Accélération matérielle pour l'imagerie sismique : modélisation, migration et interprétation

Rached Abdelkhalek

N° d'ordre : 5028

Accélération matérielle pour l'imagerie sismique : modélisation, migration et interprétation.

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Rached ABDELKHALEK**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Soutenue le : 20 décembre 2013

Devant la commission d'examen composée de :

François Bodin ...	Professeur des Universités, IRISA	Rapporteur
Henri Calandra .	Expert imagerie sismique et HPC, Total	Co-encadrant
Olivier Coulaud .	Directeur de recherche, Inria	Co-directeur de thèse
Luc Giraud	Directeur de recherche, Inria	Examinateur
Stéphane Lantéri	Directeur de recherche, Inria	Rapporteur
Guillaume Latu ..	Ingénieur de recherche, CEA	Co-directeur de thèse
Jean Roman	Professeur des Universités, Inria & IPB	Directeur de thèse

Remerciements

Je tiens tout d'abord à remercier le directeur de cette thèse, Jean Roman pour son soutien continu et ses multiples relectures méticuleuses.

Je remercie également Henri Calandra, expert en imagerie sismique et calcul haute performance chez Total, pour m'avoir ouvert les portes de cette entreprise et initié aux géosciences.

Je remercie François Bodin et Stéphane Lantéri d'avoir accepté d'être les rapporteurs de ce travail. Je remercie également Luc Giraud, d'avoir accepté de participer au jury.

A tous mes amis et à tous mes proches, un grand merci !

Table des matières

Introduction

Ce manuscrit décrit le travail réalisé dans le cadre d'une thèse CIFRE fruit d'une collaboration pluridisciplinaire (physique, mathématiques appliquées et informatique) entre TOTAL et le projet HiePACS [1] de l'INRIA Bordeaux afin de mettre en commun leurs compétences pour l'utilisation optimisée de technologies accélératrices de calcul pour l'imagerie sismique profondeur.

La donnée sismique depuis sa conception (modélisation d'acquisitions sismiques), dans sa phase de traitement (prétraitement et migration) et jusqu'à son exploitation pour en extraire les informations géologiques pertinentes nécessaires à l'identification et l'exploitation optimale des réservoirs d'hydrocarbures (interprétation), génère un volume important de calculs. Lors de la phase d'imagerie, ce volume est d'autant plus important que les différentes simulations mises en jeu se veulent fidèles à la physique du sous sol. Une puissance de calcul importante est donc nécessaire pour réduire le temps, et donc le coût, des études en imagerie sismique et pour améliorer le résultat final de ces études en reproduisant plus fidèlement les phénomènes physiques mis en jeu et en considérant de plus larges plages de fréquences. Lors de la phase d'interprétation, le calcul d'attributs sismiques (type : cohérence, lissage, analyse spectrale, etc.) offre une aide de choix à l'interprétateur. Ces calculs se font usuellement selon un cycle itératif pour sélectionner les paramètres les plus adaptés. Ce cycle est rendu fastidieux par la complexité et donc le temps des calculs. L'exploitation optimale des ressources de calcul disponibles dans la station d'interprétation est nécessaire pour raccourcir ce cycle ainsi que pour la mise en œuvre d'algorithmes de traitements plus performants.

Les technologies accélératrices permettent de déléguer certains types de calculs à des unités puissantes (GPGPU, FPGA, MIC) dans le cadre de plateformes hétérogènes en alternative au CPU utilisé habituellement. La puissance de calcul accessible par ce biais dépasse de plusieurs ordres de grandeur ce que peuvent proposer les architectures généralistes utilisées traditionnellement en calcul hautes performances. Ces nouvelles architectures sont une alternative très intéressante pour augmenter la puissance de calcul sans augmenter pour autant la puissance électrique consommée et thermique dissipée. Néanmoins, les contraintes d'utilisation font qu'à l'heure actuelle ces nouveaux types de calculateurs sont difficiles à programmer et à optimiser dans le cadre du calcul scientifique et conduisent à des codes dédiés à une architecture particulière.

1. http ://www.inria.fr/equipes/hiepacs

Les simulations reposant sur la résolution de l'équation des ondes en 2D ou 3D discrétisée sur des grilles (utilisées pour la modélisation et la migration sismiques), ainsi que les algorithmes de traitement d'images (utilisés lors de l'interprétation des données sismiques) sont des candidats potentiels pour une implémentation très efficace sur ces nouvelles architectures.

Dans cette thèse, nous proposons une étude de l'apport, des contraintes ainsi que des limites éventuelles de ces technologies accélératrices pour l'imagerie et l'interprétation sismiques. Dans la première partie du manuscrit, après une brève introduction à l'imagerie sismique dans le premier chapitre, nous passons en revue dans le deuxième chapitre les algorithmes utilisés dans ce cadre pour mettre en exergue la complexité de ces algorithmes et les besoins en puissance de calcul qui en découlent. Nous exposons ensuite dans le chapitre 3 les différentes technologies matérielles et logicielles actuelles permettant de répondre à ces besoins. Dans la deuxième partie de ce manuscrit, nous étudions l'impact de l'utilisation des technologies accélératrices en imagerie sismique (chapitre 4) et dans le cadre de l'interprétation sismique (chapitre 5).

Dans le chapitre 4, nous proposons ainsi diverses implémentations d'algorithmes utilisés en imagerie sismique reposant sur la simulation de la propagation des ondes sismiques dans le sous-sol via une discrétisation de l'équation d'onde en 2D et en 3D et sa résolution par différences finies. Nous analysons le comportement de ces implémentations sur divers types d'accélérateurs. Nous montrons qu'une prise en compte fine des ressources disponibles au niveau de l'unité de calcul (bandes passantes, capacité mémoire, organisation des données en mémoire et motifs d'accès à ses différents niveaux) est nécessaire pour tirer partie de chaque type d'architecture et au-delà de cela, de chaque génération d'une architecture donnée. De plus, les communications entre l'accélérateur et la machine hôte ont un coût qu'il est nécessaire de limiter pour ne pas pénaliser les temps de calcul. Nous proposons différentes techniques pour minimiser ces coûts et analysons leur comportement. Ces implémentations reposent sur une décomposition du domaine de simulation global, qui peut être de taille importante, en sous-domaines ce qui induit également des communications entre nœuds dans le cadre de systèmes à mémoire distribuée.

Dans le chapitre 5, une étude similaire est proposée pour le calcul d'attributs sismiques. Contrairement aux algorithmes d'imagerie sismique, ce sont les ressources de la station de travail locale qui sont exploitées pour tendre vers un calcul interactif des attributs facilitant ainsi la tâche de l'interprétateur. Une implémentation performante de la transposition de cubes sismiques 3D est proposée. Elle sert de base aux algorithmes étudiés par la suite. Est étudiée ensuite une première classe d'algorithmes basés sur le calcul de la similarité entre traces sismiques voisines : cohérence, calcul de pendage ainsi qu'un algorithme innovant mis au point lors de cette étude. Les calculs sur accélérateur graphique du lissage gaussien par filtres FIR et IIR sont comparés. Des facteurs d'accélération variant entre 8 et 160 par rapport aux processeurs classiques sont reportés.

Ces travaux ouvrent la voie à une intégration complète et systématique des accélérateurs de calcul tout le long du cycle de traitement des données sismiques et ce d'autant plus que nous avons démontré que cette intégration ne se fait pas aux dépends de la fiabilité et de la maintenabilité du code existant.

Cette étude pourrait enfin avoir comme prolongement naturel l'utilisation de machines hybrides à mémoire distribuée, similaires à celles utilisées pour l'imagerie sismique accélérée sur GPU, pour le calcul d'attributs sismiques et le rendu graphique sur de gros volumes de données.

La thèse s'est déroulée essentiellement au sein du CSTJF (Centre Scientifique et Technique Jean Féger) de Total à Pau avec des séjours de travail à l'INRIA de Bordeaux et au HGRG (Houston Geophysical Research Group), centre de recherche de Total à Houston.

Première partie

Etat de l'art

Chapitre 1

Introduction à l'Imagerie Sismique

Sommaire

L'imagerie sismique est un processus visant à obtenir une image des propriétés du sous sol à partir d'enregistrements faits en surface. Les techniques d'imagerie sismique peuvent aussi bien s'appliquer à des études environnementales des premières strates de l'écorce terrestre, qu'à l'étude des séismes à l'échelle du globe en sismologie.

En exploration pétrolière, l'imagerie sismique est la principale méthode utilisée pour découvrir de nouveaux gisements. En effet, on vise dans ce domaine à localiser les pièges stratigraphiques où les hydrocarbures ont pu s'accumuler et à partir desquels ils peuvent être extraits avec un coût raisonnable. Les campagnes d'acquisition sismique et les images sismiques qui en découlent permettent de localiser ces pièges et de les caractériser. L'effort important déployé par l'industrie pétrolière pour l'imagerie sismique est largement justifié par le coût élevé des forages d'exploration (coût pouvant atteindre une centaine de millions d'euro pour les zones complexes en *offshore* profond par exemple). L'imagerie sismique intervient également après la mise en production d'un gisement pour suivre son évolution et optimiser son exploitation.

Pour aboutir à cette image sismique (*i.e.* à une représentation spatiale des discontinuités des propriétés physiques du sous sol) et l'interpréter, deux étapes sont nécessaires : l'acquisition des données sismiques et leur traitement [1].

Afin de mieux comprendre l'intérêt de l'accélération des calculs dans le cadre de l'exploration et de la production pétrolière, une vision globale des calculs qui y sont mis en œuvre est nécessaire.

Nous présenterons au cours de ce chapitre certaines des techniques employées pour rechercher et exploiter les gisements d'hydrocarbures, ce qui permettra d'introduire le contexte et les objectifs de la thèse.

1.1 Acquisition sismique

Pour explorer une zone géographique par imagerie sismique, la première étape consiste à acquérir les données sismiques. Il s'agit d'exciter le sous-sol en utilisant une source placée en surface (par exemple un canon à air pour une acquisition marine ou un camion vibrateur en terrestre). Les particules en contact direct avec le dispositif source sont alors soumises à un mouvement vibratoire. Elles entraînent dans leur mouvement les particules avoisinantes. L'onde sismique émise en surface se propage ainsi de proche en proche dans le sous-sol. Elle est réfléchie et/ou réfractée au niveau des discontinuités des propriétés physiques du sous sol (contrastes de densités et/ou de vitesses). Des capteurs placés en surface enregistrent au cours du temps la réponse sismique du milieu à cet ébranlement. Dans le cas d'une acquisition marine ces capteurs sont appelés hydrophones : ils enregistrent une variation de pression. En terrestre on parle de géophones qui enregistrent une ou plusieurs composantes du champ de vecteurs vitesse des particules auxquelles ils sont rattachés. Les séries de données enregistrées au cours du temps par les récepteurs sont appelées traces sismiques. L'ensemble des traces enregistrées pour un

1. Cette vision linéaire est de plus en plus obsolète. En effet aujourd'hui l'acquisition, les traitements et l'interprétation interagissent de plus en plus de façon itérative dans le cadre d'une étude transverse.

même point de tir forme un enregistrement sismique ou sismogramme. Une acquisition sismique conventionnelle en 3D peut couvrir de quelques centaines à quelques milliers de kilomètres carrés et nécessiter des dizaines de milliers de points de tir.

Lors de la conception d'une acquisition sismique, les différents paramètres sont fixés en fonction des objectifs géophysiques de l'acquisition (objectif à imager, sa profondeur, son aspect structural, la résolution désirée, etc.) et des contraintes opérationnelles (coût, durée, accessibilité, topographie du terrain, etc.). Sont ainsi définis, entre autres, la surface à couvrir, la redondance des données mais également le type, le nombre ainsi que la disposition géométrique des sources et des récepteurs (voir chapitre 4 pour des exemples).

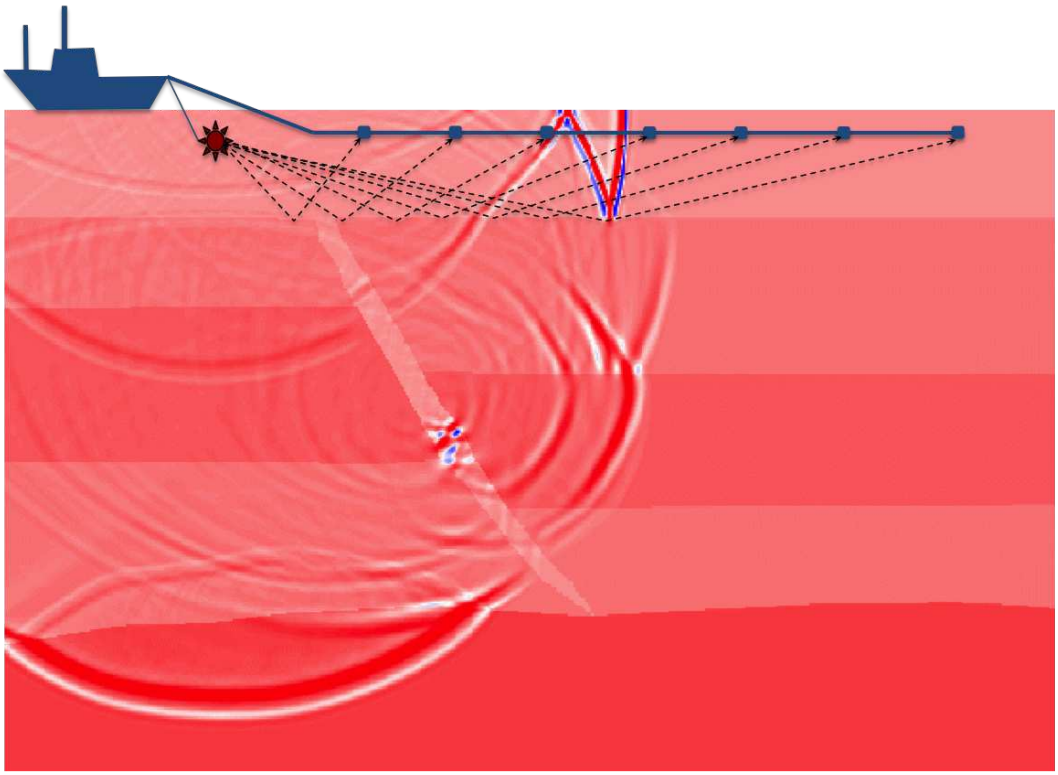


FIGURE 1.1 – Exemple de dispositif d'acquisition sismique. L'onde sismique émise en surface se propage dans le sous-sol. Elle est réfléchie et/ou réfractée au niveau des discontinuités des propriétés physiques du sous sol. Des capteurs placés en surface enregistrent au cours du temps la réponse sismique du milieu à cet ébranlement.

1.2 Propagation des ondes sismiques

L'effet des ébranlements émis en surface se propage sous la forme d'ondes de surface et d'ondes de volume. Les ondes de surface se propagent au niveau de l'interface sol-air (ondes de Love, de Rayleigh, etc.). L'amplitude de ces ondes - relativement importante au niveau de la surface - décroît de façon exponentielle avec la profondeur. Les ondes de volume se propagent

dans le sous sol. Selon l'orientation du mouvement des particules par rapport à la direction de propagation des ondes, on distingue les ondes de compression (mouvement parallèle à la direction de propagation) et les ondes de cisaillement (mouvement perpendiculaire à la direction de propagation). Les ondes de compression sont appelées ondes P ou primaires car plus rapides que les ondes de cisaillement, appelées quant à elles ondes S ou secondaires.

Trois paramètres définis en tout point de l'espace suffisent pour décrire la propagation des ondes dans un milieu élastique isotrope : les coefficients de Lamé λ, μ et la masse volumique ρ du milieu, appelée aussi par abus de langage densité. D'autres paramètres peuvent être considérés. Ils sont habituellement choisis parmi les paramètres suivants : le coefficient de Poisson ν, le module d'incompressibilité K, la vitesse et l'impédance des ondes S (V_S et Z_S) et la vitesse et l'impédance des ondes P (V_P et Z_P). Le tableau 1.1 donne à titre indicatif certaines relations entre ces différents paramètres.

$\nu = \frac{\lambda}{2(\lambda+\mu)}$ $K = \lambda + \frac{2}{3}\mu$	$V_P = \sqrt{\frac{\lambda+2\mu}{\rho}}$ $V_S = \sqrt{\frac{\mu}{\rho}}$	$Z_P = \rho V_p$ $Z_S = \rho V_S$

TABLE 1.1 – Relations entre les paramètres caractérisant les milieux de propagation des ondes sismiques.

Dans un milieu acoustique (fluide) seules les ondes P se propagent. Dans ce cas $\mu = 0$, $V_S = 0$ (V_P est alors appelée vitesse acoustique) et $\nu = 0.5$. Le tableau 1.2 donne à titre d'exemple les vitesses et les densités de certains milieux.

Lorsqu'elles se propagent, ces ondes sont gouvernées par les mêmes lois que les ondes électromagnétiques en optique géométrique (Principes de Huygens et de Fermat, loi de Snell-Descartes). Elles subissent donc des phénomènes analogues à ceux de la lumière au niveau des discontinuités des propriétés du sous sol. Les deux phénomènes les plus intéressants pour le géophysicien sont la réflexion et la réfraction.

Au niveau de l'interface entre deux milieux d'impédances différentes, une partie de l'énergie d'une onde incidente est transmise, une autre est réfléchie. Le coefficient de réflexion R est le ratio entre l'amplitude de l'onde incidente et celle de l'onde réfléchie. Il dépend du contraste d'impédance. Pour les ondes P et S et dans le cas particulier d'une incidence normale à la surface de réflexion, on a : $R_P = \frac{Z_{P2}-Z_{P1}}{Z_{P2}+Z_{P1}}$ et $R_S = \frac{Z_{S2}-Z_{S1}}{Z_{S2}+Z_{S1}}$. L'énergie transmise permet d'illuminer les interfaces plus profondes. Au niveau de l'interface entre deux milieux de vitesses différentes,

Milieu	V_P $(m\ s^{-1})$	V_S $(m\ s^{-1})$	ρ $(g\ cm^{-3})$
Eau	1450-1500	0	1
Argiles	1100-2500	200-800	2.0-2.4
Calcaire	3500-6000	2000-3300	2.4-2.7
Sel	4500-5500	2500-3100	2.1-2.3

TABLE 1.2 – Ordre de grandeur des vitesses de propagation des ondes P et S et des masses volumiques de différents milieux (d'après [81]).

l'onde est réfractée. Selon le contraste de vitesse entre les deux milieux, il peut exister un angle d'incidence critique à partir duquel l'onde réfractée se propage le long de l'interface et remonte vers la surface sous la forme d'une onde plane.

1.3 Modélisation sismique

La simulation numérique de la propagation des ondes sismiques, communément appelée modélisation sismique [28] ou encore résolution du problème direct, offre une aide importante à la conception d'acquisitions sismiques et à leur paramétrage. Cette simulation vise à générer l'ensemble des sismogrammes (appelés de fait sismogrammes synthétiques) qu'un dispositif source récepteur devrait théoriquement enregistrer en supposant une certaine structure du sous-sol.

La modélisation sismique est également un outil important pour l'interprétation des données sismiques et joue un rôle essentiel dans certains algorithmes de migration et d'inversion comme exposé dans la suite de ce chapitre.

Plusieurs techniques ont été développées pour la modélisation sismique. Ces techniques sont centrées sur la résolution de l'équation d'onde. Selon l'approche adoptée pour résoudre cette équation, elles peuvent être classées en deux grandes familles [28][139, p. 1779] : les méthodes directes basées sur la résolution de l'équation d'onde sur un domaine discrétisé en temps et en espace, et les méthodes intégrales.

Les méthodes directes peuvent être implicites en temps (une relation implicite est établie entre les champs d'onde à différents pas de temps) ou explicites (l'état du champ d'onde à un instant donné est explicitement calculé à partir de ses états aux instants précédents). L'avantage de la résolution explicite est la possibilité d'avoir des instantanés du champ d'onde (ou *snapshots*) à un instant précis. Dans le cadre de ces méthodes, diverses approximations peuvent être considérées pour décrire la propagation des ondes sismiques. L'équation d'onde acoustique qui décrit uniquement la propagation des ondes P est adaptée aux modélisations structurales où les temps de trajet (la cinématique) des signaux sont plus importants que leur amplitude. L'équation d'onde élastique prenant en compte aussi bien les ondes P que S est adaptée aux études stratigraphiques détaillées où les amplitudes aussi bien que les temps de trajet sont pris en compte. Une approximation *one-way* de l'équation d'onde permet d'accélérer les calculs mais ne permet pas de prendre en compte les multiples (ondes ayant subi plusieurs réflexions). L'équation d'onde complète (*two-way*) permet de modéliser les multiples.

Nous détaillerons dans le chapitre suivant l'utilisation de l'approche directe explicite basée sur l'équation d'onde acoustique complète.

1.4 Imagerie sismique par réflexion, approche conventionnelle

L'imagerie sismique par réflexion permet d'atteindre deux objectifs complémentaires [116, p. 101] :

– un premier objectif est l'imagerie des structures géométriques en positionnant correctement les horizons qui représentent les évènements enregistrés en temps à la surface et possédant une certaine continuité spatiale (cf 1.6) ;
– un deuxième objectif est l'imagerie quantitative des propriétés élastiques des milieux. Toute une branche des géosciences, la pétrophysique, s'intéresse au lien entre les paramètres élastiques et les paramètres pétrophysiques tels que la porosité et la perméabilité [65, pp. 46-47].

Le processus d'imagerie sismique se déroule usuellement en 3 étapes. Il commence ainsi par le prétraitement des données. Ensuite, compte tenu du contenu fréquentiel du signal enregistré [66], deux étapes sont nécessaires. La première étape consiste à construire le modèle de vitesse basse fréquence, aussi appelé macro-modèle, qui servira de support pour la migration. C'est la migration qui permettra enfin d'obtenir l'image finale haute fréquence qui sera utilisée pour l'interprétation.

1.4.1 Prétraitement des données

Une phase de prétraitement des données est nécessaire pour adapter les données aux traitements qui suivent. Cette phase vise principalement à éliminer du signal enregistré les informations non exploitables ainsi que celles qui risquent de détériorer le résultat final. Il s'agira par exemple d'éliminer les multiples, les arrivées directes et les ondes de surface, ainsi que d'augmenter le ratio signal/bruit. Par ailleurs, on procède généralement à la réorganisation des données et à la prise en compte de la géométrie du terrain d'acquisition.

1.4.2 Construction du modèle de vitesse initial

Plusieurs techniques ont été développées pour construire le modèle de vitesse initial qui servira de support pour la migration. Ces techniques sont basées sur l'utilisation de la cinématique des évènements enregistrés. Dans des milieux simples constitués de couches sédimentaires horizontales, l'approche basée sur l'analyse des vitesses de sommation définissant la correction de courbure d'indicatrice (*Normal Moveout* ou NMO) permet d'obtenir une première idée de la structure du sous-sol [139]. Pour des milieux plus complexes, c'est la tomographie par temps de trajet (pointage manuel des évènements) [19], la stéréotomographie, ou encore l'analyse de vitesse par migration (MVA) qui permettent de construire le modèle de vitesse initial.

1.4.3 Migration

La migration a pour but de reconstruire une image du sous sol à partir des évènements observés en surface en temps en les replaçant correctement à leurs positions. A ce titre, elle peut être considérée comme le problème inverse de la modélisation sismique. Tout réflecteur du sous-sol peut être considéré comme une infinité de points diffractants. Chaque point diffractant donne naissance à une hyperbole de diffraction en sismique 2D ou une hyperboloïde de diffraction

en sismique 3D. Le processus de migration permet de refocaliser chacune de ces hyperboles ou hyperboloïdes en son apex.

La migration tout comme la modélisation sismique est basée sur la solution approchée de l'équation d'onde. Tout comme pour la modélisation sismique, et selon la méthode utilisée pour modéliser la propagation de la source et des récepteurs, deux grandes familles de méthodes de migration se distinguent [18] : les méthodes intégrales par tracé de rais et celles basées sur la continuation du champ d'ondes.

1. La migration basée sur les méthodes intégrales par tracé de rayons (migration de Kirchhoff principalement [121]) a longtemps été la principale méthode utilisée -surtout en 3D- eu égard à son coût calculatoire réduit par rapport à d'autres méthodes. Ses principaux inconvénients sont la complexité limitée prise en compte par le tracé de rayons ainsi que, dans sa forme la plus simple, la non considération des multiples.

2. Les méthodes basées sur la résolution de l'équation d'onde, introduite par Claerbout dès 1970 [35], ont connu un regain d'attention ces dernières années, les progrès réalisés dans les domaines de la simulation numérique et de l'informatique les ayant rendues exploitables en pratique pour des modèles réalistes. Les premières implémentations basées sur l'équation d'onde *one-way* (*downward-continuation migration algorithms*) ont été réalisées dans le domaine fréquentiel. De nos jours, la migration à temps inverse (*Reverse Time Migration* ou RTM) [15, 37] est la méthode de choix car basée sur la résolution de l'équation d'onde complète (*two-way*). Elle permet donc de prendre en compte des trajectoires d'ondes dans des milieux complexes (ondes multiples, tournantes, prismatiques, etc.). L'utilisation de l'équation d'onde complète implique une complexité plus importante que l'approche *one-way* (complexité en temps en $\mathcal{O}(n^4)$ contre $\mathcal{O}(n^3)$ pour la *one-way*) et requiert donc des moyens de calcul plus puissants. L'évolution des moyens de calcul a conduit à un regain d'intérêt ces dernières années envers la RTM. La mise en œuvre reste tout de même complexe et requiert un effort de programmation particulier [48].

Différentes variantes existent au sein de ces familles de méthodes selon : le domaine de représentation des données (migration en temps ou en profondeur) ; les données utilisées en entrée (migration avant sommation, *prestack*, ou après sommation, *poststack*) ; l'approximation de l'équation d'onde adoptée, etc.

Le développement de ces méthodes a suivi l'évolution des moyens de calcul et la capacité à faire des approximations de plus en plus réalistes de l'équation d'onde et à traiter des quantités importantes de données. Les premières méthodes de migration utilisées consistaient à traiter les données en temps après sommation. La sommation consiste à additionner des traces issues de points de tir différents de façon à obtenir des enregistrements semblables à ceux qu'on aurait obtenus en plaçant sources et récepteurs à des positions confondues (*zero-offset*). Ceci a pour effet de réduire la taille des données à traiter et d'augmenter le ratio signal/bruit. Cependant, la qualité de l'image finale se trouve ainsi réduite, vue la perte d'information qu'implique la sommation. La migration en temps après sommation a été longtemps utilisée à cause de son effi-

cacité et de la simplicité de sa mise en œuvre. Elle est encore utilisée pour des zones géologiques simples où la vitesse varie lentement. Pour les milieux plus complexes et à fortes variations de vitesses, seule la migration en profondeur permet d'obtenir une description géométrique correcte du sous-sol et de replacer les réflecteurs à leurs positions exactes [7]. Cependant, une connaissance du modèle de vitesse est requise (variations verticales et latérales) et une plus grande complexité de calcul est mise en jeux.

L'évolution des moyens de calcul et de stockage a permis l'utilisation à plus large échelle de la migration avant sommation qui augmente d'un ordre de grandeur la taille des données à traiter. Ce sont ces méthodes qui permettent aujourd'hui d'obtenir les meilleurs résultats et d'étendre l'exploration pétrolière à des zones considérées jusqu'à récemment comme trop risquées.

Dans le chapitre suivant on présentera l'équation d'onde complète (*two-way*) et on décrira son utilisation pour la modélisation et la RTM.

1.5 Inversion de formes d'ondes

L'approche conventionnelle décrite dans la section précédente se fait en deux temps : construction du macro modèle de vitesse basse fréquence puis migration. En tant que telle, elle souffre des limitations et des approximations faites à chacune de ces étapes.

Depuis quelques années, on assiste à la popularisation d'une approche globale favorisée par l'évolution des puissances de calcul disponibles et le développement de nouvelles techniques d'acquisition sismique à redondance élevée et à large couverture angulaire (*multifold, wide azimuth acquisitions*). Cette approche est celle de l'inversion de formes d'ondes complètes (*Full Waveform Inversion* ou *FWI*) [128]. Il s'agit de remplacer le processus d'imagerie par la résolution d'un problème inverse où l'on cherche à adapter itérativement le modèle de vitesse pour minimiser les différences entre les données observées (enregistrées au niveau des récepteurs) et les données modélisées. Une revue des méthodes utilisées pour la FWI est proposée dans [134].

1.6 Interprétation des données sismiques

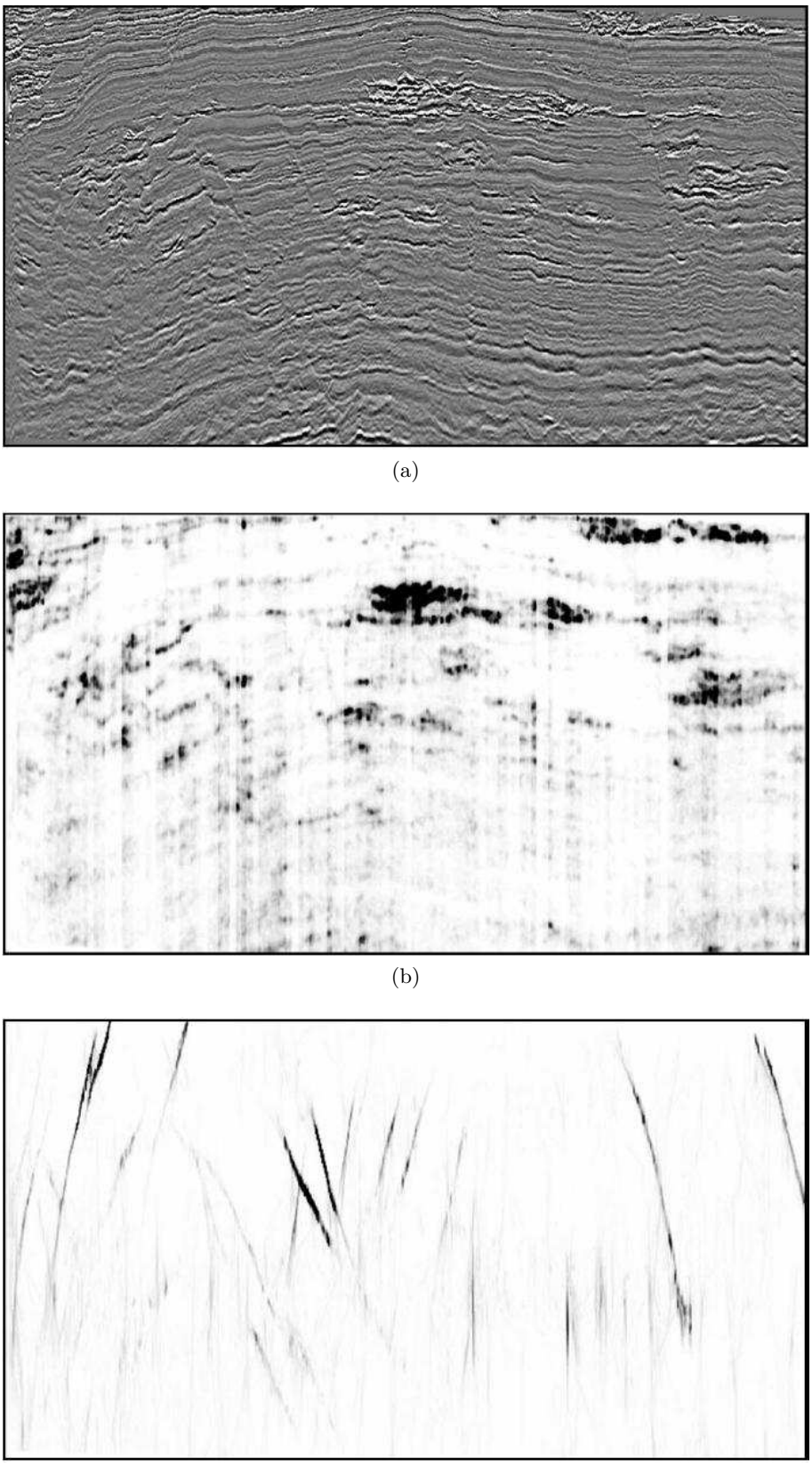
Les données sismiques produites par les traitements précédemment décrits, de concert avec les données mesurées le long de puits d'exploration et toute autre information pertinente sur le prospect, sont analysées et interprétées pour aboutir à une description spatiale du sous-sol sous la forme de modèles lithologiques, structuraux et stratigraphiques. Géologues, géophysiciens et ingénieurs réservoir sont souvent amenés à collaborer étroitement pour affiner et enrichir cette interprétation.

Ces données se présentent en 3D sous la forme de cubes de traces sismiques, dont les dimensions sont reliées à la géométrie de l'acquisition sismique, communément appelées *inline* dans la direction de l'acquisition et *crossline* dans la direction perpendiculaire. La troisième dimension quant à elle correspond au temps t du trajet source-réflecteur-récepteur (*Two Way Time*) ou à la profondeur z et ce selon le domaine dans lequel sont représentées les données.

On fait souvent la distinction entre interprétation structurale et stratigraphique et interprétation lithologique. L'interprétation structurale et stratigraphique vise à décrire la disposition spatiale des roches qui est en relation étroite avec les processus sédimentologiques qui ont abouti à la formation de ces roches. Lors de cette étape, l'interprétateur analyse les changements des amplitudes des signaux représentés par les traces sismiques qui correspondent souvent à des changements de propriétés géologiques [62]. L'interprétateur veillera à repérer en particulier certaines formations géologiques spécifiques. Une surface correspondant à une amplitude particulière qui s'étend latéralement sur plusieurs traces sismiques peut être interprétée comme une réflexion se produisant à l'interface séparant deux couches de roches de natures différentes ou strates. Il s'agit alors d'un **horizon**. Des discontinuités linéaires le long de ces horizons qui s'étendent verticalement dans les données sismiques peuvent être assimilées à des **failles** qui correspondent à des "cassures" dans les strates de roches avec déplacements relatifs des parties séparées. Ces déplacements peuvent être horizontaux ou verticaux. Selon la nature des roches présentes des deux côtés de la faille, ces formations peuvent constituer d'excellents pièges pour les hydrocarbures. Des **corps** d'amplitudes sismiques similaires peuvent également apparaître dans l'image sismique. Il peut s'agir de zones du sous-sol aux propriétés identiques telles que les dômes de sel, les chenaux ou encore des réservoirs d'hydrocarbures.

L'interprétation lithologique vise quant à elle à décrire à partir des données sismiques la nature et les caractéristiques des roches formant les couches géologiques. Pour mieux identifier ces formations, l'interprétateur s'aidera en particulier des attributs sismiques qui caractérisent chacune d'elles. Dans [32] un attribut sismique est défini comme étant une mesure quantitative d'une caractéristique sismique d'intérêt. On retrouve dans cette même référence une perspective historique de l'évolution des attributs sismiques en corrélation avec l'évolution des moyens informatiques disponibles. La panoplie d'attributs sismiques offerte aux interprétateurs s'étant significativement enrichie au fil des années, plusieurs tentatives de classification de ces attributs ont été proposées[23, 84, 126, 127]. Taner et al. [126] proposent par exemple de classer les attributs en deux catégories générales : géométriques et physiques. Les attributs géométriques visent à améliorer la visibilité des propriétés géométriques des données sismiques ; ils comprennent le *pendage*, l'*azimut* et la *continuité*. Les attributs physiques correspondent à la propagation de l'onde sismique et sont donc liés aux paramètres physiques du sous-sol ; ils incluent l'*amplitude*, la *phase* et la *fréquence*. A titre d'exemple, la figure 1.2 illustre l'intérêt de l'utilisation de deux attributs sismiques calculés à partir de données sismiques migrées pour l'identification des failles et des zones de chaos.

On verra dans le chapitre 5 que ces calculs peuvent être en effet relativement coûteux, eu égard à la nature interactive de l'exercice d'interprétation. On étudiera également l'impact que pourra avoir l'utilisation des GPU sur ces calculs.



(a)

(b)

(c)

FIGURE 1.2 – Attributs sismiques structuraux. (a) Section de sismique réflexion migrée après sommation (b) Attribut sismique mettant en évidence les zones de chaos de la sismique. (c) Attribut mettant en évidence les failles (extrait de [59]).

Chapitre 2

Modélisation et Imagerie Sismiques par Equation d'Onde

Sommaire

Dans ce chapitre sont introduites les équations régissant la propagation des ondes dans un milieu élastique classique puis l'approximation utilisée dans le cas où le milieu est considéré comme acoustique. Les méthodes de discrétisation des équations aux dérivées partielles seront ensuite décrites.

2.1 Équation d'onde acoustique

En considérant la loi de Hooke et la deuxième loi du mouvement de Newton pour un milieu élastique classique (élastique isotrope linéaire soumis à de petites déformations), on aboutit respectivement aux systèmes d'équations (2.1) et (2.2) [6, 9] :

$$\begin{aligned}
\frac{\partial \sigma_{xx}(x,y,z,t)}{\partial t} &= (\lambda(x,y,z)+2\mu(x,y,z))\frac{\partial v_x(x,y,z,t)}{\partial x} + \lambda(x,y,z)\{\frac{\partial v_y(x,y,z,t)}{\partial y} + \frac{\partial v_z(x,y,z,t)}{\partial z}\} \\
\frac{\partial \sigma_{yy}(x,y,z,t)}{\partial t} &= (\lambda(x,y,z)+2\mu(x,y,z))\frac{\partial v_y(x,y,z,t)}{\partial y} + \lambda(x,y,z)\{\frac{\partial v_x(x,y,z,t)}{\partial x} + \frac{\partial v_z(x,y,z,t)}{\partial z}\} \\
\frac{\partial \sigma_{zz}(x,y,z,t)}{\partial t} &= (\lambda(x,y,z)+2\mu(x,y,z))\frac{\partial v_z(x,y,z,t)}{\partial z} + \lambda(x,y,z)\{\frac{\partial v_x(x,y,z,t)}{\partial x} + \frac{\partial v_y(x,y,z,t)}{\partial y}\} \\
\frac{\partial \sigma_{xy}(x,y,z,t)}{\partial t} &= \mu(x,y,z)\{\frac{\partial v_x(x,y,z,t)}{\partial y} + \frac{\partial v_y(x,y,z,t)}{\partial x}\} \\
\frac{\partial \sigma_{xz}(x,y,z,t)}{\partial t} &= \mu(x,y,z)\{\frac{\partial v_x(x,y,z,t)}{\partial z} + \frac{\partial v_z(x,y,z,t)}{\partial x}\} \\
\frac{\partial \sigma_{yz}(x,y,z,t)}{\partial t} &= \mu(x,y,z)\{\frac{\partial v_y(x,y,z,t)}{\partial z} + \frac{\partial v_z(x,y,z,t)}{\partial y}\}
\end{aligned} \tag{2.1}$$

$$\begin{aligned}
\frac{\partial v_x(x,y,z,t)}{\partial t} &= \frac{1}{\rho(x,y,z)}\{\frac{\partial \sigma_{xx}(x,y,z,t)}{\partial x} + \frac{\partial \sigma_{xy}(x,y,z,t)}{\partial y} + \frac{\partial \sigma_{xz}(x,y,z,t)}{\partial z}\} \\
\frac{\partial v_y(x,y,z,t)}{\partial t} &= \frac{1}{\rho(x,y,z)}\{\frac{\partial \sigma_{xy}(x,y,z,t)}{\partial x} + \frac{\partial \sigma_{yy}(x,y,z,t)}{\partial y} + \frac{\partial \sigma_{yz}(x,y,z,t)}{\partial z}\} \\
\frac{\partial v_z(x,y,z,t)}{\partial t} &= \frac{1}{\rho(x,y,z)}\{\frac{\partial \sigma_{xz}(x,y,z,t)}{\partial x} + \frac{\partial \sigma_{yz}(x,y,z,t)}{\partial y} + \frac{\partial \sigma_{zz}(x,y,z,t)}{\partial z}\}
\end{aligned} \tag{2.2}$$

où $v_x(x,y,z,t)$, $v_y(x,y,z,t)$ et $v_z(x,y,z,t)$ correspondent aux composantes du champ vitesse des particules ; $\sigma_{ij}(x,y,z,t)$ avec $i,j = x,y$ ou z sont les contraintes ou forces exercées à l'instant t sur une particule (par exemple σ_{xz} correspond à la contrainte tangentielle exercée selon la direction z sur la face orthogonale à la direction x ; par symétrie σ_{xz}=σ_{zx}) ; $\rho(x,y,z)$ la densité ; $\lambda(x,y,z)$ et le module de cisaillement $\mu(x,y,z)$ sont les coefficients de Lamé.

On aboutit à l'approximation acoustique de l'équation d'onde en se plaçant dans un milieu fluide non visqueux. Dans un tel milieux, le module de cisaillement est nul ($\mu(x,y,z) = 0$). Les systèmes (2.1) et (2.2) sont alors réduits à (2.3) :

$$\begin{aligned}
\frac{\partial \sigma_{xx}(x,y,z,t)}{\partial t} &= \lambda(x,y,z)\{\frac{\partial v_x(x,y,z,t)}{\partial x} + \frac{\partial v_y(x,y,z,t)}{\partial y} + \frac{\partial v_z(x,y,z,t)}{\partial z}\} \\
\frac{\partial \sigma_{yy}(x,y,z,t)}{\partial t} &= \lambda(x,y,z)\{\frac{\partial v_y(x,y,z,t)}{\partial y} + \frac{\partial v_x(x,y,z,t)}{\partial x} + \frac{\partial v_z(x,y,z,t)}{\partial z}\} \\
\frac{\partial \sigma_{zz}(x,y,z,t)}{\partial t} &= \lambda(x,y,z)\{\frac{\partial v_z(x,y,z,t)}{\partial z} + \frac{\partial v_x(x,y,z,t)}{\partial x} + \frac{\partial v_y(x,y,z,t)}{\partial y}\} \\
\frac{\partial v_x(x,y,z,t)}{\partial t} &= \frac{1}{\rho(x,y,z)}\frac{\partial \sigma_{xx}(x,y,z,t)}{\partial x} \\
\frac{\partial v_y(x,y,z,t)}{\partial t} &= \frac{1}{\rho(x,y,z)}\frac{\partial \sigma_{yy}(x,y,z,t)}{\partial y} \\
\frac{\partial v_z(x,y,z,t)}{\partial t} &= \frac{1}{\rho(x,y,z)}\frac{\partial \sigma_{zz}(x,y,z,t)}{\partial z}.
\end{aligned} \tag{2.3}$$

Les trois premières équations de (2.3) impliquent que $\frac{\partial \sigma_{xx}(x,y,z,t)}{\partial t} = \frac{\partial \sigma_{yy}(x,y,z,t)}{\partial t} = \frac{\partial \sigma_{zz}(x,y,z,t)}{\partial t}$.

En combinant ces équations, on obtient le système hyperbolique du premier ordre suivant :

$$\begin{aligned}
\frac{\partial p(x,y,z,t)}{\partial t} &= K(x,y,z)(\frac{\partial v_x(x,y,z,t)}{\partial x} + \frac{\partial v_y(x,y,z,t)}{\partial y} + \frac{\partial v_z(x,y,z,t)}{\partial z}) \\
\frac{\partial v_x(x,y,z,t)}{\partial t} &= \frac{1}{\rho(x,y,z)}\frac{\partial p(x,y,z,t)}{\partial x} \\
\frac{\partial v_y(x,y,z,t)}{\partial t} &= \frac{1}{\rho(x,y,z)}\frac{\partial p(x,y,z,t)}{\partial y} \\
\frac{\partial v_z(x,y,z,t)}{\partial t} &= \frac{1}{\rho(x,y,z)}\frac{\partial p(x,y,z,t)}{\partial z}
\end{aligned} \tag{2.4}$$

où $p(x,y,z,t)$ est le champ de pression [1] défini par :

$$p(x,y,z,t) = \frac{\sigma_{xx}(x,y,z,t) + \sigma_{yy}(x,y,z,t) + \sigma_{zz}(x,y,z,t)}{3} \tag{2.5}$$

et $K(x,y,z)$ est le module d'incompressibilité. En introduisant le terme source $s(t)$ à la position (x_s, y_s, z_s), le système (2.4) s'écrit sous la forme de l'équation d'onde d'ordre 2 suivante :

$$\frac{1}{K(x,y,z)}\frac{\partial^2 p(x,y,z)}{\partial t^2} - \nabla.(\frac{1}{\rho(x,y,z)}\nabla p(x,y,z,t)) = s(t)\delta(x-x_s)\delta(y-y_s)\delta(z-z_s). \tag{2.6}$$

Dans le cas où la densité est constante, et en introduisant la vitesse $c = \sqrt{\frac{K}{\rho}}$, l'équation (2.6) devient :

$$\frac{1}{c^2(x,y,z)}\frac{\partial^2 p(x,y,z,t)}{\partial t^2} - \Delta p(x,y,z,t) = s(t)\delta(x-x_s)\delta(y-y_s)\delta(z-z_s). \tag{2.7}$$

1. on désigne par pression, par abus de langage, ce qui représente plus rigoureusement une surpression autour de la pression d'équilibre.

L'hypothèse de la densité constante a un intérêt limité en modélisation, mais elle est utilisée pour la migration.

Remarque 1 Il est facile d'établir à partir du système (2.4) que les composantes du champ vitesse et celles du champ déplacement des particules obéissent aux mêmes types d'équations différentielles que la pression. Ces équations peuvent donc être utilisées indifféremment pour ces grandeurs. On remplacera donc par la suite $p(x, y, z, t)$ par $u(x, y, z, t)$.

2.2 Modélisation directe explicite par équation d'onde acoustique

Nous allons nous intéresser à la modélisation en utilisant les méthodes directes basées sur la résolution numérique des équations (2.8) pour le cas général où la densité du milieu varie dans l'espace et (2.9) pour le cas où le milieu est supposé de densité constante :

$$\frac{1}{K(x,y,z)}\frac{\partial^2 u(x,y,z)}{\partial t^2} - \nabla.(\frac{1}{\rho(x,y,z)}\nabla u(x,y,z,t)) = s(t)\delta(x-x_s)\delta(y-y_s)\delta(z-z_s) \tag{2.8}$$

$$\frac{1}{c^2(x,y,z)}\frac{\partial^2 u(x,y,z,t)}{\partial t^2} - \Delta u(x,y,z,t) = s(t)\delta(x-x_s)\delta(y-y_s)\delta(z-z_s). \tag{2.9}$$

Nous nous limitons à l'étude des ondes acoustiques sur lesquelles a principalement porté le travail présenté par la suite. Cependant il est à souligner que plusieurs des techniques présentées ici ont pu être introduites pour la formulation plus complexe du système élastodynamique avant d'être adapté au cas acoustique.

On s'intéressera plus particulièrement à l'approche explicite où le champ d'onde à un instant donné est calculé à partir des instants précédents. L'approche explicite permet la génération d'instantanés (ou *snapshots*) complets des champs d'ondes aux pas de temps désirés. Ces instantanés peuvent être utiles dans la compréhension et l'interprétation des résultats.

Pour ces méthodes, on utilise généralement une discrétisation en temps par différences finies selon un schéma centré explicite d'ordre deux, ce qui donne en considérant un pas de temps Δt et en sommant les développements de Taylor de $u(x, y, z, t)$ aux points $t + \Delta t$ et $t - \Delta t$:

$$\frac{\partial^2 u(x,y,z,t)}{\partial t^2} \approx \frac{u(x,y,z,t+\Delta t) - 2u(x,y,z,t) + u(x,y,z,t-\Delta t)}{\Delta t^2}. \tag{2.10}$$

On note $U^n(x, y, z)$ l'approximation de $u(x, y, z, n\Delta t)$. On peut alors, en remplaçant la dérivée temporelle par son approximation dans (2.8) et (2.9), calculer $U^{n+1}(x, y, z)$ en fonction de $U^n(x, y, z)$ et de $U^{n-1}(x, y, z)$.

Ce schéma d'ordre deux est sans doute le plus utilisé pour la résolution explicite de l'équation d'onde. Cependant sous certaines conditions, des schémas moins dispersifs d'ordre plus élevés peuvent être plus adéquats. On peut trouver par exemple dans [42] une méthode permettant de construire des schémas d'ordre plus élevés en temps pour l'équation d'onde suivant la méthode de Lax-Wendroff ainsi que l'étude de l'intérêt de ce genre de schémas.

Les méthodes de modélisation tirent souvent leurs noms de la façon selon laquelle les dérivées spatiales sont calculées. On passe en revue dans les paragraphes suivants ces différentes méthodes.

Remarque 2 On notera que l'approximation (2.10) permet également d'exprimer le champ d'onde $U^{n-1}(x, y, z)$ à l'instant $(n-1)\Delta t$ en fonction de $U^n(x, y, z)$ et de $U^{n+1}(x, y, z)$. Ceci nous permettra de *remonter* le temps pour la RTM (voir 2.3).

2.2.1 Résolution par différences finies

Outre la discrétisation temporelle, les méthodes de calcul par différences finies (ou DF) reposent sur une discrétisation du domaine de calcul en espace selon une grille (le plus souvent régulière). Ces méthodes peuvent utiliser des formulations homogènes ou hétérogènes. Dans le premier cas, le domaine de résolution est composé de régions où les propriétés acoustiques du milieu sont supposées constantes. L'utilisation de cette approche est limitée aux modèles à géométries simples. La formulation hétérogène utilise l'équation des ondes pour les milieux hétérogènes et permet d'assigner à tout point de la grille ses propriétés acoustiques, ce qui permet de modéliser des domaines de différentes complexités. Divers types de grilles de discrétisation ont été introduits au cours du temps : les grilles conventionnelles, les grilles décalées, les grilles décalées avec rotation (*Rotated Staggered Grid* ou RSG [119]), etc. Nous évoquons ici les deux premiers types qui seront utilisés dans la suite du manuscrit.

Grille conventionnelle

La grille de discrétisation est dite conventionnelle lorsque toutes les quantités sont exprimées aux mêmes points de la grille. Cette approche est la plus naturelle pour le cas le plus simple de l'équation (2.9) qui décrit la propagation d'une onde acoustique dans un milieu où la densité est supposée constante.

On considère un maillage régulier du domaine de calcul de pas Δx, Δy et Δz. On note $U^n_{i,j,k}$ l'approximation de $u(i\Delta x, j\Delta y, k\Delta z, n\Delta t)$.

L'équation (2.9) est discrétisée en utilisant le schéma d'ordre 2 en temps décrit plus haut et un schéma centré d'ordre p en espace pour exprimer la dérivée seconde selon chaque dimension sous la forme :

$$\frac{\partial^2 u(i\Delta x, j\Delta y, k\Delta z, n\Delta t)}{\partial x^2} \approx \frac{1}{\Delta x^2} \sum_{l=-\frac{p}{2}}^{\frac{p}{2}} a_l U^n_{i+l,j,k}. \tag{2.11}$$

Les coefficients a_l du schéma sont obtenus en utilisant les développements de Taylor et sont donnés dans le tableau 2.1 pour $l \geq 0$, on a en outre la propriété de symétrie $a_l = a_{-l}$. On trouve dans [52] une méthode de génération de ces coefficients pour différents types de schémas et tout ordre de dérivation et de précision. Des coefficients réduisant la dispersion numérique peuvent également être utilisés [138].

L'algorithme de modélisation consistera donc à calculer l'état du champ d'onde à l'instant

Ordre	a_0	a_1	a_2	a_3	a_4
2	-2	1			
4	$\frac{-5}{2}$	$\frac{4}{3}$	$\frac{-1}{12}$		
6	$\frac{-49}{18}$	$\frac{3}{2}$	$\frac{-3}{20}$	$\frac{1}{90}$	
8	$\frac{-205}{72}$	$\frac{8}{5}$	$\frac{-1}{5}$	$\frac{8}{315}$	$\frac{-1}{560}$

TABLE 2.1 – Coefficients de Taylor pour différents ordres de schémas centrés pour le calcul de la dérivée seconde.

$(n+1)\Delta t$ pour chaque point de la grille selon l'égalité :

$$U^{n+1}_{i,j,k} = 2U^{n}_{i,j,k} - U^{n-1}_{i,j,k} + c^2\Delta t^2\left[\frac{1}{\Delta x^2}\sum_{m=-\frac{p}{2}}^{\frac{p}{2}} a_m U^{n}_{i+m,j,k} + \frac{1}{\Delta y^2}\sum_{m=-\frac{p}{2}}^{\frac{p}{2}} a_m U^{n}_{i,j+m,k} + \frac{1}{\Delta z^2}\sum_{m=-\frac{p}{2}}^{\frac{p}{2}} a_m U^{n}_{i,j,k+m}\right] \tag{2.12}$$

On aboutit à une molécule ou *stencil* de calcul à $N_{dim} \times p + 1$ points, où N_{dim} est la dimensionnalité ($N_{dim} = 2, 3$). La figure 2.1 représente ce *stencil* dans le cas $N_{dim} = 3$ et $p = 8$ (schéma centré d'ordre 8).

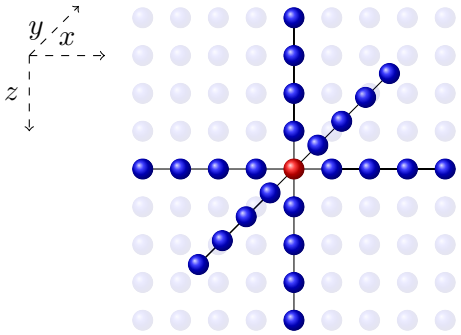


FIGURE 2.1 – Stencil 3D de modélisation en densité constante pour le schéma centré d'ordre 8.

Grille décalée (*staggered grid)*

La grille de calcul est dite décalée lorsque certaines quantités calculées le sont à des points différents de la grille (figure 2.2). L'idée pour discrétiser l'équation (2.8) (cas de la densité variable) est de considérer l'opérateur différentiel spatial comme une composition de deux opérateurs du premier ordre [49, 122] :

$$\frac{\partial^2 u}{\partial t^2} = K[\frac{\partial_-}{\partial x}(\frac{1}{\rho}\frac{\partial_+ u}{\partial x}) + \frac{\partial_-}{\partial y}(\frac{1}{\rho}\frac{\partial_+ u}{\partial y}) + \frac{\partial_-}{\partial z}(\frac{1}{\rho}\frac{\partial_+ u}{\partial z})]. \tag{2.13}$$

Les symboles ∂_+ et ∂_- expriment les opérateurs spatiaux centrés à mi-distance entre les mailles de la grille selon la direction de l'opérateur spatial respectivement dans les sens positif

et négatif. Par exemple, pour un schéma d'ordre 8, la dérivée première de u selon x, évaluée au point $(i+\frac{1}{2})\Delta x$ s'écrit :

$$\frac{\partial_+}{\partial x}u((i+\frac{1}{2})\Delta x) = \sum_{n=0}^{3} b_n[u((i+1+n)\Delta x) - u((i-n)\Delta x)] \tag{2.14}$$

où les b_n sont les coefficients du schéma DF d'ordre 8 obtenus par développements de Taylor [52] ou des coefficients optimisés pour limiter la dispersion numérique [63] .

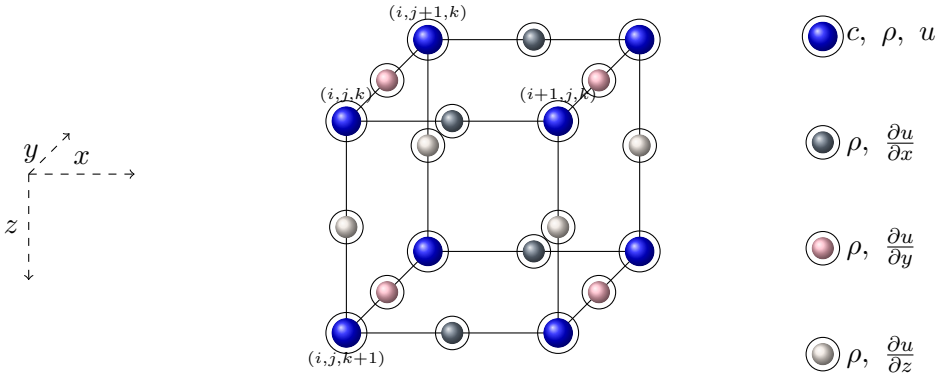


FIGURE 2.2 – Grille décalée en 3D.

Pour chaque opérateur de dérivation d'ordre 1, on aboutit à une molécule ou *stencil* à $N_{dim} \times p$ points. La taille totale du stencil est alors $N_{dim} \times (2p-2)+1$. La figure 2.3 représente le *stencil* selon la dimension x pour le cas d'un schéma d'ordre 8.

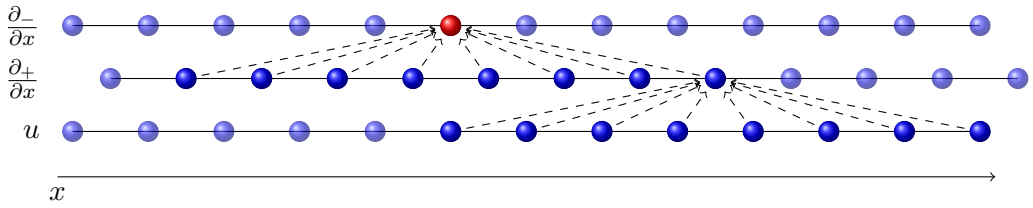


FIGURE 2.3 – Stencil 1D de modélisation en densité variable pour le schéma d'ordre 8.

Analyse des méthodes de différences finies : consistance, stabilité et CFL

L'étude des propriétés numériques des méthodes présentées dépasse le cadre de cette thèse, cependant ces notions sont importantes pour comprendre la complexité numérique de ces méthodes. C'est pour cela que le choix a été fait de les introduire brièvement ici.

Un schéma numérique est dit **consistant** si la différence entre la solution exacte et la solution fournie par le schéma (erreur de troncature $\tau(\Delta t, \Delta x)$) tend vers zéro quand les pas de discrétisation spatial et temporel tendent indépendamment vers zéro. Le schéma est dit d'ordre

p en temps et q en espace si $\tau(\Delta t, \Delta x) = O(\Delta t^p + \Delta x^q)$.

Un schéma numérique est dit **stable** s'il produit une solution bornée quand la solution exacte est bornée. Si la solution fournie par le schéma est bornée pour tous les pas de discrétisations, le schéma numérique est dit inconditionnellement stable.

Un schéma à la fois consistant et stable est **convergent**. Courant, Friedrichs et Lewy [41] ont constaté que le domaine de dépendance de la solution d'une EDP doit être inclus dans le domaine de dépendance numérique du schéma considéré pour que la solution numérique converge vers la solution analytique. A partir de ce constat, ils ont établi l'inégalité suivante connue sous le nom de condition de CFL :

$$|c\frac{\Delta t}{h}| \leq \alpha$$

où Δt et h représentent respectivement les pas de discrétisations en temps et en espace, c la vitesse maximale et α une constante qui dépend du schéma considéré.

En utilisant l'analyse de stabilité de Von Neumann et de façon analogue à l'étude faite dans [95], on montre que le schéma (2.12) d'ordre p en espace est stable si et seulement si la condition (2.15) est respectée :

$$(\frac{c\Delta t}{\Delta x})^2 + (\frac{c\Delta t}{\Delta y})^2 + (\frac{c\Delta t}{\Delta z})^2 \leq \alpha \tag{2.15}$$

avec $\alpha = 4(\sum_{l=-\frac{p}{2}}^{\frac{p}{2}} |a_l|)^{-1}$.

On retrouve dans [85] une conjecture permettant de généraliser le critère de stabilité pour des schémas de tout ordre en 2 et 3 dimensions lorsque le pas de discrétisation est le même dans toutes les directions ($\Delta x = \Delta y = \Delta z = h$). (2.15) est alors réduit à

$$(\frac{c\Delta t}{h}) \leq 2(N_{dim} \sum_{l=-\frac{p}{2}}^{\frac{p}{2}} |a_l|)^{-1/2}. \tag{2.16}$$

Outre les propriétés de stabilité, l'analyse de Von Neumann permet d'établir les propriétés de dispersion d'un schéma numérique.

Pour limiter la dispersion numérique spatiale et temporelle, des conditions supplémentaires viennent s'ajouter à celles évoquées précédemment [14, 122, 136]. Selon le schéma numérique utilisé, un nombre minimum de points par longueur d'onde spatiale et temporelle est nécessaire. La fréquence maximale à propager de pair avec la vitesse minimale du modèle (longueur d'onde minimale $\lambda_{min} = \frac{c_{min}}{f_{max}}$) déterminent alors la taille de la grille de discrétisation et constituent avec la CFL une autre limitation pour le pas en temps.

Exemple 1 Considérons un exemple concret d'une source de type ondelette de Ricker - dérivée seconde d'une Gaussienne (figure 2.4) - de fréquence centrale $f_c = 20Hz$ (Fréquence maximale $f_{max} = 2.5f_c = 50Hz$) et d'un modèle de vitesse où la vitesse minimale $c_{min} = 1500ms^{-1}$ (vitesse

dans l'eau) et la vitesse maximale $c_{max} = 4500ms^{-1}$. En considérant la nécessité d'utiliser un échantillonnage spatial de 4 points par longueur d'onde minimale ($PPW_s = 4$) on a $\Delta x = \Delta y = \Delta z = \lambda_{min}/PPW_s = \frac{c_{min}}{f_{max} * PPW_s} = \frac{1500}{50*4} = 7.5m$. L'échantillonnage temporel Δt est contraint par la CFL et par des considérations de dispersion numérique. Pour cet exemple $\Delta t \approx 1.6ms$. Pour un domaine de taille $12km \times 4km \times 10km$ et pour un simulation de $10s$ on a donc une grille numérique de taille $1600 \times 534 \times 1334$ et 6250 itérations. Ces conditions ont donc clairement un impact direct sur le coût des algorithmes de modélisation.

■

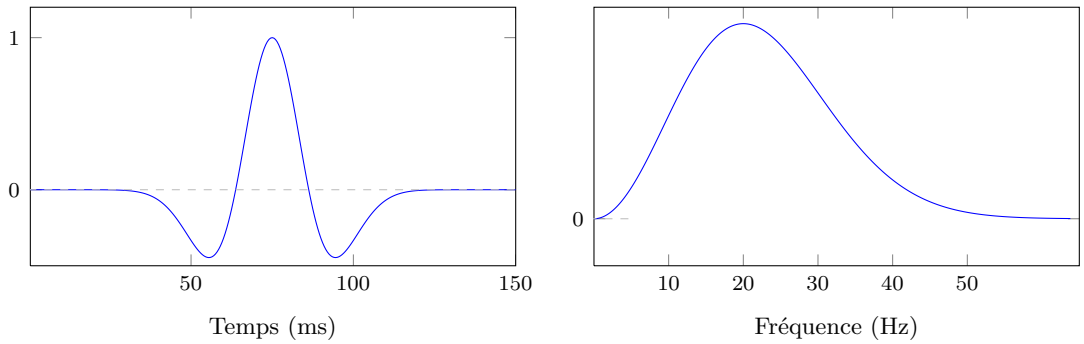


FIGURE 2.4 – Ondelette de Ricker de fréquence centrale $f_c = 20$ Hz. On considère que $f_{max} = 2.5f_c = 50Hz$.

On retrouve dans [22] une analyse de l'intérêt de l'utilisation des DF en sciences de la Terre pour résoudre les équations de propagation d'ondes, ce qui justifie son usage fréquent[20, 82, 94, 133]. On y retrouve également une analyse des principales limitations de cette méthode.

L'utilisation fréquente des DF est ainsi décrite comme la résultante de sa grande simplicité d'appréhension et d'implémentation numérique ainsi que de son adaptation aux objets étudiés en sciences de la Terre qui sont facilement représentables par des formes géométriques simples (souvent rectangulaires ou parallélépipédiques) facilement discrétisables avec des grilles régulières.

Parmi les limites qui sont citées, on retrouve celles liées aux pas de la grille de discrétisation ainsi qu'à la simulation de la topographie dans le cas d'une surface libre complexe.

Pour ce qui est du pas de la grille numérique de discrétisation, comme décrit précédemment, il est fixé de façon à avoir un nombre minimal de points par longueur d'onde simulée permettant de minimiser la dispersion numérique. Le pas de discrétisation spatial est donc contraint par la longueur d'onde minimale qui est proportionnelle à la vitesse minimale dans le milieu. C'est cette vitesse qui imposera donc le pas de discrétisation même si elle ne représente qu'une petite proportion du modèle étudié. Une façon de s'affranchir de cette contrainte est d'utiliser des grilles à pas variable [93] pour s'adapter aux vitesses des milieux considérés. En modélisation marine, résoudre analytiquement l'équation d'onde dans la couche d'eau (vitesse et densité constante) permet également d'éviter que la grille de discrétisation soit contrainte par la vitesse dans l'eau qui est généralement inférieure à celle dans les solides (cf. tableau 1.2).

Concernant la simulation de topographie complexes pour la surface libre (voir 2.2.5), comme dans le cas des piémonts de chaînes de montagnes par exemple, diverses techniques on été proposées. Une première approche repose sur l'approximation de la surface libre sous la forme d'une série de marches d'escalier qui nécessite d'adapter la grille de discrétisation à la courbure de la surface libre pour limiter les diffractions parasites ce qui entraine le sur échantillonnage de l'ensemble du milieu. Est ainsi citée comme exemple l'étude menée dans [20, 119] selon laquelle un suréchantillonnage d'un facteur 16 est nécessaire en 2D et 64 en 3D pour modéliser précisément la propagation des ondes de surface dans un modèle avec une topographie gaussienne avec un schéma numérique d'ordre deux. D'autres techniques s'appuient sur l'utilisation des coordonnées curvilignes ou une interpolation du modèle de vitesse des deux côtés de la surface libre. Toutes ces approches restent cependant limitées à des topographies simples et ont souvent pour effet de générer des diffractions parasites et d'augmenter la taille des calculs.

2.2.2 Résolution par les méthodes pseudo-spectrales

Pour résoudre les équations (2.8) et (2.9), on utilise le plus souvent la méthode des différences finies à l'ordre 4 ou 8 en espace et à l'ordre 2 en temps. L'étude de la dispersion numérique de différents schémas permet de retenir ce schéma comme celui qui offre le meilleur compromis entre quantité de calcul et précision numérique[14, 122].

Pour limiter la dispersion numérique associée à la discrétisation spatiale du Laplacien, une idée naturelle consiste à utiliser des méthodes plus précises. Les méthodes pseudo-spectrales [51, 54, 77, 114] utilisent la transformée de Fourier pour le calcul du Laplacien ce qui permet de limiter la dispersion numérique avec un pas d'échantillonnage à la limite du théorème d'échantillonnage (2 points par longueur d'onde, fréquence de Nyquist). En terme de précision, l'utilisation de la transformée de Fourier permet d'étendre l'ordre du stencil à la taille du domaine [51].

Dans ce cas, le calcul effectué en (2.10) est remplacé par :

$$\frac{\partial^2 u(x,y,z,t)}{\partial x^2} \approx \mathcal{TF}_x^{-1}[-{k_x}^2 \mathcal{TF}_x[U^n]] \tag{2.17}$$

où $\mathcal{TF}_x$ représente la transformée de Fourier selon x, $\mathcal{TF}_x^{-1}$ la transformée inverse et k_x le nombre d'onde. Ainsi le calcul de la dérivée dans le domaine spatial (équivalent à une convolution en DF) est réduit à un produit dans le domaine de Fourier. L'utilisation des méthodes pseudo-spectrales présente ainsi plusieurs intérêts :

- le noyau de calcul est basé sur l'utilisation des FFT (Fast Fourier Transform). Des bibliothèques optimisées existent aussi bien pour les processeurs généralistes (FFTW, MKL, ACML, etc.) que pour les accélérateurs (CUFFT[102] pour les accélérateurs graphiques, FFT personality pour les FPGA Convey, etc.).
- la précision élevée inhérente à la méthode permet de réduire le nombre d'échantillons par longueur d'onde, ce qui limite l'espace mémoire et de stockage nécessaire ainsi que les échanges de données entre sous domaines en général et entre l'accélérateur et le processeur dans le cas de l'utilisation d'un accélérateur matériel.

Le calcul des FFT dans leur version parallèle se fait de façon naturelle dans les systèmes à mémoire partagée, l'ensemble des unités de calcul ayant accès à l'ensemble des données. Cependant, dans le cadre des systèmes à mémoire distribuée, le calcul des FFT induisant des communications importantes, leur utilisation pour traiter un domaine réparti entre plusieurs processeurs peut avoir un coût prohibitif. Les méthodes pseudo-spectrales et les différences finies peuvent être utilisées de pair [33] pour palier à cette limitation. Dans le cas où la décomposition du domaine est nécessaire, l'idée consiste à utiliser la transformée de Fourier pour le calcul de la dérivée dans certaines directions et d'utiliser les différences finies pour d'autres ce qui autorise la décomposition du domaine selon ces dernières. L'utilisation des communications non bloquantes permet également de réduire le surcoût lié aux échanges de données[34].

Les méthodes pseudo-spectrales, comme pour les différences finies, restent basées sur des grilles régulières. Cela occasionne les mêmes limitations pour la prise en compte de topographies complexes en surface.

2.2.3 Autres méthodes

D'autres approches existent pour résoudre numériquement l'équation d'onde. On peut notamment citer les éléments finis (EF), les éléments spectraux (ES) ou encore des approches de type Galerkin discontinu (GD). Ces méthode n'ont pas été abordées dans le cadre de ce travail de thèse. Nous nous contentons donc d'une description très succincte. On pourra retrouver dans [22] une description plus détaillée. Les EF reposent sur la formulation variationnelle de l'équation d'onde [143]. Leur principal attrait réside dans la gestion naturelle qu'ils offrent des topographies et des modèles géologiques complexes grâce à l'utilisation de maillages non structurés triangulaires en 2D ou tétraédriques en 3D. Les principales limitations de l'approche résident dans la dispersivité et la précision limitée des EF et dans le fait qu'une formulation implicite est souvent de mise, ce qui nécessite l'inversion de très grandes matrices. Les éléments spectraux quant à eux ne requièrent pas d'inversion de matrices et offrent une meilleure précision que les EF. Ils nécessitent cependant des maillages par quadrangles en 2D et parallélépipèdes en 3D. Ces maillages sont difficiles à adapter aux modèles géologiques complexes et nécessitent souvent l'intervention humaine pour éliminer les mailles dégénérées.

2.2.4 Traitement des bords absorbants

Lors de la simulation numérique le domaine de propagation ne peut s'étendre indéfiniment. Pour simuler un domaine infini et éviter ainsi la réflexion des ondes aux limites du domaine, un traitement particulier aux bords artificiels du domaine fini est effectué pour absorber l'énergie. On retrouve dans la littérature deux approches principales. La première consiste en l'imposition sur les bords du domaine de conditions aux limites absorbantes (*absorbing boundary conditions* ou ABC), en s'appuyant sur une décomposition du champ d'onde local à la bordure du domaine en ondes planes sortantes et entrantes [40]. Il s'agit ensuite de ne propager que les ondes sortantes sur une distance correspondant à la taille du stencil utilisé et d'utiliser ces valeurs pour mettre

à jour le champ d'onde interne. L'utilisation de l'équation d'onde *one-way* étant très localisée (subordonnée à la largeur du stencil), cette approche se révèle très efficace en terme de coût numérique. Cependant son efficacité pour atténuer les réflexions se révèle limitée à certains angles d'incidence (angles d'incidence proches de la normale au bord du domaine).

La deuxième approche plus répandue s'appuie sur le rajout de couches absorbantes sur les bords du domaine pour atténuer progressivement l'amplitude du champ d'ondes. Dans [29, 78] les auteurs introduisent l'utilisation de bords "éponges" qui permettent d'atténuer progressivement et de façon isotrope en choisissant un profil adapté les ondes sur les couches rajoutées. Un facteur d'amortissement sera ainsi introduit dans l'équation (2.7) par exemple, qui sera remplacée alors dans les couches absorbantes par l'équation (2.18) [78] :

$$\frac{\partial^2 u(x,y,z,t)}{\partial t^2} - c^2(x,y,z)\Delta u(x,y,z,t) + 2\gamma(x,y,z)\frac{\partial u(x,y,z,t)}{\partial t} + \gamma^2 u(x,y,z,t) = s(t) \tag{2.18}$$

où $\gamma(x,y,z)$ représente le coefficient d'absorption. En négligeant le terme en γ^2 et en discrétisant à l'ordre 2 en temps on peut écrire :

$$\frac{u^{n+1} - 2u^n + u^{n-1}}{\Delta t^2} = -\gamma^2 \frac{u^{n+1} - u^{n-1}}{\Delta t^2} + c^2 \Delta u \tag{2.19}$$

On a donc

$$u^{n+1} = \frac{2u^n + (\gamma - 1)u^{n-1} + c^2 \Delta t^2 \Delta u}{\gamma + 1} \tag{2.20}$$

C'est cette formulation qui sera utilisée dans la suite.

Selon le schéma de discrétisation adopté et les fréquences propagées, l'utilisation des bords éponges peut nécessiter le rajout de couches absorbantes de taille importante (typiquement entre 20 et 100 points de grille) ce qui peut engendrer un surcoût important en termes de calcul et d'espace mémoire. Cette méthode, tout comme celle utilisant l'équation d'onde *one-way*, se montre particulièrement efficace pour les ondes à incidence normale. L'efficacité de la méthode dépend ainsi de l'angle d'incidence mais également de la fréquence des ondes incidentes. Berenger [16] introduit en 1994 l'utilisation des PML (*Perfectly Matched Layers*) comme couches absorbantes pour la simulation de la propagation d'ondes électromagnétiques (EM) en 2D. L'utilisation des PML a été étendue aux simulations EM en 3D [17, 31] ainsi que pour la simulation de propagation d'ondes acoustiques [46, 86] et élastiques [30]. Ceci a été fait le plus souvent pour les formulations de premier ordre mais également de second ordre [76].

Les PML présentent dans leur formulation analytique la propriété d'avoir un coefficient de réflexion nul pour toute onde incidente indépendamment de son angle d'incidence et de sa fréquence. Grâce à leur efficacité les PML réduisent la taille des couches d'absorption nécessaires (typiquement 10 points de grille). Cependant la méthode induit des degrés de liberté supplémentaires et peut avoir un surcoût numérique important.

En pratique, dans un monde discret, les PML montrent cependant leurs limites. En effet, la discrétisation induit la réflexion des ondes rasantes. Les CPML (*Convolutional Perfectly Matched Layers*), introduites en EM et adaptées aux ondes élastiques [75] représentent un raffinement de la méthode permettant de mieux absorber ces ondes rasantes.

2.2.5 Condition de surface libre

Au niveau de la surface libre[2] ($z = 0$), le champ d'onde est nul (condition de Dirichlet) et la vitesse est continue (Neumann). En pratique, cela signifie la mise à zéro des points de grille $z = 0$ et le rajout d'un champ antisymétrique ($u(x, y, z, t) = u(x, y, -z, t)$ pour $z < 0$) de la taille du stencil.

2.3 Imagerie sismique par équation d'onde : Reverse Time Migration

Les méthodes d'imagerie sismique reposent sur le principe d'imagerie énoncé par Claerbout en 1971[36] : *"Une réflexion dans le sous-sol existe là où l'arrivée première de l'onde descendante coïncide en temps avec une onde montante"*.

La RTM ou migration en temps inverse a été introduite pour la migration après sommation des données *zero-offset*. Pour ce type de migration, basé sur l'hypothèse d'un modèle de vitesse par couches sans variations latérales, l'approche conventionnelle consistait à extrapoler ou *"démigrer"* en profondeur les données enregistrées en surface [38]. Le modèle de vitesse initial est remplacé par un modèle où les vitesses ont été divisées par deux [87]. En remontant au temps $t = 0$ du point de tir, le front d'onde enregistré par les récepteurs est alors localisé au niveau du réflecteur (principe du réflecteur explosant ou *exploding reflector* [87]). Baysal [15], en même temps que d'autres auteurs, a proposé l'idée de démigrer non plus en extrapolant en profondeur mais en remontant le temps. Cette approche a pour avantage de ne pas se limiter à l'utilisation d'une formulation particulière de l'équation d'onde.

Quand l'hypothèse d'un modèle de vitesse par couches sans variations latérales n'est plus valide, la migration doit se faire avant sommation. Dans ce cas, l'utilisation du modèle du réflecteur explosant permettant de remonter au temps $t = 0$ n'est plus possible. Dans son utilisation actuelle pour la migration avant sommation, de toutes les méthodes de migration sismique la RTM est peut-être la retranscription la plus intuitive du principe d'imagerie. Les concepts d'ondes montantes et descendantes sont généralisés à l'utilisation du champ source et récepteurs. L'idée est donc pour aboutir à l'image sismique du sous sol $I(x, y, z)$, qui représente les coefficients de réflectivité en tout point de celui-ci, de calculer la corrélation entre le champ incident de la source $S(x, y, z, t)$ et le champ rétropropagé des récepteurs $R(x, y, z, t)$. Ces champs sont obtenus en utilisant le procédé de modélisation décrit précédemment en utilisant respectivement le signal de la source et des récepteurs comme conditions aux limites pour chaque point de tir enregistré lors de l'acquisition sismique. La condition d'imagerie consiste à accumuler au cours du temps la corrélation entre champs source et récepteurs. L'image finale est obtenue en additionnant les images obtenues pour tous les points de tirs P. Elle s'écrit alors :

$$I(x, y, z) = \sum_P \sum_t S_P(x, y, z, t) * R_P(x, y, z, t). \tag{2.21}$$

2. L'interface entre l'air et le domaine physique simulé est considérée comme surface libre.

2.4 Dimensionnement des problèmes et évaluation de la complexité calculatoire

Dans les sections précédentes plusieurs méthodes de modélisation et de migration ont été introduites. Ces méthodes ont en commun le fait de reposer sur une approximation de l'équation d'onde. Selon la complexité de cette approximation et de sa fidélité aux propriétés physiques des milieux, ces méthodes induisent des algorithmes de complexités plus ou moins importantes. Pour mettre en exergue les besoins importants en moyens de calcul en imagerie sismique, la complexité des algorithmes de modélisation acoustique par différences finies sera brièvement étudiée. Les évolutions nécessaires pour une fidélité accrue à la physique de la propagation des ondes dans les des milieux considérés seront présentées ainsi que le surcoût qu'elles introduisent.

La résolution de l'équation d'onde acoustique dans le cas isotrope a été décrite dans la section 2.2. Selon la taille des domaines, jusqu'à 98% du temps global de l'application peut être dédié à la mise à jour du champ d'onde en utilisant le calcul décrit dans les équations (2.12) et (2.14) respectivement pour les cas où la densité est supposée constante (DC) ou variable (DV). Les algorithmes correspondants sont respectivement l'algorithme 1 et l'algorithme 2 décrits ci-dessous pour :

N_t	:	Nombre de pas de temps
N_x,N_y,N_z	:	Dimension dans chaque direction
N_{dim}	:	Dimensionnalité de la simulation ($N_{dim} = 2, 3$)
$l = \frac{p}{2}$	:	p est l'ordre en espace du schéma numérique

ALGORITHME 1 : Modélisation 3D en densité constante (DC)

```
/* Boucle sur les points de tir                                          */
pour s = 1 à N_s faire
    pour n = 0 à N_t − 1 faire
        Introduire le terme source
        pour i = 1 à N_x faire
            pour j = 1 à N_y faire
                pour k = 1 à N_z faire
                    U^{n+1}_{i,j,k} ← 2U^n_{i,j,k} − U^{n−1}_{i,j,k} + c^2_{i,j,k} Δt^2 {a_0 U^n_{i,j,k}
                                + 1/Δx^2 Σ_{m=1}^{l} a_m (U^n_{i+m,j,k} + U^n_{i−m,j,k})
                                + 1/Δy^2 Σ_{m=1}^{l} a_m (U^n_{i,j+m,k} + U^n_{i,j−m,k})
                                + 1/Δz^2 Σ_{m=1}^{l} a_m (U^n_{i,j,k+m} + U^n_{i,j,k+m})}
```

ALGORITHME 2 : Modélisation 3D en densité variable (DV)

```
/* Boucle sur les points de tir                                          */
pour s = 1 à N_s faire
    pour n = 0 à N_t − 1 faire
        Introduire le terme source
        pour i = 1 à N_x faire
            pour j = 1 à N_y faire
                pour k = 1 à N_z faire
                    Dx_{i,j,k} ← 1/(ρ_{i+1/2,j,k} Δx^2) Σ_{m=0}^{p/2−1} b_m (U^n_{i+m+1,j,k} − U^n_{i−m,j,k})
                    Dy_{i,j,k} ← 1/(ρ_{i,j+1/2,k} Δy^2) Σ_{m=0}^{p/2−1} b_m (U^n_{i,j+m+1,k} − U^n_{i,j−m,k})
                    Dz_{i,j,k} ← 1/(ρ_{i,j,k+1/2} Δz^2) Σ_{m=0}^{p/2−1} b_m (U^n_{i,j,k+m+1} − U^n_{i,j,k−m})
        pour i = 1 à N_x faire
            pour j = 1 à N_y faire
                pour k = 1 à N_z faire
                    U^{n+1}_{i,j,k} ← 2U^n_{i,j,k} + U^{n−1}_{i,j,k}
                                + K_{i,j,k} Δt^2 { 1/Δx^2 Σ_{m=0}^{p/2−1} b_m (Dx_{i+m,j,k} − Dx_{i−m−1,j,k})
                                + 1/Δy^2 Σ_{m=0}^{p/2−1} b_m (Dy_{i,j+m,k} − Dy_{i,j−m−1,k})
                                + 1/Δz^2 Σ_{m=0}^{p/2−1} b_m (Dz_{i,j,k+m} − Dz_{i,j,k−m−1})}
```

La complexité $\mathcal{C}$ en nombre d'opérations de ces algorithmes dépend de l'ordre en espace du schéma numérique utilisé (on se limite aux schémas d'ordre 2 en temps). En DC, l'utilisation d'un schéma d'ordre p en espace induit $(N_{dim} \times 3l)+5$ opérations en virgule flottante par nœud de grille avec $l = \frac{p}{2}$ (p pair). En effet l'approximation du Laplacien coûte une multiplication pour le nœud central et une multiplication et deux additions pour chaque paire de nœuds symétriques par rapport au nœud central. Pour l'itération du schéma en temps, quatre opérations sont nécessaires.

En DV, un calcul similaire montre que la résolution pour un ordre $p = 2l$ en espace induit $N_{dim} \times (6l + 4)$ opérations flottantes par point de grille. La complexité de ces algorithmes est alors trivialement déduite pour un point de tir. Si $p = 2l$ on a :

$$\begin{aligned} DC: &\quad \mathcal{C} = N_t \times N_x \times N_y \times N_z \times N_{dim} \times (3l + 5) \\ DV: &\quad \mathcal{C} = N_t \times N_x \times N_y \times N_z \times N_{dim} \times (6l + 4) \end{aligned}$$

En ce qui concerne les besoins mémoire, en DC trois tableaux sont nécessaires : $c(N_x, N_y, N_z)$, $U^n(N_x, N_y, N_z)$ et $U^{n+1}(N_x, N_y, N_z)$, U^{n+1} pouvant remplacer U^{n-1} en mémoire. En DV les tableaux $\rho(N_x, N_y, N_z)$, $D_x(N_x, N_y, N_z)$, $D_y(N_x, N_y, N_z)$, et $D_z(N_x, N_y, N_z)$ sont également nécessaires.

Remarque 3 Le calcul de ρ aux nœuds décalés de la grille (par exemple $\rho(i + \frac{1}{2}, j, k)$) induit soit un coût calcul supplémentaire pour effectuer une interpolation soit un besoin mémoire supplémentaire pour des tableaux précalculés. On se place dans cette section dans le cadre plus simple où la densité ne varie que très lentement dans le modèle ($\rho(i + \frac{1}{2}, j, k) \approx \rho(i, j, k)$).

Exemple 2 Dans l'exemple 1, nous avons établi que pour une étude conventionnelle, une grille numérique de taille $1600 \times 534 \times 1334$ et 6250 itérations en temps étaient nécessaires. Dans le cas de l'utilisation d'un schéma d'ordre 2 en temps et 8 en espace (hypothèse utilisée pour le calcul du nombre de points par longueur d'onde nécessaire), le nombre d'opérations par point de grille sera de $N_{dim} \times 3l + 5 = 3 \times 3 \times 4 + 5 = 41$. Avec une utilisation de 40% des performances crête des ressources de calcul et en fixant le temps de calcul à un jour pour traiter 10000 points de tir, la puissance crête nécessaire serait alors de 80 TFlops. Cette puissance est actuellement disponible et la modélisation acoustique est utilisée quotidiennement en production chez Total.

■

La formulation acoustique décrite précédemment ne tient pas compte de l'anisotropie du milieu. L'intégration de l'anisotropie induit une complexité supplémentaire. Rappelons qu'un milieu isotrope est un milieu dans lequel les propriétés du sous-sol sont les mêmes dans toutes les directions. Par opposition, un milieu anisotrope est un milieu où les propriétés élastiques des formations rocheuses sont fonction de la direction, ceci étant particulièrement vrai pour les roches sédimentaires où une direction est privilégiée, à savoir celle de la sédimentation. C'est ce type de formation qui contient la plupart des réservoirs pétroliers. La caractérisation d'un milieu anisotrope (dans le cas *Tilted Transverse Isotropy* ou TTI) requiert quatre paramètres

supplémentaires par point de grille (paramètres de Thomsen [129] δ et ϵ , pendage et azimut de l'axe de symétrie) par rapport aux paramètres utilisés pour les milieux isotropes ($V_p et \rho$). De plus, la résolution de l'équation d'onde acoustique anisotrope est souvent formulée via un système de deux équations couplées d'ordre 2 [50]. Tout ceci représente un surcoût important en termes de complexité de calcul et de besoin en mémoire. On considère généralement que la formulation anisotrope TTI augmente d'un ordre de grandeur les coûts de calcul comparé à la formulation isotrope.

L'utilisation de l'équation d'onde acoustique reste également limitée. En effet, des phénomènes de conversion ont lieu au niveau des interfaces entre ondes S et P. Ces phénomènes sont d'autant plus importants que les contrastes d'impédances sont forts, typiquement dans le cas de l'existence de structures salifères dans le modèle [72, 108]. Pour exploiter ces informations, on assiste depuis quelques années au développement des acquisitions multi-composantes qui permettent d'enregistrer toutes les composantes du champ vitesse en plus de la pression. Pour comprendre ces phénomènes, il est nécessaire de considérer la formulation élastique de l'équation d'onde [79], celle-ci étant la seule à pouvoir transcrire les phénomènes de conversion de modes. La résolution du système élastique nécessite la résolution de 9 équations couplées. Elle peut être jusqu'à 1 ordre de grandeur plus coûteuse en ressources de calcul que pour le cas acoustique. Usuellement, les données issues des acquisitions sont traitées en utilisant les outils acoustiques.

La formulation élastique est une bonne approximation, mais elle ne prend pas en compte l'atténuation qui existe pour tout matériau et qui résulte de la transformation de l'énergie en chaleur. La formulation viscoélastique [117] prend en compte l'atténuation du milieu. Ce modèle peut être encore amélioré en introduisant l'anisotropie et la topographie ce qui le rend encore plus coûteux.

La résolution de l'image sismique est de l'ordre de grandeur de la longueur d'onde dominante considérée qui est inversement proportionnelle à la fréquence. Pour améliorer la qualité de l'image finale et imager des évènements du sous sol, il convient d'élargir le spectre des fréquences utilisées. Or, le spectre des fréquences à exploiter constitue un autre paramètre ayant une incidence directe sur les calculs. Une fréquence plus haute impose des grilles plus fines et la condition de stabilité pour les schémas explicites réduit les pas de temps. La complexité du calcul évolue ainsi en fonction de la fréquence maximale F_{max} à propager en $\mathcal{O}(F_{max}^4)$. Passer ainsi d'une fréquence maximale de 30Hz à une fréquence maximale de 60Hz multiplie, à temps de calcul égal, la puissance de calcul nécessaire par 16.

Pour certains algorithmes la complexité du milieu peut également avoir une influence sur le temps de calcul. Ce n'est pas le cas pour les différences finies.

La taille des données est également un élément important à prendre en compte. En effet le nombre de points de tir peut varier en fonction de la surface à couvrir mais aussi en fonction du type d'acquisition (considérer par exemple le cas d'un acquisition Coil, cf. Chapitre 4).

Outre tous les éléments exposés, utiliser un algorithme donné pour la RTM présente des difficultés supplémentaires en termes d'E/S et de stockage. Quant à l'inversion de formes d'ondes, elle nécessite plusieurs itérations de modélisation.

A partir de tous ces éléments, on peut établir une comparaison relative des coûts de ces différents algorithmes. Cette comparaison est récapitulée dans le tableau 2.2.

Puissance	Applications
1	RTM acoustique isotrope 30 Hz
10	RTM acoustique anisotrope 30 Hz - FWI acoustique
100	RTM acoustique isotrope 60 Hz
1000	RTM acoustique anisotrope 60 Hz - FWI élastique

TABLE 2.2 – Puissance de calcul et applications.

Pour permettre une implémentation de la RTM anisotrope avec une fréquence de 60Hz et en considérant les performances énergétiques actuelles (3,2 GFlops par Watt pour le premier calculateur au Green500 [56] de juin 2013 équipé d'accélérateurs GPU NVIDIA K20), il faudrait un système alimenté par une puissance électrique de 320 MW. Si maintenant on ne considère pas les systèmes à base de technologie hybride (accélérateurs GPU ou MIC qui occupent les quatre premières places du même classement), le système qui est à la cinquième place est un BlueGene/Q à base de processeurs Power BQC et a une efficacité énergétique de 2.3 GFlops/Watt, ce qui donnerait une puissance électrique totale de 434 MW. Le premier cluster à base de CPU Intel conventionnels (à la quarantième place au Green500 !) requerrait quant à lui encore plus du double de cette puissance (efficacité énergétique de 1,2 GFlops/Watt).

Toutes ces considérations illustrent les raisons du besoin toujours croissant de ressources de calcul en exploration pétrolière et la nécessité de systèmes plus performants et moins énergivores. Les algorithmes 1 et 2 exhibent également divers niveaux de parallélisme, aux niveaux des points de tir et des points de la grille notamment, ce qui laisse envisager des implémentations sur divers types d'architectures parallèles.

Chapitre 3

Solutions de calcul intensif pour l'imagerie sismique

A supercomputer is a device for turning compute-bound problems into I/O-bound problems. (Seymour Cray)

Sommaire

Les challenges de l'exploration pétrolière, comme ceux illustrés précédemment par les méthodes de modélisation et de migration par différences finies, requièrent des codes de calcul de plus en plus complexes et nécessitant des puissances de calcul en évolution constante. Pour suivre cette évolution, la puissance de calcul installée des compagnies pétrolières a été augmentée de trois ordres de grandeur au cours de la dernière décennie. Pour espérer exploiter cette puissance de calcul disponible, il a fallu prendre en compte les évolutions des architectures et utiliser les modèles de programmation adéquats. Nous décrivons dans ce qui suit un bref historique du développement conjoint des méthodes utilisées en imagerie sismique et des puissances de calcul des architectures "classiques" qui ont permis leur mise en œuvre. Les technologies accélératrices peuvent constituer un enrichissement, voire une alternative, intéressant pour ces architectures. Nous introduisons donc ensuite les motivations et les particularités de ces technologies.

3.1 Imagerie sismique et architectures de calcul classiques

3.1.1 Historique

On retrouve dans [26] une description de l'évolution des algorithmes d'imagerie sismique qui a suivi de près celle des moyens informatiques disponibles.

Les premières implémentations de codes de modélisation sismique (voir par exemple [114]) et de migration après sommation [64] en trois dimensions ont fait leur apparition dès les années 1980 grâce aux premiers supercalculateurs vectoriels CRAY et à leurs quelques centaines de MFlops (2 GFlops par exemple pour le CRAY-2 équipé de quatre processeurs vectoriels produit en 1985 [71]).

Pendant les années 1990, l'avènement des processeurs superscalaires puis des architectures massivement parallèles permet l'implémentation des premiers algorithmes de migration profondeur 3D de Kirchhoff avant sommation, ainsi que des algorithmes de migration par équation d'onde 3D après sommation plus performants.

Pendant la deuxième moitié de la décennie 1990, avec les machines SMP (Symmetric Multi-Processors) et les grappes de calcul (*clusters*) apparaissent deux nouvelles familles de calculateurs qui vont permettre le développement des techniques modernes d'imagerie en profondeur qui sont aujourd'hui utilisées, à savoir la migration profondeur avant sommation 3D par équation d'onde.

Les calculateurs de type SMP permettent à un nombre important de processeurs d'avoir une vision globale de la mémoire disponible sur toute la machine ce qui permet le développement d'applications gérant des volumes importants de données avec un modèle de programmation simplifié. Les grappes de calcul, quant à elles, sont construites en interconnectant des nœuds dont l'architecture est similaire à celles de simples PC via un réseau rapide peu coûteux pour au final obtenir un supercalculateur extrêmement puissant à un coût limité. Les compagnies pétrolières ont profité de cet essor et ont construit des centres de calculs dotés de machines avec plusieurs milliers de processeurs. Ceci a permis de développer la migration 3D profondeur avant sommation de type RTM qui est aujourd'hui la référence en matière d'imagerie profondeur.

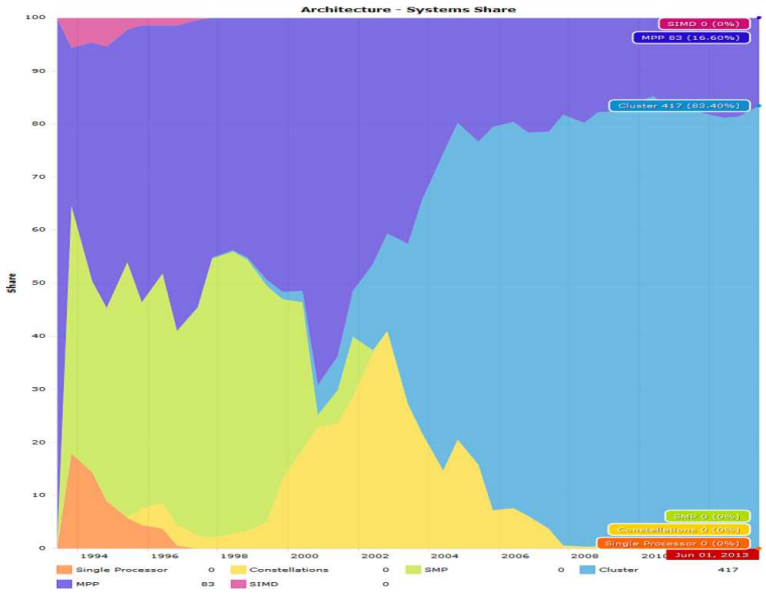


FIGURE 3.1 – Evolution de la proportion des architectures des calculateurs au Top500. En légende le nombre au classement de juin 2013 (selon [130]).

Les clusters, qui constituent aujourd'hui la majorité des systèmes (82.2% des systèmes du Top500 en juin 2013 [130] comme illustré sur la figure 3.1), ont évolué vers des machines plus spécialisées. Aujourd'hui, les clusters ont tous des réseaux d'intercommunication de plus en plus performants et les nœuds de calculs sont des machines SMP (4 à 64 cœurs suivant les constructeurs).

Les plateformes de tests utilisées pour nos travaux appartiennent à cette dernière famille. Une grande partie de ce type de plateforme est équipée de processeurs Intel Xeon de la famille Intel 64 (76%). Nous décrivons brièvement dans ce qui suit l'architecture et l'évolution de cette famille de processeurs au cours de ces dernières années. Cette description sera utile par la suite pour les comparaisons avec les accélérateurs matériels.

Les clusters récents sont souvent constitués de nœuds SMP dual-sockets comme illustré sur les figures 3.2. Ces figures représentent les processeurs Xeon Harpertown (sortis fin 2007) et Gainestown (ou Nehalem-EP, sortis en 2009). Le processeur Harpertown est un processeur quadri-cœurs conçu selon la microarchitecture Core ou Penryn introduite début 2006. Harpertown est la version pour serveur (gamme Xeon) de la famille Penryn ayant une finesse de gravure de $45nm$. Chaque cœur possède 3 ALU et 3 unités SSE (Intel Streaming SIMD Extensions) et $64ko$ de cache L1. Les cœurs sont regroupés par paires partageant $6Mo$ de cache $L2$. La microarchitecture Nehalem succède à Penryn à partir de 2008. Le processeur Gainestown est basée sur cette microarchitecture. C'est également un processeur quadri-cœur ayant une finesse de gravure de $45nm$. Chaque cœur possède 3 ALU et 3 unités SSE et $64ko$ de cache L1 et $256ko$ de cache L2. L'ensemble de cœurs partagent 4 ou $8Mo$ de cache L3 selon les versions. Les processeur Nehalem se sont affranchis de l'utilisation du Front Side Bus (FSB). Nehalem

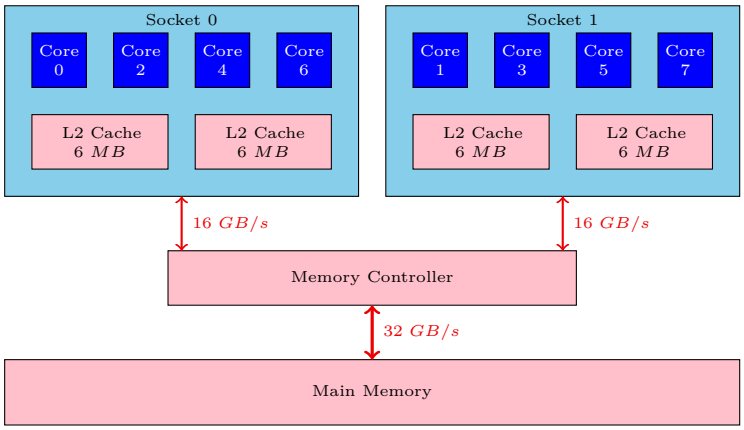


FIGURE 3.2 – Diagramme des processeurs Intel Harpertown.

utilise l'interconnexion QPI (Quick Path Interface) ainsi qu'un accès direct à la mémoire grâce à un contrôleur mémoire intégré sur la puce. Ceci a pour effet de quasiment doubler la bande passante mémoire ($32GB/s$ contre $16GB/s$ auparavant) et de réduire la latence. Cependant cette architecture NUMA (Non Uniform Memory Access) induit des temps d'accès différents selon l'emplacement des données par rapport au cœur (accès direct ou passage par le QPI). Ce problème de l'affinité mémoire impose des contraintes algorithmiques et de programmation supplémentaires [112]. Avec Nehalem, Intel signe également le retour à l'hyperthreading abandonné pour la microarchitecture Core. L'hyperthreading a été repris par les générations suivantes Westmere et Sandy Bridge, à laquelle appartiennent les processeurs Intel les plus récents.

3.1.2 Environnement de programmation parallèle classique

Pour le développement d'applications parallèles, deux modèles de programmation se sont imposés au cours des dernières années dans le monde du calcul scientifique selon l'architecture matérielle sous-jacente. Il s'agit du modèle de programmation à mémoire partagée et du modèle de programmation à mémoire distribuée par passage de messages utilisant respectivement principalement OpenMP [43] et MPI [123]. Ces outils ont été adoptés par une large communauté de développeurs et sont supportés par les constructeurs.

OpenMP (Open Multi-Processing) est une interface de programmation (API) adoptée comme standard en 1997 par une large communauté d'industriels et de constructeurs. OpenMP permet via l'annotation des sources d'un programme par des directives de compilation, de spécifier un ensemble de tâches et de régions pouvant s'exécuter en parallèle au sein d'un programme principal séquentiel. Un compilateur OpenMP pourra alors extraire ce parallélisme et créer le nombre adéquats de processus légers (*threads*) pour l'exécuter. Le programme principal est ainsi exécuté par un processus unique qui active des processus légers à l'entrée des régions parallèles.

Outre les directives de compilation, un ensemble de routines et de variables d'environnement définit le comportement du programme à l'exécution. Les communications se font de façon implicite en accédant simplement à la même zone en mémoire. OpenMP est donc particulière-

ment adapté aux systèmes à mémoire partagée. Plusieurs tentatives ont été faites pour adapter OpenMP aux architectures à mémoire distribuée en utilisant les DSM [1]. Ces tentatives se sont souvent heurtées aux limitations inhérentes à ce modèle de programmation : le surcoût engendré par la cohérence à maintenir entre différents niveaux de mémoire et la latence due à la création des threads limitent fortement la scalabilité.

La programmation d'applications ciblant des architectures à mémoire distribuée se fait plus naturellement avec le modèle par passage de messages. MPI est la spécification de la librairie de programmation par passage de message -ou communications explicites- la plus utilisée en calcul scientifique et peut être considérée de ce point de vue comme un standard de fait.

Comme détaillé en 2.4, plusieurs niveaux de parallélisme peuvent être exploités en modélisation sismique. Un premier niveau qu'on peut qualifier d'*embarrassingly parallel* concerne les points de tir qui peuvent être traités indépendamment et ne nécessitent ni communication ni synchronisation. Pour chaque point de tir, il arrive très fréquemment que les besoins en mémoire d'un problème dépassent la taille de la mémoire disponible sur un seul nœud de calcul. Une approche qui parait naturelle pour ce genre de problème est l'approche SPMD (Single Process/Program Multiple Data) par décomposition de domaines [58] requérant une décomposition explicite des données en vue de leur distribution sur un certain nombre de processus et une schéma de communications explicites entre ces processus. La figure 3.3 illustre cette approche pour le cas d'une grille 2D décomposée en quatre sous-domaines. Cette approche donne lieu dans le cas des différences finies à l'apparition de nœuds fantômes (ou *ghost nodes* ou halo qui doivent être échangés à chaque itération en temps entre les sous-domaines adjacents. Nous verrons par la suite que ces échanges peuvent engendrer des surcoûts importants.

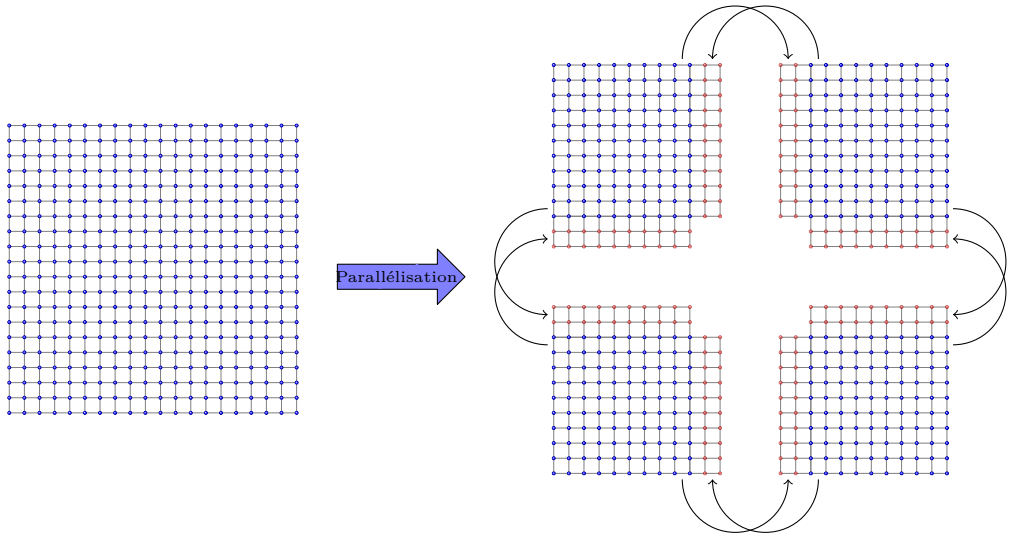


FIGURE 3.3 – Parallélisation par décomposition de domaine et échange des nœuds fantômes.

La programmation d'applications parallèles pour grappes de calcul constituées de nœuds

1. Distributed Shared Memory : mécanisme matériel ou logiciel permettant de simuler une mémoire logiquement partagée l'aide d'une mémoire physiquement distribuée.

équipés de multi-processeurs à mémoire partagée peut se faire en utilisant une approche hybride faisant appel à la fois à OpenMP et MPI [113]. Cette approche permet notamment d'éliminer la réplication et les échanges de données au niveau d'un nœud ce qui conduit à un gain en temps et en mémoire[80]. Les dernières années ont vu également une évolution vers le modèle de programmation PGAS (Partitioned Global Address Space) avec UPC (Unified Parallel C) et Co-array Fortran notamment. Ce modèle de programmation est similaire à celui d'une DSM avec la différence notable qu'une affinité est introduite entre tout processus et la mémoire locale du processeur sur lequel il s'exécute.

3.1.3 Processeurs généralistes : limites et perspectives actuelles

Après la fin de la course à la puissance séquentielle pour les processeurs, la tendance actuelle est à l'augmentation du parallélisme au sein du nœud de calcul. En effet, on assiste depuis une dizaine d'années à la stagnation des fréquences. Difficile de parler d'évolutions des moyens de calcul sans évoquer la loi exprimée en 1965 par Gordon Moore et qui porte depuis le nom de Loi de Moore. Cette loi prévoit un doublement de la complexité des semi-conducteurs proposés en entrée de gamme tous les dix-huit mois. En 1975, Moore réévalua sa prédiction en posant que le nombre de transistors des microprocesseurs (et non plus de simples circuits intégrés moins complexes car formés de composants indépendants) sur une puce de silicium doublerait tous les deux ans. Bien qu'il ne s'agisse pas d'une vraie loi physique, cette prédiction s'est révélée étonnamment exacte. Entre 1971 et 2001, la densité des transistors a doublé chaque 1,96 année. Cependant, on constate que depuis 2002 cette progression n'est plus respectée. Plusieurs facteurs expliquent ce ralentissement : des facteurs techniques notamment, liés à la dissipation thermique, ainsi qu'économiques tels que l'augmentation de la consommation électrique.

En terme d'intégration, l'évolution décrite plus haut des plateformes de calcul haute performance atteint également ses limites : occupation, consommation électrique, bande passante mémoire, etc. Pour atteindre les 100 pétaflops soutenus puis l'exaflops la tendance actuelle est à l'exploitation poussée du parallélisme à toutes les échelles, depuis les pipelines d'exécution jusqu'au cluster. Une autre tendance moins marquée est à la simplification des processeurs et à la réduction de leur fréquence ; ceci offre l'avantage de réduire leur consommation électrique, d'augmenter leur efficacité, et permettre ainsi d'augmenter leur nombre global au sein d'un système. Les récentes avancées pour l'ARM (notamment de l'implémentation Tegra de NVIDIA) et les cœurs de calcul du MIC d'Intel (Xeon Phi) s'inscrivent dans cette voie. C'est également le principe des machines BlueGene d'IBM qui ont longtemps été les machines les plus efficaces énergétiquement (20 premières places du Green500 en juin 2012 !) avant d'être détrônées par les architectures hétérogènes à base d'accélérateurs (4 premières places du classement de juin 2013, GPU de NVIDIA et d'AMD et Intel Xeon Phi).

3.1.4 Emergence des accélérateurs de calcul

La tendance actuelle des technologies matérielles qui conduit à une structure hétérogène hiérarchique des machines incite à exploiter plusieurs niveaux de parallélisme et la question cruciale réside dans le choix de la granularité des différents calculs. Cette tendance s'est affirmée suite à l'apparition, ou plutôt à la réapparition [2], des accélérateurs de calcul. Il s'agit de composants spécialisés qui viennent se greffer à des systèmes à base de processeurs standards et sur lesquels pourra être déportée une partie des calculs. Cette tendance est amenée à se confirmer au cours des prochaines années ce qui a vocation à rendre le paysage du calcul haute performance encore plus hétérogène et difficile à exploiter efficacement. Actuellement, on distingue principalement trois classes d'accélérateurs :

- les unités de calcul graphiques (*Graphical Processing Units* ou GPU) ;
- les FPGA (*Field Programmable Gate Arrays*) ;
- les accélérateurs généralistes de type Cell ou Clearspeed et plus récemment le Xeon Phi.

Il est à souligner que chaque type d'accélérateur peut de par son architecture être particulièrement bien adapté à exécuter efficacement un certain type de calcul et totalement inefficace pour d'autres ; ceci implique l'utilisation de ces accélérateurs comme co-processeurs et renforce donc le besoin constant d'un processeur généraliste. Ceci pose également d'emblée le problème des échanges de données entre ces unités de calcul qui peut très souvent constituer le goulot d'étranglement pour les applications comme nous le verrons par la suite. Se pose aussi la question de la structuration hiérarchique des applications et de la délimitation des parties à accélérer. En effet, à l'inefficacité et l'inadaptation se rajoutent la difficulté de programmation et l'effort nécessaire de portage. Les accélérateurs sont donc préconisés pour les applications dont les noyaux de calculs les plus lourds sont bien localisés.

Nous décrivons dans la suite ces différents types d'accélérateurs en détaillant en particulier les accélérateurs graphiques qui ont été les plus utilisés dans le cadre de cette étude.

3.2 Calculs généralistes sur processeurs graphiques

Le calcul généraliste sur processeurs graphiques, communément appelé GPGPU (*General-purpose computing on graphics processing units*), longtemps considéré comme une curiosité, connaît ces dernières années un essor important. En témoigne l'activité de la communauté de programmeurs (réunie autour du site GPGPU.org [55]) et l'évolution rapide des solutions matérielles ; les fabricants de cartes graphiques, qui ciblent d'habitude les domaines du jeux et de la visualisation, étendent depuis quelques années leurs gammes avec des produits dédiés au calcul dont certains modèles sont dépourvus de sorties vidéo. Ainsi en Juin 2007, NVIDIA a marqué son entrée dans le monde du calcul haute performance en lançant la gamme Tesla dédiée au calcul généraliste.

2. Le concept d'accélérateur matériel existait déjà dans les années 1980s, voir par exemple les produits du fabricant Floating Point Systems

Dans la suite sont détaillés les motivations de cet essor, les évolutions matérielles et logicielles qui y sont sous-jacentes, ainsi que les limitations actuelles et les évolutions futures du calcul sur les processeurs graphiques.

Le GPGPU est motivé aussi bien par les performances que peuvent offrir actuellement les cartes graphiques comme accélérateurs matériels que par le mouvement général des technologies de calcul vers le parallélisme. En effet, les GPU offrent une alternative intéressante aux architectures multi-cœurs. Alors que ces dernières utilisent un gros grain de parallélisme, des processus relativement lourds, et une performance optimisée par processus (pipelines, prédiction de branchement, etc.), les architectures graphiques reposent sur du parallélisme à grain très fin, des processus légers et une faible optimisation par processus. Quelques chiffres suffisent alors pour se convaincre du potentiel des architectures graphiques récentes. Il suffit ainsi de comparer la performance crête en virgule flottante (simple précision) du NVIDIA GTX 580 : 1581 GFlops, ou encore de l'AMD Radeon HD 6970 : 2703 GFlops, avec la centaine de GFlops offerte par les CPU les plus récents. Pour ce qui est de la bande passante mémoire, les cartes graphiques dépassent également tous les standards des CPU en approchant pour certains modèles les 250 Go/s (Kepler K20x). Il faut cependant signaler que la performance crête n'est qu'une indication. Dans la pratique, l'efficacité des GPU est souvent moindre que celle des CPU. Par exemple, un produit de matrices en simple précision (SGEMM) peut facilement exploiter 80% de la performance crête sur CPU mais dépasse rarement les 40% sur GPU [11]. Il est également nécessaire de souligner que les performances qu'elles soient théoriques ou soutenues doivent également être pondérées par la consommation électrique, le coût à l'achat et à l'utilisation, l'effort de programmation, etc. Pour plusieurs applications, et en prenant en compte tous ces éléments, la comparaison est en faveur des GPU. Une large communauté de scientifiques s'est ainsi intéressée à l'utilisation des cartes graphiques avec des succès plus ou moins avérés.

Avant d'émerger récemment comme une plate-forme de calcul performante, le seul objectif des cartes graphiques a été pendant longtemps l'affichage de scènes et d'images de plus en plus riches et complexes. Ceci explique leur conception, tirant parti au mieux du parallélisme de données et de flux inhérent à ce type d'applications et réduisant les possibilités de contrôle. Les premiers algorithmes généralistes portés sur cartes graphiques ont dû s'adapter à cette donne.

Constatant les performances obtenues et conscients du potentiel, les fabricants de processeurs graphiques ont imaginé de nouvelles solutions pour faciliter la programmation de ces processeurs et les adapter à ce genre d'applications. Cependant, tous les modèles de programmation existants restent fortement liés à cet héritage. Une exploitation poussée des possibilités des GPU passe donc obligatoirement (jusqu'à maintenant du moins) par une connaissance, fût-elle basique, de l'architecture et de la programmation graphiques.

Nous proposons dans cette partie une brève introduction aux architectures graphiques et à leur modèle de programmation classique. Nous verrons ensuite comment a été exploité ce modèle pour faire du calcul généraliste. Nous verrons enfin comment évoluent les architectures graphiques et les outils qui leur sont associés pour permettre notamment de faire de la programmation généraliste plus naturellement.

3.2.1 Architecture

Historique : pipeline graphique et architecture unifiée

Tous les GPU modernes organisent leurs traitements graphiques selon une structure en étages appelée pipeline graphique. Cette structure permet une implémentation matérielle efficace mettant en œuvre deux types de parallélisme : le parallélisme de tâches, en exploitant les différents niveaux du pipeline, et le parallélisme de données puisque chaque niveaux est composé de plusieurs unités de traitement SIMD.

Le pipeline reçoit en entrée du processeur hôte un ensemble d'instructions et de données décrivant une scène en trois dimensions. Il s'agit principalement de sommets de formes géométriques, appelés dans le jargon graphique *vertices* (pluriel de vertex), ainsi que d'informations sur la lumière, la position du point de vue, etc. En sortie du pipeline, on retrouve l'image en 2 dimensions telle qu'elle doit être affichée sur un écran. On retrouvera une description détaillée du pipeline graphique dans [109]. Ce pipeline se compose essentiellement des étapes suivantes :

1. Traitement des sommets. Les sommets subissent des transformations de position, d'échelle et de couleur pour être projetés d'un espace en 3D sur un plan en 2D. Les objets subissent des rotations, des translations et des mises à l'échelle pour être correctement placés parmi les autres objets de la scène. Les mêmes types de transformations sont ensuite appliqués pour placer les objets correctement par rapport à la position de la caméra et la direction de prise de vue. Le modèle d'éclairage est pris en compte pour attribuer une couleur aux sommets. Cette partie du pipeline offrant la possibilité d'être programmable, des modifications plus complexes peuvent être appliquées à l'aide de programmes appelés *vertex shaders*.
2. Rasterizer. Cette étape permet de déterminer les pixels appartenant aux formes déterminées par les sommets. Pour chacun de ces pixels, le *rasterizer* crée les informations nécessaires à leur traitement dans les étapes suivantes, par exemple : couleur, position, coordonnées de texture (ce qui permet de charger le cache de texture en avance), etc.
3. Traitement des pixels. Lors de cette étape, une couleur est attribuée à chaque pixel traité. Cette valeur peut être déterminée par interpolation, lecture en texture, ou de façon plus complexe via un programme appelé *fragment shader*.

Sur les implémentations matérielles qui ont prévalu jusqu'en 2006, on retrouve les étapes du pipeline implémentées dans des composants hardware différents du GPU avec des unités de calcul séparées. La Nvidia GeForce 7800 (sortie en 2005) est l'une des dernières représentantes de ce type d'architecture. On y retrouve notamment des *vertex processors* effectuant le traitement des sommets. Il s'agit d'unités de calcul SIMD pouvant accéder aux textures et capables d'effectuer un opération MADD (myltiply-add) par cycle d'horloge. Ils sont au nombre de 8. On y retrouve également les *fragment processors*, au nombre de 24, effectuant le traitement des pixels. Il s'agit là aussi d'unités de calcul SIMD pouvant accéder aux textures, via le *texture processor* et capables de traiter quatre pixels (appelés quad) en même temps. Ces processeurs peuvent traiter des

centaines de pixels à la fois, ce qui permet de recouvrir la latence inhérente à l'accès aux textures. Ces deux unités, effectuant à leurs débuts des opérations automatiques câblées "en dur", sont devenues au cours des dernières années de plus en plus programmables. C'est l'introduction de langages permettant de spécifier les instructions à exécuter par chaque processeur qui a donné naissance au GPGPU. On retrouvera dans [110] une description des techniques utilisées pour adapter les calculs généralistes à ce genre de plateforme ainsi qu'un état de l'art des applications portées sur GPU selon cette approche.

L'approche précédemment décrite, dédiant un certain nombre fixe d'unités spécialisées à chaque étape de transformation graphique, et outre le fait qu'elle soit intimement liée à une vision graphique du calcul ainsi qu'à une connaissance des langages et des architectures des GPU donc peu adaptée à faire des calculs généralistes, présente certains inconvénients même pour la programmation graphique.

En effet, d'une image à une autre, la complexité géométrique et donc le nombre de sommets à traiter, varie faisant ainsi varier le ratio vertices/pixels. En fonction de l'image, le goulot d'étranglement peut être situé au niveau des *vertex processors* ou des *fragment processors*. Les architectures graphiques introduites dès 2006 ont été pensées pour palier à cette limitation en introduisant des unités moins spécialisées, pouvant traiter aussi bien les sommets que les pixels. On parle alors d'architectures unifiées.

Les premiers GPU NVIDIA basés sur l'architecture unifiée sont les G80. Cette architecture ainsi que les outils logiciels de calcul généraliste qui lui sont associés, fût baptisée CUDA (*Compute Unified Device Architecture*). La figure 3.4 présente schématiquement cette architecture qui est composée de 16 multiprocesseurs appelés SM (pour *Streaming Multiprocessor*) composés chacun de 8 unités de calcul généralistes appelées SP (*Stream Processors*) qui effectuent toujours une même opération à la manière d'une unité SIMD, et de 2 unités de calcul spécialisées appelées SFU (*Super Function Unit*) dédiées aux fonctions transcendantes (cos, sin, etc.) et autres calculs particuliers. Les SM sont regroupés par paires au sein d'un TPC (Texture Processor Cluster) également muni d'un niveau 2 de cache d'instructions et de données. Ce type d'architecture, outre son intérêt pour les traitements graphiques, est mieux adapté à la programmation généraliste des GPU puisqu'il autorise une vision indépendante du pipeline graphique.

Evolution des architectures graphiques

L'introduction des architectures unifiées constitue le véritable point de départ de l'essor de la programmation généraliste de cartes graphiques. Cette architecture a rapidement évolué pour enrichir les GPU de possibilités de calcul réservées jusqu'alors aux CPU. Un numéro sous la forme $x.y$ appelé *compute capability* est associé à chaque modèle de GPU NVIDIA et permet de décrire les possibilités matérielles qu'il implémente. Ainsi les G80 étaient naturellement de *compute capability* 1.0 et la première mise à jour importante avec les processeurs GT200 ou T10 (introduits en 2008) portait le numéro 1.3. Cette mise à jour a permis notamment l'introduction du calcul en double précision. Les processeurs T10 peuvent être vus comme un ensemble de 30

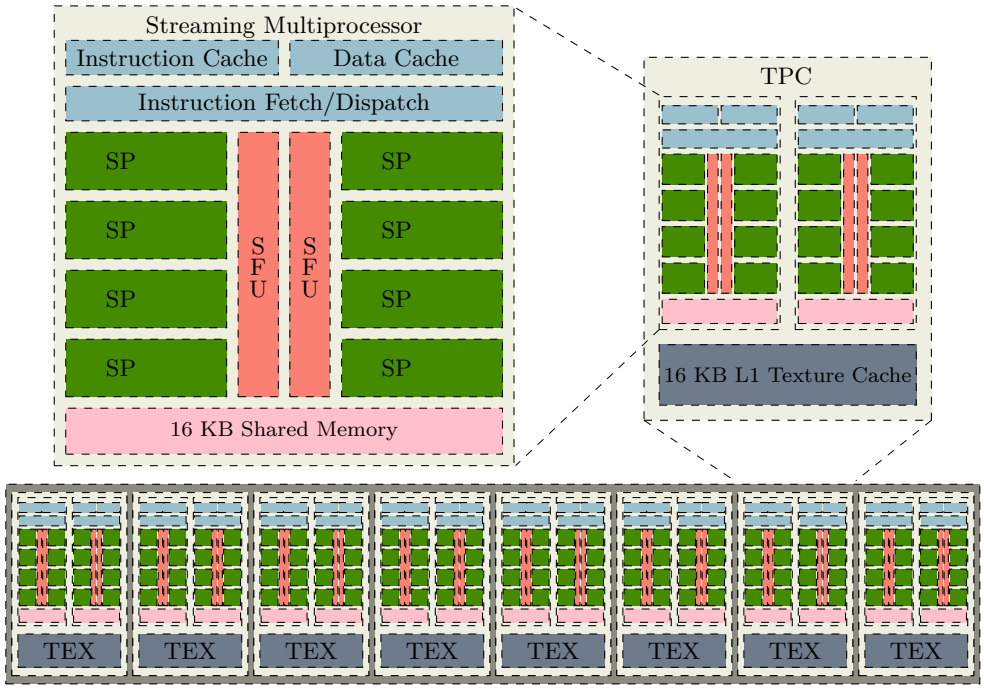


FIGURE 3.4 – Diagramme des processeurs NVIDA G80.

SM regroupés par trois au sein de *Thread Processor Array* (TPA). Comme pour la G80, chaque multiprocessor regroupe 8 SP et 2 SFU. La différence majeure vient du rajout d'une unité de calcul en double précision.

Chacun des SP peut exécuter jusqu'à 3 opérations flottantes en simple précision par cycle d'horloge. Avec une fréquence de 1.44 GHz (celle des cartes T10 des S1070 qui seront utilisées par la suite), un tel GPU peut donc fournir une puissance crête de $30 \times 8 \times 3 \times 1.44 = 1$ TFlops. Cette puissance de calcul est divisée par 8 pour le calcul en double précision (elle est divisée par 12 si on tient compte du fait que les SP peuvent exécuter 3 instructions/cycle alors que les unités 64bits ne peuvent en exécuter que 2).

Les unités de calculs regroupées au sein d'un SM ont accès à 16.384 registres 32 bits ainsi qu'à 16 Ko de mémoire partagée qui peut être vue comme un cache L1 (géré explicitement par le développeur) avec un temps de latence réduit (4 cycles d'horloge par accès s'il n'y a pas de conflits). Tous les SP ont également accès à une mémoire globale hors puce, de taille allant jusqu'à 4 Go, mais avec un temps de latence plus important (de 400 à 600 cycles d'horloge). La mémoire globale a une bande passante relativement importante. Pour les T10, chaque GPU est assorti de 4Go de GDDR3 cadencés à 792 MHz avec une interface de 512 bits. La bande passante théorique maximale est ainsi

$$(512/8) \times 2 \times 792.10^6 \times 10^{-9} \approx 102 \text{ Go/s}.$$

Cependant, comme nous le verrons par la suite, des motifs d'accès particuliers doivent être respectés pour tirer profit au mieux de cette bande passante (voir Chapitre 4).

En 2010, NVIDIA a introduit une nouvelle génération de GPU dont l'architecture fût baptisée Fermi (*compute capability* 2.x). Parmi les améliorations introduites, on notera l'augmentation des performances en double précision (désormais pour les modèles haut de gamme, un rapport 2 et non plus 8 existe entre performances en simple et en double précision, les mêmes unités pouvant traiter les deux types d'opérations à une fréquence réduite de moitié), l'introduction de niveaux de cache L1 et L2 pour optimiser les accès à la mémoire globale ainsi que l'apparition de la correction ECC pour la mémoire (GDDR5) considérée par beaucoup comme préalable indispensable pour développer des applications fiables sur GPU. Les modèles les plus récents intègrent jusqu'à 512 SP, regroupés par groupes de 32 au sein des SM, et annoncent une performance crête de 1.5 TFlops en simple précision (la moitié en double). Avec Fermi NVIDIA a également modifié le mode d'adressage mémoire en passant d'un espace d'adressage physique en 32-bit, ce qui limité la RAM adressable à 4 Go, à un espace d'adressage virtuel 64-bit et un adressage physique 40-bit ce qui autorise l'intégration de quantités plus importantes de mémoire (jusqu'à 6 Go actuellement et 12 Go dans un futur proche). Cet adressage virtuel a ouvert la voie à l'espace d'adressage unifié partagé entre CPU et GPU qui sera introduit avec la version 4 de CUDA. Un espace d'adressage unifié a entre autres avantages de permettre de traiter des données de plus grandes tailles (à partir de la mémoire de l'hôte), de permettre des transferts de données entre GPU ainsi que d'effectuer des entrées/sorties directement à partir de la mémoire GPU sans utiliser de copies intermédiaires. Nous invitons le lecteur à se reporter à [103, 104, 131] pour une description plus détaillée des évolutions introduites avec l'architecture FERMI.

Kepler est l'architecture la plus récente des GPU NVIDIA (*compute capability* 3.x). Introduite en 2012, elle intègre jusqu'à 2688 SP regroupés par groupes de 192 au sein des *Stream Multiprocessors* rebaptisés SMX, pour une puissance crête en simple précision de près de 4 TFlops. L'évolution principale de cette génération porte sur le modèle de programmation qui devient plus généraliste. A titre d'exemple, le *Dynamic Parallelism* permet désormais d'appeler à partir de noyaux de calcul GPU d'autres noyaux de calculs, ce qui facilite l'utilisation d'algorithmes récursifs (raffinement de maillage adaptatif par exemple) et de bibliothèques de noyaux de calculs GPU.

Exemple de plateforme GPGPU : NVIDIA Tesla S1070

Avec la popularité croissante du concept de GPGPU, NVIDIA a développé une gamme complète d'accélérateurs dédiés au calcul. Dans le cadre de cette thèse, nous avons eu accès à un cluster équipé de lames GPU : les Tesla S1070.

La figure 3.5 représente une vue schématique de deux nœuds de ce cluster. Une lame NVIDIA S1070 intègre 4 processeurs graphiques T10 assortis de 4Go de mémoire chacun. Chaque paire de GPU est connectée à un nœud hôte par le biais d'un bus PCIe 2.0 x16. Ce type de connexion permet une bande passante maximale de 8 Go/s dans chaque direction. En pratique nous avons

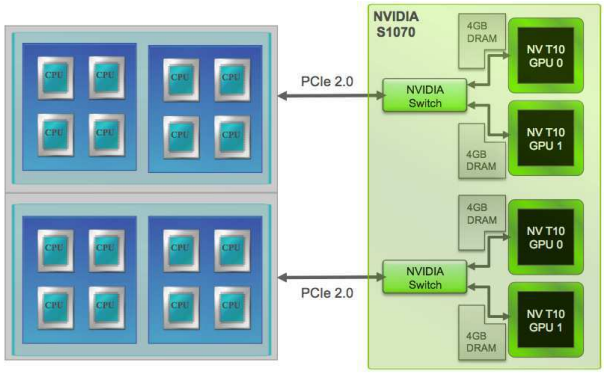


FIGURE 3.5 – Vue schématique de deux nœuds de calcul du cluster utilisé.

pu observer une débit soutenu allant jusqu'à 5.6 Go/s lorsqu'on utilise les accès DMA (voir chapitre suivant). Ce débit est réduit à 3.3 Go/s lorsqu'on utilise deux GPU partageant le même bus simultanément. Cette bande passante limitée (comparée à la bande passante mémoire de 102 Go/s), et le temps de latence important qui lui est associé (16 μs) font que les transferts de données entre GPU et hôte peuvent constituer le goulot d'étranglement principal pour les problèmes de taille limitée ou nécessitant peu de calcul.

3.2.2 Modèles de programmation

Différents choix s'offrent aux développeurs pour tirer profit de la puissance de calcul des GPU dans le cadre du développement d'outils généralistes (non graphiques) avec des degrés de complexité et d'intrusivité dans le code distincts :

- programmation directe avec des langages bas niveaux type CUDA ou OpenCl ;
- optimisation à l'aide de directives de compilation avec HMPP ou OpenAcc ;
- appel de bibliothèques externes optimisées pour les GPU telles que CUBLAS ou CUFFT.

Nous passons en revue ces différents choix dans les paragraphes suivants.

Langages de programmation et API dédiés

C'est la voie la plus complexe mais également celle qui permet d'exploiter au mieux la puissance des GPU grâce à des langages adaptés à leur architecture particulière. CUDA a été le premier langage du genre. Dans un effort de standardisation OpenCl est apparu ensuite.

CUDA Pour permettre aux développeurs non rompus à la programmation graphique d'exploiter les puissances de calcul offertes par les GPU, NVIDIA a introduit la couche d'abstraction CUDA. Ce modèle de programmation est basé sur une approche SPMD : le programmeur écrit une portion de code (le *kernel*) qui sera exécutée par plusieurs *threads* en parallèle en utilisant des données différentes. CUDA définit une hiérarchie de *threads* illustrée sur la figure 3.6 dans le but d'organiser les calculs selon une certaine topologie. Ainsi les *threads* sont regroupés au

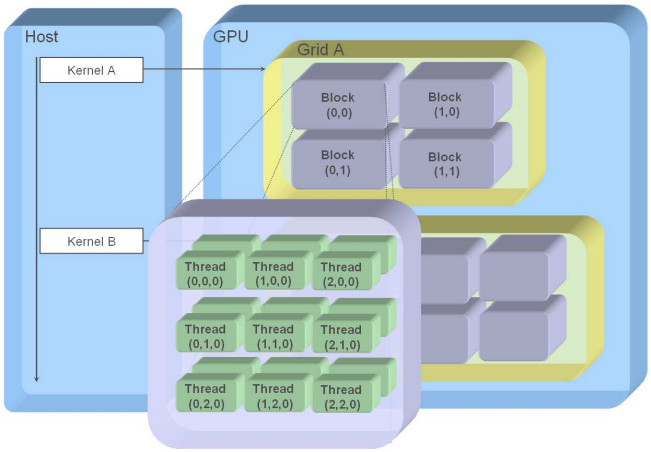


FIGURE 3.6 – Modèle de programmation CUDA : des blocs de *threads* organisés en grilles.

sein de blocs de *threads*(*thread blocks*) qui peuvent être de dimensionnalité un, deux ou trois. Ces blocs sont à leur tour organisés en grilles 1D ou 2D. Cette organisation fait le lien entre l'exécution des *threads* et l'architecture matérielle des GPU. En effet, les *threads* appartenant au même bloc s'exécuteront sur le même SM, ce qui permet d'introduire des mécanismes de partage de données et de synchronisation. Les *threads* seront regroupés et traités par *warps* de 32 *threads*. Ce regroupement, qui n'était qu'une abstraction matérielle, permet depuis les GPU de *compute capability* 1.2 d'introduire des mécanismes de vote entre *threads* d'un même *warp*. Sur les architectures où les SM sont composés de 8 SP, le traitement d'un *warp*se fait en 4 cycles d'horloge en SIMD. Plus précisément, NVIDIA préfère utiliser l'acronyme SIMT (*Single Instruction Multiple Threads*) pour décrire le modèle de fonctionnement des GPU. En effet, dans un modèle SIMD classique, tel que pour les instructions SSE des CPU, le programmeur (ou le compilateur) assemble les données (les vectorise) pour exécuter les mêmes traitements avant de les désassembler ensuite. De ce fait, il n'est pas possible de différencier les traitements en fonction des données. Dans le cas SIMT, il est possible d'effectuer cette différenciation en désolidarisant les *threads* lors de leur exécution. On parle alors de *threads* divergents. Dans le cas de divergence de *threads* d'un même *warp*, les traitements sont exécutés en série, ce qui induit une perte de performance. Les GPU avant Fermi ne permettaient de gérer qu'une seule grille de *threads* (et donc un seul noyau de calcul) à la fois. Avec Fermi a été introduite la possibilité de gérer plusieurs grilles (jusqu'à 16 actuellement et autant de noyaux) simultanément. Ceci a pour effet d'augmenter les possibilités de recouvrement entre calculs et accès aux données.

Différents niveaux de mémoire existent sur la carte graphique. Ces niveaux diffèrent selon qu'ils sont accessibles en écriture ou non par les *threads*, mis en cache ou non, visibles par tous les *threads* ou pas. La Figure 3.7 illustre schématiquement les différents espaces disponibles et les possibilités d'accès et de partage qui y sont associées. Les flèches indiquent les sens des accès possibles (par exemple accès en lecture seule à la mémoire constante de façon cohérente entre tous les *threads* et accès en lecture et en écriture aux registres privés pour chaque *thread*).

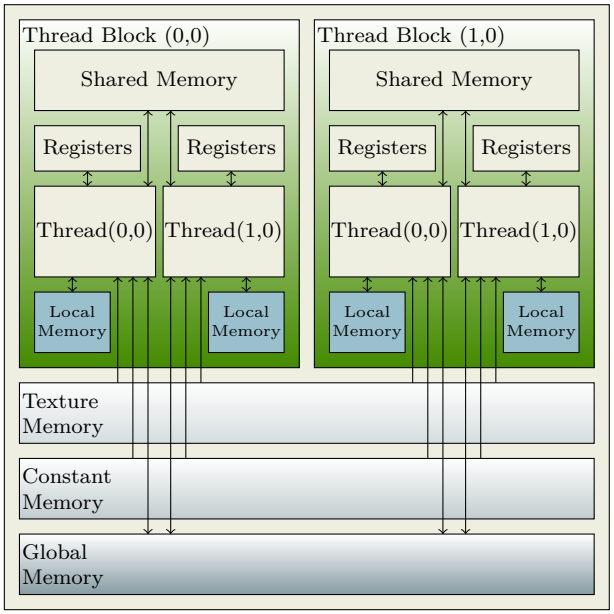


FIGURE 3.7 – Hiérarchie mémoire selon le modèle de programmation CUDA.

CUDA	OpenCL
Streaming Multiprocessor (SM)	Compute Unit (CU)
Streaming Processor (SP)	Processing Element (PE)
Shared Memory	Local Memory
Kernel	Kernel
Thread	Work item
Thread block	Work group

TABLE 3.1 – Equivalences entre les terminologies CUDA et OpenCL.

OpenCL L'effort de développement de OpenCL initié par Apple a été placé sous la houlette du Khronos Group (gérant également le standard OpenGL). OpenCL vise à offrir une plateforme de développement commune pour la variété d'architectures parallèles existantes. Il existe ainsi des implémentations d'OpenCL pour les accélérateurs graphiques d'AMD et de NVIDIA ainsi que pour les processeurs généralistes x86. Divers projets d'implémentations visant les FPGA existent également. OpenCL peut être vu comme le pendant standardisé et multi-plateformes de CUDA. En effet, il existe des similitudes marquées entre les deux plateformes, aussi bien au niveau de l'architecture que du modèle de programmation, seule une différence de vocabulaire faisant la différence. Le tableau 3.1 reprend certaines équivalences entre concepts CUDA et OpenCL. Nous proposons en Annexe un comparaison complète entre les implémentations en CUDA et en OpenCL d'un solveur 2D de l'équation d'onde.

Directives de compilation : HMPP et OpenAcc

Pour faciliter la tâche du programmeur désirant tirer parti des possibilités des GPU sans pour autant faire de la programmation graphique ou être lié à un type particulier de cartes (ou même d'accélérateurs), sont apparus des outils de plus haut niveau permettant de déléguer la génération de code adapté aux accélérateurs au compilateur suivant des annotations du programmeur formulées sous la formes de directives de compilation. HMPP, développé par la société CAPS, et OpenAcc, standard émergeant développé par CAPS, NVIDIA, PGi et Cray, sont deux implémentations de ce modèle.

Au cours de nos travaux, nous avons utilisé HMPP (*Hybrid Multicore Parallel Processing*) pour développer certaines des applications qui seront décrites dans la seconde partie de ce manuscrit. HMPP propose une ensemble d'outils permettant de gérer, et éventuellement de générer, du code pouvant s'exécuter sur divers types d'accélérateurs (SSE, CUDA ou encore OpenCL dans sa version la plus récente) par le biais principalement d'annotations introduites dans le programme d'origine. HMPP est une solution séduisante à plus d'un titre. Tout d'abord, l'utilisation des annotations (#pragma à la OpenMP) permet de garder l'intégrité du code d'origine qui pourra toujours être compilé et exécuté sur sa forme initiale. Ensuite le support d'exécution offert par HMPP permet de gérer dynamiquement la présence d'accélérateurs matériels et d'exécuter le cas échéant le code disponible sur la plateforme la plus adaptée (possibilité d'utiliser des conditionnelles sur la taille des données par exemple). C'est cette facette que nous avons le plus exploitée dans le cadre de ces travaux. Au cours de l'avancement de cette thèse, le générateur HMPP permettant de générer le code GPU a gagné en efficacité, ce qui en a fait une alternative intéressante à l'écriture de code CUDA.

OpenACC adopte une approche très similaire à celle de HMPP. A la différence de HMPP, OpenACC est cependant un standard supporté actuellement par les compilateurs CAPS, PGI et Cray. Des efforts sont actuellement menés pour intégrer ce standard au sein d'OpenMP.

Bibliothèques GPGPU

Diverses implémentations GPU de bibliothèques de calcul largement utilisées en calcul scientifique permettent aux développeurs d'exploiter la puissance de calcul des GPU sans effort de programmation particulier, en substituant simplement les appels à ces bibliothèques aux appels aux bibliothèques standards et en gérant éventuellement les transferts de données entre hôte et accélérateur. On citera à titre d'exemple cuBLAS [101], cuSparse [105] et MAGMA [1] pour l'algèbre linéaire, cuFFT [102] pour le calcul de transformées de Fourier, *NVIDIA Performance Primitives* [107] pour le traitement du signal et des images, etc.

3.2.3 Ecosystème GPGPU

Avoir des langages pour programmer les GPU est une chose, encore faut-il disposer d'un environnement de développement complet et d'une intégration poussée des GPU au sein des systèmes existants pour pouvoir développer des applications accélérées efficaces. Nous décrivons

brièvement dans ce qui suit l'outillage disponible pour la programmation GPGPU et l'intégration des GPU au sein des systèmes.

Outils CUDA

CUDA définit une architecture matérielle, mais il s'agit en pratique principalement de la partie logicielle proposée par NVIDIA pour tirer parti de ses GPU et effectuer du calcul généraliste. La partie logicielle de CUDA se compose d'un driver, d'un support d'exécution (*runtime*), de bibliothèques de calculs sur GPU (CUBLAS, CUFFT, etc.), d'une API basée sur une extension du langage C ainsi que du compilateur NVCC. NVIDIA propose en outre plusieurs outils pour assister les développeurs, on citera notamment la présence d'un débuggeur et d'un profiler CUDA.

Outils tiers

Que ce soit pour le débuggage (TotalView, Alinea, etc) ou le profilage (Tau, Vampir, etc) les outils habituellement utilisés par les développeur d'applications de calcul scientifique assurent désormais le support des GPU (ce support se limite le plus souvent aux GPU CUDA).

D'autres outils permettent d'exploiter efficacement les GPU. A titre d'exemple, StarPU [13] est un support exécutif permettant de faciliter un ordonnancement efficace des tâches à exécuter sur les différentes ressources de calcul disponibles, notamment les GPU. StarPU est notamment utilisé par la bibliothèque d'algèbre linéaire MAGMA.

Certains projets visent quant à eux à étendre l'utilisation des GPU à d'autres langages. JCUDA [137], RootBeer [111] pour Java, PyCUDA et PyOpenCl [74] pour Python en sont des exemples.

Intégration aux systèmes

Conscients de la limitation importante que constitue l'utilisation du PCIe pour les transferts entre le GPU et le reste du système (mémoire de l'hôte, autres GPU, cartes réseau pour les transferts vers les autres hôtes, E/S vers les disques, etc), les fabricants cherchent à optimiser les mécanismes de ces communications. GPUDirect est la solution proposée par NVIDIA. Cette solution permet d'éliminer le surcoût CPU induit par les copies mémoire en utilisant les transferts de type DMA et RDMA. La première mouture, introduite en 2010, permet d'accélérer les transferts entre le GPU et le réseau Infiniband et a été supportée par les fabricants de cartes réseaux Mellanox et QLogic. La version actuelle étend ce type de transferts aux communications entres cartes graphiques (pair à pair ou P2P) et permet l'accès direct dans un noyau de calcul s'exécutant sur GPU à la mémoire d'un GPU voisin.

3.2.4 Vers la fusion CPU-GPU : naissance des APU

On assiste aujourd'hui au rapprochement des CPU et des GPU. Tout d'abord intégrés au sein du même socket, les solutions actuelles s'orientent vers une intégration plus poussée au sein du même circuit intégré pour éliminer complètement le problème que représente le coût des communications entre processeurs hôtes et accélérateurs graphiques avec un accès partagé à la mémoire. Annoncé depuis 2006 sous le nom de code Fusion, les premiers APU (*Accelerated Processing Units*) AMD ont fait leur apparition en 2011, visant tout d'abord le marché des périphériques nomades, leur intégration dans les systèmes HPC est effective depuis 2013. Les solutions actuelles continuent a disposer cependant d'une partition mémoire distincte pour le GPU [27]. Intel avec Sandy Bridge fait le même chemin. Nvidia suit également la même inflexion. L'intégration depuis CUDA 4.0 d'un espace mémoire adressable globalement (UVA pour *Unified Virtual Addressing*) en témoigne. L'unification de l'espace d'adressage permet de déléguer au système la distinction entre adresses mémoire CPU ou GPU ce qui permet de faire des copies de données entre des zones mémoire sans avoir à spécifier explicitement si ces données se trouvent sur CPU ou sur GPU et d'accéder à partir du GPU à la partie non paginée (*pinned memory*) de la mémoire de l'hôte sans copies explicites (*zero copy access*). Avec Maxwell, la prochaine génération de GPU annoncée pour 2014, NVIDIA promet d'aller plus loin en permettant, tout comme l'APU Kaveri d'AMD, d'accéder virtuellement au même espace mémoire à partir du CPU et du GPU (UVM pour *Unified Virtual Memory*). Des mécanismes matériels permettront de maintenir la cohérence entre les données accessibles depuis le CPU et le GPU.

3.3 Autres accélérateurs

3.3.1 FPGA

Les FPGA (*Field Programmable Gate Arrays*) appartiennent au monde des circuits logiques programmables (*Programmable Logic Devices* ou PLD) qui constituent avec les ASIC (*Application Specific Integrated Circuit*) l'ensemble des circuits spécifiques à des applications. Alors que les ASIC sont produits par les vendeurs, les PLD offrent la possibilité d'être programmés sur site. Les FPGA offrent de plus la possibilité d'être reconfigurables. Les FPGA ont été introduits par Xilinx en 1985 et sont fabriqués aujourd'hui principalement par Altera et Xilinx. L'utilisation de ces circuits a changé les méthodes de conception des systèmes logiques en permettant d'adapter les circuit au système à concevoir, avec d'un côté un coût et une taille réduits, de l'autre plus de fiabilité, de performance et de flexibilité. Les FPGA se distinguent des autres circuits par un fort degré d'intégration, ce qui a permis d'étendre leur application à de larges domaines. L'un de ces domaines est celui de leur utilisation comme accélérateurs de calcul. Derrière ce concept, une idée simple : au lieu d'adapter les algorithmes de calcul aux architectures figées des processeurs et autres accélérateurs, pourquoi ne pas re-créer dynamiquement un processeur adapté à chaque algorithme pour en exhiber le maximum de parallélisme ? On parle alors de calcul reconfigurable. Les FPGA offrent cette possibilité.

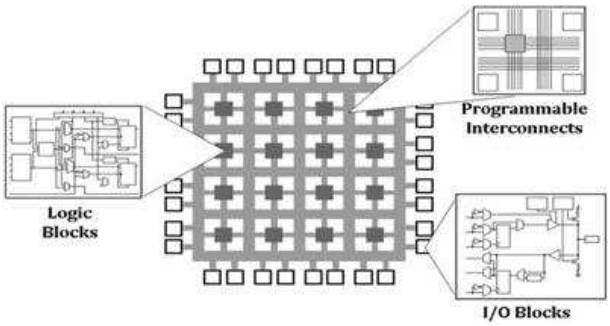


FIGURE 3.8 – Vue schématique d'un FPGA : une grille 2D reconfigurable d'unités logiques et de liens ainsi que des unités d'E/S.

Cependant, les FPGA fonctionnent à des fréquences limitées par rapport aux CPU actuels. En effet la fréquence des FPGA se compte en quelques centaines de MHz alors que les CPU récents peuvent dépasser les 4 GHz. Seule l'exploitation poussée du parallélisme permet alors de compenser cette limitation en terme de fréquence. En effet, les FPGA peuvent exécuter des milliers d'opérations par cycle d'horloge.

Leur utilisation a ainsi permis d'accélérer des applications dans divers domaines : en cryptologie et en filtrage par motifs pour lesquels ils s'avèrent particulièrement adaptés, en traitement d'images [120], en imagerie sismique [24, 53, 97] ainsi qu'en imagerie médicale. Nous passons en revue l'architecture générale des FPGA ainsi qu'un exemple de solution matérielle pour les utiliser pour le calcul haute performance. Nous verrons ensuite le modèle de programmation des FPGA.

Architectures

Chaque fabricant de FPGA propose une architecture particulière. Cependant, le schéma général est le même pour tout FPGA : une grille bidimensionnelle d'unités logiques reconfigurables (plusieurs dizaines de milliers) interconnectées par un réseau de liens reconfigurable également. Au sein de la grille on retrouve par ailleurs des unités d'E/S et des blocs mémoire. Le schéma d'architecture générale d'un FPGA est illustré sur la figure 3.8.

Les unités logiques des FPGA sont souvent implémentées par des tables de correspondance (*Look Up Table* ou LUT). Un FPGA récent, tel que le Virtex 6 de Xilinx, peut contenir plus de 360 mille LUT. Avec ces éléments reconfigurables interconnectés, il devient possible d'implémenter tout circuit électronique pourvu de disposer de ressources suffisantes.

Intégration : exemple du Convey HC-1

Comme cité précédemment, les principaux fabricants de FPGA sont Xilinx (gamme Virtex) et Altera (gamme Startix). Cependant, dans le monde du calcul scientifique, on connait les intégrateurs proposant des solutions complètes à base de FPGA plutôt que les fabricants

de ces composants. Chaque intégrateur propose un architecture spécifique avec une suite logicielle permettant de développer des applications accélérées par les FPGA. En effet, l'utilisation de langages de description de matériel (ou HDL pour *Hardware Description Language*), habituellement utilisés pour la conception de circuits électroniques, devient vite fastidieuse pour le développement d'applications scientifiques. Les outils et les approches différant largement d'un intégrateur à un autre sans compatibilité, il en résulte un lien fort entre l'application développée et la plateforme sous-jacente. Des initiatives telles que OpenFPGA [2] visent à proposer des approches et des outils standards pour les FPGA. Mais on reste actuellement loin de voir un standard unique s'imposer.

Parmi les solutions disponibles, on peut citer celles proposées par SRC avec les MAPstation, SGI et les serveurs RASC, la gamme MPC Maxeler ainsi que les solution Convey HC. C'est la version HC-1 de cette dernière solution qui a été évaluée au cours de notre étude.

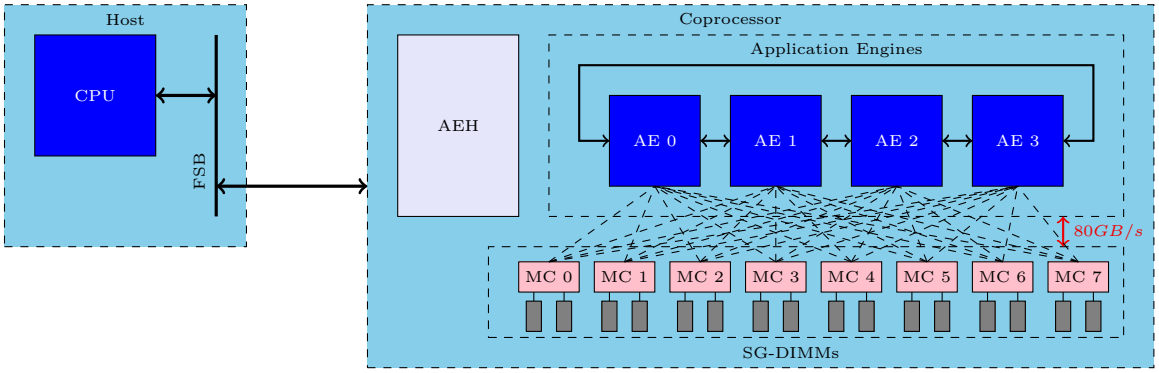


FIGURE 3.9 – Vue schématique du système Convey HC-1.

La plateforme Convey HC-1, annoncée en 2008, se compose d'un système x86 standard auquel vient se greffer le coprocesseur Convey directement via le socket CPU comme illustré sur la figure 3.9. Le coprocesseur est composé de 14 FPGA Virtex 5 : 4 LX330 dédiés au calcul appelés *Application Engines* ou AE, 8 LX155 utilisés comme contrôleurs mémoire, et 2 LX110 pour gérer transferts de données entre hôte et coprocesseur et les divers mécanismes de contrôle (*Application Engine Hub* ou AEH).

La performance crête annoncée des AE se situe entre 100 et 200 GFlops pour une consommation approximative de 450W. Cette performance dépend du profil choisi. Le concept de profil ou *personality* est introduit par Convey pour adapter les FPGA à une classe d'algorithme plutôt qu'à un algorithme en particulier. Parmi ces *personnalités* on peut citer : *Financial Analytics, SPvector, DPvector*, etc. Le choix du profil fige les FPGA à une architecture et un jeu d'instructions adaptés à un type d'algorithme donné. Le développeur peut en outre créer ses propres profils.

Pour le profil *SPvector* (profil par défaut pour le calcul en simple précision), chacun des 4 AE est divisé en 8 *Function Pipes*, chacun pouvant exécuter 4 opérations d'addition/multiplication (*fused multiply/add* ou fma) et 2 additions à chaque cycle d'horloge (3,3 ns). La performance crête est alors :

4 AE x 8 FP x 10 flop / 3,3 ns = 97 GFlops.

L'AEH est un élément central du coprocesseur. Il constitue l'interface avec le processeur hôte et le chipset d'E/S Intel, lit et décode les instructions, exécute les instructions scalaires et délègue l'exécutions des instructions étendues aux AE.

Pour ce qui est de la mémoire, le système HC-1 auquel on a eu accès est équipé de 32Go de mémoire côté hôte, et de 16Go côté coprocesseur. Parmi les 32Go de la mémoire de l'hôte, 16 Go sont associés à la mémoire du coprocesseur. Seules 16Go restent alors disponibles pour les calculs non accélérés. Côté coprocesseur, les AE accèdent à la mémoire via 8 contrôleurs mémoire (MC). Les liens entre chaque AE et MC offrent un débit de 2.5 Go/s. La bande passante totale maximale est alors de 80 Go/s ($8 \times 4 \times 2.5$).

La programmation de la plateforme Convey HC-1 se fait en annotant le programme d'origine avec des directives de compilation qui permettent de spécifier les parties du code à vectoriser et de gérer les mouvements de données. En particulier, les régions entre les directives `#pragma cny begin_coproc / cny end_coproc` sont identifiées comme devant s'exécuter sur le coprocesseur.

Traitement par flux

Une des raisons pour lesquelles les CPU présentent des performances limitées pour certaines applications hautes performances réside dans le modèle de programmation séquentiel qui ne tire pas parti du parallélisme et du schéma de communication de ces applications. Dans cette section sera décrit le modèle de traitement par flux ou *stream processing* qui constitue la base de la programmation des FPGA.

Suivant le modèle de traitement par flux, toutes les données sont organisées sous la forme de flux ou *stream* de données, c'est à dire un ensemble ordonné de données de même type. Il s'agit ensuite de définir un ensemble d'instructions, ou *kernels* à exécuter sur toutes les données d'un flux.

Un *kernel* a comme entrées un ou plusieurs flux et fournit en sorties également un ou plusieurs flux. Le paradigme impose sur les kernels deux restrictions importantes :

- les sorties d'un kernel ne sont fonction que de ses entrées ;
- un calcul sur un élément d'un flux ne doit pas avoir de dépendances avec un calcul sur un autre élément.

Ces restrictions permettent d'exploiter le parallélisme de données d'un programme. Une application est construite selon ce paradigme, en chaînant plusieurs kernels, de façon à ce que les sorties de l'un correspondent aux entrées d'un ou de plusieurs autres, ce qui permet d'exploiter le parallélisme de flux de l'application.

3.3.2 Accélérateurs généralistes

Ces accélérateurs sont composés de plusieurs cœurs de calculs simplifiés interconnectés et viennent se greffer usuellement à un nœud de calcul classique via une connexion PCIe. Au

cours des dernières années, plusieurs accélérateurs généralistes ont fait leur apparition dans le monde du HPC avec des fortunes diverses. Ainsi le Cell d'IBM/Sony/Toshiba et le PetaPath de ClearSpeed ont connu leur heure de gloire avant de se voir abandonnés, essentiellement à cause de la difficulté de leur programmation. Actuellement, c'est le processeur MIC d'Intel, héritier du Larrabee, qui tient la dragée haute aux accélérateurs GPU grâce notamment à la simplicité annoncée de sa programmation. Après le *Knights Ferry*, premier prototype à 32 cœurs sorti en 2010, la première version commerciale du MIC a été le *Knights Corner*. Sorti en 2012 sous différentes déclinaisons, regroupées depuis sous le nom Xeon Phi et composées de 57 à 61 cœurs, c'est cette version qui équipe le supercalculateur Tianhe-2, premier au top500 depuis juin 2013 avec 33.86 PFlops. Malgré une compatibilité x86, les Xeon Phi nécessitent tout de même un effort de portage similaire à celui de la programmation GPGPU pour en tirer des performances décentes.

3.4 Positionnement de la thèse

Nous avons démontré que face aux besoins toujours croissants de l'exploration pétrolière en puissance de calcul à coût énergétique maitrisé, il existe une offre riche de solutions accélératrices pouvant répondre à ces besoins. Tout comme le passage de la programmation séquentielle à la programmation parallèle a constituée une évolution importante imposant des contraintes particulières auxquelles les développeurs ont dû s'adapter [124], le passage à un parallélisme plus poussé de type *many core*, GPGPU ou FPGA pose des contraintes nouvelles qu'il faut étudier. Nous proposons de mener cette étude dans le cadre des algorithmes de modélisation, de migration et d'interprétation sismiques. Les questions auxquelles nous essayons de répondre sont essentiellement celles liées à l'adéquation entre algorithme et architecture matérielle, aux langages, aux outils, et à la performance -dans ses différentes formulations- des applications développées pour chaque architecture. Pour répondre à ces questions nous étudions diverses implémentations d'applications liées au processus d'imagerie sismique et analysons leur performance pour diverses plateformes accélératrices.

Deuxième partie

Accélérateurs de calcul pour l'imagerie sismique

Chapitre 4

Modélisation et imagerie sismiques par équation d'onde

Sommaire

Dans les chapitres précédents ont été mis en exergue les besoins croissants des applications d'imagerie sismique en ressources de calcul. Pour atteindre une puissance pétaflopique et au-delà, ces ressources doivent être énergétiquement efficaces et assurer un passage à l'échelle convenable pour ce type d'applications. Dans ce contexte a été démontré l'intérêt que peut présenter l'utilisation des accélérateurs matériels, en particulier les GPU et les FPGA. Les questions de la granularité du parallélisme à exploiter, de l'utilisation conjointe de CPU et d'accélérateurs et de la démarche d'adaptation des algorithmes ont été posées. Dans la suite de ce manuscrit nous essayons, via des exemples concrets ayant un intérêt pratique avéré, d'esquisser une réponse à ces interrogations. Nous détaillerons la démarche d'adaptation ainsi que les principales difficultés qu'elle pose. Nous mettrons également en relief les limites actuelles qui peuvent s'opposer au développement de ces approches technologiques. concrets Les travaux et les résultats concernant l'utilisation des accélérateurs graphiques décrits dans ce manuscrit se basent sur une mise en œuvre en CUDA. Nous essaierons cependant de généraliser lorsque cela est possible les conclusions tirées aux autres types de plateformes.

4.1 Modélisation sismique par différences finies sur GPU

Pour évaluer et comprendre les performances d'une application accélérée par l'utilisation d'accélérateurs graphiques, NVIDIA a introduit avec CUDA un ensemble de nouveaux concepts et de nouvelles métriques. Nous commençons par introduire ces concepts en les illustrant par un exemple n'ayant pas un intérêt pratique énorme, mais qui a au moins le mérite de la simplicité : la modélisation sismique en 2D. Cette première étape nous permettra de nous focaliser sur le noyau de calcul. En effet, à l'instar d'une large partie des problèmes de calcul intensif, la résolution de l'équation d'onde par différences finies peut être décomposée en une partie calculatoire (noyau de calcul ou *kernel*) et une partie communications. Nous commençons par étudier le noyau de calcul, qui correspond à un calcul de type *stencil*, mis en œuvre pour la modélisation. Nous nous intéresserons ensuite à l'étude des communications qui peuvent être mises en jeu lors de la décomposition du domaine de simulation (communications entre CPU différents) et lors de l'utilisation d'un accélérateur GPU (communications CPU-GPU). Nous verrons ensuite que cette décomposition calcul/communication ne peut être qu'artificielle vu que ces deux composantes peuvent (et doivent) être intimement liées.

4.1.1 Techniques d'optimisation

La pierre angulaire de l'algorithme de modélisation sismique est l'utilisation d'opérations de type *stencils* pour résoudre l'équation d'onde discrétisée telle que nous l'avons précédemment introduit dans la section 2.4, opérations qu'on retrouve notamment dans l'algorithme 1. Plus généralement, ces opérations consistent en un parcours d'une grille régulière au moyen d'un nid de boucles en effectuant en chaque point de cette grille des opérations faisant intervenir les valeurs aux points voisins. Ces opérations peuvent se faire de

façon itérative en temps. On parle alors de ISL (*Iterative Stencil Loops*). De façon plus formelle, si on modélise les opérations ce type pour calculer l'évolution d'un ensemble de points d'une grille $\Omega = \{r\}$ sur un ensemble d'itérations en temps $T = \{t\}$ par une fonction f de l'état d'un ensemble de points appartenant au voisinage du point considéré $\Omega' = \{r'|r' \in voisinage(r)\}$, ces opérations peuvent s'écrire sous la forme de l'algorithme 3 [47] :

pour tous les $t \in T$ **faire**
  pour tous les $r \in \Omega$ **faire**
    $v_{t+1}(r) \leftarrow f(v_\tau(r'), r' \in \Omega', \tau \leq t)$

$v_t(r)$ étant la valeur à l'instant t au point de grille r.

Les calculs de ce type ont fait l'objet d'une littérature très riche de par le rôle important qu'ils jouent dans diverses simulations physiques basées sur la résolution d'EDP ainsi que dans des algorithmes de traitement du signal et de l'image. Divers travaux ont étudié des techniques d'optimisation de ces calculs sur des processeurs généralistes ainsi que sur des accélérateurs matériels. Des compilateurs spécialisés pour ce genre d'opérations ont même pu voir le jour [21]. Nous décrivons ici certaines de ces techniques d'optimisation courantes sur diverses plateformes. Pour le traitement d'un seul domaine, ces techniques consistent essentiellement en l'augmentation de la localité des données par le biais de l'utilisation d'un pavage (*tiling*) et de différentes stratégies : augmentation de la réutilisation des données présentes dans le cache (*cache blocking*), diminution de nombre de *cache misses* [47, 115], exploitation du parallélisme de données SIMD via l'utilisation des instructions vectorielles (SSE) ainsi que du préchargement des données en cache [47]. Toutes ces techniques peuvent être associées à une approche d'*auto tuning* pour optimiser dynamiquement leurs paramètres d'exécution [44, 47, 132]. Lorsque le domaine est décomposé en sous-domaines dont le traitement itératif est effectué par des unités de calcul différentes, d'autres techniques peuvent être utilisées. Nous en décrivons ici deux qui seront évoquées ultérieurement.

Utilisation de zones fantômes plus grandes Cette technique repose sur l'utilisation de calculs redondants pour réduire les points de synchronisation et les échanges de données. Il existe en effet dans ce cas des nœuds dont dépend le calcul à une itération t pour un sous domaine donné, mais qui ont été mis à jour dans des sous domaines voisins par des unités de calcul différentes (cf 3.1.2). On parle dans ce cas de nœuds ou zones fantômes [1] (*ghost zones / nodes*) ou de halo. Il est nécessaire dans ce cas d'attendre que le traitement des sous-domaines voisins soit achevé et de récupérer éventuellement les données nécessaires à chaque itération en temps. La taille de ces zones dépend directement de la taille du stencil utilisé (cf figure 3.3). En utilisant des zones fantômes plus grandes, il est possible d'effectuer plusieurs itérations en temps avant d'effectuer une synchronisation et éventuellement un échange de données. Le nombre d'itérations dépend alors de la taille de ces zones. Cette technique peut être appliquée lors de l'utilisation de

1. Dans certaines références le terme *ghost zones / nodes* est réservé aux données additionnelles introduites par cette technique alors que l'on parle de halo pour les dépendances.

systèmes à mémoire partagée ou d'accélérateurs à architecture parallèle de type GPU (on réduit alors le nombre de barrières de synchronisation à chaque itération en temps), ou de systèmes à mémoire distribuée (on réduit également les échanges de données). Dans [90], l'utilisation de cette technique pour des implémentations GPU de calculs de type ISL est étudiée. Une méthode est proposée pour déterminer la taille optimale des *ghost zones* et donc le nombre d'itérations en temps qu'il est possible d'effectuer sur chaque sous domaine sans synchronisation.

Obliquité en temps A l'instar de la technique précédente, l'utilisation de l'obliquité en temps, ou *Time skewing* [99], vise à augmenter l'asynchronisme en exploitant autant que possible les données présentes dans un sous domaine en effectuant toutes les itérations en temps qu'il est possible de faire avec ces données. Conceptuellement l'idée est séduisante, cependant la mise en œuvre pratique de cette technique en 3D reste complexe et requiert un effort de programmation important. Il est également à noter, que plus l'ordre du schéma augmente (i.e. plus le stencil est "large"), plus la possibilité d'avancer en temps est réduite. La plupart des implémentations sont basées sur des stencils à 5 points en 2D et à 7 points en 3D (le point central et 1 point de chaque côté dans chaque direction) qui sont rarement utilisés en pratique.

Utilisation des accélérateurs matériels Concernant l'utilisation des accélérateurs matériels, l'exploitation des FPGA et des GPU a été notamment étudiée. A titre d'exemple, dans [44], l'équation de la chaleur est résolue en 3D en utilisant une méthode itérative de Jacobi sur diverses architectures multi-cœurs et sur GPU. En adoptant une approche basée sur l'*auto-tuning*, les auteurs montrent que lorsque l'on omet les transferts hôte-GPU, les cartes graphiques considérées (NVIDIA GTX280) donnent les meilleures performances pour l'application étudiée relativement à toutes les plateformes testées y compris le processeur Cell BE. Ils atteignent ainsi les 36 GFlops en calcul en double précision, ce qui représente un speed-up de 14.3 par rapport à la version CPU de l'application évaluée sur le processeur Intel Clovertown.

Ces études se focalisaient essentiellement sur l'optimisation des algorithmes pour une seule carte accélératrice. L'utilisation de plusieurs GPU pour ce même type de calcul est abordée dans [91]. L'étude est étendue à l'équation d'onde discrétisée sur un domaine de densité constante. Une métrique basée sur la redondance des accès à la mémoire globale des GPU est introduite pour évaluer différentes techniques de parallélisation.

4.1.2 Quelques considérations de performance : étude introductive pour le noyau 2D

Nous nous intéressons dans cette section à l'étude de la modélisation de la propagation d'une onde sismique dans un milieu 2D de densité constante. Cette étude nous permettra d'introduire les facteurs importants et les critères de performance essentiels pour l'étude de l'utilisation des GPU pour le calcul généraliste par le biais de CUDA (l'approche OpenCl découle des mêmes considérations et à ce titre l'étude peut facilement être généralisée). Dans cette partie nous

étudions donc la résolution de l'équation (4.1) introduite dans la section 2.2 :

$$\frac{1}{c^2(x,y,z)}\frac{\partial^2 u(x,y,z,t)}{\partial t^2} - \Delta u(x,y,z,t) = s(t)\delta(x-x_s)\delta(y-y_s)\delta(z-z_s). \tag{4.1}$$

La résolution par différences finies sur une grille régulière selon la méthode décrite en 2.2.1 sera considérée. Le schéma d'ordre deux en temps et quatre en espace donnant le stencil illustré sur la figure 4.1 sera utilisé.

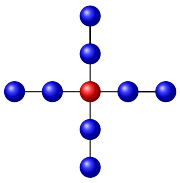


FIGURE 4.1 – Stencil de la modélisation 2D en densité constante pour le schéma centré d'ordre 4.

Organisation des calculs et notion d'occupation Comme décrit dans le chapitre précédent, la première étape consiste à écrire le *kernel* qui sera exécuté par les *threads* CUDA en parallèle. Dès cette étape se pose donc la question de l'organisation des calculs et du choix de la granularité ou du niveau de parallélisme à adopter. Pour le problème considéré, le calcul doit se faire sur une grille régulière à deux dimensions représentant la discrétisation spatiale du champ d'onde. A une itération en temps donnée, les calculs sur les différents nœuds de la grille (i.e. la mise à jour d'un point localisé du champ d'onde) sont indépendants. Une dépendance en temps existe cependant entre des nœuds voisins de la grille. Différents choix de parallélisation peuvent donc s'offrir. Par exemple, le traitement par un *thread* peut concerner un sous domaine du domaine global (une tuile, une ligne ou une colonne). Il est également possible d'assigner un point unique de la grille à un *thread*. C'est ce choix qui sera fait, le but étant de maximiser le parallélisme par l'utilisation d'un nombre aussi élevé que possible de *threads*. Utiliser un grain de parallélisme fin peut avoir comme inconvénient cependant d'augmenter la redondance entre *threads*. Une telle redondance peut être constatée dans le cas où plusieurs *threads* accèdent à la même donnée en mémoire globale par exemple. Utiliser la mémoire partagée permettra de limiter considérablement cette redondance comme il sera exposé par la suite.

Le seconde étape consiste à choisir l'organisation des calculs et leur hiérarchisation en blocs de *threads* et en grille de blocs de *threads* selon l'approche décrite dans en 3.2.2.0 (page 48). Cela revient à fixer les dimensions d'un bloc de *threads*. Ces dimensions fixées, on pourra déduire les dimensions de la grille des blocs de *threads* à partir de la grille numérique, le but étant de recouvrir cette dernière entièrement par des *threads*.

Les dimensions de la grille numérique n'étant pas forcément multiples de celles des blocs de *threads*, on créé en pratique le nombre nécessaire de blocs de *threads* pour recouvrir la totalité de la grille numérique ce qui conduit à un certain nombre de *threads* inactifs comme illustrée sur la figure 4.2. La grille de blocs aura alors comme dimensions :

$$\begin{aligned} gridDim.x &= (meshWidth + blockDim.x - 1)/blockDim.x \\ gridDim.y &= (meshHeight + blockDim.y - 1)/blockDim.y \end{aligned} \tag{4.2}$$

$blockDim.x$ et $blockDim.y$ représentant les dimensions d'un bloc de *threads* et $meshWidth$ et $meshHeight$ celles de la grille numérique.

L'algorigramme de gauche de la figure 4.3 décrit alors l'algorithme simplifié par *thread*. On se place dans le cas d'une source ponctuelle n'appartenant pas aux domaines de dépendance numérique des blocs de *threads* voisins du bloc auquel elle appartient. Dans le cas contraire, une synchronisation globale est nécessaire avant l'introduction de la source (utilisation d'un *kernel* séparé pour l'introduction de la source ou écriture directe dans la mémoire globale du GPU).

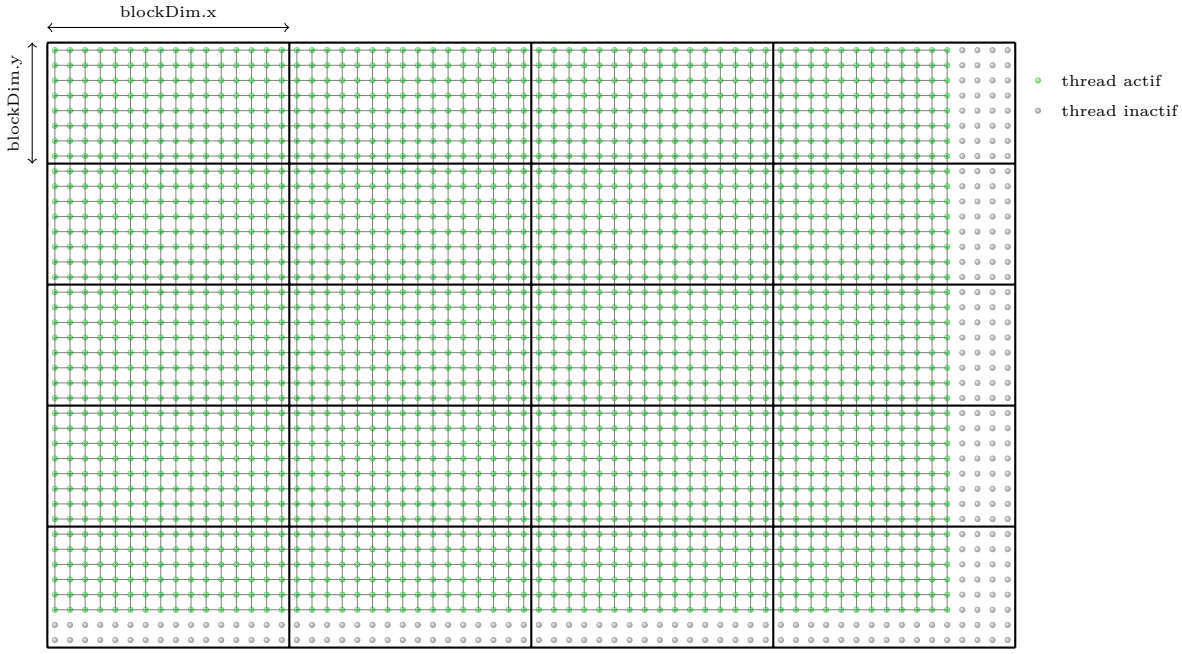


FIGURE 4.2 – Grille numérique et grille de calcul ($gridDim.x = 4$ et $gridDim.z = 5$).

Choisir les dimensions d'un bloc de *threads* obéit à plusieurs contraintes. Tout d'abord chaque génération de GPU impose une limite maximale sur le nombre de *threads* appartenant au même bloc. On pourra ainsi créer des blocs contenant jusqu'à 512 *threads* pour les GPU de *Compute Capability* (CC) < 2 et jusqu'à 1024 pour les générations plus récentes ($CC = 2.x$). Ensuite, de ce choix dépend l'occupation ou *Occupancy* du GPU. L'occupation, définie pour un *kernel* donné, représente le ratio entre le nombre de *warps* pouvant être actifs de façon concurrente sur un SM et le nombre maximal de *warps* concurrents que le GPU est capable de gérer (cf section 3.2).

L'occupation est une notion importante puisqu'elle caractérise le degré de parallélisme d'un *kernel* exécuté sur GPU. Ce parallélisme permet de recouvrir les temps d'accès de certains *warps*

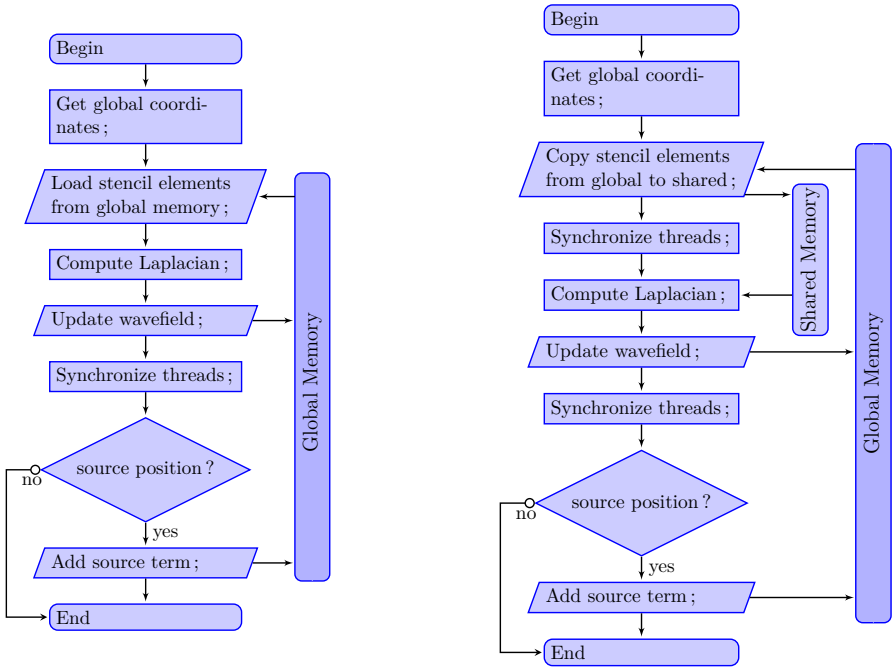


FIGURE 4.3 – Algorigrammes n'utilisant que la mémoire globale à gauche et utilisant aussi la mémoire partagée à droite.

à la mémoire globale ($\approx$ 500 cycles d'horloge) par les calculs effectués par d'autres *warps*. Le nombre de *warps* (et donc de *threads*, pour rappel : 1 *warp* = 32 *threads*) concurrents est limité par les ressources matérielles disponibles par SM.

Tout d'abord, à un GPU de CC donnée correspondent :

- une limite sur le nombre de blocs concurrents (huit pour tous les GPU actuels) ;
- un nombre maximal de *warps* concurrents.

Le tableau 4.1 décrit certaines caractéristiques des GPU selon leur CC. Ainsi, si on définit par exemple des blocs de 64 *threads*(2 *warps*), l'occupation pourra être au mieux égale à 0.66 pour les GPU de CC < 1.2 (24 *warps* concurrents), à $0,5$ pour les GPU de CC=1.2 ou 1.3 (32 *warps* concurrents) alors qu'elle ne dépassera pas $0,33$ pour les GPU de CC=2.x (48 *warps* concurrents). Dans ce cas, la taille des blocs est clairement trop petite pour pouvoir exploiter tout le parallélisme offert par la carte. Le cas inverse peut également se présenter. Si on choisit par exemple des blocs de 512 *threads*, l'occupation pourra être au mieux égale à 0.66 pour les GPU de CC < 1.2.

Ensuite, les ressources disponibles au sein des SM (registres et mémoire partagée) sont réparties dynamiquement à l'exécution d'un *kernel* entre les *warps* concurrents. Les besoins pour chaque bloc, définis à la compilation d'un *kernel*, peuvent limiter son occupation. Par exemple, si l'exécution d'un bloc de *threads* requiert plus de la moitié des registres disponibles pour un SM (voir tableau 4.1), un seul bloc pourra être résident au sein d'un SM à un instant donné,

	Compute Capability				
Ressources (par SM)	1.0	1.1	1.2	1.3	2.x
Nombre max de blocs résidents	8				
Nombre max de warps résidents	24		32		48
Nombre max de *threads* par bloc	512				1024
Nombre max de *threads* résidents	768		1024		1536
Taille de la mémoire partagée	16Kø				48Kø [2]
Nombre de registres 32bits	8.192		16.384		32.768

TABLE 4.1 – Ressources disponibles en fonction de la CC.

indépendamment de sa taille. Des options de compilation permettent de fixer le nombre maximal de registres utilisables par *thread*, ceci pouvant induire l'utilisation de la mémoire locale (physiquement basée sur la mémoire globale) avec un coût d'accès plus important et donc un impact pénalisant sur les performances du *kernel*. On montrera les limites de cette approche dans la section 4.1.3.

Dans bien des cas, il peut s'avérer utile de mener une phase de tests exhaustifs pour déterminer la meilleure configuration d'exécution. Pour le *kernel* de modélisation 2D, la figure 4.4 montre les temps d'exécution ainsi que l'occupation pour différentes dimensions de blocs de *threads* et différents GPU testés : G92 ($CC = 1.1$), T10 ($CC = 1.3$) et Fermi ($CC = 2.0$). Pour ce *kernel*, un *thread* utilise 13 registres. Une taille de bloc de 16x16 conduit alors à une occupation de 1 pour les GPU de $CC > 1.1$. Si on s'intéresse maintenant au GPU de $CC = 1.1$, où chaque SM ne dispose que de 8.192 registres, c'est le nombre de registres qui devient le facteur limitant l'occupation à partir d'une certaine taille de bloc. En effet, pour la même taille de bloc (16x16), un bloc a besoin pour être exécuté sur un SM de $16 \times 16 \times 13 = 3328$ registres. On ne pourra donc pas exécuter plus de deux blocs sur un même SM à un instant donné. On a alors 16 ($16 \times 16 \times 2/32$) *warps* concurrents contre un maximum de 24, ce qui donne une occupation égale à 0.667.

Enfin, la taille des blocs de *threads* définit également les motifs d'accès à la mémoire globale et à la mémoire partagée. De ce motif d'accès dépend la bande passante mémoire. C'est ce qui explique que sur les différentes plateformes testées, l'utilisation de blocs de dimension dimBlock.x égale à 16 donne les meilleures performances. Nous utiliserons par la suite des blocs de *threads* de taille 16×16.

Exploitation de la mémoire partagée Les accès à la mémoire globale ont un temps de latence élevé (de l'ordre de $0.4\mu s$, entre 400 et 600 cycles d'horloge) et sont grands consommateurs de bande passante, qui bien que supérieure à celle qu'on retrouve dans les CPU, peut être

2. Pour les FERMI, la taille de la mémoire partagée peut être fixée à 16Kø ou 48Kø le complément à 64Kø étant attribué au cache L1.

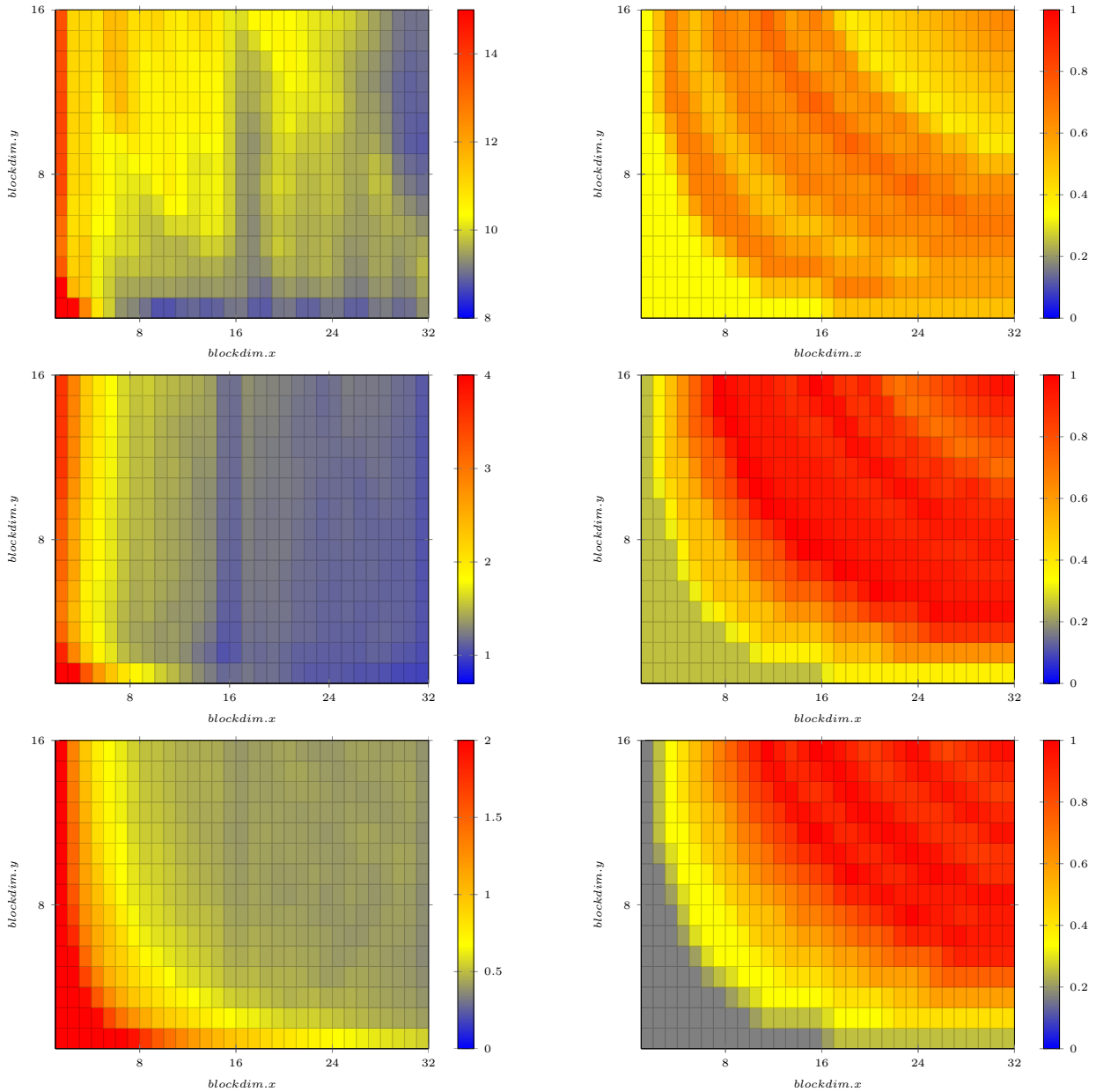


FIGURE 4.4 – Temps d'exécution en secondes (gauche) et occupation (droite) en fonction des dimensions des blocs de *threads* et du CC des GPU (CC 1.1 (haut), 1.2 (milieu) et 2.0 (bas)

rapidement saturée par les nombreux *threads* s'exécutant en parallèle. Il convient alors de limiter au maximum les accès à cette mémoire. Ceci est rendu possible par le biais des registres et de la mémoire partagée, localisés au sein des SM et disposant d'un temps d'accès de quelques cycles d'horloge. Notons tout d'abord que la mise à jour d'un point de la grille numérique représentant le champ d'onde induit plusieurs accès en lecture redondants à la mémoire globale. En effet, tel qu'illustré sur le premier schéma de la figure 4.5, chaque *thread* a besoin de lire en mémoire les neuf valeurs du champ d'onde (un point central ainsi que quatre valeurs dans chaque direction). Ces valeurs peuvent également être utilisées par des *threads* voisins. Pour limiter la redondance

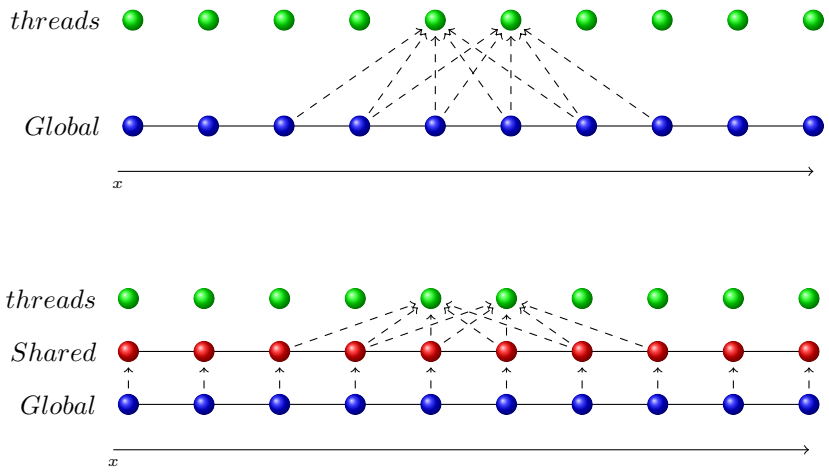


FIGURE 4.5 – Motifs d'accès à la mémoire globale (haut) et optimisation en utilisant la mémoire partagée (bas) pour le *kernel* de modélisation 2D selon la direction x.

des accès à la mémoire globale, nous introduisons l'utilisation de la mémoire partagée. Comme illustré sur la figure 4.5, chaque *thread* ira lire en mémoire une ou plusieurs valeurs du champ d'onde pour la placer en mémoire partagée. Cet espace étant visible par tous les *threads* d'un même bloc, le calcul du Laplacien pourra s'effectuer en utilisant les données qui y sont placées, comme indiqué dans l'algorigramme de droite de la figure 4.3. Nous passons alors de neuf lectures en mémoire globale pour la mise à jour d'un point de la grille à 1,5 accès en lecture par point de grille en moyenne. En effet, un bloc de *threads* de taille 16×16 effectue 16×16 lectures des points de la grille qui seront mis à jour par ce bloc ainsi que quatre zones mémoire de taille 16×2 correspondant à la largeur du *stencil* (ces zones sont à rapprocher des *ghost nodes* décrits dans la section 3.1.2). Au total, on effectue donc 384 lectures par bloc de 256 *threads*.

Le tableau 4.2 montre l'impact de l'utilisation de la mémoire partagée sur les temps d'exécution des noyaux de calcul pour différentes générations de GPU. Dans le cas des deux premières générations (G92 et GT200 respectivement de CC 1.1 et 1.2), la mémoire partagée est utilisée à la manière d'un cache explicitement géré par le développeur. L'impact est plus mesuré pour le FERMI de CC 2.0 qui dispose déjà de mémoire cache pour les accès à la mémoire globale.

La mémoire partagée obéit à des règles d'accès [106, p.72] qui permettent d'éviter la contention et donc la sérialisation des accès à cette mémoire par des *threads* différents. La mémoire

GPU	CC	Mémoire globale	Mémoire partagée
G92	1.1	9670	3930
GT200	1.2	1037.4	608.17
GF100	2.0	389.2	349.44

TABLE 4.2 – Temps d'exécution en secondes du noyau de modélisation 2D pour différentes générations de GPU et selon le type de mémoire utilisé.

partagée est en effet organisée en n banques ou modules de tailles égales et les accès à ces banques se font de manière parallèle. Par contre, les accès par des *threads* différents à des adresses situées dans la même banque peut conduire à la sérialisation de ces accès, on parle alors de conflit de banque (ou *bank conflict*). L'allocation d'espace supplémentaire (rajouter une marge d'une colonne sur le bord du tableau alloué en mémoire partagée ou *padding*) permet dans certains cas de réduire le nombre de ces contentions [92].

Bande passante mémoire et alignement des accès Comme indiqué précédemment, les motifs d'accès à la mémoire globale ont un impact important sur la bande passante mémoire et de là sur les performances globales d'un *kernel*. En effet, selon les propriétés d'alignement et de contiguïté de ces accès, le nombre de transactions mises en jeu peut varier de façon importante (un rapport de l'ordre de 1 à 16 en fonction de la CC du GPU considéré). Les accès à la mémoire globale sont gérés par demi-*warp* pour les GPU de $CC < 2.0$. Il est alors recommandé que les *threads* d'un même demi-*warp* accèdent dans l'ordre (le k-ième thread accède au k-ième mot en mémoire) à des zones mémoire contiguës alignées sur la taille totale des données à transférer. Plus précisément, considérons le cas le plus restrictif des GPU de $CC < 1.2$ et limitons-nous aux accès à des données de type flottant en simple précision (mots mémoire de 4 octets) tels que ceux mis en jeux dans le code de modélisation considéré. Dans ce cas, les 16 *threads* d'un demi *warp* doivent accéder à des adresses mémoire successives, la première devant être aligné sur 64 octets. Dans ce cas précis les accès par demi-*warp* seront regroupés en une seule transaction mémoire de 64 octets. Dans les autres cas, 16 transactions (une transaction par mot mémoire) sont nécessaires, ce qui engendre un surcoût important. Cet effet est illustré sur la figure 4.7 qui représente les temps d'exécution des *kernels* décrits précédemment avec l'introduction de zones de *padding* de taille variable aux frontières du domaine tel qu'illustré sur la figure 4.6. Ces marges, allouées en mémoire sans être utilisées modifient les motifs d'accès des *threads* en mémoire. Ces courbes montrent qu'une zone de *padding* de taille 14×4 octets induit un gain important de performance, cette taille permettant d'assurer l'alignement des accès en mémoire globale pour les blocs de *threads* de taille 16×16 utilisés, tel qu'illustré sur la figure 4.7(a).

Les contraintes décrites précédemment pour les GPU de *compute capability* 1.0 et 1.1 ont été relâchées pour les GPU de *compute capability* 1.2 et 1.3. Dans ce cas, et toujours pour des mots mémoire de 4 octets, les accès aux données par demi-*warp* se font selon les étapes suivantes :

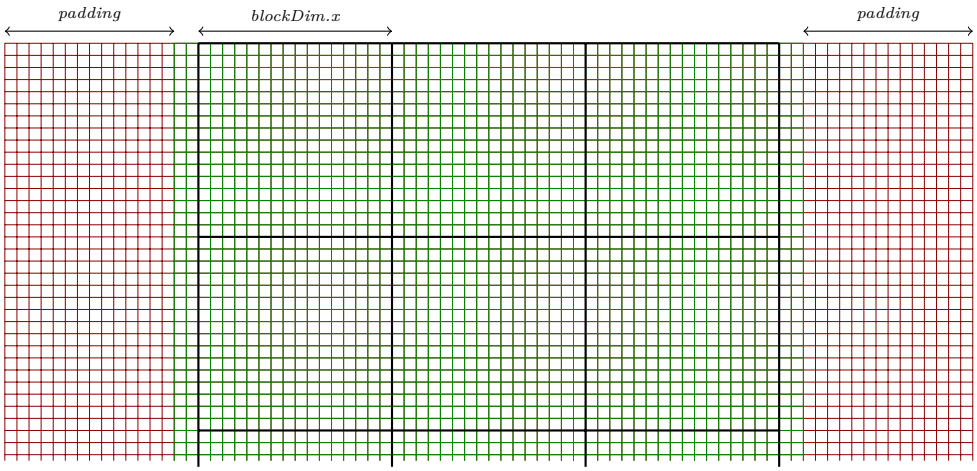


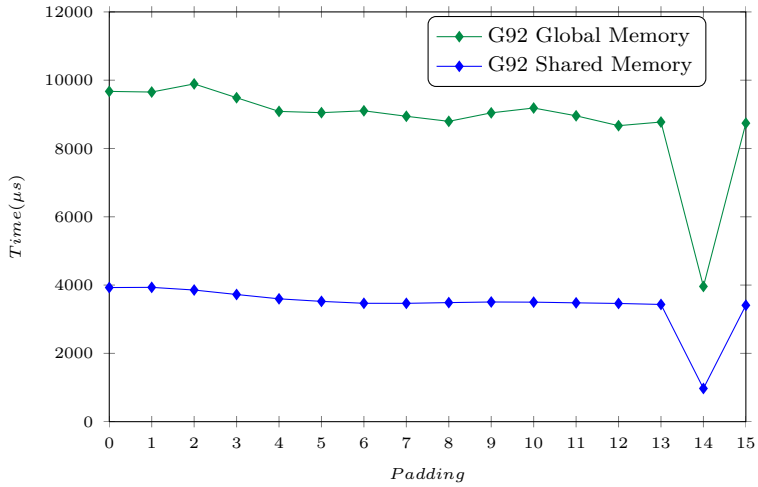
FIGURE 4.6 – L'utilisation d'un *padding* de 14 mots mémoire pour assurer l'alignement des accès en mémoire globale : tous les *threads* d'un même demi-*warp* accèdent à des zones mémoires contiguës alignées sur 64 octets(16×4).

1. trouver le segment mémoire de 128 octets contenant l'adresse requise pour le premier *thread* du demi-*warp* ;
2. trouver tous les autres *threads* accédant au même segment ;
3. trouver le segment de plus petite taille (64 ou 32 octets) satisfaisant toutes les requêtes ;
4. effectuer la transaction et marquer les *threads* servis comme inactifs ;
5. recommencer jusqu'à épuisement des threads actifs du demi-*warp*.

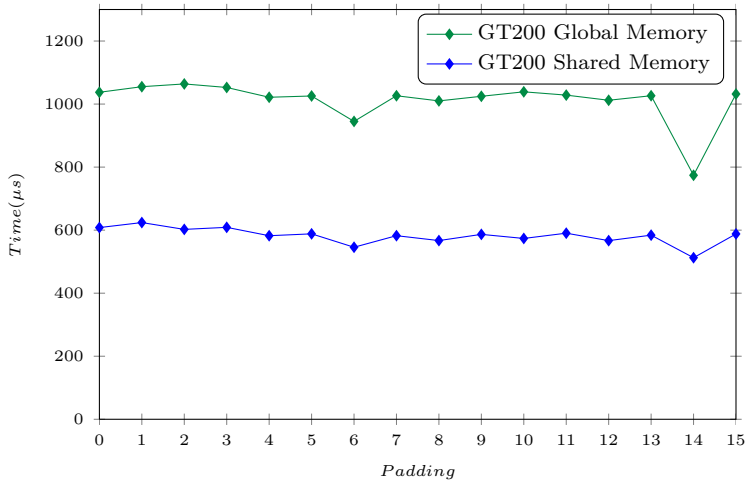
L'impact du rajout de type *padding* est donc plus réduit pour les GPU de *compute capability* 1.2 et 1.3 tel que le montrent les temps d'exécution des *kernels* de modélisation reportés sur la figure 4.7(b).

Dans le cas des GPU Fermi de dernière génération (CC=2.0), deux niveaux de cache ont été introduits pour améliorer les performances des accès à la mémoire globale. Des lignes de cache de 128 octets sont utilisées. Le rajout de *padding* fait alors apparaître des effets de cache différents de ce que l'on observe sur les générations précédentes de GPU comme illustré sur la figure 4.7(c).

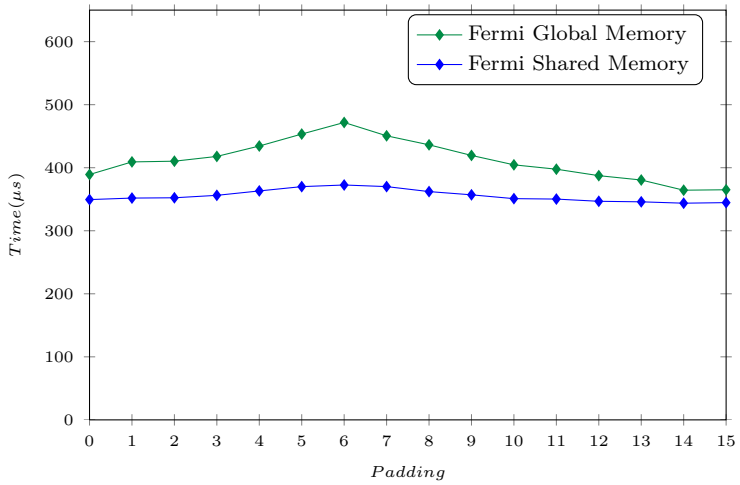
Utiliser le GPU pour . . . l'affichage ! Des mécanismes d'interopérabilité entre CUDA et OpenGL existent. Ces mécanismes permettent d'effectuer en OpenGL le rendu et l'affichage de données calculées sur GPU avec CUDA. Ces données sont présentes dans la mémoire du GPU et leur affichage ne nécessite donc pas de transferts CPU-GPU. Dans le contexte de la modélisation sismique, l'affichage pendant le calcul CUDA de l'état du champ d'onde offre un moyen pratique de valider visuellement l'exécution de l'application ou de détecter les éventuels problèmes : dispersion numérique, réflexions aux bords du domaine (*damping* insuffisant), positionnement de la source, etc. La figure 4.8 illustre ces possibilités. Nous pouvons déjà pressentir la puissance



(a)



(b)



(c)

FIGURE 4.7 – Temps d'exécution des *kernels* de modélisation 2D avec l'introduction de zones de *padding* de taille variable aux frontières du domaine pour la G92 (a), GT200 (b) et F100 (c).

d'un tel mécanisme et imaginer des applications plus poussées. Nous aborderons ce point dans le dernier chapitre de ce manuscrit.

Conclusion Dans cette section, nous avons mis en évidence par le biais de l'utilisation d'un cas simple, différents paramètres importants en programmation CUDA et étudié l'impact de leur prise en compte sur les performances de ce programme selon la génération de GPU utilisée. Ceci démontre que les performances des *kernels* GPU peuvent être très sensibles aux optimisations utilisées. Il s'avère également capital avant d'envisager toute optimisation de localiser le goulet d'étranglement du programme considéré. Il est ainsi peu utile de chercher à augmenter l'occupation des SM, si la bande passante mémoire est saturée par exemple.

4.1.3 Noyau de calcul 3D

Nous abordons maintenant le cas de la modélisation 3D qui pose des difficultés supplémentaires. Tout d'abord, la taille des données mises en jeu est plus importante, ce qui requiert la mise en œuvre de techniques de décomposition de domaine telles que décrites en 3.1.2. Il faut également adapter le noyau de calcul à la 3D en étudiant les types de mémoires à utiliser ainsi que les motifs d'accès aux données. Nous étudions dans cette section le schéma d'ordre 2 en temps et 8 en espace. On commencera par aborder le cas d'un domaine de densité constante. Ce cas est celui qui a été étudié en premier, eu égard à sa simplicité, son haut potentiel de parallélisation et la régularité des calculs mis en jeu. On exposera les différentes techniques mises en jeux pour augmenter les performances de l'application. Sera ensuite abordé le cas de la densité variable.

Modélisation en densité constante

De façon similaire au cas 2D, notre première approche a consisté à utiliser la mémoire globale et à dédier un *thread* pour la mise à jour de chaque point de la grille. On a pu atteindre avec cette approche une occupation de 66% sur les premières cartes CUDA N80. Cependant les performances obtenues restaient limitées essentiellement à cause des temps d'accès à la mémoire globale qui constituait le goulot d'étranglement du calcul. Chaque *thread* effectue en effet 26 accès en lecture à la mémoire globale et un accès en écriture. Pour limiter les accès à la mémoire globale et encore une fois de façon similaire au cas 2D, l'utilisation de la mémoire partagée a été introduite. Les données sont donc copiées depuis la mémoire globale vers la mémoire partagée pour être utilisées par les threads d'un même bloc. Cette technique permet de réduire les accès à la mémoire globale mais augmente les besoins des kernels en mémoire partagée. Pour des blocs de *threads* de taille $16 \times 4 \times 4$ cela nécessite l'allocation d'au moins 9ko de mémoire partagée. Ceci implique qu'un seul bloc de 256 *threads* peut s'exécuter à un instant donné sur un multiprocesseur au lieu des 768 pour les GPU de Compute Capability inférieure à 1.2 (33% d'occupation) et de 1024 pour les GPU de Compute Capability plus élevée (occupation de 25%).

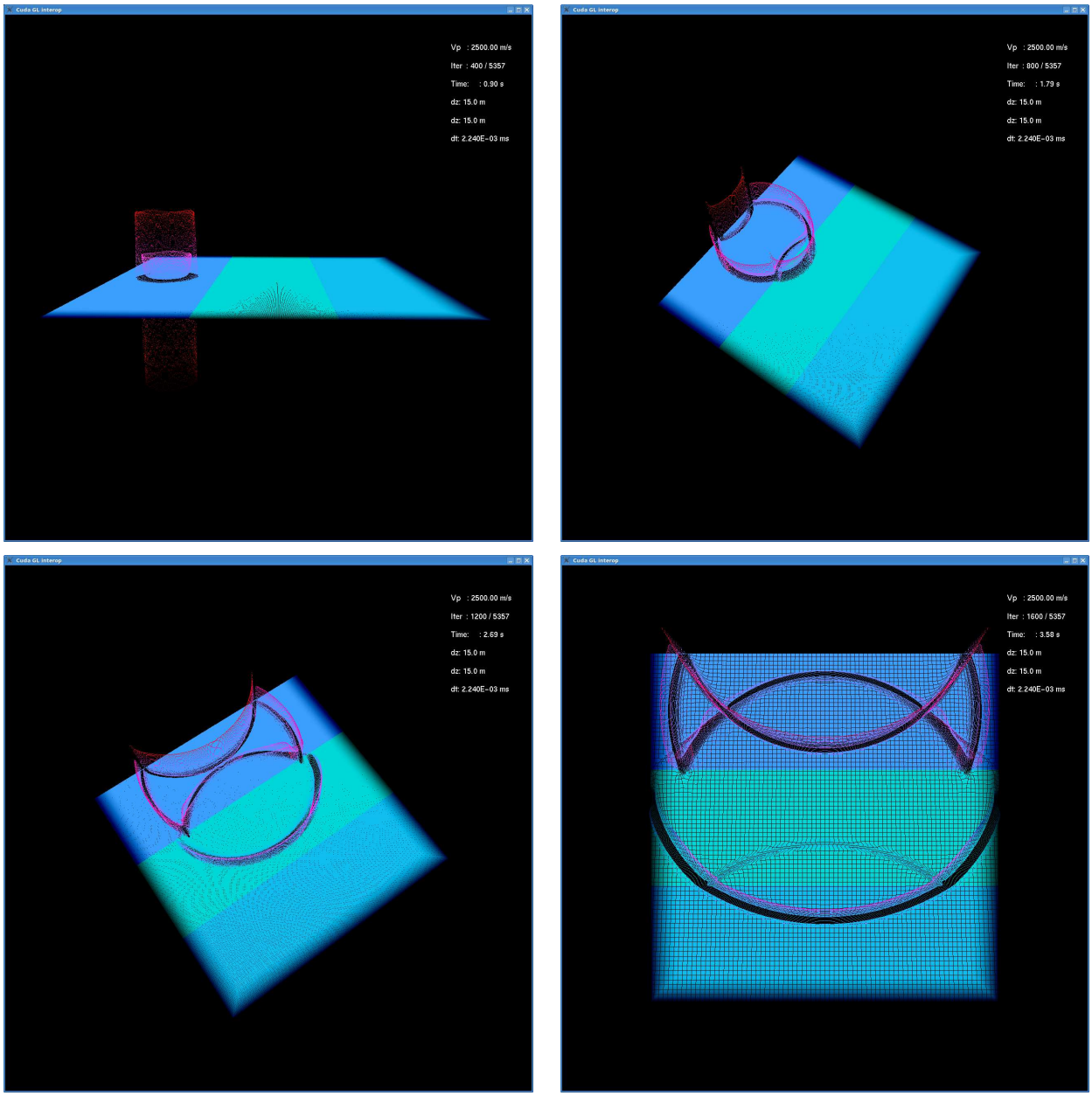


FIGURE 4.8 – Interopérabilité CUDA-OpenGL : l'état du champ d'onde est calculé et affiché en temps réel, en superposition avec le modèle de vitesse (3 couches : $V_1 = 1500ms^{-1}, V_2 = 2500ms^{-1}, V_3 = 2000ms^{-1}$), aux itérations 400, 800, 1200 et 1600.

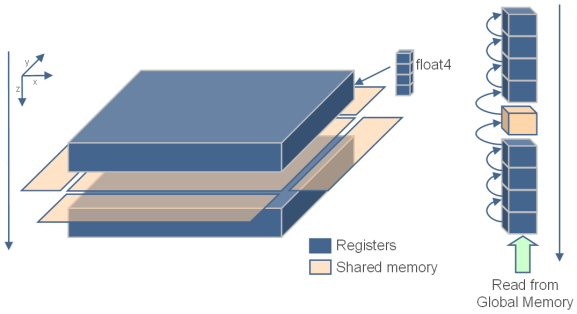


FIGURE 4.9 – Principe du noyau de la modélisation sismique en 3D : une fenêtre coulissante est utilisée pour garder les données nécessaires au calcul de la dérivée en z dans les registres

Pour limiter l'utilisation de la mémoire partagée, et espérer ainsi augmenter l'occupation, une nouvelle approche est adoptée : au lieu de dédier un *thread* à la mise à jour de chaque point de la grille, un algorithme basé sur une fenêtre glissante [91] selon la direction z (la plus "lente" ou la moins contigüe en mémoire) est mis en place. La figure 4.9 illustre le principe de cet algorithme : la mémoire partagée est utilisée pour charger le plan du domaine courant $(x, y, z = z_{courant})$. De cette manière, tous les *threads* peuvent calculer les composantes selon les directions x et y du Laplacien. Pour la composante selon la direction z, chaque *thread* charge dans des registres les éléments nécessaires à son calcul, ce qui correspond aux éléments $(x_{th}, y_{th}, z = z_i), i \in [z_{courant} - l, .., z_{courant} - 1, z_{courant} + 1, .., z_{courant} + l]$ avec ici $l = 4$. A chaque itération, correspondant à un décalage selon z, les valeurs sont décalées selon le schéma illustré sur la figure 4.9. Ainsi pour chaque *thread*, seule la valeur $(x_{th}, y_{th}, z_{courant} + 4)$ est lue à partir de la mémoire globale. C'est ce kernel (noté dans la suite *Basic*) qui s'est avéré être le plus performant. En rajoutant du padding pour assurer l'alignement des accès en mémoire globale, une réduction du temps de calcul de 15% est observée pour des GPU de Compute Capability 1.2. On reporte dans le tableau 4.3 l'*occupancy* ainsi que les ressources requises pour chaque *kernel*.

Pour limiter les réflexions artificielles sur les bordures du domaine, on utilise en 3D exclusivement les PML, les ABC moins efficaces impliquant l'utilisation de quantités importantes de mémoire. La mise en œuvre des PML (cf section 2.2.4) sur GPU requiert l'utilisation d'un *kernel* plus complexe mettant en jeu la résolution de deux équations couplées et nécessitant la mise à jour du champ d'onde ainsi que d'un tableau de coefficients de *damping* artificiels. Ce *kernel* (noté *Damping* dans la suite) est basé sur le même mécanisme de fenêtre glissante utilisée pour le noyau *Basic*. Cependant, la complexité accrue de ce noyau implique un besoin plus élevé en registres et en accès mémoire. L'occupation est alors limitée par le nombre de registres requis par chaque *thread*. Les blocs de *threads* utilisés sont de dimensions 256 (16×16). Pour atteindre une occupation de 100%, chaque *thread* doit utiliser moins de 16 $(16384/(256 \times 4))$ registres. En utilisant de 16 jusqu'à 21 registres on atteint 75% d'occupation. L'implémentation complète du kernel de calcul requiert 32 registres ce qui correspond à une occupation de 50%. Un moyen

TABLE 4.3 – Noyaux CUDA pour la modélisation, ressources et taux d'occupation

Noyau	Mémoire partagée (octets)	Registres	Taux d'occupation
Basic	2400	20	75%
Damping	2432	28	50%
WF	2432	20	75%
PML	2720	17	75%

TABLE 4.4 – Temps moyens de calcul (ms) des noyaux de modélisation pour une itération sur un domaine de taille $528 \times 254 \times 1067$

Stratégie	Damping	Basic	Total
ST1	197.05	26.27	223.32
ST2	169.41	–	169.41
ST3	113.61	22.70	136.31
ST4	101.49	22.70	124.19

de réduire le nombre de registres requis et augmenter ainsi l'occupation consiste à subdiviser le noyau de calcul et le transformer en plusieurs noyaux plus simples. Le noyau PML a ainsi été transformé en 2 noyaux distincts. Le premier, noté WF, met à jour le champ d'onde, tandis que le deuxième, noté PML, met à jour le tableau des coefficients de damping.

Pour une itération en temps sur tout le domaine, on peut procéder de deux façons différentes. La première consiste à différencier les zones d'absorption des zones de propagation en appliquant des kernels différents. La seconde consiste à utiliser en tout point de la grille le même kernel avec des coefficients d'absorption nuls dans les zones de propagation. Les temps d'exécution correspondants à ces différentes approches sont reportés dans le tableau 4.4 pour une itération en temps sur un domaine de taille $528 \times 254 \times 1067$ et comprenant 5 zones d'absorption. Dans les stratégies notées ST1, ST3 et ST4 on utilise la première approche, tandis que dans la stratégie notée ST2, on suit la deuxième. Pour le cas ST1, on a utilisé une option de compilation permettant de limiter le nombre de registres utilisés à 21 pour obtenir une occupation de 75% pour les noyaux Basic et Damping. Ceci a pour effet de recourir à l'utilisation de la mémoire locale à accès lent (*register spilling* voir 5.1.3), ce qui réduit au final les performances du noyau. Dans le cas ST3, on utilise également le noyau Damping dans les zones d'absorption sans limitation de registres, alors que dans le cas ST4 on utilise les noyaux WF et PML dans ces mêmes zones. La dernière stratégie s'est avérée être la plus performante pour le cas considéré.

Cas de la densité variable

Le passage au cas de la modélisation dans un domaine où la densité varie en espace n'est pas trivial. En effet, le stencil est deux fois plus large que dans le cas de la densité constante comme détaillé en 2.2. Ceci rend le nombre de registres insuffisant pour permettre la mise en place du mécanisme de fenêtre glissante utilisé pour la densité constante. La solution mise en place repose sur la décomposition du calcul en deux étapes. La premières effectue le calcul de la première dérivée selon z. La deuxième effectue les opérations restantes.

En densité variable, l'utilisation d'un seul noyau s'est montré 20% plus rapide que l'utilisation des noyaux Basic et Damping.

4.2 Etude des performances

Dans cette section, on décrit et on analyse les performances mesurées pour la modélisation sur GPU. Ces performances sont comparées à celles obtenues en utilisant les processeurs généralistes. On étudie en particulier le passage à l'échelle de chacune des solutions. Le temps de référence sera alors le temps nécessaire au traitement de tout le domaine sans décomposition. Le temps de référence CPU correspond alors à une exécution séquentielle sur un seul cœur et le temps de référence GPU correspond à l'utilisation d'un GPU. Ceci implique que le GPU est considéré comme une unité de calcul même s'il se compose de plusieurs multi-processeurs. Il faudra prendre cet aspect en compte avant toute conclusion.

Tout le long de cette section, on considère un cas test 3D (noté dans la suite 3DModel) dont la grille est de dimensions selon x, y et z respectivement : $521 \times 254 \times 1067$ avec $\Delta x = 12.5$, $\Delta y = 12.5$ et $\Delta z = 10$ en mètres. Ainsi, ce modèle couvre une surface de $20.7km^2$ pour une profondeur de $10.7km$.

4.2.1 Plateforme de test GPU

La plateforme de tests sur laquelle les mesures de performance ont été effectuées se compose de dix nœuds bi-socket quad-core couplés à 5 lames NVIDIA Tesla S1070. Comme illustré sur la figure 4.10, chaque nœud est connecté au serveur Tesla via un bus PCI-Express de deuxième génération. Chaque lame Tesla est composée de 4 GPU T10. Chaque paire de GPU se partage via un *switch* la même connexion PCIe 2.0. Ainsi, chaque nœud hôte a accès à 2 GPU par le biais d'un même bus. Les détails techniques de ce système sont décrits dans le tableau 4.5.

Nous avons également eu accès à un cluster plus grand basé sur une architecture similaire constitué de 128 nœuds Intel Xeon bi-socket quad-core (3.0 GHz) épaulés par 64 lames NVIDIA TESLA S1070 (256 GPU T10).

On étudie les facteurs d'accélération observés pour les implémentations décrites pour le cas de test 3DModel. Sauf indication explicite, tous les temps GPU indiqués par la suite prennent

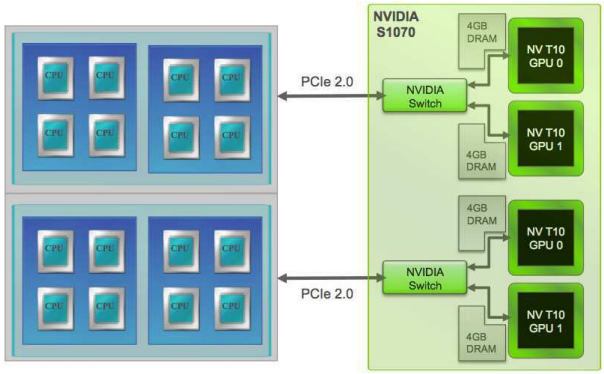


FIGURE 4.10 – Schéma représentant deux nœud de calcul de la plateforme de test.

TABLE 4.5 – Spécifications techniques du système utilisé.

Lame	Nœud hôte	NVIDIA S1070
Nombre	10	5
Processeur	Xeon 5405	T10
Sockets × cœurs	2×4	4×240
Fréquence (GHz)	2.00	1.44
Mémoire (Gbytes)	16	16 (4×4)
Mémoire cache (CPU)	$4 \times 6MB$	–
Mémoire partagée (GPU)	–	$30 \times 16KB$

TABLE 4.6 – Temps moyens en secondes pour une itération de la modélisation 3D en densité constante et en densité variable en fonction de la taille du domaine.

Nombre	Densité constante				Densité variable			
d'éléments	CPU	GPU	Ratio	Elts/s	CPU	GPU	Ratio	Elts/s
1.97E+07	0.48	0.046	10.43	4.28E+08	1.41	0.054	25.93	3.64E+08
3.71E+07	0.82	0.056	14.64	6.63E+08	2.46	0.094	26.09	3.94E+08
7.37E+07	1.69	0.079	21.39	9.33E+08	4.51	0.180	25.06	4.10E+08
1.43E+08	3.91	0.146	26.71	9.80E+08	9.92	0.342	28.95	4.18E+08

en compte aussi bien l'exécution des noyaux de calcul sur GPU que les temps des transferts hôte-GPU.

4.2.2 Montée en fréquence

Comme énoncé dans le deuxième chapitre, l'un des objectifs de l'accélération des applications d'imagerie sismique est l'amélioration de la résolution de l'image obtenue en simulant la propagation de fréquences plus hautes. L'augmentation de la fréquence implique dans le cadre de la résolution par différences finies une discrétisation de grille plus fine et des pas de temps plus petits. Le doublement de la fréquence maximale simulée induit une multiplication par 16 du coût de la simulation ($\times 8$ pour la taille de la grille et $\times 2$ pour le pas en temps). Pour notre implémentation GPU, on se propose ici d'étudier l'impact de l'augmentation de la taille de la grille. Dans le tableau 4.6 sont reportés les temps de calcul moyens pour une itération sur un seul domaine de taille croissante et les facteurs d'accélération GPU/CPU correspondant ($ratio = t_{CPU}/t_{GPU}$). On note que l'accroissement de la taille du domaine induit une augmentation de ce ratio. Ceci est dû au fait que pour un domaine de taille n^3, le temps des communications CPU-GPU augmente en $\Theta(n^2)$ (transfert de surfaces sur les bords) alors que les temps de calcul augmentent en $\Theta(n^3)$ (calculs sur des volumes). Le ratio t_{CPU}/t_{GPU} peut ainsi s'exprimer sous la forme :

$$R = \frac{t_{CPU}[\theta(n^3)]}{t_{comm}[\theta(n^2)]+t_{GPU}[\theta(n^3)]}.$$

D'un autre côté, plus le domaine est grand, plus le parallélisme engendré par le traitement du domaine est important. Un nombre plus important de blocs de threads peut ainsi être lancé en parallèle. Le temps de lancement des *kernels* et les temps d'accès à la mémoire sont alors mieux recouverts ce qui a pour effet d'augmenter le nombre moyen d'éléments traités par unité de temps. Il est ainsi recommandé pour exploiter au mieux les ressources du système, de mettre des domaines les plus grands possibles sur chaque GPU, dans les limites imposées par la taille de la mémoire de chaque GPU.

TABLE 4.7 – Temps moyens en secondes pour une itération de la modélisation 3D en densité constante et en densité variable sur le modèle 3DModel.

Nombre	Densité constante			Densité variable		
de sous-domaines	CPU	GPU	Ratio	CPU	GPU	Ratio
1	3.89	0.146	26.71	8.83	0.205	43.07
2	1.59	0.083	19.15	5.09	0.130	39.07
4	0.94	0.046	20.43	2.11	0.078	26.98
8	0.47	0.034	15.16	1.06	0.053	20.03

4.2.3 Scalabilité forte

En corollaire à ce qui a été observé dans le cas de la montée en fréquence, les résultats reportés dans le tableau 4.7 montrent que le ratio temps CPU/GPU décroit dans le cas de l'augmentation du nombre de sous-domaines (et donc de réduction de la taille des domaines par GPU). En effet le temps de calcul décroît et les temps de communication qui restent constants deviennent prédominants (communications CPU-GPU et MPI). Dans la figure 4.11 sont détaillés les coûts en temps de calcul et en échanges de données selon la géométrie de la décomposition de domaine adoptée pour la modélisation en densité variable. Les temps reportés correspondent au cas d'un modèle de taille 512^3 et au modèle 3DModel. La géométrie de la décomposition est indiquée en abscisse sous la forme du triplet $N_x - N_y - N_z$, où N_x, N_y et N_z correspondent au nombres de sous-domaines selon x, y et z respectivement. Le modèle de taille 512^3 est utilisé pour considérer des échanges de données et des domaines de tailles égales pour des décompositions équivalentes, qui peuvent s'obtenir les unes des autres par simple permutation (par exemple 2-1-1 et 1-2-1 et 1-1-2). Les résultats reportés mettent en exergue l'augmentation du coût des échanges MPI relativement à celui du calcul sur GPU. A partir de 8 sous-domaines utilisés, plus de 40% du temps d'une itération en temps est consacré aux échanges de données, ce qui réduit considérablement le facteur d'accélération relativement à la solution CPU.

La figure 4.11 montre également que la décomposition selon la dimension qui varie le plus lentement en mémoire (en l'occurrence z) est la stratégie la plus performante. En effet, comme il sera démontré plus tard, cela réduit le surcoût lié aux échanges de données. Cela assure également une meilleure scalabilité des calculs sur GPU : le nombre de blocs de *threads* pouvant s'exécuter en parallèle reste constant (car donné par les dimensions en x et en y) et le nombre d'opérations séquentielles sur la carte est réduit (taille de la boucle en z).

FIGURE 4.11 – Temps moyen pour une itération en temps de la modélisation en densité variable sur un cube de taille 512^3 (haut) et sur le modèle 3DModel (bas). En abscisse, la géométrie de décomposition : nombre de domaines selon les directions x, y et z respectivement.

TABLE 4.8 – Nombre d'éléments traités par seconde (mesuré) et nombre optimal de la modélisation à densité variable en fonction du nombre de sous-domaines (la taille d'un sous domaine est constante).

NB domaines	NB Eléments	Elts/s	Optimal
1	1.409E+08	6.34E+08	6.34E+08
2	2.823E+08	1.08E+09	1.27E+09
4	5.621E+08	1.89E+09	2.54E+09
8	1.139E+09	3.59E+09	5.07E+09
16	2.295E+09	4.97E+09	1.01E+10

4.2.4 Scalabilité faible

Le tableau 4.8 montre la performance de la modélisation en densité variable en terme de nombre d'éléments traités par seconde lorsque la taille du domaine global évolue de pair avec le nombre de sous-domaines en maintenant une taille de sous domaine fixe. Encore une fois, l'étude de cette scalabilité faible montre l'impact des communications sur le temps de calcul global. A 16 sous-domaines, on atteint moins de 50% de la performance théorique maximale (colonne Optimal).

4.2.5 Granularité CPU GPU

Comme décrit dans la section 4.2.1, un nœud de la plateforme de test est relié à deux accélérateurs GPU via la même connexion PCIe. Il est alors opportun d'évaluer la différence entre une configuration avec deux GPU reliés au même nœud et l'utilisation de deux GPU connectés à deux nœuds différents pour apprécier l'impact de la contention sur le lien PCIe et des communications réseau. Nous avons indiqué précédemment que le taux de transfert des données entre hôte et GPU diminue lorsque les deux GPU sont sollicités simultanément. Les communications entre les deux processus MPI sont également impactées : dans un premier cas on utilise la mémoire partagée du même nœud, dans le second on passe par le réseau Infiniband haute performance. On s'attend donc à avoir des temps différents puisque les temps de communication devraient être différents. Les mesures effectuées montrent cependant que les temps restent similaires puisque les temps de calcul sont prépondérants par rapport aux temps de communication. On en déduit donc que la contention au niveau du lien PCIe n'a pas d'impact notable sur les performances globales. La configuration considérée (1CPU pour 2 GPU) semble donc adaptée à la modélisation et permet de maximiser la densité du système et de réduire ainsi globalement la consommation électrique et l'occupation au sol.

4.2.6 Etude globale en grandeur réelle

Nous avons pu montrer jusque là que la solution GPU proposée pouvait se montrer concurrentielle par rapport à l'utilisation de ressources CPU conventionnelles. Cependant, pour aller au bout de la démarche, il ne suffit pas de comparer un GPU à un cœur ou un processeur CPU. Il faut en effet, pour avoir une comparaison cohérente, considérer les solutions dans leur globalité pour établir une comparaison de performance ramenée à une consommation électrique, à une occupation spatiale, aux coûts liés à l'achat et à l'exploitation, etc. Ce type de comparaison est complexe et est fortement lié aux applications et aux conditions d'utilisation. Nous proposons à titre indicatif une comparaison entre deux systèmes occupant chacun une armoire (*rack*), disponibles sur le marché. Le premier est basé sur une architecture conventionnelle tandis que le deuxième utilise le même type de GPU considéré jusque là. Cette comparaison permet, à occupation spatiale [3] équivalente, d'avoir une idée des performances et de comparer le coût et la consommation des deux solutions. Le premier rack utilisé est un SGI ICE Harpertown avec 64 nœuds CPU. Le second est un SGI ICE GPGPU équipé de 8 Tesla S1070 (32 T10 GPU) couplés avec 16 nœuds CPU.

En densité variable et en utilisant un modèle réaliste avec une grille numérique de dimensions $560(\Delta x = 12.5m) \times 560(\Delta y = 12.5m) \times 905(\Delta z = 10m)$ et une simulation de 22760 itérations en temps ($\Delta t = 0.45ms$), plusieurs points de tir sont modélisés en parallèle. Avec le *rack* GPU nous avons pu obtenir 232 points de tir pour un jour de calcul alors que le *rack* CPU ne permettait de simuler que 54 points de tir par jour. En face de ces chiffres, nous pouvons également rajouter une consommation 30% inférieure pour la configuration équipée de GPU.

4.3 Optimisation de la décomposition de domaine et des communications hôte-GPU

4.3.1 Communications CPU-GPU

Comme décrit précédemment, pour mettre à jour le champ d'onde au pas de temps t_{n+1}, les états du champ d'onde aux pas de temps t_{n-1} et t_n sont nécessaires. La mise à jour du champ d'onde est alors réalisée en échangeant les pointeurs des tableaux représentant les états du champ d'onde aux itérations t_n and t_{n+1}. Les données transférées vers la mémoire globale du GPU sont persistantes. Dans le cas de la décomposition du domaine de simulation en plusieurs sous-domaines, seuls les nœuds fantômes devront être échangés entre sous-domaines à chaque itération en temps, ce qui implique également des échanges entre GPU et nœud hôte. A partir de la figure 4.11 on a démontré que ces échanges peuvent représenter une limitation importante pour le passage à l'échelle de la solution proposée.

Il est possible dans ce cas de transférer tout le sous-domaine ou uniquement les nœuds fantômes. Réduire les transferts aux nœuds fantômes induit une mise en œuvre plus complexe, mais

3. La notion d'occupation spatiale ou *footprint* n'est pas à négliger dans les datacenters puisqu'il faut prévoir les infrastructures nécessaires.

TABLE 4.9 – Temps de transferts (en ms) du nœud hôte vers le GPU (HtoD) et du GPU vers le nœud hôte (DtoH) en fonction de la direction du transfert pour des stencils de tailles 4 et 8.

Taille de stencil	4		8	
	HtoD	DtoH	HtoD	DtoH
x	23.32 (61.3%)	46.54 (75.2%)	44.15 (70%)	77.41 (80%)
y	13.02 (34.2%)	13.94 (22.5 %)	16.15 (25%)	15.72 (17 %)
z	1.70 (4.5%)	1.40 (2.3 %)	3.04 (5%)	3.21 (3 %)
Total	38.04	61.88	63.34	96.34

permet de réduire considérablement le temps des échanges qui peuvent prendre des proportions importantes dans le temps global de la simulation. A titre d'exemple, le transfert d'un domaine 3D de taille $528 \times 254 \times 1067$ requiert approximativement $0.1s$, ce qui est à peu de chose près équivalent au temps d'une itération en temps sur le même domaine.

La solution proposée permet la décomposition du domaine de simulation selon les 3 axes x, y et z. Cependant, des échanges de types différents auront lieu entre nœuds d'une part et entre GPU et nœud hôte d'autre part, et ce selon la géométrie de la décomposition adoptée. En effet, la représentation des données en mémoire induit un axe de décomposition privilégié.

Plus précisément, considérons le cas où les données sont représentées en mémoire sous la forme d'un vecteur où la dimension x est celle qui varie le plus rapidement et z celle qui varie le plus lentement (l'accès à l'élément $u(x, y, z)$ s'effectue à l'adresse $u + (x + y \times dimx + z \times (dimx + dimy))$. Dans ce cas les échanges selon la direction z concerneront des blocs de mémoire contigus (temps de latence réduit), alors que ceux s'effectuant selon la direction x mettront en jeu un nombre important de transferts de petite taille (temps de latence important). La figure 4.12 illustre ces différents cas de figure pour un sous-domaine de taille 16^3 points. Pour rappel, le temps de latence de la connexion PCIe 2.0 entre GPU et nœud hôte est de l'ordre de $16\mu s$. Ceci explique que pour un sous-domaine de taille 512^3 il peut s'avérer plus intéressant de transférer la totalité du domaine à chaque itération en temps que de faire des transferts partiels non optimisés.

On a pu mesurer que la direction selon laquelle se font les échanges (x, y ou z) a un impact très important sur le temps nécessaire à ces échanges. En effet, transférer des données contigües dans la mémoire physique (ce qui correspond à une décomposition selon la direction la plus "lente", en l'occurrence z) est beaucoup plus rapide que les transferts effectués dans les autres directions. Ceci est illustré par les mesures rapportées dans le tableau 4.9 qui représente les temps de transfert détaillés pour chaque direction pour des stencils de taille 4 et 8. On note ainsi que les transferts selon x peuvent représenter jusqu'à 80% du temps total de transfert. Ceci nous a conduit à considérer différentes techniques pour optimiser ces transferts pour essayer de minimiser l'impact de la géométrie de la décomposition sur les temps de calcul.

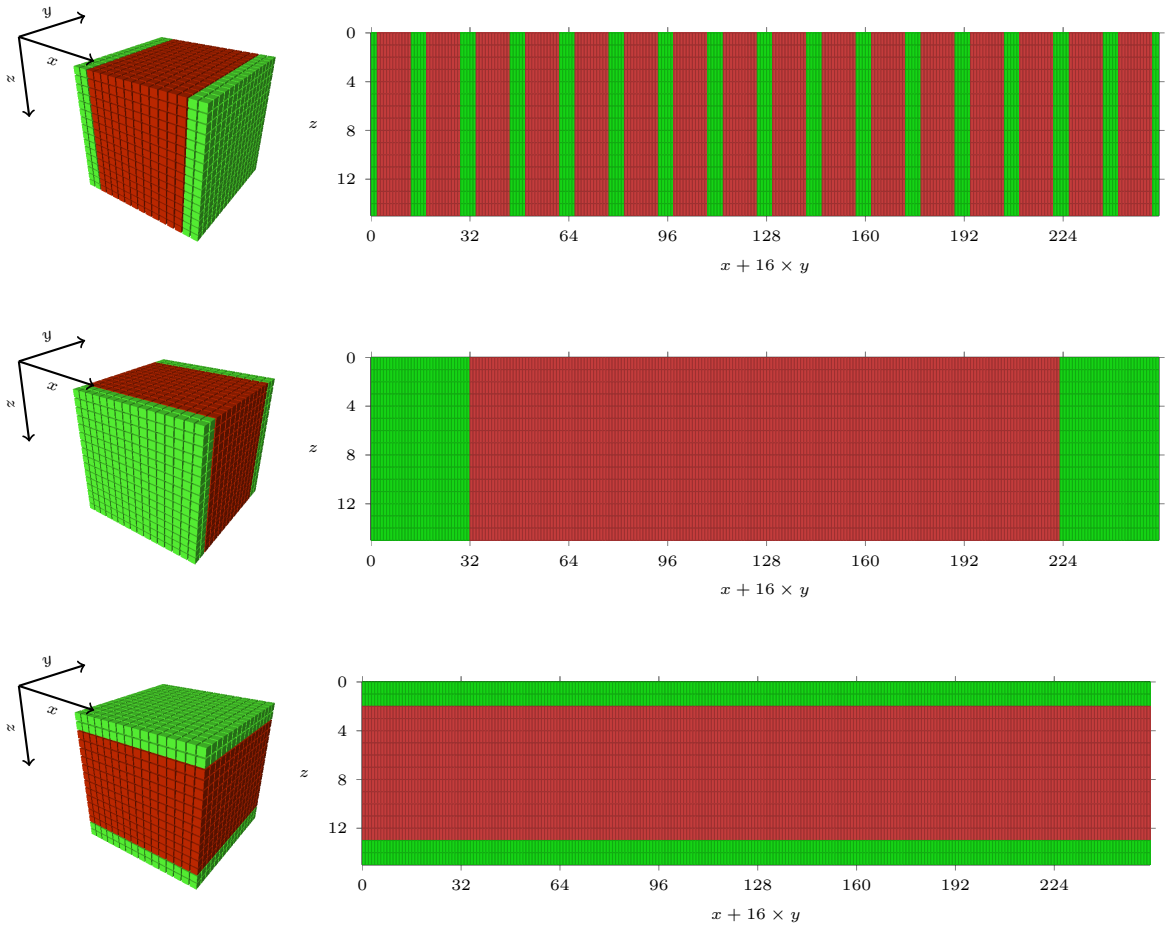


FIGURE 4.12 – Localisation en mémoire des nœuds fantômes pour un sous-domaine de dimensions 16^3 et des transferts selon les directions x (haut), y (centre) et z (bas).

TABLE 4.10 – Temps de transferts (en ms) du nœud hôte vers le GPU (HtoD) et du GPU vers le nœud hôte (DtoH) en utilisant différentes techniques.

Technique	HtoD	DtoH
Single Transfer	90.92	96.35
MemCpy2D	63.34	96.34
MemCpy2D+Transpose	27.18	31.23
Single MemCpy3D	93.51	98.35
Partial Transfer 3D	55.39	215.63

Sont reportés dans le tableau 4.10 les temps de transfert hôte-GPU correspondants à différentes techniques et possibilités offertes par l'API CUDA pour le même cube de taille $512 \times 512 \times 512$ et des bords de 8 éléments. On considère tout d'abord le transfert de la totalité du cube (Single Transfer) en un seul transfert. On utilise ensuite des copies mémoire 2D en itérant selon la dimension la plus lente pour ne copier que la zone halo (MemCpy2D). Ceci a pour effet de réduire la taille globale des données transférées mais augmente largement le nombre de transferts et ainsi les temps de latence. Comme décrit précédemment, jusqu'à 80% des temps de transferts sont dédiés aux transferts selon x. Pour réduire ce coût, une troisième stratégie (Memcpy2d+Transpose) a été mise en place consistant à réorganiser les données avant de les transférer à l'aide d'un *kernel* côté GPU qui copie ces données vers une zone mémoire contigüe. Cela revient à transposer les bords du domaine. Cette technique a permis de réduire les temps de transfert hôte-GPU par 2.33 et par 3 dans le sens GPU-hôte. L'utilisation des primitives de copies 3D pour transférer tout le domaine (Single MemCpy3D) ou seulement ses bords (Partial Transfer 3D) a montré des performances limitées.

On peut également souligner que les temps de transfert dépendent de la taille du stencil comme indiqué sur la figure 4.13.

4.3.2 Recouvrement entre calcul et communications

Nous proposons dans ce qui suit des optimisations supplémentaires qui peuvent améliorer la scalabilité de l'application en recouvrant les communications nécessaires dans le cadre d'une décomposition en sous-domaines par les calculs sur GPU.

Recouvrement des communications hôte-GPU

Il est possible d'exploiter la possibilité dont disposent certains GPU CUDA (de *Compute Capability* supérieure à 1.1) de faire des transferts mémoire et d'exécuter des noyaux de calcul en simultané (*cuncurrent copy and execution*) pour recouvrir les échanges de données des nœuds fantômes entre hôte et GPU par les calculs des zones internes. Pour faciliter ce genre d'optimisations, CUDA reprend les notions de calcul par flux (cf. 3.3.1) et définit les flux CUDA (*CUDA*

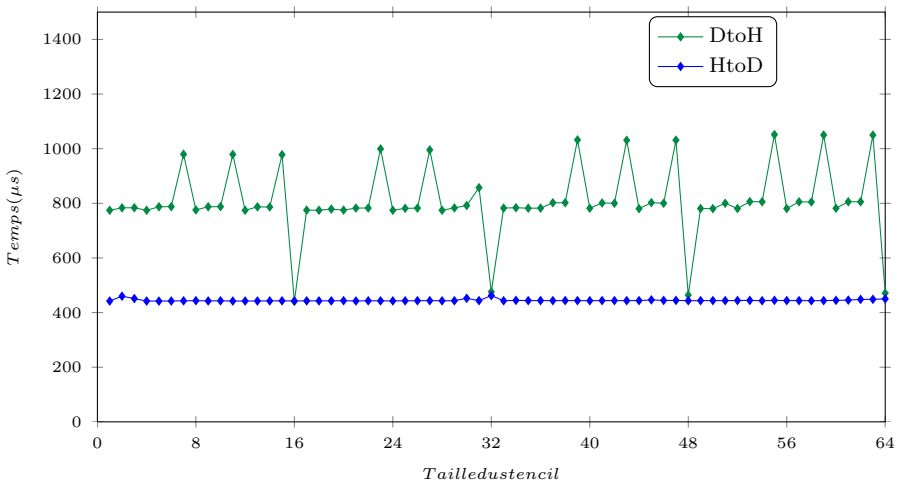


FIGURE 4.13 – Temps de transfert en fonction de la taille du stencil.

streams) comme une séquence d'opérations s'exécutant sur GPU selon l'ordre dans lequel elles sont lancées dans le code s'exécutant sur le nœud hôte. Cet ordre est respecté pour des opérations appartenant au même flux, mais si les ressources matérielles le permettent, l'exécution d'opérations appartenant à des flux différents peut se faire de façon simultanée. Selon la génération de GPU, on peut s'attendre à des comportements différents pour ce type d'optimisation. Les GPU d'architecture Tesla disposent par exemple d'un seul gestionnaire de transfert de données (*copy engine*) depuis et vers le GPU, alors que sur les GPU Fermi, chaque sens de transfert dispose de son propre gestionnaire ce qui autorise des transferts simultanés depuis et vers le GPU.

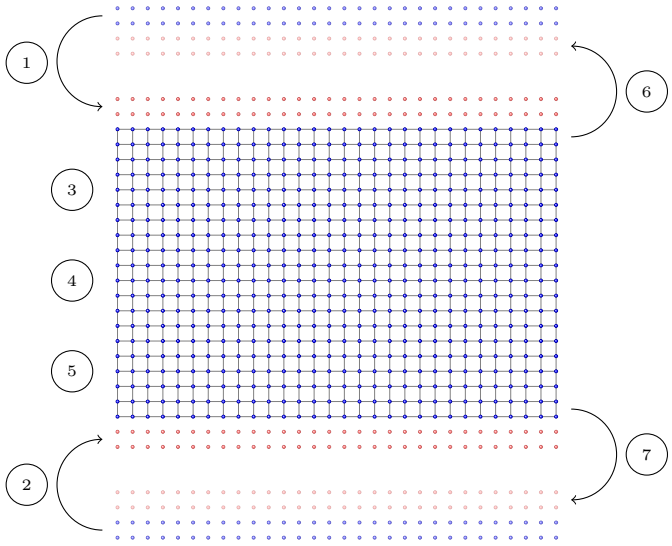


FIGURE 4.14 – Exemple d'une séquence de tâches pour mettre à jour un sous-domaine pour une itération en temps : copie vers le GPU des zones fantômes provenant des voisins nord (1) et sud (2), mise à jour du sous-domaine en utilisant les données du voisin nord (3), du sous-domaine lui-même (4) et du voisin sud (5), enfin copie à partir du GPU des zones fantômes du voisin nord (5) et sud (6).

Si on se place dans le cadre d'une décomposition du domaine global en sous-domaines selon

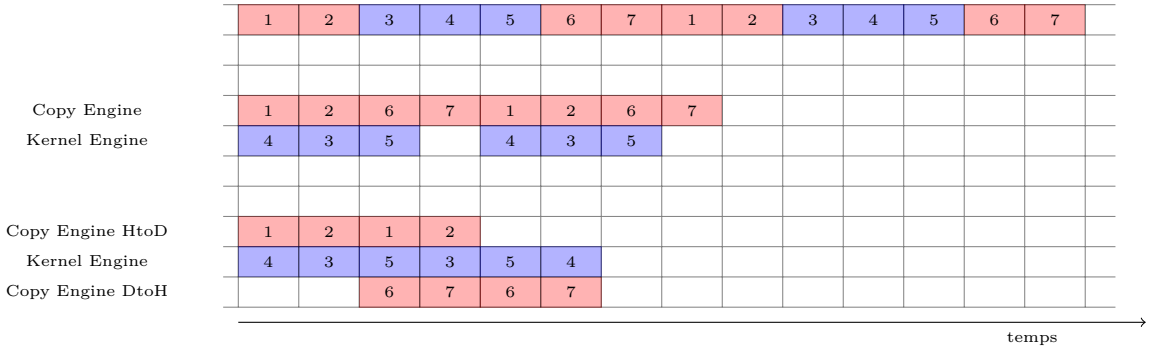


FIGURE 4.15 – Exemples d'exécution de deux itérations en flux selon l'architecture du GPU utilisé : en série (haut), avec une file de transferts unique et une file d'exécution (milieu) et avec deux files de transferts et une file d'exécution (bas).

la direction z, on peut effectuer la mise à jour de l'un des sous-domaines disposant de deux voisins (appelés ici nord et sud) par la séquence de tâches décrite à la figure 4.14. En reprenant la numérotation de la figure 4.14, cette séquence peut être décomposée en trois séquences indépendantes qui peuvent être assignées à des flux CUDA différents :

1. mise à jour de la zone interne (4) ;
2. copie vers le GPU des zones fantômes provenant du voisin nord, mise à jour de la zone qui dépend de ces données et copie vers l'hôte de la zone fantôme du voisin nord (1-3-6) ;
3. copie vers le GPU des zones fantômes provenant du voisin sud, mise à jour de la zone qui dépend de ces données et copie vers l'hôte de la zone fantôme du voisin sud (2-5-7).

La figure 4.15 montre les enchaînements simultanés que l'on peut obtenir en utilisant les architectures précédemment citées dans le cas où toutes ces tâches sont supposées être de même durée (dans le cas général, la même logique s'applique pour recouvrir les tâches de plus courte durée). On voit que lorsqu'on dispose de deux files de transfert (GPU vers hôte et hôte vers GPU), on peut espérer recouvrir totalement les coûts des échanges entre hôte et GPU.

Recouvrement des communications entre sous-domaines voisins

On peut étendre la démarche précédente en utilisant des communications MPI non bloquantes et recouvrir ainsi les temps de communication entre sous-domaines voisins. Cette technique n'a pas été mise en œuvre dans le cadre de cette étude.

4.4 Validation numérique

Pour chacune des implémentations précédemment décrites, les résultats ont rigoureusement été comparés à ceux obtenus avec les implémentations non accélérées. Seuls des écarts négligeables ont pu être observés et ceci à cause des modes de calcul légèrement différents entre CPU et GPU. A titre d'exemple, la figure 4.16 montre que pour une modélisation sismique, la

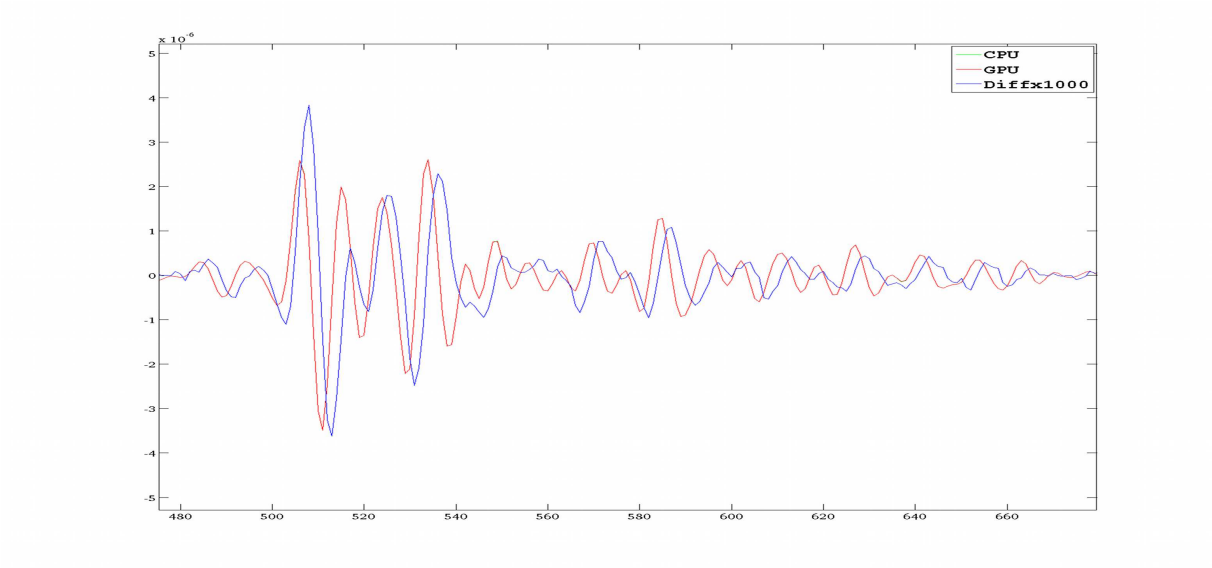


FIGURE 4.16 – Validation numérique des résultats de la modélisation sismique sur GPU : les traces sismiques produites sur CPU et sur GPU sont confondues, la différence relative est de l'ordre du 1 pour mille essentiellement pour des valeurs proches de zéro, elle est négligeable pour les valeurs significatives.

différence relative entre traces sismiques calculées sur CPU et sur GPU est négligeable pour les valeurs significatives et de l'ordre de 1 pour mille pour les valeurs proches de zéro ce qui reste largement acceptable pour de telles simulations.

4.5 Valorisation en production : étude comparative de différents dispositifs d'acquisition

Les performances prometteuses démontrées lors de cette étude ont motivé l'acquisition d'une installation importante équipée de 256 GPU. Cette installation a permis de mener plusieurs études de production. Elle a également permis d'envisager des études plus ambitieuses en terme de taille des simulations. Tel est le cas de la modélisation de nouveaux types d'acquisitions sismiques. A titre d'exemple, nous sommes parvenus à démontrer qu'en mettant en place des solutions logicielles adéquates tirant le meilleur parti des installations et des architectures disponibles, une étude comparative rapide de différents types d'acquisitions sismiques devenait faisable. Nous détaillons dans ce qui suit cette étude à titre d'exemple de cas d'application.

4.5.1 Dispositifs d'acquisition modélisés

En acquisition sismique marine, les hydrophones sont placés sur de longs câbles appelés flûtes ou *streamers* remorqués par un navire naviguant selon une trajectoire composée de lignes parallèles. Le positionnement de ces hydrophones, ainsi que celui de la source, définissent la direction

d'acquisition ou azimut, orientation selon laquelle sont illuminées les formations du sous-sol. En sismique conventionnelle les données sont acquises selon un même azimut. On parle alors d'acquisitions NATS (*Narrow Azimuth Towed Streamers*). Les cibles sont alors "éclairées" selon une seule direction principale ce qui donne des images sismiques de qualité limitée permettant difficilement d'imager les dômes de sel notamment. Les technologies dites *Wide Azimuth* (WAZ) ou *Wide Azimuth Towed Streamers* (WATS) permettent en revanche de combiner des "prises de vue" d'un même point selon différents azimuts et de s'affranchir des écrans sismiques pour obtenir une image beaucoup plus fidèle à la réalité géologique grâce à cette couverture angulaire plus large. Cependant ce type d'acquisition nécessite l'utilisation simultanée de plusieurs navires et reste complexe à mettre en œuvre, ce qui lui confère un coût élevé en temps et en ressources. Les acquisitions *Coil Shooting* (littéralement tir en bobine) [25] introduites en 2008 permettent quant à elles, d'obtenir une couverture angulaire complète (*full azimuth*) et ceci grâce à l'utilisation d'un navire similaire à celui utilisé en NATS, mais décrivant des trajectoires circulaires. Ce type d'acquisition permet en outre de réduire les temps et donc les coûts liés aux changements de lignes d'acquisition. Il nécessite cependant un nombre de points de tir relativement important ainsi que des traitements plus complexes des données [73]. En effet, les données enregistrées peuvent être fortement bruitées du fait qu'une partie des récepteurs est souvent confrontée aux courants dominants et ce eu égard à la trajectoire en arc des *streamers*. D'autre part, la couverture n'étant pas uniforme, des traitements adaptés doivent être appliqués [141].

Nous nous intéressons dans cette partie à démontrer la faisabilité d'une comparaison rapide de ces différents dispositifs -NATS, WATS et Coil- pour une cible identifiée en réalisant les modélisations sismique pour chaque acquisition. L'objectif est d'adapter notre application à la spécificité de chaque type d'acquisition pour tirer le meilleur parti des infrastructures disponibles.

Pour l'acquisition NATS, on simule une acquisition avec un navire disposant de 10 *streamers* de $8km$ de long et séparés de $100m$, et navigant le long de 9 lignes parallèles. Ces lignes sont séparées de $500m$ et un point de tir est effectué tous les $100m$ pour un total de 1638 points de tir. Pour la WATS, en utilisant quatre navires naviguant le long des mêmes lignes et deux sources on parvient à simuler le même dispositif qu'en NATS mais avec une couverture angulaire plus large. Quantau dispositif d'acquisition Coil, il s'agit de considérer un navire décrivant 18 cercles pour effectuer 2200 points de tir. Le navire remorque 10 *streamers* d'une longueur de 8km chacun disposés à $100m$ d'intervalle. Les hydrophones disposés selon ces *streamers* sont séparés de $12.5m$. La source est placée à $200m$ du premier récepteur. Les centres des cercles de $6km$ de rayon sont distants de $2km$ et les points de tirs sont effectués à intervalles réguliers de $300m$ le long de ces cercles.

4.5.2 Exécution

Contrairement aux modélisations de type NATS et WATS qui ne posent pas de contraintes particulières, les dimensions des modèles simulés, dans le cas de l'acquisition Coil, pour différents points de tir varient continuellement et de façon importante. Pour traiter les 2200 points de tir,

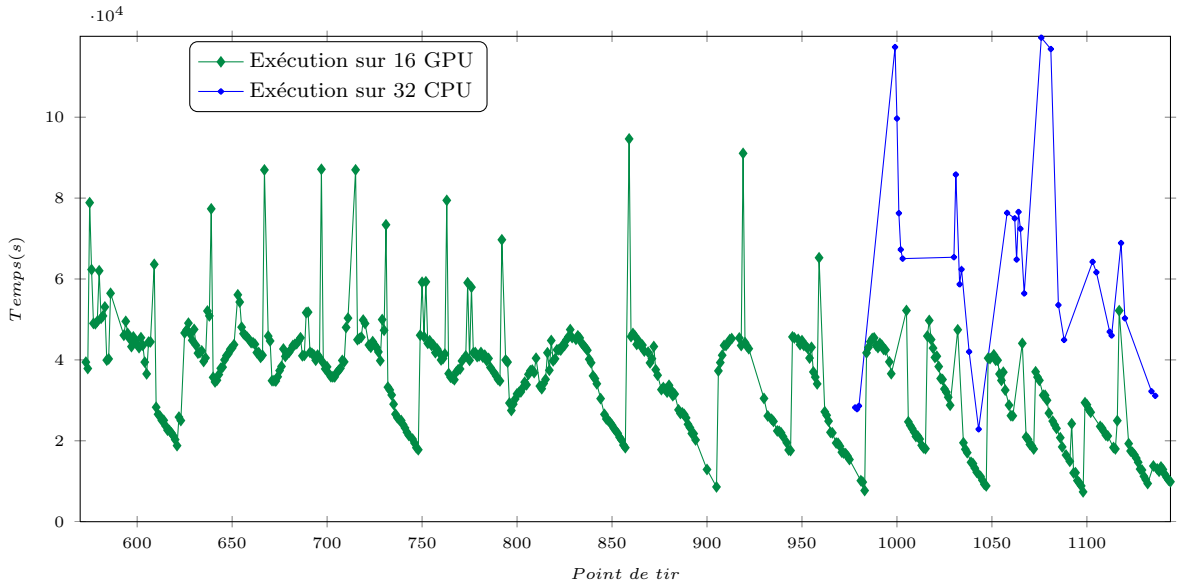


FIGURE 4.17 – Temps de modélisation en secondes par point de tir pour un dispositif Coil sur GPU et CPU.

un problème d'équilibrage de charge se pose donc lors de la distribution des points de tir. La stratégie mise en place a consisté à utiliser une solution client/serveur permettant un pilotage dynamique de l'ensemble des ressources disponibles et des traitements à effectuer : un serveur affecte un point de tir à chaque client (ensemble de nœuds) lui indiquant qu'il est disponible. A la différence d'une répartition statique des tâches, cette approche permet de solliciter l'ensemble des ressources allouées. La figure 4.17 donne une idée des variations des temps de traitement pour chaque point de tir. On note en particulier sur cette figure le rajout en cours d'exécution de ressources CPU (à partir du point de tir 960).

4.5.3 Résultats

L'objectif premier de cette section a été de démontrer la faisabilité de ce genre d'études comparatives dans un temps raisonnable et la maîtrise complète des différents outils nécessaires (conception des acquisitions, architecture client-serveur, etc.). Cet objectif a été pleinement atteint vu que l'étude a pu être menée en moins de trois semaines. Pour ce qui des résultats en terme d'image sismique produite, l'étude a confirmé que des traitements particuliers du résultat de la modélisation Coil étaient nécessaires pour obtenir une image sismique de bonne qualité. Sans ces traitements, c'est l'image produite par la migration des données générées par la modélisation WATS qui permet d'imager le mieux les différents réflecteurs. Suite à cette étude, une acquisition réelle Coil dans le golf de l'Angola a été conduite [73].

4.6 Schémas d'ordres plus élevés et méthode pseudo-spectrale

Pour résoudre l'équation d'onde en 3D, nous avons considéré jusqu'ici l'utilisation des différences finies avec un schéma d'ordre 2 en temps et 8 en espace. Nous proposons dans cette section d'étudier les alternatives à ce schéma pour une implémentation GPU.

4.6.1 Influence de l'ordre du schéma : étude théorique

Nous avons précédemment évoqué le fait que la montée en fréquence pour un schéma numérique d'ordre p donné engendre une évolution en $O(f_{max}^4)$ des calculs où f_{max} est la fréquence maximale de l'onde à propager. Ceci est dû aux conditions de stabilité et à la dispersion numérique du schéma qui imposent un nombre minimum de points par longueur d'onde minimale $\lambda_{min} = \frac{c_{min}}{f_{max}}$ (ce nombre est noté PPW(p) pour *Points Per Wavelength*) et une proportionnalité entre les pas de discrétisation en temps et en espace ($\Delta t = \alpha(p)\Delta h$). Ces conditions dépendent de l'ordre du schéma numérique considéré. Généralement, plus l'ordre du schéma est élevé, plus la grille de discrétisation peut être lâche. On retrouve dans [63] une quantification du nombre de points de grille nécessaires pour différents schémas à dispersion numérique constante. Déterminer quel est l'ordre du schéma numérique le plus adapté à la résolution d'un problème donné n'est pas un exercice trivial. En effet, selon l'architecture considérée, la simple complexité calculatoire est à pondérer par la complexité en mémoire, les mouvements de données nécessaires, etc.

Pour les plateformes GPU, nous avons démontré que le principal goulot d'étranglement réside dans l'accès à la donnée dans la mémoire globale du GPU. En considérant l'algorithme mis en place pour la résolution de l'équation d'onde en 3D dans le cas d'un domaine à densité constante, on peut exprimer le nombre N_{Gmem} des accès à la mémoire globale du GPU pour mettre à jour un élément de la grille en fonction de l'ordre p en espace du schéma utilisé, et ce pour des blocs de 256 *threads* (16×16) de la façon suivante :

$$N_{Gmem}(p) = \frac{4 \times 256 + (4 \times \frac{p}{2} \times 16)}{256}. \tag{4.3}$$

Le nombre de points de grille N^3 pour un problème de taille physique D^3, s'écrit en fonction de f_{max}, de la vitesse minimale dans le domaine c_{min} et de $PPW(p)$:

$$N^3 = (\frac{D}{\Delta h})^3 = (\frac{D}{\frac{c_{min}}{f_{max} * PPW(p)}})^3. \tag{4.4}$$

Le nombre de points N_{sim} à mettre à jour pour une simulation d'une durée T s'écrit alors :

$$N_{sim} = N^3 \frac{T}{\Delta t} = (\frac{D}{\frac{c_{min}}{f_{max} * PPW(p)}})^3 \frac{T}{\alpha(p)\Delta h}. \tag{4.5}$$

On a alors pour un domaine de taille donnée et une fréquence maximale fixée :

$$N_{sim} = K\ G(p) \tag{4.6}$$

avec $K = TD^3(f_{max}/c_{min})^4$ et $G(p) = \frac{PPW(p)^4}{\alpha(p)}$.

Et le nombre total des accès à la mémoire pour une implémentation GPU

$$N_{Total\_Gmem}(p) = K\ G(p)\ N_{Gmem}(p). \tag{4.7}$$

$\alpha(p)$ est calculée à partir de la relation 2.16 en utilisant l'algorithme présenté en [52]. En considérant la fonction $PPW(p)$ donnée en [63], on peut alors calculer $N_{Total\_Gmem}(p)$ en fonction de l'ordre p en espace du schéma numérique considéré pour une implémentation GPU. Ces éléments sont présentés dans le tableau 4.11. Si on considère que la performance de ces implémentations est inversement proportionnelle à $N_{Total\_Gmem}(p)$, on peut alors calculer le rapport entre la performance de chacune de ces implémentation et celle du schéma 2-8. Ce ratio est indiqué dans la cinquième colonne du tableau. Les résultats obtenus montrent que l'utilisation de schémas d'ordres élevés permet de réduire le nombre total d'accès à la mémoire et d'améliorer ainsi les performances. Ces schémas sont cependant plus complexes à mettre en œuvre. Il s'agit donc de trouver le meilleur compromis entre performance et simplicité de programmation. Le schéma d'ordre 12 semble de ce point de vue le plus adapté à une implémentation GPU. En effet, le passage de l'ordre 8 à l'ordre 12 devrait permettre de diviser par 3 les temps de calculs. Alors que passer de l'ordre 12 à l'ordre 30, n'offrirait selon ce modèle qu'un facteur d'accélération de 2 avec une complexité de mise en œuvre beaucoup plus élevée.

Cette analyse simplifiée est bien entendu à compléter pour prendre en compte l'impact de l'utilisation d'ordres plus élevés sur les besoins en mémoire, la décomposition en sous-domaines, les communications hôte-GPU, etc. Pour ce qui est de la décomposition en sous-domaines par exemple, des grilles plus lâches permettent de simuler un même domaine physique par des grilles numériques de tailles plus petites et requérant ainsi une décomposition en un nombre moindre de sous-domaines. Cependant, la taille des données à communiquer entre sous-domaines, qui est proportionnelle à la largeur du stencil, est plus importante. Il s'agit là aussi de déterminer le meilleur compromis en fonction de l'architecture matérielle. Bien que simplifiée, cette analyse permet cependant de souligner un des aspects de la dépendance forte qui existe entre l'architecture étudiée et le schéma numérique à considérer.

4.6.2 Méthode pseudo-spectrale

On a démontré dans la section précédente que des schémas numériques d'ordres élevés peuvent être indiqués pour l'implémentation GPU de la résolution de l'équation d'onde par différences finies puisqu'ils permettent un échantillonnage plus lâche du domaine considéré. Les méthodes pseudo-spectrales permettent en théorie d'aller encore plus loin en réduisant le pas d'échantillonnage à deux points par longueur d'onde minimale. Par ailleurs, à l'instar des bibliothèques optimisées sur CPU (MKL, FFTW, etc.), des implémentations performantes de la transformée de Fourier sur GPU existent. Nous comparons ici les performances de la bibliothèque CUFFT aux bibliothèques pour CPU et étudions son utilisation pour résoudre l'équation d'onde.

Ordre p	$\alpha(p)$	$PPW(p)$	$N_{Total\_Gmem}(p)$	Ratio
2	6.93	20.00	98124.10	676.90
4	8.00	7.00	1350.56	9.32
6	8.52	4.60	249.62	1.72
8	8.83	4.00	144.96	**1**
10	9.05	3.40	77.52	0.53
12	9.21	2.85	39.40	0.27
14	9.34	2.70	32.72	0.23
16	9.44	2.60	29.05	0.20
18	9.52	2.51	26.06	0.18
20	9.60	2.47	25.20	0.17
22	9.66	2.41	23.57	0.16
24	9.71	2.37	22.74	0.16
26	9.76	2.30	20.79	0.14
28	9.80	2.28	20.68	0.14
30	9.84	2.27	20.91	0.14

TABLE 4.11 – Performances modélisées pour des schémas numériques d'ordres 2 jusqu'à 30. Sont reportés à la colonne Ratio les facteurs d'accélération du schéma de référence par rapports autres schémas.

Transformée de Fourier rapide sur GPU

Diverses implémentations de la transformée de Fourier rapide (FFT) sur GPU existent [10, 100, 102]. Nous étudions dans cette section les performances de la bibliothèque CUFFT [102] proposée par NVIDIA et fournie avec CUDA dans l'optique de son utilisation pour la résolution de l'équation d'onde selon l'approche pseudo-spectrale qui utilise de façon intensive les transformées de Fourier. Nous mesurons les performances obtenues en utilisant la bibliothèque CUFFT (CUDA version 5.5) en 2D et en 3D pour des données de différentes tailles sur une carte Tesla C2075. Ces mesures sont comparées à celles obtenues avec les bibliothèques MKL d'Intel et FFTW qui s'exécutent de façon parallèle (*multi threads*) sur un processeur Intel Westmere. Les résultats obtenus sont reportés sur la figure 4.18. On reporte dans le cas de l'utilisation du GPU les mesures obtenues en prenant en compte les communications hôte-GPU (courbes CUFFT) et celles qui ne prennent en compte que le temps de calcul des FFT sur GPU (courbes $CUFFT_{no\_comm}$)

Ces résultats montrent que la bibliothèque CUFFT permet d'atteindre des performances jusqu'à dix fois plus élevées en 2D et en 3D lorsqu'elle est utilisée en remplacement des bibliothèques optimisées pour CPU. Dans ce cas aucun portage du reste du code n'est requis

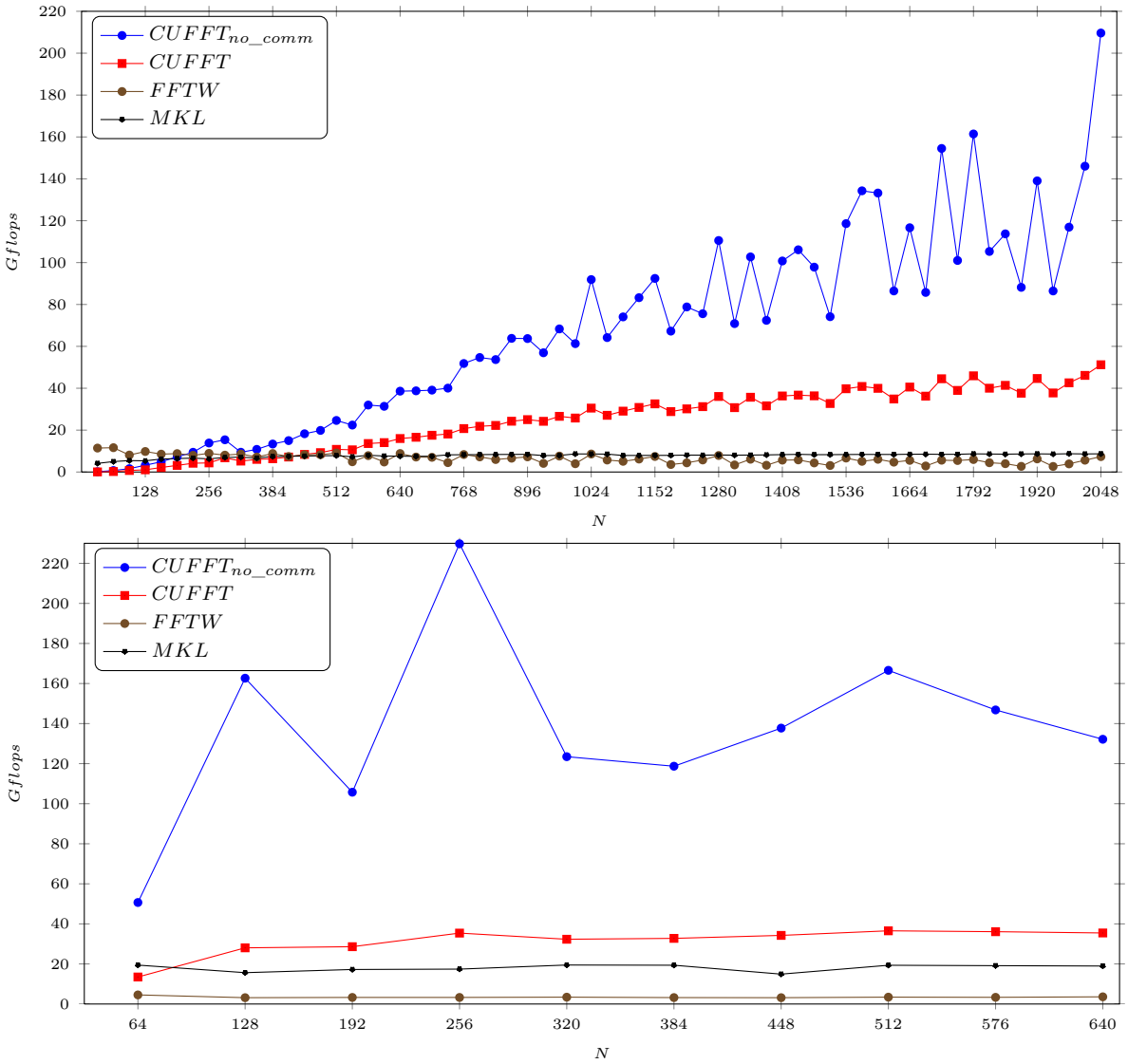


FIGURE 4.18 – Performances de différentes implémentations des FFT en 2D (haut) pour des domaines de taille N^2 et en 3D (bas) pour des domaines de taille N^3.

cependant les données d'entrée et de sortie doivent être transférées à chaque appel à la bibliothèque. Si les données résident dans la mémoire du GPU tout au long de l'application (pas de transferts, courbes $CUFFT_{no\_comm}$) le rapport entre les performances GPU et CPU est encore plus important. Pour les données de grande taille ce rapport peut être supérieur à 40 et atteint même 70 en 3D pour des cubes de taille 256^3 (rapport entre CUFFT et FFTW). Ces résultats nous encouragent à revisiter les méthodes pseudo-spectrales pour évaluer l'apport des GPU.

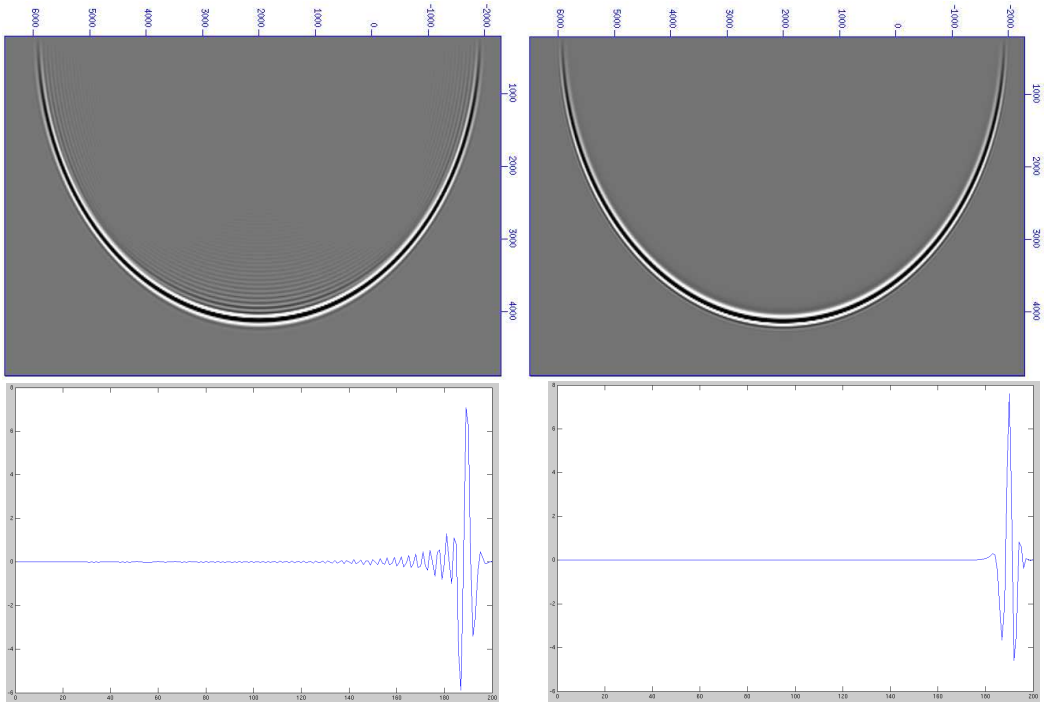


FIGURE 4.19 – Propagation d'un Ricker par différences finies (à gauche) et en utilisant l'approche pseudo-spectrale (à droite) pour $PPW = 2.5$

Cas 2D

L'utilisation des méthodes pseudo-spectrales dans le cas 2D est très simple. Il s'agit de remplacer le calcul du Laplacien par différences finies par :

$$\frac{\partial^2 U}{\partial x^2} + \frac{\partial^2 U}{\partial y^2} = FFT_x^{-1}[-{k_x}^2 FFT_x[U]] + FFT_y^{-1}[-{k_y}^2 FFT_y[U]] \tag{4.8}$$

Pour illustrer l'apport de cette méthode en terme de dispersion numérique par rapport à l'utilisation des différences finies, la figure 4.19 montre ce que donne chacune de ces méthodes pur le même échantillonnage spatial (2.5 points de grille par longueur d'onde minimale). On note la dispersion numérique importante dans le cas des différences finies engendrée par le sous échantillonnage spatial.

En terme de temps de calcul, et à dispersion numérique équivalente, l'implémentation GPU permet de réduire d'un ordre de grandeur le temps de calcul nécessaire à la modélisation.

Cas 3D

Le cas 3D peut être traité de façon similaire au cas 2D en calculant les trois dérivées spatiales dans le domaine de Fourier. On observe dans ce cas des temps de calcul équivalents à l'utilisation des différences finies pour une meilleure précision numérique. Cependant, cette approche est limitée du fait que la décomposition en sous-domaines engendre des communications importantes

entre sous-domaines pour pouvoir calculer les trois dérivées spatiales.

Des méthodes dites hybrides [33] permettent de combiner la précision numérique des méthodes pseudo spectrales et la scalabilité des différences finies en effectuant le calcul des dérivées spatiales selon deux directions dans le domaine de Fourier et la troisième par différences finies. L'utilisation de méthodes hybrides n'a pas été abordée dans nos travaux et pourra faire l'objet d'une étude ultérieure.

4.7 Etude d'une solution FPGA pour la modélisation sismique FDTD : Convey HC-1

Nous décrivons ici brièvement une étude menée pour évaluer la possibilité d'utiliser la plateforme FPGA Convey HC-1 pour la modélisation sismique. Cette étude est décrite en détail dans [3]. L'architecture de la plateforme Convey HC-1 a été exposée dans la section 3.3.1.

4.7.1 Transferts de données hôte-coprocesseur

Etant donnée la nature NUMA de l'architecture HC-1, disposer de mécanismes performants pour transférer les données entre nœud-hôte et coprocesseur est indispensable. Un gestionnaire de transferts (*Data Mover*) est proposé par Convey au sein du coprocesseur du HC-1 pour accélérer ces transferts. La figure 4.20 permet d'apprécier l'apport de ce mécanisme en comparant les débits mesurés en utilisant ce mécanisme et ceux obtenus pour des copies de données explicites effectuées par le CPU ou par le coprocesseur. Cependant, même avec ce mécanisme, les transferts via le FSB s'avèrent très lents. Les débits observés sont un ordre de grandeur en deçà de ceux observés pour les transferts entre CPU et GPU via le PCIe 2.0. Encore plus que pour les GPU, la réduction des transferts entre processeur et coprocesseur est donc à considérer avec soin. La plateforme Convey offre plus de possibilités à cet égard puisqu'elle permet d'embarquer jusqu'à 128 Go de RAM côté coprocesseur, ce qui permet de réduire les échanges avec le processeur hôte en gardant les données résidentes sur le coprocesseur. Il est également possible pour un code s'exécutant sur le coprocesseur d'accéder directement aux données présentes dans la mémoire de l'hôte. Pour évaluer la performance de ces accès nous comparons pour un programme simple s'exécutant sur le coprocesseur ($out[i] = in[i] + \alpha$) les temps d'exécution soit en effectuant les transferts de données en entrée et en sortie soit en exploitant cet accès directe. Les temps obtenus sont reportés sur le figure 4.21. Ils montrent que l'utilisation du gestionnaire de transferts du coprocesseur reste en moyenne trois fois plus rapide.

4.7.2 Bande passante mémoire

Comme pour les GPU, la bande passante mémoire est plus importante que pour les processeurs généralistes puisqu'elle est annoncée à 80 Go/s pour un débit moyen observé à 50 Go/s. Comme pour les GPU également, l'accès à la donnée peut constituer le principal goulet

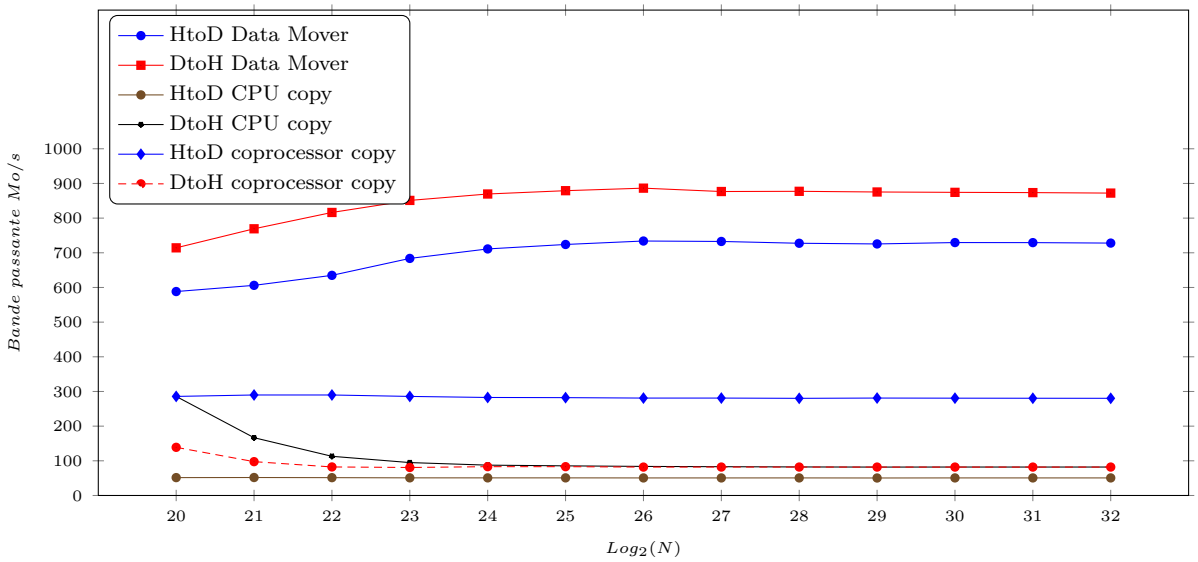


FIGURE 4.20 – Débits observés pour les transferts de données entre hôte et coprocesseur selon la technique de copie utilisée et le sens de ces transferts.

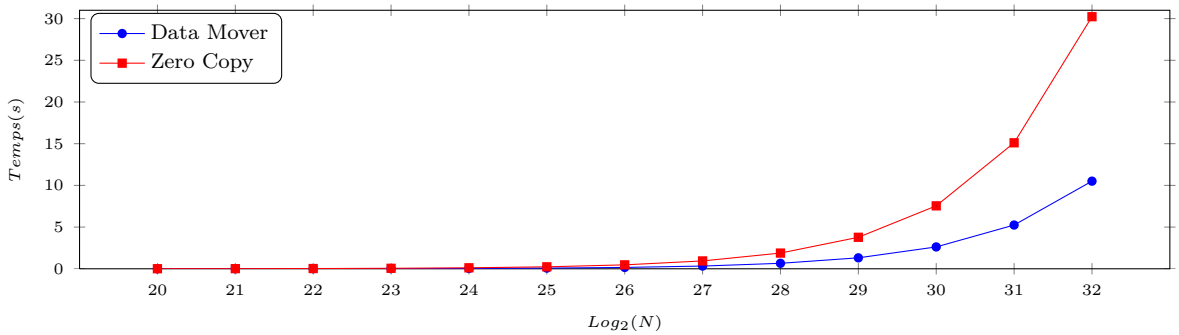


FIGURE 4.21 – Temps d'exécution en fonction de la taille des données en entrée et de la technique d'accès à la mémoire. L'accès à la mémoire locale après copie en utilisant le *Data Mover* est en moyenne trois fois plus performant que l'accès à la mémoire du nœud hôte.

d'étranglement, puisque pour atteindre la performance crête de 97 Gflops en calcul simple précision il faut effectuer 8 opérations par accès en mémoire, tandis que le ratio est de 1.5 pour la modélisation sismique par différences finies en 3D.

4.7.3 Résolution de l'équation d'onde par différences finies

Nous nous intéressons à l'implémentation de la résolution de l'équation d'onde par différences finies dans un milieu à densité constante selon le schéma d'ordre deux en temps et huit en espace décrit dans les sections précédentes. La génération de code exécutable sur FPGA se fait en annotant le programme d'origine par des directives de compilation délimitant la zone à déporter sur l'accélérateur et permettant notamment d'allouer de la mémoire et de gérer les échanges de données. A partir du programme annoté, le compilateur génère un programme

pouvant s'exécuter sur CPU (cas de l'absence d'un coprocesseur) ou sur le coprocesseur FPGA. Certaines restrictions existent cependant sur la partie déportée sur le FPGA. Celle-ci ne peut notamment pas faire d'I/O ou lancer des appels système. Le jeux de directives proposé est relativement simple et offre peu de possibilités d'optimisation. Cependant, la modification du code d'origine peut avoir un impact sur les performances du code généré (ordre des boucles, placement dans des registres des coefficients du Laplacien à la place de l'utilisation de tableaux, etc).

Sont reportés dans la figure 4.22 les performances obtenues pour l'implémentation FPGA de la résolution de l'équation d'onde par différences finies pour l'intérieur du domaine simulé (Basic) et les bords absorbants (PML) selon la taille NX de la boucle vectorisée (directive `cny prefer_vector`). Ces performances sont comparées à celles obtenues pour une exécution parallélisée avec OpenMP du même code sur un processeur Nehalem à huit cœurs (nous utilisons huit *threads* OpenMP). Les performances obtenues restent à moins de 20% de la performance crête et représentent une accélération limitée relativement au coût de la solution et à la difficulté de programmation.

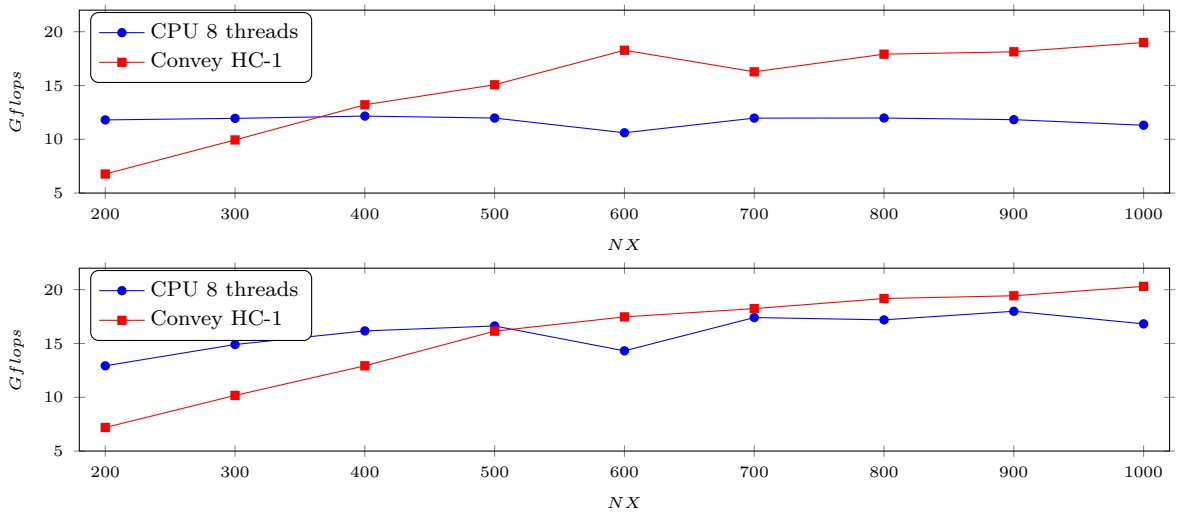


FIGURE 4.22 – Performances des implémentations Basic et PML sur Convey HC-1 et Nehalem 8 threads en fonction de la taille NX de la boucle vectorisée.

4.7.4 Conclusion

La plateforme Convey étudiée est limitée en termes de bande passante mémoire, de performance crête et même de débit des transferts hôte coprocesseur relativement aux solutions accélératrices GPU actuelles. Sa programmation reste complexe et fastidieuse (temps de compilation très long, difficulté à profiler et déboguer, etc.). Elle présente néanmoins des caractéristiques intéressantes tels que l'accès à la mémoire moins sensible aux motifs de ces accès et la taille importante que peut atteindre cette mémoire (jusqu'à 128 Go). Ces éléments permettent d'envisager des applications et des approches différentes que celles adoptées pour les GPU. A

titre d'exemple, grâce à la mémoire de taille plus importante, l'utilisation de la méthode pseudo-spectrale peut être envisagée pour des domaines de plus grande taille sans nécessiter le recours à l'utilisation des méthodes hybrides.

4.8 Problème inverse : Reverse Time Migration

Le principe de la RTM a été décrit dans la section 2.3. Il s'agit de localiser les contrastes de réflectivité en corrélant (condition d'imagerie) les champs d'onde correspondant à la propagation des signaux émis par la source et les récepteurs (géophones ou hydrophones) aux mêmes instants. Nous décrivons dans cette section notre implémentation de la RTM sur une plateforme GPU. Nous avons décrit dans les sections précédentes de ce chapitre différentes implémentations de la résolution de l'équation d'onde sur diverses plateformes ainsi que les performances et les limites associées. Le résolution de l'équation d'onde est la pierre angulaire de la RTM. Nous réutilisons donc les mêmes implémentations décrites précédemment pour calculer les champs d'onde de la source et des récepteurs.

4.8.1 Stratégies de remontée en temps inverse

Outre le fait de simuler simultanément la propagation de deux champs d'onde : celui de la source ainsi que celui des récepteurs, la RTM présente un autre challenge algorithmique important : disposer de ces deux champs aux mêmes temps t de la simulation afin d'appliquer la condition d'imagerie. Cette problématique n'est pas triviale puisque la simulation de la propagation du champ créé par la source s'effectue selon le sens des temps croissant (phase *foreward* notée FWD) alors que celui créé par les récepteurs qui est "réinjecté" dans le sous-sol s'effectue en remontant le temps i.e. dans le sens des temps décroissant (phase *backward* notée BWD). Pour disposer de ces deux champs aux mêmes instants plusieurs stratégies ont été proposées :

- sauvegarde des bords lors de la propagation du champ d'onde source et leur utilisation comme conditions aux limites pour recréer ce champ, trois simulations sont donc nécessaires : FWD source et BWD source et récepteurs ;
- sauvegarde sur disque d'un champ d'onde à intervalles réguliers et relecture au moment de l'application de la condition d'imagerie, seules deux simulations sont nécessaires : FWD source et BWD récepteurs mais un grand nombre d'I/O ;
- combinaison de façon optimale les deux approches précédentes : sauvegarder à intervalles plus lâches le champ d'onde source et recalculer des étapes intermédiaires dans la phase BWD [57, 125] ;
- conservation de l'énergie au sein du domaine sans impacter la condition d'imagerie à l'aide de bords aléatoires [39] par exemple, seules deux simulations sont nécessaires et aucune sauvegarde n'est nécessaire.

Selon l'approche suivie le goulot d'étranglement peut se situer au niveau du calcul ou des besoins mémoire et les I/O. Sur le papier c'est la quatrième méthode qui est la plus performante et la plus

TABLE 4.12 – Temps d'exécution en secondes des différentes phases de la RTM pour un point de tir avec 3700 itérations en temps sur une grille de taille $288 \times 118 \times 338$

	CPU			GPU			Ratio		
#SD	FWD	BWD	Total	FWD	BWD	Total	FWD	BWD	Total
1	923.93	1802.06	2726.00	73.99	92.79	166.78	12.49	19.42	16.35
2	526.93	953.27	1480.20	47.09	73.37	120.46	11.19	12.99	12.29
4	269.86	490.37	760.23	32.57	61.25	93.83	8.28	8.01	8.10
8	122.00	190.56	312.56	25.27	52.30	77.56	4.83	3.64	4.03

séduisante pour une implémentation GPU. Cependant les tests ont montré qu'il est difficile de garantir a priori que les réflexions sur les bords aléatoires n'auront pas d'effets sur la condition d'imagerie. Nous considérons la première stratégie pour l'implémentation GPU puisque c'est celle qui nécessite le moins de communications entre hôte et GPU au dépends du nombre des opérations à exécuter.

4.8.2 Implémentation GPU

L'implémentation GPU proposée effectue toutes les étapes de l'algorithme de migration sur le GPU : FWD du champ source, BWD des champs source et récepteur et condition d'imagerie. Cependant, les bords du domaine globale de la propagation du champ source doivent être sauvegardés lors de l'étape FWD et relus lors de l'étape BWD pour être utilisés comme conditions aux limites. L'algorithme de la RTM induit donc plus de communications hôte-GPU que celui de la modélisation. Nous avons cherché à recouvrir ces communications par les calculs suivant le technique d'utilisation des *streams* CUDA décrite p.86. Chaque *stream* consiste en une copie des données adéquates du CPU vers le GPU, une exécution de kernel et une copie de données du GPU vers le CPU. Pendant l'exécution d'un kernel donné, les transferts liés à d'autres kernels peuvent être entrepris recouvrant ainsi en partie leurs coûts.

4.8.3 Etude de performance

On reporte dans le tableau 4.12 les temps d'exécution des différentes phases des versions GPU et CPU de la RTM et les facteurs d'accélération observés pour différentes décompositions en sous-domaines pour un point de tir avec 3700 itérations en temps sur une grille de taille $288 \times 118 \times 338$. On regroupe dans les colonnes BWD les temps de rétropropagation des champs source et récepteurs ainsi que la condition d'imagerie. Les surcoûts induits par les communications supplémentaires nécessaires pour la RTM réduisent les facteurs d'accélération observés. L'utilisation de quatre *streams* permet d'accélérer l'application d'un facteur de 1.4.

Chapitre 5

Le GPGPU dans l'exploration pétrolière : au-delà de l'imagerie sismique

Sommaire

Si le GPGPU a fait son entrée dans le monde de l'exploration pétrolière grâce aux algorithmes d'imagerie sismique grands consommateurs de puissance de calcul et particulièrement bien adaptés au calcul sur accélérateurs graphiques, son utilisation s'est étendue au-delà de cette frontière. L'utilisation des GPU a permis en effet d'accélérer différents calculs de la chaîne reliant la modélisation d'acquisitions sismiques à la simulation de réservoirs pétroliers [96, 142] et en passant par l'imagerie sismique et l'interprétation des données qu'elle produit [67, 68, 69, 83]. Au-delà de cette puissance de calcul, c'est également les possibilités du calcul et du rendu interactifs qui peuvent être exploitées dans ces contextes [67, 70, 83].

5.1 Calculs d'attributs sismiques sur GPU pour l'interprétation sismique

L'utilisation du GPGPU dans le cadre du calcul des attributs sismiques comme aide à l'interprétation sismique a d'autant plus de sens que ces calculs se font usuellement selon un cycle itératif pour sélectionner les paramètres de calcul les plus adaptés, à savoir ceux qui permettent de faire ressortir au mieux les caractéristiques recherchées. Ce cycle est rendu fastidieux par la complexité des calculs et la taille des données en entrée et donc par le temps des traitements. Un premier intérêt de l'utilisation des GPU dans ce contexte réside donc dans la possibilité d'accélérer ces traitements. En outre, l'interopérabilité existante entre bibliothèques de calcul (type CUDA) et de rendu graphique (type OpenGL) peut permettre de réduire ce cycle en offrant le moyen de visualiser de façon interactive les attributs calculés et de modifier leurs paramètres à la volée [68, 69]. Ceci peut également permettre d'éviter le stockage des attributs calculés en les générant à nouveau au besoin à la volée. Les travaux décrits ici s'inscrivent dans cette démarche.

Nous commençons par décrire la mise en œuvre de la transposition de données 3D sur GPU, ce qui permettra de mettre en exergue les spécificités des GPU d'architecture Fermi utilisés dans ce chapitre et les optimisations qui sont propres à cette architecture (exploitation des niveaux de cache, grilles 3D et accès à la mémoire notamment). Nous illustrerons ensuite l'intérêt de l'utilisation des GPU pour le calcul d'attributs sismiques à travers quelques exemples de mise en œuvre. Nous rapporterons les résultats obtenus sur notre plateforme de calcul et montrerons qu'ils ouvrent la voie à l'utilisation des GPU pour le calcul interactif des attributs sismiques.

5.1.1 Transposition 3D sur GPU

Le tableau 5.1 rappelle les changements principaux apportés par l'architecture Fermi (cf 3.2.1 et [104, 131] pour plus de détails sur l'architecture Fermi et les optimisations qui lui sont propres).

Les calculs d'attributs sismiques exhibant souvent une direction de parcours privilégiée (traitements récursifs selon une dimension particulière, parcours des traces sismiques selon la direction du temps ou de la profondeur), nous commençons par introduire un algorithme de transposition 3D sur GPU simple et efficace qui nous permettra de nous affranchir de la disposition

GPU	Pré-Fermi (CC <2.0)	Fermi (CC=2.x)
Nombre de SP par SM	8	32
Nombre max. de *threads* par SM	1024	1536
Registres par SM	16 Ko	32 Ko
Banques de mémoire partagée	16	32
Banques de mémoire globale	8	6
Transactions en mémoire globale	32/64/128 octets	32/128 octets
Cache L1	-	0/16/48 Ko
Cache L2	-	764 Ko

TABLE 5.1 – Nouveautés introduites avec les GPU Fermi (CC=2.x).

des données d'entrée en mémoire. Il s'agira de permuter les dimensions d'un cube en 3D de la façon suivante $T_{in}(0:N1, 0:N2, 0:N3) \rightarrow T_{out}(0:N2, 0:N3, 0:N1)$ via une permutation $xyz \rightarrow yzx$. On pourra ensuite disposer par composition de transpositions la direction privilégiée à la position souhaitée.

Nous proposons une étude similaire à celle présentée par Micikevicius et Ruetsch pour le cas de la transposition en 2D [92]. Nous proposons ainsi de comparer notre implémentation à un noyau de calcul effectuant une simple copie en mémoire satisfaisant aux critères d'alignement en mémoire globale. Comme tout algorithme de transposition effectue au moins une lecture et une écriture par élément en entrée, sa performance ne pourra pas dépasser celle de la simple copie.

Dans un premier temps nous choisissons, de façon similaire à ce qui a été fait pour la modélisation en 3D, de parcourir le cube en entrée selon la dimension la plus "lente", *ie* celle pour laquelle les données sont les moins contigües en mémoire. Un *thread* traitera alors une colonne complète selon cette direction. Nous verrons dans un deuxième temps, grâce notamment à l'introduction des grilles 3D à partir des GPU d'architecture 2.0, une alternative plus simple et qui s'avèrera plus performante.

Comme à chaque fois, l'implémentation finale correspondra à une série d'optimisations apportées de façon itérative. Nous commençons ainsi par un algorithme (appelé Naïf) effectuant la lecture d'un élément en mémoire pour le réécrire à sa nouvelle position dans le cube transposé. Si les accès en lecture se font de façon contigüe en mémoire, les accès en écriture sont quant à eux non alignés. La première optimisation (algorithme appelé Optim1) repose sur l'utilisation de la mémoire partagée pour réaliser la transposition en effectuant des lectures et écritures contigües en mémoire globale comme illustré schématiquement sur la figure 5.1. Les données sont ainsi copiées à partir de la mémoire globale vers un tableau alloué en mémoire partagée accessible par un bloc de *threads*. Elles seront ensuite écrites de façon contigüe en transposant les *threads* au lieu de transposer les données. La technique classique du *padding* permet d'éviter la sérialisation des accès à la mémoire partagée (algorithme appelé Optim2) en éliminant les conflits se produisant lors de l'accès de *threads* différents à des données situées à des adresses différentes mais sur la même banque de mémoire partagée.

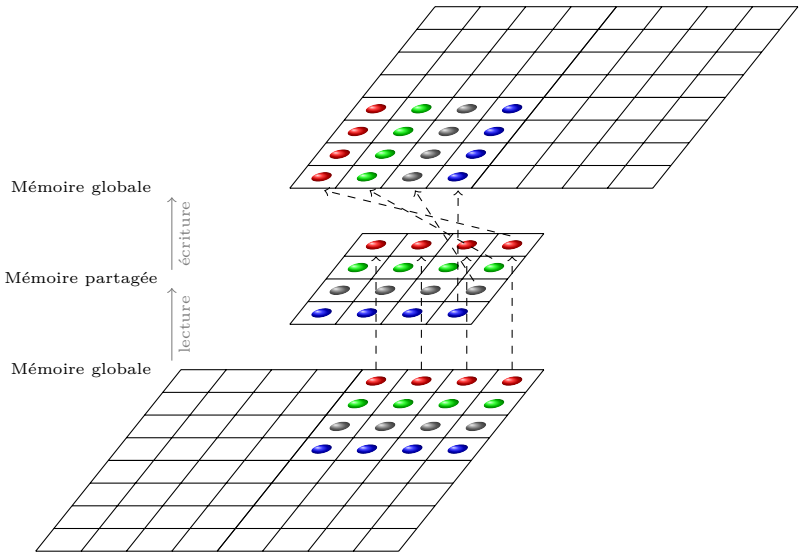


FIGURE 5.1 – Transposition en utilisant la mémoire partagée : lectures et écritures contigües en mémoire globale.

La deuxième alternative (algorithme appelé Grille 3D) exploite la possibilité d'utiliser facilement des grilles 3D à partir des cartes d'architecture 2.0. De cette façon, un bloc de *threads* va mettre à jour une seule tuile sans effectuer d'itérations sur la dimension la plus lente. En utilisant des blocs de *threads* de taille 16×16, la mise à jour de deux tuiles successives (algorithme appelé Grille 3D 2S) permettra d'exploiter encore mieux le cache L1 (lignes de 128 octets), ce qui augmente ainsi la granularité des traitements et permet de recouvrir les temps de lancement des calculs.

Le tableau 5.2 montre les résultats obtenus par les divers algorithmes pour des cubes de tailles 256^3, 512^3 et 768^3. La transposition effectuant uniquement des opérations de lectures et d'écritures (donc *bandwidth limited*), la mesure de performance pertinente est celle de la bande passante mémoire qui est reportée dans ce tableau (exprimée en Go/s). Si T représente le temps mesuré de la transposition, la bande passante mémoire est calculée pour un cube de données sismiques en simple précision selon la formule

$$N^3 \times \underbrace{2}_{R/W} \times \underbrace{4}_{float} / T \ .$$

Les mesures obtenues pour ces implémentations sont comparées à celles de la copie directe en mémoire globale. Hormis l'effet de ces optimisations, le tableau montre l'effet de l'utilisation du cache L1. Les résultats correspondants sont reportés dans les colonnes L1. Ce cache peut être désactivé avec une option de compilation, auquel cas on n'utilise que le niveau L2 ; les résultats correspondants sont reportés dans les colonnes L2. On remarque que dans certains cas, ceci améliore les performances. En effet, ceci peut réduire le nombre de préchargement de données non exploitées (*prefetch*) et donc le nombre global de transactions, les transactions se faisant alors avec une granularité de 32 octets au lieu des 128 octets correspondant à la taille de

la ligne de cache L1. Le tableau montre également pour la deuxième et troisième optimisation ainsi que pour l'implémentation en grille 3D, l'effet que peut avoir la réorganisation des blocs de *threads* pour éviter des accès simultanés de *threads* de blocs différents aux mêmes banques de mémoire globale mais à des adresses différentes (résultats correspondants dans les colonnes Camp et noCamp). La chute de performance liée à ces accès (appelée *partition camping* [5, 92]) est sensée être peu perceptible sur les GPU d'architecture Fermi où l'adressage en mémoire globale est *haché*. On remarque cependant que cette technique permet de maintenir une performance correcte pour des tailles de cube particulières (N=768 par exemple), là ou les performances chutent considérablement sans cette technique. Lors de tests plus avancés [4], on peut montrer que ces chutes de performances sont liées au nombre de banques de mémoire global (au nombre de 6 pour les Fermi).

N	256				512				768			
Partition	Camp		noCamp		Camp		noCamp		Camp		noCamp	
Cache	L1	L2	L1	L2	L1	L2	L1	L2	L1	L2	L1	L2
Copie	92.21	92.21	-	-	106.64	107.03	-	-	75.57	75.57	-	-
Naïf	14.77	14.44	-	-	13.54	13.65	-	-	9.41	9.55	-	-
Optim1	62.87	56.38	63.38	57.07	61.80	65.83	61.74	68.98	45.62	47.49	45.60	49.08
Optim2	65.49	58.49	67.71	59.22	61.80	67.35	76.27	69.63	45.84	47.73	61.81	49.45
Grille 3D	67.87	69.49	67.48	69.73	68.27	70.18	68.28	70.52	32.41	31.31	67.34	68.19
Grille 3D 2S	86.80	82.67	83.80	80.60	74.00	76.39	84.95	82.15	33.29	32.33	70.56	62.09

TABLE 5.2 – Performances moyennes exprimées en terme de bande passante mémoire (en Go/s) pour différentes techniques de transposition de cubes de tailles N^3 avec N=256, 512 et 768.

Les courbes de la figure 5.2 décrivent le comportement des différentes implémentations précédemment décrites pour des cubes de tailles variables en entrée.

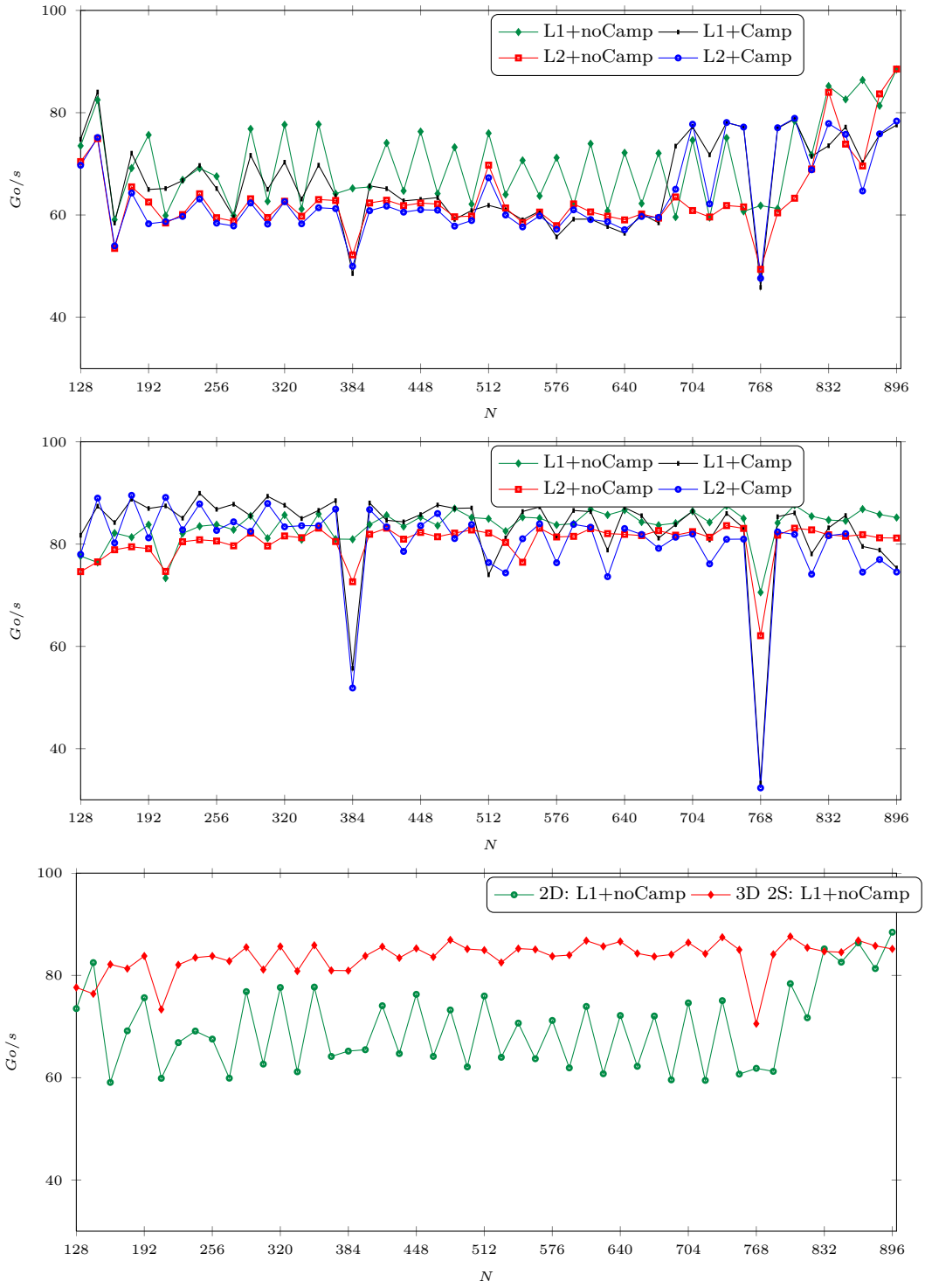


FIGURE 5.2 – Bandes passantes mémoire pour la grille 2D (haut) et 3D (milieu) pour différentes tailles du cube en entrée. La courbe du bas permet de comparer les deux meilleures implémentations 2D et 3D.

5.1.2 Similarité entre traces voisines

Nous présentons dans cette partie quelques exemples d'attributs sismiques basés sur la similarité entre traces voisines. Cette similarité peut être mesurée en utilisant la corrélation entre ces traces. Pour deux tableaux de valeurs $X(x_1, \ldots, x_n)$ et $Y(y_1, \ldots, y_n)$, le coefficient de corrélation normalisé *corr* est donné par

$$corr = \frac{\sigma_{xy}}{\sigma_x \sigma_y} = \frac{\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}} \tag{5.1}$$

avec

$$\begin{aligned} \sigma_{xy} &= \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(y_i - \bar{y}) \\ \sigma_x &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2} \\ \sigma_y &= \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y})^2} \end{aligned} \tag{5.2}$$

$\bar{x}$ et $\bar{y}$ étant respectivement la moyenne des valeurs de X et la moyenne de celles de Y.

Dans notre cas, si on note f_i^j le $j^{ième}$ élément de la $i^{ième}$ trace, les tableaux X et Y correspondant au calcul de la corrélation sur une fenêtre de taille $(2n + 1)$ entre f_i^j et f_p^q sont $X(f_i^{j-n}, \ldots, f_i^j, \ldots, f_i^{j+n})$ et $Y(f_p^{q-n}, \ldots, f_p^q, \ldots, f_p^{q+n})$.

La normalisation permet de s'affranchir de la différence d'amplitude entre les traces. On mesure donc la similarité entre les phases du signal des deux traces. La corrélation variera alors entre -1 et 1. Cette mesure servira de base pour le calcul de trois attributs sismiques : la cohérence, le gradient de corrélation ainsi que le pendage.

Attribut cohérence

L'attribut *cohérence* est une mesure moyenne de la similarité entre les traces voisines sur une fenêtre d'analyse verticale (en temps ou en profondeur). Pour chaque point du cube sismique 3D, nous comparons la variation d'amplitude de la trace centrée en ce point sur une fenêtre verticale en temps ou en profondeur avec celles des huit traces de son voisinage. Cette similarité est mesurée par le coefficient de corrélation (5.2) maximal sur le voisinage. La figure 5.3 représente une section d'un cube sismique et du cube cohérence correspondant en transparence. La cohérence permet de détecter les discontinuités telles que les failles ou les zones de fractures ainsi que les limites de corps géologiques tels que les chenaux. La figure 5.11 (haut) représente une

coupe en temps du même cube sismique (gauche) et du cube cohérence (droite). On y distingue clairement les failles et les chenaux mis en évidence par l'attribut cohérence.

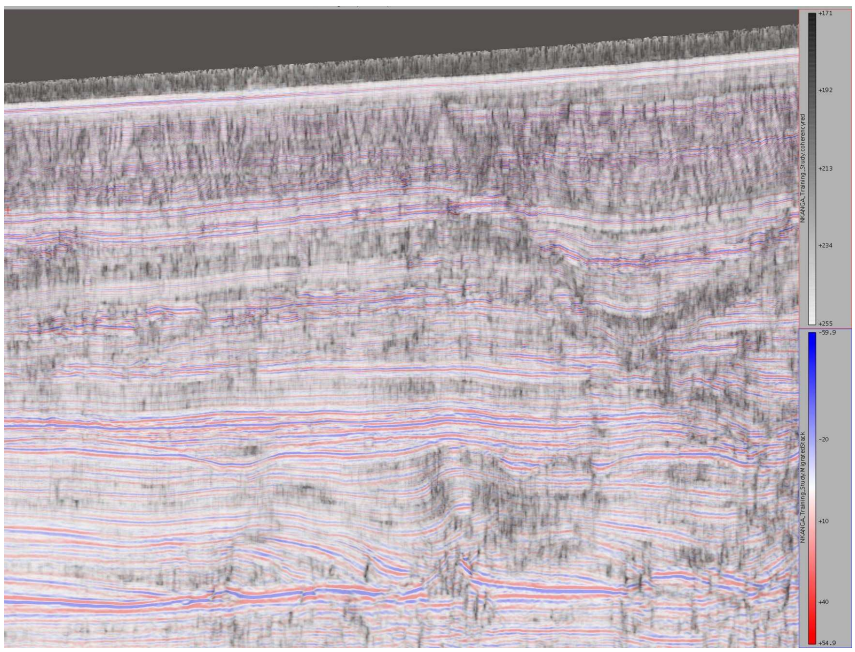


FIGURE 5.3 – Sections superposées d'un cube sismique (palette couleurs) et du cube cohérence correspondant (niveaux de gris).

Soient N_1, N_2 et N_3 les dimensions du cube respectivement dans la direction d'*inline*, de *crossline*, et de temps ou de profondeur. Soient w la taille de la fenêtre verticale sur une trace et $shift$ la taille de la fenêtre du voisinage vertical ; le calcul séquentiel de l'attribut cohérence s'effectue alors comme décrit dans l'algorithme 3 où la corrélation est calculée selon l'équation

(5.1) et la sortie (ligne 13) représente le cumul des corrélations maximales calculées.

ALGORITHME 3 : Algorithme séquentiel pour le calcul de l'attribut cohérence

```
   /* Pour tous les éléments dans le cube                                    */
 1 pour i = 1 à N1 − 1 faire
 2     pour j = 1 à N2 − 1 faire
 3         pour k = w + shift à N3 − w − shift faire
 4             data[1 .. 2*w+1] ← input[i-w .. i+w][j][k]
               /* pour tous les points du voisinage                          */
 5             pour ii = −1 à 1 faire
 6                 pour jj = −1 à 1 faire
 7                     max_correlation ← -1.0
                       /* pour le décalage sur les traces du voisinage       */
 8                     pour kk = k − shift à k + shift faire
 9                         voisin[1 .. 2*w+1] ← input[kk-w ..kk+w][j+jj][i+ii]
                           /* calcul de la corrélation selon l'équation 5.1  */
10                         correlation ← correler(data, voisin)
11                         si correlation ≥ max_correlation alors
12                             max_correlation ← correlation
13                     output[i][j][k] +← max_correlation
```

De façon similaire à l'algorithme de modélisation, pour un cube sismique 3D, nous organisons les *threads* du GPU selon une grille 2D de blocs 2D. Le nombre de *threads* dépend de la taille du cube dans les directions *inline* et *crossline*. Le cube sismique est traité coupe par coupe selon la direction z. Chaque *thread* traitera ainsi une trace sismique, ce qui correspond à la boucle sur i de l'algorithme 3. De façon similaire à l'algorithme de modélisation également, nous exploitons dans notre implémentation les registres locaux des *stream processors* pour stocker les valeurs courantes d'une fenêtre glissante de taille w le long des traces sismiques. Les variables scalaires déclarées de type primitif (`int`, `float`, ...) sont généralement allouées dans ces registres. Un multiprocesseur a 64000 registres 32 bits sur les GPU Fermi. Si le nombre de registres nécessaire dépasse celui disponible sur un multiprocesseur, une partie des variables sera allouée en mémoire locale avec un temps d'accès beaucoup plus important, ce qui fera chuter les performances pour de grandes valeurs de w. Ce phénomène est appelé *débordement de registre* (ou *register spilling*). Nous en verrons une illustration plus détaillée dans le cas du lissage (cf. 5.1.3).

Gradient de cohérence

Nous proposons l'amélioration du niveau de détail perçu sur le cube cohérence en appliquant localement (au niveau de chaque trace) un filtre gradient [89]. Il s'agit pour tout point du cube sismique de calculer la convolution des corrélations maximales obtenues sur un voisinage latéral

de taille ws avec un filtre gradient tel qu'illustré sur la figure 5.4. Dans le cube GCC (pour *Gradient Coherency Cube*) ainsi obtenu, les discontinuités apparaissent clairement sous la forme de passages par zéro.

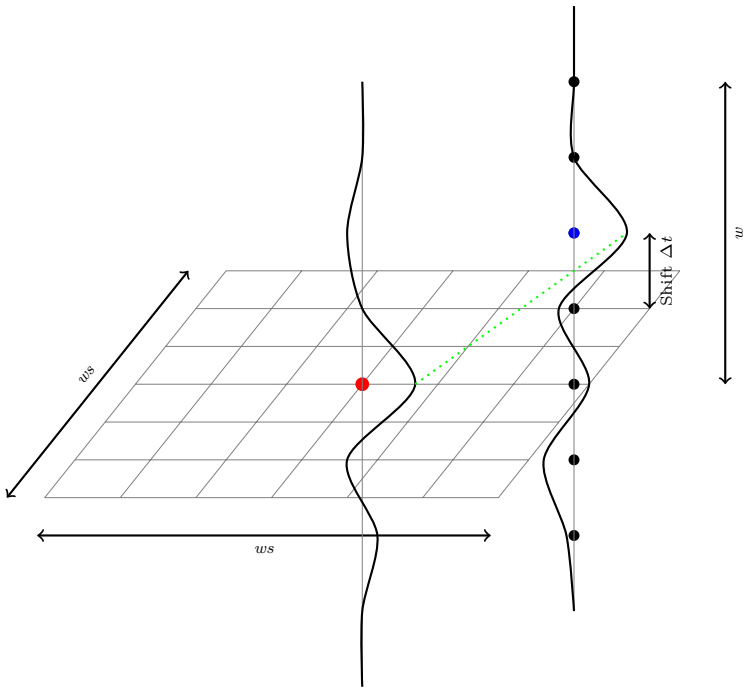


FIGURE 5.4 – Principe du calcul du cube gradient de cohérence (GCC) : à chaque point du cube sismique, un gradient est calculé à partir des corrélations maximales entre une fenêtre autour du point centrale et des fenêtres décalées sur les traces de son voisinage.

La figure 5.11 permet de comparer les attributs cohérence et GCC. Cette figure montre que l'application du gradient dans le cas du GCC permet de ressortir des détails non perceptibles sur le cube cohérence, en particulier ceux dont la direction est normale à celle du filtre.

Attributs géométriques : pendage et azimut

La géométrie d'orientation d'un horizon ou réflecteur plan passant par un point $\mathbf{x}(x, y, z)$ peut être définie de diverses façons équivalentes comme illustré à la figure 5.5 extraite de [88] dont nous reprenons les définitions et notations :

- un vecteur $\mathbf{n}(n_x, n_y, n_z)$ normal à ce plan ;
- sa direction par rapport à une direction de référence (usuellement celle du Nord) notée ψ et son pendage ou inclinaison noté θ de la ligne de plus grande pente par rapport au plan horizontal (ou les projection θ_x et θ_y de cet angle respectivement sur les plans xz et yz) ;
- son pendage θ et son azimut ϕ qui correspond à l'angle entre une direction de référence et la direction du plus fort pendage.

C'est cette dernière description qui est usuellement utilisée en imagerie sismique.

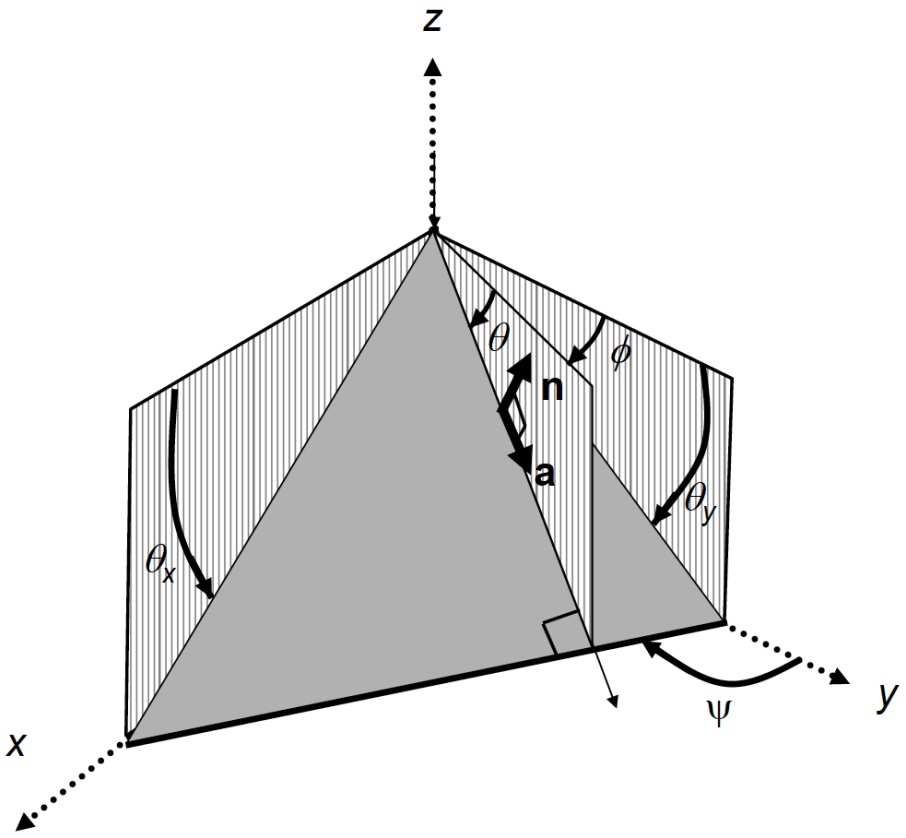


FIGURE 5.5 – (Différents éléments permettants de caractériser un réflecteur plan (extrait de [88]).

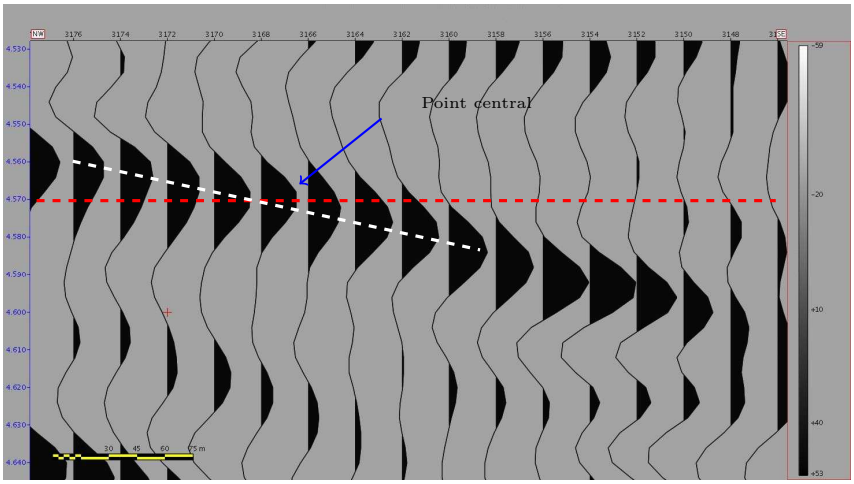


FIGURE 5.6 – Calcul des pendages à partir du plan défini par la corrélation maximale autour de chaque point. Les pointillés blancs croisent les traces sur des points ayant la corrélation maximale avec le point central.

Cette information peut avoir plusieurs utilités. Lors de l'interprétation, l'orientation permet de détecter rapidement les anomalies dans les données, ce qui peut servir à contrôler la qualité des données en entrée. Elle aide également à reconnaître les failles, à distinguer les horizons et les associer sur des plans de coupe différents et à suivre un horizon des deux côtés d'un plan de faille. Ces attributs peuvent également servir de base à des calculs plus avancés. Le *Dip Steering* consiste ainsi à prendre en compte en chaque point de l'image l'orientation des évènements pour le calcul d'un attribut sismique donné au lieu de le calculer sur un voisinage horizontal.

Les approches existantes pour le calcul des pendages reposent essentiellement sur le calcul d'un champ de gradient à partir des amplitudes des données sismiques ou de leur phase instantanée, ou de façon plus robuste sur le tenseur de structure qui en découle.

Nous proposons une approche différente qui permet de calculer l'orientation à partir du plan "le plus proche" des points du voisinage maximisant la corrélation.

Pour calculer le pendage et l'azimut d'un point dans un cube sismique, nous suivons ainsi les étapes suivantes :

- trouver les coordonnées des points ayant la corrélation maximale dans les traces du voisinage dans une fenêtre d'analyse. Le nombre des traces du voisinage et la taille de la fenêtre d'analyse sont paramétrés. Le calcul de la corrélation s'exécute comme celui du calcul des attributs cohérence et gradient de corrélation ;
- nous cherchons ensuite un plan qui minimise la somme des distances du plan à tous les points trouvés dans l'étape précédente ;
- supposons que le plan calculé soit modélisé par la fonction $f(x, y) = ax + by + c$, les coefficients a et b correspondent respectivement au pendage et à l'azimut au point d'intérêt.

La figure 5.6 illustre le calcul du pendage pour une série de traces sismiques voisines.

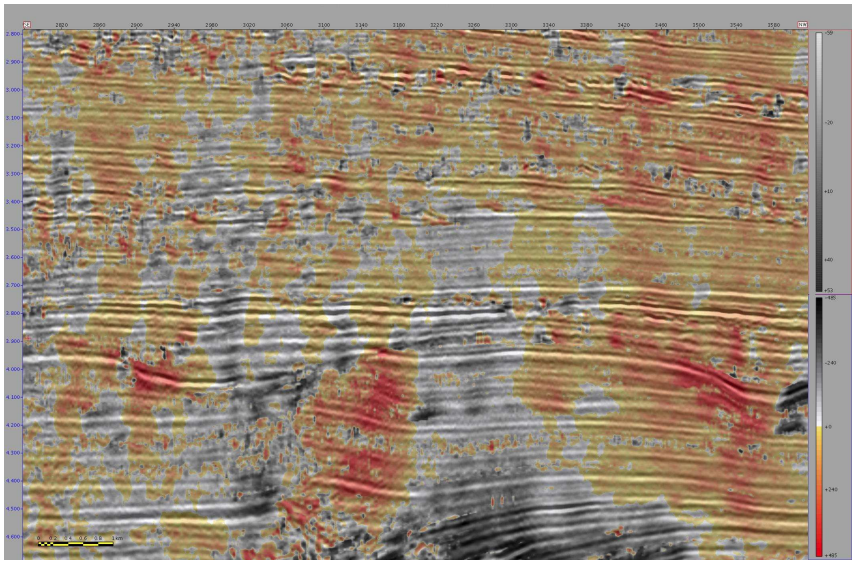


FIGURE 5.7 – Superposition section sismique et attribut pendage

Pour un plan $f(x, y) = ax + by + c$ avec $a, b, c \in \mathbb{R}$, la régression linéaire cherche à minimiser la fonction S définie par

$$S(a, b, c) = \sum_{i=1}^{n} (ax_i + by_i + c - z_i)^2.$$

S est minimale lorsque

$$\begin{cases} \frac{\partial S}{\partial a}(a, b, c) &= 0 \\ \frac{\partial S}{\partial b}(a, b, c) &= 0 \\ \frac{\partial S}{\partial c}(a, b, c) &= 0 \end{cases}$$

ce qui revient à résoudre le système suivant

$$\begin{cases} \sum_{i=1}^{n} 2x_i(ax_i + by_i + c - z_i) &= 0 \\ \sum_{i=1}^{n} 2y_i(ax_i + by_i + c - z_i) &= 0 \\ \sum_{i=1}^{n} 2(ax_i + by_i + c - z_i) &= 0 \end{cases}$$

qui peut se réécrire sous la forme matricielle $Ax = b$

$$\begin{pmatrix} \sum_{i=1}^{n} x_i^2 & \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} x_i \\ \sum_{i=1}^{n} x_i y_i & \sum_{i=1}^{n} y_i^2 & \sum_{i=1}^{n} y_i \\ \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} y_i & \sum_{i=1}^{n} 1 \end{pmatrix} \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{n} x_i z_i \\ \sum_{i=1}^{n} y_i z_i \\ \sum_{i=1}^{n} z_i \end{pmatrix}$$

Les corrélations maximales calculées sont directement accumulées dans les matrices A et le vecteur b pour éviter une utilisation inutile des ressources mémoire. Neuf registres de 4 octets suffisent pour sauver les données nécessaires au calcul du pendage : 3 pour le vecteur colonne b et 6 autres pour la matrice A qui est symétrique. Il suffit ensuite de résoudre ce système pour trouver les pendages recherchés. La figure 5.7 représente une superposition d'une section sismique est du pendage calculé sur cette sections.

5.1.3 Lissage 3D par filtre Gaussien

Le lissage est une opération importante en traitement d'images. Il permet d'appliquer un flou à une image ou d'en atténuer le bruit. Il est souvent utilisé en préalable à un autre traitement tel que la détection de bord en vision artificielle. Il s'agit dans ce cas de filtres 2D.

En imagerie sismique, le lissage peut s'appliquer en 2D ou 3D aux modèles utilisés pour la migration (vitesse, densité, etc.) pour adapter ces modèles aux algorithmes de migration. Pour l'interprétation, les données sismiques ainsi que les attributs calculés à partir de ces données peuvent également être lissés [61] pour en atténuer les bruits et faciliter leur traitement que ce soit par l'interprétateur ou par des algorithmes de traitement automatique (suivi d'horizons, de failles, etc.). Le lissage peut dans ce contexte également servir de préalable à d'autres traitements tels que le calcul de gradient ou la détection de formes. Le lissage peut être orienté de façon à conserver les informations liées à la structure du sous-sol [8, 12].

Différentes méthodes de lissage existent. Nous proposons d'étudier l'accélération par GPU du lissage par filtre gaussien. Nous reprenons brièvement les définitions et notations introduites en [60]. Le lecteur est invité à revenir à cette référence pour une description complète. Il s'agit de convoluer les données en entrée par une fonction gaussienne (loi normale centrée) s'écrivant sous la forme $g(t,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-t^2/2\sigma^2}$ où σ, écart type de la loi normale, correspond approximativement à la demi largeur à mi-hauteur de la gaussienne. La figure 5.8 représente $g(t, \sigma = 1)$.

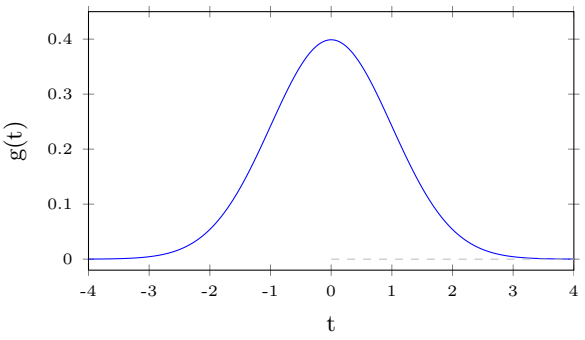


FIGURE 5.8 – Fonction gaussienne $g(t, \sigma = 1)$.

La forme de la fonction gaussienne suggère que la convolution peut être restreinte à un support fini de cette fonction. Usuellement, on se restreint à l'intervalle de $|t| \leq M\sigma$ avec $M = 4$. Si on fait le choix de réduire le temps de calcul aux dépends du résultat du lissage, on peut restreindre le filtre à un support de taille $2 \times 3\sigma + 1$ ($M = 3$). La séquence lissée $y[i]$ est

obtenue à partir de la séquence $x[i]$ en entrée par la convolution suivante :

$$y[i] = \sum_{k=i-M\sigma}^{i+M\sigma} g[i-k]x[k]. \tag{5.3}$$

Cette approximation constitue un filtre à réponse impulsionnelle finie ou filtre RIF (*Finite Impulse Response filter* ou *FIR filter*). Le passage en 2D et 3D peut s'effectuer en convoluant par une matrice 2D ou 3D ou, en exploitant la séparabilité du filtre gaussien, en convoluant respectivement sur les 2 ou 3 dimensions de façon itérative.

C'est cette approche que nous adoptons pour notre implémentation GPU du filtre RIF de lissage gaussien en 3D. Nous exploitons pour cela le noyau de transposition introduit précédemment pour garantir l'alignement des accès en mémoire pour les 3 dimensions. Nous utilisons également le mécanisme de fenêtre glissante le long de chacune des 3 directions pour minimiser le nombre des accès en mémoire partagée, un *thread* traitant une colonne du cube d'entrée.

Ce mécanisme s'appuie sur le chargement des données nécessaires au calcul courant dans des registres. La taille de ces données dépend cependant de σ. On utilise une fenêtre glissante de la taille du support du filtre à savoir $2 \times M\sigma + 1$. En augmentant σ, on utilise de plus en plus de registres ce qui décroit l'occupation et entraîne l'utilisation de la mémoire locale (*register spilling*) ce qui induit un surcoût en temps de calcul. Ce surcoût est évalué à titre d'exemple à 6% pour $\sigma = 6$ et 140% pour $\sigma = 12$ (lorsque $M = 4$). On reporte dans le tableau 5.3 les temps d'exécution en milliseconde et le nombre de registres utilisées pour le filtre RIF de lissage gaussien pour des cubes en entrée de taille N^3 avec N=256 et 512 en fonction de σ et de M. On note que la saturation des registres (utilisation de 63 registres) se produit à partir de $\sigma = 5$ pour $M = 4$ et à partir de $\sigma = 6$ pour $M = 3$. La complexité du calcul dépend également de σ. Pour un point en entrée, on effectue $M\sigma + 1$ multiplications en utilisant la symétrie du filtre et $2M\sigma$ additions.

Pour s'affranchir de cette contrainte, il est possible d'approximer le filtre gaussien RIF par un filtre récursif à réponse impulsionnelle infinie ou filtre RII (*infinite impulse response* ou *IIR*). Deriche [45] et Vliet et al. [135, 140] proposent deux approches différentes pour cette approximation. Il s'agit de remplacer (5.3) par une formulation récursive faisant intervenir l'élément courant en entrée et un nombre constant d'éléments de son voisinage indépendant de σ. Dans les deux cas le filtre est composé d'une partie causale faisant intervenir des éléments $x[k]$ avec $k \leq i$ et anti-causale fonction de $x[k]$ avec $k \geq i$ pour le calcul de $y[i]$. Alors que Dériche applique ces deux filtres sur la séquence d'entrée x[i], Vliet et al. calculent une séquence intermédiaire avec la partie causale sur laquelle est appliquée la partie anti-causale. La démarche et le résultat sont décrits sur la figure 5.9.

Pour les mêmes données de tailles 256^3 et 512^3 utilisées dans le tableau 5.3 le filtre RII s'exécute respectivement en $42ms$ et $328ms$ indépendamment de σ et M. Pour de petites valeurs de σ ($\sigma \leq 6$ pour $M = 3$ et $\sigma \leq 4$ pour $M = 4$) il est ainsi plus intéressant d'utiliser le lissage gaussien par filtre RIF. Dès que la taille du filtre ne permet plus d'utiliser les registres pour les

M	3			4		
σ	$T_{N=256}$ (ms)	$T_{N=512}$ (ms)	Nb Registres	$T_{N=256}$ (ms)	$T_{N=512}$ (ms)	Nb Registres
1	34.81	279.08	25	35.75	281.56	27
2	36.71	289.75	33	38.02	293.62	39
3	37.16	295.36	43	38.62	299.54	51
4	39.28	299.71	51	39.27	307.20	62
5	38.96	305.65	61	44.52	350.50	63
6	40.07	312.39	63	66.11	539.24	63
7	50.96	405.02	63	109.98	940.08	63
8	65.93	541.77	63	141.54	1230.81	63
9	99.18	836.73	63	172.64	1554.44	63
10	126.60	1097.74	63	200.65	1852.24	63
11	144.92	1274.11	63	221.95	2097.39	63
12	173.03	1556.74	63	239.05	2326.34	63
13	194.47	1770.59	63	251.59	2510.09	63
14	210.54	1968.71	63	266.04	2721.17	63
15	226.72	2162.65	63	275.88	2936.01	63
16	239.73	2325.90	63	284.23	3126.97	63
17	254.49	2511.90	63	289.30	3311.06	63
18	260.24	2628.29	63	290.90	3469.54	63
19	265.86	2754.87	63	291.29	3638.40	63
20	276.32	2935.25	63	287.35	3767.82	63

TABLE 5.3 – Temps d'exécution en milliseconde et nombre de registres utilisés pour le filtre RIF de lissage gaussien pour des cubes en entrée de taille N^3 avec N=256 et 512 en fonction de σ et de M. Pour ces tailles, le filtre RII s'exécute respectivement en $42ms$ et $328ms$ indépendamment de σ et M.

données courantes, l'utilisation du filtre RII devient sensiblement plus intéressante.

5.1.4 Résultats

L'échelle à laquelle se fait usuellement le calcul des attributs sismiques est significativement plus réduite en comparaison avec celle de l'imagerie sismique. Ces calculs sont souvent exécutés en effet sur de puissantes stations de travail ou sur un nombre restreint de nœuds de calcul. A la différence de l'imagerie sismique où des centaines de milliers de points de tir peuvent être traités en parallèle, l'interprétateur manipule habituellement un nombre réduit de cubes sismiques sur lesquels il effectue des traitements successifs. Les temps de calcul varient usuellement de quelques minutes à quelques heures pour ces traitements alors qu'une migration RTM avant sommation peut nécessiter plusieurs semaines de calcul.

Les résultats reportés dans cette section ont été obtenus sur une station de travail Dell T7500 équipée de deux cartes graphiques NVIDIA d'architecture Fermi : une Quadro 6000 et

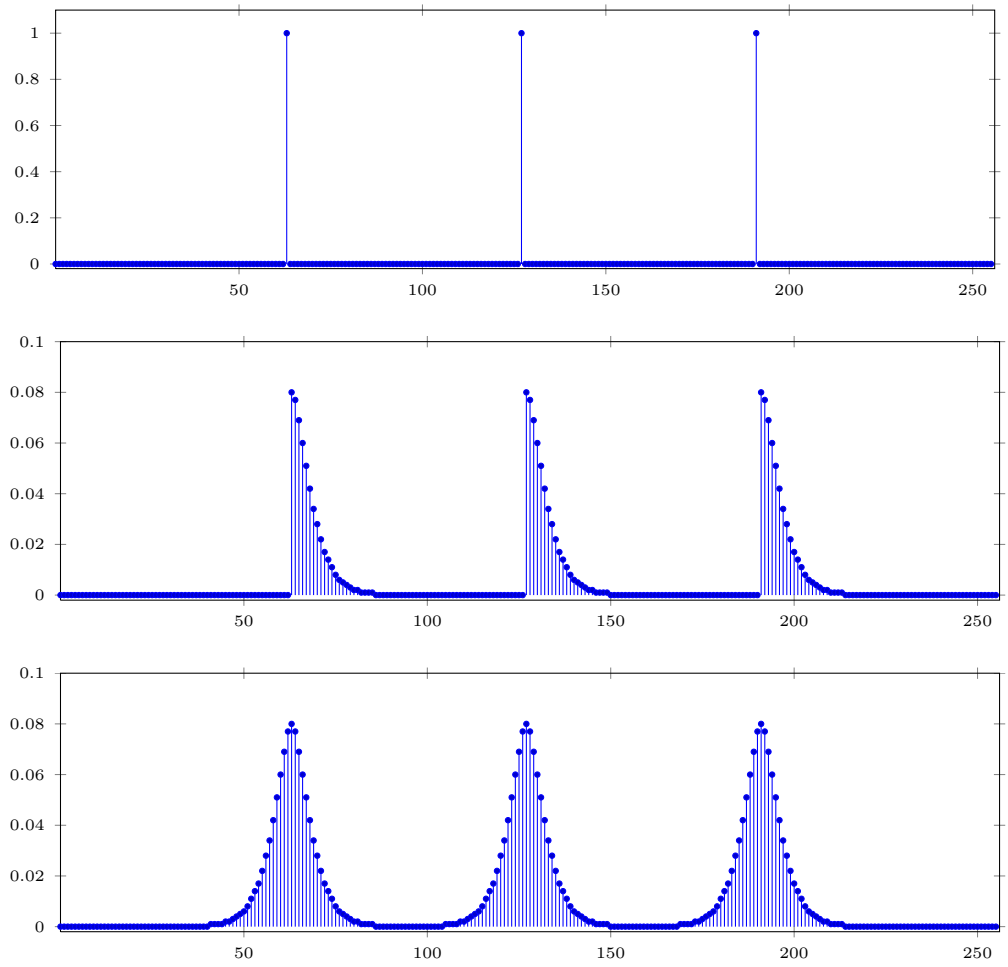


FIGURE 5.9 – Séquence en entrée (haut). Application de la partie causale du filtre (milieu). Application de la partie anti-causale du filtre sur les données intermédiaires (bas).

une C2075 disposant chacune de 6 Go de RAM et de 448 cœurs. Cuda 5 version de driver 302.06.03. Les performances des diverses implémentations GPU sont comparées à celles obtenues sur le processeur Intel Westmere (2 *sockets*× 6 cores) cadencé à 2.94 GHz. Les résultats sont obtenus sur un cube de flottants en simple précision de taille 2Go ($1024 \times 1024 \times 512$) en entrée. Pour les attributs basés sur la similarité (cohérence, GCC, pendage), nous avons développé une implémentation CPU de référence parallèle MPI+OpenMP. Pour notre architecture, nous utilisons deux processus MPI avec 6 threads OpenMP chacun. L'implémentation CPU du lissage est quant à elle séquentielle et repose sur l'approche récursive décrite précédemment. Les paramètres utilisés sont ceux qui sont usuellement utilisés par les interprétateurs à savoir :

- **cohérence** $w = 10$ et $shift = 5$;
- **GCC** $ws = 7$, $w = 3$, shift=5 ;
- **pendage** $ws = 7$, $w = 3$, shift=5 ;
- **lissage** $\sigma = 2$, M=3 ;

Dans la figure 5.10 sont reportés les résultats comparatifs détaillés des performances des implémentations CPU et GPU du calcul d'un cube GCC pour différentes taille. Ces résultats donnent la mesure de l'impact que peut avoir l'utilisation des GPU dans le processus d'interprétation classique. D'un attribut difficilement exploitable au quotidien, on passe avec l'accélération graphique à un calcul de quelques minutes utilisable par tous les interprétateurs.

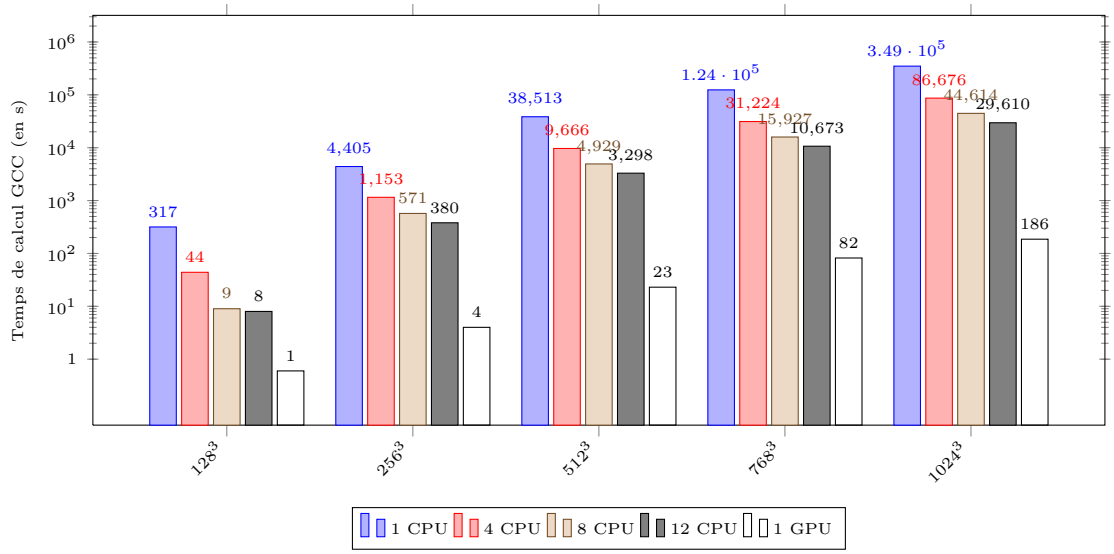


FIGURE 5.10 – Temps de calcul de l'attribut GCC en fonction de la taille du cube sismique en entrée pour les implémentations CPU et GPU.

Attribut	T_{CPU}	T_{GPU}	*Speedup*
Cohérence	689.24	13.8	50×
GCC	3240.71	28.5	160.7×
Pendage	295.36	37.16	7.9×
Lissage	27.8	1.2	23×

TABLE 5.4 – Temps d'exécution en secondes et facteurs d'accélération des implémentations CPU et GPU de calculs d'attributs sismiques

5.2 Vers l'interprétation interactive sur GPU

La réduction des temps de calcul pour les attributs sismiques permet de paver la voie à l'interprétation interactive. On peut parler d'interactivité dès que les temps de calcul descendent sous la seconde. Idéalement, des calculs permettant d'afficher trois images par seconde seront quasiment imperceptibles pour l'utilisateur. Nous avons montré que le calcul sur GPU permet pour des cubes de tailles usuelles de descendre sous la seconde de calcul. Nous pouvons réduire

encore plus ce temps en ne traitant que les données nécessaires à l'affichage du sous domaine visualisé : il peut s'agir d'une section en temps, d'une coupe en *inline* ou en *cross-line*, ou de données extraites le long d'un horizon, des faces d'un sous cube, etc. Le calcul s'effectuant sur GPU, ces données pourront être mises à disposition des bibliothèques effectuant le rendu sans surcoût de transferts CPU-GPU grâce aux mécanismes d'interopérabilité offerts par les outils de calcul (type CUDA) et de rendu (type OpenGL). Au cours de nos travaux, nous avons pu développer un prototype pour le calcul GCC et de la cohérence en interactif représenté sur la figure 5.11. Une telle solution permet de déterminer rapidement les paramètres adéquats à un certain calcul avant de l'exécuter sur l'ensemble du cube en entrée. Cette approche permet également d'ouvrir la voie à un changement radical dans l'exercice d'interprétation. En effet, si le temps de calcul est suffisamment rapide par rapport au temps d'accès aux données, il suffira de recalculer les données à afficher à la volée, selon la partie que l'on souhaite visualiser. Le calcul à la volée induit par la visualisation permet d'éviter le stockage des données calculées.

5.3 Conclusion

Nous avons décrit dans ce chapitre comment l'utilisation de l'accélération GPU pour le calcul d'attributs sismiques permettait de modifier en profondeur le travail des interprétateurs en accélérant considérablement certaines tâches de leur quotidien et en tendant vers le calcul interactif. Cependant, l'utilisation intensive de la carte graphique de la station de travail pour le calcul généraliste souffre encore de certaines limitations. Nous avons pu disposer pour nos travaux d'une station de travail équipée d'un accélérateur Tesla en plus de la carte graphique. Ceci est rarement le cas. L'utilisation de la carte graphique principale entraîne des ralentissements et une détérioration des performances du système[118]. Il existe également une limite de temps imposée par le système d'exploitation pour l'exécution des noyaux sur GPU (environ 5s). Au-delà de cette limite, le calcul est considéré comme suspendu et le système met fin à son exécution pour éviter la détérioration des performances de la station. Il convient donc de mettre en place à chaque fois les solutions algorithmiques adéquates. La mise en œuvre pratique dans le cadre de la plateforme industrielle ainsi que l'implémentation d'autres types d'attributs ont fait l'objet d'un sujet et d'un mémoire de stage[98]. Le lecteur est invité à s'y reporter.

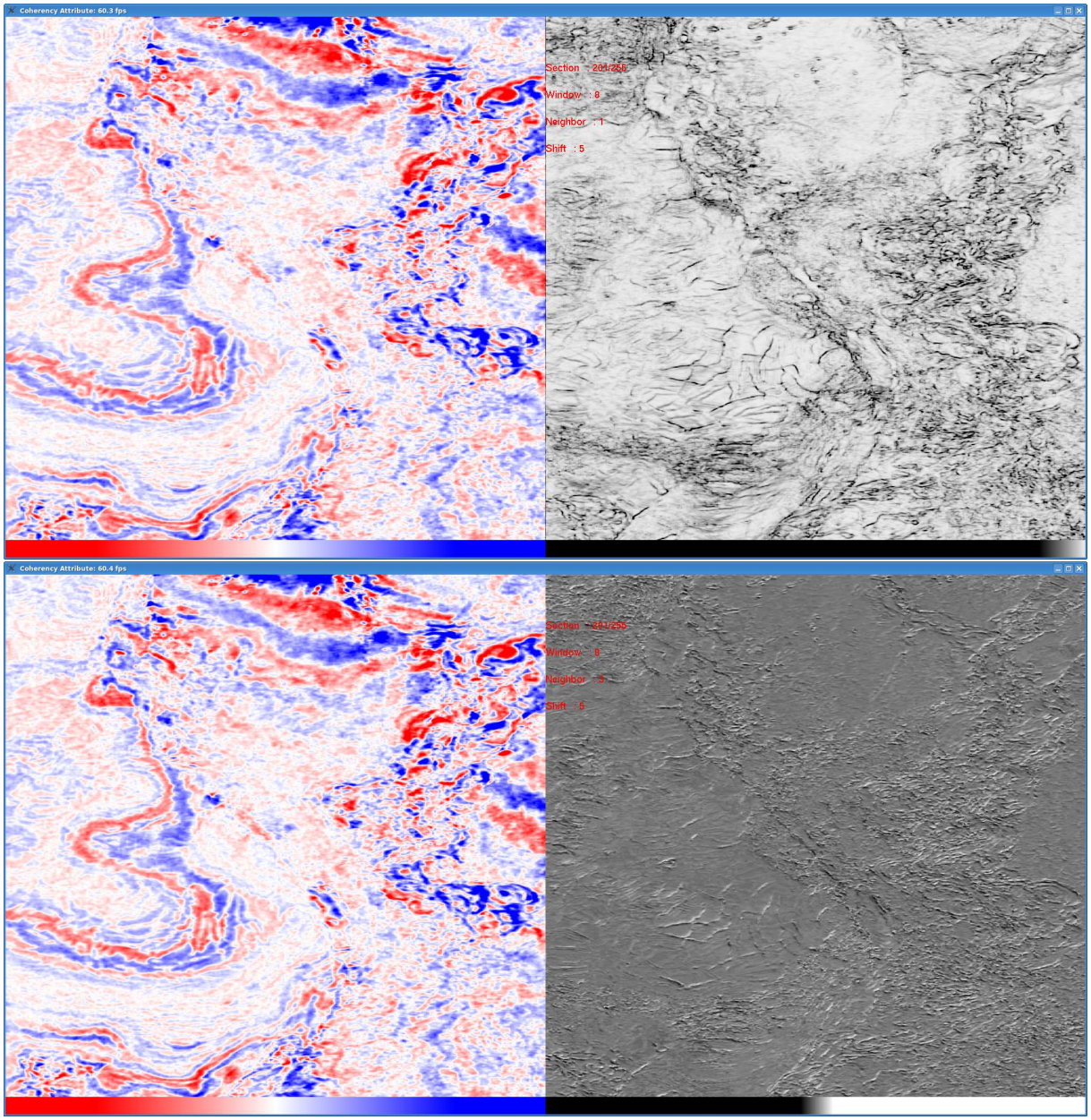


FIGURE 5.11 – Calcul interactif de la cohérence et du GBC (respectivement en haut et en bas à droite, en niveaux de gris) sur une section en temps des données sismiques (à gauche). Seules les données nécessaires au calcul de la section courante sont transférées vers le GPU.

Chapitre 6

Conclusions et perspectives

6.1 Conclusions

Dans ce travail de thèse, nous avons pu démontrer l'apport que pouvaient constituer les accélérateurs de calcul tout au long de la chaîne d'imagerie et d'interprétation sismiques. Des applications mises en œuvre à chaque étape de cette chaîne (modélisation, migration et interprétation sismiques) ont été portées et des facteurs d'accélération importants ont été reportés par rapport à l'utilisation de processeurs généralistes. Pour la modélisation sismique en densité constante un facteur 40 a été obtenu. Pour la migration RTM, un facteur plus modeste a été observé compte tenue de la taille des transferts des données. Enfin, pour certains calculs d'attributs sismiques utilisés pour l'interprétation, des accélérations allant de 8 à 160 ont été obtenues.

Cette étude a été menée selon deux points de vues différents et à deux échelles distinctes. En effet, pour l'imagerie sismique (modélisation et migration), les accélérateurs se posent comme concurrents des processeurs généralistes. Les contraintes d'espace et d'alimentation électrique des centres de calcul imposent de sélectionner les architectures les plus performantes et les plus adaptées aux applications considérées en minimisant, entre autres, l'occupation au sol et la consommation électrique. Nous avons pu démontrer en considérant toutes ces contraintes que la solution GPU était la plus indiquée pour ce type d'applications. La situation est toute autre pour l'interprétation des données sismiques. Les stations d'interprétation disposent de cartes graphiques puissantes utilisées pour le rendu graphique. L'exploitation des ressources de ces cartes se présente alors comme une possibilité complémentaire à l'utilisation du processeur généraliste. Nous avons démontré que l'utilisation de ces ressources permettait de tendre vers l'interactivité recherchée pour l'exercice d'interprétation. Nous avons également proposé une solution permettant d'atteindre cette interactivité en recalculant à la volée uniquement les parties à afficher d'un cube attribut en lieu et place du calcul du cube entier en amont, de son stockage sur disque, et ce pour pouvoir ensuite en afficher certaines parties. L'interopérabilité entre bibliothèques de calcul et d'affichage sur GPU, illustrée dans le cadre de cette thèse,

permet de rendre cette démarche efficace en maintenant les données à calculer et à afficher résidentes dans la mémoire de l'accélérateur graphique. Cette approche a également le mérite de réduire le volume de données généré.

Dans ce travail de thèse, nous avons également souligné les facteurs limitants à considérer pour obtenir des implémentations performantes sur ces plateformes accélératrices. Les mécanismes algorithmiques permettant de réduire l'impact de ces facteurs ont été explorés et illustrés par différents exemples. Etant donnée la puissance de calcul importante disponible sur les accélérateurs, les contraintes sont souvent essentiellement liées aux coûts d'accès et de transfert des données. Ces coûts peuvent être limités par le biais de l'exploitation des différents niveaux de mémoire des accélérateurs, de la localité des calculs, du recouvrement entre calcul et communication, des motifs d'accès à la mémoire (adaptation de la taille des vecteurs, *prefetch*, réduction des conflits d'accès et du *patition camping*, etc) notamment.

Nous avons enfin démontré que, de la même manière qu'il convient de rechercher l'architecture la plus adaptée à une application particulière, il convient aussi de prendre en compte les contraintes propres à chaque architecture et d'adapter les méthodes numériques considérées à ces contraintes. Dans ce contexte, l'utilisation de schémas numériques d'ordres plus élevés pour les différences finies et de méthodes pseudo-spectrales pour la résolution de l'équation d'onde sur GPU et FPGA a été explorée. La comparaison des filtres FIR et IIR pour le lissage Gaussien sur GPU a été également menée. Dans les deux cas on relève la pertinence de la démarche.

La fiabilité, la maintenabilité, la portabilité ainsi que la pérennité des codes fournis ont été des objectifs importants tout au long de cette thèse. Dans cette optique, l'utilisation de la programmation par directives de compilation avec HMPP et OpenAcc a été identifiée comme l'approche la plus appropriée pour garantir ces objectifs.

Ces travaux ouvrent la voie à une intégration complète et systématique des accélérateurs de calcul tout le long du cycle de traitement des données sismiques et ce d'autant plus que nous avons démontré que cette intégration ne se fait pas aux dépends de la fiabilité et de la maintenabilité du code existant.

6.2 Perspectives

Le travail réalisé dans le cadre de cette thèse peut être prolongé en concrétisant certaines des idées proposées. Ainsi, certaines optimisations suggérées notamment dans la section 4.3 n'ont pas pu être mises en œuvre dans le cadre de l'application en production. Leur mise en œuvre nécessite des changements importants dans l'architecture de l'application mais permettrait d'améliorer de façon notable sa scalabilité. L'étude menée dans la section 4.6.1 sur l'impact de l'ordre du

schéma numérique sur le nombre d'accès à la mémoire mériterait d'être étendue (par la prise en compte notamment de la décomposition du domaine et des transferts entre sous-domaines) et validée expérimentalement. L'utilisation de la méthode hybride pseudo-spectrale et différences finies en 3D pour résoudre l'équation d'onde nous semble également très prometteuse pour les architectures actuelles. Au delà de ces architectures, l'évolution des accélérateurs (FPGA, APU avec un accès cache cohérent GPU-CPU à 128 Go de mémoire partagée, GPU discrets avec 12 Go de mémoire) peut permettre à court terme d'envisager l'utilisation de la méthode pseudo-spectrale seule en 3D. Pour le calcul interactif d'attributs sismiques, un prototype a été fourni. Ce prototype devra être industrialisé et intégré à la station d'interprétation. Cette étude pourrait enfin avoir comme prolongement naturel l'utilisation de machines hybrides à mémoire distribuée, similaires à celles utilisées pour l'imagerie sismique, pour le calcul d'attributs sismiques et le rendu graphique distribués pour de gros volumes de données.

A moyen terme, l'analyse entreprise dans le cadre de cette thèse et les résultats obtenus permettent de paver la voie à l'évolution des applications et leur adaptation aux architectures exaflopiques. En effet, le placement des données, les motifs d'accès à la mémoire ainsi que l'asynchronisme entre calculs et communications font partie des éléments importants à considérer pour avoir un passage efficace à l'exascale.

Enfin, dans le cadre de l'industrie pétrolière, l'utilisation d'accélérateurs matériels peut s'étendre au delà de l'imagerie et de l'interprétation sismiques, pour la simulation et l'étude géostatistique des réservoirs, la modélisation numérique des propriétés pétrophysiques des roches (DRP pour *Digital Rock Physics*) ou encore l'étude du placement optimal des puits de production.

Bibliographie

[1] MAGMA : Matrix Algebra on GPU and Multicore Architectures. icl.cs.utk.edu/magma. 50

[2] OpenFPGA Inc. http://www.openfpga.org/. 54

[3] R. Abdelkhalek : Convey FPGA platform evaluation for seismic modeling and migration. Rapport technique, Total, 2009. 97

[4] R. Abdelkhalek : Utilisation des GPU pour le calcul d'attributs sismiques : adaptation aux GPU Fermi. Rapport technique, Total, 2012. 107

[5] Ashwin M. Aji, Mayank Daga et Wu-chun Feng : Bounding the effect of partition camping in GPU kernels. *In Proceedings of the 8th ACM International Conference on Computing Frontiers*, CF'11, pages 27 :1–27 :10, New York, NY, USA, 2011. ACM. 107

[6] K. Aki et P. Richards : *Quantitative Seismology : Second Edition*. University Science Books, Sausalito, 2002. 18

[7] U. Albertin, G. Brown, F. Dewey, J. Farnsworth, G. Grubitz et M. Kemme : The time for depth imaging. *Oilfield Review*, 14(1):2–15, 2002. 14

[8] Nasher M. AlBinHassan, Yi Luo et Mohammed N. Al-Faraj : 3D edge-preserving smoothing and applications. *Geophysics*, 71(4):P5–P11, 2006. 116

[9] H. Ben Hadj Ali : *Three dimensional visco-acoustic frequency-domain full waveform inversion*. Thèse de doctorat, Université Nice-Sophia-Antipolis, 2009. 18

[10] AMD : OpenCL Fast Fourier Transforms. http://clmathlibraries.github.io/clFFT. 94

[11] H. Anzt, T. Hahn, V. Heuveline et R.Örn : GPU Accelerated Scientific Computing : Evaluation of the NVIDIA Fermi Architecture ; Elementary Kernels and Linear Solvers. EMCL Preprint Series, 2010. 42

[12] Ahmed Adnan Aqrawi et Trond Hellem Boe : *Improved fault segmentation using a dip guided and modified 3D Sobel filter*, chapitre 195, pages 999–1003. 116

[13] Cédric Augonnet, Samuel Thibault, Raymond Namyst et Pierre-André Wacrenier : StarPU : A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency and Computation : Practice and Experience, Special Issue : Euro-Par 2009*, 23:187–198, février 2011. 51

[14] C. Baldassari : Implémentation de la Reverse Time Migration dans MigWE. Rapport technique, Total, 2006. 24, 26

[15] E. Baysal, D. Kosloff et J. W. C. Sherwood : Reverse Time Migration. *Geophysics*, 48(11):1514–1524, 1983. 13, 29

[16] J-P. Berenger : A perfectly matched layer for the absorption of electromagnetic waves. *J. Comput. Phys.*, 114:185–200, October 1994. 28

[17] J-P. Berenger : Three-dimensional perfectly matched layer for the absorption of electromagnetic waves. *Journal of Computational Physics*, 127(2):363 – 379, 1996. 28

[18] B. L. Biondi : *3D Seismic Imaging*. SEG, 2006. 13

[19] T. N. Bishop, K. P. Bube, R. T. Cutler, R. T. Langan, P. L. Love, J. R. Resnick, R. T. Shuey, D. A. Spindler et H. W. Wyld : Tomographic determination of velocity and depth in laterally varying media. *Geophysics*, 50(6):903–923, 1985. 12

[20] T. Bohlen et E. H. Saenger : Accuracy of heterogeneous staggered-grid finite-difference modeling of Rayleigh waves. *Geophysics*, 71(4):T109–T115, 2006. 25, 26

[21] M. Bromley, S. Heller, T. McNerney et Guy L. Steele, Jr. : Fortran at ten gigaflops : the connection machine convolution compiler. *In Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation*, PLDI '91, pages 145–156, New York, NY, USA, 1991. ACM. 62

[22] R. Brossier : *Imagerie sismique à deux dimensions des milieux visco-élastiques par inversion des formes d'ondes : développements méthodologiques et applications*. These, Université de Nice Sophia-Antipolis, novembre 2009. 25, 27

[23] Alistair R. Brown : Seismic attributes and their classification. *The Leading Edge*, 15(10): 1090, 1996. 15

[24] S. Brown : *Performance comparison of finite-difference modeling on Cell, FPGA, and multicore computers*, chapitre 427, pages 2110–2114. 53

[25] M. Buia, P. E. Flores, D. Hill, E. Palmer, R. Ross, R. Walker, M. Houbiers, M. Thompson, S. Laura, C. Menlikli, N. Moldoveanu et E. Snyder : Shooting seismic surveys in circles. *Oilfield Review*, pages 18–31, 2008. 90

[26] H. Calandra : La puissance de calcul au service de la géophysique. *Technoscoop*, (31):85–89, 2006. 36

[27] P. Calandra, R. Dolbeau, P. Fortin, J.L. Lamotte et I. Said : Evaluation of Successive CPUs/APUs/GPUs Based on an OpenCL Finite Difference Stencil. *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, 0:405–409, 2013. 52

[28] J. M. Carcione, G. C. Herman et A. P. E. ten Kroode : Seismic modeling. *Geophysics*, 67(4):1304–1325, 2002. 11

[29] C. Cerjan, D. Kosloff, R. Kosloff et M. Reshef : A nonreflecting boundary condition for discrete acoustic and elastic wave equations. *Geophysics*, 50(4):705–708, 1985. 28

[30] W. C. Chew et Q. H. Liu : Perfectly matched layers for elastodynamics : A new absorbing boundary condition. *J. Comp. Acoust*, 4:341–359, 1996. 28

[31] W. C. Chew et W. H. Weedon : A 3D perfectly matched medium from modified Maxwell's equations with stretched coordinates. *Microwave and Optical Technology Letters*, 7:599–604, September 1994. 28

[32] Satinder Chopra et Kurt J. Marfurt : Seismic attributes : A historical perspective. *Geophysics*, 70(5):3SO–28SO, 2005. 15

[33] C. Chu et P. L. Stoffa : A pseudospectral-finite difference hybrid approach for large-scale seismic modeling and rtm on parallel computers. *SEG Technical Program Expanded Abstracts*, 27(1):2087–2091, 2008. 27, 97

[34] C. Chu, P. L.. Stoffa et R. Seif : 3D seismic modeling and reverse-time migration with the parallel fourier method using non-blocking collective communications. *SEG Technical Program Expanded Abstracts*, 28(1):2677–2681, 2009. 27

[35] J. F. Claerbout : Coarse grid calculations of waves in inhomogeneous media with application to delineation of complicated seismic structure. *Geophysics*, 35(3):407–418, 1970. 13

[36] J. F. Claerbout : Toward a unified theory of reflector mapping. *Geophysics*, 36(3):467–481, 1971. 29

[37] J. F. Claerbout : *Imaging the Earth's Interior*. Blackwell Scientific Publications, 1985. 13

[38] J. F. Claerbout et S. M. Doherty : Downward continuation of moveout-corrected seismograms. *Geophysics*, 37(5):741–768, 1972. 29

[39] R. Clapp : *Reverse time migration with random boundaries*, chapitre 564, pages 2809–2813. 100

[40] R. Clayton et B. Engquist : Absorbing boundary conditions for acoustic and elastic wave equations. *Bulletin of the Seismological Society of America*, 67(6):1529–1540, 1977. 27

[41] R. Courant, K. Friedrichs et H. Lewy : On the partial difference equations of mathematical physics. *IBM J. Res. Dev.*, 11(2):215–234, 1967. 24

[42] M. A. Dablain : The application of high-order differencing to the scalar wave equation. *Geophysics*, 51(1):54–66, 1986. 20

[43] L. Dagum et R. Menon : OpenMP : an industry standard API for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55, 1998. 38

[44] Kaushik Datta, Mark Murphy, Vasily Volkov, Samuel Williams, Jonathan Carter, Leonid Oliker, David Patterson, John Shalf et Katherine Yelick : Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures. *In SC '08 : Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, pages 1–12, Piscataway, NJ, USA, 2008. IEEE Press. 62, 63

[45] Dr. Deriche, Rachid : Recursively implementating the Gaussian and its derivatives. Research Report RR-1893, INRIA, 1993. 117

[46] J. Diaz et P. Joly : A time domain analysis of PML models in acoustics. *Computer Methods in Applied Mechanics and Engineering*, 195:3820–3853, 2006. 28

[47] H. Dursun, K. Nomura, W. Wang, M. Kunaseth, L. Peng, R. Seymour, R. K. Kalia, A. Nakano et P. Vashishta : In-core optimization of high-order stencil computations. *In* Hamid R. Arabnia, éditeur : *PDPTA*, pages 533–538. CSREA Press, 2009. 62

[48] E. Dussaud, W. Symes, P. Williamson, L. Lemaistre, P. Singer, B. Denel et A. Cherrett : Computational strategies for reverse-time migration. *SEG Technical Program Expanded Abstracts*, 27(1):2267–2271, 2008. 13

[49] J. T. Etgen et M. J. O'Brien : Computational methods for large-scale 3D acoustic finite-difference modeling : A tutorial. *Geophysics*, 72(5):SM223–SM230, 2007. 22

[50] R. P. Fletcher, X. Du et P. J. Fowler : Reverse Time Migration in tilted transversely isotropic (TTI) media. *Geophysics*, 74(6):WCA179–WCA187, 2009. 33

[51] B. Fornberg : The pseudospectral method : Comparisons with finite differences for the elastic wave equation. *Geophysics*, 52(4):483–501, 1987. 26

[52] B. Fornberg : Generation of finite difference formulas on arbitrarily spaced grids. *Math. Comput*, (51), 1988. 21, 23, 93

[53] Haohuan Fu, Robert G. Clapp, Oskar Mencer et Oliver Pell : *Accelerating 3D convolution using streaming architectures on FPGAs*, chapitre 609, pages 3035–3039. 53

[54] J. Gazdag : Modeling of the acoustic wave equation with transform methods. *Geophysics*, 46:854, juin 1981. 26

[55] gpgpu : Genenral Purpse computation with GPU. http://www.gpgpu.org/. 41

[56] Green500 : . www.green500.org/. 34

[57] Andreas Griewank et Andrea Walther : Algorithm 799 : Revolve : An implementation of checkpointing for the reverse or adjoint mode of computational differentiation. *ACM Trans. Math. Softw.*, 26(1):19–45, mars 2000. 100

[58] W. D. Gropp : Parallel computing and domain decomposition. *In* Tony F. Chan, David E. Keyes, Gérard A. Meurant, Jeffrey S. Scroggs et Robert G. Voigt, éditeurs : *Fifth International Symposium on Domain Decomposition Methods for Partial Differential Equations*, Philadelphia, PA, USA, 1992. SIAM. 39

[59] Nabil Hadj Kacem : *Intégration des données sismiques pour une modélisation statique et dynamique plus réaliste des réservoirs*. Thèse de doctorat, Paris, France, 2006. 16

[60] Dave Hale : Recursive Gaussian filters. Rapport technique, Center for Wave Phenomena, Colorado School of Mines, 2006. 116

[61] Matt Hall : Smooth operator : Smoothing seismic interpretations and attributes. *The Leading Edge*, 26(1):16–20, 2007. 116

[62] S. Henry : Understanding seismic amplitudes. *AAPG Explorer*, July 2004. 15

[63] O. Holberg : Computational aspects of the choice of operator and sampling interval for numerical differentiation in large-scale simulation of wave phenomena. *Geophysical prospecting*, 35(6):629–655, 1987. 23, 92, 93

[64] C. Hsiung et W. Butscher : A numerical seismic 3D migration model for vector multiprocessors. *Parallel Computing*, 1(2):113 – 120, 1984. 36

[65] L. T. Ikelle et L. Amundsen, éditeurs. *Introduction to Petroleum Seismology*. Society Of Exploration Geophysicists, 2005. 12

[66] M. Jannane, W. Beydoun, E. Crase, D. Cao, Z. Koren, E. Landa, M. Mendes, A. Pica, M. Noble, G. Roeth, S. Singh, R. Snieder, A. Tarantola, D. Trezeguet et M. Xie : Wavelengths of earth structures that can be resolved from seismic reflection data. *Geophysics*, 54(7):906–910, 1989. 12

[67] Won-Ki Jeong, Ross Whitaker et Mark Dobin : Interactive 3D seismic fault detection on the graphics hardware, 2006. 104

[68] Benjamin J. Kadlec et Geoffrey A. Dorn : Leveraging graphics processing units (GPUs) for real-time seismic interpretation. *The Leading Edge*, 29(1):60–66, 2010. 104

[69] Benjamin J. Kadlec, Geoffrey A. Dorn et Henry M. Tufo : Interactive visualization and interpretation of geologic surfaces in 3D seismic data. *SEG, Expanded Abstracts*, 28(1):1147–1151, 2009. 104

[70] Benjamin James Kadlec : *Interactive GPU-based visulation and structure analysis of three-dimensional implicit surfaces for seismic interpretation.* Thèse de doctorat, Boulder, CO, USA, 2009. 104

[71] Frank C. Kampe et Tung M. Nguyen : Performance comparison of the Cray-2 and Cray X-MP on a class of seismic data processing algorithms. *Parallel Computing*, 7(1):41 – 53, 1988. 36

[72] W. Kessinger et M.A. Ramaswamy : Subsalt imaging using mode converted energy and acoustic depth migration. *SEG Technical Program Expanded Abstracts*, 15(1):566–569, 1996. 33

[73] N Khaled, P. Capelle, L. Bovet, S. Tchikanha et Hill D. : A coil shooting - acquisition case study in the angolan deep offshore (extended abstract). *In 74th EAGE Conference and Exhibition*, June 2012. 90, 91

[74] A. Klöckner, N. Pinto, Y. Lee, B.C. Catanzaro, P. Ivanov et A. Fasih : PyCUDA : GPU Run-Time Code Generation for High-Performance Computing. *CoRR*, abs/0911.3456, 2009. 51

[75] D. Komatitsch et R. Martin : An unsplit convolutional perfectly matched layer improved at grazing incidence for the seismic wave equation. *Geophysics*, 72(5):SM155–SM167, 2007. 28

[76] D. Komatitsch et J. Tromp : A perfectly matched layer absorbing boundary condition for the second-order seismic wave equation. *Geophysical Journal International*, 154:146–153, juillet 2003. 28

[77] D. Kosloff et E. Baysal : Forward modeling by a fourier method. *Geophysics*, 47(10): 1402–1412, 1982. 26

[78] R. Kosloff et D. Kosloff : Absorbing boundaries for wave propagation problems. *Journal of Computational Physics*, 63(2):363 – 376, 1986. 28

[79] S. Larsen, R. Wiley, P. Roberts et L. House : Next-generation numerical modeling : Incorporating elasticity, anisotropy and attenuation. *SEG Technical Program Expanded Abstracts*, 20(1):1218–1221, 2001. 33

[80] P.-Fr. Lavallée et P. Wautelet : *Programmation hybride MPI-OpenMP*. IDRIS, April 2012. 40

[81] M. Lavergne : *Méthodes Sismiques*. Technip, 1986. 10

[82] A. R. Levander : Fourth-order finite-difference P-SV seismograms. *Geophysics*, 53(11): 1425–1436, 1988. 25

[83] Jim Ching-Rong Lin et Kaihong Wei : Interactive 3D seismic-attribute volume generation with parallel graphics hardware. *SEG Technical Program Expanded Abstracts*, 26(1):907–911, 2007. 104

[84] Christopher Liner, Chun-Feng Li, Adam Gersztenkorn et John Smythe : SPICE : A new general seismic attribute. *SEG Technical Program Expanded Abstracts*, 23(1):433–436, 2004. 15

[85] L. R. Lines, R. Slawinski et R. P. Bording : A recipe for stability of finite-difference wave-equation computations. *Geophysics*, 64(3):967–969, 1999. 24

[86] Q-H. Liu et J. Tao : The perfectly matched layer for acoustic waves in absorptive media. *The Journal of the Acoustical Society of America*, 102(4):2072–2082, 1997. 28

[87] D. Loewenthal, L. Lu. and R. Roberson et J. Sherwood : The wave equation applied to migration. *Geophysical Prospecting*, (24):380–399, 1976. 29

[88] Kurt J. Marfurt : Robust estimates of 3D reflector dip and azimuth. *Geophysics*, 71(4):P29–P40, 2006. 112, 113

[89] A. Massala, N. Keskes, L. Goncalves-Ferreira, C. Onu et P. Cordier : An Innovative Attribute For Enhancing Faults Lineaments And Sedimentary Features During 2G&R Interpretation. *In SPE Annual Technical Conference and Exhibition (ATCE13)*, 2013. 111

[90] Jiayuan Meng et Kevin Skadron : Performance modeling and automatic ghost zone optimization for iterative stencil loops on GPUs. *In Proceedings of the 23rd international conference on Supercomputing*, ICS '09, pages 256–265, New York, NY, USA, 2009. ACM. 63

[91] P. Micikevicius : 3D finite difference computation on GPUs using CUDA. *In GPGPU-2 : Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units*, pages 79–84, New York, NY, USA, 2009. ACM. 63, 75

[92] P. Micikevicius et G. Ruetsch : Optimizing matrix transpose in cuda, 2009. 70, 105, 107

[93] P. Moczo, E. Bystricky, J. M. Carcione et M. Bouchon : Hybrid modeling of P-SV seismic motion at inhomogeneous viscoelastic topographic structures. *Bulletin of the Seismological Society of America*, 87:1305–1323, 1997. 25

[94] P. Moczo, J. Kristek et L. Halada : *The Finite-Difference Method for Seismologists An Introduction*. Comenius University Bratislava, 2004. 25

[95] I. R. Mufti : Large-scale three-dimensional seismic models and their interpretive significance. *Geophysics*, 55(9):1166–1182, 1990. 24

[96] J. Myre, S. D. C. Walsh, D. Lilja et M. O. Saar : Performance analysis of single-phase, multiphase, and multicomponent lattice-Boltzmann fluid flow simulations on GPU clusters. *Concurrency and Computation : Practice and Experience*, 23(4):332–350, 2011. 104

[97] T Nemeth, J. Stefani, W. Liu, Dimond R., O. Pell et R. Ergas : *An implementation of the acoustic wave equation on FPGAs*, chapitre 579, pages 2874–2878. 53

[98] H.H. Ngo : Le calcul d'attributs sismiques sur cartes graphiques. Rapport technique, Total, 2012. 121

[99] Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim et Pradeep Dubey : 3.5-D blocking optimization for stencil computations on modern CPUs and GPUs. *In Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–13, Washington, DC, USA, 2010. IEEE Computer Society. 63

[100] A. Nukada, Y. Ogata, T. Endo et S. Matsuoka : Bandwidth intensive 3-d fft kernel for gpus using cuda. *In High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–11, 2008. 94

[101] NVIDIA. *CUDA CUBLAS Library*, 1 édition, January 2007. 50

[102] NVIDIA. *CUDA CUFFT Library*, 1 édition, June 2007. 26, 50, 94

[103] Nvidia : Next generation CUDA compute architecture : Fermi., 2009. 46

[104] Nvidia : Tuning CUDA applications for Fermi, May 2011. 46, 104

[105] NVIDIA. *CUSPARSE Library*, 5 édition, October 2012. 50

[106] NVIDIA. *NVIDIA CUDA C Programming Guide*, 4.2 édition, April 2012. 69

[107] NVIDIA. *NVIDIA Performance Primitives*, 5 édition, September 2012. 50

[108] J. S. Ogilvie et G. W. Purnell : Effects of salt-related mode conversions on subsalt prospecting. *Geophysics*, 61(2):331–348, 1996. 33

[109] Opengl, D. Shreiner, M. Woo, J. Neider et T. Davis : *OpenGL(R) Programming Guide : The Official Guide to Learning OpenGL(R), Version 2 (5th Edition)*. Addison-Wesley Professional, août 2005. 43

[110] J. D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn et Tim Purcell : A survey of general-purpose computation on graphics hardware. *Computer Graphics Forum*, 26(1):80–113, mars 2007. 44

[111] P.C. Pratt-Szeliga, J.W. Fawcett et R.D. Welch : Rootbeer : Seamlessly using gpus from java. *In High Performance Computing and Communication 2012 IEEE 9th International Conference on Embedded Software and Systems (HPCC-ICESS), 2012 IEEE 14th International Conference on*, pages 375–380, 2012. 51

[112] R. Rabenseifner, G. Hager et G. Jost : Hybrid MPI/OpenMP parallel programming on clusters of multi-core smp nodes. *In PDP*, pages 427–436, 2009. 38

[113] R. Rabenseifner et G. Wellein : Communication and Optimization Aspects of Parallel Programming Models on Hybrid Architectures. *International Journal of High Performance Computing Applications*, 17(1):49–62, février 2003. 40

[114] M. Reshef, D. Kosloff, M. Edwards et C. Hsiung : Three-dimensional acoustic modeling by the fourier method. *Geophysics*, 53(9):1175–1183, 1988. 26, 36

[115] Gabriel Rivera et Chau-Wen Tseng : Tiling optimizations for 3D scientific computations. *In Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, Supercomputing '00, Washington, DC, USA, 2000. IEEE Computer Society. 62

[116] E. Robein : *Velocities, time-imaging, and depth-imaging in reflection seismics : principles and methods*. EAGE Publications, Houton, The Netherlands, 2003. 11

[117] J. O. A. Robertsson, J .O. Blanch et W. W. Symes : Viscoelastic finite-difference modeling. *Geophysics*, 59(9):1444–1456, 1994. 33

[118] Christopher J. Rossbach, Jon Currey, Mark Silberstein, Baishakhi Ray et Emmett Witchel : Ptask : operating system abstractions to manage GPUs as compute devices. *In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 233–248, New York, NY, USA, 2011. ACM. 121

[119] E. H. Saenger et T. Bohlen : Finite-difference modelling of viscoelastic and anisotropic wave propagation using the rotated staggered grid. *Geophysics*, 69:583–591, 2004. 21, 26

[120] G. Saldana-Gonzalez et M. Arias-Estrada : 2009-12-01. 53

[121] W. A. Schneider : Integral formulation for migration in two and three dimensions. *Geophysics*, 43(1):49–76, 1978. 13

[122] A. Sei et W. Symes : Dispersion analysis of numerical wave propagation and its computational consequences. *J. Sci. Comput.*, 10(1):1–27, 1995. 22, 24, 26

[123] Marc Snir, Steve W. Otto, David W. Walker, Jack Dongarra et Steven Huss-Lederman : *MPI : The Complete Reference*. MIT Press, Cambridge, MA, USA, 1995. 38

[124] Herb Sutter et James Larus : Software and the concurrency revolution. *Queue*, 3:54–62, September 2005. 56

[125] W. Symes : Reverse time migration with optimal checkpointing. *GEOPHYSICS*, 72(5): SM213–SM221, 2007. 100

[126] M. Turhan Taner, James S. Schuelke, Ronen O'Doherty et Edip Baysal : *Seismic attributes revisited*, chapitre 308, pages 1104–1106. 15

[127] T. Taner : Seismic attributes. *CSEG Recorder*, September 2001. 15

[128] A. Tarantola : Inversion of seismic reflection data in the acoustic approximation. *Geophysics*, 49(8):1259–1266, 1984. 14

[129] L. Thomsen : Weak elastic anisotropy. *Geophysics*, 51(10):1954–1966, 1986. 33

[130] TOP500 : TOP500 Supercomputing Sites. www.top500.org/. 37

[131] Y. Torres, A. Gonzalez-Escribano et D.R. Llanos : Understanding the impact of CUDA tuning techniques for Fermi. *In High Performance Computing and Simulation (HPCS), 2011 International Conference on*, pages 631 –639, july 2011. 46, 104

[132] S. Venkatasubramanian et R.W. Vuduc : Tuned and wildly asynchronous stencil kernels for hybrid CPU/GPU systems. *In ICS '09 : Proceedings of the 23rd international conference on Supercomputing*, pages 244–255, New York, NY, USA, 2009. ACM. 62

[133] J. Virieux : P-SV wave propagation in heterogeneous media, velocity-stress finite difference method. *Geophysics*, 51:889–901, 1986. 25

[134] J. Virieux et S. Operto : An overview of full-waveform inversion in exploration geophysics. *Geophysics*, 74(6):WCC1–WCC26, 2009. 14

[135] Lucas J Van Vliet, Ian T Young et Piet W Verbeek : Recursive gaussian derivative filters. *Pattern Recognition, Inte rnational Conference on*, 1:509, 1998. 117

[136] W. Wu, L. R. Lines et H. Lu : Analysis of higher-order, finite-difference schemes in 3D reverse-time migration. *Geophysics*, 61(3):845–856, 1996. 24

[137] Y. Yan, M. Grossman et V. Sarkar : Jcuda : A programmer-friendly interface for accelerating java programs with cuda. *In Proceedings of the 15th International Euro-Par Conference on Parallel Processing*, Euro-Par '09, pages 887–899, Berlin, Heidelberg, 2009. Springer-Verlag. 51

[138] F. Ye et C. Chu : Dispersion-relation-preserving finite difference operators : derivation and application. *SEG Technical Program Expanded Abstracts*, 24(1):1783–1786, 2005. 21

[139] O. Yilmaz : *Seismic Data Analysis*, volume 10 de *Investigations in Geophysics*. Society Of Exploration Geophysicists, Tulsa, USA, 2 ed édition, January 2001. 11, 12

[140] Ian T. Young et Lucas J. van Vliet : Recursive implementation of the gaussian filter. *Signal Processing*, 44(2):139 – 151, 1995. 117

[141] E Zamboni, S. Tchikanha, L Lemaistre, L. Bovet, B ; Webb et Hill D. : A coil (full azimuth) and narrow azimuth processing case study in angola deep offshore (extended abstract). *In 74th EAGE Conference and Exhibition*, June 2012. 90

[142] Y. Zhou et H. Tchelepi : Multi-core and GPU parallelization of a general purpose reservoir simulator. *In ECMOR XIII*, 2012. 104

[143] O. C. Zienkiewicz, R. L. Taylor et J. Z. Zhu : *The finite element method its basis and fundamentals*. Elsevier, 2005. six edition. 27