Contribution à la modélisation numérique de la propagation des ondes sismiques sur architectures multicoeurs et hiérarchiques

Fabrice Dupros

THÈSE

présentée à

L'UNIVERSITÉ DE BORDEAUX I

Ecole Doctorale de Mathématiques et d'Informatique

pour l'obtention du

GRADE DE DOCTEUR

Spécialité Informatique

par

Fabrice DUPROS

Contribution à la modélisation numérique de la propagation des ondes sismiques sur architectures multicœurs et hiérarchiques

Soutenue le 13 Décembre 2010 devant le jury composé de :

Raymond NAMYST	Professeur, Université de Bordeaux I	Président du jury
Stéphane GENAUD	Maître de Conférences HDR, Université de Strasbourg	Rapporteur
Stéphane LANTERI	Directeur de recherche, INRIA	Rapporteur
Hideo AOCHI	Sismologue HDR, BRGM	Examinateur
Jean-François MÉHAUT	Professeur, Université Joseph Fourier de Grenoble	Examinateur
Philippe THIERRY	Ingénieur Senior, INTEL France	Examinateur
Dimitri KOMATITSCH	Professeur, Université de Pau et des Pays de l'Adour	Directeur de thèse
Jean ROMAN	Professeur, Institut Polytechnique de Bordeaux	Directeur de thèse

Remerciements

La rédaction de ces quelques lignes marque donc l'aboutissement de cette thèse. L'aventure fut rude et complexe mais très enrichissante. Je tiens donc à remercier tous ceux qui ont contribué de manière directe ou indirecte à l'existence des paragraphes qui vont suivre.

En premier lieu, je remercie mes deux directeurs de thèse, Dimitri Komatitsch et Jean Roman qui auront su me guider tout au long de cette thèse. Leurs conseils et leur rigueur scientifique ont été précieux m'évitant parfois d'enfoncer des portes grandes ouvertes. Merci encore pour les (nombreuses) corrections, je garde les scans de dimitri comme des collectors ! J'en profite pour préciser aux futurs doctorants que 99,9% de ces corrections sont de vraies améliorations.

Cette thèse n'aurait pas existé sans les encouragements (répétés) de Jean-François Méhaut (et sans le projet NUMASIS !). Il faut se rappeler de cette rencontre initiale en Martinique il y a près de dix ans et d'une occasion ratée (reportée) de me lancer dans l'aventure. J'ai eu la chance de ne pas avoir à me demander trop longtemps si le choix effectué à ce moment était le bon, c'est un luxe que je te remercie de m'avoir offert.

Je remercie également chaleureusement les autres membres de mon jury, particulièrement les deux rapporteurs Stéphane Lantéri et Stéphane Genaud pour leur réactivité et leurs remarques concernant le manuscrit. Un grand merci à Raymond Namyst d'avoir accepté de présider mon jury et à Philippe Thierry pour ses conseils.

Un grand merci à Pascal Poullet pour sa présence lors de ma soutenance. C'est également l'occasion de te remercier de manière rétrospective pour les cours (agités) d'analyse numérique, tes encouragements lors de mon départ à Lyon et ta proposition de thèse. Ces jalons couvrent presque les quinze dernières années et j'espère que nos chemins continueront à se croiser.

Les nombreuses discussions avec les membres des équipes Inria (Hiepacs, Runtime à Bordeaux ou le groupe Méhaut à Grenoble !) m'ont été précieuses et je remercie notamment Alexandre Carissimi, Christiane Pousa et Mathieu Faverge pour leur aide.

Enfin, ces premiers remerciements ne seraient pas complets sans une référence à Marc Garbey et Damien Tromeur-Dervout qui m'ont mis le pied à l'étrier. Les expériences au CDCSP à Lyon puis à l'université de Houston restent de très bons souvenirs.

Coté BRGM, mes remerciements vont d'abord à Jean-Marc Trouillard, François Robida et Jacques Vairon qui ont encouragé ce projet. Le déroulement de cette thèse a montré que cette initiative comportait certaines difficultés (c'est un euphémisme !). Leur soutien est donc d'autant plus remarquable.

Je remercie également tous mes collègues du BRGM. Leur souplesse et leur patience m'a souvent permis d'avancer. Une mention particulière pour Hidéo, Dr.Florent, Dr.André et Pascal qui auront suivi de près les soubresauts de cette thèse.

Enfin, cette thèse n'aurait sans doute pas existée sans les conseils, les debriefing matinaux et l'amitié de Faïza. C'est fort agréable de travailler dans une telle ambiance et je te remercie pour les nombreuses idées, les corrections et les relectures (désolé pour avoir si souvent mobilisé ton tableau).

Une thèse consiste également à surmonter un état de stress permanent. Néanmoins, selon la formule consacrée : Rien ne doit supplanter le dispositif !
C'est donc avec un grand sourire que je rédige ces lignes afin de remercier Jean-Marie et Théodore pour leur profonde amitié ces dernières années (entre 25 et 30 ans pour être précis). De Zorro et Sergent Garcia à mixte A, en passant par Robocop au Lycée ou le fameux retourné acrobatique de Fouillole, vous aurez toujours été présent dans différentes configurations. J'ai également pu finir cette thèse grâce à votre aide malgré la distance (pliss foss !).

Je n'oublie pas le dispositif bien huilé de la Darse à Pointe à Pitre (vive les "petites ligues", le "Chap" dès la 15e minute et S.Kemp a 31 pts, 20 rds avec les Cavs), du Courteline à Lyon (Trinity) ou encore mes compagnons du Glenchester à Houston. Merci aux Boston (vive Melun), aux Libri (qui veux encore du foie gras ?) et à tous les autres pour leur amitié.

Enfin, ce paragraphe se termine par un grand MERCI à mes parents. Merci Papa (qui aura été de presque tous les déménagements !) et Maman (vive le téléphone gratuit) pour avoir toujours été présents. Merci également à Carine, Francine et Stéphane pour leurs encouragements.

Comme annoncé en introduction, une thèse c'est avant tout une aventure. Elle aurait très vite tournée court sans le soutien de Gladys et Joris (et ses longues nuits de sommeil !). Donc un grand MERCI pour votre présence et pour tout le reste.

Life is one big road with lots of signs. So when you riding through the ruts, don't complicate your mind. Flee from hate, mischief and jealousy. Don't bury your thoughts, put your vision to reality. Wake up and Live!

Robert Nesta Marley, Album Survival, 1979

Résumé

En termes de prévention du risque associé aux séismes, la prédiction quantitative des phénomènes de propagation et d'amplification des ondes sismiques dans des structures géologiques complexes devient essentielle. Dans ce domaine, la simulation numérique est prépondérante et l'exploitation efficace des techniques de calcul haute performance permet d'envisager les modélisations à grande échelle nécessaires dans le domaine du risque sismique.

Plusieurs évolutions récentes au niveau de l'architecture des machines parallèles nécessitent l'adaptation des algorithmes classiques utilisées pour la modélisation sismique. En effet, l'augmentation de la puissance des processeurs se traduit maintenant principalement par un nombre croissant de cœurs de calcul et les puces multicœurs sont maintenant à la base de la majorité des architectures multiprocesseurs. Ce changement correspond également à une plus grande complexité au niveau de l'organisation physique de la mémoire qui s'articule généralement autour d'une architecture *NUMA* (Non Uniform Memory Access pour accès mémoire non uniforme) de profondeur importante.

Les contributions de cette thèse se situent à la fois au niveau algorithmique et numérique mais abordent également l'articulation avec les supports d'exécution optimisés pour les architectures multicœurs. Les solutions retenues sont validées à grande échelle en considérant deux exemples de modélisation sismique. Le premier cas se situe dans la préfecture de Niigata-Chuetsu au Japon (événement du 16 juillet 2007) et repose sur la méthode des différences finies. Le deuxième exemple met en œuvre la méthode des éléments finis. Un séisme hypothétique dans la région de Nice est modélisé en tenant compte du comportement non linéaire du sol.

Mots-clés: calcul haute performance, modélisation sismique, architectures NUMA, processeurs multicœurs.

Abstract

One major goal of strong motion seismology is the estimation of damage in future earthquake scenarios. Simulation of large scale seismic wave propagation is of great importance for efficient strong motion analysis and risk mitigation. Being particularly CPU-consuming, this three-dimensional problem makes use of high-performance computing technologies to make realistic simulation feasible on a regional scale at relatively high frequencies.

Several evolutions at the chip level have an important impact on the performance of classical implementation of seismic applications. The trend in parallel computing is to increase the number of cores available at the shared-memory level with possible non-uniform cost of memory accesses. The increasing number of cores per processor and the effort made to overcome the limitation of classical symmetric multiprocessors (*SMP*) systems make available a growing number of *NUMA* (Non Uniform Memory Access) architecture as computing node. We therefore need to consider new approaches more suitable to such parallel systems.

This PhD work addresses both the algorithmic issues and the integration of efficient programming models for multicore architectures. The proposed contributions are validated with two large scale examples. The first case is the modeling of the 2007 Niigata-Chuetsu, Japan earthquake based on the finite differences numerical method. The second example considers a potential seismic event in the Nice sedimentary basin in the French Riviera. The finite elements method is used and the nonlinear soil behavior is taken into account.

Keywords: high performance computing, seismic modeling, NUMA architecture, multicore processor.

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Les réseaux d'observation sismique n'ont cessé de progresser ces dernières années, cependant la compréhension des mécanismes physiques associés aux tremblements de terre reste difficile et incomplète. Certaines régions françaises sont particulièrement concernées par le risque sismique, par exemple les Alpes, l'agglomération de Nice et les Antilles. Les récents évènements de Chine (12 mai 2008, Mw 7.9), d'Haïti (12 janvier 2010, Mw 7.7) ou du Chili (27 février 2010, Mw 8.8) ont encore montré le potentiel destructeur de ces événements. En termes de prévention du risque associé aux séismes, la prédiction quantitative des phénomènes de propagation et d'amplification des ondes sismiques dans des structures géologiques complexes devient essentielle. Dans ce contexte et en considérant les hypothèses simplificatrices fortes sous-jacentes aux solutions théoriques, la simulation numérique est un outil important.

Tout d'abord, il s'agit de comprendre et de caractériser les mécanismes à la source. La majorité des séismes sont causés par la rupture des roches en profondeur, libérant ainsi une importante quantité d'énergie. Cette rupture se produit principalement le long d'une ou de plusieurs failles et les phénomènes physiques intervenant à cette échelle sont complexes. Leur compréhension (localisation, dimension, mode de rupture) est cruciale afin de modéliser de façon réaliste la propagation des ondes générées. Les tremblements de terre produisent différents types d'ondes qui se caractérisent par des vitesses de propagation variables en fonction des propriétés du milieu. Les ondes de volume (ondes P et S) sont les premières à être enregistrées sur les sismogrammes ; elles provoquent respectivement des déformations de type compression et cisaillement. Les ondes de surface (ondes de Love et de Rayleigh) sont guidées par la surface de la terre avec une amplitude élevée. La simulation de la propagation des ondes sismiques doit donc particulièrement tenir compte de la géométrie et des caractéristiques des milieux traversés. Les échelles d'espace prises en compte varient de l'échelle régionale (quelques dizaines de kilomètres dans le cas d'un bassin) à l'échelle globale de la terre.

En revanche, lors de mouvements forts, les hypothèses d'un comportement élastique du sol (éventuellement avec de l'atténuation) sont insuffisantes. En effet, l'importance de la prise en compte des conditions géologiques locales et du comportement du sol (i.e. effets de site) dans l'analyse de la réponse sismique a été largement soulignée dans la littérature (Mexique 1985, Loma Prieta 1989, Northridge 1994, Taiwan 1999 [20, 23, 55]). La modélisation géomécanique intervient à une échelle plus locale et permet de prédire le comportement du sol sous sollicitations complexes. Dans le cadre des risques sismiques, elle permet de quantifier les effets de site non-linéaires en tenant compte de la répartition en profondeur des différentes formations géologiques et de leurs natures (propriétés des sols, saturation en eau, ...) ainsi que de la topographie. En effet, au-delà d'une certaine sollicitation, des déformations permanentes du matériau apparaissent (phénomène de plasticité).

La couverture des différents aspects du risque sismique repose sur une connaissance approfondie de la structure du sous-sol. La compilation des différentes mesures et sondages combinée à la connaissance du contexte géologique régional permet de produire des représentations tridimensionnelles du sous-sol. Ces modèles traduisent l'organisation spatiale complexe des différentes couches géologiques (plis, failles). Cependant, la description des structures géologiques et des propriétés mécaniques associées est souvent très partielle. Les différentes échelles en temps et en espace impliquées dans les phénomènes sismiques (de l'ordre du mètre à plusieurs dizaines de kilomètres) nécessitent la mise en œuvre de différentes méthodes de discrétisation numérique.

Dans le cas de l'équation élastodynamique, différentes méthodes numériques ont été proposées. La méthode des différences finies constitue l'une des techniques classiques [108, 153]. Elle repose, la plupart du temps, sur l'utilisation d'une grille dite en quinconce pour la discrétisation des champs de vitesse et de contrainte. La simplicité de son implémentation explique son succès et la littérature abondante autour de cette approche démontre son efficacité au niveau applicatif. Un état de l'art sur cette méthode peut être trouvé dans [119]. Malgré les nombreuses propositions d'amélioration [27, 73, 125, 136, 146], une des limitations réside en la difficulté de modéliser des milieux complexes d'un point de vue géométrique.

Les méthodes variationnelles constituent une alternative pour ce type de modélisation. Elles possèdent l'avantage de la prise en compte plus aisée des géométries complexes. En dépit d'une dispersion numérique importante [48, 144], la méthode des éléments finis classique a été utilisée [107, 110]. L'introduction de la méthode des éléments spectraux [93] permet de contourner ce problème en utilisant une base polynomiale d'ordre élevé. Son efficacité a été démontrée par de nombreuses applications, aussi bien à l'échelle locale [152], régionale [89], ou de la terre globale [34, 91].

Les méthodes de Galerkin discontinues proposent un cadre numérique flexible [41, 82]. Cette technique permet de définir localement l'ordre d'interpolation permettant de construire une solution définie par élément sur des maillages non structurés et éventuellement non conformes. Elle est particulièrement bien adaptée aux phénomènes présentant des discontinuités telles que la modélisation de la source [18].

Les méthodes d'éléments frontières sont également largement utilisées ; elles consistent à intégrer le problème uniquement sur le contour de chaque élément [29]. La taille du problème à résoudre est ainsi réduite d'une dimension. La prise en compte de modèles hétérogènes complexes est néanmoins difficile et le coût de calcul reste élevé en raison de l'inversion de matrices pleines non symétriques. Des techniques numériques ont récemment été proposées afin de surmonter cette dernière limitation [37].

En considérant le comportement non-linéaire du sol, la méthode des éléments finis demeure la technique de référence. Principalement appliquée à des problèmes monodimensionnels, en particulier dans le domaine de la géotechnique [17, 57, 95, 141], la mise en œuvre en 3D est rendue difficile par la complexité du paramétrage des lois de comportement [101] et la difficulté du problème numérique sous-jacent. Si les techniques d'intégration explicite en temps ont été utilisées avec succès sur des exemples tridimensionnels synthétiques [156], leur application dans des cas réalistes reste encore difficile [1, 102]. Les approches implicites basées sur un chargement incrémental sont alors privilégiées.

Les avantages et les limitations de ces différentes approches sont donc assez bien connus en terme de qualité numérique et leur adaptabilité sur architectures parallèles constitue un paramètre déterminant. En effet, la dimension et la complexité des modélisations requises dans le domaine du risque sismique dépasse largement les capacités des machines de bureau et le recours aux techniques de calcul haute performance est indispensable. Les méthodes des différences finies [39, 61, 117], des éléments finis [5] ou des éléments spectraux [88] ont déjà prouvé leur efficacité sur des architectures massivement parallèles. Dans le cas non linéaire, des modélisations ont été proposées pour des schémas d'intégration explicite [156] ou implicite [102]. Néanmoins, ces exemples classiques sont majoritairement basés sur un modèle de programmation par échange de messages (Message Passing Interface - MPI). Cette approche constituait jusqu'ici un standard pour la programmation des architectures parallèles, mais elle semble désormais moins adaptée à l'évolution récente des architectures multiprocesseurs.

En effet, depuis la fin de de la "course aux gigahertz", l'augmentation de la puissance des processeurs se traduit principalement par un nombre croissant de cœurs de calcul. Les puces multi-cœurs sont maintenant à la base de nombreuses configurations multiprocesseurs. Au niveau des clusters de calcul, la tendance n'est plus nécessairement d'augmenter le nombre de nœuds mais plutôt d'accroître le nombre de cœurs par processeur et le nombre de processeurs multi-cœurs à l'intérieur de chaque nœud. Cette évolution induit une hiérarchisation croissante de la mémoire afin d'éviter l'importante contention sur le bus mémoire qui bridait l'exploitation des architectures SMP (Symmetric MultiProcessing) majoritairement utilisées jusqu'ici.

La stratégie de hiérarchisation des accès à la mémoire permet de relacher cette contrainte en distribuant physiquement la mémoire à différents niveaux de l'architecture tout en offrant une vision globale au programmeur. La contrepartie est un temps d'accès aux données qui n'est pas uniforme entre les différents cœurs d'un nœud multiprocesseurs. On parle alors d'architectures NUMA (Non Uniform Memory Access). Des liens de type Hypertransport (AMD) ou QPI (Intel) permettent d'agréger efficacement plusieurs processeurs sur une même carte mémoire. Ces contrôleurs mémoire multicanaux intégrés permettent théoriquement de démultiplier le débit mémoire et on les retrouve au niveau des processeurs Intel Nehalem et AMD Opteron. Ensuite, des technologies de type NUMAlink (SGI), FAME Scalability Switch ou Bull Coherency Switch (Bull) permettent de relier les cartes multiprocesseurs. Les architectures ainsi organisées peuvent contenir plus d'une centaine de cœurs avec différents niveaux de hiérarchie mémoire. Ces briques de base sont ensuite interconnectées par des réseaux rapides de type Infiniband ou Myrinet.

Le caractère hiérarchique et hybride des architectures est renforcé aujourd'hui par l'introduction d'accélérateurs ou coprocesseurs tels que les GPU. Ces derniers délivrent une puissance crête élevée (environ 180 Gflops en double précision sur architecture Nvidia *Fermi*) reposant sur l'exploitation d'un parallélisme de données et sur une granularité fine des calculs. On parle alors de h^3-*computing* traduisant le caractère *h*ybride, *h*iérarchique et *h*étérogène des calculateurs modernes. L'adaptation des applications à ces évolutions est indispensable pour envisager les modélisations à grande échelle nécessaires dans le domaine du risque sismique. La majorité des références [2, 116] se concentrent sur le problème du portage des méthodes de la géophysique sur accélérateurs de type GPU. Si la puissance de calcul offerte par ces architectures a généré une importante activité de recherche, la prise en compte des différents niveaux de complexité présents au sein d'une architecture multicœurs, est encore incomplète. Quelle que soit l'orientation choisie pour l'évolution des architectures de calcul, par exemple la convergence CPU-GPU (architecture *AMD Fusion*) ou la démultiplication du nombre de cœurs (projet *Knights Corner* chez Intel)

l'exploitation d'un nombre significativement plus important de cœurs demeure une constante.

La rapidité et l'importance des évolutions matérielles comparées à la durée de vie d'une application scientifique complexe nécessitent donc de se focaliser sur les aspects algorithmiques et sur les modèles de programmation associés. En effet, les défis offerts au programmeur ne pourront pas être surmontés en adaptant les approches existantes aux futures puces. Une refonte plus large semble nécessaire en masquant aux applications la complexité des architectures. Il s'agit donc de trouver l'articulation nécessaire entre l'algorithmique spécifique de l'application et les possibilités offertes par les supports d'exécution et/ou les bibliothèques scientifiques. C'est l'objectif principal de cette thèse, qui étudie la mise en œuvre efficace des outils de modélisation sismique sur architectures modernes. Les contributions proposées se situent tant au niveau des algorithmes, des structures de données que de l'exploitation des supports d'exécution adaptés à ces architectures. Nous considérons les méthodes classiques des différences finies et des éléments finis dans le contexte du risque sismique en nous focalisant sur l'exploitation des nœuds de calcul à base de cœurs généralistes. Les contributions proposées sont validées sur des cas réels d'applications prenant en compte un comportement élastique ou des rhéologies complexes pour le sol.

Le manuscrit de cette thèse est organisé en **sept** chapitres. Le **premier chapitre** qui conclut cette **introduction générale** présente le sujet de l'étude et précise le cadre en décrivant les aspects physiques et numériques pour la modélisation de la propagation des ondes sismiques. Les méthodes des différences finies et des éléments finis sont décrites en soulignant les caractéristiques importantes de leurs implémentations. Ce chapitre propose également de positionner ces techniques numériques par rapport aux principales évolutions des architectures de calcul et des modèles de programmation associés.

La **première partie** du manuscrit aborde ensuite la mise en œuvre de la méthode des différences finies sur architectures hiérarchiques et multicœurs. Elle se décompose en **quatre chapitres**. Les algorithmes présentés ont été implémentés à partir du code ONDES3D développé au BRGM. Ce dernier est utilisé pour des problématiques d'inversion de la source sismique mais également pour la propagation d'ondes sismiques en milieu hétérogène et complexe.

Le **chapitre 2** aborde les problèmes d'équilibrage de charge induit par l'implémentation de conditions numériques absorbantes aux bords du domaine de calcul. La formulation utilisée permet en effet d'éviter la réflexion artificielle des ondes dans le domaine physique. En revanche, le coût CPU de cette formulation est plus élevé que l'évaluation des points situés à l'intérieur du domaine physique. Dans le cas d'une décomposition régulière du domaine de calcul, les points de la couche absorbantes sont donc à l'origine d'un déséquilibre de charge entre processeurs. Nous présentons plusieurs pistes permettant de surmonter cette limitation en introduisant un second niveau de parallélisme et en exploitant les mécanismes offerts par le support d'exécution.

Le **chapitre 3** concerne l'affinité mémoire. Il s'agit dans un premier temps de caractériser la pénalité NUMA en tenant compte du contexte applicatif. Les impacts du schéma d'accès aux données, de la stratégie de *multithreading* implémentée ou de la taille du problème sont évalués sur différentes architectures hiérarchiques. Deux approches sont ensuite étudiées plus spécifiquement. La première proposition consiste à implémenter un mécanisme d'ordonnancement structuré des *threads* en s'appuyant sur la grille cartésienne décrivant le domaine de simulation. La mise en œuvre explicite au niveau applicatif sera notamment décrite et discutée. La deuxième approche est basée sur l'environnement MAI[1] (Memory Affiny interface) qui propose une approche peu

1. http ://mai.gforge.inria.fr/

intrusive afin de définir la politique mémoire la mieux adaptée.

La faible exploitation de la puissance offerte par les processeurs pour les applications de type différences finies est abordée dans le **chapitre 4**. En effet, le caractère *memory bound* de ces méthodes numériques limite les performances à 20% - 30% de la puissance crête. Nous proposons donc d'explorer les gains offerts par une décomposition dans le domaine espace-temps. Cette technique est particulièrement adaptée à la fois aux modélisations cibles et aux architectures multicœurs.

L'implémentation des solutions précédentes sur des clusters de nœuds multicœurs et hiérarchiques est traitée dans le **chapitre 5**. Un cas réel d'application est présenté avec le séisme de Niigata-Chuetsu au Japon (2007). Ce dernier est utilisé comme support de l'analyse des performances parallèles des algorithmes introduits sur un exemple représentatif de modélisation à l'échelle régionale.

La **deuxième partie** du manuscrit est consacrée à l'implémentation sur architectures multicœurs de la méthode des éléments finis dans le cas de rhéologies complexes. Elle comprend **deux chapitres**. Les approches numériques et algorithmiques présentées dans cette partie ont été implémentées dans le logiciel GEFDYN (Géomecanique Elements Finis DYNamiques) [2] développé principalement au sein du Laboratoire de Mécanique des Sols, Structures et Matériaux de l'Ecole Centrale de Paris avec des contributions du BRGM.

Le **chapitre 6** détaille les aspects algorithmiques. Une technique de partitionnement hybride du maillage est notamment proposée. Elle permet de tenir compte du contexte applicatif afin d'aborder le problème de déséquilibre de charge issu des lois de comportement complexes utilisées au niveau des couches sédimentaires. Les aspects matriciels sont ensuite considérés avec une description de l'utilisation du solveur direct haute performance PASTIX [3] et une discussion sur son intégration dans un contexte de programmation hybride.

Le **chapitre 7** présente la simulation en vraie grandeur et en 3D d'un séisme hypothétique dans l'agglomération de Nice. Les résultats physiques sont présentés ainsi que les performances parallèles jusqu'à 1024 cœurs.

La dernière section propose un résumé des contributions développées dans ce manuscrit et présente quelques perspectives afin d'étendre ce travail.

2. http ://www.mssmat.ecp.fr/gefdyn
3. https ://gforge.inria.fr/projects/pastix/

Chapitre 1

Cadre de l'étude

Contents

1.1 Description du problème

Nous rappelons les principes généraux de la modélisation de la propagation des ondes sismiques en milieu géologique complexe. Les ouvrages classiques [6, 94] peuvent être consultés afin d'approfondir cette introduction .

1.1.1 Equations du mouvement

L'application des principes de la mécanique des milieux continus, notamment la conservation de la masse, de la quantité de mouvement et de l'énergie permettent d'exprimer l'équation de mouvement suivante :

$$\rho \frac{\partial^2 u_i}{\partial t^2} = \frac{\partial \sigma_{ij}}{\partial x_j} + f_i \tag{1.1}$$

avec ρ exprimant la densité du milieu, $\frac{\partial^2 u_i}{\partial t^2}$ et f_i représentant respectivement la $i$ème composante d'accélération et des forces externes f ; σ_{ij} est la composante ij du tenseur de contraintes.

Nous nous plaçons dans l'hypothèse de petites déformations permettant d'exprimer une relation linéaire entre le tenseur symmétrique des déformations ε et les déplacements u :

$$\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \tag{1.2}$$

1.1.2 Lois de comportement

La résolution des équations précédentes nécessite d'exprimer la relation entre les contraintes et les déformations. Il s'agit de la loi de comportement définissant la rhéologie du matériau (modèle élastique, viscoélastique, élastoplastique ...).

Comportement élastique

Un milieu est dit élastique s'il ne subit que des déformations réversibles : en supprimant les forces responsables de ces déformations, le milieu retrouve sa forme initiale. Dans des conditions standard, les roches se comportent de façon élastique. Pour des déformations suffisamment faibles, une relation linéaire entre contraintes et déformations est généralement observée. La loi de Hooke dans le cas d'un milieu isotrope, c'est-à-dire lorsque les propriétés du milieu sont identiques dans toutes les directions d'espace, s'écrit :

$$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij} \tag{1.3}$$

où δ_{ij} représente le symbole de Kronecker (δ_{ij}=1 si $i = j$ et 0 sinon). λ et μ sont les deux constantes nécessaires pour caractériser le milieu, il s'agit des paramètres de Lamé. Les paramètres λ, μ et ρ dépendent ici uniquement de l'espace.

La vitesse de propagation des ondes P de compression et des ondes S de cisaillement s'exprime en fonction des propriétés du milieu par :

$$v_p = \sqrt{\frac{\lambda + 2\mu}{\rho}} \qquad et \qquad v_s = \sqrt{\frac{\mu}{\rho}} \qquad avec \qquad v_s < \frac{v_p}{\sqrt{2}} \qquad . \tag{1.4}$$

La dernière relation entre v_s et v_p traduit la vitesse de propagation plus élevée des ondes P.

Rhéologies complexes

Les déformations permanentes observées lors de certains événements sismiques sont liées au comportement non linéaire du sol avec des déformations non proportionnelles à la sollicitation. Deux grandes catégories d'approches sont décrites dans la littérature pour ce type de modélisation :

- l'approche linéaire équivalente ;
- l'approche cyclique non linéaire basée sur des lois de comportement élastoplastique.

L'utilisation du modèle linéaire équivalent est historiquement très répandue notamment au niveau des bureaux d'études. Ce modèle repose sur la théorie de la viscoélasticité non linéaire, le sol est remplacé par un matériau linéaire de telle sorte que la rigidité de cisaillement et l'énergie dissipée (amortissement) soient équivalentes à celles d'un matériau non linéaire. Le comportement

du sol est représenté par une courbe reliant les contraintes (τ) et les déformations en cisaillement (γ). La figure 1.1 illustre cette relation, avec τ_c et γ_c correspondant respectivement aux modules de cisaillement en contraintes et en déformation. L'aire de la figure (A_{loop}) représente la dissipation d'énergie, elle est matérialisée en bleu. Les paramètres G_{sec} et ξ définissent les

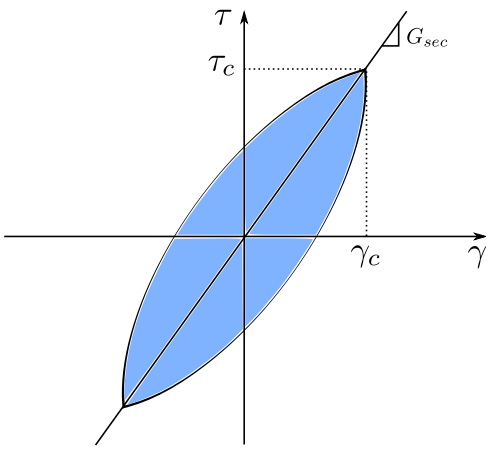

FIGURE 1.1 – Définition des paramètres de l'approche linéaire équivalente

propriétés équivalentes du sol. Ils sont donnés par les relations :

$$G_{sec} = \frac{\tau_c}{\gamma_c} \qquad et \qquad \xi = \frac{1}{2\pi}\frac{A_{loop}}{G_{sec}\gamma_c^2} \tag{1.5}$$

Il s'agit donc par la suite de fournir les courbes reliant $G-\gamma$ et $\xi-\gamma$ pour caractériser le problème. Ces dernières peuvent être déduites par des essais en laboratoire et ensuite utilisées pour des simulations numériques.

Cette procédure simplifiée n'est pas applicable dans le cas de non linéarités sévères et les modèles basés sur la théorie de l'élastoplasticité sont alors privilégiés. Parmi ces derniers, les approches élastiques ou élastoplastiques parfaites sont les plus simples. D'autres modèles issus de l'élasticité non linéaire, l'hypoélasticité, l'élasticité avec écrouissage parfait, peuvent également être exploités. La difficulté est alors d'identifier les nombreux paramètres nécessaires à leur calibration, en particulier dans le cas de modélisations tridimensionnelles [101].

Le critère de rupture de Mohr-Coulomb (modèle élastoplastique parfait) est privilégié dans cette thèse. Il permet de limiter le nombre de paramètres à définir en conservant un comportement mécanique réaliste. La relation suivante permet de le définir :

$$f(\sigma) = \mid \tau \mid -\mu\sigma_n - c \leq 0 \tag{1.6}$$

avec f désignant une fonction du tenseur des contraintes.
La figure 1.2 propose une représentation schématique de ce critère dans la plan $\sigma_n - \tau$ (i.e. contrainte normale - contrainte de cisaillement). La limite élastique est définie par la ligne $\tau = \sigma_n \tan\varphi + c$: l'ordonnée à l'origine c représente la cohésion du matériau, la pente $\mu = \tan\varphi$ est le cœfficient de frottement interne, φ est le frottement interne. Les irréversibilités ont lieu lorsqu'il y a égalité dans l'équation 1.6. En plus des paramètres élastiques, seules les grandeurs φ et c doivent être définies. Pour les roches granulaires dont les déformations plastiques sont dues à

de multiples critères (glissement relatif entre grains, microfissures, rupture de grains ...), ces paramètres sont à relier au glissement entre les grains.

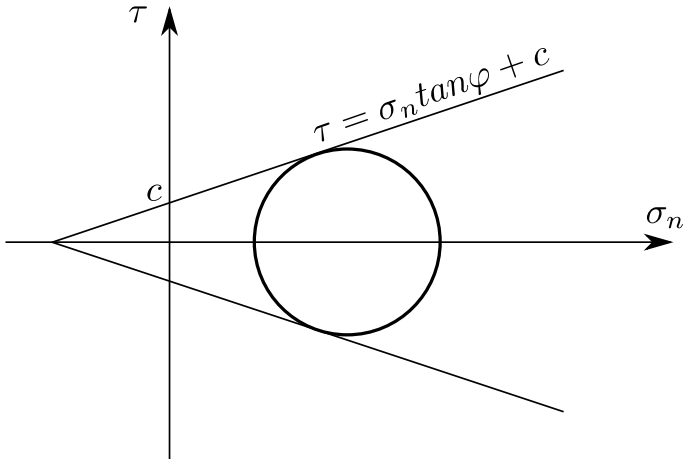

FIGURE 1.2 – Représentation schématique du critère élastoplastique de Mohr-Coulomb

1.2 Méthodes numériques

1.2.1 Méthode des différences finies

La méthode des différences finies demeure le schéma numérique le plus répandu pour la résolution de l'équation de l'élastodynamique. Elle a notamment été popularisée par Madariaga et Virieux [108, 153] avec une approximation d'ordre 2 en espace reposant sur une grille de discrétisation en quinconces. L'extension à l'ordre 4 en espace a été proposée par Levander [97]. L'implémentation du code ONDES3D repose sur cette dernière formulation.

Equations de l'élastodynamique

Le système à résoudre dans le cas d'un comportement élastique du sol est le suivant :

$$\begin{cases} \rho \frac{\partial^2 u_i}{\partial t^2} = \frac{\partial \sigma_{ij}}{\partial x_j} + f_i \\ \varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right) \\ \sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij} \end{cases} \tag{1.7}$$

Il s'agit d'un système hyperbolique d'ordre 2. Ce dernier est généralement résolu avec les champs de vitesse et le tenseur des contraintes comme inconnues. La substitution du terme $\frac{\partial u_i}{\partial t}$ dans l'équation de mouvement et la dérivation de la loi de Hooke combinée à la même substitution permettent d'écrire le système hyperbolique d'ordre 1 suivant :

$$\begin{cases} \rho \frac{\partial}{\partial t} v_i &= \frac{\partial}{\partial x_j} \sigma_{ij} + f_i \\ \frac{\partial}{\partial t} \sigma_{ij} &= \lambda \delta_{ij} \varepsilon_{kk} + \mu \left(\frac{\partial}{\partial x_j} v_i + \frac{\partial}{\partial x_i} v_j\right) \end{cases} \tag{1.8}$$

Le système précédent écrit en dimension 3 contient 9 équations :

$$\begin{cases} \rho\frac{\partial}{\partial t}v_x &= \frac{\partial}{\partial x}\sigma_{xx} + \frac{\partial}{\partial y}\sigma_{xy} + \frac{\partial}{\partial z}\sigma_{xz} + f_x \\ \rho\frac{\partial}{\partial t}v_y &= \frac{\partial}{\partial x}\sigma_{yx} + \frac{\partial}{\partial y}\sigma_{yy} + \frac{\partial}{\partial z}\sigma_{yz} + f_y \\ \rho\frac{\partial}{\partial t}v_z &= \frac{\partial}{\partial x}\sigma_{zx} + \frac{\partial}{\partial y}\sigma_{zy} + \frac{\partial}{\partial z}\sigma_{zz} + f_z \end{cases} \tag{1.9}$$

$$\begin{cases} \frac{\partial}{\partial t}\sigma_{xx} &= \lambda\left(\frac{\partial}{\partial x}v_x + \frac{\partial}{\partial y}v_y + \frac{\partial}{\partial z}v_z\right) + 2\mu\frac{\partial}{\partial x}v_x \\ \frac{\partial}{\partial t}\sigma_{yy} &= \lambda\left(\frac{\partial}{\partial x}v_x + \frac{\partial}{\partial y}v_y + \frac{\partial}{\partial z}v_z\right) + 2\mu\frac{\partial}{\partial y}v_y \\ \frac{\partial}{\partial t}\sigma_{zz} &= \lambda\left(\frac{\partial}{\partial x}v_x + \frac{\partial}{\partial y}v_y + \frac{\partial}{\partial z}v_z\right) + 2\mu\frac{\partial}{\partial z}v_z \\ \frac{\partial}{\partial t}\sigma_{xy} &= \mu\left(\frac{\partial}{\partial y}v_x + \frac{\partial}{\partial x}v_y\right) \\ \frac{\partial}{\partial t}\sigma_{xz} &= \mu\left(\frac{\partial}{\partial z}v_x + \frac{\partial}{\partial x}v_z\right) \\ \frac{\partial}{\partial t}\sigma_{yz} &= \mu\left(\frac{\partial}{\partial z}v_y + \frac{\partial}{\partial y}v_z\right) \end{cases} \tag{1.10}$$

Schémas différences finis et stencils de calcul

Classiquement, la méthode des différences finies repose sur l'exploitation des développements de Taylor afin d'approcher les dérivées spatiales. Un exemple simple est donné ci-dessous avec une approximation d'ordre 2 de la dérivée d'une fonction f. La mise en œuvre est très simple et consiste à sommer différentes combinaisons de développements. Le nombre de termes à sommer et les directions d'espace concernées définissent un stencil de calcul.

$$\begin{cases} f\left(x+\frac{h}{2}\right) = f(x) + \frac{h}{2}f'(x) + \frac{h^2}{8}f''(x) + \frac{h^3}{48}f'''(x) + \frac{h^4}{384}f^{(4)}(x) + \frac{h^5}{3840}f^{(5)}(x) + O(h^6) \\ f\left(x-\frac{h}{2}\right) = f(x) - \frac{h}{2}f'(x) + \frac{h^2}{8}f''(x) - \frac{h^3}{48}f'''(x) + \frac{h^4}{384}f^{(4)}(x) - \frac{h^5}{3840}f^{(5)}(x) + O(h^6) \end{cases} \tag{1.11}$$

La valeur h correspond à un incrément dans la direction d'espace, il s'agit du pas d'espace utilisé pour la discrétisation du problème. La différence entre les deux équations précédentes permet d'écrire la relation 1.12

$$f(x+\frac{h}{2}) - f(x-\frac{h}{2}) = hf'(x) + \frac{h^3}{24}f'''(x) + \frac{h^5}{1920}f^{(5)}(x) + O(h^7). \tag{1.12}$$

En ignorant les termes supérieurs à l'ordre 3, on obtient une expression simple du second ordre de la dérivée $f'(x)$ de la fonction f. L'évaluation de la dérivée au point x nécessite alors les informations venant des points de gauche et de droite dans le cas d'un grille cartésienne. L'extension en 2D et 3D se fait de manière similaire.

$$f'(x) = \frac{1}{h}\left\{f(x+\frac{h}{2}) - f(x-\frac{h}{2})\right\}. \tag{1.13}$$

Le nombre de points à sommer dans chaque direction d'espace définit l'ordre du schéma numérique et les opération à effectuer définissent un stencil de calcul. Il s'agit pour chaque point d'exploiter les informations des points voisins dans les directions *Nord-Sud*, *Est-Ouest* et Haut-Bas dans le cas d'une grille cartésienne. La figure 1.3 illustre une formulation centrée de la méthode des différences finies d'ordre 2.

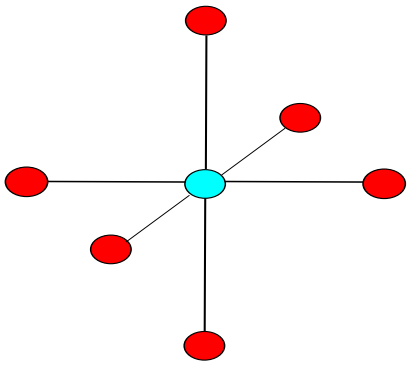

FIGURE 1.3 – Représentation simplifiée d'un stencil d'ordre 2 en 3D.

Discrétisation de l'équation de l'élastodynamique

La dispersion numérique générée par la discrétisation de l'équation de l'élastodynamique basée sur des méthodes différences finies simples [7, 84] a rapidement conduit à l'introduction de nouvelles approches. En effet, les grilles classiques rendaient difficile la prise en compte de contrastes de vitesse importants. La grille en quinconces proposée par Madariaga et Virieux [108, 153] et étendue à l'ordre 4 par Levander [97] repose sur l'évaluation des dérivées spatiales ou temporelles avec un décalage d'un demi pas en espace et en temps. Les qualités de ce schéma en terme de dispersion numérique et de prise en compte des hétérogénéités en font aujourd'hui un standard pour la modélisation élastodynamique. Dans le cas d'une approximation spatiale d'ordre 4, il est classique d'utiliser 5 points de grille par longueur d'onde. Une étude détaillée de ce dernier point est disponible dans [43].

La figure 1.4 illustre la grille de discrétisation en quinconces avec le décalage en temps et en espace du calcul des dérivées. Les indices en exposant indiquent les directions spatiales ($\sigma^{ijk} = \sigma(i\Delta s, j\Delta s, k\Delta s)$. Les constantes élastiques ρ, μ et λ sont définies aux points (i, j, k).
En se plaçant au pas de temps $t = \ell\Delta t$, il faut donc déterminer les variables suivantes au pas de temps $t = \left(\ell + \frac{1}{2}\right)\Delta t$:

$$v_x^{(i+\frac{1}{2})jk}\left(l+\frac{1}{2}\right), \qquad v_y^{i(j+\frac{1}{2})k}\left(l+\frac{1}{2}\right), \qquad v_z^{ij(k+\frac{1}{2})}\left(l+\frac{1}{2}\right) \tag{1.14}$$

puis au pas de temps $t = (\ell + 1)\Delta t$, les variables suivantes :

$$\begin{gathered} \sigma_{xx}^{ijk}\left(l+1\right), \qquad \sigma_{yy}^{ijk}\left(l+1\right), \qquad \sigma_{zz}^{ijk}\left(l+1\right) \\ \sigma_{xy}^{(i+\frac{1}{2})(j+\frac{1}{2})k}\left(l+1\right), \qquad \sigma_{zy}^{i(j+\frac{1}{2})(k+\frac{1}{2})}\left(l+1\right), \qquad \sigma_{zx}^{(i+\frac{1}{2})j(k+\frac{1}{2})}\left(l+1\right) \end{gathered} \tag{1.15}$$

Classiquement, une approximation du second ordre est utilisée pour la dérivée en temps. La précision obtenue est suffisante pour le problème considéré. De plus, cette approche ne requiert pas le stockage de l'historique des vitesses et des contraintes aux pas de temps précédents. Ces valeurs seraient nécessaires dans le cas de schéma d'ordre plus élevé. Pour chaque point de la grille de discrétisation, il faut alors allouer un minimum de 12 tableaux tridimensionnels afin de stocker les paramètres élastiques (ρ, λ et μ), les 3 composantes de vitesse (v_x, v_y, v_z) ainsi que les 6 composantes de contraintes (σ_{xx}, σ_{yy}, σ_{zz},σ_{xy}, σ_{yz}, σ_{xz}). Le détail des équations est donné dans l'annexe A.

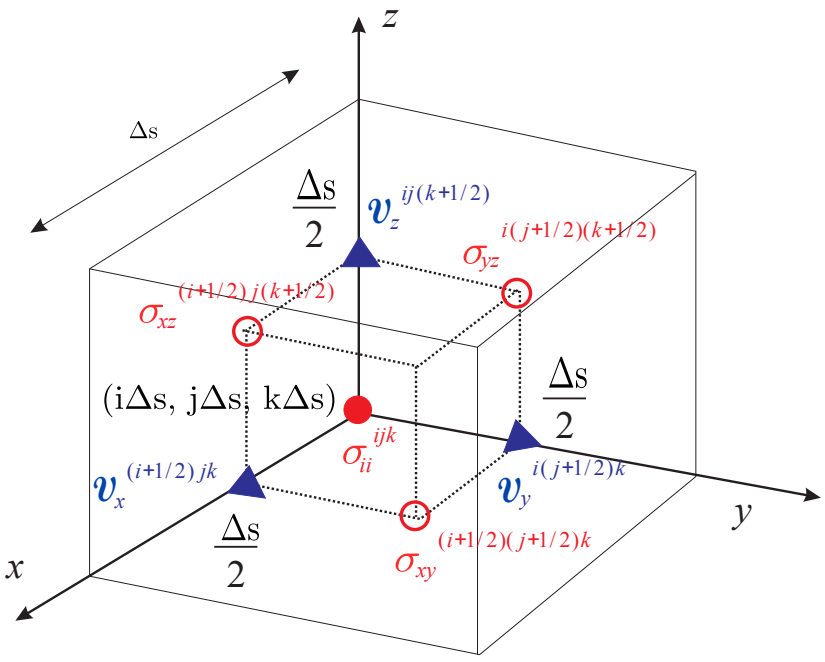

FIGURE 1.4 – Illustration de la grille de discrétisation en quinconces.

Eléments d'implémentation et parallélisme de la méthode

La figure 1.5 décrit la structure générale d'un code de calcul implémentant le schéma de différences finies introduit précédemment. Ce dernier conduit à une organisation en deux phases distinctes de calcul dédiées à l'évaluation des champs de vitesse puis de contraintes. Chaque phase de calcul se compose d'une triple boucle imbriquée balayant les trois directions d'espace (nx, ny, nz définissant le nombre de points de discrétisation dans chaque direction). Les phases d'initialisation et d'allocation sont construites suivant la même logique. Le code ONDES3D utilise cette organisation

Les références classiques d'implémentations sur architectures parallèles [26, 39, 117] reposent sur la librairie MPI [67] avec une décomposition régulière du domaine de calcul. La figure 1.6 illustre en 2D, un partitionnement horizontal en deux sous-domaines. A l'ordre 4, l'implémentation parallèle nécessite l'échange de deux mailles de recouvrement entre sous-domaines adjacents, cette valeur étant fonction de l'ordre du schéma numérique. Les valeurs des champs de contraintes et de vitesse sont donc échangées aux interfaces entre chaque sous-domaine. Pour un problème tridimensionnel, le choix du nombre de directions à partitionner dépend du rapport volume/surface permettant de recouvrir efficacement le temps de communication par du calcul en utilisant par exemple des communications non-bloquantes. Le calcul du rapport calcul/communication dans un tel cas constitue un résultat classique du parallélisme. Le tableau 1.1 résume les performances potentielles du recouvrement dans le cas d'un domaine de taille arbitraire V= $nx \times ny \times nz$ avec $nx = ny = nz$ et P processeurs. La décomposition utilisant les trois directions d'espace offre de meilleures possibilités de recouvrement.

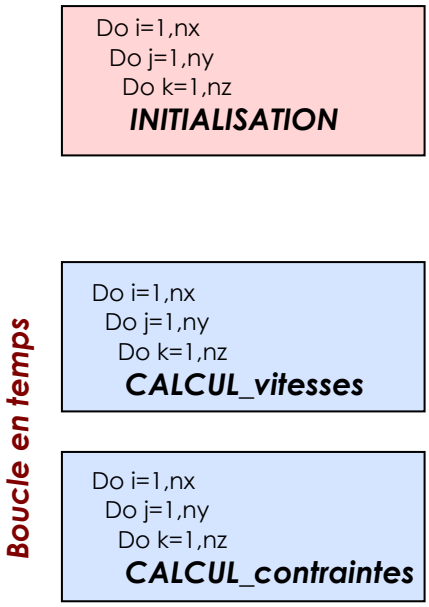

FIGURE 1.5 – Structure générale du code ONDES3D implémentant un schéma de différences finies. Les phases de calcul (en bleu) et la phase d'initialisation (en rouge) conduisent à accès similaires aux données.

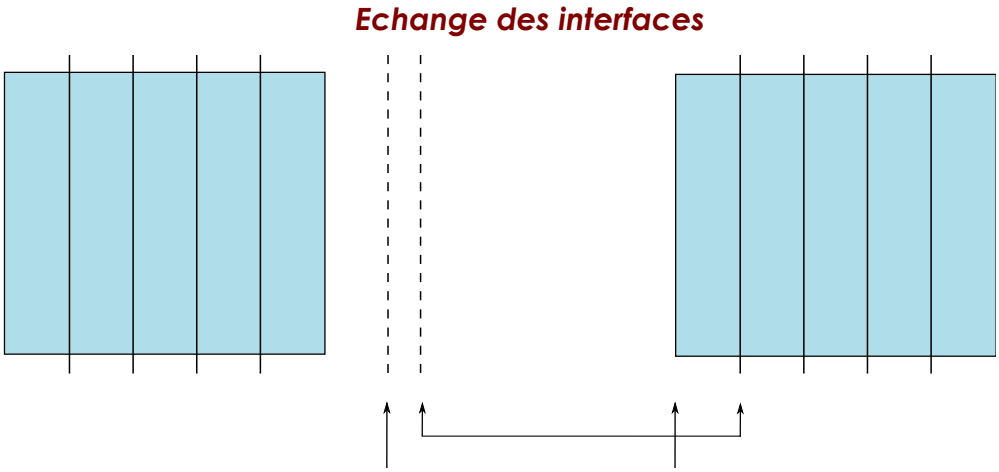

FIGURE 1.6 – Echange des interfaces entre sous-domaines dans le cas de la méthode des différences finies. Deux interfaces doivent-être échangées à l'ordre 4.

1.2.2 Méthode des éléments finis

Dans le cadre de l'étude des risques sismiques, la modélisation géomécanique intervient à une échelle plus locale. Elle permet de quantifier les effets de site non linéaires et la méthode des éléments finis est un bon candidat pour sa mise en œuvre. Il s'agit de l'approche numérique la plus répandue en mécanique des sols. Nous choisissons de souligner les étapes clés de son implémentation. La rédaction de cette section repose sur quelques références classiques qui pourront être consultées [49, 157].

Type de partitionnement	Ratio calcul/communication
Découpage 1D	$\frac{V}{P}$
Découpage 2D	$\frac{V}{\sqrt{P}}$
Découpage 3D	$\frac{V}{\sqrt[3]{P}}$

TABLE 1.1 – Performances théoriques du recouvrement des communications par du calcul en fonction de la stratégie de découpage, V représente le volume du domaine et P le nombre de processeurs.

Formulation variationnelle

La formulation faible ou variationnelle est à la base des méthodes d'éléments finis. Il s'agit de reformuler le problème initial (forme forte) en un problème équivalent (forme faible) avec une solution exprimée dans un autre espace de fonctions. En s'appuyant sur certains résultats d'analyse fonctionnelle, elle permet notamment d'affirmer l'existence et l'unicité de la solution du problème. La formulation obtenue rejoint celle issue du principe des puissances virtuelles en mécanique. L'équation initiale est multipliée par un ensemble de fonctions test $\boldsymbol{v}$ appartenant à l'espace de Sobolev $H^1(\Omega)$.
En partant de l'équation fondamentale 1.1, on cherche $u \in H^1(\Omega)$ tel que :

$$\int_\Omega \rho \frac{\partial^2 u}{\partial t^2} \boldsymbol{v} dV = \int_\Omega \sigma : \nabla \boldsymbol{v} dV + \int_\Omega f \boldsymbol{v} dV - \int_\Gamma (\sigma.n) \boldsymbol{v} d\Gamma \qquad \forall \boldsymbol{v} \in H^1(\Omega) \tag{1.16}$$

- Ω représente le domaine de calcul et Γ sa frontière
- n est le vecteur unité normal au bord Γ du domaine Ω
- $H^1(\Omega) = \{\boldsymbol{v} \in L^2(\Omega), \frac{\partial \boldsymbol{v}}{\partial x_i} \in L^2(\Omega), \forall i = 1, .., n\}$

Principales étapes de la mise en œuvre

En partant de la formulation faible, il s'agit d'évaluer les intégrales précédentes. Celles-ci sont décomposées en somme d'intégrales sur des volumes plus petits (éléments). En pratique, la méthode des éléments finis repose sur les étapes suivantes :

- La première phase consiste à mailler géométriquement le domaine initial en un ensemble d'éléments. Cette décomposition repose sur un ensemble de formes géométriques de base (en 3D des tétraèdres, hexaèdres ou prismes par exemple). Le maillage ainsi défini est composé d'un ensemble d'éléments et de nœuds. Ces derniers doivent respecter un ensemble de règles pour être conformes. Typiquement, deux éléments contigus doivent se raccorder sans recouvrement. De plus, tout sommet d'un élément doit constituer un nœud du maillage et les nœuds situés sur une interface commune sont confondus. Des logiciels dédiés (mailleur) sont utilisés afin de discrétiser la géométrie initiale du problème. L'erreur entre le domaine de calcul initial et sa représentation discrète (erreur de discrétisation géométrique) doit être minimisée en adaptant la forme et/ou le type d'élément utilisé. Notons que la forme des éléments (par exemple trop plat) pourra avoir un impact négatif sur la difficulté numérique du problème à résoudre et sur la qualité de l'approximation.

- La deuxième étape consiste à définir une approximation nodale par sous-domaine. A partir de la définition de chaque élément, il s'agit de choisir et de construire les fonctions d'interpolation $N_i(x)$ de manière à exprimer $u(x) = \sum u_i N_i(x)$. Cette approximation doit respecter certaines règles de continuité sur un même élément et entre sous-domaines. Le scalaire u_i exprime la valeur de $u(x)$ aux nœuds de l'élément. En pratique, le calcul s'effectue sur un élément de référence de dimension et de forme régulière. Le passage de l'élément de référence à un élément quelconque du domaine se fait par une transformation géométrique appelée "transformation jacobienne".

- A partir de l'approximation nodale et du maillage du domaine, la forme intégrale globale est remplacée par une somme d'intégrales locales sur chaque élément. Les matrices de rigidité élémentaires (K_e) obtenues à partir des intégrales locales traduisent le comportement mécanique de chaque élément. L'expression matricielle globale du problème est obtenue en utilisant les informations de connectivité du maillage afin de sommer les contributions élémentaires. Plusieurs éléments géométriquement connectés devront sommer leurs contributions (phase d'assemblage). La stratégie est identique pour les forces externes.

Ces phases standard de la méthode des éléments finis se retrouvent quelle que soit la technique numérique choisie pour la résolution du problème discret.

Procédure numérique et parallélisme

L'application de la méthode des éléments finis à l'analyse de problèmes incluant des lois de comportement complexes nécessite une attention particulière pour les points suivants :

- la relation non linéaire entre les contraintes et les déformations ;
- la résolution numérique du problème global reliant les déplacements et le chargement.

Au cours de la procédure de calcul, les forces sont appliquées de manière incrémentale afin d'éviter une déviation brutale de la solution du problème. L'incrément de déplacement correspondant est déduit à partir de la résolution du problème global. A partir du déplacement, l'incrément de déformation est calculé à partir de la relation non linéaire entre les contraintes et les déformations. Dans le cas plastique et sous forme incrémentale, il est nécessaire de résoudre un système d'équations différentielles défini par :

$$\dot{\sigma} = D_{ep}\ \dot{\epsilon} \tag{1.17}$$

avec D_{ep} la matrice élastoplastique et $\dot{\epsilon}$, $\dot{\sigma}$ les incréments de déformations et contraintes respectivement.
La résolution fréquente de ces équations au niveau de chaque élément et à chaque étape implique l'implémentation de techniques numériques performantes. Les schémas d'Euler implicites ou explicites sont couramment utilisés à ce niveau [1].

Au niveau global, les schémas numériques explicites pour la résolution du problème non linéaire reposent sur des phases coûteuses de sélection du pas de temps et de contrôle de l'erreur [1, 104]. Les schémas implicites sont alors privilégiés, ces derniers étant généralement couplés à une procédure itérative de type Newton permettant de linéariser le problème. La procédure en

temps repose sur la méthode de Newmark et s'écrit :

$$\begin{cases} U_{n+1} &= U_n + \Delta t \dot{U}_n + \frac{\Delta t^2}{2}[(1-2\beta)\ddot{U}_n + 2\beta\ddot{U}_{n+1}] \\ \dot{U}_{n+1} &= \dot{U}_n + \Delta t[(1-\gamma)\ddot{U}_n + \gamma\ddot{U}_{n+1}] \end{cases} \tag{1.18}$$

γ et β sont les paramètres de Newmark qui déterminent la stabilité et la précision du schéma, ils sont souvent choisis égaux à 0.5 car cela conduit à un schéma inconditionnellement stable.

La phase de linéarisation de Newton peut être décrite à partir du système suivant :

$$\dot{U} = [K_{ep}(U)]^{-1} F^{ext} \tag{1.19}$$

avec U le vecteur déplacement, $K_{ep} = \int B^T D_{ep} B dV$ la matrice de rigidité tangente (B étant la matrice reliant les déformations et les déplacements) et F^{ext} le vecteur des forces externes. La procédure de Newton standard s'écrit :

1. $R^{i-1} = F^{ext} - F^{int}$ avec $F^{int} = \int B^T \sigma^{i-1} dV$
2. $\delta U = [K_{ep}(U^{i-1})]^{-1} R^{i-1}$
3. $U_i = U_{i-1} + \delta U$.

Il s'agit de calculer le résidu (R) à partir des forces externes (F^{ext}) et internes (F^{int}). L'incrément de déplacement (δU) est calculé ensuite en résolvant le système linéaire basé sur $K_{ep}(U)$. La procédure est arrêtée lorsque le résidu atteint un certain seuil. Le critère $\|R\|_2 \leq Itol \|F^{ext}\|_2$ est généralement appliqué avec $Itol$ compris entre 10^{-3} et 10^{-6}.

L'approche *initial stress* [158] propose de remplacer K_{ep} par $K_e = \int B^T D_e B dV$. Il s'agit en fait d'utiliser la matrice D_e de rigidité élastique. L'avantage est d'éviter le calcul et la résolution du système linéaire lié à la matrice de rigidité tangente à chaque étape. Cette stratégie permet d'exploiter avantageusement les propriétés d'un schéma de type Newton-Raphson modifié et de réutiliser un grand nombre de fois la matrice initialement calculée. De plus, en cas de non linéarité sévère, la matrice de rigidité peut perdre son caractère symétrique. La possibilité de reformer occasionnellement cette matrice constitue donc un avantage malgré le taux de convergence inférieur par rapport à l'approche initiale. Cette dernière approche est choisie pour le logiciel *GEFDYN*. La procédure de calcul non linéaire repose donc principalement sur l'assemblage des vecteurs force et peut être considérée comme *pseudo-explicite*.

La figure 1.7 décrit la procédure numérique globale. La résolution du problème s'effectue en trois phases.

- La première étape de prétraitement correspond à l'initialisation des données et à la définition des conditions aux limites. Dans le cas d'une implémentation sur architecture parallèle, cette phase inclut le partitionnement du maillage en sous-domaines équilibrés qui vont être distribués sur les différents processeurs en essayant de minimiser le nombre d'interfaces entre sous-domaines. Des bibliothèques telles que METIS ou SCOTCH [81, 129] sont utilisées pour cela.
- La deuxième phase est la mise à jour au cours du temps de l'incrément de déplacement à partir de la matrice de rigidité tangente. Les forces extérieures sont également calculées à ce niveau.

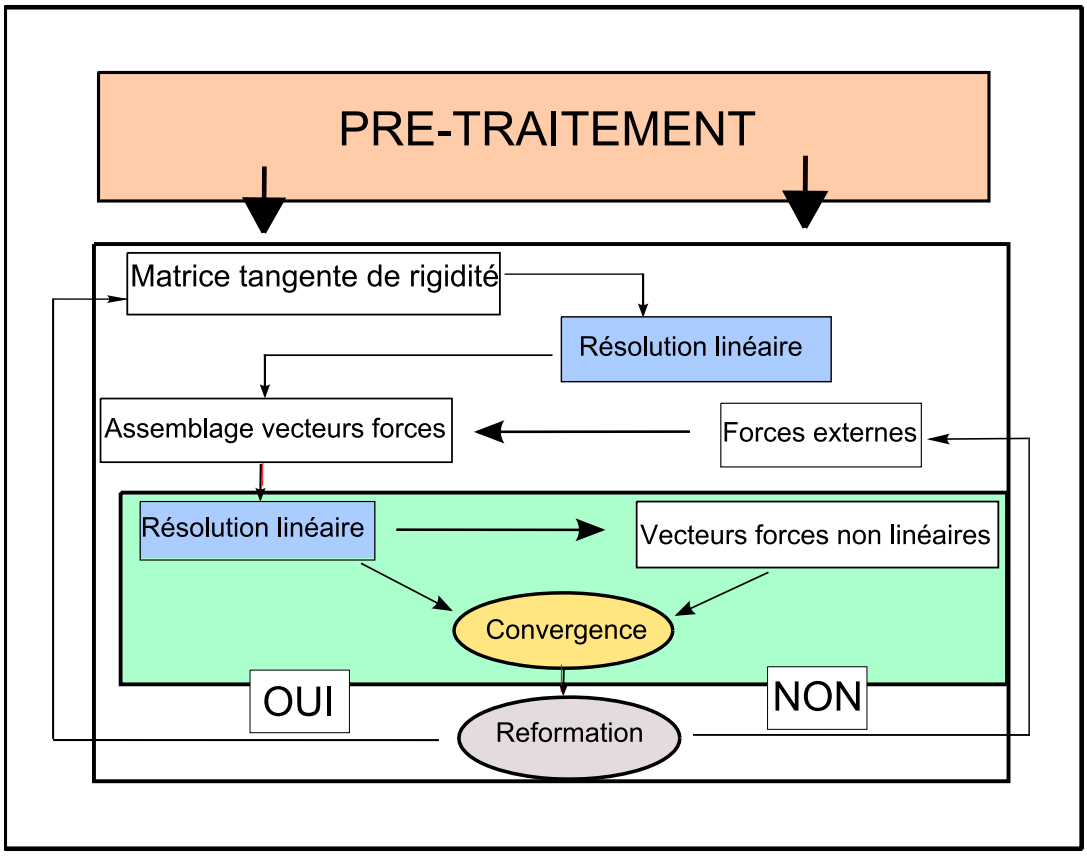

FIGURE 1.7 – Procédure numérique utilisée pour la mise en œuvre parallèle de la méthode des éléments finis. Les calculs itératifs de la procédure de Newton-Raphson sont matérialisés en vert. En bleu, on indique les opérations nécessitant des résolutions matricielles utilisant des techniques d'algèbre linéaire creuse.

– La dernière étape implémente la procédure itérative de Newton décrite précédemment.

Dans la mise en œuvre sur architecture parallèle deux types de calcul vont influencer les performances globales de l'application :

– l'assemblage des contributions qui nécessite un contrôle fin de la distribution des éléments sur les différents processeurs pour l'évaluation des lois de comportement non linéaires ;
– les calculs matriciels intervenant à différents niveaux de l'algorithme ; le parallélisme qui pourra en être extrait impactera fortement le comportement parallèle de l'ensemble de l'application.

1.2.3 Conditions aux limites

Conditions absorbantes

La modélisation de la propagation des ondes sismiques repose sur la définition d'un domaine borné alors que le phénomène physique simulé suppose un espace infini. Des conditions aux limites spécifiques sont alors implémentées afin d'éviter la réflexion des ondes vers l'intérieur du domaine physique. Il s'agit généralement d'un ensemble d'équations non physiques introduites aux bords du domaine et définissant une couche dite absorbante. Le nombre de points de grille dans cette zone (épaisseur de la couche) varie en fonction de la technique numérique implémentée. Différentes techniques numériques ont été décrites dans la littérature [36, 74]. Malheureusement, leurs performances se dégradent sensiblement en fonction de l'angle d'incidence ou de la fréquence des ondes conduisant à la réflexion d'une part importante de l'énergie vers le domaine physique.

Les conditions *PML* (Perfectly Matched Layer), introduites par Bérenger [19] pour l'électromagnétisme et étendues par Collino et Tsogka [42] pour l'élastodynamique (en formulation vitesses-contraintes), présentent la particularité d'une réflexion nulle pour tous les angles d'incidence et toutes les fréquences avant discrétisation par un schéma numérique. De manière résumé, il s'agit d'introduire dans le système une fonction d'amortissement conduisant à une décroissance exponentielle des ondes aux bords du domaine physique. Les équations modifiées sont celles comportant des dérivées normales par rapport aux frontières. Pour un domaine cubique, le nombre de composantes à amortir varie en fonction du nombre de bords extérieurs artificiels (1, 2 ou 3 bords extérieurs pour un volume donné). En 2D, avec une couche absorbante placée en $x = 0$, la fonction d d'amortissement s'introduit de la manière suivante avec $v = v^1 + v^2$:

$$\begin{cases} \rho(\frac{\partial}{\partial t} + d_x)v_x^1 = \frac{\partial}{\partial x}\sigma_{xx} \\ \rho(\frac{\partial}{\partial t})v_x^2 = \frac{\partial}{\partial y}\sigma_{xy} \\ \rho(\frac{\partial}{\partial t} + d_x)v_y^1 = \frac{\partial}{\partial x}\sigma_{xy} \\ \rho(\frac{\partial}{\partial t})v_y^2 = \frac{\partial}{\partial y}\sigma_{yy} \end{cases} \tag{1.20}$$

La procédure est identique pour les contraintes avec $\sigma = \sigma^1 + \sigma^2$. Généralement, l'épaisseur de la couche absorbante est de dix points de grille dans les trois directions d'espace.

Les *CPML* (convolution-*PML* [90]) ou *ADE-PML* (Auxiliary Differential Equation-*PML*) [90] reposent sur l'introduction d'inconnues auxiliaires au profil d'amortissement d. Cette technique permet d'améliorer les performances des *PML* classiques, principalement pour les ondes à incidence rasante. Le détail de l'implémentation de cette approche dans le code ONDES3D ainsi que les résultats obtenus sont décrits dans [51].

Les méthodes décrites précédemment peuvent-être implémentées pour une formulation déplacements-contraintes ou vitesse-contraintes dans le cas de la méthode des éléments finis. Les *PML* et les *CPML* ont été adaptées à cette situation [92, 112]. Dans le cadre des simulations effectuées avec le code GEFDYN, l'approximation paraxiale est utilisée [53, 121] notamment pour sa simplicité d'implémentation. De manière simplifiée, les contraintes (notée τ) induites par une onde au contact d'une interface sont atténuées par

$$\tau(x_1, x_2, x_3, t) = \begin{pmatrix} -\rho\beta\partial_t u_1 \\ -\rho\beta\partial_t u_2 \\ -\rho\alpha\partial_t u_3 \end{pmatrix} \tag{1.21}$$

avec (x_1, x_2, x_3) les coordonnées locales de l'élément paraxial et α, β respectivement la vitesse des ondes P et S.

Conditions de surface libre

L'enregistrement des ondes sismiques est généralement effectuée au niveau de l'interface terre-air, renforçant la nécessité de précision. La condition limite à la surface de la Terre peut s'exprimer de la manière suivante. En supposant que la surface correspond à $z = 0$, le domaine physique à $z < 0$ et que l'air est remplacé par une couche de vide ; les composantes du tenseur des contraintes dans la direction verticale doivent être nulles (condition de traction nulle dite de surface libre) :

$$\sigma_{zz}(x, y, 0) = \sigma_{xz}(x, y, 0) = \sigma_{yz}(x, y, 0) = 0 \tag{1.22}$$

Différentes techniques numériques ont été proposées dans la littérature afin d'implémenter la condition de surface libre pour la résolution de l'équation de l'élastodynamique par la méthode des différences finies. Un état de l'art des approches proposées peut être trouvé dans [43, 146]. On peut par exemple citer l'approximation de la topographie par une discrétisation en marches d'escalier proposée par Ohminato et Chouet [125] mais sa précision est insuffisante. Une autre piste consiste à adapter le maillage à la topographie puis à exprimer la condition de surface libre dans un référentiel curviligne [73, 146]. L'expression des dérivées aux alentours de la surface libre nécessite alors certaines adaptations numériques (filtre numérique, rotation de la grille en quinconces, formulation décentrée des différences finies). Aucune solution réellement satisfaisante n'est encore disponible afin de prendre en compte les topographies complexes, et certaines propositions récentes en 2D sont probablement à approfondir [100].

Une approche classique est la "méthode des images" [97, 136] appliquée à la grille de discrétisation en quinconces. Elle impose les conditions suivantes :

$$\sigma_{zz}(x, y, z) = -\sigma_{zz}(x, y, -z) \tag{1.23}$$
$$\sigma_{xz}(x, y, z) = -\sigma_{xz}(x, y, -z) \tag{1.24}$$
$$\sigma_{yz}(x, y, z) = -\sigma_{yz}(x, y, -z). \tag{1.25}$$

Ces dernières traduisent l'application de valeurs antisymétriques pour les nœuds au dessus de la surface libre (par rapport aux nœuds miroir situés dans le domaine physique). Une des limitations de cette technique est son manque de précision pour les ondes de surface [120]. En revanche, la simplicité de son implémentation permet une intégration aisée au niveau de l'application Ondes3D.

1.3 Exploitation du parallélisme

La capacité d'exploitation de machines avec plusieurs milliers de processeurs a été démontrée avec les méthodes différences finies [39, 44, 61] ou éléments finis [5, 88]. Le parallélisme est alors basé sur une décomposition spatiale du domaine de calcul et sur l'échange des informations nodales aux interfaces des sous-domaines de calcul. En fonction des caractéristiques

du maillage (approches structurées ou non structurées), la décomposition adoptée repose sur un simple découpage suivant les directions horizontales et verticales ou encore sur l'utilisation d'outils spécifiques de partitionnement [81, 129]. Néanmoins, les références classiques citées sont majoritairement basées sur la librairie MPI et n'exploitent que très partiellement les opportunités offertes par les nouvelles architectures de calcul. De manière symétrique, les difficultés de programmation sont masquées, le programmeur confondant souvent portage de son application et exploitation réelle et efficace de l'architecture sous-jacente.

1.3.1 Evolution des architectures de calcul

Deux évolutions importantes conduisent aujourd'hui à l'exploitation quotidienne d'architectures de calcul présentant un important niveau de complexité. La première est le retour des architectures NUMA (Non Uniform Memory Access) qui sont venues combler les limites en termes de contention sur le bus mémoire des architectures SMP (Symmetric MultiProcessing). L'éclatement de la mémoire physique en plusieurs sous-blocs rattachés aux différents nœuds NUMA nécessite alors des mécanismes performants d'accès aux données distantes. Le surcoût induit par ce type d'accès est appelé *facteur NUMA* ; il varie en fonction des architectures considérées, du type d'application ou encore de la charge de la machine. La deuxième évolution est liée à l'amélioration de la finesse de gravure conduisant à l'émergence des architectures massivement multicœurs.

Une synthèse des évolutions récentes au niveau des architectures de calcul soulignerait les limites atteintes par les processeurs monocœur (x86, IA-64, x86_64), puis l'émergence et la démocratisation de puces multicœur (Intel Itanium 2 Montecito et Xeon, AMD Opteron) et enfin l'intégration d'un contrôleur mémoire au niveau des processeurs (de type QPI chez Intel avec le Nehalem ou Hypertransport chez AMD). Ce nouveau saut est à l'origine d'un niveau supplémentaire de complexité au niveau des nœuds multiprocesseurs de plus en plus hiérarchique au niveau de l'organisation de la mémoire.

Une autre façon de mesurer les changements apparus ces dernières années est de garder un angle de vue fixe afin d'observer les évolutions successives. Le programme de simulation de la direction des applications militaires (DAM) du CEA est un exemple intéressant. Ce dernier est un maillon important de la capacité de dissuasion nucléaire française ; il est issu de la fin des essais nucléaires et doit permettre aux physiciens de remplacer l'expérimentation par la simulation. Les enjeux sont donc considérables et les besoins en architectures de calcul de pointe est crucial. Le programme *TERA* a démarré en 1996 pour 15 ans, il vise la mise en place de supercalculateurs de puissance croissante.

Le premier déploiement effectué en 2002 correspond à la machine *TERA-1* qui permettait d'atteindre 5 teraflops de puissance crête. La brique de base est constituée par une architecture à mémoire partagé composée de quatre processeurs EV68 (Digital) cadencés à 1 gigahertz. La puissance offerte est de 8 gigaflops par nœud multiprocesseur. Le système complet fourni par HP est composé de 640 nœuds interconnectés par un réseau rapide provenant de la société Quadrics. Si les chiffres présentés peuvent paraître faibles aujourd'hui, il s'agissait de la machine la plus puissante d'Europe classée 4e au Top 500 en Juin 2002.
Sur le plan de l'organisation de la mémoire, *TERA-1* est en fait une architecture *SMP* qui correspond bien à la tendance de l'époque. La courbe de croissance des architectures *Beowulf* est alors maximale avec un taux d'équipement élevé des laboratoires et des organismes de recherche de taille moyenne en clusters basées sur des composants *off the shelf*. On parle alors de *commodity*

computing avec des architectures faiblement standardisées mais principalement basées sur des nœuds de calcul bi-processeurs. La figure 1.8 illustre l'organisation d'un nœud de calcul pour

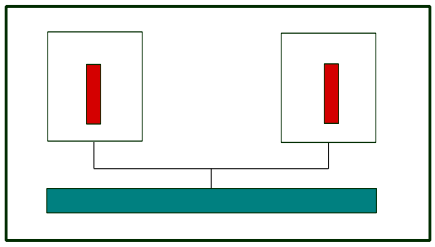

FIGURE 1.8 – Exemple d'architecture *SMP* bi-processeurs

une architecture *SMP* bi-processeurs. L'accès à la mémoire centrale est partagée de manière uniforme entre les différents processeurs. Le lien entre les processeurs se fait via un bus ou un commutateur (comme dans le cas des serveurs ES45 de *TERA-1*).

La mise en œuvre de la seconde tranche *TERA-10* illustre la nouvelle tendance qui se dégage à partir du milieu des années 2000. L'architecture fournie par BULL en 2006 repose également sur une organisation en gros nœuds de calcul à mémoire partagée, l'objectif étant d'atteindre les 60 teraflops de puissance crête. La QBB (Quad Brick Block) constitue l'unité de base et permet de connecter quatre processeurs dual-cœurs de type Intel Itanium 2 Montecito. Chaque nœud de calcul est composée de 4 QBB proposant 8 processeurs au total dans la configuration retenue . L'interconnexion entre QBB repose sur de la technologie Fame Scalability Switch conduisant à une architecture NUMA.
La première rupture avec *TERA-1* provient de l'introduction de processeurs multicœurs. Les différentes stratégies de conception de ces puces (Niagara 2 de SUN ou Power 4 d'IBM) posent néanmoins des questions communes en terme de partage des différents niveaux de mémoire cache ou d'exploitation de la bande passante mémoire. La figure 1.9 présente une architecture

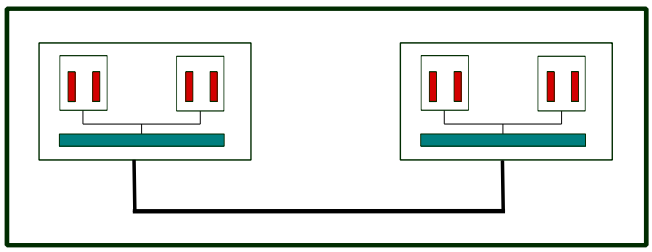

FIGURE 1.9 – Exemple d'architecture NUMA bi-processeurs dual-cœurs

bi-processeurs dual-cœurs. La mémoire est partagée de manière symétrique entre processeurs au même niveau, en revanche le coût d'un accès à la mémoire distante va être supérieur.
Plus largement, le parallélisme au niveau des puces est renforcé par l'exploitation du Simultaneaous Multithreading (SMT) donnant la vision de plusieurs processeurs virtuels qui peuvent

être a priori exploités de manière optimale en fonction de l'occupation des unités de calcul. Les résultats dans le domaine du calcul scientifique sont généralement assez décevants [149].
Le second point est l'utilisation d'une technologie d'interconnexion menant à une organisation hiérarchique de la mémoire. Cette tendance se retrouve également chez SGI avec la réapparition d'approches éprouvées dans le passé avec la série des calculateurs Altix (après les machines SGI Origin 2000 de la fin des années 90). Typiquement, le facteur NUMA est compris entre 2 et 2.5 entre deux nœuds de l'architecture Bull Novascale. L'écart dans le cas d'une implémentation séquentielle de la méthode des différences finies est proche de 30% entre un placement local des données et un placement sur un nœud distant.

La tranche la plus récente du programme de simulation, *TERA-100*, a été mise en production au cours de l'année 2010. Elle est composée de 4300 nœuds de type Bull et la puissance atteinte est de 1.25 petaflops. Chaque nœud accueille quatre processeurs octo-cœurs de la série Intel Xeon 7500. De plus, le système accueillera des *fat nodes* construits en agrégeant plusieurs nœuds via la technologie d'interconnexion BCS (Bull Coherence System). Parmi les évolutions importantes, on peut souligner l'introduction d'un contrôleur mémoire au niveau des processeurs, la mémoire étant désormais directement attachée à ce niveau. L'architecture présente donc différents niveaux de hiérarchie mémoire (entre processeurs multicœurs, entre nœuds multiprocesseurs) en plus de la hiérarchie classique des niveaux de cache (Figure 1.10).

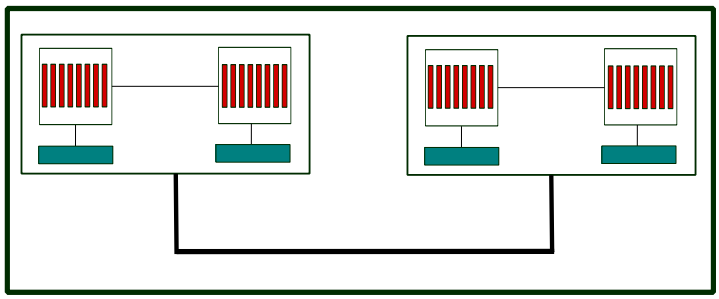

FIGURE 1.10 – Exemple d'architecture NUMA bi-processeurs octo-cœurs

1.3.2 Expression du parallélisme

Le modèle de programmation à plat par échange de messages (par exemple via la librairie MPI) ne parait pas le mieux adapté pour l'exploitation efficace des grappes de calcul composées de gros nœuds à mémoire partagée. En effet, la librairie utilisée doit être optimisée pour l'architecture sous-jacente en utilisant des mécanismes adaptés à la fois pour les communications inter-nœuds et intra-nœud. Si les implémentations MPI propriétaires fournissent des librairies optimisées pour les architectures cibles, les implémentations standard telles que *MPICH* [67] ou *Open MPI* [62] ne proposent pas le même niveau de performance en mode hybride. Ceci est d'autant plus critique dans un environnement massivement multicœurs qui nécessite des stratégies avancées de placement des processus afin de maximiser les performances [115]. Les évolutions les plus importantes concernant les nœuds de calcul multiprocesseurs, la stratégie généralement

retenue est de combiner l'approche par échange de messages entre nœuds multiprocesseurs à un autre modèle de programmation mieux adapté au parallélisme à grain fin à l'intérieur des nœuds. Les librairies et environnements spécifiquement évalués dans cette thèse sont indiqués en gras.

Multithreading

Les processus légers ou *threads* offrent une grande flexibilité pour l'expression du parallélisme, en particulier dans le cas d'algorithmes irréguliers sur des nœuds de calcul multicœurs. Le programmeur peut ainsi décrire de manière précise son algorithme et exploiter un niveau de parallélisme à grain fin. En effet, en fonction du modèle de threads choisi le coût de management et d'ordonnancement des threads peut être faible, permettant ainsi l'introduction d'un multithreading massif (technique de surcharge ou virtualisation avec l'ordonnancement de plus de threads que de processeurs disponibles). C'est le cas de la bibliothèque de threads **Marcel** [45, 123] qui propose un ordonnancement mixte. Les threads noyaux sont fixés aux cœurs et les threads de niveau utilisateur effectuent ainsi des opérations de changement de contexte rendues peu coûteuses par le contournement des appels systèmes.

Une voie d'exploitation du multithreading est proposée par des environnements tels que **MPC** (Multiprocessor Communications [130]) ou AMPI (Adaptive-MPI [24]). Ces derniers proposent de reproduire le modèle d'exécution MPI en exploitant des bibliothèques de threads optimisées pour la gestion d'un parallélisme de données. AMPI est ainsi bâti au dessus de CHARM++ [79] qui fournit de plus des mécanismes de sauvegarde/reprise et de migration des threads pour l'équilibrage de charges. La bibliothèque MPC repose sur un ordonnanceur de threads mixte particulièrement optimisé pour les communications collectives et le placement des threads et l'allocation mémoire sur architectures hiérarchiques. Ces deux environnements offrent des possibilités d'équilibrage de charge exploitant la virtualisation des cœurs. Dans le cas de la librairie AMPI, les informations provenant de l'application au cours de l'exécution (déséquilibre de charge, schéma de communications) sont utilisées afin d'optimiser la répartition des tâches. La prise en compte du déséquilibre de charge s'effectue au niveau applicatif pour l'environnement MPC.

Langages et extensions de langages

La proposition et l'extension de langages parallèles efficaces est un domaine de recherche particulièrement actif. Cette activité est motivée par la complexité croissante des architectures et la volonté de permettre au programmeur de se concentrer uniquement sur les aspects algorithmiques. On peut mentionner le modèle de programmation *PGAS* (Partitioned Global Adress Space) qui s'appuie sur une vision *DSM* (Distributed Shared Memory) de l'architecture. Les approches proposées par *Chapel*, *Fortress*, *Co-Array Fortran* ou *Unified Parallel C (UPC)* visent prioritairement la simplification de l'expression du parallélisme [105]. La difficulté pour le programmeur est alors de modifier (ou plus probablement de réécrire) son application à partir de ces nouvelles propositions. Le niveau de performance atteint est variable en fonction du contexte applicatif, des compilateurs et des architectures. Les benchmarks effectués dans le cadre du projet PRACE[4] (Partnership for Advanced Computing in Europe) ont montré que UPC et Co-Array Fortran n'atteignait que 10% de la puissance crête dans le cas d'un produit matrice-matrice dense

4. www.prace-project.eu

ou d'un produit matrice-vecteur creux.

De même, Cilk [25] et la librairie Threading Building Blocks [133] proposés par Intel suggèrent des extensions des langages C et C++. Ils offrent une abstraction des détails de l'architecture et du mécanisme de multithreading avec pour objectif un confort d'implémentation pour le programmeur.

OpenMP demeure l'approche la plus courante pour l'exploitation des nœuds à mémoire partagée [38]. Par un ensemble de directives, OpenMP propose les outils pour créer de façon simple des régions parallèles dans un code séquentiel. L'expression du parallélisme est particulièrement adaptée aux applications construites à partir de nids de boucle. Le standard permet également la mise en œuvre de régions parallèles imbriquées. Pour chaque nouvelle région parallèle rencontrée, le thread maitre va créer une nouvelle équipe de threads et s'y intégrer, ce processus étant répété autant de fois qu'il existe de régions imbriquées. Généralement, le nombre de threads créés dans ce contexte est supérieur au nombre de threads générés dans le cas classique. Ce point dépend néanmoins de la stratégie au niveau du support d'exécution OpenMP car ce dernier peut choisir de sérialiser l'exécution des régions imbriquées.

1.3.3 Placement des données

L'impact sur les performances applicatives de la hiérarchie mémoire croissante dans les nœuds de calcul actuels est renforcé par le grand nombre de cœurs accédant à la mémoire conduisant à des problèmes de contention et de déséquilibre de charge. Maximiser *l'affinité mémoire* consiste donc à réduire la distance entre la localisation des données et les cœurs de calcul. La figure 1.11 illustre ce point avec différentes possibilités de placement des données et des threads sur une architecture NUMA comportant 4 nœuds, les couleurs indiquant les données qui sont utilisées par les threads de calcul. L'ordonnancement permettant de maximiser l'affinité mémoire consiste à placer les threads et les données de couleur identique sur le même nœud NUMA.

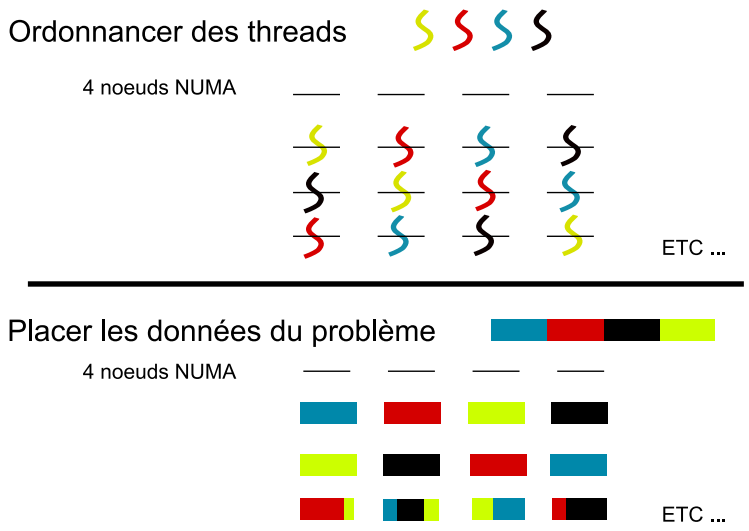

FIGURE 1.11 – Placement des threads et des données sur une machine avec quatre nœuds NUMA, les couleurs indiquent les données nécessaires aux quatre threads de calcul.

Différentes techniques ont été proposées afin d'aborder ce problème. Elles se situent à différents niveaux (système, bibliothèque au niveau utilisateur, algorithmique) avec des résultats très variables en termes de performances ou de portabilité [10, 65, 85, 109, 122]. La connaissance des caractéristiques de l'application cible constitue dans ce cas un atout important.

Approches génériques

Pour le système d'exploitation Linux, deux directions principales peuvent être identifiées afin de contrôler le placement des données. L'approche explicite permet au programmeur de gérer le placement des données de son application. Les solutions automatiques mettent en œuvre de manière systématique une politique mémoire prédéfinie.

Les techniques explicites, basées sur des appels système ((*mbind*(), *set_mempolicy*() et *get_mempolicy*()), la librairie *libnuma* ou la commande *numactl* sont relativement difficiles à exploiter pour un programmeur peu averti. Par exemple, les appels système offrent la possibilité de définir une politique mémoire spécifique pour une partie de la mémoire de l'application. Le programmeur devra alors gérer au niveau applicatif les pages mémoires et les pointeurs associés. La *libnuma* propose une mise en œuvre simplifiée des appels système précédents. Les outils disponibles avec la commande *numactl* permettent de contrôler l'ensemble de la mémoire allouée par l'application. Dans le cas de l'implémentation d'un schéma de type différences finies, il s'agira de fixer les threads de calcul lors des phases d'allocation des données puis de calcul. Néanmoins, une liste de nœuds et de CPUs/cœurs doit être précisée ce qui limite la portabilité de cette approche.

Les approches automatiques semblent plus confortables. Le système d'exploitation est alors responsable de la prise de décision en termes de placement des données. La politique mémoire par défaut du système d'exploitation Linux est *first touch*. Dans ce cas, les données sont placées sur le nœud qui y accède en premier. D'autres politiques mémoire ont été introduites afin d'améliorer l'affinité mémoire. Par exemple, l'approche *affinity-on-next-touch* effectue une migration des données à chaque fois qu'un *thread* situé sur un nœud distant y accède. Le coût de ces migrations peut devenir un goulôt d'étranglement dans le cas d'applications irrégulières.

Environnements et approches dédiés

Dans le cas d'une implémentation basée sur le standard OpenMP, l'affinité mémoire peut être contrôlée en distribuant la boucle de calcul qui correspond à l'initialisation et à l'allocation des données afin de profiter de la politique d'allocation mémoire *first touch*. Cette technique est particulièrement bien adaptée à la méthode des différences finies et consiste simplement à ajouter des directives OpenMP au premier nid de boucle de la figure 1.5. Cependant, cette solution ne garantit pas que les mêmes décisions de placement des threads soient prises entre la phase d'initialisation et les différentes phases de calcul.

L'utilisation de la plateforme **MAI** [134] est une autre voie possible afin d'optimiser l'affinité mémoire. Cette dernière permet de choisir la politique d'allocation mémoire la mieux adaptée au problème en utilisant un ensemble de fonctions spécifiques. L'avantage est de pouvoir utiliser simplement différentes politiques mémoire pour la même application. En fait, le développeur définit la stratégie qui lui semble la mieux adaptée pour chaque tableau de son application en fonction du schéma d'accès aux données ou de l'architecture sous-jacente. L'API de la plateforme MAI

évite au programmeur de manipuler des pointeurs ou des pages mémoires, mais se concentre sur l'allocation et le placement des tableaux importants de l'application. Elle repose principalement sur les appels systèmes Linux *mbind* et *mmap*. Différentes politiques d'allocation mémoire sont proposées par MAi. Elles peuvent-être divisées en trois groupes. Les stratégies ayant pour objectif de fixer la mémoire sur les différents nœuds se retrouvent dans le groupe *BIND*. La mémoire peut alors être divisée de façon linéaire ou par blocs. Les politiques du groupe *CYCLIC* visent une distribution de type *round robin* des pages ou des blocs de pages mémoire. Enfin une répartition de type aléatoire est également implémentée. La figure 1.12 illustre les politiques mémoire *bind_ all* (le programmeur définit le nœud ou l'ensemble des nœuds ciblés pour le placement des données) et *cyclic*.

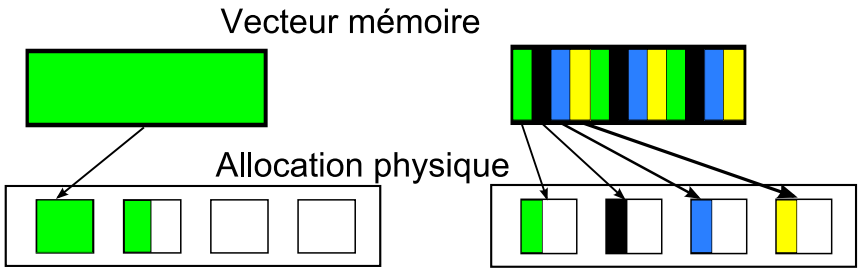

FIGURE 1.12 – Politiques mémoire *bind_ all* (gauche) et *cyclic* (droite). Les pages mémoires sont matérialisées par les zones de couleur en supposant un placement sur 4 nœuds NUMA.

La thèse de S.Thibault [149] a permis d'étendre les stratégies d'ordonnancement de la bibliothèque de threads MARCEL afin de prendre en compte les architectures hiérarchiques. Des structures récursives (*bulles*) sont utilisées afin d'exprimer l'affinité entre deux threads travaillant par exemple sur le même jeu de données. La figure 1.13 illustre la structure hiérarchique qui est induite avec l'organisation de six threads en une hiérarchie de trois bulles dont une bulle *root*. Ces dernières vont alors être ordonnancées sur la hiérarchie de l'architecture cible. La connaissance de l'algorithmique de l'application est importante et le programmeur doit explicitement exprimer la hiérarchie entre les différents threads de calcul. La plateforme **BUBBLESCHED** ainsi proposée permet au programmeur de choisir l'ordonnanceur le mieux adapté au comportement de son application. La plateforme **FORESTGOMP** [31] repose à la fois sur la bibliothèque de

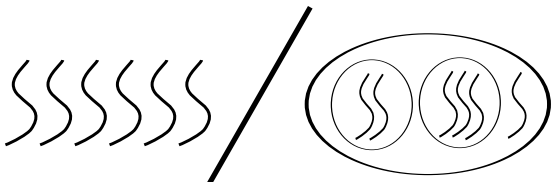

FIGURE 1.13 – Exemple d'ordonnancement par bulles de six threads en une hiérarchie de trois bulles.

threads MARCEL et sur l'environnement BUBBLESCHED. L'expression du parallélisme s'appuie sur l'utilisation de directives OpenMP. L'imbrication des sections parallèles permet la création automatique des *bulles* afin de guider l'ordonnancement des threads. Cette approche permet de

profiter de la structuration du parallélisme exprimée par le programmeur grace aux directives OpenMP. La librairie GOMP est modifiée afin de générer automatiquement les structures de *bulles* correspondantes. L'ordonnanceur intégré à la plateforme permet de diriger la répartition des threads en fonction de l'affinité mémoire et de mettre en œuvre de façon transparente les éventuelles opérations de migration de données nécessaires [65].

1.4 Impact sur la modélisation sismique

Les techniques numériques implémentées pour la modélisation sismique nécessitent donc certaines adaptations pour une mise en œuvre efficace sur architectures hiérarchiques et multicœurs. Ces adaptations doivent se faire en partie au niveau algorithmique, mais également en profitant des possibilités offertes par les supports d'exécution. Ces points sont déterminants dans l'optimisation des performances et constituent les problématiques qui seront approfondies dans cette thèse.

- L'un des changements majeurs apportés par les architectures multicœur est l'augmentation du nombre d'unités de calcul. Le programmeur doit donc pouvoir exploiter l'ensemble de ces ressources afin d'améliorer réellement les performances de son application. Cette évolution conduit à devoir considérer un parallélisme à grain fin renforçant ainsi les problématiques de communication et de synchronisation entre les différentes tâches de calcul correspondant aux sous-domaines géométriques du problème discrétisé. Ce changement induit également un déséquilibre de charge plus important (algorithmique ou provenant de la physique du problème) lié à l'exploitation d'un grand nombre de cœurs.

- De plus, l'augmentation de la hiérarchie mémoire va également impacter les performances applicatives. Ce point est relativement peu discuté dans la littérature scientifique pour des applications complexes [10, 54, 99, 147]. La prédominance du modèle de programmation MPI, notamment dans le domaine de la modélisation sismique, est probablement la principale cause du faible nombre de références analysant des applications réelles. L'adaptation nécessaire des applications aux architectures massivement multicœurs conduit à l'introduction de modèles de programmation hybride exploitant les possibilités du multithreading. L'optimisation de l'affinité mémoire devient donc un critère essentiel.

- Enfin, le niveau de performance atteint par les architectures multicœurs est principalement issu de l'augmentation du nombre d'unités de calcul. Le nombre important de défauts de cache, qui constituait classiquement un goulot d'étranglement sur processeurs monocœurs, demeure malheureusement une limitation importante. Il s'agit en fait d'un raffinement de la problématique précédente avec la nécessité de minimiser le nombre d'accès aux zones mémoire (niveaux de cache, mémoire partagée) distantes durant les phases de calcul.

1.4.1 Cas de la méthode des différences finies

Conditions aux limites absorbantes et déséquilibre de charge

La section 1.2.3.0 a soulignée l'importance des conditions aux limites absorbantes dans le cas de la modélisation sismique. La formulation ainsi introduite aux bords du domaine physique est à l'origine d'un déséquilibre de charge entre les différents sous-domaines issus de la décomposition régulière du maillage. L'écart entre l'évaluation d'un point de la grille situé à l'intérieur du

domaine physique ou dans la zone implémentant les *PML* est compris entre 1.5 et 3. Un parallélisme à grain plus fin influera donc fortement sur le comportement parallèle de l'application. En effet, l'augmentation du nombre de cœurs à exploiter conduit à une dégradation plus sensible des performances par rapport à une situation avec un nombre de processeurs plus faible offrant une puissance de calcul équivalente. Dans ce cas, le ratio points physique/points dans les bords absorbants devient plus rapidement défavorable, notamment pour les sous-domaines correspondant aux bords du domaine de calcul. Les possibilités offertes par les supports d'exécution constituent un piste intéressante afin d'implémenter de manière transparente un mécanisme dynamique de régulation de la charge.

Affinité mémoire et caractère *memory bound* du noyau de calcul

L'affinité mémoire est un point critique dans le cas de l'implémentation de la méthode des différences finies sur architectures hiérarchiques. En effet le noyau de calcul repose sur des structures de données et des types d'opérations offrant une grande régularité mais conduisant également à un nombre important de défaut de cache. Intrinsèquement, la méthode offre peu de localité mémoire. Un premier niveau d'affinité peut être assuré en guidant de manière coordonnée le placement de la mémoire et des threads de calcul sur les différents nœuds NUMA. Il s'agit d'exploiter la politique mémoire par défaut *first-Touch*. Malheureusement cette stratégie est insuffisante, elle ne permet pas l'implémentation d'algorithmes de vol de travail ou encore la réduction du nombre de défauts de cache. D'autres directions doivent donc être envisagées en offrant notamment une plus grande flexibilité en termes d'expression du parallélisme.
Un deuxième niveau de localité mémoire peut être assuré en réduisant le trafic mémoire de l'algorithme. En termes de performance crête les valeurs observées pour ce type de noyaux de calcul se situent autour de 30 % de la puissance offerte par les processeurs. Ces résultats s'expliquent par le faible nombre d'opérations effectuées à chaque point de la grille de discrétisation. Les opérations dominantes sont alors les accès mémoire qui s'effectuent principalement hors des différents niveaux de mémoire cache. Ces méthodes sont alors dites *memory bound* ce qui traduit le fait que la bande passante mémoire est le facteur limitant. L'introduction de différents niveaux de hiérarchie mémoire est un facteur aggravant car chaque accès mémoire peut alors correspondre à un accès distant. Les noyaux de calcul basés sur des stencils simples de type Jacobi sont intensivement étudiés dans la littérature scientifique. La majorité des propositions reposent sur des optimisations très spécifiques nécessitant une connaissance très fine de l'architecture ou des transformations de code effectuées lors de la phase de compilation [14, 46, 52]. De plus, les expériences menées sur architectures massivement multicœur (GPU, CELL ou Cyclops-64 [2, 46, 116, 127]) sont difficilement extensibles au cas des cœurs généralistes que nous ciblons. Les techniques utilisées sont très dépendantes des architectures cibles. Certains de ces algorithmes doivent donc être revisités afin de tenir compte des architectures sous-jacentes et de l'algorithmique du problème.

1.4.2 Cas de la méthode des éléments finis

Impact de la physique du problème

Les lois de comportement non linéaires implémentées dans le cas de la méthode des éléments finis conduisent à un écart significatif entre le coût de calcul des zones caractérisées par un comportement mécanique plastique ou élastique. La mise en œuvre de ces simulations sur architectures modernes va renforcer l'hétérogénéité de la répartition des éléments finis, le grain de calcul plus fin conduisant à des écarts potentiellement plus importants entre sous-domaines. Des

techniques avancées de redistribution dynamique reposant sur des librairies telles que METIS ou SCOTCH pourraient être proposées, mais la granularité des problèmes envisagés conduiraient à des résultats décevants. En effet, l'efficacité de telles approches est fonction de la fréquence des phases de redistribution mais également du nombre de sous-domaines. Il s'agit d'exploiter la connaissance du contexte applicatif afin de proposer un mécanisme efficace de répartition des éléments finis.

Consommation mémoire et algèbre linéaire creuse

Un impact important de l'émergence des puces multicœurs est une augmentation de la consommation mémoire pour certaines applications. Dans le cas d'un partitionnement standard, l'augmentation du nombre de cœurs rend défavorable le ratio taille des zones de recouvrement/taille des zones de calcul intérieures. Ce point est particulièrement pénalisant en termes d'opportunité de recouvrement du temps de communication mais également de consommation mémoire. Dans le cas d'un modèle de programmation par échange de messages, les informations redondantes augmentent alors quasi proportionnellement au nombre de cœurs.
Les solveurs linéaires creux sont particulièrement sensibles à l'augmentation du nombre de sous-domaines. Pour les solveurs itératifs, cela correspond généralement à une dégradation importante de la qualité numérique avec un plus grand nombre d'itérations de convergence. Dans le cas des solveurs directs, la consommation mémoire va croître de façon importante en raison de la phase d'agrégation des contributions à des blocs d'inconnues distantes et aux buffers temporaires alloués durant cette étape. Le recours à des techniques de programmation hybride est généralement nécessaire et l'intégration de ces bibliothèques numériques au sein des applications sismiques demeure complexe pour des modélisations à très grande échelle.

Première partie

Modélisation élastodynamique par la méthode des différences finies

Chapitre 1

Mécanismes d'équilibrage de charge

Contents

Ce chapitre aborde la prise en compte du déséquilibre de charge induit par l'utilisation des conditions aux limites absorbantes pour le domaine de calcul. Le calcul des points situés dans cette zone est plus coûteux que pour les points situés à l'intérieur du domaine car un plus grand nombre d'équations doivent être traitées. Les implémentations parallèles standard reposent sur la librairie MPI et la décomposition régulière utilisée induit un déséquilibre de charge entre sous-domaines. En effet les processeurs correspondant au bord du domaine physique avec un grand nombre de points appartenant aux couches absorbantes ont alors une charge de calcul supérieure à ceux calculant les sous-domaines intérieurs. Nous verrons donc comment aborder ce problème en introduisant un mécanisme dynamique de régulation de la charge. Sauf mentions contraires, l'exemple de test de propagation sismique utilisé dans cette section est de dimension $500\times$ 500 $\times$ 325 points de grille et consomme environ 12 Go de mémoire. Il s'agit d'une simulation académique illustrant la propagation des ondes sismiques dans un milieu homogène avec une topographie régulière et une couche absorbante de type *CPML* (voir 1.2.3.0) de 10 mailles d'épaisseur. Son analyse physique ne présente pas d'intérêt particulier.

1.1 Description et caractérisation du déséquilibre de charge

La modélisation de la propagation des ondes sismiques dans un milieu élastique à l'échelle du bassin sédimentaire conduit à des domaines de calcul avec une géométrie de cuvette. La figure 1.1 détaille une telle géométrie avec trois zones de calcul différentes matérialisées par différentes couleurs (les bords absorbants, la condition de surface libre et le domaine physique). Ces trois

zones correspondent à des approches numériques différentes (ordre et forme du stencil de différences finies, nombre d'inconnues).
Les approches numériques implémentées sont celles décrites dans le premier chapitre de ce ma-

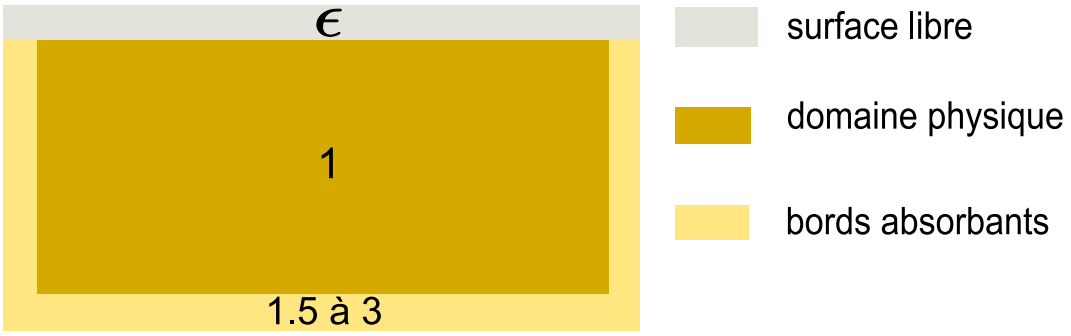

FIGURE 1.1 – Géométrie synthétique du bassin sédimentaire étudié avec les différentes zones de calcul ; leurs poids respectifs sont également indiqués.

nuscrit dans les sections 1.2.1 et 1.2.3.0. Le domaine physique est évalué de manière uniforme en utilisant un stencil d'ordre 4 en espace et nécessitant deux mailles de recouvrement dans chaque direction. Dans le cas de l'implémentation des conditions absorbantes de type PML, le coût de calcul varie en fonction de la méthode et des directions d'amortissement. Le ratio mesuré par rapport aux points intérieurs du domaine physique varie de 1.5 à 3. L'autre zone de calcul est la condition de surface libre. Il apparaît que le coût CPU du traitement numérique de cette zone peut être significatif pour des calculs tridimensionnels [73]. Dans notre cas, la "méthode des images" implémentée présente un coût CPU relativement faible mais avec une précision plus faible.

1.2 Approche par passage de messages

Dans le cas des implémentations standards, l'équilibrage de charge repose sur l'attribution d'un nombre uniforme de points sur les différents processeurs. Mais une telle stratégie ne tient pas compte du coût de calcul différent pour les points de la grille appartenant aux couches absorbantes et pour ceux du domaine physique. Une première possibilité pour résoudre ce problème est d'utiliser un découpage horizontal suivant les directions X et Y. La direction Z reste globale, ce qui permet une répartition relativement égale entre les processeurs de la couche absorbante à la base du domaine. Il est cependant important de souligner la non-uniformité du coût CPU des points de la grille dans cette zone en raison des spécificités de l'absorption dans les coins du domaine.

Ce découpage horizontal ne conduit pas à une solution optimale en termes de performance parallèle (cf. tableau 1.1 du premier chapitre). Néanmoins, il s'agit d'un compromis permettant de conserver une symétrie dans les directions horizontales, en évitant ainsi de distribuer la direction verticale Z qui est bornée par la condition de surface libre et par les conditions absorbantes en profondeur. Dans notre cas, un autre avantage de cette stratégie est de maximiser la taille des tableaux dans la direction verticale, ce qui permet d'optimiser les performances du *prefetching* dans la direction contigüe de la mémoire (*unit stride*).

Une autre approche envisageable est la prise en compte explicite du poids des différentes zones de calcul lors du découpage du domaine de calcul. L'utilisation de techniques basées sur le partitionnement de graphe de type METIS ou SCOTCH ne semble pas adaptée à notre cas car elles conduiraient à perdre l'avantage d'une implémentation simple reposant sur la structure régulière de la grille de discrétisation. Les approches d'équilibrage dynamique utilisées dans le cas d'applications MPI reposant sur des grilles non-structurées semblent trop lourdes à mettre en œuvre dans notre cas. Elles sont généralement utilisées pour des codes très irréguliers implémentant par exemple des méthodes numériques de type *AMR* (Adaptive Mesh Refinement [124]).
L'approche la plus simple à implémenter est un découpage dit quasi-statique qui permet d'ajus-

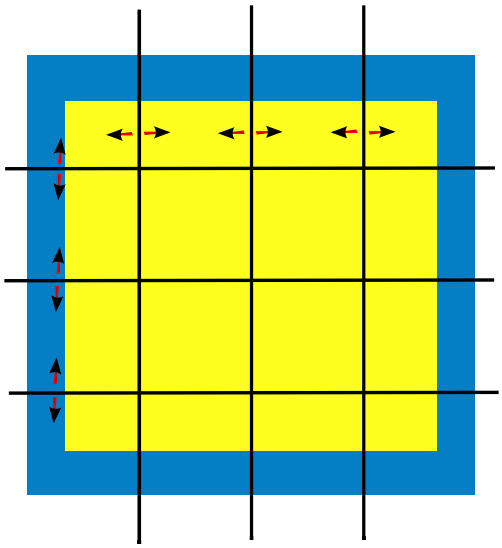

FIGURE 1.2 – Exemple d'un partitionnement quasi-statique en 2D, le domaine physique est matérialisé en jaune et la couche absorbante en bleu. La taille des sous-blocs est variable dans les deux directions d'espace afin de prendre en compte le coût différent des zones de calcul.

ter la taille des sous-domaines en fonction de poids définis statiquement. Ce découpage variable est défini avant le début de la boucle d'évolution en temps et la taille des sous-domaines est donc pré-calculée. L'évaluation précise du ratio entre les zones physiques et les zones PML est donc déterminante pour l'efficacité d'une telle approche. Cette stratégie est décrite sur la figure 1.2 avec une correction possible dans les deux directions horizontales. Elle a été utilisée dans le cas des équations de Maxwell en électromagnétisme [142] ; l'implémentation de ces dernières étant très similaire à celle des équations de l'élastodynamique au niveau de l'organisation des calculs et du coût des conditions aux limites absorbantes.
Pour le calcul du ratio nombre de points physiques/nombre de points de la zone absorbante, plusieurs stratégies peuvent être envisagées :

- Une évaluation basée sur le nombre d'opérations de la méthode numérique est relativement inefficace car elle ne prend pas en compte l'implémentation réelle (optimisations du compilateur, effets de cache ...). Globalement, la correspondance entre le nombre théorique d'opérations et le temps CPU mesuré est assez mauvaise, le temps d'exécution calcul étant fortement influencé par l'organisation des données et la manière d'y accéder.
- Une évaluation basée sur la mesure dans le code des routines pertinentes conduit à des résultats beaucoup plus fiables. Malheureusement, la prise en compte des zones situées

dans les coins du domaine 3D reste difficile car le coût de calcul est variable, en fonction du nombre de faces extérieures dans la couche absorbante.

En prenant l'exemple d'un domaine cubique, le choix du grain pour le rééquilibrage est déterminant. Dans le cas d'un partitionnement 1D du domaine tridimensionnel, la correction induite par la migration d'un plan 2D entre deux sous-domaines sera inadaptée à la valeur réelle du déséquilibre de charge. En effet le coût de calcul de l'ensemble des points de grille composant ce plan pourra alors se révéler très supérieur à l'écart en temps de calcul entre sous-domaines.
Nous effectuons une première évaluation en utilisant un modèle théorique pour les coûts des différentes zones de calcul, dans le cas du découpage 2D d'un modèle tridimensionnel. Un exemple de calibration de ce modèle théorique est décrit par la figure 1.3 sur les 80 cœurs de la plateforme `Borderline`. Nous définissons le déséquilibre de charge par l'écart maximum de temps entre sous-domaines. La figure 1.4 présente les résultats obtenus en utilisant le modèle théorique de coût pour un calcul tridimensionnel avec 2.1×10^9 points, 10 points d'épaisseur étant utilisés pour la couche absorbante. Cet exemple simplifié correspond aux dimensions des simulations décrites dans [39]. La valeur retenue pour le ratio entre le coût des différentes zones est 2.4. Elle est issue d'un pré-calcul dans les différentes portions du domaine géométrique. La comparaison de l'implémentation d'un découpage horizontal statique ou quasi-statique révèle des gains importants, sur 256 cœurs le déséquilibre de charge passe de 21% à 5% dans le cas quasi-statique.

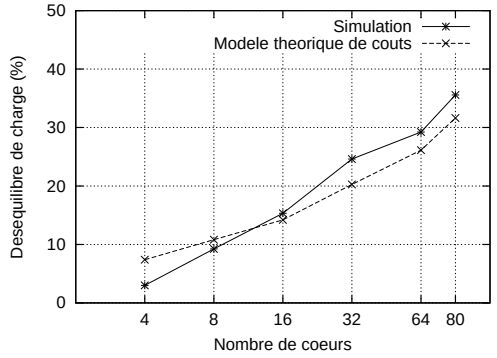

FIGURE 1.3 – Comparaison entre le modèle théorique et la simulation réelle sur la plateforme `Borderline`

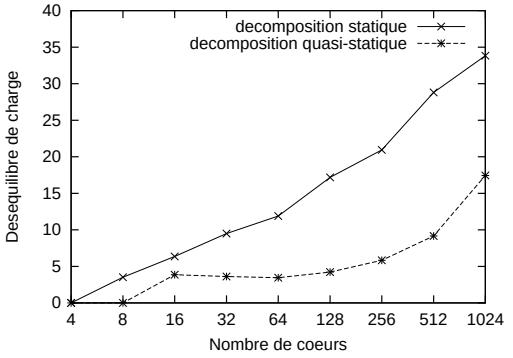

FIGURE 1.4 – Déséquilibre de charge dans les cas statique et quasi-statique avec un modèle théorique de coûts sur 1024 cœurs

Malheureusement, ces bons résultats ne sont valables que pour des tailles de sous-domaines physiques dont l'épaisseur est très largement supérieure à celle de la couche absorbante. Dès que le grain de calcul est raffiné avec l'augmentation du nombre de processeurs, le déséquilibre de charge augmente très sensiblement dans le cas quasi-statique, avec des valeurs supérieures à 18% sur 1024 cœurs. Cette stratégie favorise donc l'utilisation d'un parallélisme à gros grain permettant un premier niveau de correction. L'introduction d'un deuxième niveau de parallélisme, offrant une plus grande flexbilité, semble donc nécessaire.

1.3 Approche dynamique et implémentation OpenMP

1.3.1 Parallélisme imbriqué et impact du support d'exécution

L'exploitation de la mémoire partagée à l'échelle des nœuds de calcul multiprocesseurs peut permettre la mise en œuvre d'un mécanisme dynamique de régulation de la charge. Dans ce cas, le modèle de programmation OpenMP apparaît comme une solution simple et bien adaptée à l'application ONDES3D. En effet la méthode numérique utilisée se décompose en deux triples boucles imbriquées et cette structure permet l'utilisation du parallélisme issu de ces deux nids de boucles. L'implémentation est donc immédiate mais elle n'introduit pas encore un partage dynamique des tâches ; elle est illustrée par le pseudo-code de l'algorithme 1 et sera qualifiée de "standard" dans le reste du manuscrit.

Par défaut, le découpage de ces boucles conduira à une répartition égale du nombre de points

Algorithm 1 Implémentation OpenMP standard pour la méthode des différences finies

```
#pragma omp parallel for
for i = 1 to nx do
  for j = 1 to ny do
    for k = 1 to nz do
      Calcul des vitesses ou des contraintes
    end for
  end for
end for
```

de calcul entre les différents threads de calcul. Le recours aux fonctions avancées de distribution offertes par le standard OpenMP est donc nécessaire. Dans notre cas, nous utilisons un ordonnancement dynamique permettant de diviser les boucles de calcul avec un grain minimal. Les possibilités offertes par la clause *schedule(dynamic,1)* sont exploitées avec une granularité égale à 1, cette distribution utilise un plan 2D comme unité élémentaire d'ordonnancement. La figure 1.5 illustre cette approche dans le cas d'un domaine cubique. L'exemple de test défini en préambule

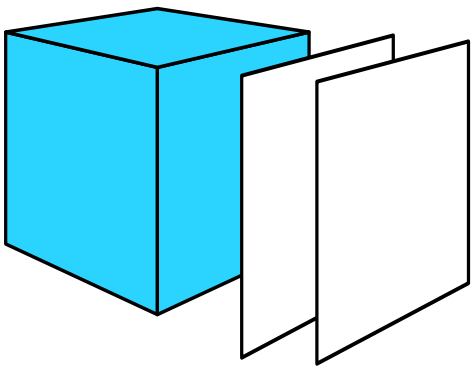

FIGURE 1.5 – Ordonnancement OpenMP dynamique en considérant un domaine de calcul cubique. Les plans 2D sont les unités de calcul réparties sur les différents cœurs.

de ce chapitre est utilisé sur un nœud de la plateforme `Phoebus` (voir annexe B.2.1). Les résultats obtenus sont encourageants, avec une réduction significative du déséquilibre de charge et une accélération de 25% en temps par rapport à l'implémentation standard. Cependant, un certain nombre de restrictions sont à souligner. En effet, le partage des tâches se fait exclusivement par

rapport à la boucle externe du nid de boucles. Cette distribution monodimensionnelle ne permet pas d'extraire totalement le parallélisme de l'application. De plus, à l'instar de l'implémentation MPI, le grain de rééquilibrage pourrait s'avérer trop grossier dans certains cas. Il semble donc opportun de prendre en compte les autres directions d'espace afin de maximiser l'efficacité de la distribution. L'imbrication des différents niveaux de parallélisme est alors nécessaire.

Globalement, l'imbrication de directives OpenMP (*OMP_NESTED*) est assez peu répandue

Algorithm 2 Implémentation OpenMP avec imbrication de deux niveaux de parallélisme

```
#pragma omp parallel for
for i = 1 to nx do
  #pragma omp parallel for
  for j = 1 to ny do
    for k = 1 to nz do
      Calcul des vitesses ou des contraintes
    end for
  end for
end for
```

dans le domaine du calcul scientifique. Néanmoins, cette stratégie à déjà montré son efficacité (solveurs multizones, problèmes de granularité, applications adaptatives, ...). Le faible niveau de performance des supports d'exécution OpenMP pour la gestion de ce type d'implémentation explique principalement ce faible engouement. Il est important de noter que le nombre de threads crées dans ce contexte est généralement très supérieur au nombre de threads utilisés dans le cas standard. De plus, en fonction des implémentations, la création des threads n'intervient pas uniquement en début de calcul mais se répète pour chaque région parallèle. La prise en compte efficace du grand nombre de threads générés nécessite donc un faible coût de création, de synchronisation et d'ordonnancement de ces derniers ; peu d'implémentations du standard OpenMP permettent donc l'exploitation réelle de plusieurs niveaux imbriqués de parallélisme [70].

La plateforme FOREST GOMP (décrite section 1.3.3.0) offre un support avancé du parallélisme imbriqué en s'appuyant sur la bibliothèque de threads MARCEL. Une boucle de calcul de Jacobi en 3D est implémentée afin de mesurer ses performances par rapport à l'implémentation OpenMP *GNU*. Cet exemple de dimension 500×500×500 points de grille permet de reproduire les principales caractéristiques du code ONDES3D en termes de schéma d'accès aux données et d'intensité de calcul. Afin d'optimiser les performances, seules deux directions d'espace sont imbriquées (cf. algorithme 2). Le serveur `Idkoiff` (voir annexe B.1.5) est utilisé pour ces expériences.

La figure 1.6 présente les résultats de ces expérimentations. En abscisse, on retrouve le nombre total de threads utilisés, chaque courbe correspondant à un nombre de threads différent pour le niveau intermédiaire de parallélisme. Par exemple 64 threads au total avec 4 threads pour le niveau intermédiaire correspond à une situation avec 16 threads pour le niveau le plus externe. On peut tirer les enseignements suivants :

- Globalement, on constate dans les deux cas de meilleures performances en utilisant un minimum de threads internes. Cela traduit la dégradation très sensible des performances avec l'imbrication des niveaux de parallélisme. En utilisant un nombre constant de threads, il est plus efficace de se limiter à la boucle externe de calcul. Cette dégradation est beaucoup

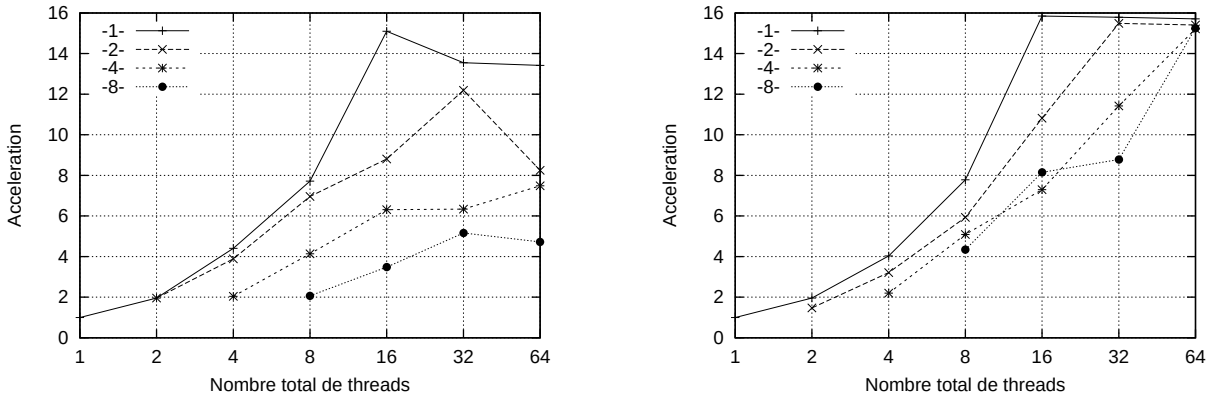

FIGURE 1.6 – Parallélisme imbriqué avec les librairies GOMP (gauche) et FOREST-GOMP (droite) dans le cas d'une boucle de Jacobi 3D.

moins marquée avec FORESTGOMP comparativement à l'implémentation *GNU* OpenMP. En effet la situation de surcharge induite est difficilement compatible avec l'utilisation de threads de niveau noyau qui occasionnent des appels système fréquents et coûteux.

– Un comportement quasiment *asymptotique* de la plateforme FORESTGOMP est mesuré, les résultats s'améliorant avec l'augmentation du nombre de threads. On obtient une accélération supérieure à 15 dans tous les cas en utilisant 64 threads sur 16 cœurs. Ce résultat est remarquable compte tenu des accès mémoire non uniformes. Ce point sera détaillé dans le chapitre suivant. Dans ce contexte, le surcoût dû à l'imbrication des régions parallèles semble très faible. De façon plus précise, les performances sont optimales lorsque le nombre de bulles créées (niveau externe de parallélisme) correspond au nombre de cœurs disponibles sur la machine. C'est la situation la plus favorable en termes d'ordonnancement.

GOMP standard *(16)*	**FORESTGOMP** standard *(16/64)*	**GOMP** imbriqué *(64)*	**FORESTGOMP** imbriqué *(64)*
21.03 s	21.98/16.47 s	73.33 s	22.70 s

TABLE 1.1 – Parallélisme imbriqué avec le code ONDES3D, comparaison des temps d'exécution avec les librairies GOMP et FORESTGOMP. Le nombre de threads est indiqué entre parenthèses

Le cas test utilisé précédemment est repris afin d'évaluer les performances de la plateforme FORESTGOMP avec le code ONDES3D. Les environnements GOMP et FORESTGOMP sont comparés et les résultats obtenus sur la plateforme `Idkoiff` sont donnés dans le tableau 1.1. Les implémentations OpenMP standard et imbriquées sont analysées avec un nombre variable de threads. En considérant les versions utilisant un parallélisme imbriqué (8 threads pour les boucles externes et internes), l'écart significatif entre les librairies GOMP et FORESTGOMP souligne l'impact sur les performances de la bibliothèque de threads supportant le parallélisme OpenMP. En revanche, la comparaison des résultats obtenus avec 16 threads semble indiquer un léger surcoût des versions FORESTGOMP (en version standard ou imbriqué). Cet écart relativement faible (inférieur à 10%) ne se justifie pas d'un point de vue théorique et nécessiterait

des investigations complémentaires. De plus, notre implémentation ne semble pas bénéficier des possibilités de vol de travail automatique entre bulles intégrées à la plateforme ForestGOMP. Des tests complémentaires ont été menés afin d'évaluer le lien entre ces performances et les directions choisies pour l'imbrication du parallélisme. Les conclusions ne sont pas définitives mais un lien semble apparaître entre la forme et le poids de calcul des sous-domaines à migrer et les performances obtenues. Enfin, le manque de contrôle au niveau applicatif sur le mécanisme de migration implémenté rend difficile une analyse complète de cet aspect des résultats.
Néanmoins, l'implémentation ForestGOMP standard avec 64 threads sur 16 cœurs permet une accélération par rapport aux versions standards GOMP et ForestGOMP avec 16 threads (21%). Ce résultat provient des possibilités offertes par l'ordonnancement dynamique des threads Marcel avec une correction partielle du déséquilibre de charge induit par les conditions absorbantes.

1.3.2 Limites du modèle de programmation

Les possibilités offertes par le standard OpenMP permettent d'aborder les problèmes de déséquilibre de charge de façon relativement efficace. Cependant les limitations identifiées en termes d'extraction du parallélisme sont importantes. L'imbrication de plusieurs régions parallèles dégrade sévèrement les performances si le programmeur se base sur des implémentations OpenMP standard et faiblement optimisées pour la gestion de ce type de parallélisme. En revanche, des environnements tels que ForestGOMP permettent d'améliorer significativement les résultats. Cependant, l'utilisation de directives OpenMP reste peu flexible et pourrait constituer un goulot d'étranglement en vue de l'implémentation d'algorithmes plus complexes, pour l'optimisation de la localité mémoire par exemple. En revanche, les résultats obtenus en utilisant un mécanisme de virtualisation des processeurs et un modèle de threads adapté sont encourageants. L'environnement de programmation MPC (voir section 1.3.2.0), optimisé pour l'ordonnancement d'un grand nombre de threads, pourrait alors constituer une alternative au modèle OpenMP en offrant une plus grande souplesse dans l'expression du parallélisme.

1.4 Exploitation du mécanisme de virtualisation

1.4.1 Expériences avec l'environnement de programmation MPC

Le passage d'une version MPI à une version MPC du code *Ondes3D* est largement simplifié par l'implémentation initiale qui ne définit aucune variable globale. Cette condition est nécessaire afin de garantir des accès locaux aux zones mémoire allouées pour chaque thread MPC. Il s'agit donc simplement d'effectuer une commande *find/replace* avec un éditeur de texte pour modifier les appels aux fonctions *MPI_foo* par les fonctions miroirs *MPC_foo*. La figure 1.7 décrit les expériences menées sur la plateforme `Teranova` (voir annexe B.1.1). Le cas test sismique simplifié défini en introduction est exploité dans le cas d'un découpage 2D. Les résultats des implémentations MPI et MPC sont globalement identiques ce qui montre que les stratégies de découpage et d'équilibrage sont identiques. En termes de déséquilibre de charge, les choix faits pour l'implémentation de la bibliothèque de threads MPC expliquent les performances obtenues. Cette dernière possède en effet certaines particularités qu'il est intéressant de rappeler [130] :

- l'ordonnanceur de threads de niveau utilisateur n'est pas préemptif, les applications scientifiques étant ciblées, il est alors important de profiter d'éventuels effets de cache qu'un mécanisme de préemption pourrait annihiler ;

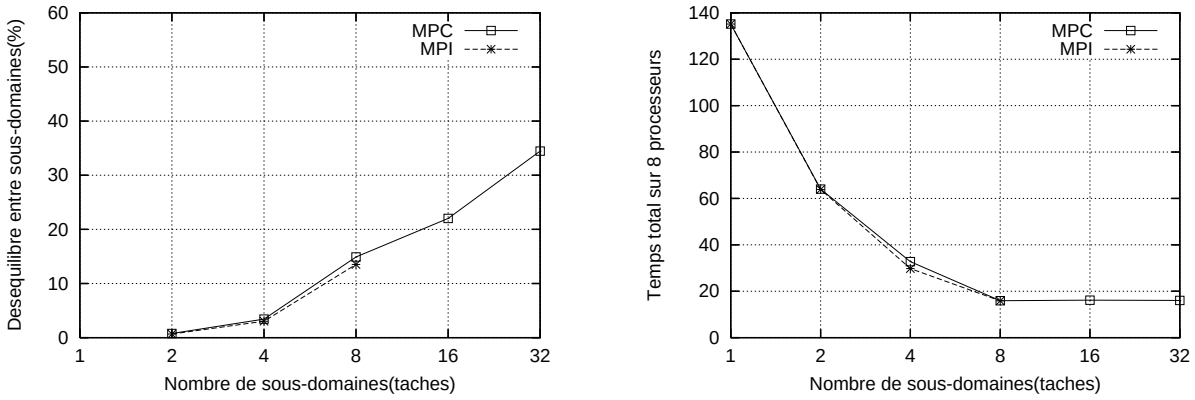

FIGURE 1.7 – Comparaison des implémentations MPI et MPC en termes de déséquilibre de charge (gauche) et de temps de calcul (droite).

- MPC n'implémente pas de priorité au niveau des threads car l'objectif initial est de reproduire le modèle d'exécution MPI ;
- MPC ne permet pas le partage des informations d'ordonnancement entre processeurs virtuels (threads noyaux). Dès sa création, chaque thread utilisateur est affecté à un processeur virtuel et, sauf décision venant de l'application par une migration explicite, cette affectation ne change pas durant l'exécution du code.

Les threads utilisateurs sont donc "vissés" sur les threads noyaux et l'ordonnanceur ne peut pas corriger la charge sur certains cœurs moins chargés (par exemple les threads de calcul correspondant au domaine physique).

1.4.2 Impact de la stratégie de multithreading

Afin d'évaluer différentes bibliothèques de threads, nous implémentons un cas test simple permettant d'évaluer les opportunités de régulation dynamique de la charge. L'exemple utilisé définit 4 tâches avec des poids de calcul $(P + X, P, P, P)$. Nous supposons une distribution de ces tâches sur une architecture composée de 4 cœurs. Le coût de la tâche X est défini comme constant, en revanche le coût des tâches P est divisé par deux lorsque le nombre de threads est multiplié par deux. La figure 1.8 illustre cet exemple.
Le coût des tâches P est également initialement défini comme très inférieur au coût de la tâche X. L'objectif est d'observer les gains possibles en exploitant la virtualisation des cœurs, la situation optimale étant d'atteindre un temps de calcul égal à $X + \epsilon$. Ce résultat correspondrait à l'ordonnancement d'un grand nombre de threads permettant de lisser le coût des tâches de type P sur les différents cœurs jusqu'à un surcoût ϵ par rapport à X. Typiquement, une implémentation basée sur un modèle de programmation MPI conduirait à un temps de calcul égal à $P + X$ sur 4 cœurs.
Trois bibliothèques de threads ont été évaluées. Il s'agit de la librairie spécifique de threads sur laquelle repose l'environnement MPC, de la librairie Pthreads et de la bibliothèque de threads MARCEL. Cette dernière a été compilée avec l'option *SMP* et la préemption est désactivée (*marcel_thread_preemption_disable()*). L'ordonnanceur glouton *MARCEL_SHARED* est utilisé, ce dernier permet une répartition dynamique des threads sur les différents cœurs à partir d'une liste

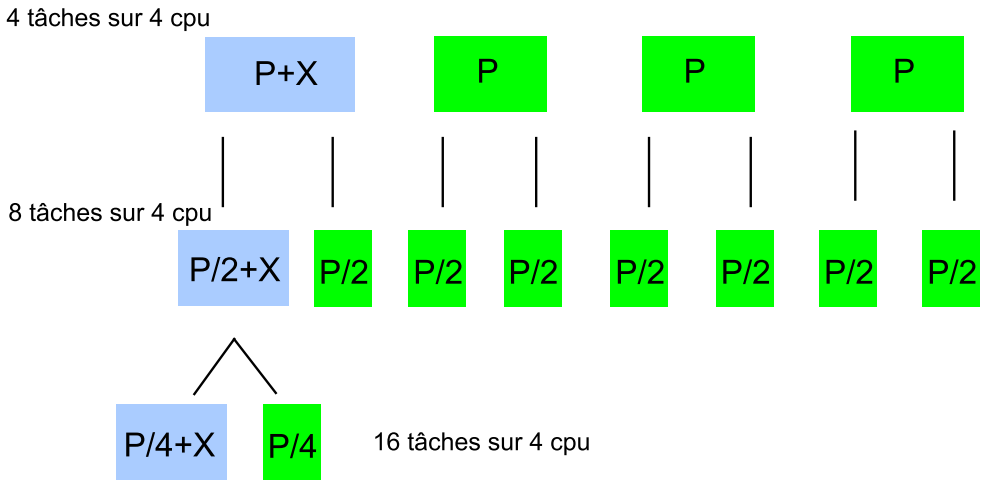

FIGURE 1.8 – Cas test théorique reproduisant un problème déséquilibré avec quatre tâches de poids $(P + X, P, P, P)$ sur quatre cœurs.

unique de threads disponibles. Le test est réalisé sur un nœud de la machine `Lias` (voir annexe B.1.2) qui est composé de 2 processeurs dual-cœur. L'implémentation des tâches est réalisé à partir de noyaux de calcul de type Jacobi et plus de 60% de la mémoire disponible (8 Go) est occupée.

Performances en exploitant la virtualisation des processeurs

Tout d'abord, un test parfaitement uniforme (P, P, P, P) est réalisé en définissant $X = 0$. L'objectif de ces simulations est double. Dans un premier temps, il s'agit d'observer les effets de la préemption dans un cas équilibré. Dans un deuxième temps, nous souhaitons comparer le surcoût induit par les nombreux changements de contexte des threads en situation de surcharge. La figure 1.9 présente les résultats obtenus.

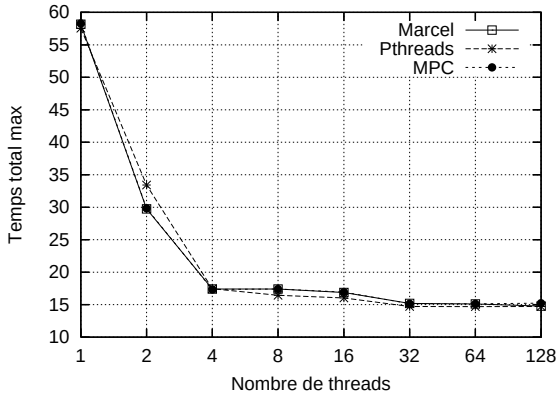

FIGURE 1.9 – Temps total d'exécution en considérant quatre tâches de même poids (X=0).

Les performances des trois bibliothèques se révèlent assez similaires quand le nombre de threads est inférieur ou égal au nombre de cœurs. Ce résultat était attendu en l'absence de contraintes importantes au niveau de l'ordonnancement des threads. A partir de 8 threads le

mécanisme de virtualisation est utilisé. Le temps total d'exécution est également quasiment confondu ce qui semble démontrer un impact faible du modèle de threads utilisé. En effet, des écarts plus significatifs auraient été assez logiques entre des threads POSIX coûteux (en termes d'appel système) et des threads exploitant un ordonnancement à deux niveaux (MPC et MARCEL). Dans notre cas, les performances similaires sont encourageantes mais il est probable que l'augmentation du nombre de threads ou l'utilisation d'une architecture comportant plus de cœurs modifierait ces conclusions.

Répartition dynamique de la charge et ordonnancement des threads

Les simulations sont maintenant réalisées à partir du cas déséquilibré (cf. figure 1.8). Les courbes de la figure 1.10 présentent les résultats obtenus. Les temps de calcul sont relativement similaires jusqu'à 4 threads confirmant les mesures de la section précédente. De manière identique, on observe un léger surcoût pour la librairie *Pthreads* avec 2 threads. En revanche, les courbes obtenues avec plus de 4 threads varient significativement en fonction de la bibliothèque de threads utilisées.

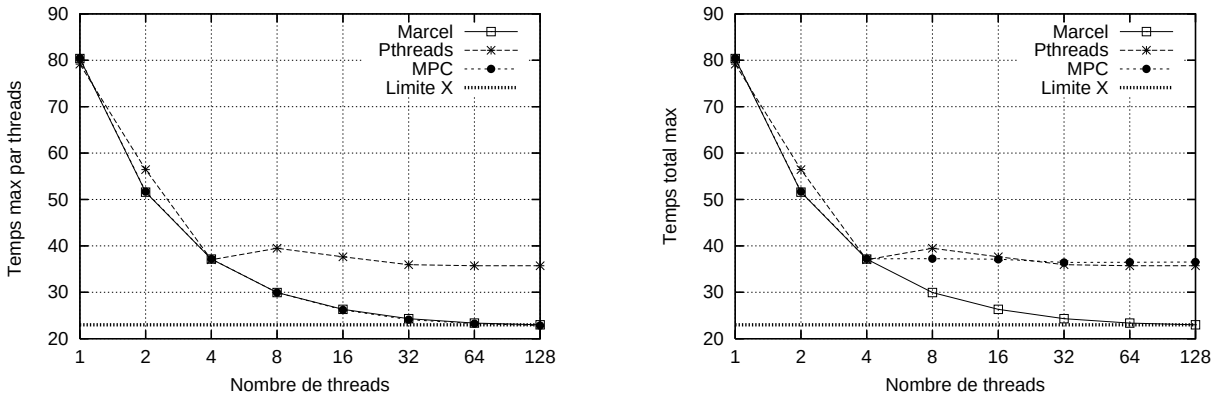

FIGURE 1.10 – Temps maximum d'exécution par thread (gauche) et temps total d'exécution (droite) en considérant un problème déséquilibré.

- **Threads POSIX :** Dans le cas de la librairie *Pthreads*, la courbe décrivant le temps maximal par thread et celle décrivant le temps maximal de la simulation suivent la même tendance avec l'augmentation du nombre de threads. En effet, l'ordonnanceur de threads cherche à partager l'utilisation des ressources de façon équitable entre les différents threads. Cette stratégie conduit à un lissage très faible de la charge totale du problème, et le temps mesuré sur 128 threads est assez éloigné de la borne inférieure X.

- **Threads MPC :** Les résultats obtenus avec la bibliothèque de threads MPC sont assez différents de ceux observés dans le cas des threads POSIX. En effet, le temps maximum par thread est conforme aux conclusions de la section 1.4.1 utilisant l'environnement de programmation bâti au-dessus de cette bibliothèque de threads ; on constate une absence de partage d'informations d'ordonnancement entre les processeurs virtuels. Cette approche conduit à un temps maximum par thread qui correspond à la tâche $P + X$, le coût de

la tâche P diminuant avec le nombre de threads utilisé. On tend donc bien vers la borne minimum X. En revanche, le temps total de simulation correspond au cumul des temps d'exécution d'un nombre fixe de threads MPC sur un des 4 processeurs virtuels. L'affectation des threads sur ces derniers étant pré-calculée, il n'est pas possible d'équilibrer la charge dynamiquement. Pour des raisons différentes, les performances finales se révèlent donc relativement proches de celles obtenues avec les threads POSIX.

– **Threads MARCEL :** Les courbes MPC et Marcel sont équivalentes au niveau du temps maximum par thread, mais l'impact des choix différents au niveau de l'ordonnanceur apparaît explicitement au niveau du temps total mesuré. Contrairement à MPC, la librairie MARCEL permet d'obtenir un résultat optimal au niveau du temps total de calcul. Dans ce cas, nous bénéficions de l'approche gloutonne utilisée. Celle-ci permet de tenir compte de la charge avant de décider de l'ordonnancement d'un nouveau thread. Les possibilités de priorité sont également utilisées dans ce cas afin de privilégier l'ordonnancement des tâches les plus coûteuses au début de l'exécution et ensuite lisser la charge sur l'ensemble de la machine à l'aide des tâches les moins coûteuses.

1.4.3 Validation

Dans cette section, nous évaluons plus finement les mécanismes de régulation de charge décrits précédemment en reprenant le problème sismique décrit en introduction de ce chapitre. Un nœud de la plateforme `Phoebus` est exploité pour ces tests. La figure 1.11 présente les résultats obtenus en comparant deux implémentations du code Ondes3D. Dans le premier cas, la version OpenMP dynamique divise le domaine de calcul 3D en plans qui sont ensuite répartis sur les différents processeurs (cf. figure 1.5). La librairie GOMP est utilisée. L'API de la librairie Marcel est utilisée pour la deuxième version. La procédure générale de calcul comprend deux phases de création/synchronisation des threads qui correspondent aux deux triples boucles imbriquées définissant le calcul des vitesses et des contraintes (cf. algorithme 3). L'écart mesuré entre la destruction réelle des threads entre les phases de calcul ou le fait de les bloquer sur une condition est faible.

Algorithm 3 Schéma d'exécution de l'implémentation multithreadée du code Ondes3D

Pré-traitement : Définition des sous-domaines affectés à chaque thread
for $t = 1$ to $TMAX$ **do**
  Création des threads
  Calcul des vitesses
  Synchronisation des threads
  Calcul des contraintes
  Synchronisation des threads
end for

Une décomposition horizontale identique au modèle de décomposition MPI est ainsi introduite. L'avantage est de pouvoir exploiter un grand nombre de threads dans ce cas. Chaque thread correspond à un sous-domaine physique. Les phases de calcul des vitesses et des contraintes peuvent uniquement générer de la concurrence entre threads pour la lecture des données (au bord de chaque sous-domaine). Le stencil de calcul conduit chaque thread à écrire de manière

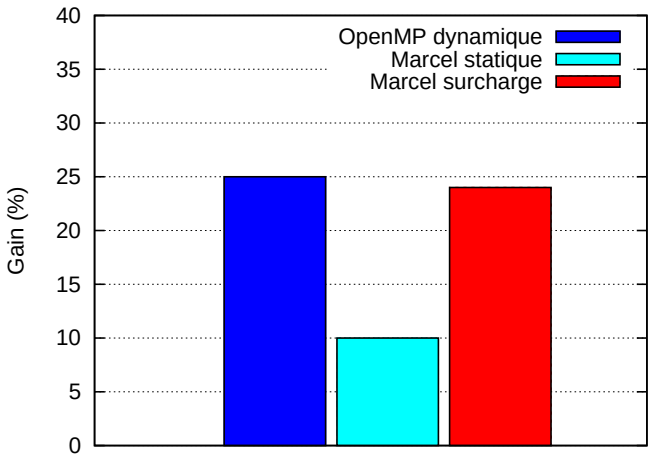

FIGURE 1.11 – Gains associés à différentes techniques d'équilibrage de charge sur la plateforme `Phoebus`. La version OpenMP standard est prise comme référence.

unique dans la zone mémoire qui correspond aux données du sous-domaine calculé. Les gains présentés sur la figure 1.11 sont calculés par rapport à la version OpenMP standard.

En version statique (le nombre de threads est égal au nombre de cœurs de l'architecture cible), l'implémentation basée sur des threads MARCEL conduit à de meilleurs résultats que la version OpenMP standard (environ 10%). Cette différence s'explique par le découpage 2D mis en œuvre dans le premier cas alors que les directives OpenMP ne permettent qu'un découpage 1D. Ce dernier est moins performant en termes de correction du déséquilibre de charge applicatif à cause du grain plus grossier de rééquilibrage. En revanche, les performances des versions dynamiques sont relativement similaires. Concernant l'implémentation MARCEL, le coût des phases de synchronisation ne semble pas constituer un goulot d'étranglement car l'ordonnancement des 800 threads utilisés semble transparent. Des évaluations plus fines permettant de quantifier la taille optimale du grain par rapport à la dégradation des performances provenant du support d'exécution seraient nécessaires. Dans notre cas, les gains observés sont de l'ordre de 25% par rapport à une implémentation OpenMP standard.

Le tableau 1.2 permet de compléter l'analyse du mécanisme de surcharge utilisé avec la bibliothèque de threads MARCEL. Le nombre limité de cœurs disponibles sur chaque nœud (4) facilite

	P_v #0	P_v #1	P_v #2	P_v #3
4 threads avec préemption	9.44	10.81	11.01	9.82
4 threads sans préemption	10.91	10.54	9.75	9.77
52 threads sans préemption	10.26	10.04	11.01	9.82
196 threads sans préemption	9.97	9.86	9.85	9.90

TABLE 1.2 – Temps de calcul et équilibrage de charge sur `Lias` en utilisant un mécanisme de virtualisation

l'analyse des résultats même si le déséquilibre de charge reste modeste. Nous observons les per-

formances sur chaque processeur virtuel (noté P_v dans le tableau 1.2) pour différentes situations. Les deux premières lignes permettent de comparer les résultats obtenus avec ou sans l'utilisation d'un mécanisme de préemption. Nous utilisons quatre threads sur les quatre cœurs disponibles. Aucune différence réellement significative n'apparaît à ce niveau, le déséquilibre de charge est même légèrement supérieur avec la préemption (déséquilibre de 14.25% avec la préemption par rapport à 10.63% sans la préemption). Cela rejoint les résultats décrits dans la section 1.4.2 illustrant la faible efficacité de ce mécanisme par rapport à une approche gloutonne.

Le déséquilibre de charge est quasiment constant lorsque le nombre de threads passe de 4 à 52 (le déséquilibre de charge passe de 10.63% à 10.8%). On peut supposer que le grain de calcul n'est pas encore suffisamment fin pour lisser la charge de façon efficace. Des gains plus significatifs sont observés avec 196 threads, le déséquilibre de charge étant ramené autour de 1%. Le temps de calcul est alors diminué de 9.16 %. Un léger effet de cache est obtenu en utilisant la surcharge. En effet, la charge totale de calcul (en cumulant le temps pour tous les processeurs virtuels) passe de 40.72 s pour 4 threads à 39.59 s pour 196 threads. Ce gain s'explique par la taille des micro-domaines, ces derniers pouvant désormais rentrer dans le cache. Le gain reste cependant faible ; dans notre cas, il est de l'ordre de 2.7%.

Le mécanisme dynamique de régulation de charge introduit dans ce chapitre repose sur l'exploitation d'un multithreading intensif. Le choix du support d'exécution est alors déterminant afin de tenir compte de l'architecture sous-jacente et notamment du coût non uniforme des accès à la mémoire.

Chapitre 2

Prise en compte de l'affinité mémoire

Contents

Habitué à disposer sans effort d'une puissance de calcul croissante (en se reposant uniquement sur l'augmentation de la fréquence), le programmeur doit désormais faire face à la problématique de l'utilsation efficace de véritables architectures NUMA (Non-Uniform Memory Access). Le placement efficace des données par rapport aux tâches de calcul est alors crucial afin de maximiser l'affinité mémoire. Il s'agit dans un premier temps de caractériser la pénalité NUMA en tenant compte du contexte applicatif. Les impacts du schéma d'accès aux données, de la stratégie de multithreading implémentée ou de la taille du problème seront évalués sur différentes architectures. Deux approches sont ensuite étudiées plus spécifiquement. La première proposition consiste à implémenter un mécanisme d'ordonnancement structuré des threads en s'appuyant sur la grille cartésienne décrivant le domaine de simulation ; la mise en œuvre explicite au niveau applicatif sera décrite et discutée. La deuxième approche est basée sur l'environnement MAi [134] qui propose une approche peu intrusive afin de définir la politique mémoire la mieux adaptée.

2.1 Caractérisation des accès aux données

Le lien de cause à effet entre la stratégie de multithreading et le schéma d'accès aux données est important à caractériser. En effet, les possibilités d'optimisation du placement des données ne seront pas les mêmes en fonction de l'implémentation choisie.

L'implémentation basée sur des directives OpenMP conduit à un découpage monodimensionnel. Dans le cas standard, le partitionnement obtenu sera alors régulier, un même sous-domaine étant pris en charge par le même thread durant tous les pas de temps de la simulation. OpenMP se borne en effet à distribuer le nombre de points traités par la boucle externe de façon égale. Les choix d'ordonnancement effectués pour les threads de calcul sont donc extrêmement importants à ce niveau. En fonction des décisions prises, les mêmes threads peuvent se retrouver (ou pas) sur les mêmes cœurs tout au long du calcul. Cela correspondrait à la répétition d'une répartition identique pour tous les pas de temps. Ce point sera discuté plus précisément dans la section suivante.

L'utilisation de mécanismes dynamiques est également envisageable, en particulier par des techniques de virtualisation des cœurs. C'est le cas des implémentations reposant sur la librairie MARCEL et décrites dans le chapitre précédent. L'attribution des sous-domaines de calcul aux différents threads est alors variable d'un pas de temps à l'autre. Dans un tel contexte, il sera difficile d'envisager des solutions reposant sur une relation statique entre le placement des threads et celui des données. Toute approche pré-calculée risque de limiter le caractère dynamique de l'ordonnancement des threads de calcul, réduisant ainsi les possibilités de régulation de la charge.

La situation est un peu différente dans le cas des directives OpenMP utilisant un répartition dynamique des sous-domaines (clause *schedule(dynamic,1)*). On peut espérer extraire des patterns répétitifs de placement des tâches permettant ainsi à un mécanisme de migration de pages mémoire de corriger un placement initial défaillant. Cette possibilité repose sur l'hypothèse d'un monitoring régulier de la stratégie de répartition des tâches sur les différents cœurs.

2.2 Pénalité NUMA

2.2.1 Expériences séquentielles

Dans cette section nous évaluons le code ONDES3D dans sa version séquentielle, le schéma d'exécution correspond à la figure 1.5. On retrouve notamment les 2 séries de triples boucles imbriquées décrivant les trois directions d'espace. Cette version du code est mise en œuvre sur les plateformes `Malm`, `Idkoiff` et `Borderline`. La taille des données du cas test correspond à la mémoire disponible sur un nœud NUMA.

Nous faisons varier alternativement le placement des données ou des threads de calcul sur les différents nœuds NUMA. La stratégie est d'occuper successivement l'ensemble des nœuds NUMA de chaque architecture afin d'évaluer l'impact de la hiérarchie mémoire pour notre algorithme ainsi que certains effets plus locaux.

Différents mécanismes peuvent influencer de manière importante les résultats mesurés, deux points sont particulièrement importants à souligner.

- Les mécanismes de prefetching (au niveau hardware ou software) peuvent avoir un impact important pour ce type d'algorithmes [145]. En effet, les accès aux données étant très réguliers, l'effet de la latence mémoire peut être partiellement masqué par de telles optimisations permettant de rapprocher automatiquement les données des threads de calcul. Ces mécanismes de *prefetching* sont beaucoup moins significatifs dans le cas d'accès irréguliers

aux données.

– La procédure numérique employée définit différentes zones de calcul (couches absorbantes, domaine physique, surface libre). Un impact variable de la hiérarchie mémoire sur ces différents stencils pourrait être observés. Ce point n'est pas analysé dans cette section car nous focalisons notre étude sur le schéma numérique correspondant au domaine physique.

Les tableaux 2.1 et 2.2 présentent les résultats issus de l'exécution séquentielle du code ONDES3D sur les plateformes `Malm` et `Borderline`. Les threads sont fixés en utilisant la fonction *sched_setaffinity* du système d'exploitation Linux (*sched.h*).Notons que le placement s'effectue dans ce cas sur les différents cœurs et non pas sur les nœuds NUMA comme dans le cas de commandes telles que *numactl* de la *libnuma*.

Le placement des threads varie selon les colonnes et le placement de la mémoire selon les lignes. Les valeurs nulles (**0**) correspondent à la référence, c'est le temps le plus faible relevé par lignes (en fixant la mémoire sur un cœur et en faisant varier le placement du thread de calcul) . Les résultats sont exprimés en pourcentage d'augmentation du temps de calcul par rapport à la situation de référence.

Résultats sur la plateforme `Malm`

Sur la plateforme `Malm`, la valeur maximale pour la pénalité NUMA est de 11.21%. Cette valeur est relativement faible mais correspond bien à une architecture peu hiérarchique avec deux processeurs quad-cœurs reliés par un lien *Hypertransport*. Si on analyse les résultats dans le détail, on peut en déduire la topologie de l'architecture (numérotation des cœurs). En effet les valeurs de pénalité sont inférieures à 1% en plaçant les données et les threads de calcul sur les cœurs *(0, 1, 2, 3)* qui appartiennent au même processeur et donc au même nœud NUMA (voir la description en Annexe B.1.2). Un placement distant des données et du thread de calcul correspond systématiquement à des valeurs de pénalité autour de 10% pour notre application.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0.44	**0**	0.02	0.06	10.89	10.95	10.95	10.92
1	0.44	0.13	**0**	0.23	10.96	10.95	10.92	11.19
2	0.58	0.05	**0**	0.11	11.02	10.96	11.02	11.01
3	0.51	0.06	**0**	0.01	11.02	11.21	11.02	11.27
4	11.12	10.66	10.61	11.19	**0**	0.10	0.07	0.12
5	11.17	10.50	10.55	10.70	0.28	**0**	0.11	0.11
6	11.04	10.78	10.52	10.55	**0**	0.12	0.01	0.07
7	11.00	10.66	10.58	10.51	0.01	0.03	**0**	0.01

TABLE 2.1 – Evaluation séquentielle de l'application ONDES3D sur l'architecture `Malm`.

Résultats sur la plateforme `Borderline`

Concernant la plateforme `Borderline` , la pénalité maximale est de 20.88%. L'écart par rapport à la plateforme `Malm` s'explique par une distance maximale de deux liens *Hypertransport* entre les données et le thread de calcul. Typiquement, un saut d'un lien *Hypertransport* correspond à une pénalité de l'ordre de 10% et un saut de deux liens à une pénalité de 20%. De façon similaire aux expériences sur `Malm` , on peut également retrouver la topologie de l'architecture en identifiant pour chaque ligne, les colonnes avec des valeurs proches de 1%. Par exemple la ligne numéro 4 indique que les cœurs 0 et 4 appartiennent au même processeur dual-cœur (voir l'Annexe B.1.4). Ce raisonnement peut être étendu afin de déduire la topologie complète de la machine.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	**0**	8.97	8.20	18.67	1.14	8.74	7.43	18.73
1	8.01	**0**	18.87	8.24	9.26	0.04	18.39	7.92
2	7.62	18.96	0.62	8.43	9.07	18.91	**0**	7.88
3	19.12	9.10	9.56	**0**	20.88	9.06	8.62	0.38
4	**0**	9.14	8.21	18.85	1.37	9.04	7.55	18.53
5	9.09	**0**	20.12	9.45	10.47	1.60	19.61	8.89
6	7.65	19.33	0.17	8.64	8.43	19.35	**0**	8.46
7	19.67	9.78	9.34	0.75	20.70	9.50	9.13	**0**

TABLE 2.2 – Evaluation séquentielle de l'application ONDES3D sur l'architecture `Borderline`.

Résultats sur la plateforme `Idkoiff`

Les résultats obtenus sur la plateforme `Idkoiff` (Tableau 2.3) sont plus difficiles à analyser et correspondent à une topologie plus complexe sans doute lié à l'imbrication des deux cartes mères composant cette architecture (voir Annexe B.1.5). La valeur maximale observée est de 25.94%. Cette dernière est très proche de la pénalité NUMA maximale mesuré (23.1%) avec le benchmark STREAM [113]. En effet les noyaux de calcul utilisés par ce benchmark sont assez proches des BLAS de niveau 1 et se rapprochent d'une implémentation de la méthode des différences finies.
Cependant, les résultats sont dissymétriques, contrairement aux plateformes `Malm` et `Borderline` . Par exemple, les mesures entre les cœurs 0 et 10 sont assez différentes en fonction d'opérations principalement effectuées en lecture ou des écriture. On peut supposer l'intéraction de certains effets directement attachés à quelques nœds (par exemple les entrées/sorties). La valeur maximale de pénalité NUMA correspond à un écart de trois liens *Hypertransport*. Si on prend le cas des cœurs 0 et 13 distants également de trois liens, une valeur 2 fois plus petite est mesurée (11.3 %). Ces remarques montrent bien la difficulté de la détermination et de l'exploitation des valeurs théoriques caractérisant l'effet NUMA pour une architecture donnée.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
0	3.06	**0**	0.77	2.30	3.37	0.76	3.03	2.82	8.18	7.17	3.49	5.22	10.89	11.37	9.95	8.17
1	0.34	2.19	**0**	1.31	3.36	2.02	2.14	2.33	7.05	6.48	2.87	4.40	10.34	10.47	9.11	9.09
2	2.26	1.42	**0**	1.57	3.40	2.02	2.24	2.30	8.37	7.89	3.24	3.12	11.18	11.18	7.67	6.16
3	2.81	**0**	0.15	1.68	2.98	2.26	2.69	3.91	8.44	7.84	1.76	3.30	11.47	9.82	8.02	8.15
4	12.57	10.56	8.49	10.20	0.90	0.18	**0**	0.16	3.36	2.35	7.10	7.15	9.85	11.52	14.38	10.23
5	11.52	8.94	8.24	9.98	0.95	0.54	**0**	0.17	3.38	2.76	6.42	6.40	11.83	9.87	15.74	10.63
6	14.28	13.05	10.38	12.42	2.86	**0**	2.39	2.01	8.92	8.32	5.35	2.23	13.56	13.00	10.47	12.47
7	11.64	9.06	8.81	10.50	1.15	**0**	0.22	0.52	7.45	5.03	1.92	1.54	11.75	11.50	8.74	8.75
8	11.30	9.75	25.58	25.85	3.25	1.79	14.07	15.58	0.88	**0**	7.52	7.98	8.66	10.30	6.34	5.36
9	9.62	8.43	24.05	24.12	2.58	0.77	14.48	14.25	**0**	5.16	5.57	5.56	9.13	9.26	7.67	5.16
10	25.94	24.62	9.77	11.41	18.24	17.93	0.58	0.44	9.17	8.29	1.62	**0**	4.89	6.35	8.51	9.05
11	25.62	24.28	9.44	9.44	17.23	16.05	2.12	1.86	8.78	7.68	1.52	**0**	5.66	5.88	9.61	9.41
12	14.85	13.66	15.45	14.04	5.69	5.75	5.29	4.00	3.28	1.08	3.62	3.77	1.56	**0**	15.55	8.34
13	14.80	13.39	15.17	14.16	5.50	5.35	5.59	3.64	3.00	2.58	4.98	3.56	1.44	**0**	7.76	6.41
14	12.89	14.19	14.05	11.37	4.50	4.47	2.75	4.57	3.26	2.08	3.75	2.67	5.91	4.07	**0**	3.33
15	11.88	12.22	12.87	11.94	3.52	3.60	3.42	3.57	2.09	1.25	3.32	**0**	3.06	3.10	0.40	0.43

TABLE 2.3 – Evaluation séquentielle de l'application ONDES3D sur l'architecture `Idkoiff`.

2.2.2 Expériences parallèles.

Cette section est consacrée à l'analyse des performances de l'application ONDES3D sur les architectures `Malm`, `Hades`, `Idkoiff`, `Borderline` et `Lias`. Ces plateformes sont décrites en Annexe et proposent différents types d'organisation de la mémoire, elles sont basées sur des processeurs Intel Nehalem (pour `Hades` ou AMD Opteron (pour les autres).

L'implémentation OpenMP standard du code ONDES3D est utilisée pour évaluer l'évolution de la pénalité NUMA en fonction de la taille des données, de la topologie de l'architecture cible ou du schéma d'accès aux données. Les performances sont comparées entre deux versions du code. La première version force un placement local des données en utilisant *sched_ setaffinity* lors des phases d'initialisation et de calcul. Il s'agit d'exploiter la politique *first touch* en forçant une initialisation parallèle. La seconde implémentation est standard et utilise la politique mémoire par défaut *first touch*.

Impact de l'architecture cible

Les cas tests utilisés dans cette partie exploitent la mémoire disponible sur un seul nœud NUMA. La figure 2.1 présente la pénalité maximale mesurée sur les différentes machines. Elle résume les résultats obtenus par rapport à une stratégie de placement standard basée sur la politique par défaut du système d'exploitation. L'impact des caractéristiques NUMA de l'architecture sous-jacente est important sur les plateformes `Idkoiff` (temps de calcul multiplié par un facteur deux) et `Borderline` (53 % de pénalité). Ces deux architectures possèdent la topologie la plus complexe. Pour les plateformes constituées de bi-processeurs dual ou quad-cœur les valeurs mesurées sont comprises entre 10% et 20%. Les performances sur la plateforme `Hades`, à base de processeurs Intel Nehalem, montrent une sensibillté inférieure à la hiérarchie mémoire.
A l'instar des résultats décrits dans la section précédente, il existe une forte variabilité des

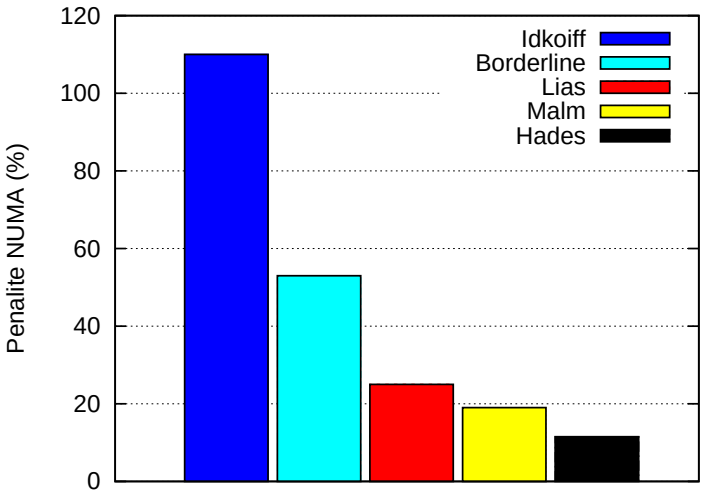

FIGURE 2.1 – Pénalité NUMA pour le code ONDES3D sur différentes architectures hiérarchiques.

performances en fonction du nœud choisi pour placer les données. Par exemple, une différence de plus de 50% est constatée en faisant varier le placement des données sur `Idkoiff` , le même comportement est observé sur le serveur `Borderline` avec des différences de l'ordre de 33%. Encore une fois, les interconnexions spécifiques de chaque nœud (entrées/sorties, ...) semblent

fournir une explication cohérente. En revanche, les architectures `Hades`, `Lias` et `Malm` proposent un comportement extrêmement uniforme des différents nœuds NUMA.

La contention s'exerçant sur le bus mémoire est également importante. Cette dernière provient principalement de l'accès concurrent des différents threads aux données situées sur le même nœud NUMA. Par exemple, c'est la situation induite si le programmeur ne définit aucune de stratégie de placement des données. La topologie des plateformes fournit déjà certaines informations sur le comportement futur. Le nombre de liens ou de chemins entre 2 nœuds distants est un élément important.

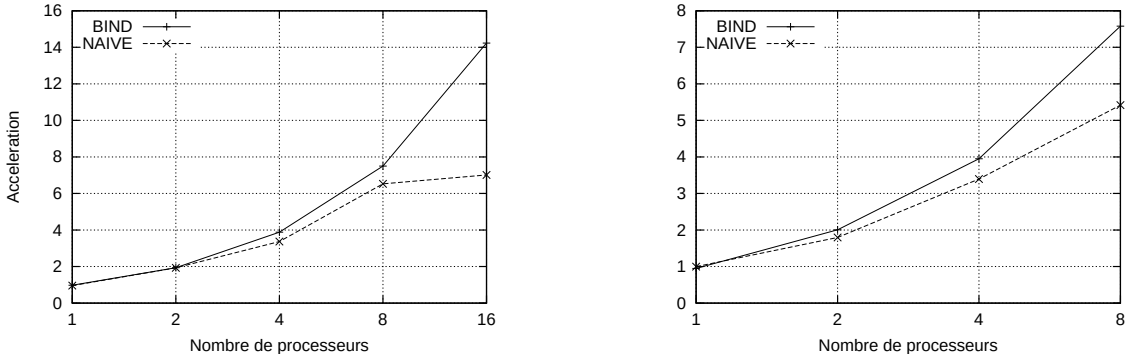

FIGURE 2.2 – Accélérations sur les plateformes `Idkoiff` (gauche) et `Borderline` (droite). Comparaison de la stratégie *first touch* et d'un placement optimisé de la mémoire.

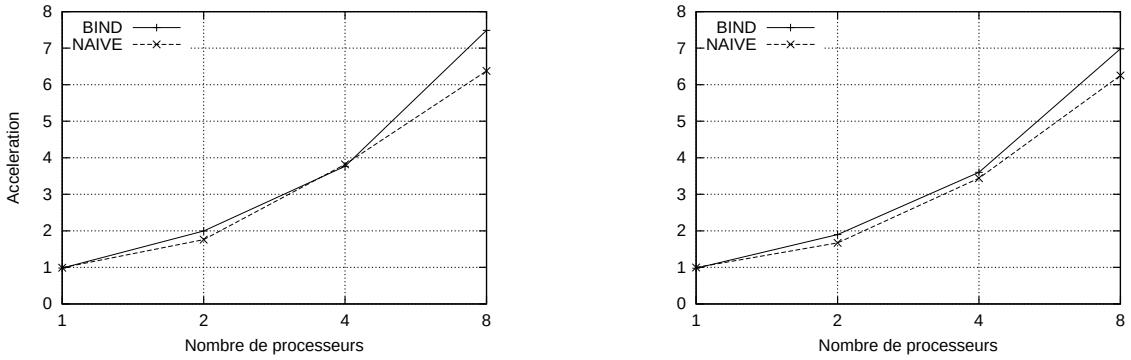

FIGURE 2.3 – Accélérations sur les plateformes `Malm` (gauche) et `Hades` (droite). Comparaison de la stratégie *first touch* et d'un placement optimisé de la mémoire.

Les figures 2.2, 2.3 comparent l'accélération parallèle sur les plateformes `Idkoiff` , `Borderline` , `Malm` et `Hades` pour la politique mémoire par défaut (notation *NAIVE*) et pour le placement coordonné des threads de calcul et des données (notation *BIND*). Ces courbes complètent les résultats précédents car elles expriment également une pénalité NUMA importante sur `Idkoiff` et `Borderline` et des valeurs moins significatives sur `Malm` et `Hades`. De plus, elles soulignent l'effet de l'utilisation de tous les cœurs disponibles. En effet, les résultats sont assez similaires en utilisant uniquement la moitié des ressources disponibles sur chaque machine (en termes de nombre de cœurs). La bande passante disponible est alors utilisée de façon maximale. Au-delà de cette limite la dégradation des résultats est très dépendante des caractéristiques de l'architecture

avec un impact plus faible sur les architectures bi-processeurs (Intel ou AMD).

Le tableau 2.4 est une synthèse et compare les valeurs maximales de pénalité NUMA dans le cas d'exécutions séquentielles et parallèles. L'ensemble des cœurs de chaque machine est exploité dans le cas parallèle. La tendance générale est à une augmentation très nette de la contention avec l'utilisation de l'ensemble des cœurs de chaque machine. Par exemple la pénalité est multipliée par un facteur 4 sur `Idkoiff` et par un facteur 2.5 sur `Borderline`. Sur les configurations plus faiblement hiérarchiques telles que les machines bi-processeurs, l'effet de la contention reste très visible. Les résultats mesurés sur la plateforme `Hades` sont néanmoins très différents de ceux observés sur architectures Opteron. Une réduction de la pénalité NUMA est constatée entre les exécutions séquentielles et parallèles. Les résultats parallèles sont très bons (par rapport à l'architecture `Malm` topologiquement assez proche) et soulignent une bande passante mémoire importante pour les applications parallèles. En revanche, les résultats observés dans le cas séquentiel souligneraient une latence importante entre les deux nœuds NUMA de `Hades`. Ces mesures sont à étendre afin de valider ces premières hypothèses.

Pénalité NUMA maximale		
	Cas séquentiel (%)	Cas parallèle (%)
`Idkoiff`	27	110
`Borderline`	21	53
`Malm`	11	19
`Lias`	13	25
`Hades`	25	11.5

TABLE 2.4 – Effet de la contention sur les valeurs de pénalité NUMA. Comparaison des cas séquentiels et parallèles sur différentes plateformes.

Impact de la taille du problème

Jusqu'ici, nous avons utilisé des cas synthétiques occupant la mémoire disponible sur un seul nœud NUMA. C'est une situation particulièrement défavorable, mais elle permet d'avoir une évaluation de la pénalité NUMA maximale. Des situations plus courantes dans le domaine du calcul scientifique sont analysées dans cette section avec des données distribuées sur plusieurs nœuds NUMA. L'objectif est de mesurer l'effet de la taille des données. On peut supposer que la pénalité NUMA sera moindre avec une meilleure répartition des données sur l'ensemble des nœuds. Cependant, la bande passante mémoire peut être rapidement saturée dans ce cas. De plus, la stratégie par défaut de placement des données ne permet pas de connaitre a priori la répartition des pages mémoire sur les différents nœuds. Un déséquilibre entre nœuds en termes d'allocation mémoire peut alors apparaître lorsque la machine n'est pas complètement chargée. Les expériences menées prennent en compte différentes situations. On considère trois cas qui correspondent à une occupation de la mémoire disponible sur un seul nœud NUMA (notation *1 nœud* sur les graphiques), sur plusieurs nœuds (typiquement la moitié de la machine - notation *plusieurs nœuds* sur les graphiques) ou à la saturation de l'ensemble de la machine (notation *machine*).

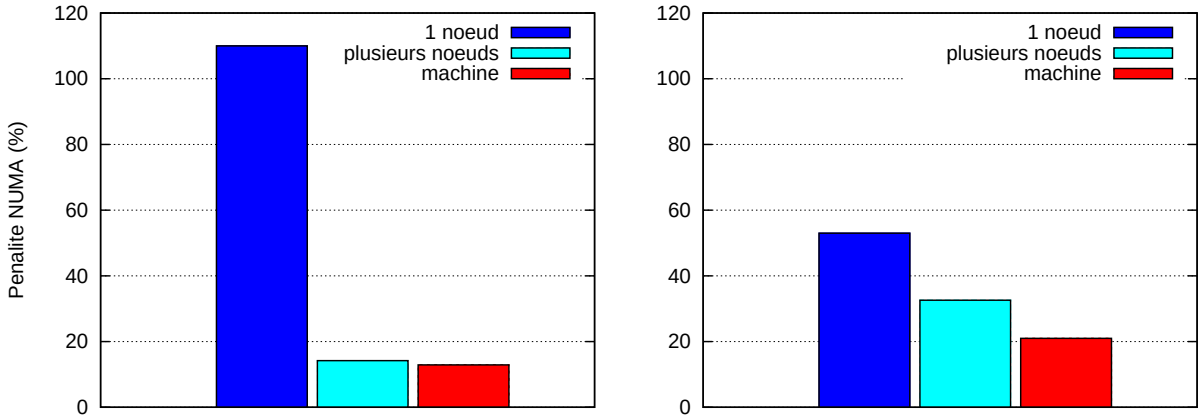

FIGURE 2.4 – Impact de la taille des données sur la pénalité NUMA, les mesures sont effectuées sur les plateformes `Idkoiff` (gauche) et `Borderline` (droite)

Les figures 2.4 et 2.5 présentent les résultats obtenus sur les plateformes `Idkoiff`, `Borderline`, `Malm` et `Hades`. L'augmentation de la taille des données conduit à une diminution importante de la pénalité NUMA. Ce résultat était attendu. La différence est très significative sur les architectures `Idkoiff` et `Borderline`. Par exemple sur `Idkoiff` , la pénalité passe de 110 % à des valeurs comprises entre 15% et 20% lorsque la machine est chargée. Les résultats suivent la même tendance sur `Borderline` avec une corrélation plus forte selon l'occupation mémoire. Les performances ne sont pas similaires sur les machines bi-processeurs telles que `Hades` et `Malm`. Globalement les résultats sont assez proches dans les deux cas, l'occupation mémoire semblant avoir un impact limité. Ces résultats peuvent s'expliquer par la distance plus faible entre les threads de calcul et les données, comparativement aux architectures composées de 4 ou 8 nœuds NUMA. Statistiquement, un thread a donc moins de chance d'accéder à des données distantes. De plus, le prix à payer pour accéder à ces données est beaucoup plus faible.

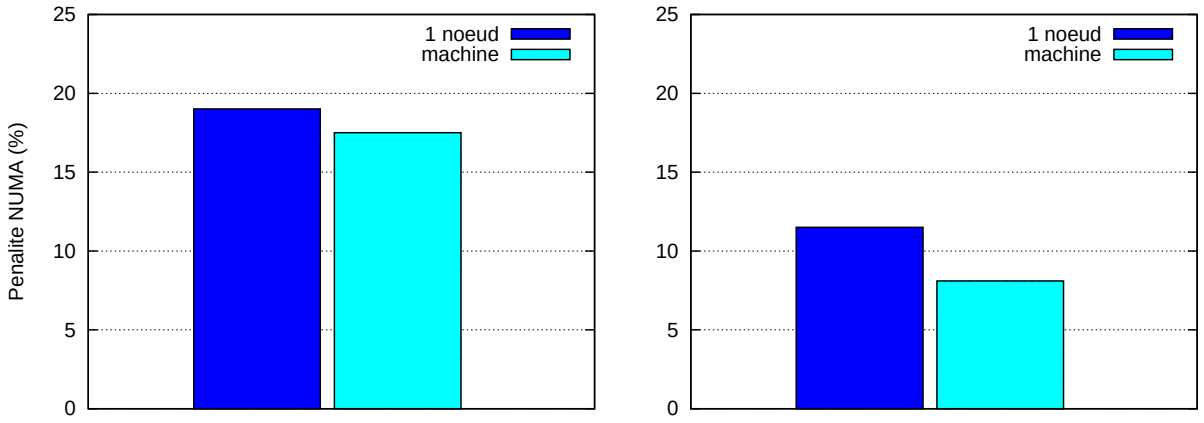

FIGURE 2.5 – Impact de la taille des données sur la pénalité NUMA, les mesures sont effectuées sur les plateformes `Malm` (gauche) et `Hades` (droite)

Impact du schéma d'accès aux données

Nous avons détaillé dans le chapitre précédent la mise en œuvre d'algorithmes d'équilibrage de charge, ces derniers induisant des accès irréguliers à la mémoire. Ce changement dans le schéma d'accès aux données est dû aux stratégies d'ordonnancement dynamique et il impacte nécessairement les valeurs de pénalité NUMA mesurées précédemment.

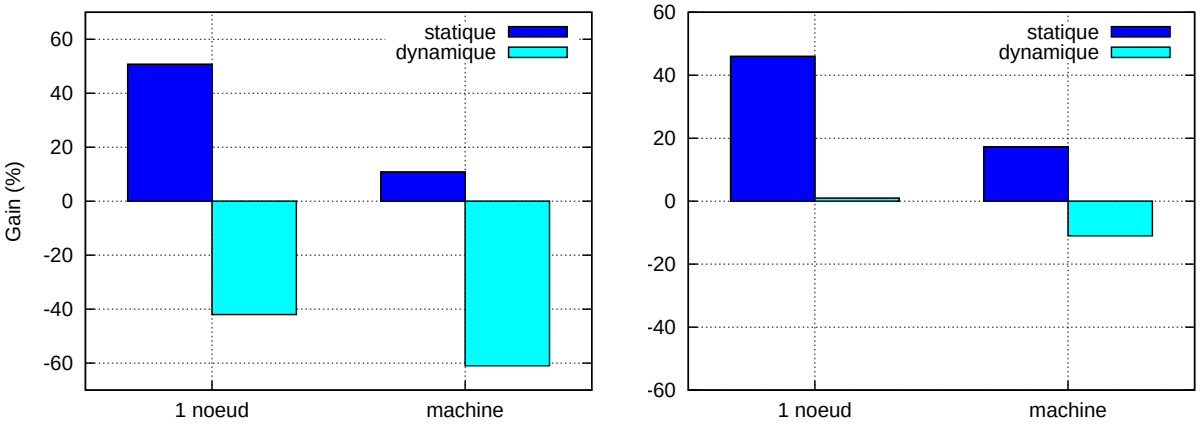

FIGURE 2.6 – Impact du schéma d'accès aux données sur la pénalité NUMA, les mesures sont effectuées sur les plateformes `Idkoiff` (gauche) et `Borderline` (droite)

La figure 2.6 présente à gauche les résultats obtenus sur la plateforme `Idkoiff`. Nous comparons deux versions du code ONDES3D utilisant respectivement un ordonnancement statique ou dynamique au niveau du découpage des boucles OpenMP. Les bords absorbants sont supprimés pour ce test afin de limiter les intéractions avec la correction du déséquilibre de charge qui est inégale entre les deux implémentations. Le cas test choisi permet de maintenir les données sur un seul nœud NUMA. Sans surprises, l'approche consistant à "visser" les threads et les données sur les différents cœurs est la plus efficace dans le cas statique. Dans le cas où les données sont allouées sur un seul nœud NUMA, les gains sont proches de 50% par rapport à une implémentation standard exploitant la politique *first touch*.

En revanche, dans le cas d'accès irréguliers aux données, la stratégie précédente n'est plus adaptée. Par rapport à une implémentation standard, les performances se dégradent. Les conclusions sont similaires sur la plateforme `Borderline` lorsque la stratégie de placement des données est inadaptée par rapport au mode d'accès aux données. Sur les plateformes bi-processeurs `Malm` et `Hades` (figure 2.7) la tendance est similaire avec des gains très faibles dans le cas dynamique.

Dans le cas d'accès réguliers à la mémoire, le placement statique permet d'optimiser la latence minimisant ainsi le nombre d'accès distants. Cette stratégie n'est plus adaptée au cas dynamique avec un placement des données qui ne peut pas correspondre au placement des threads. En effet le découpage et le placement des données est pré-calculé alors que l'ordonnancement des tâches sur les différentes ressources de calcul est dynamique. Les résultats négatifs observés sur `Idkoiff` et `Borderline` et les gains très faibles observés sur `Malm` et `Hades` s'expliquent ainsi. Les caractéristiques plus faiblement NUMA sur `Hades` et `Malm` limitent les écarts mesurés. Globalement un

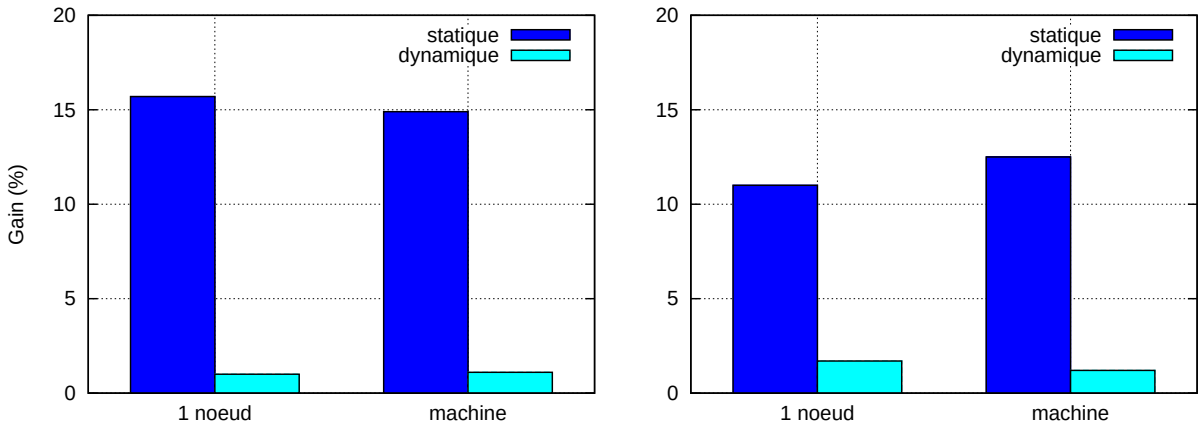

FIGURE 2.7 – Impact du schéma d'accès aux données sur la pénalité NUMA, les mesures sont effectuées sur les plateformes `Malm` (gauche) et `Hades` (droite).

rapport maximum de deux est observé entre une stratégie mémoire adaptée à l'ordonnancement des tâches de calcul et une situation défavorable (exemple `Idkoiff`).

Ces expériences montrent l'importance du choix de la stratégie de placement mémoire en fonction de l'architecture mais également en tenant compte de l'algorithmique de l'application. Par exemple, l'irrégularité des accès mémoire ne sera pas tout à fait de même nature dans le cas d'une implémentation OpenMP ou dans le cas d'une stratégie de surcharge avec des threads MARCEL (Chapitre précédent). La première situation correspond à accès en forme de "tranches ou plans 2D" dans un volume 3D et seule la direction d'espace externe est découpée. Chaque bloc issu de ce découpage est alors traversé de manière classique en respectant le *unit-stride*. Dans le cas de la virtualisation des cœurs et de l'utilisation de la bibliothèque MARCEL, chaque sous-domaine correspond à une "frite" issu d'un découpage horizontal explicite dans les 2 directions externes d'espace.

2.2.3 Limites de l'exploitation de la politique mémoire *first touch*

L'implémentation du code ONDES3D permet de structurer simplement les données et les threads de calcul durant l'exécution. Il suffit de fixer les threads au niveau de la boucle OpenMP d'initialisation et de suivre la même stratégie pour les 2 boucles de calcul (voir la figure 1.5). Une telle approche peut difficilement être envisagée pour une application plus complexe qui ne présenterait pas la même régularité au niveau du schéma d'exécution. De plus, les algorithmes de vol de travail discutés dans le chapitre précédent sont incompatibles avec ce placement mémoire. Par exemple la librairie d'algèbre linéaire PASTIX a nécessité la mise en œuvre d'un algorithme complexe de vol de travail tenant compte du placement initial des données [54]. En fait, cela revient à prendre en compte la structure de l'architecture sous-jacente au niveau de l'application. Des librairies telles que HWLOC (Portable Hardware Locality [30]) peuvent être utilisées afin de faire remonter l'information au niveau applicatif. On pourrait également fixer les threads sur les différents cœurs, au hasard, sans se soucier de la structure hiérarchique de la machine. Outre le faible niveau de performances induit par cette solution, elle offre une flexibilité quasiment nulle au niveau algorithmique.

Les approches basées sur l'exploitation du lien entre les phases d'allocation et les phases de calcul sont présentées comme des solutions alternatives dans un certain nombre de références [147, 99]. Il s'agit de s'appuyer à la fois sur la structure de l'application et sur la stratégie d'allocation mémoire par défaut *first touch*. Les données vont donc être placées à proximité des threads qui vont initialement "toucher" les pages mémoire. Le programmeur peut donc distribuer la phase d'initialisation afin de créer artificiellement une distribution du placement des données sur les différents nœuds NUMA. Dans le cas du code ONDES3D, cela correspond à l'ajout de directives OpenMP au niveau des boucles d'initialisation. Les threads ne sont donc pas explicitement fixés sur les différents cœurs, l'objectif est d'exploiter la régularité de l'application afin d'obtenir un placement des threads identiques entre les phases d'initialisation et les différentes phases de calcul. Les courbes de la figure 2.8 comparent cette technique (notation *PINIT*) à une approche où les threads sont *binder* en utilisant la fonction *sched_setaffinity* (notation *BIND*). Les résultats sont présentés sur les plateformes `Borderline` et `Idkoiff`. Les courbes obtenues sur `Malm` et `Hades` conduisent à des conclusions similaires.

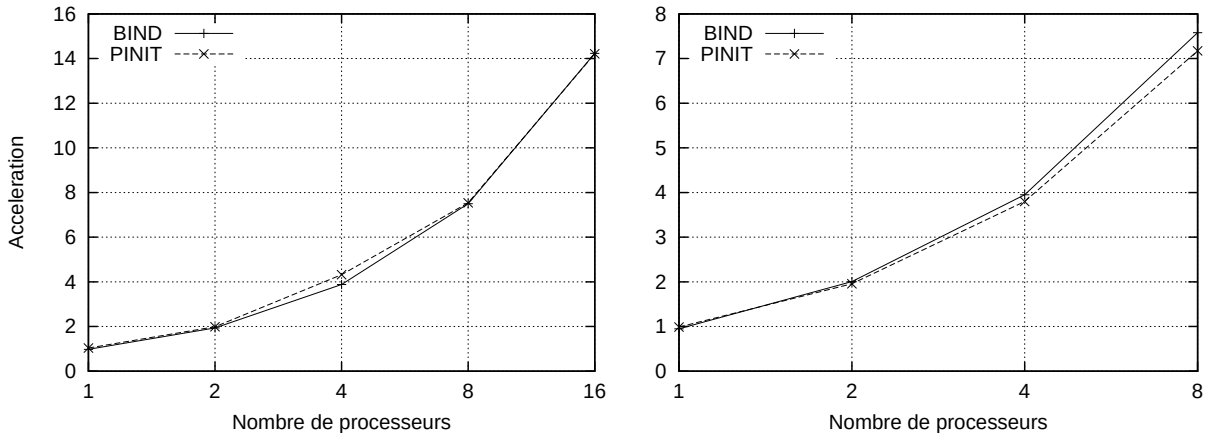

FIGURE 2.8 – Comparaison d'une approche basée sur l'initialisation parallèle (notation *PINIT*) ou sur un placement guidé de la mémoire (notation *BIND*). Evaluation réalisée sur les plateformes `Idkoiff` (gauche) et `Borderline` (droite).

Les résultats moyennés obtenus sont proches avec un écart inférieur à 5%. Néanmoins, les mesures réalisées dans le cas *PINIT* présentent une certaine volatilité (entre 10% et 20%). En effet l'ordonnanceur peut décider de déplacer les threads au moment de l'initialisation ou durant le calcul sans aucun contrôle du programmeur, cassant ainsi l'affinité mémoire supposée. Cette solution simple propose donc un premier niveau d'optimisation pour des situations relativement restreintes. De plus, elle ne permet pas d'envisager une intégration correcte d'algorithmes irréguliers.

2.3 Ordonnancement structuré sur architecture hiérarchique.

2.3.1 Contexte et motivations

Les expériences décrites dans cette section sont principalement basées sur la thèse de S.Thibault [149]. La plateforme BUBBLESCHED proposait alors en 2008 un certain nombre de fonctionnalités permettant au programmeur de structurer l'ordonnancement des threads de son application. Dans le premier chapître de ce manuscrit, nous avons présenté les approches retenues pour la conception des environnements BUBBLESCHED et FORESTGOMP [31]. Ces travaux sont en cours d'extension, principalement en ce qui concerne la gestion de la mémoire et les mécanismes de migration associés [65]. De même, l'intégration de l'ordonnancement par *bulles* au modèle de programmation OpenMP a évolué concernant la gestion du parallélisme imbriqué et la prise en compte de la hiérarchie mémoire. Les résultats présentés ici sont donc à confronter aux évolutions régulières de ces plateformes. La philosophie de ces environnements est de tirer partie des informations fournies par l'application afin de "coller au mieux" à la hiérarchie de l'architecture sous-jacente. Le concept de *bulles* propose une structure récursive permettant d'induire une hiérarchie de tâches qui pourront ensuite être ordonnancées de manière efficace. Typiquement le programmeur est amener à exprimer le fait que 2 threads travaillent sur le même ensemble de données et qu'il serait judicieux de les garder proche au moment de l'ordonnancement afin de maximiser l'affinité mémoire.

L'application ONDES3D propose un parallélisme structuré. En effet la méthode numérique utilisée repose sur une grille cartésienne et permet une décomposition spatiale régulière. Il est donc aisé de déduire une organisation en sous-domaines contigus et de définir une organisation spatiale du parallélisme. Les threads devant travailler sur une zone mémoire commune sont identifiables durant la phase pré-traitement. De plus, l'organisation des calculs (figure 1.5) est favorable avec une séparation des phases de calcul dédiées à l'évaluation des composantes de vitesse ou de contraintes. Il s'agit maintenant de choisir l'approche la mieux adaptée afin d'exprimer les différents niveaux de parallélisme et de profiter des possibilités d'ordonnancement hiérarchique.

2.3.2 Expression de la hiérarchie des tâches

Plateforme FORESTGOMP

FORESTGOMP permet de créer de façon automatique une hiérarchie de tâches en utilisant les informations provenant de l'imbrication des différents niveaux de parallélisme OpenMP. Dans le cas de l'application ONDES3D, une décomposition 2D du domaine de calcul est immédiate à mettre en œuvre. En effet, le niveau externe de parallélisme permettra la création des bulles correspondant à la première direction spatiale de découpage. Les threads créés à l'intérieur de chaque bulle correspondent alors à la seconde direction de décomposition. La section 1.3.1 a permit d'illustrer l'efficacité de FORESTGOMP par rapport à la librairie GNU OpenMP. Les tests effectués sur une boucle synthétique de type Jacobi et sur l'application ONDES3D confirmant ainsi des gains significatifs (un rapport supérieur à 3 par rapport à la librairie GOMP dans le cas de l'imbrication de deux niveaux de parallélisme).

A priori, la plateforme FORESTGOMP offre donc un ensemble de fonctionnalités bien adaptées aux problématiques de la modélisation sismique en différences finies sur architectures NUMA et multicœurs. Le placement efficace des données et les problèmes d'équilibrage de charge peuvent être pris en charge de façon transparente en se basant sur le modèle de programmation OpenMP.

Notons que l'efficacité réelle du mécanisme de migration automatique employé par l'ordonnanceur *memory* est encore à valider dans le contexte applicatif. En effet, en fonction de la fréquence de ces migrations et du volume de données à déplacer, cette stratégie perdra une partie de son intérêt. La politique mémoire *next touch* adoptée pourrait alors conduire à un surcoût important par rapport au mécanisme de virtualisation implémentée.

Néanmoins, cette approche n'a pas été approfondie car le modèle de programmation OpenMP constitue une limitation importante par rapport aux objectifs fixés initialement. En effet, nous souhaitons aborder de façon concurrente les aspects équilibrage de charge, les problématiques d'affinité mémoire et la mise en œuvre d'algorithmes fortement irréguliers pour la réduction du trafic mémoire. Dans cette optique, le modèle de programmation OpenMP offre peu de flexibilité pour l'expression d'algorithmes plus complexes qui ne reposeraient pas sur des nids de boucles.

Plateforme Bubblesched

Une autre possibilité pour l'exploitation de l'ordonnancement par *bulles* est la création explicite des structures correspondantes au niveau applicatif. Cette approche est certes plus intrusive, mais elle permet de conserver un modèle de programmation souple. Par rapport au schéma régulier d'accès aux données dans le cas statique, l'ordonnanceur *Explode*, décrit dans la thèse de S.Thibault [149], semble un bon candidat afin de structurer les différents niveaux de parallélisme. Ce dernier permet au programmeur de définir le niveau d'éclatement souhaité pour chaque *bulle*. C'est une solution simple, permettant de mesurer l'impact de l'intégration de la plateforme en termes de modification du code initial et d'évaluer rapidement les performances. Cet ordonnanceur est également interessant car il offre au programmeur un controle important sur les stratégies d'ordonnancement mises en œuvre.

La figure 2.10 résume le modèle d'exécution souhaité avec un premier niveau d'ordonnancement des *bulles* sur les différents nœuds NUMA permettant de distribuer les données. Un second niveau sur les différents cœurs permet de gérer les threads de calcul. En termes de découpage, chaque *bulle* correspond à un sous-domaine issu d'un partitionnement horizontal régulier, les threads de calcul pouvant être vus comme des micro-domaines de calcul rattachés aux macro-domaines (voir la figure 2.9).. Le nombre de lignes de code à insérer dans l'application ONDES3D est relativement faible. En effet, le partitionnement du domaine de calcul existe déjà pour le mécanisme de virtualisation à partir des threads MARCEL. Les principales fonctions à rajouter sont la création (*marcel_bubble_init*) puis l'intégration des *bulles* définissant la hiérarchie (*marcel_bubble_insertbubble*). Les différents niveaux d'ordonnancement sont définis avec la fonction (*marcel_bubble_setschedlevel*). Dans notre cas, une cinquantaine de lignes de code permettent d'intégrer ces différentes fonctionnalités.
En fait, le modèle d'exécution décrit est assez proche d'un modèle MPI. Hors mécanisme explicite, les données du problème ne sont pas partagées entre *bulles* adjacentes. Cependant, le stencil différences finies implémenté nécessite un partage d'informations entre sous-domaines (et donc entre *bulles*) voisins. La solution la plus simple est la mise en place d'une zone mémoire commune aux différentes bulles constituant une zone tampon pour l'échange des informations à l'interface. Cette zone mémoire ne bénéficie donc pas de l'optimisation du placement des données sur la hiérarchie de la machine. Le nombre de bulles ayant vocation à être limité et cette zone mémoire étant uniquement dédiée aux informations d'interface, ce tampon ne constitue pas un goulot d'étranglement dans l'analyse des performances. Nous verrons néanmoins que ce point est à affiner car pouvant constituer une limitation à termes.

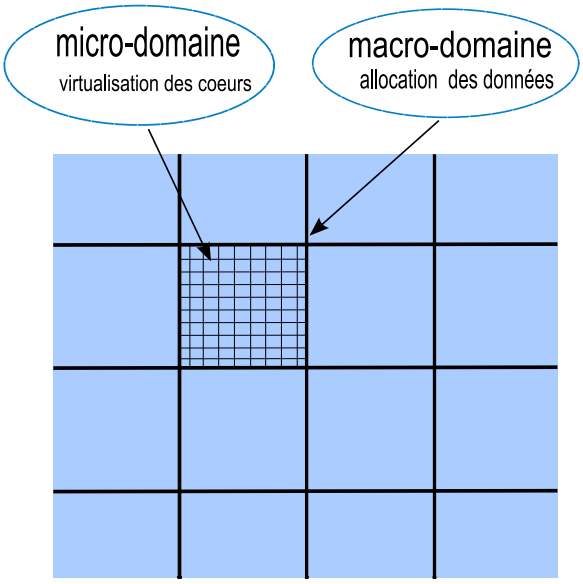

FIGURE 2.9 – Partitionnement du domaine de calcul. Les macro-domaines correspondent aux threads d'allocation sur les nœuds NUMA et les micro-domaines au threads de calcul.

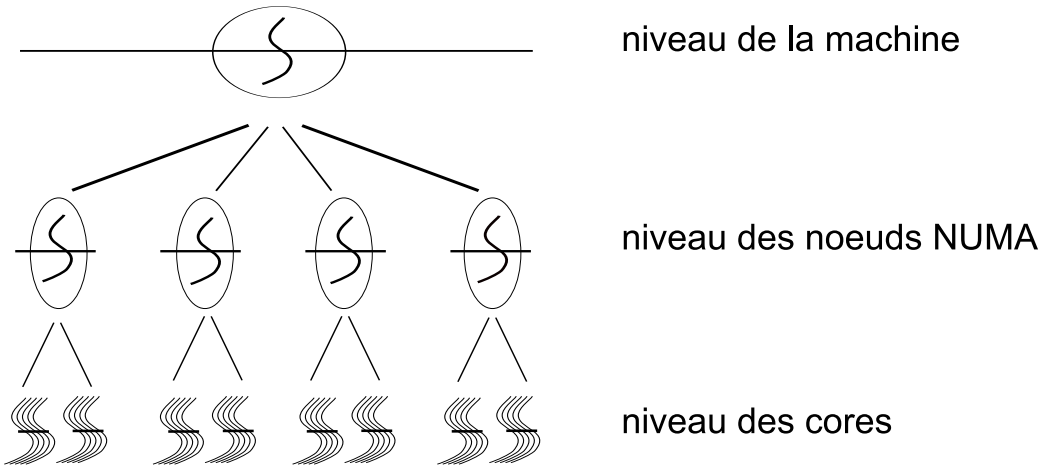

FIGURE 2.10 – Exemple d'ordonnancement en situation de surcharge sur une machine comprenant quatre nœuds NUMA.

2.3.3 Analyse des résultats et discussion

Analyse des résultats

L'exemple simplifié de modélisation sismique décrit dans la section 2.2.2 est repris ici. Le serveur `Idkoiff` est utilisé pour ces tests en saturant toute la mémoire disponible et en exploitant

les 16 cœurs disponibles. Nous comparons deux implémentations exploitant à la fois la technique de virtualisation des cœurs avec les threads MARCEL et l'ordonnancement par *bulles* (selon le schéma 2.10). La différence entre ces deux implémentation est le niveau choisi pour l'allocation des données.

- Dans le premier cas, les données sont allouées au niveau global de la machine (niveau du thread *root*). Dans ce cas, les threads de calcul sont bien regroupés par affinité, mais les données ne sont pas prises en compte et la mémoire est placée sur les différents nœuds NUMA en utilisant la politique mémoire par défaut.

- Dans le deuxième cas, les données sont allouées au niveau des nœuds NUMA permettant de maximiser la localité mémoire.

Les résultats obtenus sont résumés par le tableau 2.5. Ces derniers sont moyennés sur dix exécutions permettant de tenir compte du caractère non reproductif des simulations basées sur la politique mémoire *first touch*.

Nombre de threads	16	32	64	128	256	512
Gain (%)	13.1	10.3	16.2	26.2	27.5	37.4

TABLE 2.5 – Gain en utilisant une allocation des données par *bulles* par rapport à la politique mémoire *first touch* dans le cas d'un ordonnancement structuré des threads de calcul. Les tests sont effectués sur la plateforme `Idkoiff`.

La situation avec 16 threads correspond à une répartition statique des threads de calcul sur les différents cœurs. La comparaison s'effectue donc entre un cas où le placement des données et des threads est coordonné et un cas reposant sur la politique *first touch*. Les résultats sont meilleurs pour l'approche coordonnée avec un gain de l'ordre de 13%, cela correspond également aux résultats décrits dans la section 2.2.2 dans un contexte similaire.

Lorsque le nombre de threads de calcul augmente, les gains augmentent également jusqu'à atteindre un maximum de 37% avec 512 threads sur 16 cœurs. Cette situation provient de la contention sur la bande passante qui augmente avec le nombre de threads. Par exemple, plus le sous-domaine de calcul sera petit, plus la probabilité sera élevée que l'ensemble des données qui lui sont utiles soient distantes. Dans le cas de la stratégie d'allocation *first touch*, cette situation conduit donc à un trafic croissant au niveau de la bande passante mémoire. De manière générale, le mécanisme de surcharge augmente l'irrégularité dans les accès aux données générant des accès à la mémoire plus fréquents et plus difficiles à optimiser.

Discussions

La stratégie choisie pour notre implémentation limite les possibilités de régulation de la charge au niveau d'un nœud NUMA et non pas de la machine. La mise en œuvre est donc préliminaire et l'utilisation d'un ordonnanceur tel que *memory* permettrait la prise en compte de la hiérarchie de l'architecture. Une *bulle* pourrait ainsi voler du travail sur un nœud distant et de façon symétrique attirer les données associées.

Les figures 2.11 et 2.12 décrivent deux modèles d'exécution pour le code ONDES3D.

– Dans le premier cas, il s'agit de l'implémentation basée sur l'ordonnanceur *Explode* avec une gestion explicite de la mémoire et une allocation effectuée par le programmeur au niveau des *bulles*. Cette approche est relativement peu souple car elle ne permet pas de lier dynamiquement l'ordonnancement des *bulles* au placement des données.

– Le deuxième schéma décrit une implémentation reposant sur l'ordonnanceur *memory* et la librairie MAMI [65]. Cette dernière permet d'attacher de la mémoire au niveau des *bulles* avec l'opportunité de lier les données et l'ordonnancement des threads. A l'instar de l'intégration de ce mécanisme dans la plateforme FORESTGOMP, l'efficacité des phases de migration est déterminante. L'allocation des données peut se faire ici au niveau du thread root durant la phase d'initialisation. Le découpage du domaine de calcul étant connu de façon précise, il est aisé d'attacher les bonnes zones mémoires aux *bulles*.

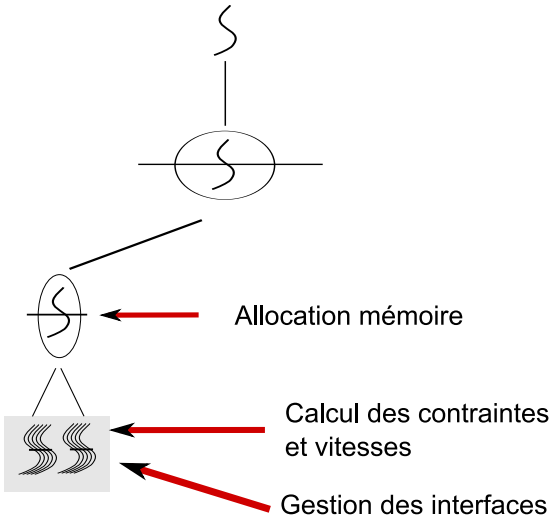

FIGURE 2.11 – Schéma d'une implémentation exploitant l'allocation mémoire au niveau des *bulles*.

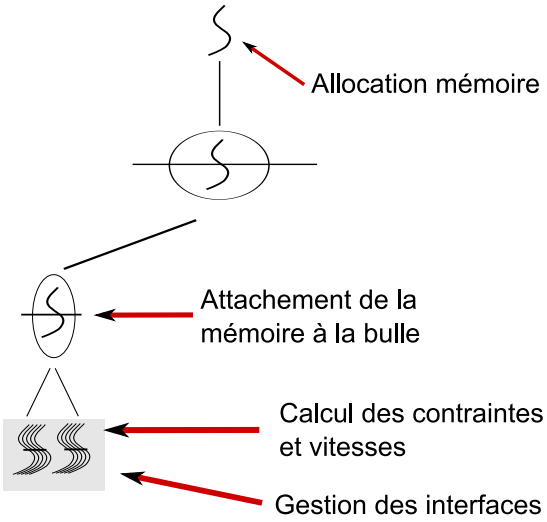

FIGURE 2.12 – Schéma d'une implémentation reposant sur la librairie MAMI.

Quelle que soit la stratégie choisie, un certain nombre de questions demeurent. Tout d'abord, la gestion des données à partager entre les différentes *bulles* doit être abordée. Dans les deux cas décrits précedemment, une zone mémoire explicitement partagée entre les différents threads de calcul doit être définie. De même, l'intégration des phases de communication au niveau du modèle d'exécution doit être abordée. Il parait peu envisageable d'implémenter ces communications au niveau de la phase de surcharge, des plateformes telles que MPC sont optimisées pour ce type d'exécution. La possibilité de recréer les threads de surcharge ou encore de les bloquer sur une condition, à chaque étape en temps, a été testée sans différence significative en termes de performance. Une possibilité serait donc de revenir au niveau du thread root de chaque *bulles* pour les phases de communication.

2.4 Une stratégie orientée vers l'application : évaluation de la plateforme MAi

2.4.1 Intégration au niveau de l'application Ondes3D

Par rapport aux stratégies d'ordonnancement par *bulles* décrites précédemment, la librairie MAi est également assez peu intrusive. En effet, quelques lignes de code suffisent à redéfinir la politique d'allocation mémoire. La figure 2.13 décrit les fonctions utilisées et leur niveau d'insertion dans l'application, une seule composante de vitesse étant décrite dans notre schéma. MAi propose des fonctions permettant l'allocation de tableaux 1D, 2D ou 3D. La fonction *mai_ alloc_ 3D* permet d'allouer les tableaux tridimensionnels de l'application Ondes3D en masquant les opérations de décalage de pointeurs afin de disposer d'une zone mémoire simple à manipuler. On retrouve en fait la même stratégie que la fonction *d3tensor* disponible dans les librairies *Numerical Recipies* utilisées dans la version originale du code. La politique mémoire est ensuite appliquée de façon indépendante (l'approche cyclique dans notre exemple).
Pour le programmeur, la difficulté réside dans le choix de l'approche la plus efficace par rapport

```
int main()
{
mai_init();
...
vx = mai_alloc_3D(nx,ny,nz,DOUBLE);
mai_cyclic(vx);
...

do t=1,Tfinal
   calcul_contraintes
   calcul_vitesses
enddo
.....
mai_final();
}
```

Figure 2.13 – Intégration des appels MAi dans le code Ondes3D.

à son application et à l'architecture sous-jacente. Nous avons vu dans les paragraphes précédents que de multiples paramètres peuvent influer sur la pénalité NUMA (architecture de la machine, taille des données, schéma d'accès aux données ...).

MAi permet de découpler l'ordonnancement des threads et la politique de placement des données (hormis pour les politiques de type *BIND*), le programmeur devant donc effectuer des choix cohérents entre ces 2 niveaux. Par exemple, dans le cas d'un ordonnancement dynamique, la politique mémoire optimale sera probablement différente de celle sélectionnée dans un cas statique. De plus, en fonction du choix d'optimiser la bande passante ou la latence, les modalités de placement seront différentes.

Dans sa version statique, l'application Ondes3D se caractérise par la régularité du schéma d'accès aux données. Dans ce cas, les stratégies mémoire viseront donc principalement à rapprocher les données des threads de calcul. Les stratégies *BIND* paraissent les mieux adaptées dans ce contexte. Il s'agira de réduire, et même dans ce cas de supprimer, les accès mémoire

distants et d'optimiser la latence. Ce point est particulièrement sensible sur les architectures *BULL Novascale* [134].

Les algorithmes décrits dans la section précédente pour corriger le déséquilibre de charge (virtualisation ou directives OpenMP dynamique) sont inadaptées à un placement statique des données. Une piste consiste alors à optimiser l'utilisation de la bande passante en effectuant une distribution cyclique ou bloc cyclique des données. Ces approches permettent de maximiser les chances qu'une donnée utile à un thread lui soit proche. Le grain utilisé pour la répartition est ainsi mieux adapté aux situations dynamiques contrairement aux placements statiques qui répartissent de gros blocs de données sur les différents nœuds.

2.4.2 Evaluation des performances

Les sections précédentes ont souligné le lien entre l'accès aux données et la politique d'allocation mémoire choisie. Nous évaluons le gain obtenu avec MAi en utilisant un placement cyclique des données. La figure 2.14 correspond à une version simplifiée du code Ondes3D n'implémentant pas les bords absorbants, cela permet de se focaliser sur l'affinité mémoire. La taille des données permettent de saturer entièrement la mémoire disponible sur les architectures `Idkoiff`, `Borderline` et `Malm`.

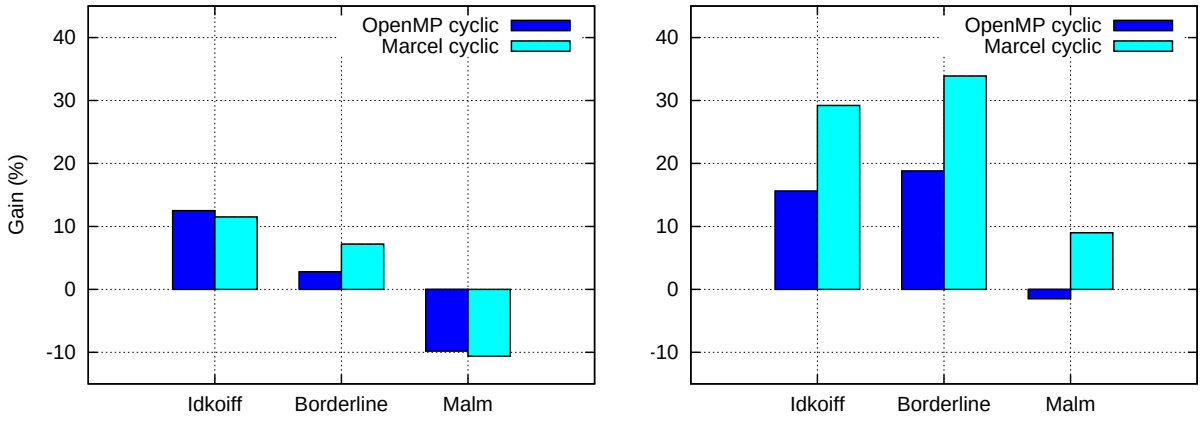

Figure 2.14 – Comparaison de différentes politiques mémoire pour les implémentations OpenMP et Marcel. La courbe de gauche correspond à un ordonnancement statique des threads et celle de droite à ordonnancement dynamique.

La courbe de gauche de la figure 2.14 compare l'utilisation de la politique mémoire cyclique pour les versions statiques OpenMP et Marcel (dans ce cas le nombre de threads est égal au nombre de cœurs de l'architecture) à l'implémentation OpenMP basée sur un placement précalculé des threads et des données. Cette dernière option proposait jusqu'ici les meilleurs résultats dans le cas d'accès réguliers aux données.

Les gains mesurés sur la plateforme `Idkoiff` sont relativement modestes. On observe 12.5 % de gain au niveau du temps d'exécution entre les 2 versions OpenMP (politiques *BIND* et *CYCLIC*) et 11.4% de gain par rapport à l'implémentation utilisant les threads Marcel. La même tendance est observée sur la plateforme `Borderline` avec de meilleurs résultats pour la

version basée sur les threads MARCEL. Sur `Malm`, on observe une dégradation des performances en utilisant une politique cyclique par rapport à l'approche pré-calculée.

Intuitivement la stratégie *BIND* semble la plus adaptée au cas statique et les gains attendus avec l'utilisation d'un placement cyclique des données sont donc faibles. Néanmoins, les gains sur `Borderline` et `Idkoiff` traduisent l'importance de l'optimisation de la bande passante par rapport à la latence sur ces architectures. Ce critère étant moins prépondérant sur `Malm` (architecture bi-processeur faiblement hiérarchique), la politique cyclique est peu performante. Globalement, le comportement des implémentations MARCEL ou OpenMP est assez similaire.

La courbe de droite de la figure 2.14 compare également les versions OpenMP et MARCEL de l'application sismique, cette fois l'ordonnancement des threads sur les différents cœurs est dynamique. Nous utilisons 1024 threads dans le cas de la stratégie de surcharge. La politique mémoire par défaut *first touch* est prise comme référence, cette dernière fournissant jusqu'ici les meilleurs résultats dans le cas où l'accès à la mémoire était irrégulier. Un gain maximum de 34% sur `Idkoiff` et 9% sur la plateforme `Malm` est mesuré en exploitant le placement cyclique des données. Les gains sont plus élevés avec l'utilisation des threads MARCEL par rapport à la version OpenMP. L'explication vient en partie d'un léger effet de cache dû a la réduction de la taille des sous-domaines dans le cas de la virtualisation. De plus, la politique d'allocation mémoire cyclique est sans doute mieux adaptée au caractère fortement irrégulier du placement des threads par rapport à l'ordonnancement dynamique avec OpenMP reposant sur un découpage suivant une seule direction d'espace.

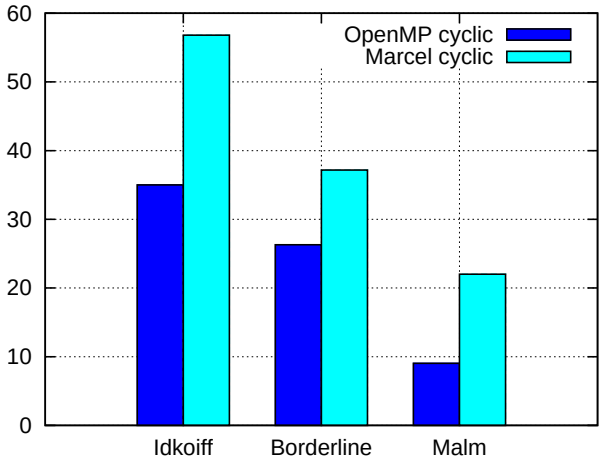

FIGURE 2.15 – Gain par rapport à une version standard en combinant la surcharge et le placement mémoire optimisé pour l'application ONDES3D.

La figure 2.15 présente les résultats de la dernière expérimentation menée. La version complète de l'application est utilisée avec l'implémentation des bords absorbants. Dix points de grille sont utilisés pour la mise en œuvre de la couche absorbante et nous évaluons le gain cumulé obtenu en combinant à la fois le vol de travail et la pénalité NUMA. Par rapport à un version standard du code, les gains sont de l'ordre de 56.8% sur la plateforme `Idkoiff` . Pour rappel, la pénalité NUMA dans cette situation est proche de 17%. En fait, l'implémentation MARCEL dynamique offre de meilleures performances que la version OpenMP dynamique dans tous les cas de figure. L'écart observé s'explique par les mêmes arguments que l'expérience précédente. De

plus, l'intégration des mécanismes de vol de travail et des politiques offertes par la bibliothèque MAI semble plus efficace. Sur les plateformes `Borderline` et `Malm` les gains sont respectivement de 37% et 22%. Ces plateformes étant a priori moins hiérarchiques, la pénalité NUMA à corriger est modérée (principalement dans le cas de `Malm`). Le nombre inférieur de cœurs (8 au lieu de 16) induit également un déséquilibre de charge plus faible comparativement à `Idkoiff`.

Chapitre 3

Décomposition espace-temps et réduction du trafic mémoire

Contents

La faible exploitation de la puissance offerte par les processeurs est une limitation importante des applications basées sur la méthode des différences finies . En effet le caractère *memory bound* de cette méthode numérique limite en général les performances à 20% à 30% de la puissance crête. Ce problème est plus large que le domaine de la modélisation sismique car les noyaux de calcul impliqués sont utilisés dans de nombreux domaines applicatifs conduisant à la résolution d'équations aux dérivées partielles (EDP). Ces techniques se retrouvent également en algèbre linéaire avec des méthodes classiques telles que les procédures itératives de Jacobi ou de Gauss-Seidel. La figure 4 illustre une procédure de Jacobi (identique à une procédure de type différences finies d'ordre 2). Les performances obtenues lors de l'implémentation des stencils différences finies

Algorithm 4 Pseudo-code pour une implémentation standard d'un stencil à 7 points de type Jacobi.

for $i = 1$ to nx **do**
  for $j = 1$ to ny **do**
    for $k = 1$ to nz **do**
      $X^{n+1}(i,j,k) = X^n(i,j,k) + X^n(i,j,k+1) + X^n(i,j,k-1) + X^n(i,j+1,k)$
      $+X^n(i,j-1,k) + X^n(i+1,j,k) + X^n(i-1,j,k)$
    end for
  end for
end for

sur architectures multicœurs sont donc très faibles. La bande passante mémoire disponible sur

ces architectures en est la principale cause. En effet le design de ces nœuds multiprocesseurs conduit à une augmentation très rapide du nombre d'unités de calcul sans une augmentation équivalente de la bande passante mémoire. L'aspect *memory bound* de ces algorithmes est donc d'autant plus critique. L'introduction de différents niveaux de hiérarchie mémoire est un facteur aggravant car chaque accès mémoire peut correspondre à un accès mémoire distant. La mise en œuvre efficace sur architectures hiérarchiques et multicœurs de la méthode des différences finies est donc un sujet ouvert, les approches discutées dans le premier chapitre devant être adaptées aux architectures sous-jacentes pour tirer le meilleur des supports d'exécution disponibles. C'est l'objet de ce chapitre qui introduit un algorithme de décomposition dans le domaine espace-temps adapté à la méthode des différences finies pour l'élastodynamique en prenant en compte les aspects hiérarchiques et multicœurs des nœuds de calcul.

3.1 Approches séquentielles

3.1.1 Limites des techniques de décompositions spatiales

L'implémentation standard des noyaux de calcul de type différences finies conduit à un niveau de performance proche de celui observé pour les routines BLAS de niveau 1 ou 2. Par exemple pour un stencil tridimensionnel d'ordre 2, il est nécessaire d'accéder aux 6 points voisins. Ces derniers sont distants respectivement de 1, Nz et $Ny \times Nz$ emplacements mémoire avec Z définissant la direction d'accès privilégiée.
La figure 3.1 illustre les accès mémoire dans ce cas avec $6 \times Nx \times Ny \times Nz$ opérations à effectuer

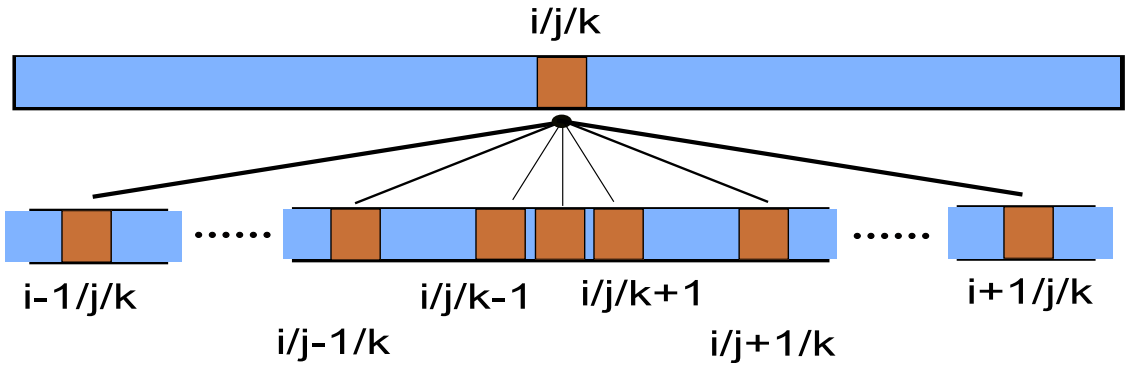

FIGURE 3.1 – Accès mémoire dans le cas d'une boucle de Jacobi en 3D.

sur un problème de taille $Nx \times Ny \times Nz$. Le ratio entre les calculs et les accès mémoire est en $O(1)$ alors que pour des BLAS de niveau 3, ce ratio est en $O(n)$ permettant ainsi une bien meilleure réutilisation des données présentes dans le cache. L'approche proposée par Rivera et Tseng [135] est basée sur l'idée qui consiste à bloquer deux directions spatiales et à utiliser la troisième afin d'accumuler des plans 2D. Cette stratégie permet d'éviter la décomposition en blocs tridimensionnels de très petite taille. Pour des algorithmes de type différences finies, ces blocs offriraient alors très peu de possibilité de réutilisation des données présentes dans la mémoire cache. La figure 3.2 détaille cette approche avec la direction Z comme direction de translation.
Nous proposons de mettre en œuvre cette approche sur un architecture à base de processeurs Intel *Itanium* afin d'évaluer les gains potentiels en termes de réduction du taux de défauts de cache et d'illustrer le lien entre ce gain et la réduction de la pénalité NUMA. Cette plateforme permet également une remontée précise des informations provenant des compteurs hardware. Dans un

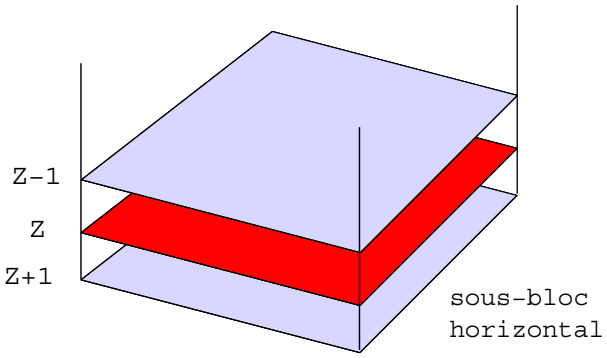

FIGURE 3.2 – Schéma de l'algorithme de Rivera et Tseng basé sur l'accumulation des plans 2D dans la direction verticale.

premier temps, nous évaluons le nombre de plans à accumuler dans la mémoire cache afin de pouvoir observer une accélération. En effet, le parcours du domaine tridimensionnel est désormais basé sur la réutilisation des données du plan Z par le plan Z+1. Il faut donc maximiser le nombre de plans contenus dans la mémoire cache. Nous implémentons une boucle tridimensionnelle de Jacobi ; le domaine de calcul est de taille $1024 \times 1024 \times 512$ et permet d'occuper la moitié de la mémoire disponible sur un nœud NUMA (QBB) de la machine `Teranova` (voir Annexe B.1.1). La courbe de gauche de la figure 3.3 décrit le taux de défauts de cache par rapport au nombre

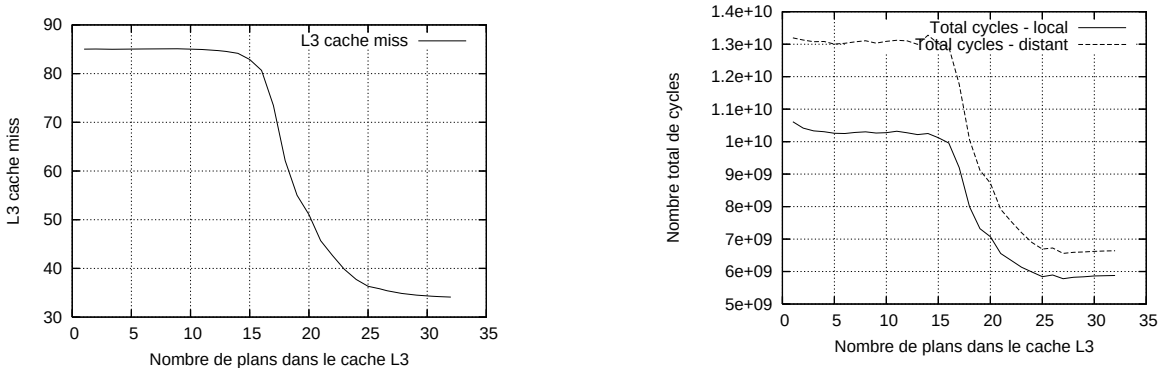

FIGURE 3.3 – Taux de défauts de cache (gauche) et nombre total de cycles (droite) pour l'algorithme de Rivera et Tseng.

de plans résidant dans le cache de niveau L3. Sur cette architecture qui propose un cache L3 de 3 Mo, les performances commencent à s'améliorer à partir d'une quinzaine de plans. Le taux de défauts de cache passe alors de 80% à moins de 40% sur notre problème. De manière symétrique, l'évolution du nombre de cycles avec le nombre de plans résidant dans la mémoire cache illustre une réduction de près de 50% du temps de calcul (courbe de droite de la figure 3.3 - cas local). Cela montre bien le lien existant entre le taux de défauts de cache prédominant pour cet algorithme et les performances observées. On peut souligner l'amélioration des performances qui passent d'une moyenne de 1 gigaflops à 1.8 gigaflops, toujours à partir de 15 plans stockés dans le cache L3.

Pour illustrer l'importance de l'algorithmique sur la pénalité NUMA, nous plaçons les données et l'exécution sur 2 nœuds différents en utilisant la *libnuma*. La figure 3.3 illustre les valeurs identiques de défauts de cache dans les deux cas. Ce résultat était attendu car il s'agit du même algorithme, le taux de défauts de cache étant indépendant du placement sur un nœud local ou

distant. En revanche le nombre total de cycles augmente significativement lorsque les données sont placées de façon distante par rapport au thread de calcul. Cela traduit le coût supérieur d'un cache-miss dans le cas distant. Une différence de l'ordre de 30% apparaît dans notre cas. La première partie de la courbe de droite la figure 3.3 correspond à un effet NUMA maximum ; on est ici dans le cas où la réutilisation des données présentes dans la mémoire cache est faible, l'écart entre placement local ou distant étant donc maximale (30%).
Dès que le nombre de plans dans le cache L3 est suffisant pour optimiser la réutilisation des données, l'écart de performance entre un placement local ou distant devient beaucoup plus faible. L'effet de la hiérarchie mémoire sur le nombre de cycles passe de 30% à 5%.

Dans le cas de l'algorithme de Rivera et Tseng pour le problème de l'élastodynamique, il faut stocker neuf vecteurs correspondant aux composantes de vitesse et de contraintes. Les composantes du vecteur force et les paramètres du modèle doivent éventuellement être rajoutées. C'est donc plus de 12 composantes qu'il faut stocker pour constituer des plans 2D qui vont être accumulées dans la mémoire cache. Par rapport aux résultats précédents, il faut réussir à stocker un minimum de 15 plans dans la mémoire cache pour le cas du stencil de Jacobi, ces 15 plans correspondant à la mémoire occupé par 2 à 3 composantes dans le cas du problème élastodynamique. De plus, la nécessité de maximiser les accès dans la direction conduisant à des accès mémoire contigus (direction du *unit-stride*), ne permet pas de réduire le volume de données par plan que l'on doit amener dans la mémoire cache. En effet, le découpage de cette direction introduit de nombreux sauts dans l'accès à la mémoire ce qui limite l'effet du *prefetching* (hardware ou software) et dégrade de manière importante les performances [46].
Le tableau 3.1 illustre ce point en implémentant une boucle de Jacobi en 3D, la dimension des blocs dans la direction du *unit-stride* (direction verticale Z) étant progressivement augmentée jusqu'à atteindre la taille limite du problème ($500 \times 500 \times 500$ points de grille). Les tests sont réalisés sur la plateforme `Malm` à base de processeurs quad-cœur *Opteron*. Pour référence, le temps de l'implémentation standard ou naïve (triple boucle simple) est 23.05 s. On constate bien l'effet de la dimension verticale qui doit être maximisée afin de garantir un niveau optimal de performance.

bloc-X	bloc-Y	bloc-Z	Temps (s)
20	20	10	61.2
20	20	20	48.9
20	20	30	39.4
20	20	40	35.1
20	20	50	30.6
20	20	100	22.6
20	20	200	19.6
20	20	300	17.5
20	20	400	17.5
20	20	500	14.8

TABLE 3.1 – Impact du *prefetching* dans le cas de l'algorithme de Rivera et Tseng sur l'architecture `Malm`.

L'application de cette méthode est donc difficile à envisager pour l'élastodynamique et les gains obtenus sont inférieurs à 2%. Des approches plus évoluées doivent donc être considérées en exploitant par exemple la dimension temporelle afin d'augmenter la granularité des calculs.

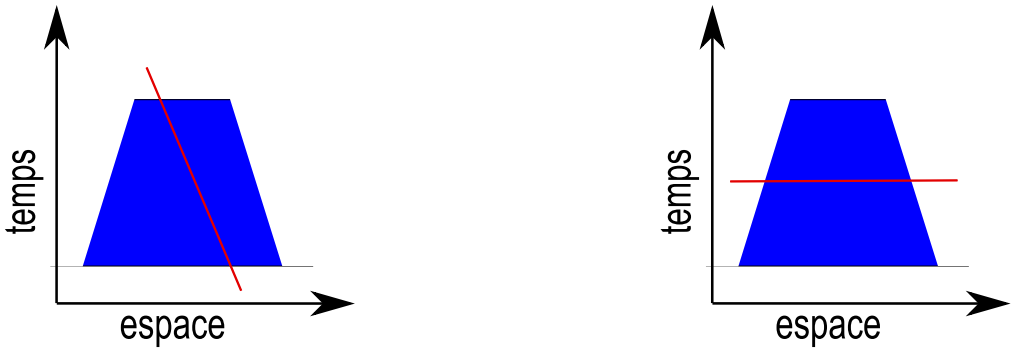

FIGURE 3.4 – Coupe récursive dans les directions spatiales (gauche) ou temporelles (droite) pour la méthode *cache-oblivious*.

3.1.2 Techniques de décomposition espace-temps

Une autre approche développée dans la littérature scientifique est l'utilisation conjointe des dimensions d'espace et de temps pour la décomposition du domaine initial de calcul. Typiquement, il s'agit d'évaluer plusieurs pas de temps successivement en considérant un sous-domaine de calcul constant. L'idée est donc de traverser le domaine espace-temps ainsi constitué en respectant les dépendances dans les directions spatiales et temporelles. Cette approche permet d'améliorer le ratio calcul/accès mémoire. En effet, en gardant des sous-domaines de petite taille et en augmentant le nombre d'opérations effectuées sur cet ensemble de données, la réutilisation des données présentes au plus près de l'unité de calcul est bien meilleure. Cependant, ces approches ne peuvent pas être généralisées. Par exemple, lorsque d'autres types de calcul sont nécessaires entre les boucles d'espace ou de temps, l'espace des itérations ne peut pas être réarrangé aisément. Cet algorithme est néanmoins particulièrement bien adapté au cas de la modélisation sismique.

Appliquées aux stencils tridimensionnels, les techniques dites de *cache-oblivious* [59] reposent sur un partitionnement alternativement en espace et en temps, du domaine de calcul en fonction de la taille et de la forme de ce dernier. Un algorithme récursif est ainsi utilisé afin d'obtenir des sous-domaines de plus en plus petits jusqu'à l'application du stencil à un sous-domaine composé de quelques points de la grille initiale de calcul. La figure 3.4 décrit la procédure de partitionnement récursif utilisée. L'intérêt de cet algorithme est de masquer le critère de taille pour les coupes successives en espace et en temps. Il repose sur la théorie plus générale proposée par Prokop, Frigo et Leiserson [58, 131]. Cette dernière permet de fournir des algorithmes avec une efficacité théorique optimale et faisant abstraction de l'architecture sous-jacente.

Néanmoins, de récentes évaluations de cet algorithme sur des nœuds de calculs modernes ont montré le faible niveau de performance obtenu avec cette approche [80]. En fait, l'algorithme conserve son efficacité en termes de réduction du nombre de défauts de cache, mais la procédure récursive ne permet pas de gains significatifs en temps de calcul. En effet, les coupes succéssives

effectuées ne préservent pas la direction du *unit-stride*. Nous avons vu dans la section précédente que ce point est critique pour les compilateurs (exploitation des possibilités de *prefetching*) afin d'optimiser manuellement ce type de noyau de calcul. Un certain nombre de réorganisations doivent donc être introduites au moment de l'implémentation (notamment la limitation de la récursivité et donc du caractère *oblivious* de l'algorithme) afin d'optimiser les performances. Les approches dites explicites, avec un critère défini par le programmeur afin de fixer la taille des sous-domaines sont donc privilégiées.

Les techniques dites de *time skewing* et plus généralement d'optimisation du traitement des nids de boucles sont largement discutées dans la littérature [154, 114, 77]. L'idée initiale est également de procéder à une décomposition dans le domaine espace-temps. La principale différence avec les algorithmes de *cache-oblivious* est la définition statique et explicite de la taille des sous-domaines. De plus, l'algorithme de partitionnement est appliqué uniquement dans les directions spatiales. Certaines variantes ont été introduites, principalement pour la prise en compte des stencils avec des dépendances de données comme pour les itérations de type Gauss-Seidel [77].
La figure 3.5 illustre la procédure pour un problème 1D. Le partitionnement en espace est défini

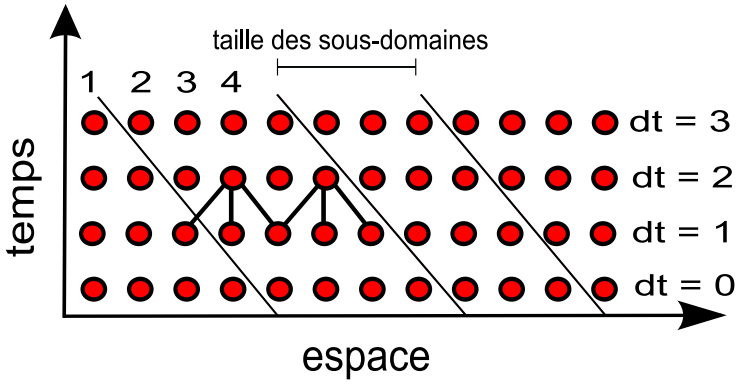

FIGURE 3.5 – Algorithme séquentiel de *time skewing*.

de façon à conserver dans un même sous-domaine tous les points nécessaires à l'évaluation simultanée de plusieurs pas de temps. Pour $dt = 1$, nous pouvons calculer les points numéro 1, 2 et 3 en utilisant les points numéro 1, 2, 3 et 4 du pas de temps $dt = 0$ pour le même sous-domaine. Le respect de la pente conduit à une réduction du nombre de point que l'on peut calculer lorsque le pas de temps évolue. La pente correspondant à l'ordre du stencil doit ainsi être respectée. A l'instar des méthodes *cache-oblivious*, on espère ainsi maximiser les opérations sur des données présentes dans la mémoire cache.

L'une des caractéristiques principales de la méthode de *time skewing* est le critère explicite pour la définition de la taille des sous-domaines. La recherche de la taille et de la forme optimale des sous-domaines est un point crucial mais extrêmement complexe à aborder du à la taille de l'espace des paramètres à explorer.

Un certain nombre de spécificités algorithmiques doivent être prises en compte au moment de l'implémentation. Tout d'abord, le domaine spatial est traversé en suivant un ordre prédéterminé, le calcul démarre par le sous-domaine qui correspond à la frontière gauche et progresse jusqu'à la frontière droite du domaine en respectant la dépendance de données (cf. figure 3.6). Une autre remarque concerne le nombre de points affectés à chaque sous-domaine. Les sous-domaines internes

effectuent le même nombre d'opérations alors que les sous-domaines aux bords ont une charge de calcul différente afin de tenir compte des conditions aux limites. En fonction de l'application, ces conditions limites peuvent changer de façon importante la charge aux bords du domaine en augmentant celle-ci ou en changeant la forme du stencil utilisé.

Le nombre maximum de pas de temps pouvant être calculé simultanément est un critère important. Avec $Nloc$ définissant la taille d'un domaine 1D, on peut calculer $Nloc - 1$ pas de temps avec un stencil d'ordre 2. Une procédure de restart est alors nécessaire pour calculer les pas de temps suivants, cette dernière consistant à bloquer la direction temporelle. L'augmentation de l'ordre du stencil constitue une limitation importante au nombre de pas de temps successifs que l'on peut calculer sur un même sous-domaine. Si on passe d'un stencil d'ordre 2 à un stencil d'ordre 4, le nombre maximum de pas de temps est seulement de $\frac{Nloc-1}{2}$.

Dans le cas d'un calcul tridimensionnel comme décrit dans la section précédente avec des sous-domaines de taille $N = Nxloc \times Nyloc \times Nzloc$, le noyau de calcul est caractérisé par $2N$ accès aux données et $6 \times N \times (N-1)$ opérations flottantes. Le ratio obtenu ($3N$) est donc beaucoup plus favorable dans ce cas (par rapport à une situation en $O(1)$ dans le cas standard) avec une meilleure réutilisation des données dans la mémoire cache.

3.2 Mise en œuvre sur architecture multicœurs et hiérarchiques

3.2.1 Algorithmique parallèle pour la technique de *time skewing*

Cas 1D et 2D

L'implémentation sur architectures multicœurs et hiérarchiques des algorithmes précédents est décrite dans cette section. Concernant la méthode de *time skewing* deux directions principales

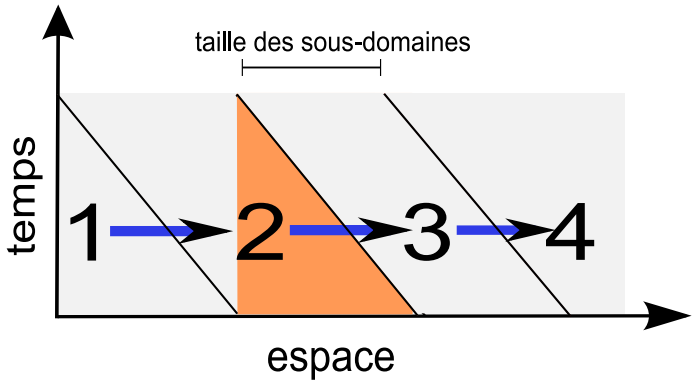

FIGURE 3.6 – Dépendance entre sous-domaines dans le cas de l'algorithme de *time skewing*

peuvent être explorées.

– La première piste serait de suivre l'implémentation séquentielle en distribuant les différents sous-domaines sur les cœurs disponibles. L'algorithme reposant sur un ordre précis pour le calcul des points de grille et donc des sous-domaines, une des difficultés résiderait alors en l'introduction des mécanismes de synchronisation adaptés. Par exemple, la figure 3.6 montre que le calcul du second sous-domaine doit être démarré avant l'évaluation du troisième. La zone colorée illustre la dépendance de données, le calcul complet de la zone

2 n'est pas nécessaire. Cette approche conduit à un algorithme à front d'ondes dont le parallélisme est difficile à extraire. En termes d'implémentation, le programmeur doit notamment masquer les nombreuses phases de synchronisation nécessaires. Néanmoins, des approches spécifiques ont été introduites afin d'optimiser le recouvrement [40] ; une analyse détaillée de l'algorithmique parallèle de ces implémentations est présentée dans [155].

– Une autre proposition est l'utilisation d'un algorithme *red and black* afin de casser la dépendance entre sous-domaines. Comme illustré en 2D (espace et temps) par la figure 3.7, il s'agit de diviser le domaine de calcul en 2 classes de géométrie. Les triangles gris ou rouges sur la figure peuvent ainsi être calculés sans mécanisme de synchronisation pour les sous-domaines appartenant à la même classe. Une barrière est néanmoins nécessaire à la fin de chaque phase de calcul, le déséquilibre de charge à l'intérieur d'une même couleur subsistant donc. La figure 3.8 illustre la forme de ces sous-domaines dans le cas d'un problème 2D ; le respect de la dépendance de données conduit à une géométrie pyramidale en incluant le temps. L'approche que nous mettons en œuvre dans la suite de cette section repose sur ce type de décomposition.

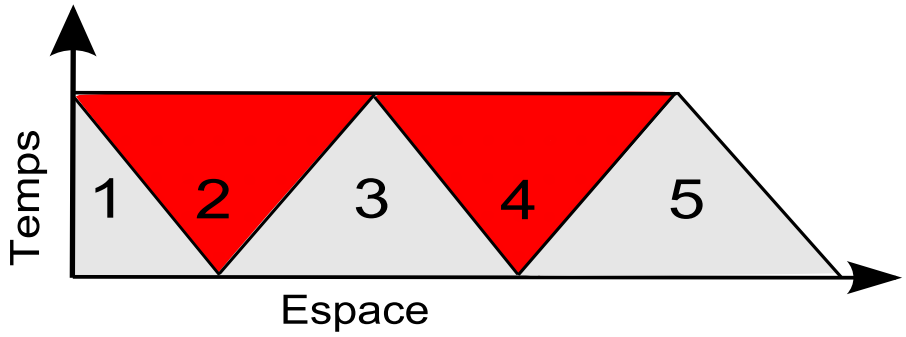

FIGURE 3.7 – Algorithme *red and black* appliqué à l'algorithme de *time skewing* dans le cas d'une géométrie 1D.

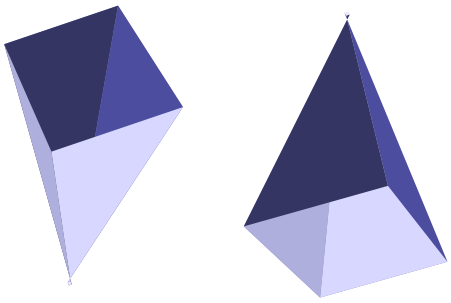

FIGURE 3.8 – Géométrie des sous-domaines 2D (deux directions spatiales plus le temps) obtenue après une décompositions de type *red and black*.

Cas 3D

L'application de la technique de *time skewing* à un domaine 3D soulève différentes questions. L'une des plus importantes s'avère être le choix des directions à partitionner. Un autre défi est

de définir les possibilités d'ordonnancement des différents calculs en respectant la pente liée à la direction temporelle et à l'ordre du schéma numérique. Les remarques des sections précédentes concernant l'impact de la direction du *unit stride* conduisent à privilégier un traitement local de la direction verticale. Le partitionnement spatial repose donc uniquement sur les directions horizontales. L'algorithme s'organise comme suit :

- Le partitionnement 2D horizontal va générer quatre classes différentes de sous-domaines (*red and black*, *red-red*, *black-red*, *black-black*) nécessitant un parcours en quatre étapes. Une représentation globale de cet algorithme est donnée par la figure 3.9 avec l'évolution de la forme des sous-domaines pour le calcul successif de trois pas de temps. Les classes de sous-domaines sont représentées par des couleurs différentes. Par exemple au pas de temps $dt = 1$, le calcul s'effectue en quatre étapes (sous-domaines bleus, puis oranges, puis rouges, puis gris par exemple). Les sous-domaines ont tous la même taille au cours du premier pas de temps. Lors de l'évolution du pas de temps, le respect de la dépendance des données va conduire à un déséquilibre dans le nombre de points à évaluer pour chaque classe. Typiquement les sous-domaines orange sont de plus en plus couteux à calculer par rapport au sous-domaines gris. De même, les conditions aux limites sont à l'origine d'un déséquilibre entre sous-domaines d'une même classe de couleur conduisant à des géométries de plus en plus irrégulières.

- Le traitement de la direction verticale est séquentiel en suivant l'algorithme original de la figure 3.5.

Si on considère maintenant le ratio entre les opérations flottantes et les accès aux données, les résultats sont moins bons que ceux de l'algorithme original ($3 \times Nloc$). En effet, dans le cas d'une décomposition *red and black*, la forme des sous-domaines réduit le nombre de pas de temps que l'on peut évaluer pour un même sous-domaine. En supposant un problème 1D de taille $Nloc+1$ dans notre cas, nous pouvons évaluer $\frac{Nloc}{2}$ pas de temps à la fois dans le cas d'un stencil d'ordre 2 et $\frac{Nloc}{4}$ dans le cas d'une approximation d'ordre 4. Le ratio est alors de l'ordre de $\frac{3 \times Nloc}{4}$ pour un stencil d'ordre 2 et $\frac{3 \times Nloc}{8}$ pour un stencil d'ordre 4.

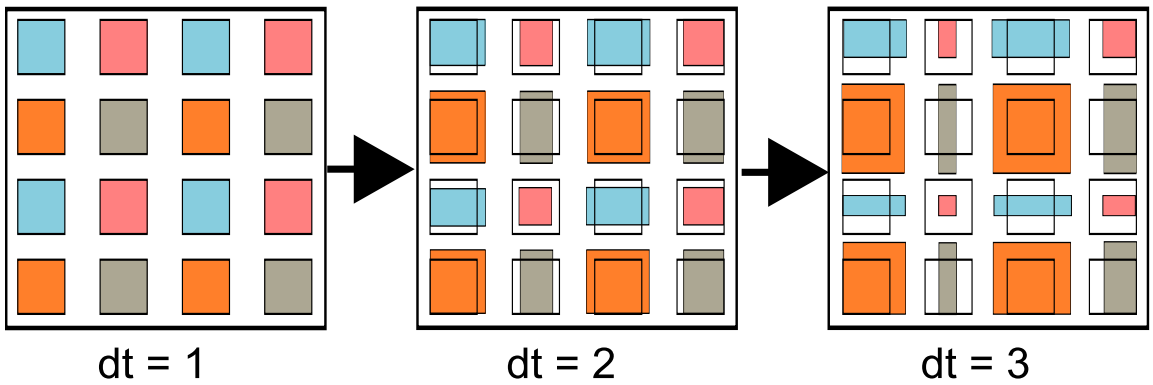

FIGURE 3.9 – Représentation schématique de l'algorithme parallèle de *time skewing* avec quatre classes de sous-domaines.

3.2.2 Expression du parallélisme et éléments d'implémentation

Aucun des standards classiques du parallélisme (MPI ou OpenMP) ne permettent d'exprimer aisément l'algorithme décrit par la figure 3.9. Ces 2 modèles de programmation n'offrent pas la flexibilité nécessaire à une telle implémentation. Les principales caractéristiques de cet algorithme sont :

- Un déséquilibre de charge important entre les sous-domaines appartenant à des classes différentes. Les sous-domaines d'une même classe présentent un nombre de points variable à calculer, ceci étant dû à la prise en compte des conditions limites. Un mécanisme de vol de travail est donc à introduire à ce niveau.
- L'algorithme de *time skewing* utilisé conduit à un parallélisme à grain fin induit par la décomposition horizontale. Afin de pouvoir disposer de sous-domaines dont la taille correspond aux capacités de la mémoire cache disponible, le partitionnement conduit à des sous-domaines de très petite taille. Une solution envisageable est l'introduction d'une décomposition à deux niveaux avec des macro-domaines pour le parallélisme à gros grain et un parallélisme à grain fin à l'intérieur. La gestion des interfaces entre ces macro-domaines demeure une difficulté avec une décomposition respectant la dépendance des données.

En termes d'implémentation, des approches telles que les TBB (Intel Threading Building Blocks) ou la librairie Cilk [25] pourraient être exploitées. Elles proposent des mécanismes transparents de vol de travail et ont été utilisées dans le cas des techniques *cache-oblivious* sur architecture parallèle [60, 50].

En fait, la situation décrite est assez proche de celle exposée dans les chapitres précédents, avec une situation de multithreading massif et la nécessaire prise en compte de la hiérarchie mémoire. Une autre proposition serait donc d'exploiter les mêmes mécanismes. Par exemple la librairie de threads Marcel peut être avantageusement utilisée afin d'aborder le déséquilibre de charge algorithmique. En effet, l'efficacité du mécanisme de virtualisation des cœurs pour ce type de problème a été démontré. Le parallélisme à grain fin sera alors d'autant mieux exploité.

En termes de schéma d'accès aux données, la situation est assez similaire à la description effectuée dans la section 2.1 pour la modélisation élastodynamique. Les accès à la mémoire sont massivement irréguliers dans notre cas, la différence principale réside dans la forme des blocs mémoires accédés. Dans le cas de la modélisation sismique, les briques de base étaient constituées de parallélépipèdes (en forme de frites), dans notre cas la forme de base est beaucoup plus complexe à décrire dans le cas 3D. Les politiques de gestion mémoire offertes par la librairie MAi peuvent donc être utilisées afin de contrôler le placement des données.

3.3 Adaptation au cas de la modélisation sismique

Une des particularités de la méthode numérique utilisée pour la résolution de l'équation de l'élastodynamique est l'emploi d'une grille en quinconces pour la discrétisation. Le calcul des dérivées est décalé conduisant à un découplage de l'évaluation des composantes de contraintes et de vitesse. Cette propriété importante permet une application quasi-immédiate de notre algorithme de *time skewing*. En effet, la dépendance alternée entre le calcul des champs de vitesse et de contrainte peut être exploitée pour une décomposition dans le domaine espace-temps. La figure 3.10 illustre l'alternance du calcul des champs avec un demi-pas de temps. Le lien avec les approches proposées par Graves [66] est évident. Ce dernier proposait en 1996 une technique as-

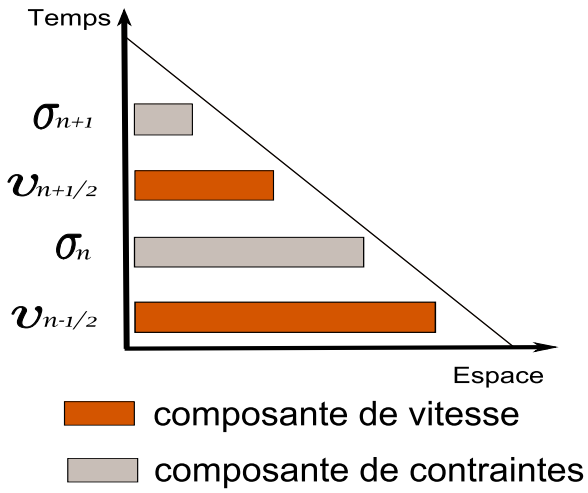

FIGURE 3.10 – Dépendance alternée entre les composantes de vitesse (v_n) et de contraintes (σ_n) pour l'algorithme de *time skewing* implémenté pour un schéma de différences finies et une grille de discrétisation en quinconces.

sez similaire au niveau algorithmique pour l'implémentation de l'équation de l'élastodynamique dans un contexte *out-of-core*. Dans son cas, l'objectif était de limiter les accès au disque. La définition de la taille optimale des sous-domaines et l'ordonnancement des transferts entre la mémoire centrale et le disque reposaient sur le programmeur. L'idée proposée semblait originale car aucune référence n'est faite aux nombreux travaux en cours à la même époque sur les techniques de *time skewing* dans le domaine informatique et particulièrement de la compilation.

Par rapport à l'émergence des architectures multicœurs et hiérarchiques, la problématique actuelle est plutôt liée à l'optimisation des performances. Une modification de la méthode de *time skewing* initiale est nécéssaire afin de coller au stencil utilisé avec l'introduction du demi-pas de temps. Les 3 composantes de vitesse et les 6 composantes de contrainte sont successivement calculées de manière groupée en respectant le découpage du domaine espace-temps. Deux paramètres peuvent significativement influencer les performances de cette implémentation :

- Dans le cas élastodynamique, il est nécessaire de stocker successivement 3 et 6 composantes de vitesse et de contrainte à chaque étape. Cela induit beaucoup plus d'accès mémoire que dans le cas simple avec 2 tableaux pour les boucles de Jacobi. L'occupation de la mémoire cache est plus importante avec une évaluation successive du même nombre de pas de temps.
- Le stencil d'ordre 4 utilisé et le nombre supérieur d'accès mémoire conduit à un ratio calcul/mouvement de données proche de $0.175 \times N$ dans le cas d'un problème de taille N.

3.4 Evaluation des performances

Contexte et dimensionnement des expériences

Dans cette section, nous détaillons les résultats obtenus sur les plateformes `Malm`, `Borderline` et `Idkoiff`. L'architecture symétrique `Jade` est utilisée afin de montrer l'intérêt de cette approche pour limiter la contention sur le bus mémoire cadencé à 1600 MHz. Les simulations effectuées sur les architectures à base de processeurs Opteron utilisent le compilateur GNU gcc en version 4.1 ou supérieure. Sur la plateforme `Jade` , le compilateur Intel C est utilisé. Ce point est critique

car le choix des options de compilation peut se révéler contraire dans certains cas à l'optimisation algorithmique proposée [145].

Un grand nombre d'options ont été évaluées, l'impact observé en utilisant des options agressives est relativement modeste par rapport à l'utilisation des flags classiques. Les exemples sont donc compilés avec les options standards *-O3 -ffast-math* pour le compilateur *gcc* et *-fast* pour le compilateur *icc*. Le caractère reproductible des résultats est ainsi renforcé.

Concernant l'impact de la hiérarchie mémoire, la première solution envisagée est de se reposer sur l'approche *first touch* par défaut. La libraire MAI a été utilisée afin d'implémenter les stratégies de placement des données *bind* et *cyclic*. Les caractéristiques de l'architecture sous-jacente pourront ainsi conduire à optimiser la bande passante ou la latence.

Deux types de stencil sont utilisés au cours des expériences. Une boucle de Jacobi est tout d'abord implémentée afin de mettre en évidence les performances sur un noyau très largement utilisé dans la littérature scientifique. Le second exemple est basé sur la méthode des différences finies appliquées au problème de l'élastodynamique. Nous utilisons le schéma standard d'ordre 4 en espace et basé sur une grille de discrétisation en quinconces. L'implémentation utilise successivement le calcul des composantes de vitesse et de contrainte soit 9 vecteurs au total à stocker.

Les exemples implémentés sont tridimensionnels. En termes de gestion mémoire, l'allocation repose sur l'utilisation de la fonction *malloc* définissant un tableau 1D. Afin de prendre en compte le caractère 3D de nos calculs, ce tableau est ensuite accédé en effectuant des sauts dans la mémoire pour passer d'une dimension d'espace à l'autre. Contrairement à la stratégie utilisée pour l'implémentation du code *Ondes3D*, nous n'utilisons pas ici les fonctions avancées proposées par *Numerical Recipies* afin d'allouer des structures 3D reposant sur des tableaux cascadés de pointeurs. Les performances observées pourront donc être assez différentes.

	Malm	Borderline	Idkoiff
Noyau de calcul Jacobi	$500 \times 500 \times 500$	$625 \times 625 \times 625$	$625 \times 625 \times 625$
Noyau de calcul sismique	$300 \times 300 \times 300$	$375 \times 375 \times 375$	$375 \times 375 \times 375$

TABLE 3.2 – Taille des problèmes en nombre de points de grille afin de saturer la mémoire disponble sur un nœud NUMA de chaque architecture.

La taille choisie pour les différents cas tests est détaillée dans le tableau 3.2. La stratégie utilisée est de définir des problèmes permettant de saturer la moitié de la mémoire disponible sur chaque nœud NUMA. Cela permet d'évaluer la pénalité maximale. Ces dimensions initiales sont ensuite étendues afin de saturer l'ensemble de la machine lorsque cela est nécessaire (par exemple la multiplication par un facteur deux pour une machine comportant deux nœuds NUMA). L'évaluation simultanée de 4 pas de temps est implémentée pour l'algorithme de *time skewing*. Dans le cas de l'intégration de cette procédure dans une application réelle, il sera nécessaire de trouver un compromis entre le nombre de pas de temps à calculer simultanément (localité temporelle) et la taille des sous-blocs (localité spatiale).

Expériences séquentielles

La version séquentielle de l'algorithme de *time skewing* est plus spécifiquement évaluée dans cette section. L'efficacité de notre stratégie n'est pas forcément maximale par rapport aux approches décrites précédemment (techniques *cache-oblivious* ou techniques initiales de *time ske-*

wing) car l'algorithme proposé cible prioritairement les architectures multicœurs ; il est cependant important d'évaluer son efficacité séquentielle.
Les tableaux 3.3 et 3.4 résument les résultats obtenus sur l'architecture `Idkoiff`. La librairie

	Défauts de cache	Nombre total de cycles
Noyau Jacobi - *standard*	$10.8\text{x}10^6$	$15.9\text{x}10^9$
Noyau Jacobi - *time skewing*	$7.3\text{x}10^6$	$10.9\text{x}10^9$

TABLE 3.3 – Statistiques des compteurs hardware dans le cas du noyau de calcul Jacobi.

	Défauts de cache	Nombre total de cycles
Noyau sismique - *standard*	$341\text{x}10^6$	$30.3\text{x}10^9$
Noyau sismique - *time skewing*	$57.5\text{x}10^6$	$23.3\text{x}10^9$

TABLE 3.4 – Statistiques des compteurs hardware dans le cas du noyau de calcul sismique.

PAPI (Performance Application Programming Interface) est utilisée afin de collecter les informations statistiques des compteurs hardware (défauts de cache et nombre total de cycles).

L'analyse du nombre total de cycles montre une réduction de 31.4% pour le noyau de Jacobi et 23.1% pour le noyau de calcul sismique. Dans le cas de la boucle de Jacobi, la réduction du nombre de cycles et la réduction du taux de défauts de cache (L2 DCM) semblent fortement corrélées. La situation est assez différente dans le cas du noyau de calcul sismique avec une très forte réduction du taux de défauts de cache (un facteur de 5.9) mais une diminution beaucoup plus faible du nombre de cycles. Globalement, le gain obtenu pour une boucle de Jacobi est plus faible. La complexité plus grande du schéma numérique pour le problème élastodynamique et le plus grands nombres de vecteurs impliqués dans le noyau de calcul expliquent certainement ce résultat. On peut également avancer l'hypothèse que le premier problème est plutôt *memory bound* alors que le second problème est *cpu bound* avec une réduction plus importante du taux de défauts de cache.

Une exploration large des paramètres possibles a été effectuée afin de déterminer la meilleure taille de blocs dans le cas de l'algorithme de *time skewing*. Les performances sont optimales quand les dimensions des blocs dans les directions horizontales sont minimales. Dans le cas d'un stencil d'ordre 2 avec le calcul simultané de 4 pas de temps, les valeurs optimales observées sont comprises entre 9 et 15 points de grille dans chaque direction. La direction verticale est beaucoup plus sensible au *prefetching*, la taille optimale varie entre 10 et 110 points pour un problème de taille $500 \times 500 \times 500$. Cependant, l'impact du prefetching est plus faible par rapport aux expériences détaillées précédemment sur l'architecture `Malm` avec l'algorithme de Rivera et Tseng.
Le tableau 3.5 résume les gains obtenus sur différentes architectures. Seules les caractéristiques du processeur ou du compilateur influent pour une exécution séquentielle. Les performances obtenues sont assez similaires avec des gains qui varient entre 1.6 et 1.9 pour la boucle de Jacobi. Dans le cas du noyau de calcul sismique, les accélérations observées sont plus faibles (entre 1.2 et 1.5). Dans le cas du noyau sismique, le faible nombre d'opérations par rapport aux accès mémoire explique ce résultat. Le faible impact des différences entre processeurs, principalement par rapport à la taille et l'organisation des mémoires cache, constitue un point à approfondir.

	Malm	Borderline	Idkoiff	Jade
Noyau Jacobi	1.6	1.9	1.6	1.6
Noyau sismique	1.5	1.2	1.4	1.4

TABLE 3.5 – Accélérations obtenues sur différentes architectures pour l'algorithme séquentielle de *time skewing*.

Globalement, les résultats sont légèrement inférieurs à ceux décrits dans [46] avec un stencil similaire. La différence entre les algorithmes de *time skewing* explique en partie ces résultats. De plus, le parcours de l'espace des dimensions possibles pour les sous-blocs est probablement moins efficace dans notre cas. L'approche initiale proposée dans [46] a par la suite été enrichie afin de proposer une véritable stratégie d'auto-tuning [47].

Impact du trafic mémoire et des caractéristiques de l'architecture

Dans cette section, nous analysons l'impact du trafic mémoire sur les performances des différents noyaux de calcul. Le premier test est effectué en utilisant les versions séquentielles des deux noyaux de calcul. Nous comparons le gain obtenu entre l'implémentation standard et l'algorithme de *time skewing* dans le cas d'un placement local ou distant de la mémoire. Les simulations sont réalisées sur le serveur `Idkoiff` avec des tailles de problèmes correspondant à la mémoire disponible sur un nœud. Le tableau 3.6 synthétise les résultats en présentant le rapport entre un placement local ou un placement distant de la mémoire pour chacun des noyaux et chacune des implémentations.

Dans le cas de la boucle de Jacobi, la pénalité NUMA passe de 25.1% pour une implémentation standard à 3.6% pour l'algorithme de *time skewing*. A l'instar des expériences précédentes basées sur l'algorithme de Rivera et Tseng, la réduction du nombre de défauts de cache permet de réduire significativement l'impact de la hiérarchie mémoire. Dans le cas du noyau de calcul sismique, l'écart est beaucoup plus faible ceci étant dû à l'efficacité moindre de l'algorithme de time skewing.
La mise en œuvre parallèle de ces deux noyaux de calcul souligne les problèmes de contention

	Noyau Jacobi	Noyau sismique
Implémentation *standard* (%)	25.1	20.2
Implémentation *time skewing* (%)	3.6	11.5

TABLE 3.6 – Rapport du nombre de cycles en exploitant un placement local ou distant des données.

mémoire. Ces derniers sont particulièrement importants en prenant en compte le caractère *memory bound* de ces calculs. Dans un premier temps, nous implémentons une version parallèle standard des noyaux de calcul Jacobi et sismique. Les directives OpenMP sont utilisées afin de distribuer les boucles de calcul. Nous utilisons cette fois un jeu de données permettant de saturer la mémoire disponible sur les différentes machines.

Les résultats de ces tests sont représentés sur la figure 3.11 qui décrit l'accélération en fonction du nombre de cœurs utilisés. Dans le cas du noyau Jacobi, les performances sont assez

mauvaises sur toutes les architectures avec une efficacité parallèle maximale sur `Malm` (50%) et des résultats inférieurs à 25% sur les autres architectures. La situation est similaire dans le cas du noyau sismique, les accélérations mesurées étant légèrement meilleures grace à un grain de calcul plus grossier. Ces résultats sont largement inférieurs aux expériences menées avec l'application complète ONDES3D ; le grain de calcul plus grossier et la complexité supérieure des boucles de calcul dans ce cas réel expliquent cette différence.

Les performances mesurées s'expliquent principalement par l'impact de la hiérarchie mémoire dans le cas des architectures à base de processeurs Opteron. Les performances sont meilleures sur `Malm` car cette dernière architecture est beaucoup moins hiérarchique que les serveurs `Borderline` ou `Idkoiff`. En revanche, sur l'architecture `Jade`, les performances très faibles s'expliquent principalement par les limitations du bus mémoire. Ce goulot d'étranglement est bien connu sur les processeurs Intel Hapertown et rendent difficile l'exploitation efficace des huit cœurs disponibles.

Afin d'améliorer les performances applicatives, la librairie MAI est utilisée afin d'optimiser le

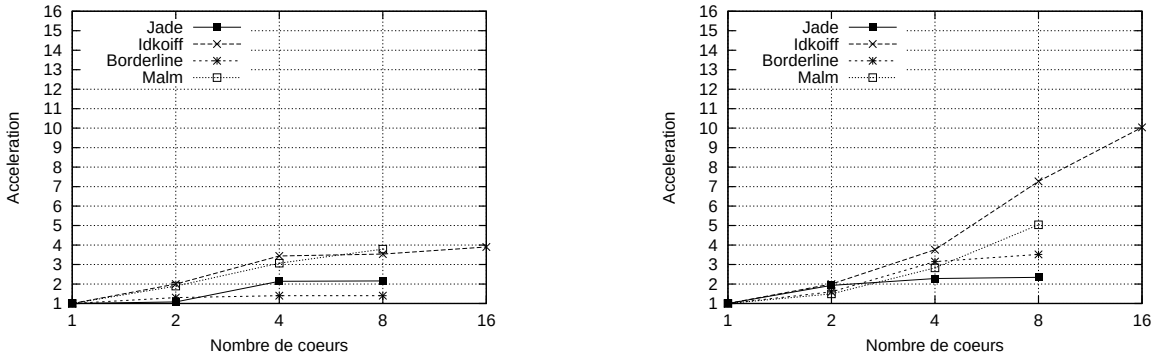

FIGURE 3.11 – Accélération pour une implémentation standard OpenMP du noyau de Jacobi (gauche) et du noyau sismique (droite) sur différentes architectures.

choix de la politique mémoire. Nous comparons l'implémentation du noyau de calcul sismique sur les plateformes `Malm` et `Idkoiff` en utilisant trois stratégies différentes (*first touch*, *bind* et *cyclic*). Les résultats sont décrits par la figure 3.12. Sur la plateforme `Idkoiff`, la meilleure approche consiste à fixer les threads sur les différents nœuds de calcul. L'accélération sur 16 processeurs passe de 10.02 à 12.91.

Les conclusions sont assez différentes sur `Malm` ; dans ce cas, la politique cyclique donne les

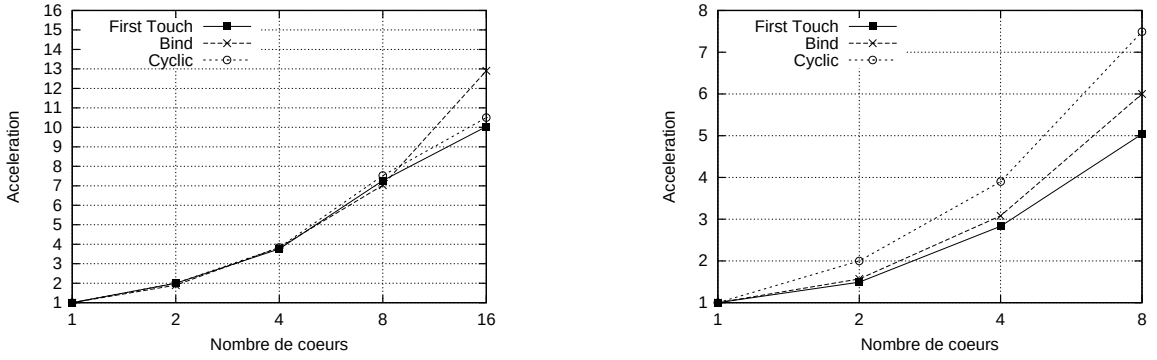

FIGURE 3.12 – Impact de la politique mémoire sur le noyau de calcul sismique pour les architectures `Idkoiff` (gauche) et `Malm` (droite).

meilleurs résultats. L'accélération mesurée passe de 5.04 à 7.59 sur 8 cœurs. Ces expériences montrent l'impact important de la hiérarchie mémoire et du choix de la politique la mieux adaptée. Ils illustrent également la difficulté de ce choix avec des résultats différents d'une architecture à l'autre. La structure des allocations mémoire conduisent également à des résultats différents de ceux observés dans le cas de l'application ONDES3D.

Aucune stratégie efficace ne peut être appliquée dans le cas la plateforme `Jade`. De plus les résultats obtenus sur `Idkoiff` et `Malm` sont loin d'être optimaux. Il est donc nécessaire d'introduire d'autres pistes afin d'optimiser le niveau de performance.

Performances sur architectures multicœurs

Les performances obtenues sur différentes architectures sont résumées par la figure 3.13. Les données sont allouées sur un nœud NUMA ou sur l'ensemble de la machine. L'implémentation parallèle basée sur OpenMP est définie comme référence ; nous présentons l'accélération obtenue en utilisant la même implémentation avec une stratégie mémoire optimisée ou l'algorithme de *time skewing* (avec ou sans un placement optimisé de la mémoire).

Tout d'abord, les expériences menées ont permis d'illustrer certaines limitations de l'algorithme décrit par la figure 3.9. En effet, lorsque la taille du domaine de calcul augmente, le nombre de threads à ordonnancer devient trop grand. En effet ce dernier est directement défini à partir de la taille optimale permettant de rentrer dans la mémoire cache. Typiquement, pour un problème tridimensionnel de taille $1000 \times 1000 \times 1000$ points de grille et des sous-blocs de dimension 20 points de grille dans notre cas, la librairie MARCEL doit alors ordonnancer efficacement environ 2500 threads. Un goulot d'étranglement apparaît clairement au niveau de l'unique file d'ordonnancement nécessitant l'introduction d'un deuxième niveau de partitionnement. La solution implémentée est de limiter le nombre de sous-blocs issus du découpage *red and black* ce qui permet de borner le nombre de threads à ordonnancer. Afin d'obtenir des sous-blocs de petites tailles et rentrer dans la mémoire cache, la procédure séquentielle *red and black* est de nouveau appliquée à chacun des sous-domaines issu de la procédure parallèle. La seule différence avec la stratégie utilisée pour la direction verticale est le fait que les bords des sous-domaines ne soient pas droits (la forme correspond aux sous-domaines décrits par la figure 3.8). L'algorithme général correspond alors à un premier niveau de découpage parallèle pour générer différents sous-blocs, ces derniers étant de nouveau découper séquentiellement pour rentrer dans la mémoire cache.

Des gains significatifs sont obtenus dans tous les cas. Ces gains sont maximum dans le cas où les données ne saturent pas toute la mémoire disponible, il s'agit de la situation la plus défavorable en termes de probabilités d'accès mémoire distants. Les résultats obtenus avec un placement optimisé des données sont relativement proches de ceux correspondant à l'algorithme de *time skewing* basé sur la politique mémoire par défaut *first touch*. Ces résultats comparables reposent sur des mécanismes très différents. Dans le premier cas, il s'agit des bénéfices tirés de l'optimisation de la bande passante ou de la latence mémoire. Le nombre total d'accès mémoire réalisés hors des différents niveaux de mémoire cache restant constant. En revanche, l'algorithme de *time skewing* réduit le nombre de défauts de cache limitant implicitement les opportunités d'accès distants à la mémoire. On peut également analyser ces résultats en soulignant le remplacement de la phase de recherche de la stratégie mémoire la mieux adaptée (en fonction de l'algorithme ou de l'architecture) par l'utilisation d'un algorithme de *time skewing* exploitant la politique mémoire par défaut *first touch*.

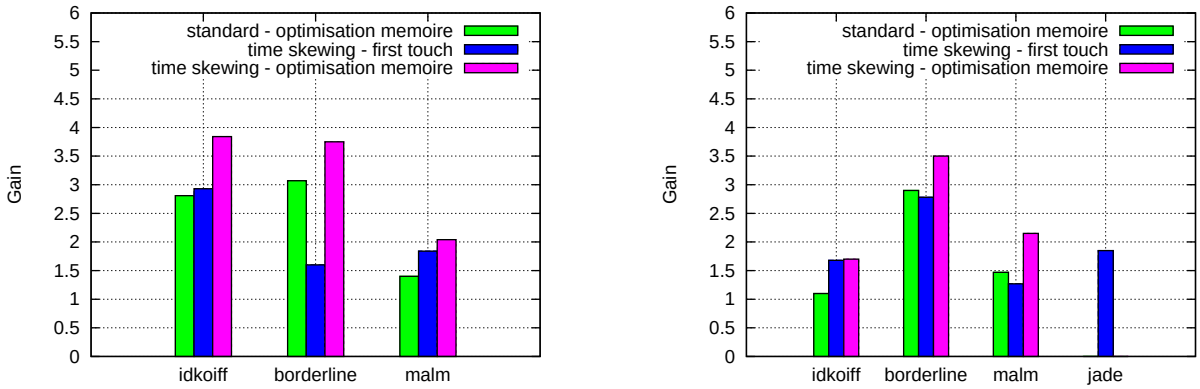

FIGURE 3.13 – Accélération mesurée par rapport à une implémentation standard du noyau de calcul sismique dans les cas où la taille des données correspond à la mémoire disponible sur un nœud NUMA (gauche) ou sur l'ensemble de la machine (droite).

Le niveau de performance optimal est atteint dans tous les cas lorsqu'une strategie mémoire optimisée est combinée à l'algorithme de *time skewing*. Le temps de calcul est divisé par au moins un facteur 2, la situation la plus favorable étant observée sur la plateforme `Idkoiff` (gain d'un facteur 3.84).

Les résultats sur la plateforme `Jade` illustrent les limites du bus mémoire qui ne fournit pas une bande passante suffisante afin d'alimenter les huit cœurs disponibles sur chaque nœud de calcul. Cette limitation de l'architecture est renforcée par le caractère *memory bound* du noyau de calcul différences finies. L'algorithme de *time skewing* permet de réduire significativement la contention mémoire et une accélération d'un facteur 1.84 est mesurée.

Chapitre 4

Mise en œuvre sur cluster de nœuds multicœurs

Contents

La mise en œuvre de simulations tridimensionnelles réalistes nécessite l'utilisation de plateformes de calcul de grande taille principalement composées de clusters de nœuds multicœurs. Les chapitres précédents ont permis d'étudier de façon détaillée le comportement de l'algorithme de différences finies sur architectures multicœurs en tenant compte de l'aspect NUMA de la mémoire partagée. Dans un contexte hybride, l'implémentation des algorithmes dédiés à la hiérarchie mémoire est identique. Les extensions nécessaires des algorithmes de *time skewing* sur architectures à mémoire distribuée sont encore à développer. En revanche, les approches proposées pour résoudre le déséquilibre de charge sont à étendre afin d'introduire un deuxième niveau de parallélisme. Ce dernier point est abordé dans ce chapitre. Une discussion sur l'efficacité des implémentations hybrides en terme de schéma de communication est également proposée. Enfin, un cas d'application est présenté avec la modélisation du séisme de Niigata-Chuetsu au Japon (2007).

4.1 Introduction d'un deuxième niveau de parallélisme

4.1.1 Mécanisme de virtualisation dans un contexte d'exécution hybride

L'introduction d'un deuxième niveau de parallélisme est une stratégie naturelle afin de prendre en compte le caractère hybride des architectures. Dans notre cas, il s'agit de combiner la virtualisation au niveau multicœurs et une approche quasi-statique au niveau des processus MPI.

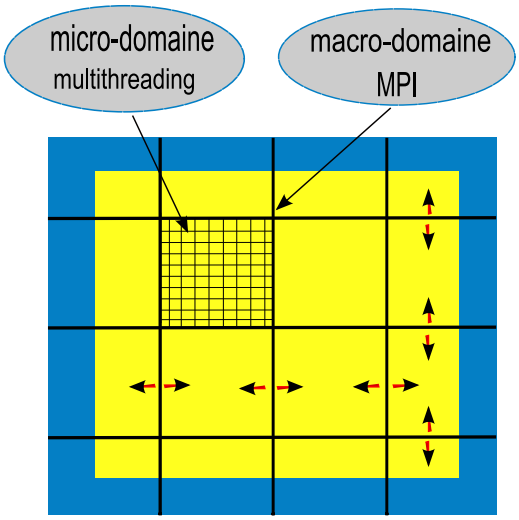

FIGURE 4.1 – Partitionnement hybride du domaine de calcul.

– Le premier niveau de découpage en macro-domaines permet d'utiliser une distribution plus grossière définissant ainsi les sous-domaines qui seront affectés aux processus MPI. La charge entre ces sous-domaines est équilibrée par la procédure quasi-statique décrite dans la section 1.2. De manière résumée, il s'agit d'équilibrer statiquement la taille des sous-domaines en fonction du coût de calcul des différentes zones. L'avantage est de pouvoir limiter le nombre de macro-domaines et de maximiser les performances de la régulation quasi-statique. L'évaluation du critère de répartition est effectuée durant la phase de pré-traitement.

– Le deuxième niveau de découpage correspond au parallélisme interne aux nœuds de calcul multicœurs. Ce niveau conduit à la création d'un grand nombre de micro-domaines traités par les threads MARCEL. Afin d'optimiser ce deuxième niveau de parallélisme, un partitionnement hiérarchique est introduit. Nous découpons tout d'abord chaque micro-domaine en N sous domaines correspondant aux N cœurs disponibles. Le poids de chaque micro-domaines est ensuite évalué. Les N/2 sous-domaines les plus coûteux étant ordonnancés avec une priorité plus grande. Les N/2 sous domaines restants sont ensuite redécoupés afin de proposer à l'ordonnanceur des tâches très petites permettant de lisser la charge sur l'ensemble des cœurs. Cette heuristique conduit aux meilleurs résultats obtenus.

La figure 4.1 décrit ce découpage hybride avec des micro-domaines de taille variable qui seront placés dans la file d'ordonnancement de chaque nœud et des macro-domaines de taille également variable afin de minimiser le déséquilibre de charge entre les nœuds de calcul. Plusieurs paramètres sont fixés de façon expérimentale, comme par exemple le nombre optimal de threads nécessaires afin de maximiser l'équilibrage de charge. Ce nombre est nécessairement borné en raison de la nécessité de ne pas dégrader les performances en créant une contention trop forte au niveau de l'unique file d'ordonnancement.

L'articulation du niveau MPI et du niveau threads est cruciale. Dans notre cas, la stratégie choisie est conservative car elle dissocie totalement les phases de calcul multithreadées des phases de communication. L'application se prête bien à ce découpage en raison d'une structure

très séquentielle entre les phases de calcul des composantes de vitesses et de contraintes et les phases de communication. De plus, cette approche nous permet d'éviter un couplage fort entre la bibliothèque MPI et la librairie de multithreading.

Dans le cas contraire, il faudrait permettre à tous les threads d'effectuer des envois de messages. Les librairies MPI offrant un support complet du multithreading (*THREAD_MULTIPLE*) ne sont pas courantes et peuvent limiter la portabilité [54]. Enfin, la mise en œuvre efficace d'une telle stratégie reste difficile en raison du mécanisme de virtualisation des processeurs utilisés. Des environnements dédiés tels que MPC ont développé des techniques optimisant la réactivité des communications. Nous nous limitons donc à la situation où un seul thread s'occupe de l'envoi et de la réception des messages, ce qui correspond au niveau de support *FUNNELLED* qui est le plus courant.
La figure 4.2 illustre le modèle d'exécution hybride implémenté. Chaque processus MPI va créer

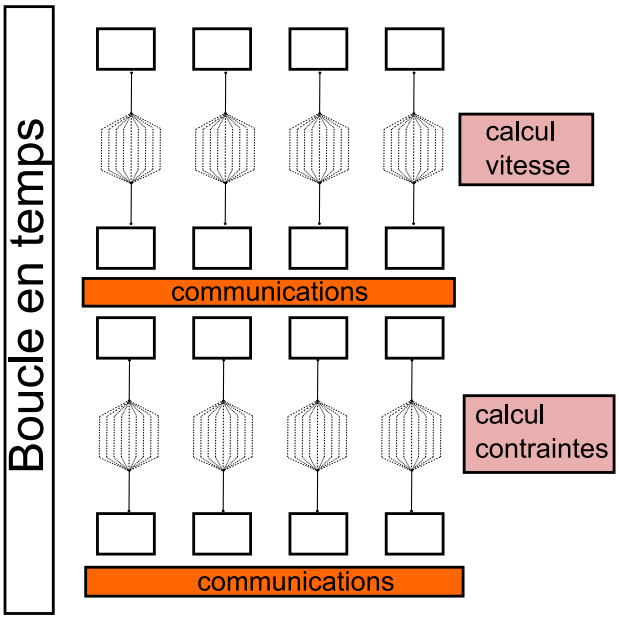

FIGURE 4.2 – Modèle d'exécution hybride associant macro et micro domaines.

un grand nombre de micro-domaines au niveau de chaque nœud de calcul multicœurs. L'objectif est d'obtenir un déséquilibre de charge très faible entre micro-domaines sur un même nœud permettant ainsi de contrôler le déséquilibre global de l'application par le niveau MPI.

4.1.2 Stratégies de communication et impact du modèle de programmation

Il existe un lien important entre le choix du modèle de programmation et la stratégie de communication. En fonction de la granularité du problème, l'écart de performances entre un schéma de communication bloquante ou non bloquante sera variable. La progression réelle des communications est une problématique abordée de différentes manières au sein des supports d'exécution (exploitation des fonctionnalités offertes par la carte réseau, utilisation d'un thread de progression ou intégration forte des mécanismes de communication et de calcul au niveau de l'ordonnanceur). Un résumé des différentes approches est disponible dans [150]. Ces différents mécanismes sont généralement transparents pour le programmeur qui n'a pas de prise réelle sur

le niveau de performances offert. Par exemple, en fonction du contexte applicatif ou des performances du support d'exécution, les approches synchrones ou asynchrones peuvent conduire à des performances assez proches.

Afin d'évaluer l'impact du modèle de programmation dans le contexte applicatif, nous considérons deux exemples synthétiques mis en œuvre sur la plateforme `Phoebus`. Le premier correspond à une faible occupation mémoire (2.6 Go) et il permet de mesurer l'impact de la granularité du problème. L'augmentation régulière du nombre de cœurs disponibles impose une efficacité maximale des applications en considérant un grain de calcul très fin. Le deuxième exemple permet d'évaluer l'extensibilité en maintenant une occupation de 2.6 Go par cœur. Les couches absorbantes CPML sont désactivées pour ces expériences afin de limiter l'impact du déséquilibre de charge, ce dernier étant par ailleurs traité différemment en fonction du modèle de programmation mis en œuvre. Les résultats présentés intègrent à la fois les phases de communication et les phases de recopie de buffers avant et après envoi.

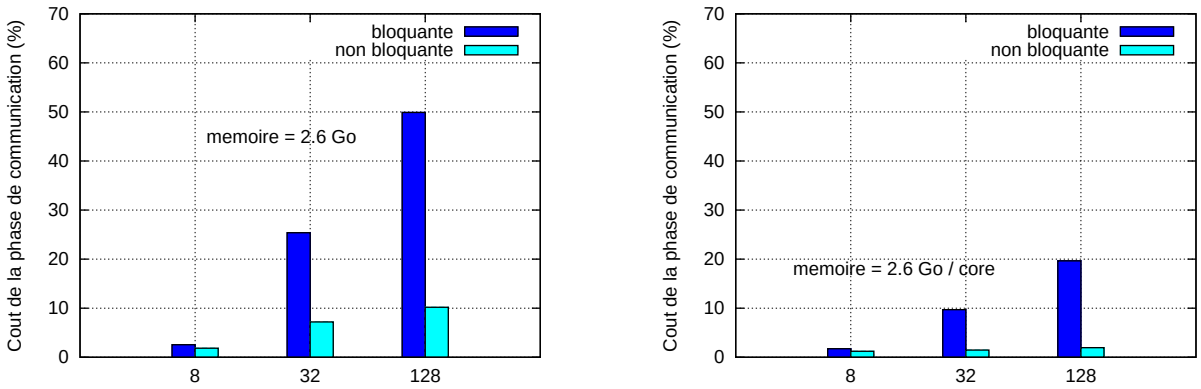

FIGURE 4.3 – Mesure de l'impact de la stratégie de communication dans le cas d'une implémentation basée sur le modèle de programmation MPI. La taille des données est fixe (gauche) ou elle augmente avec le nombre de cœurs (droite).

Les deux courbes de la figure 4.3 décrivent les résultats obtenus avec la librairie MPI. Nous comparons les implémentations utilisant des stratégies bloquantes ou non bloquantes. Dans le cas d'un problème de taille modeste (2.6 Go), le choix du mécanisme de communication est déterminant. En considérant plus de 8 cœurs, les communications bloquantes deviennent très coûteuses ; elles représentent plus de 25 % du temps total de calcul sur 32 cœurs et 49% sur 128 cœurs. Les communications non bloquantes représentent donc la seule option envisageable pour des problèmes présentant un grain fin au niveau du parallélisme. On remarque néanmoins une augmentation du ratio du temps de communication atteignant jusqu'à 10% du temps total sur 128 cœurs. En effet, lorsque le nombre de cœurs utilisés augmente, les possibilités de recouvrement des communications par du calcul diminuent. Cette tendance devrait donc s'aggraver en utilisant un plus grand nombre de nœuds de calcul. Nos expériences semblent montrer que 500 000 points est une valeur critique pour la taille des sous-domaines afin d'obtenir un recouvrement optimal. Cette valeur est extrêmement dépendante des performances de l'interconnexion entre les nœuds (réseau Infiniband dans notre cas) et des différentes caractéristiques de l'architecture et de la librairie de communication.

La courbe de droite de la figure 4.3 montre le comportement régulier de l'approche non bloquante avec une taille de problème qui croît avec l'augmentation des ressources de calcul. Le volume de calcul est quasiment constant dans ce cas. Dans le cas bloquant, les directions et les volumes de communication pour chaque processeur demeurent a priori constants. Cependant le nombre total de communications effectuées augmentent. La part du temps de communication par rapport au temps de calcul devrait être quasiment constante dans un cas idéal. Il est probable que des phénomènes de contention apparaissent au niveau du réseau, conduisant au surcoût observé. Les deux courbes de la figure 4.4 permettent de comparer les implémentations MPI et hybride.

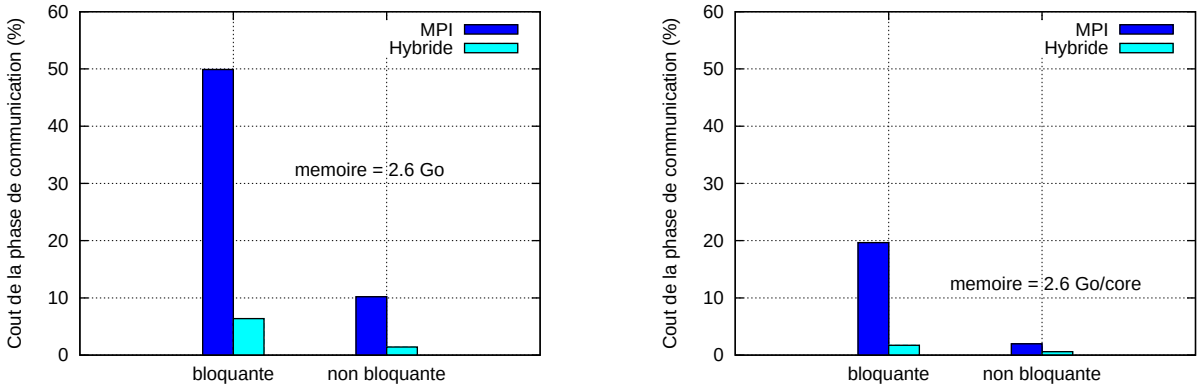

FIGURE 4.4 – Mesure de l'impact de la stratégie de communication dans le cas d'une implémentation basée sur le modèle de programmation hybride. La taille des données est fixe (gauche) ou elle augmente avec le nombre de cœurs (droite).

Nous présentons uniquement les résultats sur 128 cœurs. L'écart de performances lié au modèle de programmation est particulièrement significatif. Le volume de communication est inférieur à 10 % du temps total de calcul dans tous les cas. La situation combinant parallélisme à grain fin et communications bloquantes présente un gain important car la part des communications passe de 49% à 6.3% entre une approche MPI et une approche hybride. Dans le cas asynchrone, nous bénéficions du meilleur ratio volume/surface permettant de mieux recouvrir les communications par du calcul. De manière générale, le nombre de messages est réduit et leur taille plus importante permet aux mécanismes sous-jacents d'optimisation de la librairie de communication de jouer pleinement leur rôle.

Contrairement aux conclusions que l'on pourrait intuitivement proposer, l'approche bloquante peut constituer une alternative dans le cas d'un modèle de programmation hybride. Ce point est particulièrement intéressant dans le contexte des algorithmes de *time skewing* décrits précédemment. En effet, dans ce cas les possibilités de recouvrement des communications par les phases de calcul sont difficiles à exprimer et une phase partielle ou totale de communications bloquantes entre sous-domaines distants semble inévitable.

Les meilleures performances sont obtenues en utilisant une stratégie non bloquante avec un volume de communication inférieur à 2% dans les deux cas. L'implémentation hybride permet donc de considérer efficacement le parallélisme à grain fin et se révèle donc précieuse pour l'exploitation efficace de l'aspect multicœur des architectures.

4.1.3 Validation sur un exemple synthétique

Dans ce paragraphe, nous proposons une première validation de notre mécanisme de virtualisation basé sur l'utilisation de deux niveaux de parallélisme. Nous nous intéressons particulièrement au mécanisme mixte liant macro et micro-domaines. Nous considérons un exemple simple de propagation des ondes sismiques dans un milieu hétérogène, la zone de calcul est de taille 500 × 500 × 325 points de grille. Il s'agit d'une version simplifiée de la modélisation du bassin de Nice décrite dans [56]. Le problème comporte 81 millions de points, avec dix points supplémentaires pour les couches absorbantes qui sont implémentées en utilisant la technique des CPML. Ce problème est mis en œuvre sur le cluster `Borderline` composé de 10 nœuds de calcul comportant chacun 8 cœurs. La bibliothèque MAi est utilisée au niveau des nœuds multicœurs afin d'optimiser le placement des données en utilisant une politique de placement cyclique . L'algorithme décrit par la figure 4.1 est implémenté.

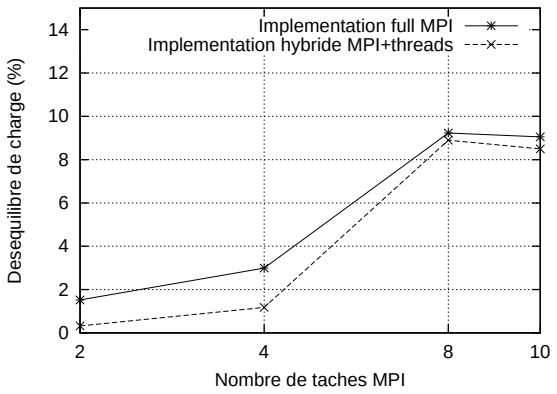

FIGURE 4.5 – Déséquilibre de charge au niveau des macro-domaines.

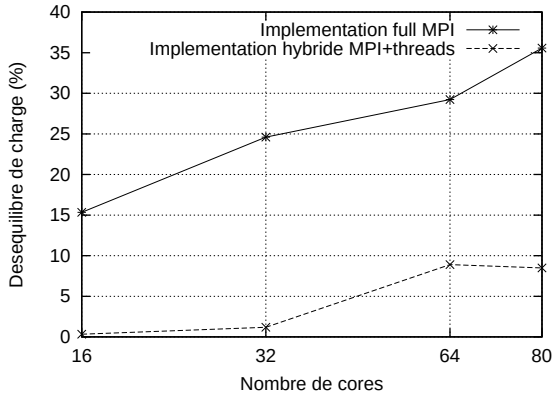

FIGURE 4.6 – Déséquilibre de charge en utilisant l'ensemble des cœurs disponibles.

Notre partitionnement hybride repose sur la prépondérance du niveau de parallélisme MPI correspondant aux macro-domaines. L'efficacité de cette décomposition est donc conditionnée par un équilibrage de charge quasi parfait entre les différents cœurs d'un même nœud de calcul. Nous comparons une implémentation complètement basée sur la librairie MPI à une approche hybride. Dix processus MPI (un par nœud) sont utilisés dans le premier cas ; l'implémentation hybride utilise les 80 cœurs disponibles (10 processus MPI et exploitation des cœurs avec les threads MARCEL). La figure 4.5 présente les résultats.
Les courbes montrent des évolutions similaires du déséquilibre de charge avec un maximum autour de 9%. Nous utilisons 772 threads à l'intérieur de chaque macro-domaine avec un surcoût quasiment nul. Au niveau des micro-domaines, le déséquilibre de charge entre les différents cœurs est inférieur à 2% . La régulation de la charge étant quasi optimale à l'intérieur de chaque nœud, le partitionnement hybride nous permet donc une multiplication par 8 du nombre de cœurs utilisés sans pénalité.

La figure 4.6 compare les approches MPI et hybride en utilisant dans les deux cas tous les cœurs disponibles sur chaque nœud. Sur 80 cœurs, le déséquilibre de charge passe alors de 36% à 8%. Globalement, on remarque que le gain observé augmente avec le nombre de cœurs utilisés pour la simulation, ce qui est un résultat attendu. En effet, l'augmentation du nombre de cœurs conduit à une dégradation plus rapide des performances dans le cas MPI par rapport

à l'implémentation hybride.

4.2 Séisme de Niigata-Chuetsu, Japon 2007

4.2.1 Description du problème

Le séisme du 16 juillet 2007 a causé d'importants dégâts dans la préfecture de Niigata au Japon, principalement autour de Kashiwazaki. Le choc principal, d'une magnitude de 6.6, a été suivi par de nombreuses répliques d'intensité importante (magnitude maximale de 5.6). On a recensé 15 morts et environ 2000 blessés. Parmi les nombreux bâtiments endommagés durant cet épisode, la centrale nucléaire de Kashiwazaki-Kariwa, située à une dizaine de kilomètres au sud de l'épicentre, a concentré toutes les attentions. Elle a dû être complètement arrêtée suite aux secousses et son fonctionnement est aujourd'hui partiel. Il s'agit de la plus importante unité de production nucléaire dans le monde. L'étude et la caractérisation des mécanismes à l'origine de ce tremblement de terre constituent une problématique importante, notamment en ce qui concerne la difficulté de compréhension du mécanisme de la source sismique.

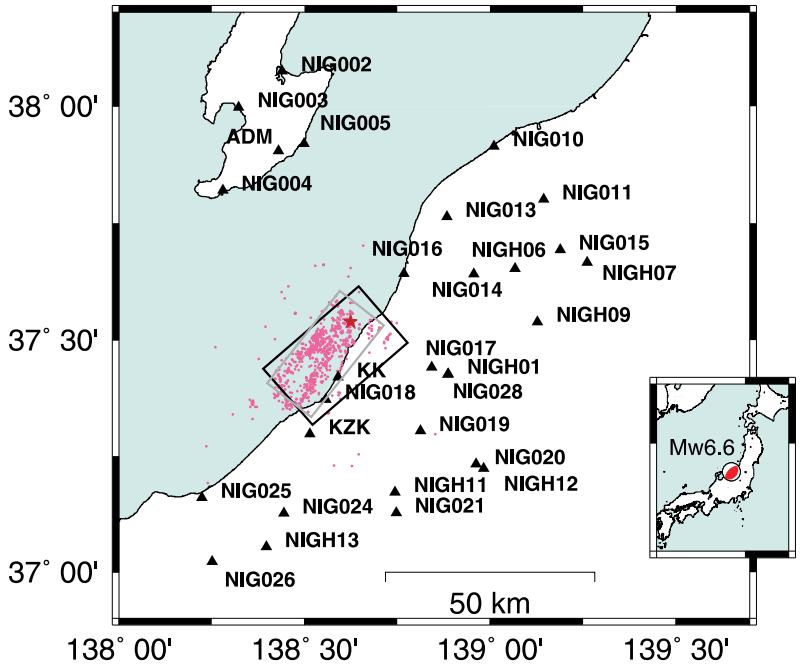

FIGURE 4.7 – Réseau d'observations et localisation de l'épicentre et des répliques pour le séisme de Niigata, 2007.

La figure 4.7 montre l'épicentre du séisme (étoile pleine rose située dans le rectangle) et la distribution des répliques. Les différentes stations d'enregistrement (réseaux *K-net*, *Kik-net*, *F-net* et *TEPCO*) sont également matérialisées par des triangles avec leurs noms. La localisation de l'épicentre est utilisée comme origine des simulations (x,y) = (0,0). La distribution des répliques, du 16 juillet au 29 août 2007, est représentée par une série de points roses autour de l'épicentre.

Les deux rectangles en noir et en gris représentent les modèles de sources sismiques proposés respectivement par Aoi et al. [12] et Hikima et al. [75]. Le réseau d'observation permet une bonne couverture de la zone d'intérêt ; cependant la zone de rupture est située sous le niveau de la mer

à une profondeur d'environ 10 km, ce qui complique la caractérisation de la source sismique. Les simulations réalisées dans ce contexte peuvent être comparées à de nombreuses observations en profitant de la densité du réseau accélérométrique japonais. Par rapport à la secousse principale, certaines stations font apparaître un comportement non linéaire du sol (par exemple la station *NIG018* du réseau *K-NET* sur la figure 4.7). Des effets non linéaires sont également supposés aux alentours du site nucléaire de Kashiwazaki-Kariwa. Ces résultats ne peuvent pas être comparés directement aux simulations effectuées ici car elles utilisent une loi de comportement linéaire.

Une autre difficulté est la complexité de la caractérisation du mécanisme de la source sismique. En effet, la géométrie de la faille est encore mal connue et les modèles proposés [12, 118] sont encore à préciser afin de pouvoir réellement simuler les mouvements forts. Ce point est critique car le mouvement du sol en champ proche de la source sismique est fortement affecté par le processus de rupture le long de la faille [11]. Afin de simplifier la discussion des résultats physiques, nous considérons la modélisation des répliques dont le mécanisme à la source est beaucoup plus simple que celui du séisme principal. Nous soulignons seulement les principales tendances des résultats.

Trois modèles tridimensionnelles de structure géologique ont été considérées, la figure 4.8 montre une coupe de chaque modèle. La zone comprend des failles actives identifiées notamment par le biais d'observations GPS [138]. La géologie est donc relativement complexe mais la structure de bassin est visuellement assez similaire.

- Le premier modèle (*ERI*) est obtenu par tomographie sismique à partir des différentes observations correspondant aux répliques du séisme principal [83]. Les résultats sont disponibles avec un pas minimum de 3 km au centre du modèle. En s'éloignant de la localisation des répliques, la résolution est comprise entre 10 km et 20 km dans les directions horizontales et verticales.

- Le second modèle (*NIED*) est disponible en ligne sur le site du *J-SHIS* (Japan Seismic Hazard Information Station [5]). Il couvre l'ensemble du territoire du Japon pour l'évaluation des mouvements forts à l'échelle nationale. Il est composé de 32 couches.

- Le troisième modèle (*GSJ*) propose une améloration du modèle *NIED* pour la région de Niigata avec une calibration détaillée des paramètres des couches géologiques [143].

La taille du domaine est de 110 km × 120 km × 30 km avec un pas de discrétisation spatiale de 100 m. Le pas de temps est de 0.005 s et la durée physique de simulation est de 60 s. Les sources sismiques cinématiques sont introduites sous forme d'un double couple de forces dans le cas de source ponctuelle [11].

4.2.2 Analyse des performances parallèles.

Les simulations sont effectuées sur la machine `Jade` en utilisant un maximum de 512 cœurs. Les performances des mécanismes d'équilibrage de charge sont étudiées sur un cas réel. L'impact de la hiérarchie mémoire n'est pas discuté ici et les nœuds de calcul utilisés étant composés de processeurs Intel Hapertown. La discrétisation du domaine physique de calcul conduit à un modèle comprenant 1100 × 1200 × 300 points. L'implémentation des conditions absorbantes de type CPML avec 10 points supplémentaires dans les directions horizontales et en profondeur conduit à un domaine de calcul de taille 423 millions de points. La consommation mémoire est de

5. http ://www.j-shis.bosai.go.jp

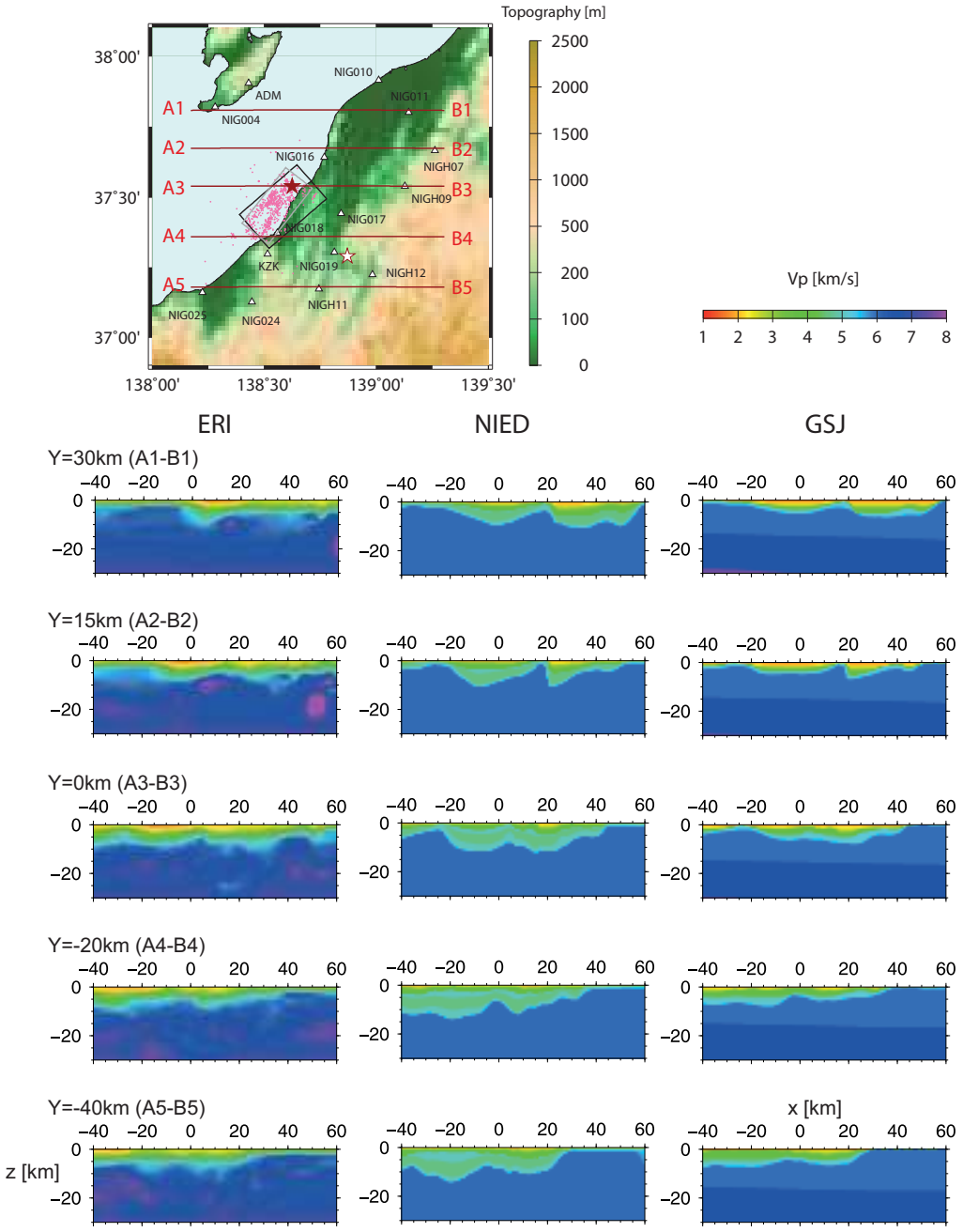

FIGURE 4.8 – Coupe des trois modèles tridimensionnelles de la géologie utilisée pour les simulations du séisme de Niigata, 2007.

65 Go pour cette simulation. Ces dimensions permettent de mesurer les effets d'un parallélisme à grain fin.

La figure 4.9 présente les résultats obtenus en appliquant différentes stratégies afin de limiter l'impact du déséquilibrage de charge dû au coût des bords absorbants. Dans le cas d'une implémentation reposant sur le modèle MPI, un découpage horizontal est utilisé. L'approche statique considère un partitionnement visant à équilibrer de manière fixe le nombre de points par sous-domaines. La variante quasi-statique ajuste la taille des sous-domaines en fonction du

coût des différentes zones de calcul. La pondération pour les différentes zones est calculée lors du prétraitement. Les implémentations hybrides statiques et quasi-statiques reposent sur les mêmes approches, la différence étant l'utilisation d'une technique de virtualisation des cœurs au niveau des nœuds de calcul.

Tout d'abord, on peut souligner le meilleur comportement des approches hybrides (statiques ou quasi-statiques) par rapport aux implémentations MPI. En utilisant 512 processeurs, le déséquilibre de charge passe de 38.2% dans le cas MPI classique à 7.7% dans le cas hybride. Si l'on compare les implémentations MPI, l'écart entre les deux stratégies est certes significatif mais ne permet pas d'envisager l'utilisation de l'approche quasi-statique sur un grand nombre de processeurs. En effet le déséquilibre reste relativement élevé sur 512 processeurs (21.5%), principalement en raison de la difficulté à évaluer précisément le poids de chaque zone. L'importance de la précision de ce pré-calcul est renforcé par le grain grossier de rééquilibrage. En effet, les plans dans les directions X et Y constituent les entités de base à migrer d'un sous-domaine à l'autre. Malheureusement le coût de calcul d'un de ces plans peut se révéler largement supéreur au déséquilibre mesuré.

La figure 4.10 permet de traduire les améliorations en termes de distribution de la charge en accélération du temps de simulation. Sur 512 processeurs, nous gagnons environ 25% de temps CPU en réduisant le déséquilibre de charge de 38.2% à 7.7%. De façon logique, ce gain croît avec l'écart entre les approches MPI classiques et les implémentations mixtes hybrides proposées. En termes de speedup, nous passons de 12.7 dans le cas MPI à 15.17 dans le cas hybride en prenant comme référence le temps de calcul sur 32 processeurs.

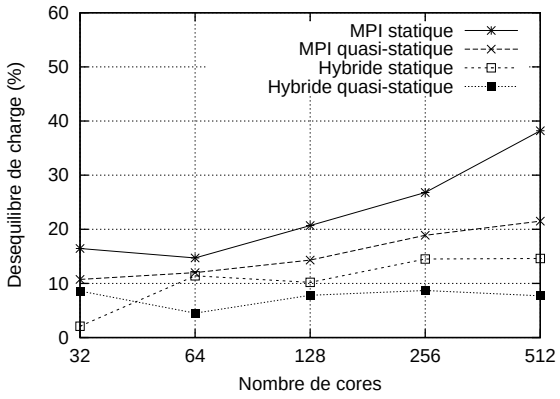

FIGURE 4.9 – Comparaison de différentes stratégies d'équilibrage de charge sur cluster de nœuds multicœurs.

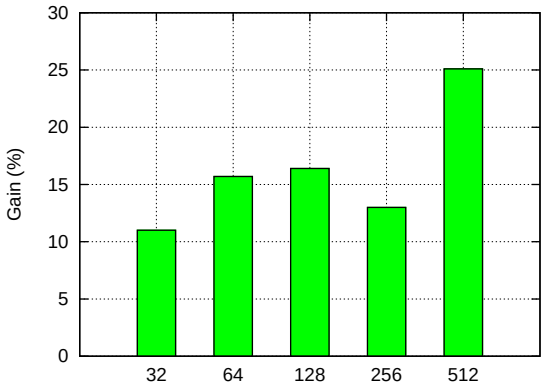

FIGURE 4.10 – Gain en temps CPU - Comparaison d'une version hybride quasi-statique par rapport à une version MPI.

4.2.3 Simulation des répliques

Nous considérons deux événements dont les caractéristiques (distance focale, magnitude) sont fournies de façon relativement précise par le réseau japonais d'observation *F-NET*. Il s'agit des répliques du 16 juillet 2007 (21h08 - Mw4.4) et du 18 juillet 2007 (16h53 - Mw4.3). La fonction source est décrite par une gaussienne de durée égale à 0.5 s.

Mouvement en champ proche

La figure 4.11 comparent les sismogrammes synthétiques et les observations pour ces deux répliques en considérant la station *KZK*. Cette dernière est localisée à 60 m de profondeur dans une zone de rocher. Les résultats ne sont pas filtrés. Les temps d'arrivée des ondes P sont cohérents dans tous les cas (composantes *UD*). Cependant, les résultats pour les ondes S sont assez variables. Le modèle *ERI* propose une bonne correspondance avec les observations, ce résultat s'explique par sa calibration basée sur des séismes et par la tomographie 3D qui couvre une large zone (de la source à la station *KZK*). Les modèles *NIED* et *GSJ* conduisent à un décalage de quelques secondes (composantes *EW* et *NS*). La structure de ces modèles nécessiterait un calibrage plus fin des paramètres. Les écarts importants concernant la première réplique (composante *EW*) avec le modèle *ERI* nécessitent également une analyse plus fine. Cette tendance se retrouve le long du littoral, notamment avec les stations *NIG018* et *KK* pour la première réplique.

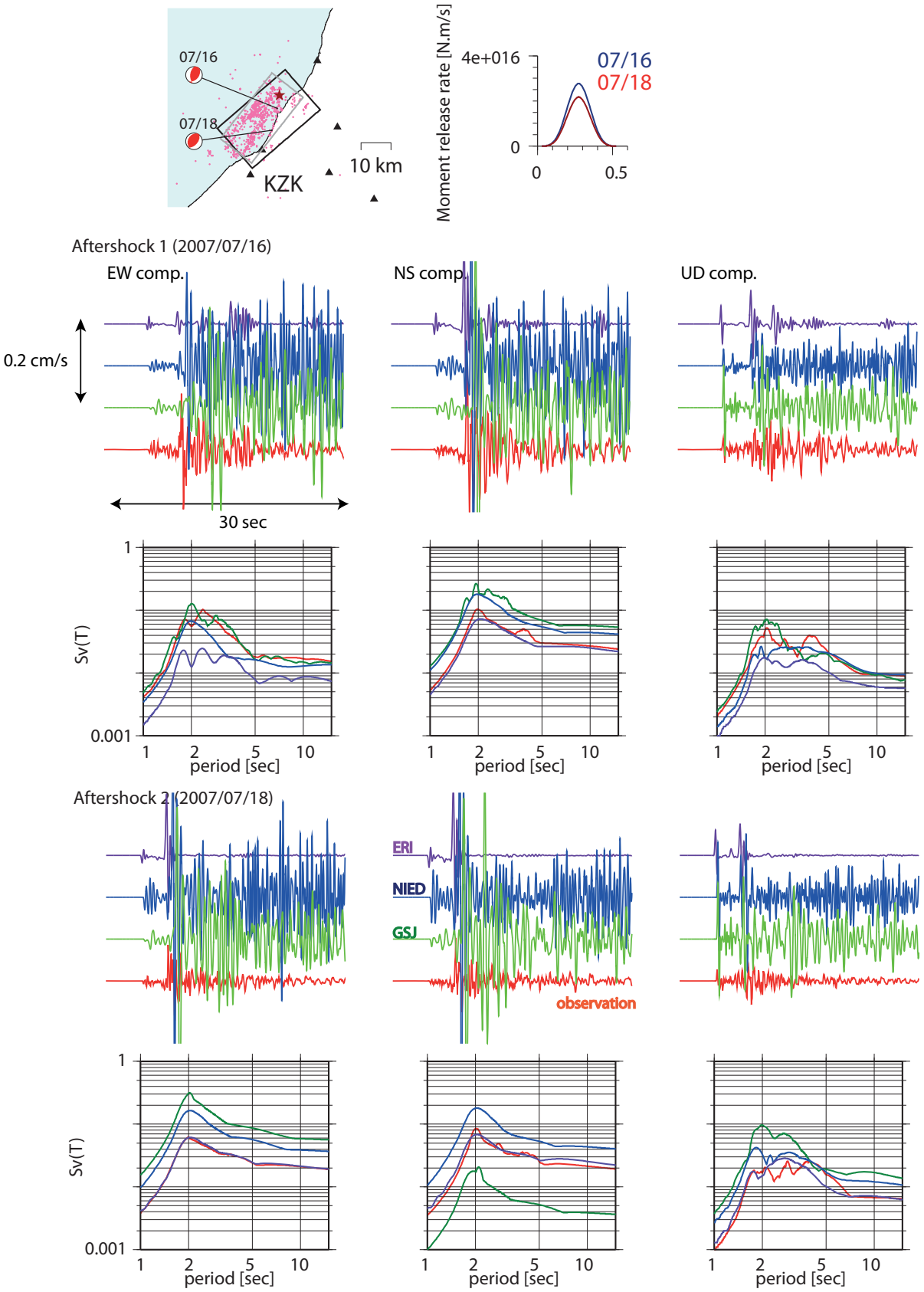

FIGURE 4.11 – Comparaison des sismogrammes synthétiques pour la station *KZK* pour les deux répliques.

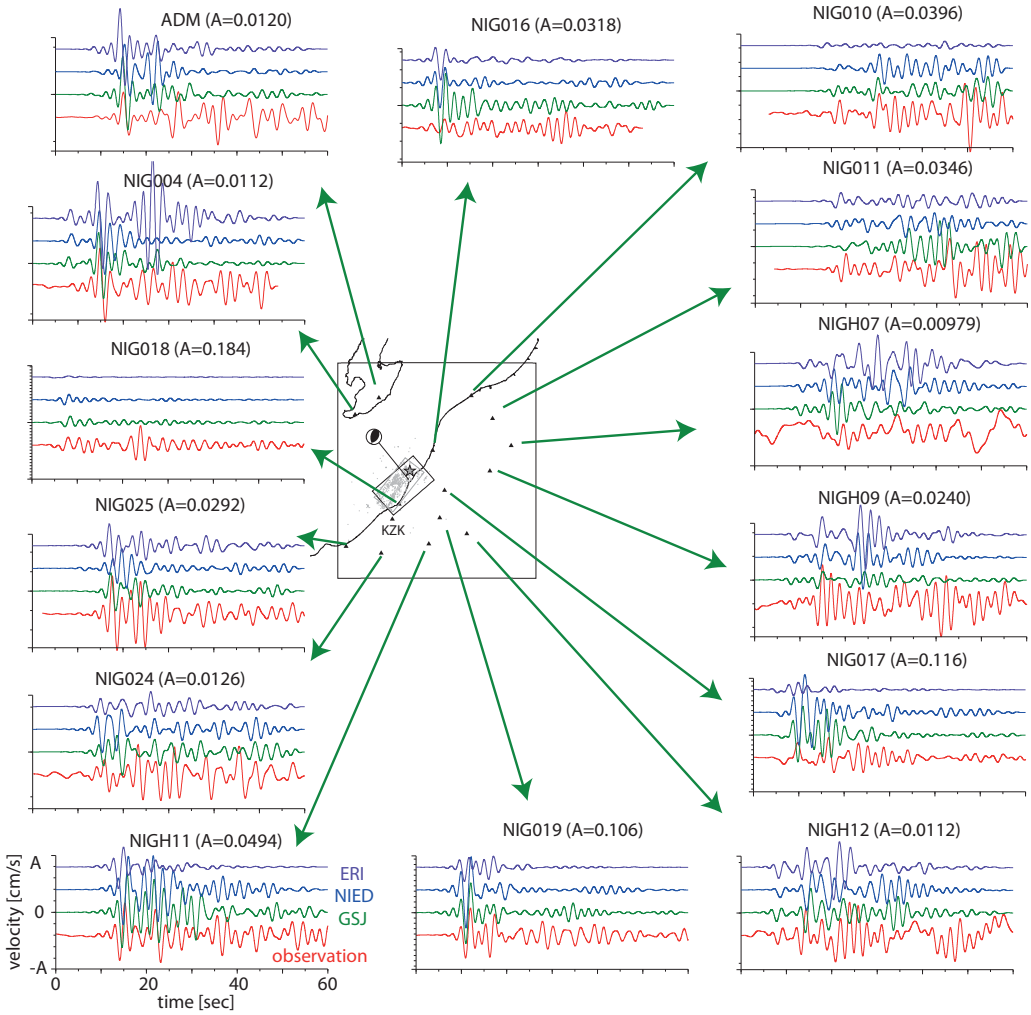

FIGURE 4.12 – Comparaison des sismogrammes synthétiques et des observations pour la réplique du 16 juillet 2007.

Echelle régionale

Les figures 4.12 et 4.13 présentent la comparaison pour quelques stations, des trois modèles de structure. Tous les signaux représentés sont filtrés entre 0.1-0.5 Hz et seule la composante *E-W* est représentée. L'analyse précise des résultats est assez difficile mais certaines remarques générales peuvent être faites.

Globalement, l'adéquation entre les simulations et les observations demeurent assez bonne pour la plupart des stations. Cela confirme une bonne sélection des paramètres pour la source et la structure. Néanmoins, un décalage en temps apparait par rapport aux observations, notamment pour les stations *NIG004* et *NIG025*. A l'instar des simulations en champ proche, les résultats obtenus pour la première réplique sont moins bons que ceux de la deuxième réplique. La description de la source ou de la structure tridimensionnelle seraient donc à préciser mais la correspondance des temps d'arrivée serait une indication de la correcte localisation de l'événement.

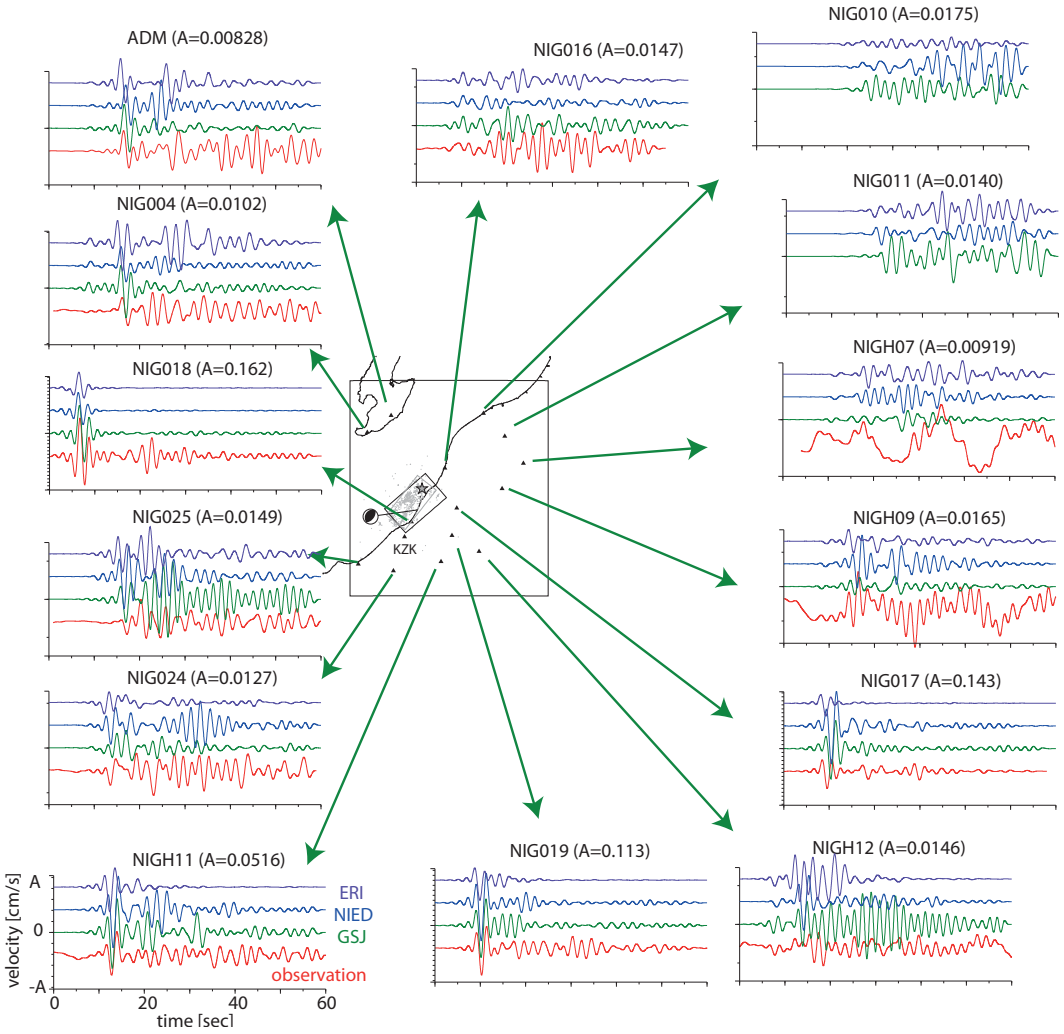

FIGURE 4.13 – Comparaison des sismogrammes synthétiques et des observations pour la réplique du 18 juillet 2007.

Deuxième partie

Modélisation non linéaire par la méthode des éléments finis

Chapitre 1

Algorithmique numérique et parallèle

Contents

La mise en œuvre de la méthode des éléments finis sur architecture parallèle est un problème largement discuté dans la littérature scientifique. Dans le cas d'un schéma d'intégration implicite en temps, deux phases critiques sont classiquement soulignées avec un impact important sur les performances parallèles. Il s'agit tout d'abord des phases de calcul matriciel correspondant à la résolution du grand système linéaire creux (plusieurs dizaines de millions d'inconnues) issu de la discrétisation. La stratégie utilisée à ce niveau est déterminante.

La phase d'assemblage des contributions élémentaires constitue l'autre point sensible en termes de performances. En effet chaque cœur devra procéder à l'évaluation des forces (internes ou externes) qui correspondent au sous-domaine attribué, l'objectif étant alors d'optimiser la phase d'échanges d'informations entre sous-domaines voisins.

Dans ce chapitre, nous verrons que la complexité des modélisations cibles nécessite de revisiter certaines approches traditionnellement implémentées. L'intérêt d'un deuxième niveau de parallélisme est notamment souligné avec l'exploitation de bibliothèques dédiées permettant d'étendre les performances.

1.1 Solveurs creux

1.1.1 Enjeux et justification des choix dans le contexte applicatif

La figure 1.7 du premier chapitre souligne l'importance des phases de calcul matriciel. Au niveau de la boucle en temps, il s'agit d'inverser la matrice de rigidité tangente à chaque pas de

temps afin d'en déduire un incrément de déplacement. La boucle itérative de Newton-Raphson nécessite la résolution fréquente d'un système creux de même taille que celui la phase précédente. La matrice utilisée lors de la procédure de linéarisation demeure inchangée au cours des itérations de convergence.

Le mauvais conditionnement des matrices à inverser constitue un autre un point critique. Cette propriété est directement liée à la physique du problème traité avec des géomatériaux de nature différente. Les couches de type sédimentaire présenteront un comportement mécanique plastique alors que les couches profondes de type rocher sont élastiques. La rigidité décroissante dans la couche sédimentaire conduit donc localement à des valeurs élevées de conditionnement. Les caractéristiques de la loi de comportement peuvent conduire à une perte de symétrie des matrices de rigidité au cours de la simulation. Ce point est notamment discuté dans [157]. Les solveurs itératifs, par exemple basés sur les méthodes de Krylov, peuvent être envisagés [137]. Ils présentent l'avantage d'une bonne efficacité sur un grand nombre de cœurs et d'une faible consommation mémoire. Deux points doivent néanmoins être soulignés :

1. L'algorithmique du problème conduit à une inversion de la matrice uniquement hors de la boucle de Newton-Raphson. Il est donc possible de réutiliser les facteurs calculés lors des phases précédentes à l'intérieur de la boucle de linéarisation. Cette propriété peut être avantageusement exploitée dans le cas d'un solveur direct qui permettra de stocker les cœfficients de la matrice triangulaire une fois pour toute, limitant ainsi les calculs à des phases de descente/remontée.

2. L'autre point concerne la stabilité numérique du problème. En effet, la nécessité d'améliorer le conditionnement du système conduit généralement à une longue phase d'expérimentations permettant d'optimmiser le choix du préconditionneur en fonction du problème. Cette procédure est difficilement compatible avec la recherche d'une solution numérique robuste et générique [63, 103].

Une autre alternative consiste à baser la résolution linéaire sur des solveurs hybrides. Les approches récemment proposées [16, 76] permettent de combiner les avantages des solveurs directs (en termes de robustesse) et des solveurs itératifs (en termes de consommation mémoire et de performances). Le problème algébrique est décomposé en une série de sous-problèmes résolus avec différentes stratégies. Une méthode directe est implémentée pour les sous-domaines intérieurs alors que le problème à l'interface est résolu avec une méthode itérative. Le comportement numérique et parallèle de ces solveurs doit encore être analysé de manière plus fine. Par construction, les performances obtenues dépendent de l'accélération correspondant à l'augmentation du nombre de cœurs (principalement la partie du problème résolue avec une méthode directe). La contrepartie de ces gains est une dégradation de la qualité du préconditionnement en raison du nombre croissant de sous-domaines, le nombre d'itérations de convergence nécessaire lors de la phase itérative augmentant alors de manière significative. Des modèles de programmation hybrides ont récemment été introduits afin de prendre en compte ce problème [64].

Dans le cas d'un problème tridimensionnel en géophysique (*SEG-EAGE Overthrust model*), une comparaison entre le solveur direct MUMPS [9] et l'approche proposée par A.Haidar [16] est détaillée par ce dernier. Cette comparaison conduit à :

– Un gain d'un facteur 2.3 en termes de consommation mémoire en faveur du solveur hybride.

- Un gain d'un facteur 22 en termes de factorisation numérique en faveur du solveur hybride.
- Un gain d'un facteur 5 en faveur du solveur direct pour la phase de descente/remontée. Dans le cas du solveur direct la matrice est déjà factorisée, ce qui n'est pas possible en utilisant une méthode hybride.

La procédure numérique utilisée dans notre cas implique un faible nombre de factorisations mais conduit à de nombreuses phases de descente/remontée. Les avantages provenant de l'implémentation d'une méthode hybride sont donc à approfondir.

Les solveurs directs semblent donc constituer le meilleur choix dans notre contexte applicatif. Dans le cas de décompositions LU, LL^t ou LDL^t, utilisées en fonction des propriétés de la matrice, ils conduisent à des solutions robustes et précises. De plus, l'algorithme *initial stress*, décrit dans le premier chapitre [158], permet de réutiliser la matrice factorisée un grand nombre de fois afin d'effectuer uniquement des phases de descente/remontée au cours de la boucle de linéarisation. Enfin, ces techniques offrent un grande robustesse numérique et sont largement utilisées dans le domaine industriel en dépit de leur importante consommation mémoire. Leur efficacité parallèle a été démontrée sur des architectures comportant plusieurs milliers de cœurs (8192 cœurs sont par exemple utilisés dans [69]). Le niveau de performance obtenu repose sur leur capacité à exploiter les bibliothèques BLAS de niveau 3 en s'appuyant sur une factorisation par blocs.

1.1.2 Intérêt du solveur Pastix

L'utilisation d'un solveur direct creux est la meilleure option en analysant l'algorithme décrit par la figure 1.7. Son intégration est généralement aisée en tenant compte de la structure matricielle utilisée pour l'application cible. Parmi toutes les bibliothèques disponibles (MUMPS [9], SuperLU [98], WSMP [68],...), le choix est souvent effectué de manière aléatoire car les solveurs sont généralement utilisés en boite noire. Notons que le solveur MUMPS a été utilisé avec succès pour des applications en géophysiques [126]. Des progrès importants ont été réalisés afin de réduire la consommation mémoire par des approches algorithmiques [8] ou en s'appuyant sur une implémentation out-of-core [4]. L'exploitation efficace des architectures hiérarchiques est encore à étendre. Une comparaison récente de différentes bibliothèques d'algèbre linéaire est disponible dans [69] avec une analyse de leurs performances à grande échelle. Trois points critiques motivent notre choix :

- L'exploitation efficace d'un grand nombre de cœurs.
- La limitation de la consommation mémoire.
- La prise en compte de l'évolution récente des architectures de calcul.

Le solveur Pastix répond à ces différentes exigences. Les références suivantes [71, 132] proposent une description précise des différents algorithmes sur lesquels ils reposent. Un des points clés est le choix d'un pivotage statique pour la factorisation. Dans ce cas, la structure par blocs des facteurs et les opérations numériques à effectuer sont connues par avance ce qui permet une régulation statique de la distribution des données et de l'ordonnancement des tâches. L'exploitation de l'architecture sous-jacente est donc optimisée par un couplage fort entre l'algorithmique et les caractéristiques de la machine cible.

La prise en compte de la hiérarchie mémoire des architectures modernes est décrite dans la thèse de M.Faverge [54]. Un placement statique des données est effectué sur les différents nœuds

NUMA. Cette stratégie est couplée à un mécanisme de vol de travail tenant compte du placement initial. Les gains mesurés par l'auteur sont variables (entre 20% et 40%) en fonction de l'architecture et du problème.

La consommation mémoire des solveurs directs est un verrou important en vue de leur utilisation pour des modélisations avec plusieurs millions d'inconnues. Cette dernière augmente de façon importante et non linéaire avec le nombre de cœurs. Le surcoût mémoire provient principalement de la phase d'agrégation des contributions. Le solveur Pastix aborde ce problème en introduisant un deuxième niveau de parallélisme permettant de tirer partie du caractère hybride des architectures avec nœuds de calcul multiprocesseurs/multicœurs ([72]). Les communications requises au niveau des nœuds à mémoire partagée peuvent ainsi être avantageusement remplacées par des accès directs à la mémoire. Une approche mélangeant MPI au niveau des nœuds multicœurs et multithreading à l'intérieur des nœuds est donc utilisée. La consommation mémoire provenant des contributions distantes est donc réduite proportionnellement au nombre de threads utilisés, la version hybride permettant d'éviter de stocker les blocs-colonnes temporaires. Des performances optimales sont obtenues en utilisant une librairie MPI offrant un support complet du multithreading (*THREAD_MULTIPLE*).

1.2 Assemblage parallèle des contributions

1.2.1 Partitionnement par les techniques *node-cut* et *element-cut*

La qualité de la distribution des différents éléments du maillage sur les cœurs influe de façon importante sur les performances de la phase d'assemblage des contributions. En effet, l'assemblage au sens des éléments finis conduit à sommer les contributions des éléments ayant des sommets communs. Il s'agira donc de maximiser le nombre d'éléments géométriquement connectés sur un même cœur afin de limiter les échanges d'informations. L'autre point important est l'équilibrage de charge qui repose sur une répartition uniforme du nombre d'éléments entre cœurs.

La technique largement utilisée consiste à utiliser une description sous forme de graphe du maillage d'éléments finis. Les sommets de ce graphe correspondent aux éléments géométriques du maillage et les arêtes aux connexions entre les différents éléments. Cette approche permet de décrire de façon explicite la topologie du maillage. Des algorithmes issus de la théorie des graphes sont alors utilisés afin de mettre en œuvre un partitionnement efficace du problème. Des librairies spécialisées telles que METIS ou SCOTCH fournissent des partitions équilibrées minimisant les interconnexions.

Le problème peut également être décrit de façon duale, les nœuds du maillage définissant alors les sommets du graphe. Ces deux approches sont détaillées et comparées dans [128]. Typiquement l'approche *node-cut*, qui correspond au schéma 1.1, permet une attribution unique des éléments aux différents cœurs.

Dans ce cas, les partitions sont créées en effectuant une coupe le long des nœuds. L'un des avantages est d'exploiter l'unité de calcul significative pour le partage (les éléments et non pas les nœuds). En revanche, l'échange explicite des contributions entre sous-domaines distants est indispensable. En effet, les nœuds situés aux frontières de chaque sous-domaine ne connaissent pas l'ensemble des contributions correspondant aux éléments topologiquement connectés. La phase

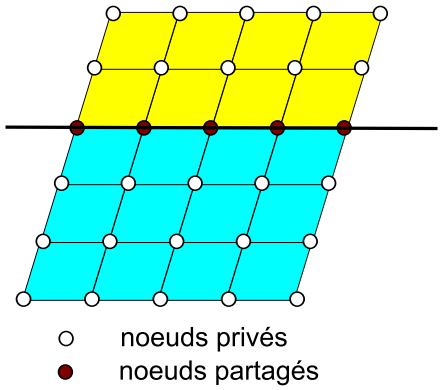

FIGURE 1.1 – Schéma de la méthode de partitionnement *node-cut*.

explicite de communication requise offre cependant d'importantes opportunités de recouvrement du temps de communication par du calcul. En effet les directions et le volume de communications nécessaires peuvent être déterminés statiquement lors de la phase de prétraitement. Il est alors possible de calculer les éléments frontières du domaine dans un premier temps et ensuite d'implémenter un schéma de communications non-bloquantes, puis de recouvrir le temps d'échanges par les calculs correspondant au reste du domaine. Une évaluation de ce type de stratégie peut-être trouvée dans [111] avec la comparaison des approches bloquantes ou non-bloquantes pour la méthode des éléments spectraux.

Une autre piste consiste à utiliser une coupe du maillage en traversant les éléments (figure 1.2). L'avantage est alors de disposer, pour chaque cœur, de toutes les informations nécessaires afin d'effectuer l'assemblage des contributions. Aucune phase de communications n'est requise à ce niveau. Le prix à payer est une duplication des éléments frontières entre les différents sous-domaines, le nombre total d'éléments évalués est supérieur à la taille initiale du maillage.

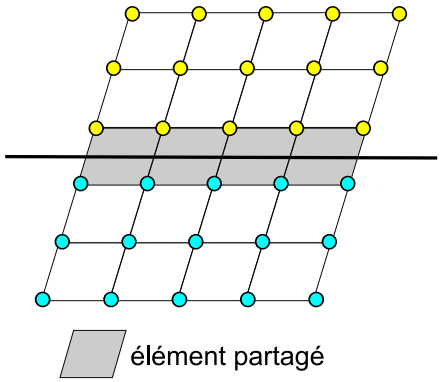

FIGURE 1.2 – Schéma de la méthode de partitionnement *element-cut*.

Les références [128] et [102] comparent ces deux méthodes par rapport à la taille du problème et au nombre de cœurs utilisés. Un résultat important est l'équivalence de ces deux approches (duales) en termes de scalabilité asymptotique. Leur mise en œuvre sur des problèmes concrets montre des limitations importantes. Intuitivement, il semble que le ratio volume/surface sera rapidement défavorable à l'approche *element-cut*. En effet, la taille de la zone tampon (ou

halo) correspondant aux éléments redondants augmentera significativement avec le nombre de cœurs utilisés. La topologie du maillage ou les caractéristiques numériques du problème pourront impacter la qualité du partitionnement de manière significative.

1.2.2 Influence de la topologie du maillage

Nous supposons un domaine tridimensionnel de forme parallélépipédique. Celui-ci est maillé avec le logiciel *GiD* de deux manières différentes en utilisant les options par défaut du mailleur : d'une part en utilisant des hexaèdres conduisant à un maillage structuré, d'autre part avec des tétraèdres. Les informations concernant ces maillages sont résumées dans le tableau 1.1. Dans les deux cas le nombre d'éléments du problème est fixé à 200 000.

	Cas structuré	Cas non structuré
Nombre de nœuds	210 681	36 261
Nombre d'éléments	200 000	199 916

TABLE 1.1 – Impact de la topologie du maillage sur l'approche *element-cut*.

Le partitionnement est effectué en utilisant la technique *element-cut*, la librairie METIS est utilisée afin de distribuer les éléments sur les différents cœurs. Le tableau 1.2 compare la qualité du partitionnement pour les deux maillages en mesurant le déséquilibre de charge. Par exemple, sur 64 cœurs le cas hexaédrique conduit à un déséquilibre de 33 %, valeur qui est significativement plus élevée dans le cas du maillage tétraédrique (49%). La tendance est la même en faisant varier le nombre de cœurs. Cet exemple simple illustre à la fois le niveau élevé du déséquilibre pour l'approche *element-cut* et l'impact de la topologie du maillage. Les résultats sont très défavorables au cas non-structuré. La régularité de la zone de recouvrement dans le cas hexaédrique semble limiter le déséquilibre de charge. La taille et la forme de cette zone sont variables dans le cas non-structuré ce qui explique les résultats obtenus.

Nombre de cœurs	**2**	**4**	**8**	**16**	**32**	**64**
Cas structuré (%)	1.33	3.80	5.98	12.22	21.95	33.22
Cas non structuré (%)	3.26	6.88	10.77	23.12	36.39	49.09

TABLE 1.2 – Déséquilibre de charge - cas de maillages structurés et non structurés

1.2.3 Influence de la physique du problème

Les modélisations cibles s'appuient sur des modèles géologiques tridimensionnels décrivant la structure du sous-sol et l'organisation des différentes couches. Un certain nombre de propriétés géomécaniques sont ensuite associées à ces couches géologiques. Par exemple, il est important de décrire finement les couches sédimentaires, ces dernières conduisant à une amplification du signal sismique.

Afin d'évaluer l'impact des caractéristiques du problème sur la qualité du partitionnement, un exemple préliminaire de modélisation des mouvements sismiques dans l'agglomération de Nice a

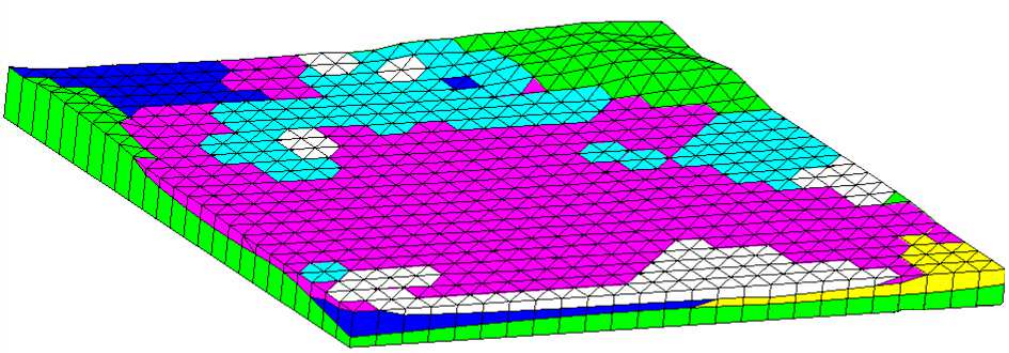

FIGURE 1.3 – Maillage utilisé pour le centre historique de la ville de Nice avec 5 couches sédimentaires et une couche de rocher (en vert).

été réalisé. Nous ciblons le centre historique de la ville (2 km × 2 km × 180 m), en supposant un séisme hypothétique en mer. Les aspects physiques sont plus largement décrits dans [56]. Le maillage est décrit par la figure 1.3. Il comprend cinq couches sédimentaires (bleu foncé - A7, violet - A456, blanc - A21, jaune - A8, bleu ciel - A3). La couche de rocher en profondeur est indiquée en vert (socle). Les caractéristiques détaillées des différentes couches sont données dans le tableau 2.1 en réutilisant les labels précédents. Les zones sédimentaires meubles sont modélisées en utilisant le critère élastoplastique de Mohr-Coulomb. La couche correspondant au rocher est modélisée avec le critère de Von Mises. La complexité en termes de description des mécanismes plastiques est plus grande dans le premier cas ; nous verrons que cela entraîne un coût de calcul plus important.

Label	Description	Vp (m/s)	Vs (m/s)	Densité (kg/m^3)
A21	sables, graviers, remblais	200	370	1800
A3	limons, argiles	200	375	1700
A456	sables, graviers	250	400	2000
A7	limons, argiles	260	490	1800
A8	sables, graviers	300	560	2000
Socle	rocher	1050	1700	2100

TABLE 1.3 – Description des couches dans le cas de la modélisation de la ville de Nice.

Le maillage utilisé est composé de 279 456 éléments tétraédriques. Les couches d'alluvions représentent 90 532 éléments, le reste du maillage permettant de modéliser le rocher. La taille des éléments varie entre 3 m et 24 m. Le calcul a pour objectif de permettre une analyse dans la plage de [0 ;7] Hz.

Le domaine est partitionné suivant l'approche *node-cut*. Le tableau 1.4 donne les accélérations mesurées pour la phase d'assemblage par rapport à une exécution séquentielle. Le serveur `Malm` est utilisé pour ces calculs. Les résultats décevants (accélération de 20.8 par rapport à une maximum théorique de 64 sur 64 cœurs) sont significatifs d'un important déséquilibre de charge entre sous-domaines, les phase de communication étant largement recouvertes par l'implémentation

de communications non-bloquantes. Ce déséquilibre se mesure tout d'abord en espace avec un écart maximum d'un facteur 4.8 entre sous-domaines avec un pas de temps fixe. L'écart est également important en temps avec un facteur de 5.4 pour un même sous-domaine au cours du temps.

En fonction du chargement imposé et de la zone considérée, les propriétés géomécaniques différentes du sol conduisent à des écarts significatifs au niveau du coût de calcul. Le caractère dynamique du déséquilibre de charge est donc directement lié à la modélisation en termes de géométrie/géologie (découpage des couches avec des propriétés différentes) et de propriétés physiques (modélisation non linéaire et chargement appliqué variable). Les approches statiques basées sur l'attribution d'un poids en fonction des zones de calcul ne permettraient donc pas d'améliorer significativement les résultats du partitionnement.

Nombre de cœurs	8	16	32	64
Accélération	4.2	7.8	13.5	20.8

TABLE 1.4 – Accélération de l'implémentation *node-cut* par rapport au cas séquentielle dans le cas de la modélisation de la ville de Nice.

1.3 Techniques de coloration de maillages

1.3.1 Cas des couches sédimentaires non linéaires

Les stratégies décrites dans la section précédente supposent un coût de calcul uniforme des éléments composant le maillage. La situation est très différente pour les modélisations envisagées à cause des lois de comportement complexes implémentées. Typiquement, les couches sédimentaires sub-surfaciques sont modélisées avec un comportement mécanique permettant l'apparition d'un phénomène de plasticité, ce qui n'est pas le cas pour les couches de rocher en profondeur. Cette différence de coût induit un problème dynamique de déséquilibre de charge. Différentes techniques sont alors envisageables.

Sur le plan algorithmique, les approches de type AMR (Adaptive Mesh Refinement) [22] constituent une piste intéressante. Elles ont été mises en œuvre récemment sur un grand nombre de cœurs pour l'équation de l'élastodynamique ([33]). Dans le cas général, il s'agit de suivre le front d'ondes afin de réduire les coûts de calcul dans les zones ou le champ d'ondes est négligeable. Malheureusement, cette approche semble assez peu adaptée. La première raison est le caractère irrégulier du domaine en termes d'interfaces ou de failles. Chaque onde sismique (onde P ou S) qui va se propager à travers une surface de discontinuité peut générer jusqu'à quatre ondes (ondes P réfléchie et transmises, ondes S réfléchie et transmise) en fonction de son angle d'incidence et des caractéristiques des deux milieux. L'intérêt des modèles venant souvent de leur complexité, des ondes se propagent donc rapidement dans tout le milieu et rendent cette technique peu adaptée.

Des bibliothèques de redistribution dynamique telles que PYRAMID [124] ou PARMETIS [140] semblent également trop lourdes à mettre en œuvre. Le caractère dynamique en espace et en temps de notre problème conduirait à de fréquentes phases de redistribution afin d'équilibrer la répartition des calculs au cours du temps. Le coût de ces phases de repartitionnement est donc

à mettre en balance avec les gains espérés.

Enfin, les approches reposant sur l'exploitation de supports d'exécution offrant des mécanismes de régulation de charge (telles que les techniques de virtualisation des cœurs décrites dans le chapitre 2) sont difficiles à mettre en œuvre dans notre cas notamment à cause de la structure du code et du langage utilisée. De plus, l'architecture générale de GEFDYN est largement basée sur l'emploi de zones mémoires communes (COMMON) rendant périlleux l'introduction d'un deuxième niveau de parallélisme.

Les approches décrites précédemment sont relativement génériques, il est cependant possible d'exploiter le contexte. Les points importants sont les suivants :

- Les couches géologiques en profondeur ont un comportement mécanique quasiment homogène et élastique.
- Le déséquilibre de charge provient essentiellement des couches sédimentaires.
- Au cours du temps, deux éléments géométriquement proches ont une probabilité élevée d'avoir un comportement mécanique similaire (par exemple de rentrer en plasticité) au niveau de la couche sédimentaire. Cette probabilité est plus faible avec deux éléments distants.

Les remarques précédentes (principalement les points 2 et 3) conduisent à proposer une décomposition spécifique pour les couches sédimentaires. Il s'agit de distribuer sur des cœurs différents les éléments du maillage géométriquement proches. Cette stratégie permet de minimiser la probabilité d'avoir deux éléments appartenant à la même zone du domaine de calcul sur un même cœur. Le principe adopté est ainsi l'inverse d'une décomposition standard d'un maillage consistant à regrouper les éléments proches. L'attribution d'un nombre sensiblement équivalent d'élements sur chaque cœur demeure.

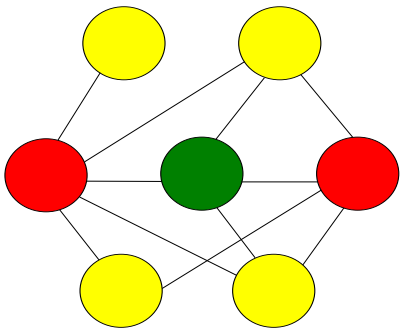

FIGURE 1.4 – Exemple de coloration d'un graphe avec sept sommets, le nombre chromatique est trois dans ce cas.

La méthodologie proposée est assez voisine des algorithmes de coloration de graphe. Dans le cas classique, il s'agit de pouvoir colorier un graphe en déterminant le nombre minimum de couleurs nécessaires afin que deux sommets connectés ne soient pas identifiés par la même couleur. Ce nombre minimum de couleurs est appelé nombre chromatique. La figure 1.4 résume ce problème avec un graphe comportant sept sommets, le nombre chromatique dans ce cas étant trois. La détermination de cette valeur constitue un problème difficile voire NP-complet dans le cas général. Un certain nombre d'heuristiques sont disponibles dans la littérature afin de résoudre

ce problème. Les références suivantes proposent une synthèse de la théorie des graphes et une introduction aux problèmes de coloration de graphe [21, 106].

Par rapport aux techniques habituelles de coloration, notre objectif est d'utiliser un nombre de couleurs fixe correspondant au nombre de cœurs que nous souhaitons utiliser pour la simulation ; c'est un problème de *k-coloration*.

Le deuxième point important est l'obtention d'une répartition uniforme des éléments du maillage sur les différents cœurs. Cette contrainte n'est pas critique pour les techniques standard de coloration de graphe. Dans notre cas, nous souhaitons à la fois maximiser la distance entre deux éléments appartenant à la même couleur et répartir de manière optimale le nombre total d'éléments du maillage.

Algorithm 5 Algorithme de coloration de maillage

for $i = 1$ to *ensemble des éléments* **do**
  for $i = 1$ to *ensemble des couleurs* **do**
    calculer le nombre d'éléments dont la $distance \leq dist$
    calculer le déséquilibre en cas d'affectation à la couleur courante
  end for
  Restriction du choix en fonction de la distance dans la liste des couleurs en fonction du paramètre *crit*.
  Choix de la meilleure couleur en termes de déséquilibre de charge.
end for

L'algorithme 5 décrit la stratégie utilisée pour le partitionnement de la zone supérieure du maillage qui correspond aux couches sédimentaires. Il s'agit d'un algorithme glouton dont l'objectif est de maximiser la distance entre les éléments situés sur un même cœur. Un double critère est utilisé, la distance entre éléments est une première étape pour l'affectation de l'élément courant à une couleur. Le choix définitif de la couleur doit permettre de favoriser l'équilibrage du nombre d'éléments par cœur. Un sous-partie de la liste des couleurs possibles en termes de distance à l'intérieur de chaque couleur est alors retenue. Le paramètre *crit* permet de réduire la liste initiale. Les distances entre éléments sont calculées entre barycentres. L'algorithme consiste à compter le nombre d'éléments situés dans la sphère ayant pour centre le barycentre de l'élément avec un rayon de *dist*. L'approche décrite est une heuristique pour résoudre ce problème, la recherche d'une solution optimale semblant difficile.

Le choix des paramètres *crit* et *dist* repose sur une phase d'expérimentation. Le premier permet de donner plus ou moins de poids à la distance ou à l'équilibrage de charge dans l'algorithme. Le second exprime une distance minimale entre éléments. Idéalement, le paramètre *dist* doit exprimer de manière physique la distance entre deux éléments ou zones susceptibles d'avoir des comportements mécaniques différents. Il semble difficile de proposer une valeur a priori.

1.3.2 Validation

Nous appliquons l'algorithme 5 au maillage correspondant à la modélisation de l'agglomération de Nice décrite dans la section 1.2.3. La taille moyenne des éléments est de 3 m dans la couche sédimentaire. La figure 1.5 montre un zoom sur le répartition des éléments du maillage dans le cas de l'application d'une technique de coloration pour des hexaèdres.

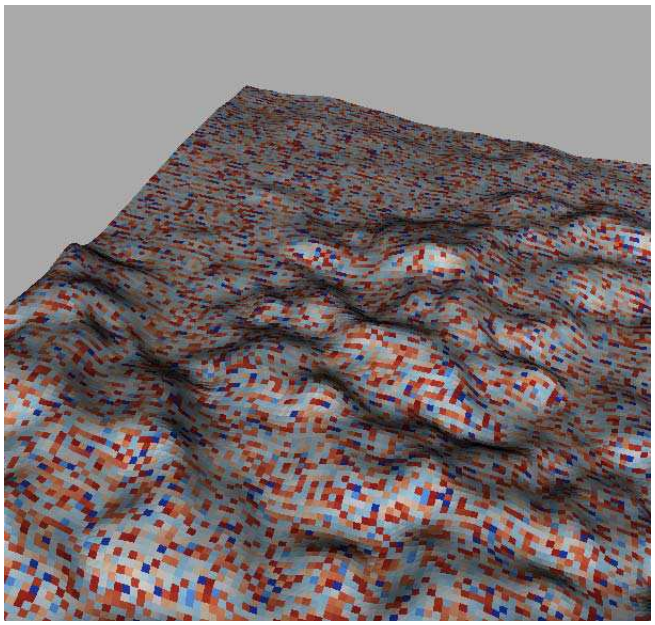

FIGURE 1.5 – Visualisation du maillage colorié de l'agglomération de Nice pour des hexaèdres.

Nous fixons tout d'abord le paramètre *crit* à une valeur de 0.5 qui permet d'équilibrer la décomposition à la fois en termes de répartition de la charge et du nombre d'éléments. L'algorithme glouton de coloration consiste donc à choisir pour chaque élément la couleur qui comporte le moins d'éléments dans la sphère de rayon $dist = 50\,\text{m}$. Le tableau 1.5 présente les résultats obtenus en termes de distances minimales entre éléments d'une même couleur. L'intervalle correspond aux valeurs minimales et maximales obtenues pour les différentes couleurs.

Nombre de cœurs	32	64	128	256	512	1024
Distances minimales (m)	2.3 - 6.4	2.8 - 9.0	3.2 - 14.	6.4 - 50.2	50 - 50.3	50 - 65

TABLE 1.5 – Intervalles des distances minimales en mètres entre éléments pour différents nombres de cœurs/couleurs.

On observe une amélioration des résultats avec l'augmentation du nombre de cœurs utilisés. En effet, asymptotiquement la situation optimale pour notre algorithme correspond à un nombre de cœurs identique au nombre d'éléments du maillage. Cette dernière n'est pas réaliste en termes de performances parallèles. L'écart moyen entre éléments appartenant à une même couleur est donc plus important quand le nombre de couleurs augmente.
La figure 1.6 présente la répartition moyenne des éléments à l'intérieur d'une couleur pour une partition avec 256 couleurs. Les cinq intervalles utilisés correspondent à la proportion d'éléments situés à moins de 250 m, entre 250 m et 500 m, entre 500 m et 750 m, entre 750 m et 1000 m, et à plus de 1000 m. L'histogramme illustre un bon respect du critère initial avec un très faible nombre d'éléments distants de moins de 50 m. Dans tous les cas la distance moyenne entre éléments d'une couleur est proche de 1000 m, quel que soit le nombre de cœurs.

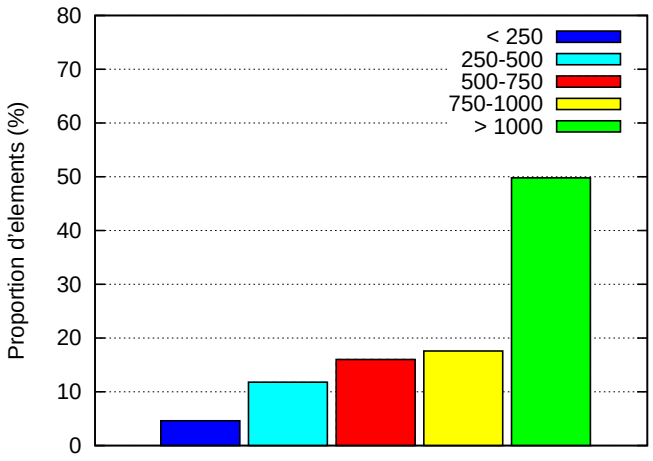

FIGURE 1.6 – Répartition des éléments en fonction des intervalles de distances pour une partition avec 256 cœurs/couleurs.

Le paramètre *crit* est déterminant sachant qu'il est effet difficile d'équilibrer à la fois le nombre d'éléments et d'optimiser la distance entre ceux-ci. Par exemple, avec 256 cœurs et une valeur de 0.1 pour le paramètre *crit*, le nombre d'éléments par couleur varie entre 1 et 536. Pour $crit = 0.2$, le déséquilibre de charge est de 385 % et à partir de $crit > 0.2$ la répartition des éléments devient optimale avec un écart inférieur à 1%.

1.3.3 Partitionnement à deux niveaux et schéma de communication

L'algorithme de coloration présenté dans la section 1.3 conduit à une approche à deux niveaux pour le partitionnement complet du domaine de calcul. Les couches géologiques profondes, qui représentent la part la plus importante du maillage, sont modélisées avec un comportement mécanique homogène. Les éléments du maillage correspondant à ces couches sont donc distribués avec un partitionnement *node-cut* standard. Les couches sédimentaires sont partitionnées en utilisant l'algorithme 5.

La figure 1.7 propose une représentation de la distribution obtenue dans les cas de couches profondes ou sédimentaires. En termes de géométrie, les deux approches se complètent afin de correspondre au domaine initial de calcul. En revanche, il n'y a pas de correspondance entre les nœuds de la zone homogène et ceux de la zone sub-surfacique. Au niveau de l'interface des deux méthodes, des nœuds identiques géométriquement peuvent donc être attribués à des cœurs différents. Des techniques spécifiques de raccordement sont donc à prévoir au niveau des procédures d'assemblage.

En effet, la technique de coloration de maillage conduit à l'utilisation de communications bloquantes car il n'est pas possible dans ce cas de définir de zones internes ou externes de la zone de calcul afin d'implémenter le recouvrement des phases de communication par le calcul des contributions pour les éléments intérieurs. Au sein d'une couleur donnée, la situation correspond à la maximisation de la distance entre éléments du maillage. Les connexions entre éléments étant minimales au niveau d'un cœur, tout raisonnement reposant sur le voisinage des sous-domaines est inadapté. L'implémentation des techniques habituelles de sommation de contributions suivie

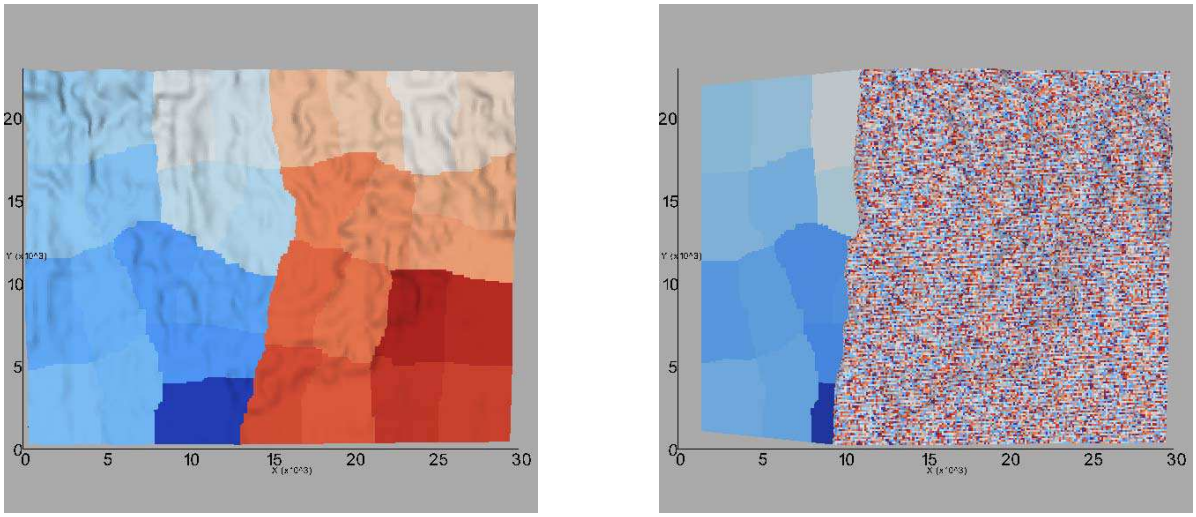

FIGURE 1.7 – Vue de dessus d'un maillage tridimensionnel. Les couches profondes sont partitionnées par la méthode *node-cut* (vue de gauche) et les couches sédimentaires par la méthode de coloration (vue de droite).

des phases de communications *point à point* n'est pas possible dans ce cas.

L'implémentation que nous proposons est donc basée sur des communications collectives MPI au niveau de la couche sédimentaire :

1. Les contributions élémentaires sont calculées pour l'ensemble des éléments correspondant aux couches sédimentaires. Cette partie du calcul est standard, seule la distribution des éléments est différente.
2. Les informations à échanger entre cœurs sont ensuite structurées. Il s'agit en fait de passer d'une représentation globale à une représentation locale des données.
3. Les contributions calculées sont échangées entre tous les cœurs pour tous les éléments appartenant aux couches sédimentaires. La fonction $MPI\_Allgather$ est utilisée. A ce niveau, le volume d'information échangé est directement proportionnel au nombre d'éléments affecté aux cœurs. Le nombre de cœurs utilisés et le nombre de communications augmentent donc de manière proportionnelle. En revanche le volume de données évolue de manière inversement proportionnelle.
4. La dernière phase est consacrée à l'assemblage, au sens des contributions, des informations locales et des informations issues des communications.
5. Le reste du maillage (les couches profondes) est partitionné en utilisant un algorithme de type *node-cut* et des communications non bloquantes pour l'échange des informations entre sous-domaines voisins. Les informations nécessaires afin de compléter les contributions entre les éléments frontière des couches sédimentaires et des couches profondes sont disponibles à partir de la communication collective précédente.

Un des points critiques de cette procédure est l'appel à la fonction $MPI\_Allgather$ au cours de la phase d'assemblage parallèle. Les performances de cette opération collective sont par exemple discutées dans [148]. En définissant α comme la latence, β comme la bande passante, p comme le nombre de cœurs et N comme la taille des données, un modèle simplifié de coût

donne $T = (p-1) \times \alpha + (p-1) \times N \times \beta$. Cette évaluation repose sur un algorithme simple pour l'opération $MPI\_Allgather$, c'est à dire des communications point à point effectuées de manière récursive sur un anneau virtuel de processeurs. Le coût se décompose en une part liée à la latence qui croît en fonction du nombre de cœur utilisée. La deuxième contribution dont le coût est lié à la bande passante s'équilibre entre le nombre de cœurs qui croît et la taille des données qui décroît. Dans notre cas, le coût de cette opération est donc guidé par la latence qui est de l'ordre de quelques microsecondes sur les architectures modernes de calcul dotées d'un réseau rapide. Ce modèle de coût est théorique et ne tient pas compte d'effets de bord tels que la contention mémoire ou les effets provenant du caractère multicœurs et hiérarchique des architectures. De plus, de nombreuses optimisations ont été proposées ou implémentées dans les bibliothèques MPI actuellement disponibles pour l'implémentation de cette opération collective (latence en $logP$, communications collectives non bloquantes, ...).

Chapitre 2

Exemple applicatif : étude de l'agglomération de Nice

Contents

Nous modélisons la propagation des ondes sismiques dans la région de Nice en considérant un séisme hypothétique localisé dans une zone de failles actives. L'importance de la prise en compte du contexte géologique local dans cette région a été soulignée par différentes références [56, 139]. Cependant, notre étude repose sur un certain nombre de simplifications tant au niveau de la géologie de la zone d'étude que des paramètres mécaniques du sol. L'objectif est de démontrer la faisabilité de modélisations non linéaires à grande échelle en conservant un contexte applicatif réaliste. Classiquement, la mise en œuvre de ce type de problème en mécanique des sols nécessite une phase lourde de calibration des lois de comportement et de caractérisation des matériaux. La stratégie adoptée ici permet une première validation de notre approche.

2.1 Description du modèle.

La figure 2.1 montre une vue générale de la zone d'étude. L'agglomération de Nice est matérialisée par un rectangle et l'étoile permet de localiser l'épicentre du séisme. Une représentation 3D de la zone de calcul est montrée sur la fgure 2.2. La géologie a été simplifié mais la topographie du modèle initial est conservée. L'origine O du repère (x, y, z) est située à la latitude/longitude $43.66°/7.11°$. L'axe des x est positif dans la direction ouest-est. La dimension du domaine est 30 km x 23 km x 10 km. Quatre couches géologiques sont modélisées. Les trois couches en profondeur sont constituées de rocher ("seismological and engineering bedrock") et la couche sub-surfacique est composée de sédiments. La couche géologique la plus profonde est supposé infinie, les trois autres couches ont une épaisseur de 900 m, 400 m and 100 m.

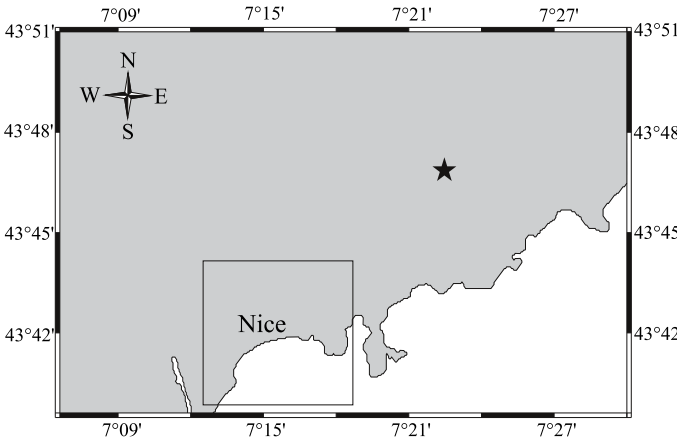

FIGURE 2.1 – Vue générale de la zone d'étude. Le rectangle situe l'agglomération de Nice et l'épicentre du séisme est localisé par une étoile.

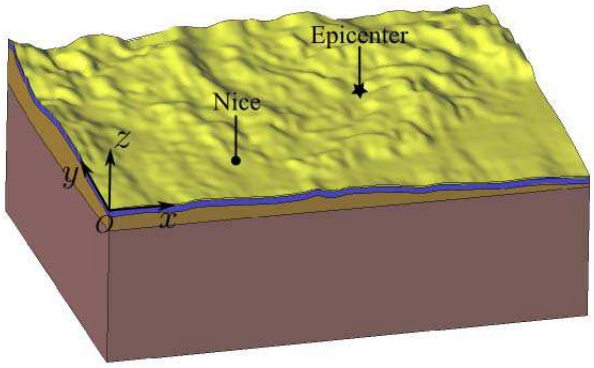

FIGURE 2.2 – Vue 3D de la zone d'étude et des différentes couches géologiques.

Les couches de rocher sont modélisées avec un comportement mécanique élastique et le critère élastoplastique de Mohr-Coulomb est utilisé pour la couche sédimentaire. Les paramètres élastiques des couches sont données dans le tableau 2.1.

Matériaux	Densité (kg/m^3)	Vp (m/s)	Vs (m/s)
Seismological bedrock 1	2200	4330	2500
Seismological bedrock 2	2100	2598	1500
Engineering bedrock	2000	1385	800
Sediments	1800	595	300

TABLE 2.1 – Description des quatre couches géologiques pour la modélisation de l'agglomération de Nice.

Concernant la couche de sédiments, le paramètre de cohésion non linéaire est défini à 100 kPa. Afin d'éviter d'éventuelles instabilités numérique, l'angle de friction interne (φ) est mis à zéro. De plus, ce choix permet d'éviter la phase statique d'initialisation des contraintes généralement

nécessaire avant la simulation dynamique non linéaire. Deux maillages ont été construits pour des fréquences maximales de la source sismique de 0.5 Hz et 0.6 Hz. Ces maillages définissent des problèmes de taille respective 2 470 593 inconnues (maillage grossier) et 5 632 011 inconnues (maillage fin). Le séisme simulé correspond à une zone active : la faille de Blausasc. La source sismique est introduite par un double couple de forces [6], les paramètres du plan de faille étant les suivants : azimut=204°, pendage=77° et angle de glissement= 15°. Les directions de radiation des ondes secondaires (figure 2.3) sont situées directement en direction de la ville de Nice. Les coordonnées de la source sont x=18.128 km,, y=10.139 km and z=-5 km. Une fonction de type tangente hyperbolique est utilisée pour définir la variation temporelle de la source sismique. Les récepteurs sont localisés aux alentours de la ville de Nice afin d'observer les effets de site. La durée physique de la simulation est 36 s.

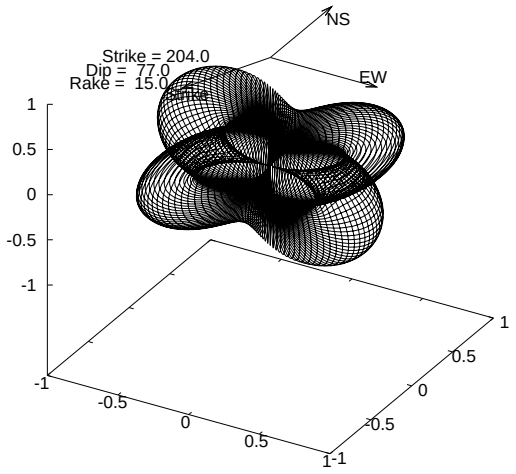

FIGURE 2.3 – Radiation des ondes S dans un milieu homogène infini pour le modèle de source utilisé pour cette étude.

2.2 Analyse des performances parallèles

2.2.1 Assemblage parallèle des contributions

Nous supposons un modèle homogène avec un comportement mécanique identique pour toutes les couches. Cette modélisation nous permet d'évaluer les performances des décompositions *node-cut* et *element-cut*. La figure 2.4 résume les résultats obtenus sur la plateforme de calcul `Jade` en utilisant un maximum de 1024 cœurs. Les accélérations sont données par rapport aux résultats obtenus sur 32 cœurs. Les performances sont satisfaisantes dans les deux cas avec un avantage pour la technique *node-cut*. Ces résultats sont cohérents avec les conclusions proposées dans [128] concernant le caractère dual de ces deux méthodes de partitionnement. Néanmoins, dans le cas de l'approche *element-cut*, la couche supplémentaire d'éléments introduit un déséquilibre de charge qui augmente significativement avec le nombre de cœurs utilisé. Cet écart est supérieur à 53% sur 1024 CPU avec le maillage fin.

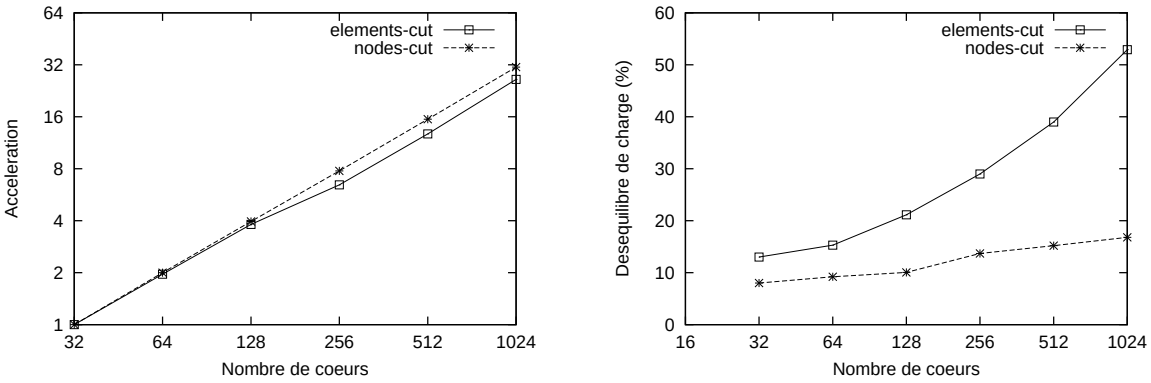

FIGURE 2.4 – Performance de la phase d'assemblage des contributions (gauche) et mesure du déséquilibre de charge (droite) avec les méthodes *node-cut* et *element-cut* pour le maillage fin.

Les tableaux 2.2 et 2.3 détaillent le volume de calculs supplémentaires effectués dans le cas de la décomposition *element-cut*. Le prix à payer afin d'éviter l'échange des contributions entre sous-domaines voisins est élevé. Pour le maillage à 0.5 Hz les calculs supplémentaires sont de l'ordre de 18% sur 256 cœurs et cette valeur atteint 25% dans le cas du modèle à 0.6 Hz sur 1024 cœurs. Ces tests soulignent les limites de cette approche à la fois en termes de déséquilibre de charge et de volume de calcul.

CPU	Calculs supplémentaires (%)
16	3.51
32	5.22
64	7.87
128	11.48
256	17.42

TABLE 2.2 – Volume de calculs supplémentaires pour le partitionnement *element-cut* dans le cas du maillage grossier.

CPU	Calculs supplémentaires (%)
64	6.37
128	9.22
256	13.49
512	18.87
1024	24.98

TABLE 2.3 – Volume de calculs supplémentaires pour le partitionnement *element-cut* dans le cas du maillage fin.

La figures 2.5 et le tableau 2.4 détaillent les résultats obtenus dans un cas plus réaliste. Les couches sont hétérogènes, la physique du problème est néanmoins simplifiée afin de limiter le comportement plastique au niveau des couches sédimentaires. Ce cas test permet de mesurer, l'impact du coût de calcul différent entre les lois de comportement utilisées (cas hétérogène). Le cas non linéaire correspond à la modélisation complète en introduisant toute la complexité nécessaire au niveau de la zone sédimentaire. Nous mesurons l'impact du comportement plastique de certaines zones du domaine de calcul, le maillage grossier étant utilisé pour ces simulations.

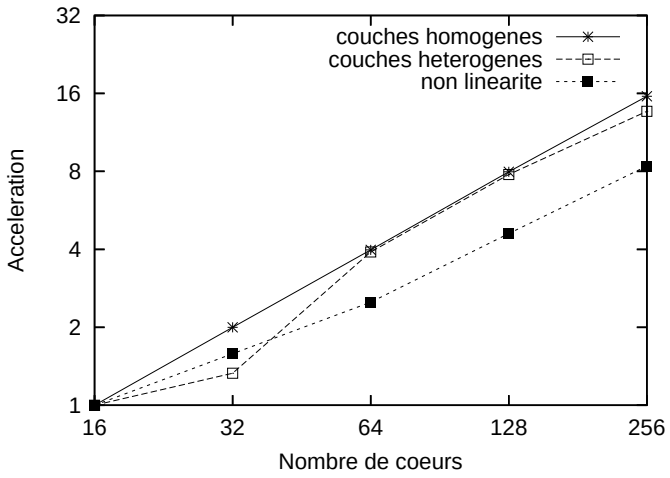

FIGURE 2.5 – Impact de la couche sédimentaire sur les performances du partitionnement *node-cut* pour le maillage grossier.

CPU	Homogène (%)	Hétérogène (%)	Non linéaire (%)
16	3.3	13.1	294
32	3.4	18.6	412
64	4.8	22.4	545
128	5.4	25.2	600
256	7.3	63.41	849

TABLE 2.4 – Impact de la couche sédimentaire sur les performances du partitionnement *node-cut*.Le déséquilibre de charge pour le maillage grossier est mesuré.

La courbe 2.5 illustre un impact relativement faible de la physique du problème sur le comportement parallèle de l'algorithme de partitionnement. L'accélération mesurée dans le cas non linéaire (autour de 8 par rapport à 16 dans le cas idéal) est acceptable mais le calcul reste très déséquilibré sur 256 cœurs. Les cas homogènes et hétérogènes présentent un niveau de performance proche du maximum théorique.

Le tableau 2.4 permet d'affiner cette analyse. En effet, un déséquilibre de charge de plus de 63% est mesuré dans le cas hétérogène. En tenant compte du comportement plastique de certaines zones du domaine (cas non linéaire), ce déséquilibre de charge passe alors à 849% sur 256 cœurs. Ces résultats montrent les limites de l'application des techniques standard pour ce type de modélisation.

2.2.2 Performances de la méthode de coloration de maillages

Les simulations basées sur la méthode *node-cut* sont utilisées comme référence dans cette section. Les figures 2.6 et 2.7 détaillent les résultats obtenus pour 10 s de simulation ce qui correspond au début de la phase de non linéarité physique. Les performances sont mesurées sur 64 cœurs.

La figure 2.6 présente, à chaque pas de temps, le déséquilibre de charge entre les sous-domaines de la couche sédimentaire pour l'approche *node-cut*. L'écart entre sous-domaines est assez irrégulier au début de la simulation avec de nombreux pics. La valeur moyenne commence à augmenter

significativement (écart de plus de 80%) avec l'entrée en plasticité de certaines zones du domaine de calcul. Cette augmentation brutale se produit autour de 10 s, les accélérogrammes de la figure 2.9 confirmant le début de la phase fortement non linéaire du calcul.

.

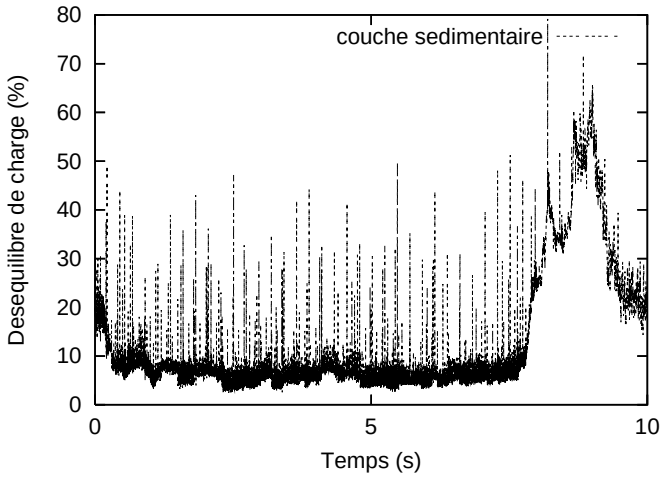

FIGURE 2.6 – Evolution au cours du temps du déséquilibre de charge induit par le coût CPU variable dans la couche sédimentaire.

La figure 2.7 montre des résultats complémentaires concernant le déséquilibre de charge cumulé au cours de la simulation. Les résultats obtenus avec un partitionnement hybride du maillage sont comparés à ceux de la technique standard *node-cut*. La première courbe concerne les couches profondes. Les résultats sont assez similaires car la technique de décomposition est identique dans les deux cas. Les valeurs mesurées pour le déséquilibre de charge sont très faibles (en moyenne inférieures à 5%), le comportement de ces couches étant considéré comme linéaire et homogène. La courbe de droite présente les résultats obtenus pour les couches sédimentaires. Une échelle logarithmique est utilisée pour l'axe vertical. Le bénéfice de la décomposition hybride est significatif. Le déséquilibre de charge est inférieur à 10% à comparer à une décomposition classique qui conduit à un écart moyen d'environ 40%.

Le tableau 2.5 compare l'accélération du calcul obtenue pour la modélisation complète de l'agglomération de Nice. Les résultats mesurés pour l'approche *node-cut* sur 32 cœurs sont utilisés comme référence. Dans le cas d'un partitionnement classique sur 256 cœurs, les performances sont satisfaisantes avec une accélération de 5.3 (par rapport à 8 comme maximum théorique). Ce niveau de performance est conforme à l'analyse de la courbe 2.5 du chapitre précédent. Les améliorations obtenues avec le partitionnement hybride sont en revanche significatives. Par rapport à l'approche initiale, l'accélération est de 18.8 sur 256 cœurs et un gain d'un facteur 2.7 est observé sur 32 cœurs.

Les performances de la phase de communications collectives requise pour l'échange des contributions entre sous-domaines de la couche sédimentaire étant déterminantes pour envisager la mise en œuvre à grande échelle. Le tableau 2.6 détaille les accélérations de cette phase jusqu'à 256 cœurs, les résultats sur 32 cœurs sont utilisés comme référence. Les valeurs sont proches de 1, ce qui illustre l'équilibre entre la réduction de la taille de messages et l'augmentation du nombre

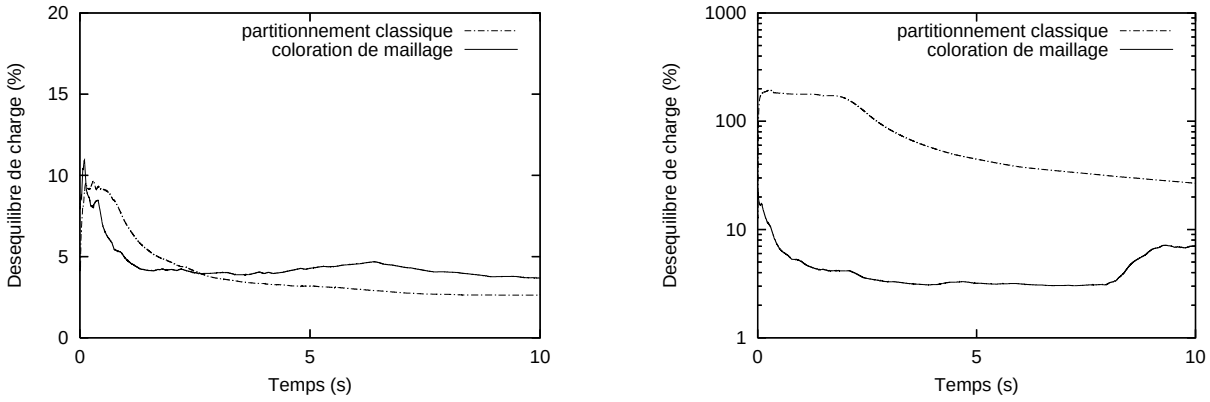

FIGURE 2.7 – Comparaison du déséquilibre de charge cumulé pour les couches profondes (gauche) et les couches sédimentaires (droite).

CPU	32	64	128	256
Nodes-cut	1	1.5	2.9	5.3
Coloration de maillage	2.7	5.1	10.2	18.8

TABLE 2.5 – Accélération du calcul pour les techniques de décomposition *node-cut* ou par coloration de maillage dans le cas d'un maillage grossier.

de messages à envoyer. Le profilage de l'application a montré que le coût de l'appel à la fonction *MPI_Allgather* représentait moins de 3% du coût total de la phase d'assemblage.

CPU	16	32	64	128	256
Echange global	1	1.22	1.12	0.87	0.90

TABLE 2.6 – Performances de la phase de communications collectives requise pour l'assemblage des contributions dans la couche sédimentaire.

2.2.3 Solveur direct parallèle

Dans cette section, nous analysons le comportement du solveur Pastix. Les caractéristiques des matrices issues des deux maillages (Nice05Hz et Nice06Hz) sont décrites dans le tableau 2.7. Ces matrices sont double précision, symétriques et définies positives. Nous détaillons leur dimension (nombre de colonnes) ainsi que le nombre de termes hors diagonale pour la partie triangulaire de la matrice tangente (NNZ_A) et pour la matrice factorisée (NNZ_L). Le nombre d'opérations nécessaires à la factorisation (LDL^T) est également précisé. Les plateformes `Borderline`, `Decrypthon` et `Jade` sont utilisées pour nos tests.

Scalabilité

Les tableaux 2.8, 2.9, 2.10 et 2.11 présentent les performances parallèles pour les phases de factorisation et de descente/remontée linéaire. Ils permettent de comparer les implémentations MPI et hybrides afin de mesurer l'impact du modèle de programmation. Les tests réalisés sur

Nom	Colonnes	NNZ_A	NNZ_L	Opérations
Nice05Hz	2 470 593	96.8×10^6	3.9×10^9	2.6×10^{13}
Nice06Hz	5 632 011	224.2×10^6	13.8×10^9	1.7×10^{14}

TABLE 2.7 – Caractéristiques des matrices issues des modélisations de la région de Nice.

la plateforme `Borderline` utilisent les possibilités de placement des données sur architectures à mémoire hiérarchique.

CPU	MPI	Threads	Factorisation (s)	Descente/remontée (s)
8	4	2	820	5.77
8	2	4	810	5.96
16	8	2	464	3.4
16	4	4	450	3.64
16	2	8	436	3.64
32	16	2	242	2.06
32	8	4	236	1.99
32	4	8	257	2.31
32	2	16	268	2.33

TABLE 2.8 – Temps pour les phases de factorisation et de descente/remontée sur la plateforme `Decrypthon` pour une approche hybride et le maillage Nice05Hz.

CPU	Factorisation (s)	Descente/remontée (s)
8	877	5.64
16	454	3.46
32	283	2.0

TABLE 2.9 – Temps pour les phases de factorisation et de descente/remontée sur la plateforme `Decrypthon` pour une approche purement MPI et le maillage Nice05Hz.

Dans le cas d'une implémentation hybride, la meilleure configuration correspond dans la majorité des cas au fait de maximiser le nombre de threads. Par exemple, avec 32 cœurs sur `Decrypthon` (deux processus MPI et 16 threads pour chaque processus MPI), le temps de factorisation est de 268 s. Le temps de factorisation est de 283 s pour une implémentation uniquement basée sur l'échange de messages.

Il est important de rappeler que la version MPI nécessite beaucoup plus de communication, principalement au cours de la phase d'agrégation des contributions. La différence relativement faible entre ces deux implémentations sur `Decrypthon` s'explique par le niveau de performances des communications en mode intra- et inter-node. En effet, la librairie MPI optimisée sur cette plateforme (implémentation *MPI IBM*) ainsi que la qualité du réseau (*IBM-Fédération*) permettent d'excellentes performances des communications et les deux implémentations du solveur Pastix conduisent donc à des résultats assez proches.

Les différences entre la version hybride et la version MPI sont plus significatives sur la plateforme `Borderline`. Par exemple sur 64 cœurs, le temps de factorisation est de 215 s pour

CPU	MPI	Threads	Factorisation(s)	Descente/remontée (s)
8	4	2	1240	3.98
8	2	4	1140	4.2
16	8	2	944	2.45
16	4	4	719	2.54
16	2	8	679	3.19
32	8	4	452	1.37
32	4	8	342	1.58
64	8	8	215	0.8

TABLE 2.10 – Temps pour les phases de factorisation et de descente/remontée sur la plateforme `Borderline` pour une approche hybride et le maillage Nice05Hz.

CPU	Factorisation (s)	Descente/remontée (s)
8	2110	4.57
16	1340	2.45
32	830	1.57
64	775	1.28

TABLE 2.11 – Temps pour les phases de factorisation et de descente/remontée sur la plateforme `Borderline` pour une approche purement MPI et le maillage Nice05Hz.

la version hybride mais de 775 s en mode MPI. Concernant la phase de descente/remontée, le gain en temps est de 37% sur cet exemple. Ces différences s'expliquent par la qualité inférieure du réseau (Myrinet-10G) par rapport à la plateforme `Decrypthon`. Les performances de la librairie MPI sont également assez différentes car il s'agit d'une implémentation (*MPICH2-MX*) proposant des optimisations moins avancées pour la gestion des communications intra-nœuds.

L'impact des communications peut également être analysé en comparant les résultats obtenus sur 16 cœurs. La différence entre un calcul utilisant 2 threads ou 8 threads à l'intérieur d'un nœud à mémoire partagée est de 30% sur `Borderline` mais n'est que de 5% sur `Decrypthon`. Ces comparaisons montrent l'importance de l'approche hybride afin d'optimiser les performances sur des architectures de calcul principalement basées sur des nœuds de calcul standardisés et interconnectées par de réseaux rapides de type Infiniband ou Myrinet. Le niveau de performance fourni par la plateforme IBM reste difficilement reproductible sur d'autres architectures.

CPU	Factorisation(s)	Descente/remontée (s)
16	359	4.99
32	175	2.53
64	108	1.44
128	64.9	0.76
256	46.6	0.46
512	39.6	0.30
1024	44.6	0.26

TABLE 2.12 – Temps pour les phases de factorisation et de descente/remontée sur la plateforme `Jade` pour une approche hybride et le maillage Nice05Hz.

Les tableaux 2.12 et 2.13 présentent le temps d'exécution pour la factorisation et la phase de descente/remontée linéaire sur la plateforme `Jade`. Nous exploitons l'implémentation hybride

CPU	Factorisation (s)	Descente/remontée (s)
64	673	4.62
128	369	2.57
256	237	1.44
512	196	0.91
1024	153	0.86

TABLE 2.13 – Temps pour les phases de factorisation et de descente/remontée sur la plateforme `Jade` pour une approche hybride et le maillage Nice06Hz.

avec 8 threads par nœud multicores, les tests sont réalisés sur un maximum de 1024 cœurs avec les deux tailles de matrice.

Pour l'exemple du maillage Nice05Hz, le temps de factorisation diminue jusqu'à 512 cœurs. Dans ce cas, la taille du problème ne permet pas de tirer partie de l'utilisation de plus de ressources. Ce comportement s'explique par le grain trop fin du problème au-delà de cette limite. Les résultats sont meilleurs pour l'exemple du maillage Nice06Hz, la matrice issue de ce maillage étant plus grosse et permettant d'exploiter 1024 cœurs avec une bonne efficacité. L'accélération mesurée est assez décevante dans les deux cas. Des difficultés (temps de simulation incohérents, situation de *deadlocks*) ont été rencontrées pour réaliser ces tests en mode THREAD_MULTIPLE de manière efficace. Il est probable que les implémentations MPI disponibles au CINES ne délivrent pas le niveau optimal de performance permettant de tirer le maximum de l'implémentation hybride du solveur Pastix, notamment à grande échelle. Cependant, le temps de calcul pour la phase de descente/remontée linéaire diminue régulièrement. Ce point est crucial pour l'algorithme de Newton-Raphson modifié implémenté au niveau du solveur éléments finis.

Consommation mémoire

La consommation mémoire, souvent jugée prohibitive par rapport aux solveurs itératifs, est classiquement identifiée comme un goulot d'étranglement pour la mise en œuvre à grande échelle des méthodes directes. Dans cette section nous évaluons l'apport d'une implémentation hybride par rapport aux approches MPI standard.

CPU	MPI	threads	mémoire (Gb)
128	64	2	186
128	32	4	92.9
128	16	8	53.2

TABLE 2.14 – Consommation mémoire sur 128 cœurs pour une implémentation hybride et le maillage Nice05Hz.

CPU	MPI	threads	mémoire (Gb)
128	64	2	430
128	32	4	230
128	16	8	180

TABLE 2.15 – Consommation mémoire sur 128 cœurs pour une implémentation hybride et le maillage Nice06Hz.

Les tableaux 2.14 et 2.15 détaillent les résultats obtenus sur 128 cœurs en utilisant différentes configurations en termes de nombre de processus MPI ou de nombre de threads. Les meilleurs résultats sont toujours obtenus en utilisant un nombre maximum de threads. La consommation mémoire est divisée par un facteur 3.5 pour l'exemple Nice05Hz et par un facteur 2.4 pour l'exemple Nice06hz. Ces résultats s'expliquent par l'allocation mémoire qui est effectuée au niveau des processus MPI, l'utilisation d'un plus grand nombre de threads permet donc de limiter le nombre de buffers temporaires nécessaires au stockage des contributions provenant des autres processus MPI.

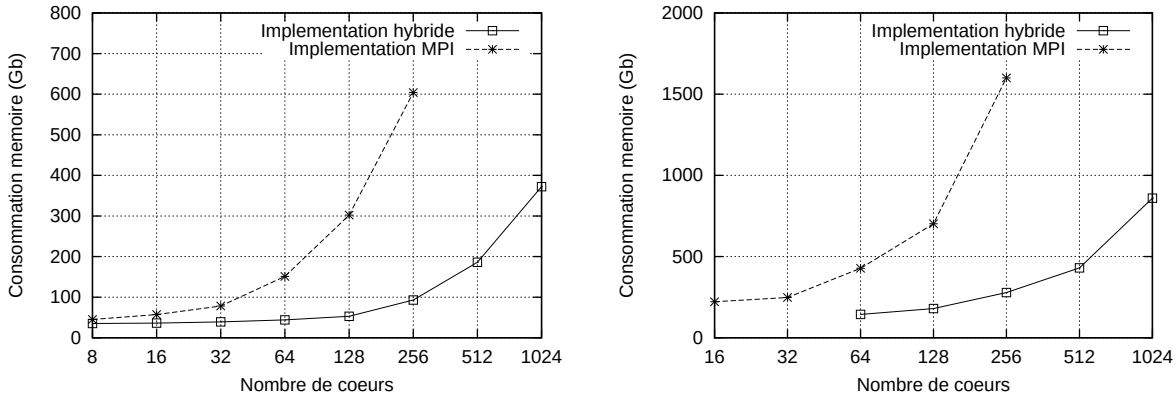

FIGURE 2.8 – Consommation mémoire pour les approches MPI et hybride et les maillages Nice05Hz (gauche) et Nice06Hz (droite).

Les deux courbes de la figure 2.8 présentent la consommation mémoire pour le cas des maillages Nice05Hz et Nice06Hz jusqu'à 1024 cœurs. Pour l'exemple du maillage Nice06Hz (courbe de droite), la consommation mémoire sur 256 cœurs dans le cas MPI nécessite d'effectuer le calcul sur 64 nœuds (1.6 TB) alors que l'implémentation hybride permet de factoriser la matrice sur seulement 16 nœuds (278 GB).

Le stockage des cœfficients de la matrice factorisée représente une part décroissante de la mémoire totale consommée lorsque le nombre de processus MPI augmente. Cela représente 112 Gb pour le cas du maillage Nice06Hz et 32 Gb pour le cas du maillage Nice05Hz. Les buffers dédiés au stockage temporaire des contributions représentent 66% de la mémoire totale pour le cas du maillage Nice05Hz sur 256 cœurs pour la version hybride de Pastix et 95% pour la version MPI pure. Les rapports sont de 59% et 93% pour le cas du maillage Nice06Hz sur le même nombre de cœurs.

2.3 Analyse des résultats physiques

Les résultats de trois simulations basées sur des séismes hypothétiques de magnitude croissante : ($M_w = 5.7$, $M_w = 6.0$ et $M_w = 6.2$) sont comparés. Un comportement exclusivement élastique ou potentiellement plastique pour la couche sédimentaire est supposé. La figure 2.9 présente les résultats obtenus pour les composantes du déplacement, de la vitesse et de l'accélération dans la direction est-ouest pour le maillage Nice05Hz.
L'objectif des simulations est de proposer une validation préliminaire des approches algorithmiques sur un exemple simplifié de l'agglomération de Nice. Les simulations effectuées ne per-

mettent pas une analyse quantitative fine des phénomènes à cause du choix arbitraire de certains paramètres. Elles sont néanmoins conformes aux résultats théoriques attendus.

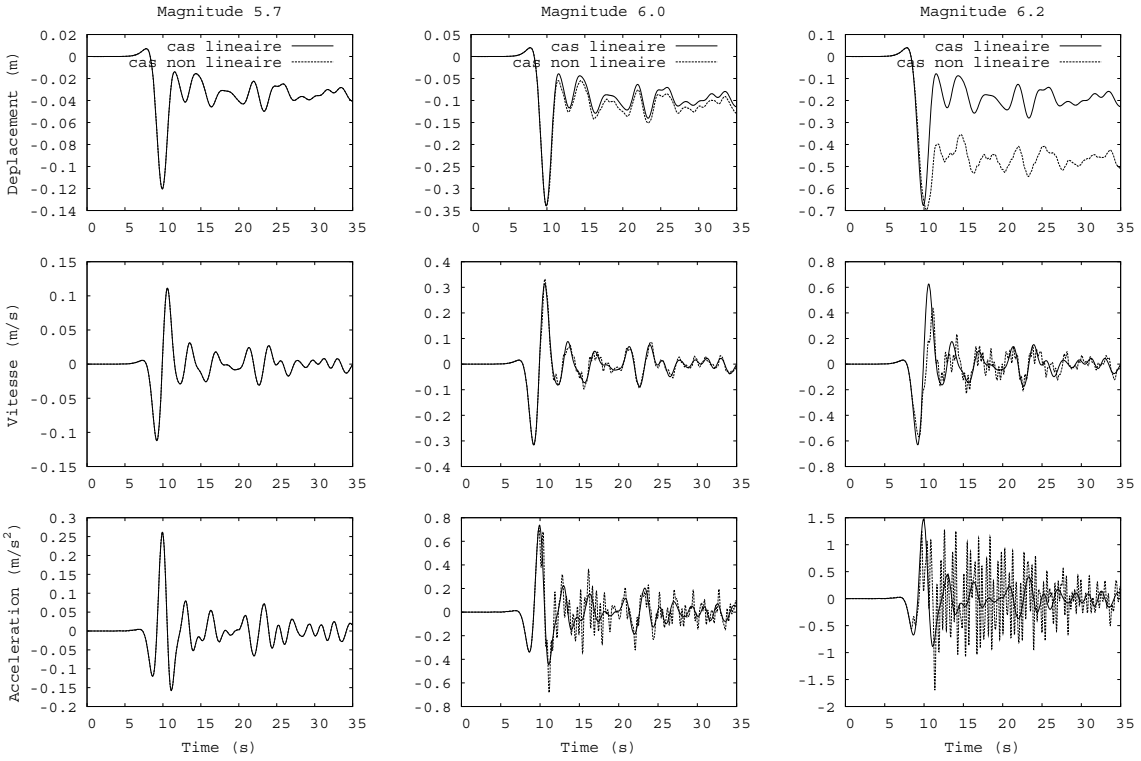

FIGURE 2.9 – Composantes est-ouest du déplacement (haut), de la vitesse (milieu) et de l'accélération (bas) pour un récepteur proche du centre de la ville de Nice. Des séismes de magnitude 5.7 (gauche), 6.0 (milieu) et 6.2 (droite) sont modélisés.

Pour le séisme de magnitude 5.7, les composantes du déplacement, de la vitesse et de l'accélération sont similaires dans les cas linéaires et non linéaires. Le mouvement n'est pas assez fort dans ce cas pour observer l'apparition d'un comportement plastique.

L'augmentation de la magnitude des séismes simulés (magnitudes 6.0 et 6.2) permet d'observer des écarts importants entre la modélisation élastique et l'introduction d'une loi de comportement complexe pour les couches de sédiments. Par exemple, la dissipation d'énergie liée à la non linéarité conduit à une réduction de l'amplitude pour la composante de vitesse, particulièrement dans le cas du séisme de magnitude 6.2 entre 8 s et 12 s. L'effet sur les composantes d'accélération est plus difficile à analyser, on observe principalement l'apparition de hautes fréquences [20] et une augmentation significative de l'amplitude, notamment après la portion correspondant à l'onde S (après 12 s). Les hautes fréquences introduites par le comportement non linéaire du sol nécessiteraient sans doute de raffiner le maillage initial.

Les figures 2.10, 2.11, 2.12 et 2.13 présentent respectivement les résultats de Peak Groud Displacement (PGD) et de Peak Ground Velocities (PGV) dans les cas linéaire et non linéaire pour le maillage à 0.6 Hz. On peut tout d'abord souligner l'adéquation des simulations avec les directions théoriques de radiation des ondes S décrites par le schéma 2.3. Globalement, on observe que le comportement non linéaire des couches sédimentaires se traduit par une augmentation

du PGD, en particulier dans les zones présentant initialement des valeurs élevées dans le cas linéaire. Ces résultats sont attendus et correspondent aux lois de comportement utilisées pour cet exemple (e.g [94]).

De manière classique, on observe une augmentation du PGV dans le cas linéaire au sommet des collines. Il s'agit d'un effet de site due à la topographie (e.g. [28]). L'effet de la non linéarité (figure 2.13) est une plus grande dissipation de l'énergie conduisant à une réduction locale du PGV d'environ 28%.

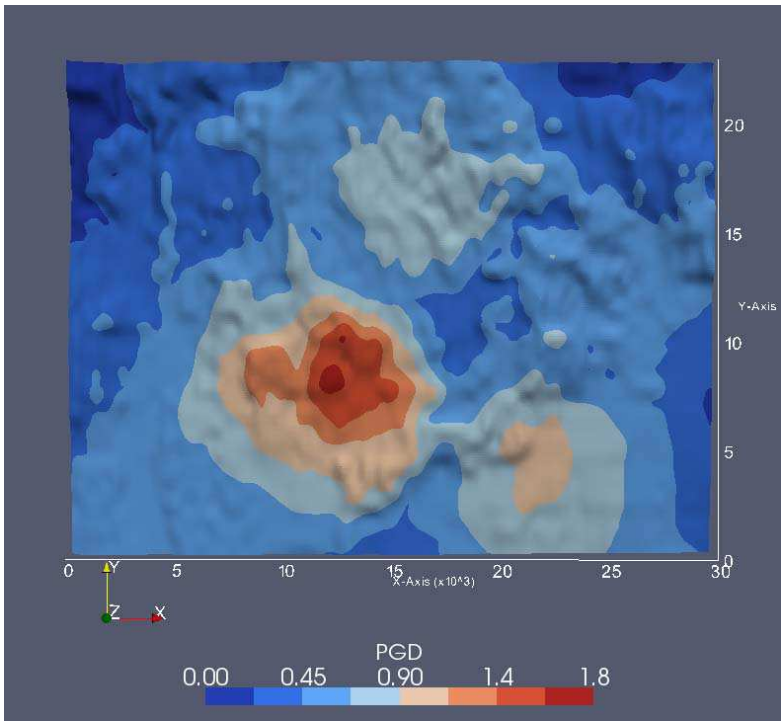

FIGURE 2.10 – Peak Ground Displacement (PGD) dans le cas linéaire pour modélisation à 0.6 Hz.

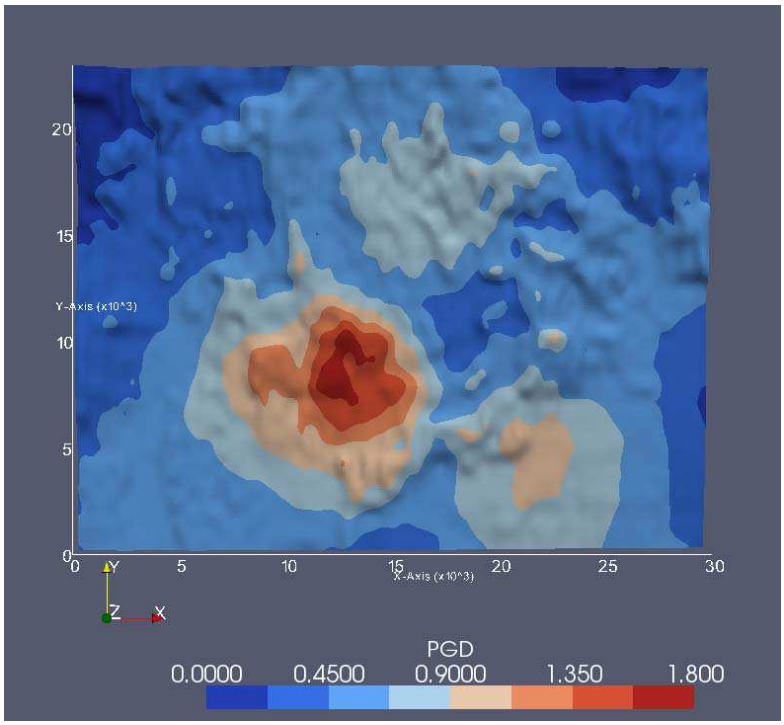

FIGURE 2.11 – Peak Ground Displacement (PGD) dans le cas non linéaire pour une modélisation à 0.6 Hz.

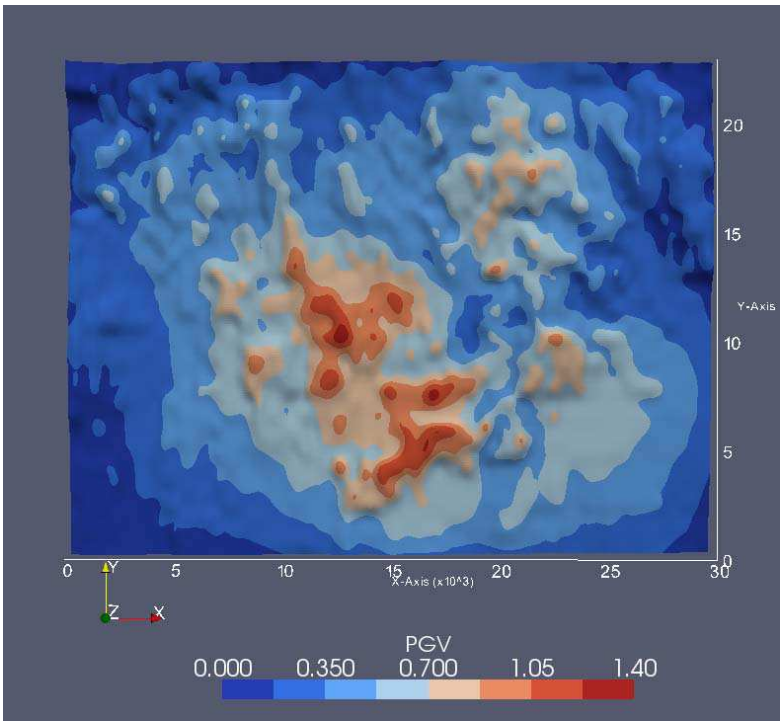

FIGURE 2.12 – Peak Ground Velocity (PGV) dans le cas linéaire pour une modélisation à 0.6 Hz.

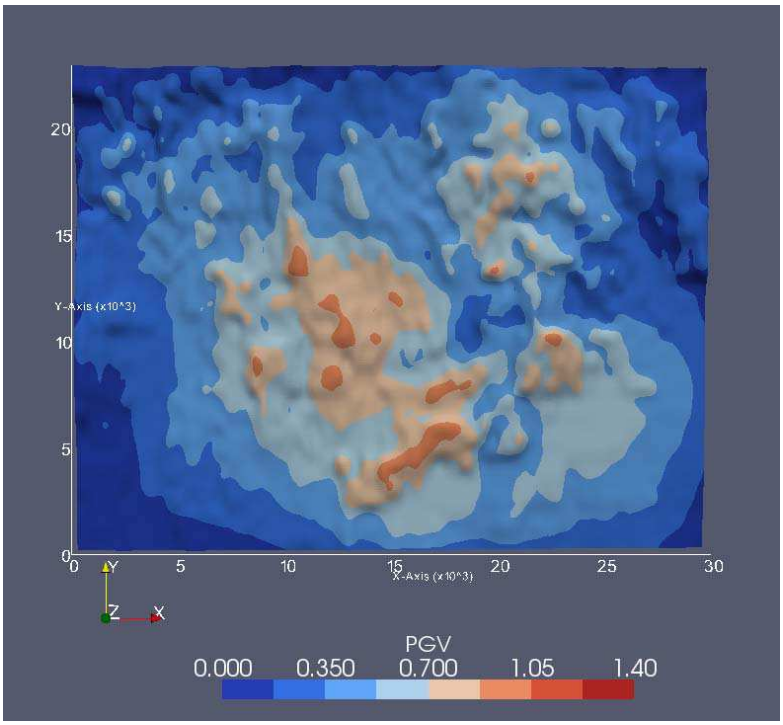

FIGURE 2.13 – Peak Ground Velocity (PGV) dans le cas non linéaire pour une modélisation à 0.6 Hz.

Conclusion et perspectives

Conclusion et perspectives

Contents

8.1 Contributions

Les propositions détaillées dans cette thèse abordent différentes difficultés en vue de la mise en œuvre d'applications sismiques sur des architectures multicœurs et hiérarchiques. La première partie de ce manuscrit est consacrée à la méthode des différences finies.
La définition de conditions aux limites absorbantes performantes et leur implémentation sur architectures multicœurs constituent un enjeu important de la modélisation sismique. Le chapitre 2 aborde ce problème en soulignant l'intérêt des approches quasi-statiques, impléméntées au niveau MPI, afin de corriger le déséquilibre de charge dû aux bords absorbants. Malgré les améliorations très sensibles apportées par cet algorithme, il doit néanmoins être limité à un faible nombre de sous-domaines pour être performant. L'introduction d'un deuxième niveau de parallélisme est donc nécessaire, afin de coller au mieux à l'architecture hybride des supercalculateurs. Une approche basée sur la virtualisation est donc utilisée à l'intérieur des nœuds. Cette technique s'appuie sur la bibliothèque de threads Marcel et l'exploitation d'un simple algorithme d'ordonnancement glouton. Ce mécanisme, habituellement faiblement exploité pour les applications scientifiques, permet notamment d'éviter l'implémentation d'un algorithme spécifique dédié à la régulation dynamique de la charge.

Une évaluation détaillée de l'impact de l'affinité mémoire sur les performances applicatives est menée dans le chapitre 3. Celle-ci souligne le nombreux paramètres (architecture, schéma d'accès aux données, consommation mémoire) pouvant significativement modifier l'effet de la pénalité NUMA sur les performances. Le mécanisme de virtualisation implémenté s'accompagne d'une nécessaire flexibilité en termes de placement des données. La plateforme Bubblesched et l'ordonnancement hiérarchique des threads apportent certaines réponses. Mais la pertinence du mécanisme de migration des données et des stratégies d'ordonnancement des threads implémentées sont encore à valider dans un contexte de surcharge. Les résultats obtenus avec l'environnement MAi sont concluants, avec une intrusion minimale au niveau applicatif. Dans ce cas, le placement des données est découplé de l'ordonnancement des threads. La politique mémoire

correspondant au placement cyclique des pages mémoire permet néanmoins de tenir compte du schéma irrégulier d'accès aux données, optimisant ainsi l'utilisation de la bande passante disponible. Des gains de l'ordre de 34% ont été mesurés sur une architecture composée de 8 nœuds NUMA.

Le caractère *memory bound* des applications basées sur la méthode des différences finies est étudié dans le chapitre 4. L'algorithme proposé afin d'améliorer la localité mémoire provient en partie de références dans la littérature informatique et géophysique. Il permet d'évaluer plusieurs pas de temps de manière simultanée : le domaine de calcul n'est plus simplement partitionné selon les directions spatiales mais également dans le domaine temporel. L'implémentation sur architecture multicœurs exploite les contributions des chapitres 2 et 3, avec la combinaison d'un mécanisme de surcharge au niveau multithreading et l'optimisation du placement des données basé sur la librairie MAi. La comparaison avec une implémentation OpenMP standard démontre des gains variant entre 1.4 et 3.8 suivant les architectures considérées.

La modélisation d'un séisme de 2007 dans la région de Niigata-Chuetsu au Japon, est présentée dans le chapitre 5 en utilisant les contributions des chapitres précédents. Les techniques à deux niveaux introduites dans le chapitre 2 sont notamment étendues, le déséquilibre de charge résultant principalement de la différence de temps de calcul entre les nœuds multicœurs. Ce déséquilibre peut ainsi être compensé par l'algorithme quasi-statique appliqué au niveau MPI. En effet, le mécanisme de régulation dynamique issu du multithreading massif utilisé conduit à un écart très faible entre les cœurs d'un même nœud. L'analyse des résultats physiques démontre un bon accord entre les sismogrammes synthétiques et les observations pour les répliques étudiées. Les résultats des simulations effectuées autour de la région de Kashiwazaki soulignent néanmoins l'importance d'une meilleure compréhension des mécanismes de la source sismique et de la prise en compte du caractère non linéaire de la réponse sismique.

La deuxième partie de ce manuscrit est consacrée à la méthode des éléments finis. Certains points spécifiques aux modélisations géomécaniques sont abordés. Le chapitre 6 souligne l'importance de la phase de partitionnement des éléments du maillage. Afin de tenir compte de la physique du problème et du comportement non linéaire de certaines zones de calcul, une décomposition à deux niveaux est introduite. Une approche classique est privilégiée dans les zones correspondant aux couches profondes principalement composées de rochers avec un coût de calcul homogène. Les couches sédimentaires, plus enclines à un comportement mécanique complexe et hétérogène, sont partitionnées en exploitant une technique de coloriage de graphe permettant de maximiser l'écart entre les éléments géométriquement proches. Cette stratégie se situe à rebours des techniques implémentés dans le cas standard, mais elle conduit à des résultats très satisfaisants avec des gains en temps d'un facteur 3.5 par rapport à l'approche standard.

La procédure générale de calcul repose sur l'utilisation intensive de noyaux de calcul matriciels notamment dus à l'algorithme de Newton-Raphson implémenté. Cette spécificité a été avantageusement utilisée avec l'exploitation de bibliothèques d'algèbre linéaire fortement optimisées pour les architectures multicœurs. C'est le scénario mis en œuvre dans le code éléments finis Gefdyn, dont la conception initiale rendait difficile l'adaptation au parallélisme à grain fin. L'intégration du solveur PASTIX a ainsi permis d'envisager des modélisations à grande échelle dans le domaine de la modélisation sismique non linéaire. Un séisme hypothétique est simulé dans le bassin sédimentaire de la région niçoise et les résultats obtenus sur 1024 cœurs démontrent à la fois un bon comportement parallèle mais également l'importance de la prise en compte du

caractère non linéaire de la réponse du sous-sol du point de vue géophysique.

8.2 Perspectives

8.2.1 Meilleure exploitation des supports d'exécution

Les contributions présentées dans cette thèse sont encore à étendre, un des points clés étant de mieux faire remonter les informations fournies par les applications au niveau des supports d'exécution. Tout d'abord, les possibilités d'ordonnancement hiérarchique des threads sont à adapter de manière plus fine au contexte applicatif dans le cas de la méthode des différences finies. En effet, les expériences menées ont montré des gains possibles en plaçant de manière coordonnée les threads de calcul et les données sur les différents cœurs et blocs mémoire de l'architecture. En revanche, la prise en compte du mécanisme de surcharge est encore incomplète et nécessiterait probablement la définition d'un ordonnanceur spécifique. La plateforme Bubblesched, permettant au programmeur (averti) d'écrire son propre ordonnanceur, est un bon candidat pour la mise en œuvre d'une telle solution.

Ensuite, l'analyse du mécanisme de surcharge implémenté doit être approfondi. La file unique d'ordonnancement utilisée par l'algorithme glouton pourrait rapidement constituer un goulot d'étranglement limitant les gains issus de l'exploitation d'un parallélisme à grain fin. Dans un premier temps, il s'agirait de pouvoir quantifier cette contention et ensuite de définir certaines bornes en terme de nombres de threads à utiliser.

De plus, les politiques de placement des données offertes par l'environnement MAi sont également à rapprocher du contexte applicatif. Par exemple, le programmeur pourrait choisir de placer des blocs de pages mémoire correspondant à la forme des sous-domaines, en lieu et place d'une distribution cyclique basée sur des pages ou des blocs de pages mémoire sans lien avec l'algorithmique de l'application. Cette évolution conduirait probablement à l'écriture de nouvelles politiques mémoire. Le dimensionnement précis de la taille des blocs ainsi que la meilleure option pour leur répartition constituent également des points d'amélioration.

Enfin, la décomposition espace-temps introduite dans le chapitre 5 et sa mise en œuvre sur architecture à mémoire distribuée sont également à étendre. En effet, la définition d'une stratégie efficace d'échanges d'informations à l'interface entre sous-domaines distants est relativement complexe. Il s'agit d'ordonnancer de manière optimale les phases de communication et les phases de calcul. Les pistes proposées dans [40] pourraient constituer un point de départ. La définition de la forme optimale du sous-domaine espace-temps constitue un autre point crucial pour l'optimisation de cet algorithme. En effet, la solution que nous avons retenue permet simplement de respecter la forme du stencil ce qui conduit à une forme géométrique assez simple. Les propositions récentes introduites dans [127] permettent de déduire la géométrie optimale en terme de localité mémoire.

L'ensemble de ces contributions sont à étudier de manière globale, notamment pour la sélection des paramètres de chaque algorithme et l'exploitation optimale des supports d'exécution. Par exemple, l'ordonnancement hiérarchique des threads pourrait significativement contribuer à l'amélioration de la localité mémoire en plaçant les threads correspondant aux sous-domaines adjacents sur des cœurs partageant le même niveau de mémoire cache. De plus, la géométrie des sous-domaines traduisant la politique mémoire de l'environnement MAi devrait être reliée à

la géométrie des sous-domaines de la décomposition espace-temps. Une stratégie d'*auto-tuning* permettrait ainsi d'optimiser la politique mémoire, la taille et la forme des sous-domaines ou encore le nombre de threads de surcharge en fonction de l'architecture. La taille de l'espace de recherche nécessiterait alors le recours à des heuristiques spécifiques [47].

8.2.2 Hétérogénéité des architectures multicœurs

L'émergence des architectures hybrides intégrant processeurs multicœurs et accélérateurs graphiques constitue désormais un défi supplémentaire pour exploiter des nœuds de calcul multicœurs hétérogènes. Plusieurs références discutent de l'implémentation de méthodes numériques classiques de la géophysique sur GPU [2, 87, 116]. Les accélérations obtenues sont significatives, même si les implémentations standard choisies comme référence ne sont souvent que partiellement optimisées sur CPU [96].

Le véritable défi réside dans la capacité à exploiter les différents niveaux de parallélisme offerts par les architectures hybrides. Quelle que soit l'évolution de ces architectures et l'intégration plus ou moins forte qui existera entre les cœurs spécialisés et les cœurs généralistes, les supports d'exécution joueront un rôle clé pour l'adaptation des applications. Les contributions développées dans cette thèse, basées sur l'exploitation d'un multithreading massif pour la méthode des différences finies ou l'utilisation d'une bibliothèque d'algèbre linéaire fortement optimisée dans le cas de la méthode des éléments finis, constituent une base de travail à prolonger. Une des différences majeures se situera dans l'expression du parallélisme avec une décomposition en tâches de l'application permettant de déléguer la complexité de l'ordonnancement hybride au support d'exécution [13, 15]. La maîtrise du flot d'exécution de l'application sera alors prépondérante.

Dans le cas de la méthode des éléments finis, l'évolution des bibliothèques d'algèbre linéaire autour des architectures hybrides est relativement avancée avec des projets tels que PLASMA ou MAGMA [3]. Le programmeur pourrait alors adopter une démarche peu intrusive en se reposant de manière importante sur ces contributions externes, et en ne modifiant son application que de manière marginale.

8.2.3 Approches numériques

Les approches numériques standard considérées durant cette thèse invitent à explorer de nouvelles pistes permettant d'améliorer la qualité des schémas. Les verrous classiques de la méthode des différences finies pour la description plus précise de la topographie ou encore l'implémentation de conditions aux limites absorbantes plus performantes (en qualité numérique et en coût de calcul) sont des sujets très actifs avec des propositions qui peuvent significativement améliorer la qualité des simulations [100].

Dans le cas de la méthode des éléments finis, une meilleure articulation entre le niveau local et le niveau global de résolution serait probablement bénéfique. En effet, le niveau local d'intégration des lois de comportement basé sur la définition de sous-incréments est à relier aux critères de convergence globaux provenant de la résolution du problème de Newton. Il est probablement inutile de "sur-résoudre" le problème local par rapport au problème global [1]. Dans la même vision des choses, l'introduction d'un solveur itératif, en lieu et place du solveur direct utilisé, permettrait de lier la convergence de la méthode de Newton à la convergence de la méthode de Krylov implémentée (méthode de Newton-Krylov). Cela permettrait en outre de s'affranchir de la

construction explicite de la matrice globale, la procédure s'appuyant sur une approche "Jacobian-free" décrite dans [86]. Une autre piste pourrait être de profiter du faible nombre de reformations de la matrice de rigidité globale afin d'améliorer la méthode itérative dans un contexte multi second-membre [78, 151].

Les techniques numériques de décomposition de domaine proposent également des pistes intéressantes. Elles conduisent à une décomposition à deux niveaux du problème globale en définissant dans un premier temps plusieurs problèmes locaux eux-mêmes résolus de manière concurrente sur plusieurs processeurs. Les problématiques d'équilibrage de charge liées aux rhéologies complexes des matériaux sont donc plus difficiles à maitriser dans ce contexte. Une autre solution serait de conserver la formulation globale actuellement utilisée et de se reposer sur les méthodes hybrides proposées au sein de solveurs matriciels [16, 76]. Le choix des techniques numériques (solveur direct ou itératif en interne ou à l'interface) les mieux adaptées au problème demeure difficile.

8.2.4 Modélisation sismique

Les modélisations sismiques détaillées dans cette thèse abordent principalement le problème direct dans le contexte de l'évaluation des risques sismiques. L'amélioration des techniques algorithmiques et numériques permettraient tout d'abord d'atteindre des fréquences supérieures à 1 Hz, qui sont nécessaires dans le domaine de la géotechnique et de l'ingénierie. Ceci requiert évidemment une meilleure connaissance des modèles géologiques et un raffinement important des maillages.
De plus, un des défis majeurs de la sismologie est la compréhension de la structure interne du sous-sol au travers de méthodes d'inversion sismique. En considérant la nécessaire compréhension des mécanismes de la source sismique, les approches cinématiques traditionnellement utilisées ou encore l'implémentation de techniques d'inversion dynamique permettant de prendre en compte l'hétérogénéité du milieu reposent généralement sur des procédures itératives [35]. De même les techniques de tomographie sismique, par exemple basée sur la méthode du gradient, nécessitent également la résolution d'un grand nombre de problèmes directes [32]. Dans les deux cas, plusieurs centaines voire plusieurs milliers d'itérations sont nécessaires pour des problèmes 2D ou 3D. Les performances parallèles des applications sont donc déterminantes et permettraient d'envisager de nouvelles perspectives en termes de modélisation sismique.

Annexe A

Discrétisation du problème élastodynamique par la méthode des différences finies

Nous utilisons les notations de la section 1.2.1.0 avec $\Delta s = \Delta x = \Delta y = \Delta z$ pour la discrétisation des équations de l'élastodynamique (système d'équations 1.10 du chapitre 1) par la méthode des différences finies à l'ordre 4 en espace et à l'ordre 2 en temps.
Au temps $t = (\ell + 1/2)\Delta t$, les composantes de vitesse s'écrivent :

$$
\begin{aligned}
v_x^{(i+\frac{1}{2})jk}\Big(l+\frac{1}{2}\Big) &= v_x^{(i+\frac{1}{2})jk}\Big(l-\frac{1}{2}\Big) + \bar{b_x}\Delta t F_x^{(i+\frac{1}{2})jk} \\
&+ \bar{b_x}\Delta t\frac{9}{8}\Big[\frac{\sigma_{xx}^{(i+1)jk}-\sigma_{xx}^{ijk}}{\Delta x} + \frac{\sigma_{xy}^{(i+\frac{1}{2})(j+\frac{1}{2})k}-\sigma_{xy}^{(i+\frac{1}{2})(j-\frac{1}{2})k}}{\Delta y} + \frac{\sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{1}{2})}-\sigma_{xz}^{(i+\frac{1}{2})j(k-\frac{1}{2})}}{\Delta z}\Big] \\
&- \bar{b_x}\Delta t\frac{1}{24}\Big[\frac{\sigma_{xx}^{(i+2)jk}-\sigma_{xx}^{(i-1)jk}}{\Delta x} + \frac{\sigma_{xy}^{(i+\frac{1}{2})(j+\frac{3}{2})k}-\sigma_{xy}^{(i+\frac{1}{2})(j-\frac{3}{2})k}}{\Delta y} + \frac{\sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{3}{2})}-\sigma_{xz}^{(i+\frac{1}{2})j(k-\frac{3}{2})}}{\Delta z}\Big]
\end{aligned}
\tag{A.1}
$$

$$
\begin{aligned}
v_y^{i(j+\frac{1}{2})k}\Big(l+\frac{1}{2}\Big) &= v_y^{i(j+\frac{1}{2})k}\Big(l-\frac{1}{2}\Big) + \bar{b_y}\Delta t F_y^{i(j+\frac{1}{2})k} \\
&+ \bar{b_y}\Delta t\frac{9}{8}\Big[\frac{\sigma_{yx}^{(i+\frac{1}{2})(j+\frac{1}{2})k}-\sigma_{yx}^{(i-\frac{1}{2})(j+\frac{1}{2})k}}{\Delta x} + \frac{\sigma_{yy}^{i(j+1)k}-\sigma_{yy}^{ijk}}{\Delta y} + \frac{\sigma_{yz}^{i(j+\frac{1}{2})(k+\frac{1}{2})}-\sigma_{yz}^{i(j+\frac{1}{2})(k-\frac{1}{2})}}{\Delta z}\Big] \\
&- \bar{b_y}\Delta t\frac{1}{24}\Big[\frac{\sigma_{yx}^{(i+\frac{3}{2})(j+\frac{1}{2})k}-\sigma_{yx}^{(i-\frac{3}{2})(j+\frac{1}{2})k}}{\Delta x} + \frac{\sigma_{yy}^{i(j+2)k}-\sigma_{yy}^{i(j-1)k}}{\Delta y} + \frac{\sigma_{yz}^{i(j+\frac{1}{2})(k+\frac{3}{2})}-\sigma_{yz}^{i(j+\frac{1}{2})(k-\frac{3}{2})}}{\Delta z}\Big]
\end{aligned}
\tag{A.2}
$$

$$
\begin{aligned}
v_z^{ij(k+\frac{1}{2})}\Big(l+\frac{1}{2}\Big) &= v_z^{ij(k+\frac{1}{2})}\Big(l-\frac{1}{2}\Big) + \bar{b_z}\Delta t F_z^{ij(k+\frac{1}{2})} \\
&+ \bar{b_z}\Delta t\frac{9}{8}\Big[\frac{\sigma_{zx}^{(i+\frac{1}{2})j(k+\frac{1}{2})}-\sigma_{zx}^{(i-\frac{1}{2})j(k+\frac{1}{2})}}{\Delta x} + \frac{\sigma_{zy}^{i(j+\frac{1}{2})(k+\frac{1}{2})}-\sigma_{zy}^{i(j-\frac{1}{2})(k+\frac{1}{2})}}{\Delta y} + \frac{\sigma_{zz}^{ij(k+1)}-\sigma_{zz}^{ijk}}{\Delta z}\Big] \\
&- \bar{b_z}\Delta t\frac{1}{24}\Big[\frac{\sigma_{zx}^{(i+\frac{3}{2})j(k+\frac{1}{2})}-\sigma_{zx}^{(i-\frac{3}{2})j(k+\frac{1}{2})}}{\Delta x} + \frac{\sigma_{zy}^{i(j+\frac{3}{2})(k+\frac{1}{2})}-\sigma_{zy}^{i(j-\frac{3}{2})(k+\frac{1}{2})}}{\Delta y} + \frac{\sigma_{zz}^{ij(k+2)}-\sigma_{zz}^{ij(k-1)}}{\Delta z}\Big]
\end{aligned}
\tag{A.3}
$$

Au pas de temps $t = \ell\Delta t$, les composantes de contraintes s'écrivent :

$$\begin{aligned}
&\sigma_{xx}^{ijk}(l+1) = \sigma_{xx}^{ijk}(l) \\
&+ \Delta t\frac{9}{8}\Big[(\lambda^{ijk}+2\mu^{ijk})\frac{v_x^{(i+\frac{1}{2})jk}-v_x^{(i-\frac{1}{2})jk}}{\Delta x}+\lambda^{ijk}\frac{v_y^{i(j+\frac{1}{2})k}-v_y^{i(j-\frac{1}{2})k}}{\Delta y}+\lambda^{ijk}\frac{v_z^{ij(k+\frac{1}{2})}-v_z^{ij(k-\frac{1}{2})}}{\Delta z}\Big] \\
&- \Delta t\frac{1}{24}\Big[(\lambda^{ijk}+2\mu^{ijk})\frac{v_x^{(i+\frac{3}{2})jk}-v_x^{(i-\frac{3}{2})jk}}{\Delta x}+\lambda^{ijk}\frac{v_y^{i(j+\frac{3}{2})k}-v_y^{i(j-\frac{3}{2})k}}{\Delta y}+\lambda^{ijk}\frac{v_z^{ij(k+\frac{3}{2})}-v_z^{ij(k-\frac{3}{2})}}{\Delta z}\Big]
\end{aligned} \tag{A.4}$$

$$\begin{aligned}
&\sigma_{yy}^{ijk}(l+1) = \sigma_{yy}^{ijk}(l) \\
&+ \Delta t\frac{9}{8}\Big[\lambda^{ijk}\frac{v_x^{(i+\frac{1}{2})jk}-v_x^{(i-\frac{1}{2})jk}}{\Delta x}+(\lambda^{ijk}+2\mu^{ijk})\frac{v_y^{i(j+\frac{1}{2})k}-v_y^{i(j-\frac{1}{2})k}}{\Delta y}+\lambda^{ijk}\frac{v_z^{ij(k+\frac{1}{2})}-v_z^{ij(k-\frac{1}{2})}}{\Delta z}\Big] \\
&- \Delta t\frac{1}{24}\Big[\lambda^{ijk}\frac{v_x^{(i+\frac{3}{2})jk}-v_x^{(i-\frac{3}{2})jk}}{\Delta x}+(\lambda^{ijk}+2\mu^{ijk})\frac{v_y^{i(j+\frac{3}{2})k}-v_y^{i(j-\frac{3}{2})k}}{\Delta y}+\lambda^{ijk}\frac{v_z^{ij(k+\frac{3}{2})}-v_z^{ij(k-\frac{3}{2})}}{\Delta z}\Big]
\end{aligned} \tag{A.5}$$

$$\begin{aligned}
&\sigma_{zz}^{ijk}(l+1) = \sigma_{zz}^{ijk}(l) \\
&+ \Delta t\frac{9}{8}\Big[\lambda^{ijk}\frac{v_x^{(i+\frac{1}{2})jk}-v_x^{(i-\frac{1}{2})jk}}{\Delta x}+\lambda^{ijk}\frac{v_y^{i(j+\frac{1}{2})k}-v_y^{i(j-\frac{1}{2})k}}{\Delta y}+(\lambda^{ijk}+2\mu^{ijk})\frac{v_z^{ij(k+\frac{1}{2})}-v_z^{ij(k-\frac{1}{2})}}{\Delta z}\Big] \\
&- \Delta\frac{1}{24}\Big[\lambda^{ijk}\frac{v_x^{(i+\frac{3}{2})jk}-v_x^{(i-\frac{3}{2})jk}}{\Delta x}+\lambda^{ijk}\frac{v_y^{i(j+\frac{3}{2})k}-v_y^{i(j-\frac{3}{2})k}}{\Delta y}+(\lambda^{ijk}+2\mu^{ijk})\frac{v_z^{ij(k+\frac{3}{2})}-v_z^{ij(k-\frac{3}{2})}}{\Delta z}\Big]
\end{aligned} \tag{A.6}$$

$$\begin{aligned}
&\sigma_{xy}^{(i+\frac{1}{2})(j+\frac{1}{2})k}(l+1) = \sigma_{xy}^{(i+\frac{1}{2})(j+\frac{1}{2})k}(l) \\
&+ \Delta t\bar{\mu}_{xy}\frac{9}{8}\Big[\frac{v_x^{(i+\frac{1}{2})(j+1)k}-v_x^{(i+\frac{1}{2})jk}}{\Delta y}+\frac{v_y^{(i+1)(j+\frac{1}{2})k}-v_y^{i(j+\frac{1}{2})k}}{\Delta x}\Big] \\
&- \Delta t\bar{\mu}_{xy}\frac{1}{24}\Big[\frac{v_x^{(i+\frac{1}{2})(j+2)k}-v_x^{(i+\frac{1}{2})(j-1)k}}{\Delta y}+\frac{v_y^{(i+2)(j+\frac{1}{2})k}-v_y^{(i-1)(j+\frac{1}{2})k}}{\Delta x}\Big]
\end{aligned} \tag{A.7}$$

$$\begin{aligned}
&\sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{1}{2})}(l+1) = \sigma_{xz}^{(i+\frac{1}{2})j(k+\frac{1}{2})}(l) \\
&+ \Delta t\bar{\mu}_{xz}\frac{9}{8}\Big[\frac{v_x^{(i+\frac{1}{2})j(k+1)}-v_x^{(i+\frac{1}{2})jk}}{\Delta z}+\frac{v_z^{(i+1)j(k+\frac{1}{2})}-v_z^{ij(k+\frac{1}{2})}}{\Delta x}\Big] \\
&- \frac{1}{24}\Big[\frac{v_x^{(i+\frac{1}{2})j(k+2)}-v_x^{(i+\frac{1}{2})j(k-1)}}{\Delta z}+\frac{v_z^{(i+2)j(k+\frac{1}{2})}-v_z^{(i-1)j(k+\frac{1}{2})}}{\Delta x}\Big]
\end{aligned} \tag{A.8}$$

$$\begin{aligned}
&\sigma_{yz}^{i(j+\frac{1}{2})(k+\frac{1}{2})}(l+1) = \sigma_{yz}^{i(j+\frac{1}{2})(k+\frac{1}{2})}(l) \\
&+ \Delta t\bar{\mu}_{yz}\frac{9}{8}\Big[\frac{v_y^{i(j+\frac{1}{2})(k+1)}-v_y^{i(j+\frac{1}{2})k}}{\Delta z}+\frac{v_z^{i(j+1)(k+\frac{1}{2})}-v_z^{ij(k+\frac{1}{2})}}{\Delta y}\Big] \\
&- \Delta t\bar{\mu}_{yz}\frac{1}{24}\Big[\frac{v_y^{i(j+\frac{1}{2})(k+2)}-v_y^{i(j+\frac{1}{2})(k-1)}}{\Delta z}+\frac{v_z^{i(j+2)(k+\frac{1}{2})}-v_z^{i(j-1)(k+\frac{1}{2})}}{\Delta y}\Big].
\end{aligned} \tag{A.9}$$

(A.10)

$\bar{b}$ est donné par :

$$\bar{b_x} = \frac{1}{2}\Big[\frac{1}{\rho^{ijk}} + \frac{1}{\rho^{(i+1)jk}}\Big] \tag{A.11}$$

$$\bar{b_y} = \frac{1}{2}\Big[\frac{1}{\rho^{ijk}} + \frac{1}{\rho^{i(j+1)k}}\Big] \tag{A.12}$$

$$\bar{b_z} = \frac{1}{2}\Big[\frac{1}{\rho^{ijk}} + \frac{1}{\rho^{ij(k+1)}}\Big]. \tag{A.13}$$

$\bar{\mu}$ est donné par :

$$\bar{\mu}_{xy} = \Big[\frac{1}{4}\Big(\frac{1}{\mu^{ijk}} + \frac{1}{\mu^{(i+1)jk}} + \frac{1}{\mu^{i(j+1)k}} + \frac{1}{\mu^{(i+1)(j+1)k}}\Big)\Big]^{-1} \tag{A.14}$$

$$\bar{\mu}_{xz} = \Big[\frac{1}{4}\Big(\frac{1}{\mu^{ijk}} + \frac{1}{\mu^{(i+1)jk}} + \frac{1}{\mu^{ij(k+1)}} + \frac{1}{\mu^{(i+1)j(k+1)}}\Big)\Big]^{-1} \tag{A.15}$$

$$\bar{\mu}_{yz} = \Big[\frac{1}{4}\Big(\frac{1}{\mu^{ijk}} + \frac{1}{\mu^{i(j+1)k}} + \frac{1}{\mu^{ij(k+1)}} + \frac{1}{\mu^{i(j+1)(k+1)}}\Big)\Big]^{-1}. \tag{A.16}$$

Annexe B

Description des architectures utilisées

Les différents schémas donnés dans cette section indiquent la numérotation des processeurs en orange et l'organisation des différents blocs mémoire dans le cas des machines NUMA.

B.1 Architectures NUMA

B.1.1 Plateforme `Teranova`

Chaque nœud de la plateforme `Teranova` est une machine Bull Novascale 5160 équipée de 16 processeurs Itanium2 Madison. Ces processeurs sont cadencés à 1.6 Ghz avec une mémoire cache L3 de niveau de 9 Mo. La machine est organisée en 4 nœuds (QBB : Quad Brick Block) de 4 processeurs chacun et la mémoire est divisée en 4 blocs de 16 Go. Ces nœuds sont interconnectés par le système FAME Scalability Switch (FSS) développé par BULL (cf. schéma B.1).

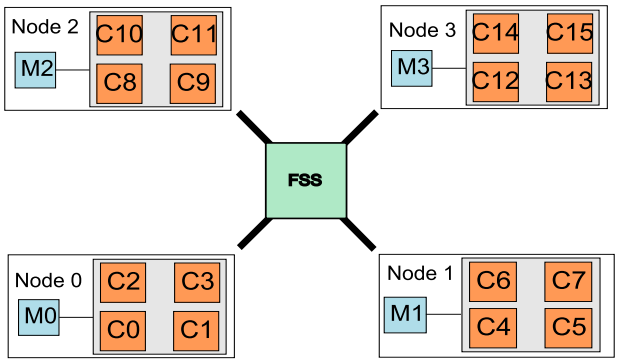

FIGURE B.1 – Plateforme `Teranova`.

B.1.2 Plateformes `Lias` et `Malm`

Les plateformes `Lias` et `Malm` constituent deux sous clusters du système de calcul Dogger localisé au BRGM à Orléans. L'architecture `Lias` est composée de seize nœuds de calcul biprocesseurs de type AMD Opteron reliés par un réseau Gigabit et disposant chacun de 8 Go de mémoire. Chaque processeur est cadencé à 2.6 Ghz et dispose de 2 Mo (2×1 Mo) de mémoire cache (Opteron-285).
Le cluster `Malm` contient des processeurs AMD Opteron quad-cœurs cadencés à 2.3 Ghz et disposant de 2 Mo (4×512 Ko) de mémoire cache (Opteron-2356). Le système est composé de 32 nœuds

biprocesseurs reliés par un réseau Myrinet, chacun de ces nœuds ayant 16 Go de mémoire. Les figures B.2 et B.3 décrivent respectivement l'architecture générale d'un nœud de calcul de `Lias` et `Malm`; l'interconnexion entre les deux nœuds NUMA s'appuie sur la technologie HyperTransport.

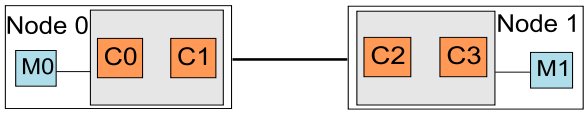

FIGURE B.2 – Plateforme `Lias`.

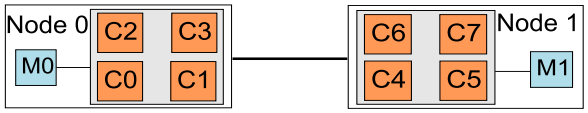

FIGURE B.3 – Plateforme `Malm`.

B.1.3 Plateforme `Hades`

La plateforme `Hades` est localisée à Bordeaux, il s'agit d'un serveur de développement de l'équipe INRIA Hiepacs. Ce serveur comprend deux processeurs Intel Nehalem quad-cœurs disposant chacun de 8 Mo de mémoire cache et cadencé à 2.9 Ghz (Xeon-5570). Ce serveur dispose de 48 Go de mémoire. L'interconnexion entre les deux nœuuds NUMA repose sur le Quick Path Interconnect d'Intel.

B.1.4 Plateforme `Borderline`

L'architecture `Borderline` est localisée à Bordeaux (INRIA Bordeaux Sud-Ouest) et fait partie de la grille d'expérimentation *Grid'5000*. Il s'agit d'un cluster de dix nœuds de calcul reliés par des réseaux rapides de type Myrinet et Infiniband. Chaque nœud est composé de quatre processeurs dual-cœur Opteron (Opteron-2218) constituant une architecture NUMA décrite sur la figure B.4. Les processeurs sont cadencés à 2.6 Ghz et disposent de 2 Mo (2×1 Mo) de mémoire cache.

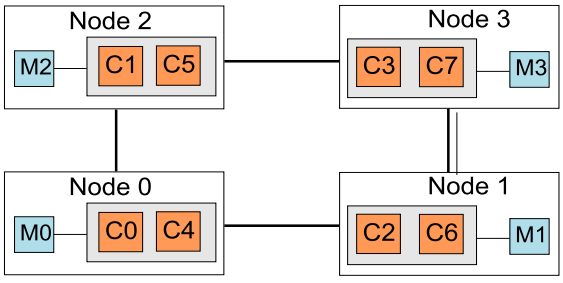

FIGURE B.4 – Plateforme `Borderline`.

B.1.5 Plateforme `Idkoiff`

Le serveur d'experimentation `Idkoiff` est composé de huit processeurs dual-cœur AMD Opteron (Opteron-875) formant huit nœuds NUMA dont la structure et les interconnexions sont

décrites sur la figure B.5. Chaque processeur est relié à un bloc mémoire de 4 Go, la machine dispose ainsi d'un total de 32 Go. Cette machine est composée d'une première carte mère regroupant les processeurs *cpu0*, *cpu1*, *cpu3* et *cpu4* connectés par un lien HyperTransport bidirectionnel offrant jusqu'a 6.4 Go/s et assurant également la cohérence des caches. Notons que les *cpu0* et *cpu1* prennent en charge les entrées/sorties. Une deuxième carte mère est reliée à la première par un connecteur HypertTransport. Cette dernière dispose également de quatre processeurs Opteron. L'ensemble constitue donc une machine relativement complexe avec des chemins d'accès à la mémoire distante qui varient. Les processeurs possèdent 2 Mo (2×1 Mo) de mémoire cache partagée par les deux cœurs et sont cadencés à 2.2 Ghz.

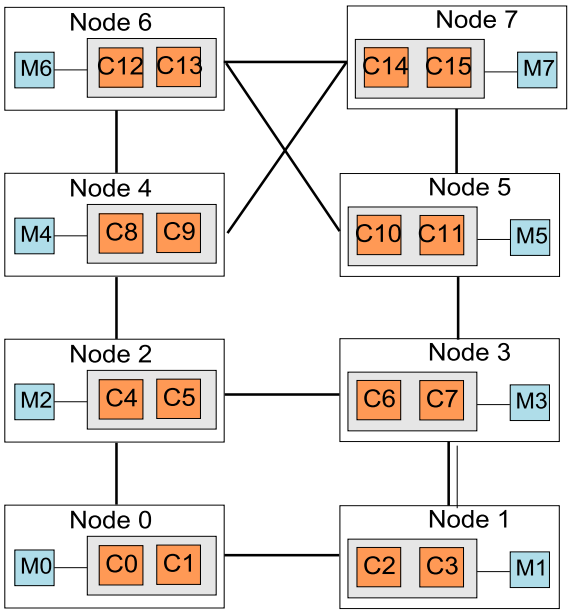

FIGURE B.5 – Plateforme `Idkoiff`.

B.2 Architectures SMP

B.2.1 Plateforme `Phoebus`

Le Centre de Calcul Scientifique en région Centre (CCSC) héberge notamment la plateforme de calcul `Phoebus` . Cette dernière comprend 42 nœuds de calcul interconnecté par un réseau de Infiniband. Chaque nœud est composé de 2 processeurs quad-cœurs Intel Harpertown cadencés à 3.0 GHz (Xeon-5450) avec 12 Mo de mémoire cache (2×6 Mo). Le bus mémoire (FSB) est de 1333 Mo/s et la mémoire totale disponible sur chaque nœud est de 25 Go.

B.2.2 Plateforme M3PEC `Decrypthon`

Décrypthon est une machine SMP du pôle M3PEC de l'Université de Bordeaux 1 composé de 16 nœuds IBM P575 interconnectées par un réseau IBM Fédération à 12 Gb/s. Chaque nœud contient 8 processeurs Power5 dual-cœur disposant de 36 Mo de mémoire cache et cadencés à 1.5 Ghz. Ces derniers dispose de 32 Go de mémoire vive.

B.2.3 Plateforme GENCI-CINES `Jade`

La plateforme `Jade` du GENCI-CINES, est localisée à Montpellier (GENCI-CINES), elle est composée (avant extension en 2010) de 1536 nœuds interconnectés par un réseau rapide Infiniband. Chacun de ces nœuds est composé de deux processeurs Intel Harpertown quad-cœur cadencé à 3.0 GHz et disposant de 12 Mo (2×6 Mo) de mémoire cache (Xeon-5472). Le bus mémoire (FSB) est de 1600 Mo/s et la mémoire totale disponible sur chaque nœud est de 30 Go.

Bibliographie

[1] Abbo, A. J. *Finite Element Algorithms for elastoplasticity and consolidation.* PhD thesis, University of Newcastle, United Kingdom, 1997.

[2] Abdelkhalek, R., Calandra, H., Coulaud, O., Roman, J., and Latu, G. Fast Seismic Modeling and Reverse Time Migration on a GPU Cluster. In *The 2009 High Performance Computing & Simulation - HPCS'09* (Leipzig, Allemagne, 2009), pp. 36–43.

[3] Agullo, E., Demmel, J., Dongarra, J., Hadri, B., Kurzak, J., Langou, J., Ltaief, H., Luszczek, P., and Tomov, S. Numerical linear algebra on emerging architectures : The PLASMA and MAGMA projects. *Journal of Physics : Conference Series 180*, 1 (2009), 012037.

[4] Agullo, E., Guermouche, A., and L'Excellent, J.-Y. A parallel out-of-core multifrontal method : Storage of factors on disk and analysis of models for an out-of-core active memory. *Parallel Computing 34*, 6-8 (2008), 296–317.

[5] Akcelik, V., Bielak, J., Biros, G., Epanomeritakis, I., Fernandez, A., Ghattas, O., Kim, E. J., Lopez, J., O'Hallaron, D., Tu, T., and Urbanic, J. High Resolution Forward and Inverse Earthquake Modeling on Terasacale Computers. In *Supercomputing'03, 2003 ACM/IEEE conference on Supercomputing* (Phoenix, USA, November 2003), pp. 52–66.

[6] Aki, K., and Richards, P. G. *Quantitative Seismology, 2nd ed.* University Science Books, 2002.

[7] Alterman, Z., and Karal, F. C. Propagation of elastic waves in layered media by finite difference methods. *Bull. Seismol. Soc. Am. 58* (1968), 367–398.

[8] Amestoy, P., Guermouche, A., L'Excellent, J.-Y., and Pralet, S. Hybrid scheduling for the parallel solution of linear systems. *Parallel Computing 32*, 2 (2006), 136–156.

[9] Amestoy, P. R., Duff, I. S., L'Excellent, J.-Y., and Koster, J. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIAM J. Matrix Anal. Appl. 23*, 1 (2001), 15–41.

[10] Antony, J., Janes, P. P., and Rendell, A. P. Exploring Thread and Memory Placement on NUMA Architectures : Solaris and Linux, UltraSPARC/FirePlane and Opteron/HyperTransport. In *Proceedings of HiPC 2006* (Bangalore, India, Dec 2006), pp. 338–352.

[11] Aochi, H., and Madariaga, R. The 1999 Izmit, Turkey, Earthquake : Nonplanar Fault Structure, Dynamic Rupture Process, and Strong Ground Motion. *Bulletin of the Seismological Society of America 93*, 3 (2003), 1249–1266.

[12] Aoi, S., Sekiguchi, H., Morikawa, N., and Kunugi, T. Source process of the 2007 niigata-ken chuetsu-oki earthquake derived from near-fault strong motion data. *Earth, Planets, and Space 60* (Nov. 2008), 1131–1135.

[13] Augonnet, C., Thibault, S., Namyst, R., and Wacrenier, P.-A. StarPU : A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. In *Euro-Par 2009 Parallel Processing, 15th International Euro-Par Conference* (Delft, The Netherlands, August 2009), pp. 863–874.

[14] Augustin, W., Heuveline, V., and Weiss, J.-P. Optimized Stencil Computation Using In-Place Calculation on Modern Multicore Systems. In *Euro-Par 2009 Parallel Processing, 15th International Euro-Par Conference* (Delft, The Netherlands, August 2009), pp. 772–784.

[15] Ayguadé, E., Badia, R. M., Igual, F. D., Labarta, J., Mayo, R., and Quintana-Ortí, E. S. An Extension of the StarSs Programming Model for Platforms with Multiple GPUs. In *Euro-Par 2009 Parallel Processing, 15th International Euro-Par Conference, Delft, The Netherlands, August 25-28, 2009. Proceedings* (Aug. 2009), vol. 5704 of *Lecture Notes in Computer Science*, pp. 851–862.

[16] Azzam Haidar. *Sur l'extensibilité parallèle de solveurs linéaires hybrides pour des problèmes tridimensionels de grandes tailles.* PhD thesis, Institut National Polytechnique de Toulouse, 2008.

[17] Bardet, J., and Tobita, T. Nera : A computer program for nonlinear earthquake site response analysis of layered soil deposits. Tech. rep., Department of Civil Engineering, University of Southern California, USA, 2000. http ://geoinfo.usc.edu/gees.

[18] Benjemaa, M. *Simulation numérique de la rupture dynamique des séismes par des méthodes volumes finis en maillages non structurés.* PhD thesis, Université de Nice-Sophia Antipolis, Nov. 2008.

[19] Bérenger, J. P. A Perfectly Matched Layer for the absorption of electromagnetic waves. *J. Comput. Phys. 114* (1994), 185–200.

[20] Beresnev, I., and Wen, K. Nonlinear soil response - a reality ? *Bull. Seism. Soc. Am. 86*, 6 (1996), 1964–1978.

[21] Berge, C. *Théorie des graphes et ses applications.* Collection universitaire de mathématiques. Dunod, 1958.

[22] Berger, M. J., and Oliger, J. Adaptive Mesh Refinement for Hyperbolic Partial Differential Equations. *J. Comput. Phys. 53* (1984), 484–512.

[23] Bernardie, S., Foerster, E., and Modaressi, H. Non-linear site response simulations in chang-hwa region during the 1999 chi-chi earthquake, taiwan. *Soil Dynamics and Earthquake Engineering 26*, 11 (2006), 1038 – 1048.

[24] Bhandarkar, M. A., Kalé, L. V., de Sturler, E., and Hoeflinger, J. Adaptive Load Balancing for MPI Programs. In *ICCS 2001, International Conference on Computational Science* (San Francisco, USA, May 2001), pp. 108–117.

[25] Blumofe, R. D., Joerg, C. F., Kuszmaul, B. C., Leiserson, C. E., Randall, K. H., and Zhou, Y. Cilk : an efficient multithreaded runtime system. *SIGPLAN Not. 30*, 8 (1995), 207–216.

[26] Bohlen, T. Parallel 3-D viscoelastic finite difference seismic modelling. *Computers and Geosciences 28* (Oct. 2002), 887–899.

[27] Bohlen, T., and Saenger, E. H. Accuracy of heterogeneous staggered-grid finite-difference modeling of Rayleigh waves. *Geophysics 71*, 4 (2006), T109–T115.

[28] Bouchon, M. Effect of topography on surface motion. *Bull. Seism. Soc. Am. 63*, 3 (1973), 615–632.

[29] Bouchon, M., and Sánchez Sesma, F. J. Boundary integral equations and boundary elements methods in elastodynamics. *Advances in Geophysics 48* (2007), 157–189.

[30] Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., and Namyst, R. hwloc : a Generic Framework for Managing Hardware Affinities in HPC Applications. In *PDP2010, 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing* (Pisa, Italia, Feb. 2010), pp. 180–186.

[31] Broquedis, F., Furmento, N., Goglin, B., Wacrenier, P.-A., and Namyst, R. ForestGOMP : an efficient OpenMP environment for NUMA architectures. *International Journal on Parallel Programming, Special Issue on OpenMP 38*, 5 (2010), 418–439.

[32] Brossier, R. *Imagerie sismique à deux dimensions des milieux visco-élastiques par inversion des formes d'ondes : développements méthodologiques et applications.* PhD thesis, Université de Nice-Sophia Antipolis, France, Nov. 2009.

[33] Burstedde, C., Burtsher, M., Ghattas, O., Stadler, G., Tu, T., and Wilcox, L. C. ALPS : A Framework for Parallel Adaptive PDE Solution. *Journal of Physics : Conference Series 180* (2009).

[34] Capdeville, Y., Chaljub, E., Vilotte, J. P., and Montagner, J. P. Coupling the spectral element method with a modal solution for elastic wave propagation in global Earth models. *Geophys. J. Int. 152* (2003), 34–67.

[35] Carli, S. D. *Inversion des séismes par approximation elliptique : Application au séisme de Tottori.* PhD thesis, Université Pierre et Marie Curie, France, 2008.

[36] Cerjan, C., Kosloff, D., Kosloff, R., and Reshef, M. A nonreflecting boundary condition for discrete acoustic and elastic wave equation. *Geophysics 50* (1985), 705–708.

[37] Chaillat, S. *Fast Multipole Method for 3-D elastodynamic boundary integral equations. Application to seismic wave propagation.* PhD thesis, ENPC, Dec. 2008.

[38] Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., and McDonald, J. *Parallel Programming in OpenMP.* Morgan Kaufmann, Elsevier, San Francisco, USA, 2000.

[39] Chavez, M., Cabrera, E., Madariaga, R., Perea, N., Moulinec, C., Emerson, D., Ashworth, M., and Salazar, A. Benchmark Study of a 3D Parallel Code for the Propagation of Large Subduction Earthquakes. In *Proceedings of the 15th Euro PVM/MPI Users' Group Meeting* (Dublin, Ireland, 2008), pp. 303–310.

[40] Clauss, P.-N., Gustedt, J., and Suter, F. Out-of-Core Wavefront Computations with Reduced Synchronization. In *PDP '08 : Proceedings of the 16th Euromicro Conference on Parallel, Distributed and Network-Based Processing* (Toulouse, France, 2008), pp. 293–300.

[41] Cockburn, B., Karniadakis, G. E., and Shu, C.-W. *Discontinuous Galerkin Methods : Theory, Computation and Applications.* Springer-Verlag Berlin, 1999.

[42] Collino, F., and Tsogka, C. Application of the PML absorbing layer model to the linear elastodynamic problem in anisotropic heterogeneous media. *Geophysics 66*, 1 (2001), 294–307.

[43] Cruz-Atienza, V.-M. *Rupture Dynamique des Failles Non-Planaires en Différences Finies.* PhD thesis, Université de Nice Sophia - Antipolis, FRANCE, 2006.

[44] Cui, Y., Moore, R., Olsen, K., Chourasia, A., Maechling, P., Minster, B., Day, S., Hu, Y., Zhu, J., and Jordan, T. Toward petascale earthquake simulations. *Acta Geotechnica 4* (2009), 79–93.

[45] Danjean, V. *Contribution à l'élaboration d'ordonnanceurs de processus légers performants et portables pour architectures multiprocesseurs.* PhD thesis, École normale supérieure de Lyon, FRANCE, Dec. 2004.

[46] Datta, K., Kamil, S., Williams, S., Oliker, L., Shalf, J., and Yelick, K. Optimization and performance modeling of stencil computations on modern microprocessors. *SIAM Review 51*, 1 (2009), 129–159.

[47] Datta, K., Murphy, M., Volkov, V., Williams, S., Carter, J., Oliker, L., Patterson, D., Shalf, J., and Yelick, K. Stencil computation optimization and autotuning on state-of-the-art multicore architectures. In *SC '08 : Proceedings of the 2008 ACM/IEEE conference on Supercomputing* (Austin, Texas, USA, 2008), pp. 1–12.

[48] De Martin, F. *Influence of the Nonlinear Behavior of Soft Soils on Strong Ground Motions.* PhD thesis, École Centrale de Paris, France, June 2010.

[49] Dhatt, G., and Touzot, G. *The finite element method displayed.* John Wiley and sons, New York, 1984.

[50] Donath, S., Igelberger, K., Wellein, G., Zeiser, T., Nitsure, A., and Rude, U. Performance comparison of different parallel lattice boltzmann implementations on multi-core multi-socket systems. *Int. J. Comput. Sci. Eng. 4*, 1 (2008), 3–11.

[51] Ducellier, A., and Aochi, H. Implementing a Convolutional Perfectly Matched Layer in a finite difference code for the simulation of seismic wave propagation in a 3D elastic medium. Progress report. Technical Report BRGM/RP-55922-FR, BRGM, 2007.

[52] Dursun, H., Nomura, K.-I., Peng, L., Seymour, R., Wang, W., Kalia, R. K., Nakano, A., and Vashishta, P. A Multilevel Parallelization Framework for High-Order Stencil Computations. In *Euro-Par 2009 Parallel Processing, 15th International Euro-Par Conference* (Delft, The Netherlands, August 2009), pp. 642–653.

[53] Engquist, B., and Majda, A. Absorbing boundary conditions for the numerical simulation of waves. *Math. Comp. 31* (1977), 629–651.

[54] Faverge, M. *Ordonnancement hybride statique-dynamique en algèbre linéaire creuse pour de grands clusters de machines NUMA et multi-coeurs.* PhD thesis, LaBRI, Université Bordeaux I, Talence, Dec. 2009.

[55] Field, E. H., Johnson, P. A., Beresnev, I. A., and Zeng, Y. Nonlinear ground-motion amplification by sediments during the 1994 Northridge earthquake. *Nature 390* (Dec. 1997), 599–602.

[56] Foerster, E., Courrioux, G., Aochi, H., De Martin, F., and Bernardie, S. Seismic hazard assessment through numerical simulation at different scales : application to Nice city (French Riviera) . In *14th World Conference on Earthquake Engineering* (Beijing ,China, October 2008), p. 4.

[57] Foerster, E., and Modaressi, H. Nonlinear numerical method for earthquake site response analysis II - case studies. *Bulletin of Earthquake Engineering 5* (2007), 325–345.

[58] Frigo, M. *Portable High-Performance Programs.* PhD thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, USA, 1999.

[59] Frigo, M., and Strumpen, V. Cache oblivious stencil computations. In *ICS '05 : Proceedings of the 19th annual International conference on Supercomputing* (Cambridge, Massachusetts, 2005), pp. 361–366.

[60] Frigo, M., and Strumpen, V. The cache complexity of multithreaded cache oblivious algorithms. *Theor. Comp. Sys. 45*, 2 (2009), 203–233.

[61] Furumura, T., and Chen, L. Parallel simulation of strong ground motions during recent and historical damaging earthquakes in Tokyo, Japan. *Parallel Computing 31*, 2 (2005), 149–165.

[62] Gabriel, E., Fagg, G. E., Bosilca, G., Angskun, T., Dongarra, J. J., Squyres, J. M., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R. H., Daniel, D. J., Graham, R. L., and Woodall, T. S. Open MPI : Goals, Concept, and Design of a Next Generation MPI Implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting* (Budapest, Hungary, September 2004), pp. 97–104.

[63] George, T., Gupta, A., and Sarin, V. An empirical analysis of iterative solver performance for spd systems. Tech. Rep. RC24737, IBM T. J. Watson Research Center, 2009.

[64] Giraud, L., Haidar, A., and Pralet, S. Using multiple levels of parallelism to enhance the performance of domain decomposition solvers. *Parallel Comput. 36*, 5-6 (2010), 285–296.

[65] Goglin, B., and Furmento, N. Enabling high-performance memory migration for multithreaded applications on LINUX. In *IPDPS '09 : Proceedings of the 2009 IEEE International Symposium on Parallel&Distributed Processing* (Rome, Italy, 2009), pp. 1–9.

[66] Graves, R. W. Simulating seismic wave propagation in 3D elastic media using staggered-grid finite differences. *Bull. Seismol. Soc. Am. 86*, 4 (1996), 1091–1106.

[67] Gropp, W., Lusk, E., Doss, N., and Skjellum, A. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing 22*, 6 (Sept. 1996), 789–828.

[68] Gupta, A. A shared- and distributed-memory parallel general sparse direct solver. *Appl. Algebra Eng., Commun. Comput. 18*, 3 (2007), 263–277.

[69] Gupta, A., Koric, S., and George, T. Sparse matrix factorization on massively parallel computers. Tech. Rep. RC24809, IBM T. J. Watson Research Center, 2009.

[70] Hadjidoukas, P. E., and Dimakopoulos, V. V. Nested Parallelism in the OMPi OpenMP/C Compiler. In *Euro-Par 2007, Parallel Processing, 13th International Euro-Par Conference* (Rennes, France, 2007), pp. 662–671.

[71] Hénon, P. *Distribution des Données et Régulation Statique des Calculs et des Communications pour la Résolution de Grands Systèmes Linéaires Creux par Méthode Directe.* PhD thesis, LaBRI, Université Bordeaux I, Talence, Talence, France, Nov. 2001.

[72] Hénon, P., Ramet, P., and Roman, J. On using an hybrid MPI-Thread programming for the implementation of a parallel sparse direct solver on a network of SMP nodes. In *PPAM 2006, Sixth International Conference on Parallel Processing and Applied Mathematics* (Poznan, Pologne, Sept. 2005), pp. 1050–1057.

[73] Hestholm, S., and Ruud, B. 3D free-boundary conditions for coordinate-transform finite-difference seismic modelling. *Geophysical Prospecting 50* (2002), 463–474.

[74] Higdon, R. L. Absorbing boundary conditions for elastic waves. *Geophysics 56* (1991), 231–241.

[75] Hikima, K., and Koketsu, K. Joint inersion of the 2007 Niigata-ken Chuetsu-Oki earthquake from geodetic, farfield and nearfield data. In *Japan Geoscience Union Meeting, Makuhari, Japon, 25-30 May* (2008).

[76] Jeremie Gaidamour. *Conception d'un solveur linéaire creux parallèle hybride direct-itératif.* PhD thesis, Université Bordeaux I, FRANCE, Dec. 2009.

[77] Jin, G., Mellor-Crummey, J., and Fowler, R. Increasing temporal locality with skewing and recursive blocking. In *Supercomputing '01 : Proceedings of the 2001 ACM/IEEE conference on Supercomputing* (Denver, Colorado, USA, 2001), pp. 43–43.

[78] Julien Langou. *Résolution de systèmes linéaires de grande taille avec plusieurs seconds membres.* PhD thesis, INSA de Toulouse, FRANCE, 2003.

[79] Kale, L. V., and Zheng, G. Charm++ and AMPI : Adaptive Runtime Strategies via Migratable Objects. In *In Advanced Computational Infrastructures for Parallel and Distributed Applications.* Wiley-Interscience, 2009, pp. 265–282.

[80] Kamil, S., Datta, K., Williams, S., Oliker, L., Shalf, J., and Yelick, K. Implicit and explicit optimizations for stencil computations. In *MSPC'06, the 2006 Workshop on Memory System Performance and Correctness* (San Jose, USA, Oct. 2006), pp. 51–60.

[81] Karypis, G., and Kumar, V. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM J. Sci. Comput. 20*, 1 (1998), 359–392.

[82] Käser, M., and Dumbser, M. An Arbitrary High Order Discontinuous Galerkin Method for Elastic Waves on Unstructured Meshes I : The Two-Dimensional Isotropic Case with External Source Terms. *Geophysical Journal International 166*, 2 (2006), 855–877.

[83] Kato, A., Kurashimo, E., Igarashi, T., Sakai, S., Iidaka, T., Shinohara, M., Kanazawa, T., Yamada, T., Hirata, N., and Iwasaki, T. Reactivation of ancient rift systems triggers devastating intraplate earthquakes. *Geophys. Res. Lett. 36* (2009), L05301.

[84] Kelly, K. R., Ward, R. W., Treitel, S., and Alford, R. M. Synthetic seismograms : a finite difference approach. *Geophysics 41* (1976), 2–27.

[85] Kleen, A. A NUMA API for Linux. Tech. Rep. Novell-4621437, April 2005.

[86] Knoll, D. A., and Keyes, D. E. Jacobian-free newton-krylov methods : a survey of approaches and applications. *J. Comput. Phys. 193*, 2 (January 2004), 357–397.

[87] Komatitsch, D., Göddeke, D., Erlebacher, G., and Michéa, D. Modeling the propagation of elastic waves using spectral elements on a cluster of 192 GPUs. *Computer Science Research and Development 25*, 1-2 (2010), 75–82.

[88] Komatitsch, D., Labarta, J., and Michéa, D. A simulation of seismic wave propagation at high resolution in the inner core of the Earth on 2166 processors of MareNostrum. *Lecture Notes in Computer Science 5336* (2008), 364–377.

[89] Komatitsch, D., Liu, Q., Tromp, J., Süss, P., Stidham, C., and Shaw, J. H. Simulations of ground motion in the Los Angeles basin based upon the spectral-element method. *Bull. Seismol. Soc. Am. 94*, 1 (2004), 187–206.

[90] Komatitsch, D., and Martin, R. An unsplit convolutional Perfectly Matched Layer improved at grazing incidence for the seismic wave equation. *Geophysics 72*, 5 (2007), SM155–SM167.

[91] Komatitsch, D., and Tromp, J. Spectral-element simulations of global seismic wave propagation-I. Validation. *Geophys. J. Int. 149*, 2 (2002), 390–412.

[92] Komatitsch, D., and Tromp, J. A Perfectly Matched Layer absorbing boundary condition for the second-order seismic wave equation. *Geophys. J. Int. 154*, 1 (2003), 146–153.

[93] Komatitsch, D., and Vilotte, J. P. The spectral-element method : an efficient tool to simulate the seismic response of 2D and 3D geological structures. *Bull. Seismol. Soc. Am. 88*, 2 (1998), 368–392.

[94] Kramer, S. *Geotechnical Earthquake Engineering.* Prentice Hall, 1996.

[95] Lee, M., and Finn, W. Desra-2 : Dynamic effective stress response analysis of soil deposits with energy transmitting boundary including assessment of liquefaction potential. Tech. rep., Faculty of Applied Science, University of British Columbia, Vancouver, Canada, 1978.

[96] Lee, V. W., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A. D., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R., and Dubey, P. Debunking the 100X GPU vs. CPU myth : an evaluation of throughput computing on CPU and GPU. *SIGARCH Comput. Archit. News 38*, 3 (2010), 451–460.

[97] Levander, A. R. Fourth-order finite-difference *P-SV* seismograms. *Geophysics 53* (1988), 1425–1436.

[98] Li, X. S., and Demmel, J. W. SuperLU_DIST : A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Math. Softw. 29*, 2 (2003), 110–140.

[99] Löf, H., Nordén, M., and Holmgren, S. Improving Geographical Locality of Data for Shared Memory Implementations of PDE Solvers. In *International Conference on Computational Science* (2004), pp. 9–16.

[100] Lombard, B., Piraux, J., Gélis, C., and Virieux, J. Free and smooth boundaries in 2-D finite-difference schemes for transient elastic waves. *Geophys. J. Int. 172*, 1 (2007), 252–261.

[101] Lopez-Caballero, F., Modaressi-Farahmand Razavi, A., and Modaressi, H. Nonlinear numerical method for earthquake site response analysis I-elastoplastic cyclic model and parameter identification strategy. *Bulletin of Earthquake Engineering 5(3)* (2007), 303–323.

[102] Lu, J. *Parallel Finite Element Modeling of Earthquake Site Response and Liquefaction.* PhD thesis, University of California San Diego, USA, 2006.

[103] Lu, J., Peng, J., Elgamal, A., Yang, Z., and Law, K. H. Parallel finite element modeling of earthquake ground response and liquefaction. *Earthquake Engineering and Engineering Vibration 3* (June 2004), 23–37.

[104] Luccioni, L. X., Pestana, J. M., and Taylor, R. L. Finite element implementation of non-linear elastoplastic constitutive laws using local and global explicit algorithms with automatic error control. *International Journal for Numerical Methods in Engineering 50*, 5 (2001), 1191–1212.

[105] Lusk, E. L., and Yelick, K. A. Languages for High-Productivity Computing : the DARPA HPCS Language Project. *Parallel Processing Letters 17*, 1 (2007), 89–102.

[106] Lévêque, B. *Coloring graphs : structures and algorithms.* PhD thesis, Université Joseph Fourier de Grenoble, FRANCE, Oct. 2007.

[107] Lysmer, J., and Drake, L. A. A finite element method for seismology. In *Methods in Computational Physics*, B. Alder, S. Fernbach, and B. A. Bolt, Eds., vol. 11. Academic Press, New York, USA, 1972, ch. 6, pp. 181–216.

[108] Madariaga, R. Dynamics of an expanding circular fault. *Bull. Seismol. Soc. Am. 66*, 3 (1976), 639–666.

[109] Marathe, J., and Mueller, F. Hardware Profile-Guided Automatic Page Placement for ccNUMA Systems. In *PPoPP '06 : Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming* (New York, NY, USA, 2006), pp. 90–99.

[110] Marfurt, K. J. Accuracy of finite-difference and finite-element modeling of the scalar and elastic wave equation. *Geophysics 49(5)* (1984), 533–549.

[111] Martin, R., Komatitsch, D., Blitz, C., and Goff, N. L. Simulation of Seismic Wave Propagation in an Asteroid Based upon an Unstructured MPI Spectral-Element Method : Blocking and Non-blocking Communication Strategies. In *Vecpar 2008, 8th International Conference on High Performance Computing for Computational Science* (Toulouse, France, 2008), pp. 350–363.

[112] Martin, R., Komatitsch, D., and Gedney, S. D. A variational formulation of a stabilized unsplit convolutional perfectly matched layer for the isotropic or anisotropic seismic wave equation. *Comput. Model. Eng. Sci. 37*, 3 (2008), 274–304.

[113] McCalpin, J. D. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), 19–25.

[114] McKinley, K. S., Carr, S., and Tseng, C.-W. Improving data locality with loop transformations. *ACM Trans. Program. Lang. Syst. 18*, 4 (1996), 424–453.

[115] Mercier, G., and Clet-Ortega, J. Towards an efficient process placement policy for mpi applications in multicore environments. In *16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface* (Espoo, Finland, 2009), pp. 104–115.

[116] Michéa, D., and Komatitsch, D. Accelerating a 3D finite-difference wave propagation code using GPU graphics cards. *Geophys. J. Int. 182*, 1 (2010), 389–402.

[117] Minkoff, S. E. Spatial Parallelism of a 3D Finite Difference Velocity-Stress Elastic Wave Propagation Code. *SIAM J. Sci. Comput. 24*, 1 (2002), 1–19.

[118] Miyake, H., Koketsu, K., Hikima, K., Shinohara, M., and Kanazawa, T. Source Fault of the 2007 Chuetsu-oki, Japan, Earthquake. *Bull. Seismol. Soc. Am. 100*, 1 (2010), 384–391.

[119] Moczo, P., Robertsson, J., and Eisner, L. The finite-difference time-domain method for modeling of seismic wave propagation. In *Advances in Wave Propagation in Heterogeneous Media*, vol. 48 of *Advances in Geophysics*. Elsevier - Academic Press, 2007, ch. 8, pp. 421–516.

[120] Moczo, P., Robertsson, J., and Eisner, L. The finite-difference time-domain method for modeling of seismic wave propagation. In *Advances in Wave Propagation in Heterogeneous Media*, R.-S. Wu and V. Maupin, Eds., vol. 48 of *Advances in Geophysics*. Elsevier - Academic Press, 2007, ch. 8, pp. 421–516.

[121] Modaressi, H. *Modélisation numérique de la propagation des ondes dans les milieux poreux anélastiques*. PhD thesis, École Centrale de Paris, France, 1987.

[122] Mu, T., Tao, J., Schulz, M., and McKee, S. A. Interactive Locality Optimization on NUMA Architectures. In *SoftVis '03 : Proceedings of the 2003 ACM Symposium on Software Visualization* (San Diego, California, 2003), pp. 133– 141.

[123] Namyst, R., and Méhaut, J.-F. PM2 : Parallel multithreaded machine. a multithreaded environment on top of PVM. In *Proceedings of the 2nd Euro PVM Users' Group Meeting* (Lyon, France, 1995), pp. 179–184.

[124] Norton, C. D., Lou, J. Z., and Cwik, T. A. Status and Directions for the PYRAMID Parallel Unstructured AMR Library. In *IPDPS 2001, 15th International Parallel & Distributed Processing Symposium* (San Francisco, USA, 2001), pp. 120–128.

[125] Ohminato, T., and Chouet, B. A. A free-surface boundary condition for including 3D topography in the finite difference method. *Bull. Seismol. Soc. Am. 87* (1997), 494–515.

[126] Operto, S., Virieux, J., Amestoy, P., L'Excellent, J.-Y., Giraud, L., and Ben Hadj Ali, H. 3D finite-difference frequency-domain modeling of visco-acoustic wave propagation using a massively parallel direct solver : A feasibility study. *Geophysics 72*, 5 (2007), SM195–SM211.

[127] Orozco, D., and Gao, G. R. Mapping the FDTD Application to Many-Core Chip Architectures. In *ICPP 2009, International Conference on Parallel Processing* (Vienna, Austria, September 2009), pp. 309–316.

[128] Patzák, B., Rypl, D., and Bittnar, Z. Parallel explicit finite element dynamics with nonlocal constitutive models. *Computers and Structures 79* (2001), 2287–2297.

[129] Pellegrini, F., and Roman, J. SCOTCH : A Software Package for Static Mapping by Dual Recursive Bipartitioning of Process and Architecture Graphs. In *HPCN Europe 1996, International Conference and Exhibition on High-Performance Computing and Networking* (1996), pp. 493–498.

[130] Pérache, M. *Contribution à l'élaboration d'environnements de programmation dédiés au calcul scientifique hautes performances.* PhD thesis, Université de Bordeaux 1, FRANCE, Oct. 2006.

[131] Prokop, H. Cache-Oblivious Algorithms. Master's thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, USA, 1999.

[132] Ramet, P. *Optimisation de la Communication et de la Distribution des Données pour des Solveurs Parallèles Directs en Algèbre Linéaire Dense et Creuse.* PhD thesis, LaBRI, Université Bordeaux I, Talence, France, Jan. 2000.

[133] Reinders, J. *Intel Threading Building Blocks.* O'REILLY, 2007.

[134] Ribeiro, C. P., Méhaut, J.-F., Carissimi, A., Castro, M. B., and Fernandes, L. G. Memory affinity for hierarchical shared memory multiprocessors. In *21st International Symposium on Computer Architecture and High Performance Computing, SBAC-PAD 2009* (São Paulo, Brazil, october 2009), pp. 59–66.

[135] Rivera, G., and Tseng, C.-W. Tiling optimizations for 3D scientific computations. In *Supercomputing '00 : Proceedings of the 2000 ACM/IEEE conference on Supercomputing* (Dallas, United States, 2000), pp. 32–38.

[136] Robertsson, J. O. A. A numerical free-surface condition for elastic/viscoelastic finite-difference modeling in the presence of topography. *Geophysics 61* (1996), 1921–1934.

[137] Saad, Y. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, 2003.

[138] Sagiya, T., Miyazaki, S., and Tada, T. Continuous GPS Array and Present-day Crustal Deformation of Japan. *Pure Appl. Geophys. 157* (2000), 2303–2322.

[139] Salichon, J., Kohrs-Sansorny, C., Bertrand, E., and Courboulex, F. A Mw 6.3 earthquake scenario in the city of Nice (southeast France) : ground motion simulations. *Journal of Seismology 14*, 3 (2010), 523–541.

[140] Schloegel, K., Karypis, G., and Kumar, V. A unified algorithm for load-balancing adaptive scientific simulations. In *Supercomputing '00 : Proceedings of the 2000 ACM/IEEE conference on Supercomputing* (Dallas, United States, 2000), pp. 59–65.

[141] Schnabel, P. B., Lysmer, J., and Seed, H. B. SHAKE : A computer program for earthquake response analysis of horizontally-layered sites. Tech. rep., University of California, Berkeley, USA, 1972. Report No. UCB/EERC-72/12.

[142] Seguin, S., Cracraft, M., and Drewniak, J. Static and quasi-dynamic load balancing in parallel FDTD codes for signal integrity, power integrity, and packaging applications. In *IEEE International Symposium on Electromagnetic Compatibility* (Santa Clara, USA, August 2004), pp. 107–112.

[143] Sekiguchi, H., and Yoshimi, M. Broadband Ground Motion Reconstruction for the Kanto Basin during the 1923 Kanto Earthquake. *Pure Appl. Geophys.* (2010), 1–22.

[144] Semblat, J.-F., and Brioist, J. J. Efficiency of higher order finite elements for the analysis of seismic wave propagation. *Journal of Sound and Vibration 231*, 2 (2000), 460–467.

[145] Strumpen, V., and Frigo, M. Software engineering aspects of cache oblivious stencil computations. Technical Report W0608-077, IBM, 2006.

[146] Tarrass, I. *3D seismic modeling over very large models : theory, implementation and applications to acoustic and elastic media in the presence of topography.* PhD thesis, Université de Toulouse - ENSEEIHT, Dec. 2009.

[147] Terboven, C., Mey, D. A., Schmidl, D., Jin, H., and Reichstein, T. Data and Thread affinity in OpenMP programs. In *2008 workshop on Memory access on future processors* (Ischia, Italy, 2008), pp. 377–384.

[148] Thakur, R., and Gropp, W. Improving the Performance of Collective Operations in MPICH. In *Recent Advances in Parallel Virtual Machine and Message Passing Interface,10th European PVM/MPI Users' Group Meeting* (Venice, Italy, 2003), pp. 257–267.

[149] Thibault, S. *Ordonnancement de processus légers sur architectures multiprocesseurs hiérarchiques : BubbleSched, une approche exploitant la structure du parallélisme des applications.* PhD thesis, Université Bordeaux I, FRANCE, Dec. 2007.

[150] Trahay, F. *De l'interaction des communications et de l'ordonnancement de threads au sein des grappes de machines multi-cœurs.* PhD thesis, Université Bordeaux 1, FRANCE, Nov. 2009.

[151] Tromeur-Dervout, D., and Vassilevski, Y. Choice of initial guess in iterative solution of series of systems arising in fluid flow simulations. *J. Comput. Phys. 219*, 1 (2006), 210 – 227.

[152] Vilotte, J.-P., and Festa, G. Spectral element simulation of rupture dynamics on curvilinear faults. *EOS 85*, 47 (2004), Abstract S32B–08.

[153] Virieux, J. *P-SV* wave propagation in heterogeneous media : velocity-stress finite-difference method. *Geophysics 51* (1986), 889–901.

[154] Wolf, M. E., Maydan, D. E., and Chen, D.-K. Combining loop transformations considering caches and scheduling. In *MICRO 29 : Proceedings of the 29th annual ACM/IEEE international symposium on Microarchitecture* (Washington, DC, USA, 1996), IEEE Computer Society, pp. 274–286.

[155] Wonnacott, D. Time skewing for parallel computers. In *Languages and Compilers for Parallel Computing* (1999), pp. 477–480.

[156] Xu, J., Bielak, J., Ghattas, O., and Wanga, J. Three-dimensional nonlinear seismic ground motion modeling in inelastic basins. *Physics of the Earth and Planetary Interiors 137* (2003), 81–95.

[157] Zienkiewicz, O. C., and Taylor, R. L. *The Finite Element Method. Basic Formulation and Linear Problems*, vol. 1. McGraw-Hill, 1989.

[158] Zienkiewicz, O. C., Valliappan, S., and King, I. P. Elasto-plastic solutions of engineering problems 'initial stress', finite element approach. *International Journal for Numerical Methods in Engineering 1* (1969), 75–100.

Liste des publications

Revues internationales avec comité de lecture

[1] Dupros, F., Pousa, C., Aochi, H., Méhaut, J.-F, Komatitsch, D., and Roman, J. **Efficient stencil computation on multicore and hierarchical architectures : Application to seismic wave propagation.** *Concurrency and Computation : Practice and Experience*, Note : Soumis

[2] Dupros, F., De Martin, F., Foerster, E., Komatitsch, D., and Roman, J. **High-performance finite-element simulations of seismic wave propagation in three-dimensional non linear inelastic geological media.** *Parallel Computing* (2010), vol. 36, 5-6, pp. 308-325.

Actes de conférences et workshops internationaux

[3] Dupros, F., Pousa, C., Carissimi, and Méhaut, J.-F. **Parallel simulations of seismic wave propagation on NUMA architectures.** In *Parallel Computing : From Multicores and GPU's to Petascale, ParCo'09, September 1-4*, pages 67-74, (Lyon, France, 2009).

[4] Pousa, C., Castro, M., Dupros, F., Carissimi, A., Fernandes, L.-G., and Méhaut, J.-F. **High-performance Applications on Hierarchical Shared Memory Multiprocessors.** In *Colloquium of computation : Brazil / INRIA, Cooperations, Advances and Challenges, Colibri'09, July 21-24*, pages 55-60, (Bento-Goncalves, Brazil, 2009).

[5] Dupros, F., Aochi, H., Ducellier, A., Komatitsch, D., and Roman, J. **Exploiting Intensive Multithreading for the Efficient Simulation of 3D Seismic Wave Propagation.** In *11th IEEE International Conference on Computational Science and Engineering, CSE'08, July 16-18*, pages 253-260, (São Paulo, Brazil, 2008).

Communications orales dans des conférences et workshops internationaux

[6] Dupros, F., Ducellier, A., Aochi, H., Komatitsch, D., and Roman, J. **Three-dimensional parallel finite-difference modeling of seismic wave propagation on multicore architectures : Application to the Nice, France, urban area.** In *9th International Conference on Mathematical and Numerical Aspects of Waves Propagation, Waves'09, June 15-19*, (Pau, France, 2009).

[7] Dupros, F., De Martin, F., Foerster, E., Komatitsch, D., and Roman, J. **High-performance finite-element simulations of seismic wave propagation in 3D inelastic media.** In *SIAM 5h International workshop on Parallel Matrix and Application, PMAA'08, June 20-22*, (Neuchâtel, Suisse, 2008).