEEE/ACM International Conference on Grid Computing (GRID 2006). IEEE Computer Society. Barcelona. September 2006.

[49] Franck Cappello and Eddy Caron and Michel Dayde and Frederic Desprez and Yvon Jegou and Pascale Vicat-Blanc Primet and Emmanuel Jeannot and Stephane Lanteri and Julien Leduc and Nouredine Melab and Guillaume Mornet and Benjamin Quetier and Olivier Richard. *Grid'5000: A Large Scale and Highly Reconfigurable Grid Experimental Testbed* SC'05: Proc. The 6^{th} IEEE/ACM International Workshop on Grid Computing. Pages 99–106. IEEE/ACM. Seattle, Washington, USA. November 2005.

[50] James Demmel and Inderjit Dhillon and Huan Ren. *On the correctness of parallel bisection in floating point.* Report. UCB/CSD 94/805. Computer Science Division (EECS), University of California. Berkeley, CA, USA. March 1994.

[51] Silvia A. Crivelli and Elizabeth R. Jessup. *Toward an Efficient Parallel Implementation of the Bisection Method for Computing Eigenvalues.* Proceedings the 6^{th} Distributed Memory Computing Conference. Pages 443–446. IEEE Computer Society. Portland, OR, USA. April 1991.

[52] Sy-Shin Lo and Bernard Philippe and Ahmed Sameh. *A Multiprocessor Algorithm for the Symmetric Tridiagonal Eigenvalue Problem.* SIAM Journal on Scientific and Statistical Computing. Volume 8. Number 2. Pages 155–165. 1987.

[53] Jeff S. Reeve and M. Heath. *An efficient parallel version of the householder-QL matrix diagonalisation algorithm.* Parallel Computing. Volume 25. Number 3. Pages 311–319. Elsevier Science Publishers B. V. 1999.

[54] Christopher Smith and Bruce Hendrickson and Elizabeth Jessup. *A parallel Algorithm for Householder Tridiagonalization.* Proceedings of the 5^{th} SIAM Conference Applied Linear Algebra. Pages 361–365. 1994.

[55] Juan Touriño and Ramon Doallo and Emilio Zapata. *Sparse Givens QR Factorization on a Multiprocessor.* 2^{nd} International Conference on Massively Parallel Computing Systems. Ischia, Italy. 1996.

[56] Omer Egecioglu and Ashok Srinivasan. *Givens and Householder Reductions for Linear Least Squares on a Cluster of Workstations.* Technical Report TRCS95-10. University of California at Santa Barbara. 1995.

[57] Kesheng Wu and Horst D. Simon. *A parallel Lanczos method for symmetric generalized eigenvalue problems.* Technical Report 41284, Lawrence Berkeley National. 1997.

[58] Françoise Tisseur and Jack Dongarra, *Parallelizing the Divide and Conquer Algorithm for the Symmetric Tridiagonal Eigenvalue Problem on Distributed Memory Architectures.* SIAM Journal on Scientific Computing. Volume 20. Issue 6. Pages 2223 - 2236. 1999.

[59] Nahid Emad and Serge G. Petiton and Guy Edjlali. *Multiple Explicitely Restarted Arnoldi Method for solving large eigenproblems.* SIAM Journal on Scientific Computing. Volume 27. Number 1. Pages 253–277. 2005.

[60] Nahid Emad and Sayed Abolfazl Shahzadeh-Fazeli and Jack Dongarra. *An asynchronous algorithm on the NetSolve global computing system.* In Future Generation Computer Systems. Volume 22. Number 3. Pages 279–290. 2006.

[61] Lamine M. Aouad and Serge G. Petiton and Mitsuhisa Sato, *Grid and Cluster Matrix Computation with Persistent Storage and Out-of-Core Programming.* Cluster

2005: International Conference on Cluster Computing. IEEE Computer Society. Boston, Massachusetts, USA. September 2005.

[62] Frank Leymann. *Web services Flow language (wsfl 1.0).* IBM Software Group. 2001.

[63] Satish Thatte. *Xlang web services for business process design.* Microsoft Corporation. 2001.

[64] Petia Wohed, Wil M.P. van der Aalst, Marlon Dumas, and Arthur H.M. ter Hofstede. *Pattern based analysis of BPEL4WS.* 2002.

[65] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. *The physiology of the grid : An open grid services architecture for distributed systems integration.* 2002.

[66] Sriram Krishnan, Patrick Wagstrom, and Gregor von Laszewski. *GSFL: A worflow framework for grid services.*

[67] Tao Yang and Apostolos Gerasoulis. *Pyrros : static task scheduling and code generation for message passing multiprocessors.* In Proceedings of the 6^{th} international conference on Supercomputing. Pages 428–437. ACM Press. Minneapolis, Minnisota, USA. November 1992.

[68] Ishfaq Ahmad and Yu-Kwong Kwok and Min-You Wu and Wei Shu. *CASCH: A Software Tool for Automatic Parallelization and Scheduling of Programs on Multiprocessors.* Journal IEEE Concurrency. Volume 8. Number 4. Pages 21–33. 2000.

[69] Virginia M. Lo, Sanjay Rajopadhye, Samik Gupta, David Keldsen, Moataz A. Mohamed, Bill Nitzberg, Jan Arne Telle, and Xiaoxiong Zhong. *Oregami : Tools for mapping parallel computations to parallel architectures.* Parallel Programming. Volume 20. Number 3. June 1991.

[70] Kaizar Amin, Mihael Hategan, Gregor von Laszewski, Nestor J. Zaluzec, Shawn Hampton, and Al Rossi. *GridAnt: A Client-Controllable Grid Workflow System.* In 37^{th} Hawaii International Conference on System Science. January 2004.

[71] Peter Newton and James C. Browne. *The code 2.0 graphical parallel programming language.* In Proceedings of the 6^{th} international conference on Supercomputing. Pages 167–177. ACM Press. 1992.

[72] Adam Beguelin, Jack Dongarra, George Al Geist, Robert Manchek, and Keith Moore. *HeNCE : a heterogeneous network computing environment.* Scientific Programming Journal. Volume 3. Number 1. Pages 49–60. Spring 1994.

[73] Ozalp Babaoglu and Lorenzo Alvisi and Alessandro Amoroso and Renzo Davoli and Luigi Alberto Giachini. *Paralex: an Environment for Parallel Programming in Distributed Systems.* In Proceedings of the 6^{th} ACM International Conference on Supercomputing. Pages 178–187. Washington, D.C., USA. 1992.

[74] Markus Lorch and Dennis Kafura. *Symphony - A Java-Based Composition and Manipulation Framework for Computational Grids.* CCGRID '02: Proceedings of the 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid. Page 136. IEEE Computer Society. Washington, DC, USA. 2002.

[75] Tomas Forkert, Hans-Peter Kersken, Andreas Schreiber, Martin Strietzel, and Klaus Wolf. *The distributed engineering framework tent.* In the 4th International Conference on Vector and Parallel Processing. Pages 38–46. Springer-Verlag, 2001.

[76] Shalil Majithia and Matthew Shields and Ian Taylor and Ian Wang. *Triana: A Graphical Web Service Composition and Execution Toolkit.* ICWS '04: Proceedings of the IEEE International Conference on Web Services (ICWS'04). Pages 514. IEEE Computer Society. Washington, DC, USA. 2004.

[77] Oliver Sinnen and Leonel Sousa. *A comparative analysis of graph models to develop parallelising tools.* In 8th IASTED Int'l Conference on Applied Informatics (AI'2000). Innsbruck, Austria. February 2000.

[78] Erol Akarsu, Geoffrey C. Fox, Wojtek Furmanski, and Tomasz Haupt. *Webflow : high-level programming environment and visual authoring toolkit for high performance distributed computing.* In Proceedings of the 1998 ACM/IEEE conference on Supercomputing. Pages 1–7. IEEE Computer Society. 1998.

[79] Toyotaro Suzumura, Hidemoto Nakada, Masayuki Saito, Satoshi Matsuoka, Yoshio Tanaka, and Satoshi Sekiguchi. *The ninf portal : an automatic generation tool for grid portals.* In Proceedings of the 2002 joint ACM-ISCOPE conference on Java Grande. Pages 1–7. ACM Press. 2002.

[80] Sriram Krishnan, Randall Bramley, Dennis Gannon, Madhusudhan Govindaraju, Jay Alameda, Richard Alkire, Timothy Drews, and Eric Webb. *The XCAT science portal.* Proceedings of the 2001 ACM/IEEE conference on Supercomputing (SC'01). ACM Press. Denver, Colorado, USA. November 2001.

[81] Mary Thomas, Stephen Mock, Jay Boisseau. *Development of Web Toolkits for Computational Science Portals: The NPACI HotPage.* 9^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'00). Page 308-309. IEEE Conputer Society. Pittsburgh, Pennsylvania, USA. August 2000.

[82] Ryan P. McCormack, John E. Koontz, and Judith Devaney. *Seamless computing with WebSubmit.* Concurrency : Practice and Experience. Volume 11. Number 15. Pages 949–963. 1999.

[83] Giovani Aloisio and Massimo Cafaro. *Web-based access to the Grid using the Grid Resource Broker Portal.* Concurrency and Computation: Practice and Experience. Volume 14. Grid Computing environments Special Issue 13-14. 2002.

[84] Rajkumar Buyya and David Abramson and Jonathan Giddy *Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid.* In proceedings of the 4^{th} International Conference/Exhibition on High

Performance Computing in the Asia-Pacific Region (ASIA'2000). Volume 1. Pages 283–289. IEEE CS Press. Beijing, China. May 2000.

[85] Anand Natrajan and Anh Nguyen-Tuong and Marty A. Humphrey and Andrew S. Grimshaw. *The Legion Grid Portal.* Concurrency and Computation: Practice and Experience. Volume 14. Grid Computing environments Special Issue 13-14. 2002.

[86] Mary Thomas, Steve Mock, Maytal Dahan, Kurt Mueller, and Don Sutton. *The gridport toolkit : a system for building grid portals.* In the proceedings of the 10^{th} IEEE International Symposium on High Performance Distributed Computing (HPDC'01). Pages 216–227. San Francisco, CA, USA. 2001.

[87] Wes Bethel, Cristina Siegrist, John M. Shalf, Praveenkumar Shetty, T.J. Jankun-Kelly, Oliver Kreylos, Kwan-Liu Ma. *VisPortal: Deploying Grid-enabled Visualization Tools through a Web-portal Interface.* In Proceedings of the 3^{rd} Annual Workshop on Advanced Collaborative Environments. 2003.

[88] Jerry C. Yan. *Performance tuning with AIMS – an Automated Instrumentation and Monitoring System for multicomputers.* Vol.II: Software Technology, Proceedings of the Twenty-Seventh Hawaii International Conference on System Sciences. Pages 625–633. 1994.

[89] Wolfgang E. Nagel, Alfred Arnold, Michael Weber, Hans-Christian Hoppe, and Karl Solchenbach. *VAMPIR : Visualization and analysis of MPI resources.* Supercomputer. Volume 12. Number 1. Pages 69–80. 1996.

[90] Daniel A. Reed, Ruth A. Aydt, Tara M. Madhyastha, Roger J. Noe, Keith A. Shields, and Bradley W. Schwartz. *An overview of the pablo performance analysis environment.* Technical report 61801, University of Illinois, Urbana, Illinois. November 1992.

[91] Barton P. Miller, Mark D. Callaghan, Joanthan M. Cargille, Jeffrey K. Hollingsworth, Bruce R. Irvin, Karen L. Karavanic, Krishna Kunchithapadam, and Tia Newhall. *The paradyn parallel performance measurement tool.* IEEE Computer. Volume 28. Number 11. Pages 37–46. 1995.

[92] Shirley Moore and David Cronk and Kevin S. London and Jack Dongarra. *Review of Performance Analysis Tools for MPI Parallel Programs.* Proceedings of the 8^{th} European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Pages 241–248. Springer-Verlag. Santorini/Thera, Greece. September 2001.

[93] Martin Quinson. *Un outil de prédiction dynamique de performances dans un environnement de metacomputing.* Journal Technique et Science Informatique. Numéro spécial RenPar'13. Number 5. 2002.

[94] Darren J. Kerbyson, John S. Harper, A. Craig, and Graham R. Nudd. *Pace : A toolset to investigate and predict performance in parallel systems.* European Parallel Tools Meeting ONERA, Paris. October 1996.

[95] Hyojong Song, Xin Liu, Dennis Jakobsen, Ranjita Bhagwan, Xianan Zhang, Kenjiro Taura, and Andrew A. Chien. *The microgrid : a scientific tool for modeling computational grids.* In Supercomputing, ACM/IEEE 2000 Conference. Page 53. November 2000.

[96] Benjamin Quétier and Franck Cappello. *A survey of Grid research tools: simulators, emulators and real life platforms.* The 17th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation (IMACS'05). Paris, France. July 2005.

[97] WuŋChun Feng. *Making a case for efficient supercomputing.* The Big Power Squeeze. Volume 1. Number 7. Pages 54–64. October 2003. ISSN:1542ŋ7730.

[98] David M. Brooks, Pradip Bose, Stanley E. Schuster, Hans Jacobson, Prabhakar N. Kudva, Alper Buyuktosunoglu, JohnŋDavid Wellman, Victor Zyuban, Manish Gupta, and Peter W. Cook. *Powerŋaware microarchitecture: Design and modeling challenges for nextŋgeneration microprocessors.* IEEE Micro. Volume 20. Number 6. Pages 26–44. 2000.

[99] KrisztiŠan Flautner and Trevor Mudge. *Vertigo: automatic performance setting for linux.* Proceedings of the 5th symposium on Operating systems design and implementation (OSDI'02). ACM SIGOPS Operating Systems Review. Volume 36. Issue SI (Winter 2002). Pages 105–116. 2002.

[100] Tom Kellera Ramakrishna Kotla, Soraya Ghiasi and Freeman Rawson. *Scheduling processor voltage and frequency in server and cluster systems.* In 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05). Workshop 11. 2005.

[101] Wu-Chun Feng and Chung-Hsing Hsu. *Green destiny and its evolving parts.* In Innovative Supercomputer Architecture Award. 19th International Supercomputer Conference. Heidelberg, Germany. june 2004.

[102] Feng Pan, Vincent W. Freeh, David K. Lowenthal and Nandani Kappiah. *Using multiple energy gears in mpi programs on a powerŋscalable cluster.* In Principles and Practices of Parallel Programming (PPOPP). Pages 164–173. june 2005.

[103] Mahmut T. Kandemir, Guangyu Chen, Konrad Malkowski and Padma Raghavan. Reducing power with performance constraints for parallel sparse applications. In 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05). Workshop 11. Page 231a. 2005.

[104] Hideaki Kimura, Mitsuhisa Sato, Yoshihiko Hotta, Taisuke Boku, Daisuke Takahashi. *Empirical study on Reducing Energy of Parallel Programs using Slack Reclamation by DVFS in a Power-scalable High Performance Cluster* IEEE International Conference on Cluster Computing, 2006 Pages 1-10. Barcelona, Spain. September 2006.

[105] Sudhanva Gurumurthi, Anand Sivasubramaniam, Mahmut Kandemir, and Hubertus Franke. *Drpm: dynamic speed control for power management in server class disks*. SIGARCH Comput. Archit. News. Volume 31. Number 2. Pages 169–181. 2003.

[106] Qingbo Zhu, Francis M. David, Christo F. Devaraj, Zhenmin Li, Yuanyuan Zhou, and Pei Cao. *Reducing energy consumption of disk storage using powerŋaware cache management*. Proceedings of the 10th International Symposium on High Performance Computer Architecture (HPCA '04). Page 118 Washington, DC, USA. 2004.

[107] Eduardo Pinheiro and Ricardo Bianchini. *Energy conservation techniques for disk arrayŋbased servers*. Proceedings of the 18th annual international conference on Supercomputing (ICS '04). Pages 68–78. New York, NY, USA. 2004.

[108] Eduardo Pinheiro and Ricardo Bianchini and Enrique V. Carrera and Taliver Heath. *Dynamic cluster reconfiguration for power and performance*. Compilers and operating systems for low power. Pages 75–93. 2003. Kluwer Academic Publishers.

[109] Michael S. Warren, Eric H. Weigle, and WuŋChun Feng. Highŋdensityŋcomputing: a 240ŋprocessor beowulf in one cubic meter. Proceedings of the 2002 ACM/IEEE conference on Supercomputing (Supercomputing '02). Pages 1–11. Los Alamos, CA, USA. 2002.

[110] Hiroshi Nakashima, Hiroshi Nakamura, Mitsuhisa Sato, Taisuke Boku, Satoshi Matsuoka, Daisuke Takahashi, and Yoshihiko Hotta. Megaproto: A lowŋpower and compact cluster for highŋperformance computing. In 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05). Workshop 11. Page p. 231b. 2005.

[111] NR Adiga, G Almasi, GS Almasi, Y Aridor, R Barik, D Beece, R Bellofatto, G Bhanot, R Bickford, M Blumrich, AA Bright, and J. An overview of the bluegene/l supercomputer, 2002.

[112] Kirk W. Cameron Xizhou Feng, Rong Ge. Power and energy profiling of scientific applications on distributed systems. In 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05). Denver, CO, USA. April 2005.

[113] Margaret Martonosi Canturk Isci. *Runtime power monitoring in highŋend processors: Methodology and empirical*. Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture(MICRO 36). December 2003.

[114] Daisuke Takahashi, Yoshihiko Hotta, Mitsuhisa Sato and Taisuke Boku. *Measurement and characterization of power consumption of microprocessors for powerŋaware cluster*. COOL Chips VII. April 2004.

[115] Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Piotr Luszczek, and Stanimire Tomov. *Using Mixed Precision for Sparse Matrix Computations to Enhance the Performance while Achieving 64-bit Accuracy*. Submitted to ACM TOMS. 2007.

[116] Susan L. Graham, Marc Snir, Cynthia A. Patterson (Eds.). *Getting up to Speed: the Future of Supercomputing.* National Academies Press. 2004.

[117] Serge G. Petiton, Christophe Calvin, Franck Cappello, Michel Dayde, Thierry N'Kaoua, Brigitte Plateau, Marie-Madeleine Rohmer. *Supercalculateurs au Japon* (in french). Rapport de mission. Ambassade de France à Tokyo, Service pour la Science et la Technologie. Juin 2007.