Algorithmique hiérarchique parallèle haute performance pour les problèmes à N-corps

Pierre Fortin

N° d'ordre : 3273

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Pierre FORTIN**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Algorithmique hiérarchique parallèle haute performance pour les problèmes à N-corps

---

Soutenue le : 27 novembre 2006

Après avis des rapporteurs :

Frédéric Desprez	Directeur de Recherche	
Yousef Saad	Professeur	

Devant la commission d'examen composée de :

Evangélie Athanassoula	Astronome	
Olivier Coulaud	Directeur de Recherche	Directeur de thèse
Frédéric Desprez	Directeur de Recherche	Président
Juan Elezgaray	Chargé de Recherche ..	Rapporteur
Jean Roman	Professeur	Directeur de thèse
Yousef Saad	Professeur	

– 2006 –

Remerciements

Je tiens tout d'abord à remercier vivement Frédéric Desprez et Yousef Saad pour l'honneur qu'ils m'ont fait en acceptant de rapporter ce (long) manuscrit. Je sais que leur temps est précieux, et je leur suis reconnaissant de m'en avoir consacré une partie. Je remercie à nouveau Frédéric Desprez pour avoir accepté de présider le jury de cette thèse.

Je tiens ensuite à remercier Lia Athanassoula pour l'intérêt continu qu'elle a porté à mes travaux, pour avoir accepté d'être membre du jury et pour m'avoir d'ores et déjà accueilli à Marseille. Je ne doute pas que notre collaboration sera fructueuse.

Je souhaite aussi remercier Juan Elezgaray d'avoir accepté d'être membre du jury et pour le regard différent qu'il a porté sur mes travaux.

Je tiens bien sûr à remercier Olivier Coulaud et Jean Roman pour avoir encadrer cette thèse. Outre l'expérience qu'ils m'ont transmise, et le temps qu'ils ont pris sur leurs weekends et leurs soirées afin de guider mes travaux et d'y contribuer, je leur suis très reconnaissant de leur écoute et de leur soutien, et des nombreuses, mais précieuses, « corrections au crayon rouge » !

Merci également à tous les membres du projet ScAlApplix (administratifs inclus !), notammment pour les réponses à mes (nombreuses) questions.

D'autre part, ces années de thèse resteront pour moi associées aux divers enseignements que j'ai eu le plaisir de pouvoir effectuer à l'Université Bordeaux 1 en informatique. Je tiens à remercier tous ceux avec qui j'ai pu collaboré et qui m'ont aidé dans cette tâche, parfois ingrate, mais toujours vivifiante et épanouissante. Je suis ici particulièrement reconnaissant à Raymond Namyst et à Pierre-André Wacrenier.

Sur un plan plus personnel, je tiens à saluer tous ceux avec qui j'ai partagé ces années au LaBRI (et surtout en dehors) : les membres de la salle 122, de ScAlApplix et du reste du labo. Je les retrouverai toujours avec plaisir, que ce soit à Bordeaux ou ailleurs, et je ne compte pas laisser filer les amitiés que j'y ai nouées. Ce fût par ailleurs pour moi un très grand honneur d'être membre de la légendaire équipe de foot du LaBRI, et je reste encore aujourd'hui particulièrement fier de nos nombreuses (? !) victoires.

J'ai bien sûr aussi une pensée pour mes parents et mon frère, que je remercie profondément d'avoir toujours su être présents malgré l'éloignement.

Enfin, je ne saurais comment remercier Laure pour tout ce qu'elle m'a apporté ; pour son soutien constant, malgré les délais sans cesse repoussés et les promesses non tenues. Sans elle, je n'aurais pas pu parvenir à un tel résultat.

Algorithmique hiérarchique parallèle haute performance pour les problèmes à N-corps

Résumé : Cette thèse porte sur la méthode dite « méthode multipôle rapide » qui résout hiérarchiquement le problème à N-corps avec une complexité linéaire pour n'importe quelle précision. Dans le cadre de l'équation de Laplace, nous souhaitons pouvoir traiter efficacement toutes les distributions de particules rencontrées en astrophysique et en dynamique moléculaire.

Nous étudions tout d'abord deux expressions distinctes du principal opérateur (« multipôle-to-local ») ainsi que les bornes d'erreur associées. Pour ces deux expressions, nous présentons une formulation matricielle dont l'implémentation avec des routines BLAS (Basic Linear Algebra Subprograms) permet d'améliorer fortement l'efficacité de calcul. Dans la gamme de précisions qui nous intéresse, cette approche se révèle plus performante que les améliorations existantes (FFT, rotations et ondes planes), pour des distributions uniformes ou non.

Outre une nouvelle structure de données pour l'octree sous-jacent et des contributions algorithmiques à la version adaptative, nous avons aussi efficacement parallélisé notre méthode en mémoire partagée et en mémoire distribuée. Enfin, des comparaisons avec des codes dédiés justifient l'intérêt de notre code pour des simulations en astrophysique.

Mots clés : problème à N-corps, méthode multipôle rapide, algorithme de Barnes & Hut, équation de Laplace, équation de Poisson, astrophysique, dynamique moléculaire, borne d'erreur, Transformée Rapide de Fourier, rotations, ondes planes, routines BLAS, octree, parallélisme, mémoire partagée, mémoire distribuée

Discipline : Informatique

LaBRI (UMR CNRS 5800) et Projet INRIA Futurs ScAlApplix[1],
Université Bordeaux 1,
351, cours de la libération
33405 Talence Cedex (FRANCE)

[1]ScAlApplix : Schémas et Algorithmes Hautes Performances pour les Applications Scientifiques Complexes, `http ://www.labri.fr/scalapplix`.

High performance parallel hierarchical algorithmic for N-body problems

Abstract: This thesis focuses on the Fast Multipole Method which hierarchically solves the N-body problem with a linear operation count for any given precision. When considering Laplace equation, we aim at treating efficiently all particle distributions that arise in astrophysics and in molecular dynamics.

We first study two different expressions of the main operator ("multipole-to-local") as well as the corresponding error bounds. For these two expressions, we present a matrix formulation whose implementation with BLAS routines (Basic Linear Algebra Subprograms) offers impressive runtime speedup. For the targeted precisions, this approach appears to outperform the existing enhancements (FFT, rotations and plane waves), in case of both uniform and non uniform distributions.

In addition to a new octree data structure and to algorithmic improvements of the adaptive version, we have also efficiently parallelized our method for shared and distributed memory architectures. Finally, comparisons with specialized codes justify the interest of our code for astrophysical simulations.

Keywords: N-body problem, Fast Multipole Method, Barnes-Hut algorithm, Laplace equation, Poisson equation, astrophysics, molecular dynamics, error bound, Fast Fourier Transform, rotations, plane waves, BLAS routines, octree, parallelism, shared memory, distributed memory

Discipline: Computer Science

LaBRI,
Université Bordeaux 1,
351, cours de la libération
33405 Talence Cedex (FRANCE)

Table des matières

Chapitre 1

Introduction générale

Le problème à N-corps dans les simulations numériques

De nos jours, les simulations numériques sont couramment employées pour étudier et approfondir la compréhension de nombreux domaines physiques. Ces simulations peuvent être conçues à partir de la modélisation mathématique des phénomènes physiques : c'est par exemple le cas des méthodes dites des « éléments finis » et des « différences finies » qui sont basées sur des équations aux dérivées partielles. Mais ces simulations peuvent aussi être réalisées en manipulant un grand nombre de « particules élémentaires » qui interagissent entre elles deux à deux selon une loi physique ; on parle alors de problème à N-corps.

Ces N particules peuvent correspondre à un corps physique identifiable (molécule ou atome en dynamique moléculaire, étoile en astrophysique) ou résulter de la discrétisation d'une fonction de densité. Dans les deux cas, elles permettent de modéliser un phénomène physique à l'échelle supérieure (amas moléculaire, galaxie) sur plusieurs pas de temps. A chaque pas de temps, les interactions entre les N corps doivent être calculées afin de pouvoir mettre à jour leur énergie et éventuellement leur position (à l'aide d'un schéma d'intégration en temps). La pertinence de la simulation ainsi effectuée dépend alors essentiellement du nombre de corps N utilisés. Plus N est important et plus la simulation sera précise, ou plus grande sera la taille du phénomène simulé pour une précision donnée. En pratique, la valeur maximale de N supportée par les puissances de calcul actuellement disponibles varie d'un domaine physique à l'autre, en fonction de la précision requise pour le phénomène observé.

En astrophysique, plusieurs millions de particules sont ainsi couramment utilisées pour l'étude des galaxies. Les N corps interagissent alors selon la loi de la gravitation universelle de Newton et la précision ici requise est basse (entre 10^{-2} et 10^{-3}).

En biologie, des systèmes moléculaires de plus en plus gros, comportant des centaines de milliers de molécules, sont nécessaires pour modéliser le comportement d'un ensemble de protéines (ADN, ARN). Les N corps interagissent alors selon la loi de Coulomb dans le cadre de la dynamique moléculaire ; la précision requise est alors plus importante (entre 10^{-5} et 10^{-7}).

On retrouve ce problème à N-corps dans encore bien d'autres domaines d'application : en dynamique des fluides (codes SPH, méthode du vortex), en électromagnétisme (avec une formulation intégrale des équations de Maxwell ou d'Helmholtz), en ingénierie des ma-

tériaux (où la dynamique moléculaire est employée pour calculer le coefficient d'élasticité propre au modèle de Young), en calcul de capacitance, en modélisation d'objets $3D$ ...

Pour les simulations de grande taille, une parallélisation des codes de simulation est par ailleurs indispensable. Les architectures parallèles de grande taille permettent en effet de passer à l'échelle tant sur le plan du gain vis-à-vis du temps de calcul nécessaire que sur le plan de la taille des données à stocker en mémoire.

Cependant, la méthode directe pour évaluer les interactions entre les $N(N-1)$ paires conduit à une complexité en $\mathcal{O}(N^2)$; ce coût de calcul devient rapidement prohibitif lorsque N augmente, y compris dans le cas parallèle. De nouvelles méthodes ont donc été introduites au cours des années 1980 : en autorisant une certaine marge d'erreur dans les calculs (par ailleurs déjà affectés par la précision machine), ces nouveaux algorithmes approchent tout ou partie de ces interactions ce qui a pour effet de réduire fortement les temps de calcul nécessaires. Plus précisément, le potentiel est décomposé en un champ proche, calculé selon la méthode directe, et un champ lointain qui est approché.

Parmi ces méthodes, les méthodes hiérarchiques se basent sur une structure arborescente, un *octree*, pour découper l'espace. Les deux méthodes hiérarchiques les plus répandues sont l'algorithme de Barnes & Hut, qui s'exécute en $\mathcal{O}(N \log N)$, et la méthode « multipôle rapide » (*Fast Multipole Method*, ou FMM) qui présente une complexité linéaire grâce à l'introduction de développements multipôles et de développements locaux. De nombreux travaux ont progressivement permis d'améliorer l'efficacité de ces méthodes en minimisant le nombre d'opérations à effectuer.

Cependant sur les architectures modernes, la vitesse de calcul des processeurs est actuellement bien supérieure à la vitesse d'accès à la mémoire. Cette différence génère des temps d'attente conséquents pour accéder aux données en mémoire : le processeur est alors inutilisé, ce qui affecte considérablement les temps de calcul nécessaires à ces simulations. En modifiant l'algorithmique des méthodes employées et l'agencement des données en mémoire, d'important gains en performance sont ainsi possibles. Ceci est couramment employé par exemple dans les domaines de l'algèbre linéaire dense et creuse, et est donc susceptible de contribuer efficacement à la résolution du problème à N-corps.

Objectifs et organisation de la thèse

Cette thèse a pour objectif de traiter efficacement le problème à N-corps dans le cadre de l'équation de Poisson, avec des distributions de particules uniformes ou non, et pour une gamme de précisions couvrant celles requises en astrophysique et en dynamique moléculaire. La méthode retenue sera la méthode multipôle rapide et nous souhaitons plus précisément améliorer la FMM au niveau algorithmique, au niveau de son implémentation haute-performance et de sa parallélisation.

Algorithmique : il s'agit de mieux adapter la FMM aux distributions non uniformes de particules, ce qui implique aussi la conception d'une nouvelle structure de données ;

Implémentation haute-performance : nous souhaitons mettre en place une approche matricielle afin de pouvoir employer des routines BLAS (Basic Linear Algebra Subprograms). Ces routines effectuent en effet très efficacement des opérations standards en algèbre linéaire en remplissant au maximum les pipelines des unités de calcul flottant du processeur, et ce grâce à une prise en compte optimale des différentes couches de

la hiérarchie mémoire. Le nombre d'opérations effectué est le même mais les gains en termes de temps CPU et de puissance de calcul sont très importants. Cette approche matricielle sera mise en place pour l'opérateur multipôle-vers-local qui convertit un développement multipôle en un développement local, et représente l'opération la plus coûteuse dans le calcul du champ proche. Elle sera ainsi comparée et validée par rapport aux améliorations existantes qui visent toutes à réduire la complexité théorique de cet opérateur (Transformée Rapide de Fourier, rotations et ondes planes) ;

Parallélisation : nous souhaitons tirer parti des deux approches principales dans ce domaine : l'approche multi-*threads* en mémoire partagée et l'approche multi-processus en mémoire distribuée.

Le fruit de ce travail consistera en un code implémentant et validant toutes ces contributions. Ce code, programmé en langage C, est appelé FMB pour pour *Fast Multipole with BLAS* (ou *Fast Multipole in Bordeaux* ...).

Cette thèse est divisée en quatre parties. La première partie présente une étude de l'état de l'art des différentes méthodes mises en place pour traiter le problème à N-corps (chapitre 2). Nous nous intéressons principalement aux méthodes hiérarchiques et nous étudions en particulier l'algorithme de Barnes & Hut et la FMM. Nous justifierons alors le choix de la FMM, et nous présenterons un état de l'art détaillé de cette méthode dans le cadre de la résolution de l'équation de Poisson. A la fin de cette première partie, le chapitre 3 décrira précisément notre positionnement vis-à-vis de cette méthode et les contributions que nous allons apporter. La deuxième partie de cette thèse est consacrée à l'introduction de notre approche matricielle basée sur les routines BLAS et à sa comparaison avec les autres schémas de calcul permettant d'accélérer l'opérateur multipôle-vers-local (chapitres 4 à 7). Ces travaux ont été présentés dans le rapport de recherche [1] et ont fait l'objet de la soumission [2]. Cette partie ayant été limitée au cas d'une distribution uniforme de particules, nous étendrons notre approche BLAS aux cas des distributions non uniformes dans la troisième partie (chapitre 8), tout en contribuant à la version adaptative de la FMM sur le plan algorithmique et sur le plan des structures de données. Ces travaux ont l'objet d'une publication [3] et d'une soumission [4]. La dernière partie (chapitre 9) sera consacrée à la double parallélisation de notre méthode : en mémoire partagée à l'aide d'une programmation en mode multi-*thread* (*threads* POSIX) et en mémoire distribuée à l'aide d'une programmation par échange de messages entre différents processus (standard MPI). Enfin, le chapitre 10 nous permettra de présenter un bilan de nos contributions avant d'ouvrir plusieurs perspectives de recherche.

Première partie

Etat de l'art et positionnement

Chapitre 2

Etat de l'art & description de la méthode multipôle rapide (FMM)

Sommaire

2.1 Le problème à N-corps

Dans le cadre des simulations numériques, le problème à N-corps décrit le calcul des interactions entre N corps, chaque interaction mettant en jeu une paire parmi ces N corps. Ces simulations sont réalisées sur de nombreux pas de temps, et à chaque pas de temps, on doit évaluer les interactions au sein de chacune des $N(N-1)$ paires parmi les N corps et en déduire les accélérations correspondantes grâce à la deuxième loi de Newton (ou principe fondamental de la dynamique). Les positions des N corps sont alors mises à jour à l'aide d'un schéma d'intégration en temps. Ce problème est classique en astrophysique et en dynamique

moléculaire ; l'équation de Laplace ($\Delta\Phi = 0$), qui n'est valable que dans le vide, est alors remplacée à cause de la distribution de particules par l'équation de Poisson :

$$\Delta\Phi + \rho = 0, \tag{2.1}$$

ρ désignant la densité de charge ou de masse suivant le cas.

En astrophysique [20] [43] [120], les simulations gravitationnelles modélisent à l'aide de N corps les interactions entre galaxies ou au sein d'une galaxie : ces N corps peuvent soit directement correspondre à N étoiles, soit résulter de la discrétisation d'une fonction de densité. La loi de gravitation universelle de Newton est alors utilisée, et le potentiel de gravitation $\Phi(r)$, à une distance r d'une masse m, est donné par la solution élémentaire de l'équation de Poisson (ici en trois dimensions) :

$$\Phi(r) = -\mathcal{G}\frac{m}{r},$$

où $\mathcal{G}$ est la constante de gravitation universelle : $\mathcal{G} \approx 6,67259.10^{-11} N.m^2.kg^{-2}$. Le vecteur de la force exercée par une particule[1] A de masse m_A sur une particule B de masse m_B s'exprime ainsi (les vecteurs sont notés en gras dans ce document) :

$$\mathbf{F}_{A\to B} = -\mathcal{G}\frac{m_A m_B}{r_{AB}^2}\mathbf{u}_{AB}$$

où r_{AB} est la distance entre A et B, et $\mathbf{u}_{AB}$ le vecteur unitaire entre A et B défini par : $\mathbf{u}_{AB} = \frac{\mathbf{r}_B - \mathbf{r}_A}{r_{AB}}$, $\mathbf{r}_A$ et $\mathbf{r}_B$ étant les vecteurs positions de A et de B.

En dynamique moléculaire [7] [24] [44] [49] [51] [119], le principe est le même : les interactions électrostatiques, régies par la loi de Coulomb, entre N atomes ou N molécules, sont évaluées au sein des $N(N-1)$ paires. Le potentiel de Coulomb à une distance r d'une charge q est alors donné, en trois dimensions, par

$$\Phi(r) = \frac{1}{4\pi\epsilon_0}\frac{q}{r},$$

où ϵ_0 est la permittivité diélectrique du vide : $\epsilon_0 \approx 8,854.10^{-12} F.m^{-1}$. Le vecteur force correspondant, dû à une particule A de charge q_A, sur une particule B de charge q_B, s'exprime ainsi :

$$\mathbf{F}_{A\to B} = \frac{1}{4\pi\epsilon_0}\frac{q_A q_B}{r_{AB}^2}\mathbf{u}_{AB}.$$

En plus de l'astrophysique et de la dynamique moléculaire, le problème à N-corps est au cœur de nombreux autres types de simulations régies par d'autres potentiels :

- à nouveau en dynamique moléculaire, avec le potentiel de Lennard-Jones utilisé pour le calcul de la force de Van der Waals [44] [51], et dont l'expression est de la forme :

$$\Phi(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right];$$

- pour le potentiel de Yukawa, ou potentiel de Coulomb écranté (*screened Coulomb potential*) [28] [63] ;

[1] Dans ce document nous parlerons indifféremment de particule et de corps.

- en dynamique des fluides, avec les codes SPH (*Smoothed Particle Hydrodynamics*) [114] [122] ou avec la méthode dite du vortex [45] [109] ;
- en physique des plasmas [73] [109] ;
- en radiosité [109] [111] ;
- en électromagnétisme avec une formulation intégrale des équations de Maxwell [39] [117] ou de Helmholtz [52] [64], dont le noyau oscillant est de la forme $\frac{e^{ik|x-y|}}{|x-y|}$;
- pour des calculs de capacitance (*VLSI capacitance extraction*) [93] [94] ;
- en modélisation d'objets $3D$ (évaluation rapide de *Radial Basis Functions*) [31] ;
- et dans encore bien d'autres domaines : voir par exemple [61] [107] [109] [127].

Dans toutes ces simulations, la méthode directe calcule l'énergie du système en évaluant les interactions entre les $N(N-1)$ paires avec une complexité quadratique en termes de nombre d'opérations. Le *calcul direct* se présente en effet sous la forme suivante (ici dans le cas d'un potentiel électrostatique, q_i étant la charge de la particule i localisée en z_i) :

$$E = \sum_{i=1}^{N} \sum_{j \neq i} \frac{q_i q_j}{|z_i - z_j|}. \tag{2.2}$$

Certes la troisième loi de Newton sur le principe des interactions réciproques s'applique pour certains potentiels, ce qui implique : $\Phi_{A \to B} = \Phi_{B \to A}$, et $\mathbf{F}_{A \to B} = -\mathbf{F}_{B \to A}$. Ceci permet d'évaluer en même temps l'interaction de i sur j et l'interaction de j sur i, et de diviser ainsi le nombre d'opérations par deux. Mais la complexité demeure quadratique et le temps de calcul devient rapidement prohibitif. Même avec les processeurs les plus rapides actuellement disponibles, il est tout simplement inenvisageable d'effectuer de telles simulations par calcul direct pour des valeurs de N dépassant quelques dizaines de milliers. Or les simulations visées comportent un nombre de particules variant de plusieurs centaines de milliers en chimie moléculaire [49] [51], à plusieurs millions en astrophysique [120], voire bien plus encore dans d'autres cas [128] grâce à des machines parallèles à mémoire distribuée. De nouvelles méthodes ont donc été introduites afin de réaliser de telles simulations : en introduisant une certaine marge d'erreur dans les calculs, ces méthodes approchent tout ou partie de ces interactions et permettent de réduire fortement le nombre d'opérations. En effet les potentiels considérés présentent généralement la propriété suivante :

$$\lim_{|x| \to +\infty} \Phi(|x|) = 0.$$

Il est donc possible d'approcher l'influence due à un groupe éloigné de particules par une particule fictive, ou mieux, par certains types de développements. Certaines de ces méthodes, et notamment les méthodes hiérarchiques, traitent donc différement les particules proches et les particules éloignées. Plus précisément, le potentiel créé au point P est décomposé en une partie *champ proche* et une partie *champ lointain*. Le champ proche, dû aux particules proches de ce point, est calculé directement alors que le champ lointain, dû aux particules éloignées est approché.

$$\Phi = \Phi_{\text{Proche}} + \Phi_{\text{Lointain}}.$$

L'erreur tolérée dépend du type de la simulation physique considérée : alors qu'en astrophysique on accepte une erreur relative comprise entre 10^{-2} et 10^{-3} (« basses » précisions) [30] [43], la dynamique moléculaire requiert plutôt une erreur située entre 10^{-5} et 10^{-7} (« moyennes » précisions).

Avant de détailler ces différentes méthodes, nous abordons un point important qui est celui des différentes distributions de particules rencontrées dans les simulations astrophysiques et en chimie moléculaire. Il est connu (voir par exemple [26]) que les caractéristiques de ces distributions sont diamétralement opposées entre ces deux domaines : outre la différence de précision présentée ci-dessus, la dynamique moléculaire a recours à des distributions uniformes, ou présentant de larges zones uniformes, alors que les simulations gravitationnelles manipulent des distributions hautement non uniformes, où la fonction de densité varie fortement. Comme nous le verrons par la suite, ces différences auront des conséquences importantes sur les méthodes utilisées pour effectuer ces simulations. Nous précisons de plus qu'on peut utiliser en dynamique moléculaire des *conditions aux limites périodiques* où le système moléculaire est répliqué à l'infini dans toutes les dimensions : ceci permet de minimiser le nombre d'atomes dans la simulation et de réduire les effets de surface. Par opposition, nous parlerons de *conditions aux limites libres* dans le cas de système isolé. Enfin, en ce qui concerne les autres potentiels, on rencontre généralement des distributions non uniformes dans l'espace, mais réparties uniformément sur une surface de l'espace : la surface d'un avion en électromagnétisme par exemple [117], d'une structure conductrice dans les calculs de capacitance VLSI [92], ou encore d'une hélice [128].

Dans la suite de ce chapitre nous nous proposons donc d'exposer les différentes méthodes introduites pour résoudre le problème à N-corps. Dans ce cadre général nous nous restreignons à l'évaluation de potentiels gravitationnels ou électrostatiques, et nous nous intéressons aussi bien aux distributions uniformes qu'aux distributions non uniformes de particules, et aux basses comme aux moyennes précisions (les hautes précisions, de l'ordre de 10^{-12}, n'étant pas considérées comme prioritaires). Nous commencerons donc par présenter les méthodes non hiérarchiques, et nous détaillerons ensuite deux méthodes hiérarchiques : la méthode de Barnes & Hut et la méthode dite *Fast Multipole Method*.

2.2 Méthodes non hiérarchiques

Dans cette section, nous présentons des méthodes qui n'ont pas recours à une décomposition hiérarchique de l'espace. Ces méthodes, ainsi que les méthodes hiérarchiques, sont présentées dans le désormais classique *Amara's Recap of Particle Simulation Methods* [58].

2.2.1 Calcul direct

Si la méthode directe reste inexploitable au niveau implémentation logicielle, elle a tout de même été implémentée de façon matérielle sur une carte spéciale, nommée GRAPE pour *GRAvity PipE*, qui est dédiée aux calculs des forces gravitationnelles [77]. Pour quelques dizaines de milliers de particules le résultat est très efficace, et permet même d'obtenir des temps de calculs inférieurs à toutes les autres méthodes. Couplé à la méthode de Barnes & Hut (décrite en section 2.3.2), le résultat est encore plus efficace pour des nombres de particules plus importants [15]. Ces cartes sont cependant uniquement dédiées aux forces garvitationnelles et leur coût est élévé.

2.2.2 Méthode du rayon de coupure

La méthode dite du rayon de coupure (*cut-off method*) est classique en dynamique moléculaire [7]. Comme le potentiel de Coulomb décroît en $1/r$, la majeure partie du potentiel en un point donné est due aux particules environnantes (champ proche) : les contributions dues aux particules éloignées (champ lointain), à savoir avec $r > r_c$ où r_c est le rayon de coupure, sont alors ignorées. Pour les distributions uniformes rencontrées en chimie moléculaire, chaque particule interagit alors avec un nombre constant de particules indépendant de N, et la complexité obtenue est alors en $\mathcal{O}(N)$. Mais cette méthode introduit de sérieuses erreurs dans le calcul des forces [51] [119] : la dynamique de la simulation n'est alors plus cohérente avec les phénomènes physiques normalement observés. La décroissance en $1/r$ n'est en effet pas suffisante pour se permettre d'ignorer les interactions à longue distance du champ lointain.

Le potentiel de Lennard-Jones est lui plus adapté à la méthode du rayon de coupure, ses deux termes variant en $1/r^6$ et en $1/r^{12}$. Mais il apparaît que des erreurs et des instabilités peuvent tout de même être détectées dans ce cas [51].

2.2.3 Méthode *Particle-Mesh (PM)*

Dans la méthode *Particle-Mesh* (PM) [73] un maillage de l'espace est utilisé pour résoudre le problème à N-corps en traitant l'équation de Poisson, présentée à l'équation (2.1), comme une *équation aux dérivées partielles*. La densité en chaque point du maillage est déterminée en fonction des particules voisines et une fois les potentiels et les forces calculés au niveau du maillage, on les ramène aux positions des particules par interpolation. La régularité du maillage permet d'utiliser des techniques à base de Transformées Rapides de Fourier pour accélérer le calcul. La complexité théorique de la méthode s'exprime donc en $\mathcal{O}(N + N_m \log N_m)$, où N_m est le nombre de points du maillage. Le maillage impose cependant une certaine régularité dans la variation du potentiel, et cette méthode est donc mal adaptée à des distributions non uniformes de particules présentant des zones à forte concentration de particules [58] [127].

2.2.4 Méthode *Particle-Particle/Particle-Mesh (P3M)*

Afin d'améliorer la méthode *Particle-Mesh* dans le cas de distributions non uniformes de particules, la méthode *Particle-Particle/Particle-Mesh* (P3M) a été introduite [73]. Alors que la composante à variation lente, due aux interactions à longue distance (champ lointain), est toujours calculée par la méthode PM, la composante à variation rapide, due aux interactions à courte distance (champ proche), est elle calculée par la méthode directe. La précision de la méthode est donc améliorée dans les zones à forte concentration de particules.

A noter que cette méthode P3M a aussi été couplée dans [108] à l'utilisation de développements semblables à ceux utilisés dans la *Fast Multipole Method* décrite ci-dessous.

La méthode PM est aussi à la base de nombreuses autres méthodes, regroupées sous l'appellation de « méthode PM à grilles emboîtées » (*nested-grid particle-mesh methods*, voir [58]).

2.2.5 Méthode TreePM

A la place du calcul direct, il est plus efficace d'avoir recours à une méthode hiérarchique comme l'algorithme de Barnes & Hut (*treecode*) que nous détaillerons ci-après. Un *treecode* est alors utilisé pour le calcul du champ proche, particulièrement dans les zones où les particules sont fortement concentrées, et le champ lointain est évalué par la méthode PM. Cette méthode dite *TreePM* a par exemple été implémentée dans le code d'astrophysique GADGET-2 [114].

2.3 Méthodes hiérarchiques

Les méthodes dites hiérarchiques font référence à un découpage hiérarchique de l'espace, en deux ou en trois dimensions, dont le résultat est la création d'un arbre sur lequel est basé l'algorithmique.

Historiquement, la première méthode hiérarchique pour les problèmes à N-corps a été introduite par Appel [12] à l'aide de *k-d trees*. Un *k-d tree* est construit dans un espace à k dimensions en découpant successivement l'espace dans chacune des dimensions. La complexité initiale de l'algorithme d'Appel en $\mathcal{O}(N \log N)$ a ensuite été ramenée à $\mathcal{O}(N)$ par Esselink (voir [53] pour un résumé de la preuve).

En 1986, Barnes & Hut [20] introduisent les *octrees* (ou *quadtrees* en 2 dimensions) et obtiennent un algorithme dont la complexité s'exprime en $\mathcal{O}(N \log N)$. La définition précise de ces octrees et quadtrees est donnée à la section suivante, et par abus de langage, nous aurons tendance dans ce document à désigner par *octree* aussi bien un octree qu'un quadtree. L'année suivante Greengard et Rokhlin mettent au point la méthode dite FMM, pour *Fast Multipole Method*, qui permet d'obtenir une complexité en $\mathcal{O}(N)$ et qui garantit une borne théorique sur l'erreur introduite [65].

Ces deux méthodes, qui seront présentées en détail aux section 2.3.2 et 2.3.3, reposent toutes deux sur une approximation de la force exercée par un ensemble de particules en un point P donné. L'approximation est acceptée par l'algorithme si la distance séparant le point P de l'ensemble des particules est suffisament importante : c'est la structure hiérarchique de l'octree qui va permettre de déterminer aisément si cette distance est suffisante. Chaque niveau de l'octree constitue en effet une partition de l'espace ; la partition étant de plus en plus raffinée au fur et à mesure que la profondeur de l'octree augmente. Chaque nœud de l'octree constitue donc une portion d'espace, ou *cellule*, contenant un ensemble de particules sources dont la force exercée sur une particule extérieure cible (Barnes & Hut), ou sur une autre cellule cible (FMM), peut être approchée. Lorsque l'approximation ne peut être effectuée, on utilise le calcul direct décrit dans l'équation (2.2).

En reprenant la décomposition du potentiel présentée en section 2.1, le champ proche est donc calculé directement alors que le champ lointain, dû aux particules éloignées et regroupées par cellules, est évalué par une approximation représentant le potentiel induit par toutes les particules de la cellule.

2.3.1 Quadtree & Octree

Nous décrivons ici les quadtrees et les octrees qui sont des structures de données en arbre implémentant un découpage hiérarchique de l'espace en deux dimensions (quadtrees) ou en

trois dimensions (octrees) [105]. Ces structures de données sont en effet mieux adaptées au problème à N-corps que les *k-d trees* (voir [11] par exemple).

On définit tout d'abord la notion de *boîte de calcul* qui correspond à l'ensemble de l'espace considéré contenant toutes les particules. On impose que cette boîte de calcul ait une forme carrée en $2D$ ou cubique en $3D$. La racine de l'octree correspond alors à cette boîte de calcul. Dans un quadtree, on attribue ensuite à chacun des 4 fils d'une cellule un quadrant de l'espace du père, les quadrants étant tous de même taille. Un exemple de quadtree, avec le découpage de l'espace correspondant, est donné à la figure 2.1.

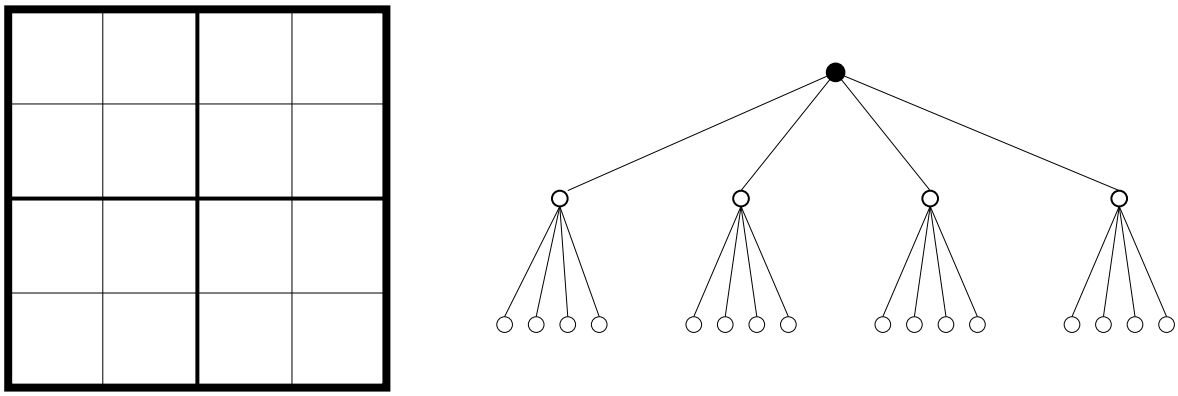


FIG. 2.1 – Un quadtree complet à 3 niveaux.

Pour les octrees en $3D$, le principe est le même avec une dimension supplémentaire : on subdivise la cellule du père en 8 octants de même taille.

Au final, on obtient donc bien un arbre, manipulé au travers des liens « père - fils » habituels. Un nœud qui ne possède pas de fils est appelé ***feuille***, et on désigne par ***nœud interne***, ou ***cellule interne***, tout nœud qui n'est pas une feuille. En pratique les particules sont liées aux feuilles. Lorsqu'on parle des particules d'une cellule interne de l'octree, on fait référence à l'union des particules possédées par ses descendants au niveau des feuilles. Par ailleurs, on considère que la racine de l'octree est au niveau 0. La ***hauteur*** de l'octree est alors définie comme son niveau maximum : un octree constitué uniquement de sa racine aura donc une hauteur 0, tout comme un octree vide.

2.3.1.1 Construction

La construction de ces octrees peut être faite de deux manières :

1. Dans le premier cas, la hauteur de l'octree H est établie au préalable par l'utilisateur : elle va correspondre au degré de raffinement souhaité. Un arbre complet, d'arité 4 ou 8, et de profondeur H est alors construit et on y « insère les particules » une par une par la racine : à chaque nœud interne rencontré, on attribue la particule au fils correspondant à ses coordonnées, et on poursuit jusqu'à atteindre une feuille. L'octree ainsi construit est bien adapté aux distributions uniformes de particules.
2. Dans le second cas l'utilisateur fixe le nombre maximal s_{max} de particules par feuille. L'octree est initialement constitué uniquement de sa racine qui est alors une feuille.

On insère ensuite les particules une par une au niveau de la racine, et pour chaque cellule interne rencontrée, on attribue la particule au fils la contenant jusqu'à atteindre la feuille correspondant à ses coordonnées. Si cette feuille contient alors plus de s_{max} particules, elle est subdivisée en 4 (ou 8) feuilles et elle devient elle-même une cellule interne, ses particules étant distribuées parmi ses fils. Après que toutes les particules ont été insérées, les feuilles vides sont éliminées et le résultat obtenu est donc un arbre d'arité 4 ou 8, mais déséquilibré, avec des feuilles présentes à des niveaux différents. Ce type d'octree est a priori plus adapté à des distributions non uniformes de particules, mais nous reviendrons sur ce point en section 2.3.3.9.

Dans le cas uniforme, on remarque que la construction de l'octree s'effectue en $\mathcal{O}(N \log N)$, $\mathcal{O}(\log N)$ repésentant le coût moyen d'une descente dans l'octree pour chaque particule (la hauteur de l'octree complet obtenu dans ce cas est en effet de l'ordre de $\mathcal{O}(\log N)$). Dans le cas non uniforme, on obtient le même résultat si la hauteur de l'octree est encore en $\mathcal{O}(\log N)$. Cependant, la construction de l'arbre peut être considérée comme une étape initiale qui n'a besoin d'être effectuée qu'une seule fois. Le pas de temps des simulations étant réduit par souci de stabilité, le déplacement des particules est relativement faible entre deux itérations, et seul un petit nombre de particules changent donc de feuille entre deux pas de temps consécutifs. Il est alors préférable de mettre à jour l'octree plutôt que de le reconstruire à chaque nouveau pas de temps.

2.3.1.2 Implémentation

En ce qui concerne la structure de données implémentant ces octrees, il y a là encore deux possibilités dues au caractère uniforme ou non de la distribution. Avant de présenter ces deux structures de données, nous insistons sur la nécessité de pouvoir accéder rapidement au contenu de n'importe quelle cellule à partir des coordonnées spatiales du centre de la cellule. Ceci est surtout essentiel pour une implémentation efficace de la FMM, et est obtenu dans ce cas grâce à l'ordre de Morton [102] [121] dont un exemple est donné en figure 2.2 : les coordonnées du centre de chaque cellule du niveau l de l'octree variant de 0 à $2^l - 1$, les bits de ces coordonnées sont entrelacés afin de former l'indice de Morton de la cellule. Toutes les cellules de l'octree sont alors indexées suivant cet ordre, et des opérations arithmétiques sur les bits de ces indices permettent alors d'effectuer rapidement tous les accès requis par l'algorithme de la FMM, comme détaillé en section 2.3.3.6.

Tableaux unidimensionnels de pointeurs. Pour des distributions uniformes de particules, ou pour des distributions présentant de larges zones uniformes, l'ordre de Morton est mis en place grâce à des tableaux unidimensionnels : à chaque niveau l de l'octree, un tableau de 8^l pointeurs est utilisé pour stocker l'adresse de chaque cellule non vide (une cellule vide correspondant à l'adresse *NULL*). L'indice de Morton d'une cellule correspond alors à l'indice de son pointeur dans le tableau. Ces tableaux sont par exemple utilisés dans le code DPMTA ([51] [102], voir aussi en section 2.3.3).

Pour de telles distributions uniformes, la hauteur de l'octree complet varie de 4 à 8 en pratique, alors que des distributions non uniformes peuvent nécessiter des octrees déséquilibrés de hauteur supérieure à 10. Dans ce cas, les besoins mémoire des tableaux unidimensionnels, contenant tous les indices à chaque niveau, deviennent rapidement prohibitifs alors que la plupart des cellules sont vides.

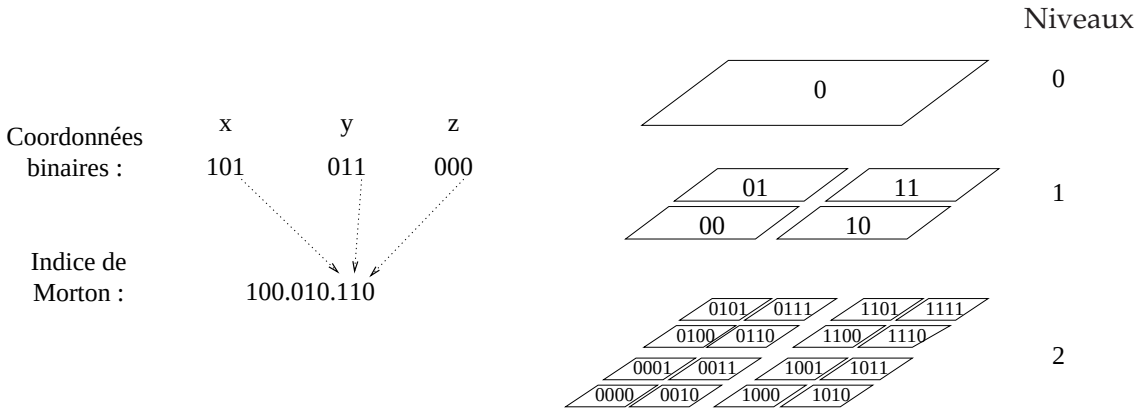


FIG. 2.2 – Ordre de Morton : entrelacement des bits des coordonnées (figure de gauche) et exemple d'un quadtree de hauteur 2 (figure de droite).

Hashed octree. Le *hashed octree* proposé par Warren & Salmon dans [121] permet de stocker de tels octrees déséquilibrés, avec une hauteur pouvant atteindre (théoriquement) 20, tout en conservant l'ordre de Morton. Cette structure de données correspond en fait à une table de hachage dont les clés correspondent aux h bits les moins significatifs de l'indice de Morton. Lorsque plusieurs indices partagent la même clé, ces indices (ainsi que les pointeurs vers les cellules correspondantes) sont stockés dans une liste chaînée ; afin d'accélérer les temps d'accès, les éléments récemment accédés sont déplacés en tête de liste [122].

Cependant dans le cas uniforme, l'efficacité du *hashed octree* décroît pour des hauteurs croissantes de l'octree : chaque niveau supplémentaire apporte de nouvelles cellules dont les clés vont créer de nouveaux conflits dans la table de hachage, augmentant ainsi les temps d'accès. Afin de justifier cette affirmation, nous considérons le niveau l d'un octree construit à partir d'une distribution uniforme de particules, avec $l > \lceil \frac{h}{3} \rceil$, et nous notons $c(l)$ le nombre de cellules au niveau l qui partagent une même clé dans la table de hachage : $c(l)$ a la même valeur pour toutes les clés puisque la distribution est uniforme. Au niveau $l+1$, nous avons : $c(l+1) = 8.c(l)$, car tous les fils partagent la clé de leur père, ce qui signifie que $9.c(l)$ cellules, sur l'ensemble des niveaux l et $l+1$, partagent la même clé. Par conséquent le nombre de collisions possibles croît de façon exponentielle avec la hauteur de l'octree, et le temps d'accès au contenu de chaque cellule à travers la table de hachage se dégrade donc de la même façon. Ce problème n'apparaissait pas avec les distributions gravitationnelles étudiées par Warren & Salmon [121] [122] car la majeure partie des indices de Morton correspondaient dans ce cas à des cellules vides qui n'étaient pas stockées dans le *hashed octree*.

Aucune de ces deux structures de données n'est donc adaptée pour traiter efficacement aussi bien des distributions uniformes que des distributions non uniformes de particules. D'autres structures de données, intrinsèquement liées à la version de la FMM dédiée aux distributions non uniformes de particules, seront présentées en section 2.3.3.9.

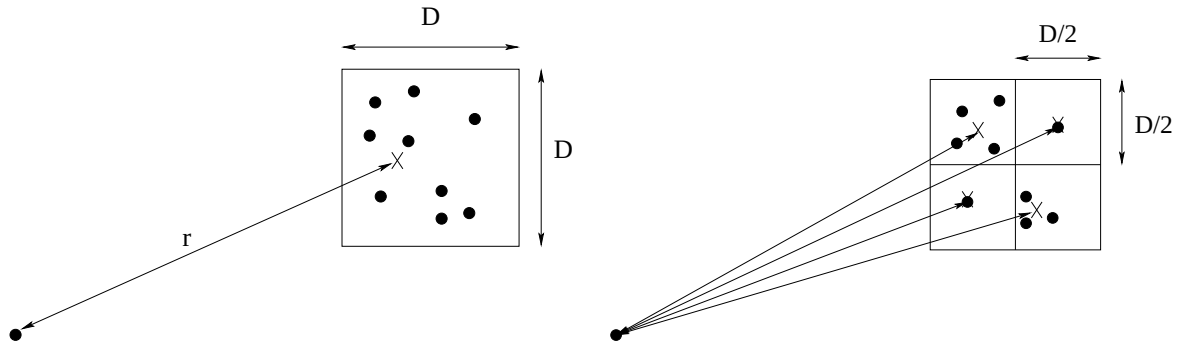
2.3.2 L'algorithme de Barnes & Hut

Dans l'algorithme de Barnes & Hut, la validation de l'approximation est basée sur un *critère d'acceptation* dont le paramètre θ est fixé par l'utilisateur. Nous nous plaçons ici dans le cas de simulations gravitationnelles, mais la méthode peut aussi s'appliquer aux simulations

moléculaires en remplaçant les masses par les charges. En notant D la taille du côté de la cellule, et r la distance du point étudié au centre de masse des particules présentes dans la cellule (voir figure 2.3(a)), l'approximation est acceptée si

$$\frac{D}{r} < \theta.$$

Plus la valeur de θ est faible, et plus il est difficile d'utiliser des approximations pour des cellules de grosse taille, ou pour des cellules proches de la particule courante : l'algorithme devient alors plus précis, mais aussi plus coûteux. En pratique (en astrophysique par exemple) la valeur de θ est comprise entre $0,5$ et $0,9$. La valeur $\theta = 0,7$ est couramment utilisée.



(a) Au niveau d'une cellule donnée.

(b) Au niveau de ses fils.

FIG. 2.3 – Critère d'acceptation de l'algorithme de Barnes & Hut.

A l'origine, l'approximation considérée consiste à remplacer le potentiel créé par chacune des particules de la cellule par le potentiel créé par une particule fictive dont la masse est égale à la somme des masses des particules, et qui est placée au centre de masse des particules. De nouveaux critères d'acceptation sont ensuite apparus [103] [121] où les termes (ou moments) d'ordre supérieur sont pris en compte en fonction de la distribution des particules au sein de la cellule. En effet, lorsqu'on décompose la force exercée par un ensemble de particules en puissance de $\frac{1}{r^{n+2}}$, où r est la distance au centre de masse, le premier terme en $\frac{1}{r^2}$ correspond à la somme des masses au centre de masse (monopôle), et le second terme (dipôle, $n = 1$) disparaît par définition du centre de masse. Mais il reste tous les termes d'ordre supérieur qui sont sensibles à la distribution des particules au sein de la cellule : alors que pour une précision donnée le critère initial considère à chaque fois la plus mauvaise distribution en omettant ces termes, ces nouveaux critères permettent d'accepter des cellules supplémentaires sans mettre en cause la précision souhaitée, et donc d'améliorer la performance de l'algorithme. De tels critères d'acceptation sont appelés *Multipole Acceptance Criterion*, ou MAC [43] [103] [114], en référence aux développements multipôles utilisés. De plus l'utilisation de ces termes d'ordre supérieur permet bien sûr d'obtenir des approximations plus précises qu'avec le seul monopôle : en pratique les développements sont généralement tronqués au terme quadripolaire ($n = 2$).

L'autre particularité de la méthode de Barnes & Hut est qu'elle met en œuvre des interactions de type *particule - cellule*. Le principe (récursif) de la méthode est en effet de faire « entrer » les particules une par une dans l'octree par sa racine : si le critère d'acceptation entre la particule courante p et la cellule source considérée est validé, alors l'approximation

est acceptée pour calculer la force subite en p due aux particules présentes dans la cellule source. Sinon, on relance le critère d'acceptation sur chacun des fils de la cellule source, qui ont un côté de taille $D/2$, et qui sont donc susceptibles de satisfaire le critère (voir figure 2.3(b)). Lorsque le critère d'acceptation échoue sur une feuille, on utilise le calcul direct pour évaluer les interactions entre la particule courante et toutes les particules sources contenues dans cette feuille.

En supposant qu'en pratique la hauteur de l'octree est toujours en $\mathcal{O}(\log N)$ (N désignant le nombre de particules), la descente de l'octree ainsi réalisée s'effectue en $\mathcal{O}(\log N)$. On obtient alors une complexité globale en $\mathcal{O}(N \log N)$ à chaque pas de temps. Il a aussi été montré, dans [23] [103], que le nombre de cellules sources interagissant avec une particule cible donnée varie suivant $\mathcal{O}(\theta^{-3})$. Cependant, aucune borne d'erreur précise n'est disponible sur le plan théorique, et le contrôle de l'erreur via θ reste empirique.

Outre des MAC plus élaborés, une autre amélioration a ensuite été apportée à l'algorithme de Barnes & Hut. On remarque en effet que le champ lointain diffère peu entre deux particules proches : il est donc tentant de mettre en commun les approximations du champ lointain à l'intérieur d'un groupe de particules. Ceci a été introduit par Barnes [19] pour une exécution sur des processeurs vectoriels, puis approfondi par Warren & Salmon [122] grâce à des séries de Taylor qui débouchent sur un parcours doublement récursif de l'octree.

En plus du *treecode* qui est le nom du code original de Barnes [18], la méthode a aussi été couplée avec l'implémentation matérielle GRAPE du calcul direct (voir section 2.2.1), le résultat s'avérant très efficace [15] [43]. De plus, comme annoncé en section 2.2.5, le code GADGET-2 [114] combine l'algorithme de Barnes & Hut, doté d'un MAC amélioré, avec un algorithme de type Particle-Mesh pour former une méthode TreePM.

Le caractère intrinséquement adaptatif de cette méthode, et la simplicité des développements utilisés, rendent cet algorithme et ses variantes particulièrement efficaces pour des distributions hautement non uniformes nécessitant des basses précisions. C'est pourquoi elles ont été beaucoup utilisées en astrophysique.

Cependant, dans le cas de potentiels électrostatiques, la nullité du second terme de ces développements basés sur le centre de masse n'est bien sûr plus valable (car le potentiel est alors généré par les charges et non plus par les masses), et contrairement aux simulations gravitationnelles où le premier terme prédomine, ce premier terme est ici presque nul pour des cellules quasiment neutres électriquement [23] [26]. A précision égale, il faut alors inclure plus de termes dans les développements. Les simulations en dynamique moléculaire requièrent de plus davantage de précision qu'en astrophysique ; or les développements utilisés ici ne présentent pas d'expression analytique en fonction de leur ordre. Il n'est donc pas aisé en pratique d'inclure des termes d'ordre supérieur à 3 ou 4 (voir par exemple [103], ou [122]), et donc d'utiliser ces développements en dynamique moléculaire.

2.3.3 La « Fast Multipole Method » (FMM)

La méthode dite *Fast Multipole Method* (FMM) résout le problème à N-corps en $\mathcal{O}(N)$, et ce pour n'importe quelle précision souhaitée. Elle a d'abord été développée par Greengard & Rokhlin pour des distributions uniformes en deux dimensions [65] (FMM uniforme), puis étendue aux distributions non uniformes [32] (FMM adaptative), et aux simulations en trois dimensions [67] grâce à de nouvelles bases mathématiques. Ces articles sont rassemblés dans [60].

Initialement élaborée pour des potentiels électrostatiques ou gravitationnels, la FMM a ensuite été étendue à une grande partie des problèmes à N-corps présentés en section 2.1 : le potentiel de Lennard-Jones [44] [51], les équations de Maxwell [39] et de Helmholtz [52] [64], les calculs de capacitance [94] [93], la modélisation d'objets $3D$ [31], le potentiel de Yukawa [63], *etc.*. Le principe de la FMM a aussi été appliqué à d'autres domaines, comme les problèmes d'élasticité linéaire [55], ou l'établissement d'une Transformée Rapide de Gauss [69].

Parmi les nombreuses implémentations de la FMM, nous étudierons plus particulièrement le code DPMTA (*Distributed Parallel Multipole Tree Algorithm*, version 3.1.2p3) développé à l'Université de Duke [51] [102]. Ce code correspond plutôt à une FMM uniforme et est destiné aux simulations en dynamique moléculaire.

2.3.3.1 Principe

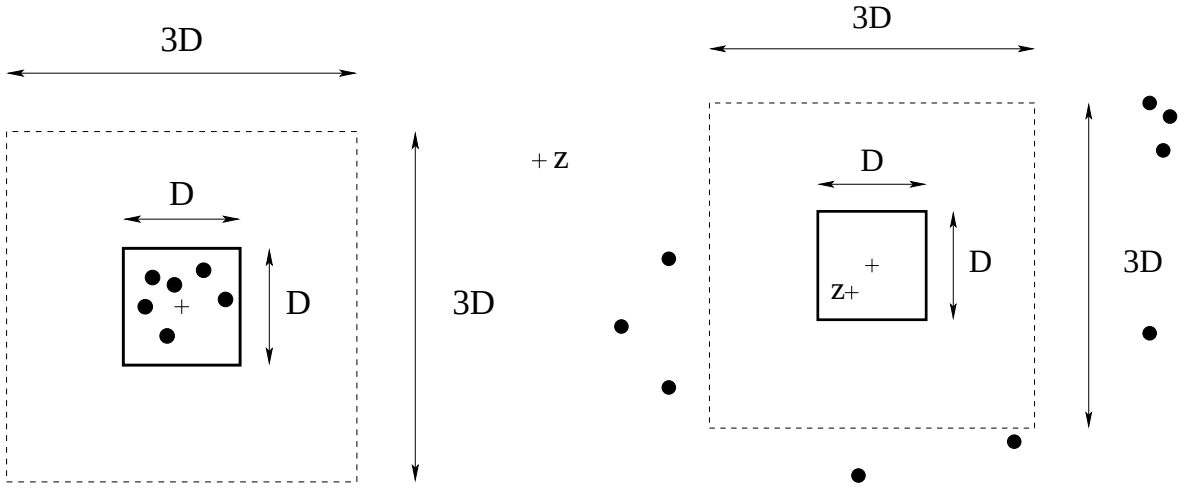
Avant de détailler complètement la FMM, nous en présentons ici les principales caractéristiques.

La première différence par rapport à l'algorithme de Barnes & Hut est que les interactions mises en jeu sont maintenant de type *cellule - cellule* : l'approximation du champ lointain n'est plus valable pour une particule donnée mais pour l'ensemble de la cellule cible. Ceci est rendu possible grâce à l'utilisation d'un deuxième type de développements : *les développements locaux*. Contrairement aux *développements multipôles*, qui représentent un champ dû aux particules à l'intérieur d'une cellule c, en un point suffisament éloigné de c, les développements locaux représentent le champ en un point situé à l'intérieur de la cellule, ce champ étant dû à des particules suffisament éloignées de la cellule : voir figure 2.4. Le gain en complexité par rapport à l'algorithme de Barnes & Hut vient essentiellement de l'introduction de ces développements locaux.

Le principe du calcul du champ lointain à l'aide de ces deux types de développements est donné en figure 2.5 : le développement multipôle est construit au niveau des feuilles à partir des particules sources, et éventuellement translaté aux niveaux internes. Ce développement multipôle est ensuite convertit en un développement local, qui est éventuellement translaté jusqu'au niveau des feuilles, où il est finalement évalué en chaque point cible.

Les développements multipôles, comme les développements locaux, sont basés sur le centre de chaque cellule, et non plus sur le centre de masse des particules de la cellule (dans le cas gravitationnel). Ceci implique que leur validité et leur convergence ne dépendent que de la géométrie des cellules et de la distance entre deux cellules, et non plus de leur contenu. La FMM n'utilise donc pas de critère d'acceptation semblable à ceux employés dans l'algorithme de Barnes & Hut et dans ses variantes postérieures. Dans la version originale de la FMM, l'acceptation de l'approximation est déterminée de manière beaucoup plus rigide : en considérant des cellules de même taille, c'est-à-dire en se plaçant à un niveau donné de l'octree, l'approximation de la force exercée par une cellule sur une autre n'est valable que s'il y a au moins une troisième cellule séparant les deux premières. En d'autres termes, et comme représenté en figure 2.6, le calcul des interactions entre deux cellules ayant un sommet, une arête, ou une face en commun, doit être effectué de manière directe, c'est-à-dire sans approximation. Nous verrons par la suite, en section 2.3.3.10, que ce cadre rigide peut néanmoins être paramétré.

La dernière caractéristique essentielle de la FMM concerne l'erreur engendrée. Contrairement à l'algorithme de Barnes & Hut, il existe une borne d'erreur théorique qui permet



(a) Développement multipôle : le développement multipôle, dû aux particules situées dans la cellule de côté D, est considéré comme valable en tout point z situé hors de la cellule concentrique de côté $3D$.

(b) Développement local : le développement local, dû aux particules situées hors de la cellule de côté $3D$, est considéré comme valable en tout point z situé à l'intérieur de la cellule concentrique de côté D.

FIG. 2.4 – Validité des développements multipôles et locaux pour une cellule de côté D. Le rapport de 3, entre le côté de la cellule interne et celui de la cellule externe, correspond au cas utilisé dans la FMM.

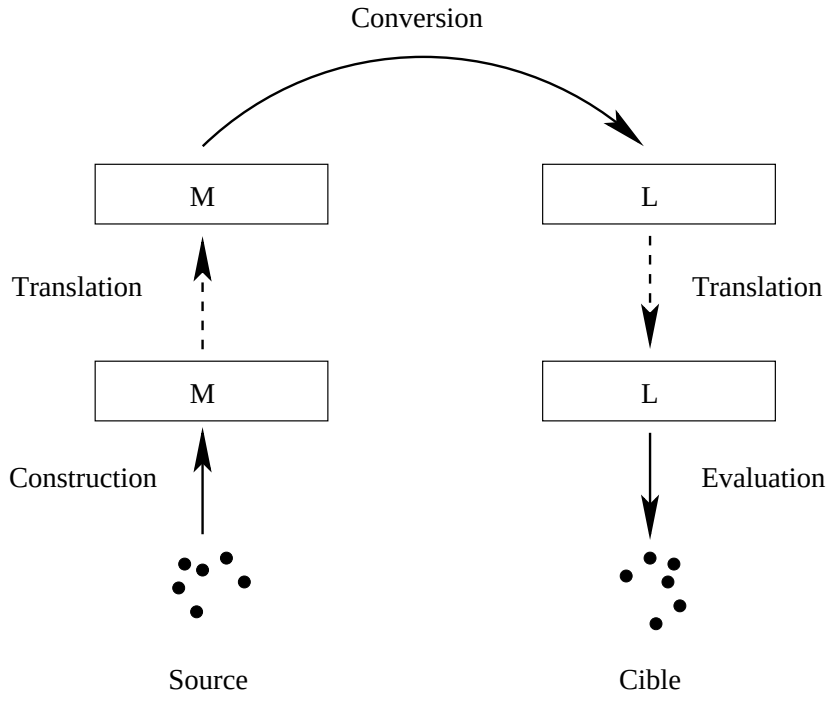


FIG. 2.5 – Principe du calcul du champ lointain avec un développement multipôle et un développement local.

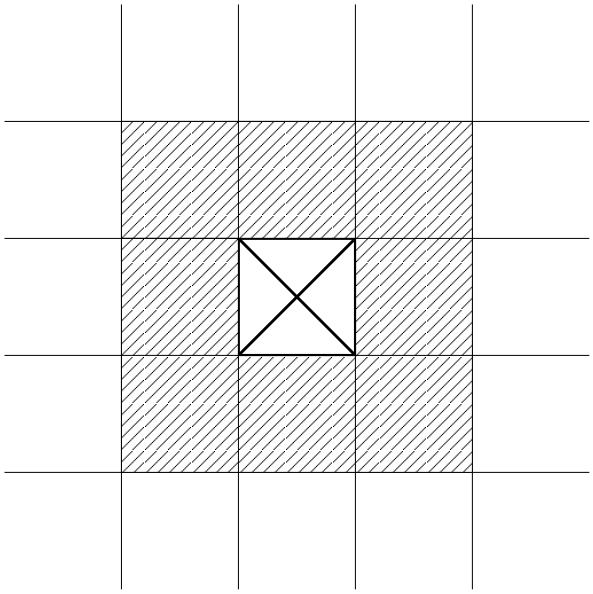


FIG. 2.6 – Validité des approximations dans la FMM. Les interactions entre la cellule marquée d'une croix et les cellules rayées sont obtenus par calcul direct. Les interactions entre la cellule marquée d'une croix et les cellules situées au-delà sont évaluées de façon approchée.

de garantir la précision obtenue en pratique. Cette borne d'erreur est directement paramétrée par le nombre (non limité) de termes employés dans les développements multipôles et locaux. Plus le nombre de termes utilisé sera grand et plus la précision du calcul sera importante : en théorie il n'y a donc pas de limite à la précision souhaitée, exceptée celle due à la précision machine. Cependant la complexité linéaire affichée par la FMM masque un important facteur constant (« constant » par rapport à N) dont le coût « explose » lorsque le nombre de termes utilisés augmente.

Nous allons désormais présenter en détails la FMM. Dans cette présentation, nous considérerons que chacun des N corps est à la fois *source* et *cible* : il est cependant possible de modifier l'algorithme pour utiliser des points *cibles*, où le potentiel est évalué, distincts des particules *sources*, génératrices de ce potentiel (voir par exemple [115]).

Nous commencerons par les bases mathématiques de la méthode en section 2.3.3.2, et les bornes d'erreur associées (section 2.3.3.3), ainsi que par une distinction entre deux expressions différentes pour l'opérateur multipôle-vers-local (section 2.3.3.4). Puis nous présenterons l'algorithme de la FMM uniforme, dans le cas de systèmes isolés, et sa complexité théorique en section 2.3.3.5. Après quelques précisions sur l'implémentation de la FMM (section 2.3.3.6), nous verrons en section 2.3.3.7 comment introduire les conditions aux limites périodiques, et en section 2.3.3.8 nous présenterons une implémentation originale d'une FMM « sans multipôles ». L'adaptation non triviale de la FMM aux distributions non uniformes a été conçue de plusieurs manières : ces différentes FMM adaptatives seront exposées en section 2.3.3.9. La section 2.3.3.10 détaillera ensuite les différentes façons de définir les zones où les développements sont acceptés. Puis nous étudierons les diverses parallélisations mises en œuvre pour la FMM en section 2.3.3.11, et nous présenterons en section 2.3.3.12 les différentes améliorations apportées à la FMM afin de réduire l'impact du nombre de termes

utilisés sur le coût global de la méthode. Enfin la section 2.3.3.13 sera consacrée à une méthode hybride combinant des caractéristiques de la FMM et de l'algorithme de Barnes & Hut.

2.3.3.2 Formules

Nous considérons ici un potentiel de Green de l'équation de Laplace (sans terme source). Toute fonction deux fois continûment dérivable vérifiant cette équation est appelée fonction harmonique. Plus précisément, nous allons considérer par la suite soit un potentiel gravitationnel, correspondant à des interactions régies par la loi de la gravitation universelle de Newton, soit un potentiel électrostatique, correspondant à des interactions régies par la loi de Coulomb. La constante de gravitation universelle, ou la permittivité diélectrique du vide suivant le cas, sera généralement omise, de même que la différence de signe entre le potentiel électrostatique et le potentiel gravitationnel. Et comme dans [60], nous nous plaçons par défaut dans le cas d'un potentiel électrostatique pour la description des formules.

Dans le cas $2D$, en identifiant $\mathbb{R}^2$ et $\mathbb{C}$, le potentiel en z dû à une charge q en z_0 s'exprime donc ainsi

$$\Phi(z) = -q \log \left(\| z - z_0 \| \right).$$

Les formules de la FMM pour le calcul du potentiel sont alors obtenues grâce à l'existence, pour toute fonction harmonique de deux variables à valeurs réelles, d'une fonction analytique de deux variables à valeurs complexes, dont la partie réelle est égale à la fonction harmonique. On utilise aussi le développement en série entière de la fonction analytique $\log(1-z)$, $\forall z \in C / |z| < 1$, et la formule du binôme de Newton (généralisée aux puissances négatives). Les formules complètes, ainsi que leurs preuves, sont détaillées par Greengard & Rokhlin dans [65]. Quant au vecteur force, il est déduit de la dérivée Φ' du potentiel complexe Φ grâce aux conditions de Cauchy-Riemann par :

$$\mathbf{F}(z) = -\mathbf{grad}(Re(\Phi(z))) = (-Re(\Phi'(z)), Im(\Phi'(z))),$$

où Re et Im désignent respectivement la partie réelle et la partie imaginaire d'un nombre complexe.

En $3D$, le potentiel dû à une charge q en z_0 s'écrit

$$\Phi(r) = \frac{q}{r}.$$

Mais l'utilisation de fonctions analytiques complexes n'est pas extensible à un espace en trois dimensions. Dans ce cas, le potentiel sera décomposé en utilisant les polynômes de Legendre et les harmoniques sphériques. Ces bases mathématiques plus complexes qu'en deux dimensions ont aussi un impact sur le nombre d'opérations de l'algorithme. En effet, en notant P le degré maximum utilisé dans les développements, la taille des développements varie en $\mathcal{O}(P)$ en $2D$ et les opérations requises sur ces développements s'effectuent en $\mathcal{O}(P^2)$. Ce nombre d'opérations représente alors le facteur multiplicatif inclus dans la complexité en $\mathcal{O}(N)$ de la FMM. Par contre, en $3D$, la taille des développements varie en $\mathcal{O}(P^2)$, et les opérations nécessaires s'effectuent en $\mathcal{O}(P^4)$, ce qui accroît sensiblement la complexité théorique de la FMM en trois dimensions. Nous reviendrons par la suite sur cette complexité

en $\mathcal{O}(P^4)$. Contentons nous de remarquer pour l'instant que la version $3D$ de la FMM est par conséquent moins efficace que celle en $2D$. En d'autres termes, le point de *cross-over*, qui est le nombre de particules pour lequel l'algorithme étudié supplente le calcul direct, se trouve être beaucoup plus élevé en trois dimensions qu'en deux dimensions [121].

Pour illustrer les valeurs de P possibles, nous rappelons qu'en astrophysique de faibles précisions sont suffisantes : en pratique, pour obtenir une erreur relative comprise entre 10^{-2} et 10^{-3} [30],P sera compris entre 1 et 3. En revanche en dynamique moléculaire, de plus importantes précisions sont requises (entre 10^{-5} et 10^{-7}), et P pourra alors varier de 3 à 15 (voir [51] ou [102]). A titre complémentaire, pour d'autres potentiels (et donc pour d'autres formes de développements), le degré (ou ordre) maximum des développements peut être très faible (entre 0 et 6 en calcul de capacitance [94] [93]) ou très grand (jusqu'à plusieurs dizaines en électromagnétisme [117]). Une autre particularité de l'application de la FMM à l'électromagnétisme est que, contrairement aux cas des potentiels électrostatiques et gravitationnels, ce degré maximum peut varier d'un niveau à l'autre de l'octree.

Nous allons désormais détailler ces formules dans le cas d'un potentiel électrostatique ou gravitationnel, et d'un espace en dimensions trois. Si dès 1987, Zhao a mis au point des formules pour la FMM en trois dimensions en utilisant les coordonnées cartésiennes [129], c'est la version en coordonnées sphériques mise au point par Greengard et Rokhlin en 1988 [67] qui a été très largement retenue (à quelques exceptions près, comme par exemple [108]).

Cependant comme l'opérateur *M2L*, qui effectue la conversion d'un développement multipôle en un développement local, représente la majeure partie du temps de calcul consacré au champ lointain (voir section 2.3.3.5), plusieurs modifications dans les formules originales ont été introduites afin de simplifier, et donc d'accélérer, cet opérateur *M2L*. Nous pouvons citer ici les travaux d'Epton & Dembart [52], White & Head-Gordon [124], et ceux réalisés à l'Université de Duke (Caroline Du Nord, USA), voir par exemple [51]. Les formules choisies doivent de plus vérifier une condition importante, à savoir la propriété de symétrie entre les termes d'ordres opposés dans les développements multipôles et locaux (voir les équations (2.11) et (2.13) ci-après) : celle-ci permet en effet de ne calculer que les termes d'ordre positif. Au final, les formules retenues sont celles présentées par Epton & Dembart [52], principalement à cause de la clarté des théorèmes sous-jacents et de leurs preuves.

Les preuves des théorèmes et équations présentées dans cette section ont donc été établies par Epton & Dembart [52], et Greengard & Rokhlin ([60] par exemple). En ce qui concerne les bases de la théorie du potentiel, nous renvoyons le lecteur aux références citées dans [60].

Afin de construire l'agorithme de la FMM, nous avons besoin de plusieurs opérateurs agissant sur les développements multipôles et locaux. Ces opérateurs permettent de construire, translater, convertir et évaluer ces développements. Plus précisément, une version uniforme de la FMM requiert les opérateurs suivants :

- l'opérateur *P2M* qui construit le développement multipôle d'une feuille à partir des particules qu'elle contient,
- l'opérateur *M2M* qui translate le centre d'un développement multipôle,
- l'opérateur *M2L* qui convertit un développement multipôle en un développement local,
- l'opérateur *L2L* qui translate le centre d'un développement local,
- et l'opérateur *L2P* qui évalue un développement local en chacune des particules d'une cellule. Nous considérons ici que le potentiel et la force sont évalués, mais il est aussi

possible de n'évaluer que l'un des deux.

Pour la version adaptative de la FMM, nous avons aussi besoin des opérateurs suivants :

- l'opérateur *M2P* qui évalue le potentiel et la force dus à un développement multipôle en chacune des particules d'une cellule,
- et l'opérateur *P2L* qui construit le développement local dû aux particules d'une cellule.

Par souci de commodité, nous définissons aussi l'opérateur *P2P* qui effectue le calcul direct entre toutes les particules de deux cellules. Cet opérateur pourra aussi être appelé sur une seule cellule pour traiter, toujours par calcul direct, toutes les interactions au sein de cette cellule. Par défaut, le principe des interactions réciproques est utilisé (voir section 2.1).

Nous renvoyons le lecteur à l'annexe A pour la définition des polynômes de Legendre P_n, des fonctions associées de Legendre P_l^m et des harmoniques sphériques Y_l^m, et nous présentons uniquement ici deux résultats essentiels pour la FMM.

Nous avons tout d'abord la décomposition suivante (voir par exemple [52]), $\mathbf{X} = (r, \theta, \phi)$ et $\mathbf{X}' = (r', \theta', \phi')$ désignant deux vecteurs de $\mathbb{R}^3$:

$$\frac{1}{\|\mathbf{X} - \mathbf{X}'\|} = \sum_{n=0}^{+\infty} \frac{r_<^n}{r_>^{n+1}} P_n(\cos\gamma), \tag{2.3}$$

où $r_>$ (respectivement $r_<$) désigne le maximum (respectivement le minimum) entre r et r', et γ l'angle entre les vecteurs $\mathbf{X}$ et $\mathbf{X}'$ (défini par $\cos\gamma = \cos\theta\cos\theta' + \sin\theta\sin\theta'\cos(\phi - \phi')$). La fraction $\frac{1}{\|\mathbf{X}-\mathbf{X}'\|}$ est bien sûr à rapprocher du potentiel en $\mathbf{X}$ dû à une charge unitaire en $\mathbf{X}'$.

Le deuxième résultat essentiel est le théorème d'addition des harmoniques sphériques (voir [60]) :

Théorème 1 (Théorème d'addition des harmoniques sphériques)**.** *Soient 2 vecteurs* $\mathbf{P_1} = (r1, \theta1, \phi1)$ *et* $\mathbf{P_2} = (r2, \theta2, \phi2)$*, on note* γ *l'angle entre les* 2 *vecteurs défini par* $\cos\gamma = \cos\theta_1\cos\theta_2 + \sin\theta_1\sin\theta_2\cos(\phi_1 - \phi_2)$*, et on a selon notre définition des harmoniques sphériques :*

$$P_l(\cos\gamma) = \sum_{m=-l}^{l} Y_l^m(\theta_1, \phi_1) Y_l^{-m}(\theta_2, \phi_2).$$

Afin de décrire précisément tous les opérateurs de la FMM, nous commençons par définir, comme Epton & Dembart [52], les fonctions *Outer* et *Inner* par respectivement :

$$O_l^m(r, \theta, \phi) = \frac{(-1)^l i^{|m|}}{A_l^m} Y_l^m(\theta, \phi) \frac{1}{r^{l+1}} \qquad \forall\, (l, m) \in \mathbb{N} \times \mathbb{Z} \quad \text{avec} \quad 0 \leq |m| \leq l, \tag{2.4}$$

et :

$$I_l^m(r, \theta, \phi) = i^{-|m|} A_l^m Y_l^m(\theta, \phi) r^l \qquad \forall\, (l, m) \in \mathbb{N} \times \mathbb{Z} \quad \text{avec} \quad 0 \leq |m| \leq l, \tag{2.5}$$

où :

$$A_l^m = \frac{(-1)^l}{\sqrt{(l-m)!(l+m)!}}.$$

Comme pour les harmoniques sphériques, nous soulignons que les fonctions O_j^k et I_j^k sont considérées comme identiquement nulles pour $j < 0$ ou $|k| > j$. Comme leur nom le

suggère, les fonctions *Outer* correspondent aux développements multipôles et les fonctions *Inner* aux développements locaux. Elles peuvent de plus aussi s'écrire :

$$O_l^m(r,\theta,\phi) = i^{-|m|}(l-|m|)!P_l^{|m|}(\cos\theta)e^{i.m.\phi}\frac{1}{r^{l+1}}, \tag{2.6}$$

$$I_l^m(r,\theta,\phi) = \frac{(-1)^l i^{|m|}}{(l+|m|)!}P_l^{|m|}(\cos\theta)e^{i.m.\phi}r^l. \tag{2.7}$$

Les relations de symétrie entre des ordres opposés (voir l'équation (A.3) en annexe A) deviennent ici :

$$O_l^{-m} = (-1)^m\overline{O}_l^m, \tag{2.8}$$

$$I_l^{-m} = (-1)^m\overline{I}_l^m, \tag{2.9}$$

où $\overline{z}$ désigne le conjugué complexe de $z \in \mathbb{C}$.

Soient $\mathbf{X}$ et $\mathbf{X}'$ deux vecteurs dans l'espace à trois dimensions, nous donnons maintenant les principaux théorèmes sur lesquels se base la FMM ; leur preuve est donnée dans [52].

Théorème 2 (Théorème de translation classique)**.** *Sous l'hypothèse* $\|\mathbf{X}\| > \|\mathbf{X}'\|$*, nous avons :*

$$\frac{1}{\|\mathbf{X}-\mathbf{X}'\|} = \sum_{n=0}^{+\infty}\sum_{l=-n}^{n}(-1)^n I_n^{-l}(\mathbf{X}')O_n^l(\mathbf{X}).$$

Idée de la preuve. Cette équation est obtenue à partir de la décomposition de $\frac{1}{\|\mathbf{X}-\mathbf{X}'\|}$ en polynômes de Legendre présentée à l'équation (2.3). Les polynômes de Legendre sont ensuite eux-mêmes décomposés grâce au théorème d'addition des harmoniques sphériques (théorème 1). □

Le théorème suivant permet d'établir les opérateurs *M2M* (ou *Outer-to-Outer*) et *M2L* (ou *Outer-to-Inner*).

Théorème 3 (Théorème de translation de Laplace pour *Outer-to-Outer* et *Outer-to-Inner*)**.** *Avec l'hypothèse* $\|\mathbf{X}\| > \|\mathbf{X}'\|$*, nous avons :*

$$O_n^l(\mathbf{X}-\mathbf{X}') = \sum_{j=0}^{+\infty}\sum_{k=-j}^{j}(-1)^j I_j^{-k}(\mathbf{X}')O_{n+j}^{l+k}(\mathbf{X}) \qquad \forall\,(n,l)\in\mathbb{N}\times\mathbb{Z} \quad \textit{avec} \quad 0\le|l|\le n.$$

Le dernier théorème correspond au *Troisième Théorème d'Addition* présenté dans [60], et il est utilisé pour l'opérateur *L2L* (ou *Inner-to-Inner*).

Théorème 4 (Théorème de translation de Laplace pour *Inner-to-Inner*)**.**

$$I_n^l(\mathbf{X}-\mathbf{X}') = \sum_{j=0}^{n}\sum_{k=-j}^{j}(-1)^j I_j^k(\mathbf{X}')I_{n-j}^{l-k}(\mathbf{X}) \qquad \forall\,(n,l)\in\mathbb{N}\times\mathbb{Z} \quad \textit{avec} \quad 0\le|l|\le n.$$

Grâce à ces théorèmes, nous allons maintenant donner l'expression des opérateurs *P2L*, *M2M*, *M2L*, *L2L* et *L2P* qui sont utilisés par l'algorithme de la FMM pour des distributions uniformes. La configuration utilisée est présentée en figure 2.7.

FIG. 2.7 – Cas étudié pour la présentation des opérateurs de la FMM. Les charges, représentées par •, sont toutes situées dans la sphère de centre $\mathbf{Z_0}$ et de rayon R_0.

Définition 2.1 (Opérateur *P2M*)**.** *Soient m charges $(q_i)_{i\in[\![1,m]\!]}$ localisées en $\mathbf{Q_i} = (\rho_i, \alpha_i, \beta_i)$ et dont les coordonnées relatives par rapport à un centre $\mathbf{Z_0}$ sont : $\mathbf{Q_i} - \mathbf{Z_0} = (\rho'_i, \alpha'_i, \beta'_i)$, nous définissons :*

$$M_j^k = (-1)^j \sum_{i=1}^{m} q_i I_j^k(\rho'_i, \alpha'_i, \beta'_i) \qquad \forall\,(j,k) \in \mathbb{N}\times\mathbb{Z} \quad \textit{avec} \quad 0 \le |k| \le j.$$

Le potentiel est alors donné par le *développement multipôle*, centré en $\mathbf{Z_0}$, suivant :

$$\Phi(\mathbf{Z}) = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} M_j^k O_j^{-k}(\mathbf{Z} - \mathbf{Z_0}). \tag{2.10}$$

Les termes M_j^k sont appelés *termes du développement multipôle*. De plus, nous avons la propriété de symétrie suivante (d'après l'équation (2.9)) :

$$M_j^{-k} = (-1)^k \overline{M}_j^k. \tag{2.11}$$

Définition 2.2 (Opérateur *M2M*)**.** *Soient ${M'}_n^l$ (avec $n \ge 0$, $|l| \le n$) les termes de du développement multipôle centré en $\mathbf{Z_0}$, les termes du développement multipôle centré en $\mathbf{Z_1}$ sont donnés par :*

$$M_j^k = \sum_{n=0}^{j} \sum_{\substack{l=-n,\\ |k-l|\le j-n}}^{n} {M'}_n^l I_{j-n}^{k-l}(\rho, \alpha, \beta) \qquad \forall\,(j,k) \in \mathbb{N}\times\mathbb{Z} \quad \textit{avec} \quad 0 \le |k| \le j$$

où (ρ, α, β) sont les coordonnées sphériques du vecteur $\mathbf{Z_1} - \mathbf{Z_0}$.

Définition 2.3 (Opérateur *M2L*)**.** *Soient M_n^l (avec $n \ge 0$, $|l| \le n$) les termes du développement multipôle centré en $\mathbf{Z_1}$, les termes du développement local centré en $\mathbf{Z_2}$ sont donnés par :*

$$L_j^k = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} M_n^l O_{j+n}^{-k-l}(\rho, \alpha, \beta) \qquad \forall\,(j,k) \in \mathbb{N}\times\mathbb{Z} \quad \textit{avec} \quad 0 \le |k| \le j$$

où (ρ, α, β) sont les coordonnées sphériques du vecteur *M2L* $\mathbf{Z_2} - \mathbf{Z_1}$.

Le potentiel est alors donné par le *développement local* suivant :

$$\Phi(\mathbf{Z}) = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} L_j^k I_j^k(\mathbf{Z} - \mathbf{Z_2}). \tag{2.12}$$

Les termes L_j^k sont appelés *termes du développement local*. La propriété de symétrie de l'équation (2.9) entre des ordres opposés est aussi valable pour ces termes L_j^k :

$$L_j^{-k} = (-1)^k \overline{L}_j^k. \tag{2.13}$$

Ceci peut être prouvé en considérant que $\Phi(\mathbf{x}) \in \mathbb{R}$ et que les termes L_j^k sont uniques (dans la base des harmoniques sphériques) : la démonstration complète est donnée en annexe A.3.

Définition 2.4 (Opérateur *L2L*)**.** *Soient L'^l_n (avec $n \geq 0$, $|l| \leq n$) les termes de du développement local centré en $\mathbf{Z_2}$, les termes du développement local centré en $\mathbf{Z_3}$ sont donnés par :*

$$L_j^k = \sum_{n=j}^{+\infty} \sum_{\substack{l=-n, \\ |l-k| \leq n-j}}^{n} L'^l_n I^{l-k}_{n-j}(\rho, \alpha, \beta) \qquad \forall\, (j,k) \in \mathbb{N} \times \mathbb{Z} \quad \textit{avec} \quad 0 \leq |k| \leq j$$

où (ρ, α, β) sont les coordonnées sphériques du vecteur $\mathbf{Z_3} - \mathbf{Z_2}$.

Remarque 2.1. *Dans le cas de l'opérateur* L2L*, lorsqu'on traite en pratique de développements finis de degré maximum P, les termes du développement local de degré strictement supérieurs à P sont nuls et la somme infinie s'écrit alors $\sum_{n=j}^{P}$.*

Avec les opérateurs *M2M*, *M2L* et *L2L*, nous translatons ou convertissons un « ancien » développement (multipôle ou local) en un « nouveau » développement (multipôle ou local), à chaque fois grâce à des termes O_j^k ou I_j^k auxiliaires. Ces termes sont alors appelés *fonctions de transfert* M2M, M2L *ou* L2L.

Evaluation d'un développement local (opérateur *L2P*). L'opérateur *L2P* permet d'évaluer, en chacune des particules d'une cellule, le potentiel et la force dus à un développement local. Nous considérons tout d'abord l'évaluation du potentiel. Pour un développement local fini de degré maximum P, donné par L_j^k, $0 \leq j \leq P$, $|k| \leq j$, nous pouvons évaluer le potentiel induit en un point $\mathbf{Z}$ grâce à l'équation (2.12) :

$$\Phi(\mathbf{Z}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k I_j^k(r, \theta, \phi), \tag{2.14}$$

où (r, θ, ϕ) sont les coordonnées sphériques du vecteur $\mathbf{Z} - \mathbf{Z_2}$, $\mathbf{Z_2}$ étant le centre du développement local. Cependant, d'après les propriétés (2.9) et (2.13), nous avons : $L_j^k I_j^k + L_j^{-k} I_j^{-k} = L_j^k I_j^k + \overline{L}_j^k \overline{I}_j^k = 2\, Re(L_j^k I_j^k)$, où Re désigne la partie réelle d'un nombre complexe. Le potentiel peut donc être évalué plus rapidement par :

$$\Phi(\mathbf{Z}) = \sum_{j=0}^{P} \left[L_j^0 I_j^0(r, \theta, \phi) + \sum_{k=1}^{j} 2\, Re(L_j^k I_j^k(r, \theta, \phi)) \right].$$

Pour l'évaluation de la force, nous partons de l'équation (2.14) et nous obtenons la force $\mathbf{F}(\mathbf{Z})$ au point $\mathbf{Z}$ en calculant l'opposé du gradient du potentiel $\Phi(\mathbf{Z})$; dans la base locale des coordonnées sphériques (avec notre convention, exprimée en annexe A, selon laquelle θ est la coordonnée co-latitudinale et ϕ la coordonnée longitudinale), cela donne :

$$F_r(\mathbf{Z}) = -\frac{\partial\Phi(\mathbf{Z})}{\partial r}, \quad F_\theta(\mathbf{Z}) = -\frac{1}{r}\frac{\partial\Phi(\mathbf{Z})}{\partial\theta}, \quad F_\phi(\mathbf{Z}) = -\frac{1}{r\sin\theta}\frac{\partial\Phi(\mathbf{P})}{\partial\phi}.$$

Comme :

$$\begin{aligned}
\frac{\partial I_j^k(r,\theta,\phi)}{\partial r} &= \frac{j}{r} I_j^k(r,\theta,\phi),\\
\frac{\partial I_j^k(r,\theta,\phi)}{\partial\theta} &= i^{-|k|} A_j^k r^j \frac{\partial Y_j^k(\theta,\phi)}{\partial\theta},\\
\frac{\partial I_j^k(r,\theta,\phi)}{\partial\phi} &= ikI_j^k(r,\theta,\phi),
\end{aligned}$$

nous avons donc (voir en annexe A pour le calcul de $\frac{\partial Y_j^k(\theta,\phi)}{\partial\theta}$) dans la base locale des coordonnées sphériques :

$$\begin{aligned}
F_r(\mathbf{Z}) &= -\sum_{j=1}^{P}\sum_{k=-j}^{j}\frac{j}{r}L_j^k I_j^k(r,\theta,\phi)\\
&= -\frac{1}{r}\left(\sum_{j=1}^{P} jL_j^0 I_j^0(r,\theta,\phi) + \sum_{k=1}^{j} 2j\ Re\left(L_j^k I_j^k(r,\theta,\phi)\right)\right)
\end{aligned}$$

$$\begin{aligned}
F_\theta(\mathbf{Z}) &= -\frac{1}{r}\sum_{j=0}^{P}\sum_{k=-j}^{j} L_j^k\frac{\partial I_j^k(\theta,\phi)}{\partial\theta}\\
&= -\frac{1}{r}\left(\sum_{j=0}^{P} L_j^0\frac{\partial I_j^0(r,\theta,\phi)}{\partial\theta} + \sum_{k=1}^{j} 2\ Re\left(L_j^k\frac{\partial I_j^k(r,\theta,\phi)}{\partial\theta}\right)\right)
\end{aligned}$$

$$\begin{aligned}
F_\phi(\mathbf{Z}) &= -\frac{1}{r\sin\theta}\sum_{j=0}^{P}\sum_{k=-j}^{j} ikL_j^k I_j^k(r,\theta,\phi)\\
&= -\frac{1}{r\sin\theta}\sum_{j=0}^{P}\sum_{k=1}^{j}(-2k)\ Im\left(L_j^k I_j^k(r,\theta,\phi)\right)
\end{aligned}$$

où Im désigne la partie imaginaire d'un nombre complexe.

Pour la version adaptative de la FMM, nous avons besoin de deux opérateurs supplémentaires : l'opérateur *P2L* et l'opérateur *M2P*.

Définition 2.5 (Opérateur *P2L*)**.** *Soient m charges q_i localisées en $\mathbf{Q_i} = (\rho_i, \alpha_i, \beta_i)$ et dont les coordonnées relatives par rapport à un centre $\mathbf{Z_2}$ sont : $\mathbf{Q_i} - \mathbf{Z_2} = (\rho'_i, \alpha'_i, \beta'_i)$, nous définissons :*

$$L_j^k = (-1)^j \sum_{i=1}^{m} q_i O_j^{-k}(\rho'_i, \alpha'_i, \beta'_i) \qquad \forall\, (j,k) \in \mathbb{N} \times \mathbb{Z} \quad \textit{avec} \quad 0 \leq |k| \leq j. \tag{2.15}$$

D'après le théorème d'addition des harmoniques sphériques (voir théorème 1), le potentiel est alors donné par le développement local centré en $\mathbf{Z_2}$ suivant :

$$\Phi(\mathbf{Z}) = \sum_{j=0}^{+\infty} \sum_{k=-j}^{j} L_j^k I_j^k(\mathbf{Z} - \mathbf{Z_2}).$$

Evaluation d'un développement multipôle (opérateur *M2P*). L'opérateur *M2P* est similaire à l'opérateur *L2P*, mais il s'applique à un développement multipôle. Comme pour l'opérateur *L2P*, nous commençons par l'évaluation du potentiel. Pour un développement multipôle fini de degré maximum P, donné par M_j^k, $0 \leq j \leq P$, $|k| \leq j$, nous pouvons évaluer le potentiel induit en un point $\mathbf{Z}$ grâce à l'équation (2.10) :

$$\Phi(\mathbf{Z}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} M_j^k O_j^{-k}(r, \theta, \phi), \tag{2.16}$$

où (r, θ, ϕ) sont les coordonnées sphériques du vecteur $\mathbf{Z} - \mathbf{Z_0}$, $\mathbf{Z_0}$ étant le centre du développement multipôle. Mais comme : $M_j^k O_j^{-k} + M_j^{-k} O_j^k = M_j^k O_j^{-k} + \overline{M}_j^k \overline{O}_j^{-k} = 2Re(M_j^k O_j^{-k})$, nous pouvons évaluer plus rapidement le potentiel par :

$$\Phi(\mathbf{Z}) = \sum_{j=0}^{P} \left[M_j^0 O_j^0(r, \theta, \phi) + \sum_{k=1}^{j} 2Re(M_j^k O_j^{-k}(r, \theta, \phi)) \right].$$

Pour l'évaluation de la force, nous partons de l'équation (2.16), et comme pour l'opérateur *L2P*, nous souhaitons calculer l'opposé du gradient du potentiel. Comme :

$$\begin{aligned}
\frac{\partial O_j^{-k}(r, \theta, \phi)}{\partial r} &= -\frac{j+1}{r} O_j^{-k}(r, \theta, \phi), \\
\frac{\partial O_j^{-k}(r, \theta, \phi)}{\partial \theta} &= \frac{(-1)^j i^{|k|}}{A_j^{-k} r^{j+1}} \frac{\partial Y_j^{-k}(\theta, \phi)}{\partial \theta}, \\
\frac{\partial O_j^{-k}(r, \theta, \phi)}{\partial \phi} &= -ik O_j^{-k}(r, \theta, \phi),
\end{aligned}$$

nous avons donc, dans la base locale des coordonnées sphériques :

$$\begin{aligned} F_r(\mathbf{Z}) &= -\sum_{j=0}^{P}\sum_{k=-j}^{j}\left(-\frac{j+1}{r}\right)M_j^k O_j^{-k}(r,\theta,\phi) \\ &= \frac{1}{r}\left(\sum_{j=0}^{P}(j+1)M_j^0 O_j^0(r,\theta,\phi) + \sum_{k=1}^{j}2(j+1)Re\left(M_j^k O_j^{-k}(r,\theta,\phi)\right)\right), \end{aligned}$$

$$\begin{aligned} F_\theta(\mathbf{Z}) &= -\frac{1}{r}\sum_{j=0}^{P}\sum_{k=-j}^{j}M_j^k\frac{\partial O_j^{-k}(r,\theta,\phi)}{\partial\theta} \\ &= -\frac{1}{r}\left(\sum_{j=0}^{P}M_j^0\frac{\partial O_j^0(r,\theta,\phi)}{\partial\theta} + \sum_{k=1}^{j}2Re\left(M_j^k\frac{\partial O_j^{-k}(r,\theta,\phi)}{\partial\theta}\right)\right), \end{aligned}$$

$$\begin{aligned} F_\phi(\mathbf{Z}) &= -\frac{1}{r\sin\theta}\sum_{j=0}^{P}\sum_{k=-j}^{j}(-ik)M_j^k O_j^{-k}(r,\theta,\phi) \\ &= \frac{1}{r\sin\theta}\sum_{j=0}^{P}\sum_{k=1}^{j}(-2k)Im\left(M_j^k O_j^{-k}(r,\theta,\phi)\right). \end{aligned}$$

2.3.3.3 Bornes d'erreur

En pratique, les développements manipulés sont bien sûr finis, et plus leur degré maximum P est grand, plus l'erreur induite par l'utilisation de développements finis est faible. Afin de préciser le comportement de cette erreur, nous allons utiliser le lemme suivant.

Lemme 1. *Soit une charge q placée au point $\mathbf{Q} = (\rho,\alpha,\beta) \in R^3$ et un point $Z = (r,\theta,\phi) \in R^3$. On note γ l'angle entre les deux vecteurs $\mathbf{Z}$ et $\mathbf{Q}$.*

En supposant que $r > \rho$, on a alors la borne d'erreur suivante pour le développement multipôle du potentiel en $\mathbf{Z}$:

$$\left|\frac{q}{\|\mathbf{Z}-\mathbf{Q}\|} - \sum_{n=0}^{P}\frac{q\rho^n}{r^{n+1}}P_n(\cos\gamma)\right| \leq \frac{|q|}{r-\rho}\left(\frac{\rho}{r}\right)^{P+1}.$$

De la même façon, en supposant que $r < \rho$, on a alors la borne d'erreur suivante pour le développement local du potentiel en $\mathbf{Z}$:

$$\left|\frac{q}{\|\mathbf{Z}-\mathbf{Q}\|} - \sum_{n=0}^{P}\frac{qr^n}{\rho^{n+1}}P_n(\cos\gamma)\right| \leq \frac{|q|}{\rho-r}\left(\frac{r}{\rho}\right)^{P+1}.$$

Idée de la preuve. Dans le cas du développement multipôle, le potentiel au point $\mathbf{Z}$ est décomposé à l'aide des polynômes de Legendre selon l'équation (2.3) :

$$\frac{q}{\|\mathbf{Z}-\mathbf{Q}\|} = \sum_{n=0}^{+\infty}\frac{\rho^n}{r^{n+1}}P_n(\cos\gamma)$$

Comme $P_n(\cos\gamma) \leq 1$ (voir en annexe A), et d'après les résultats sur les séries géométriques, on en déduit la borne d'erreur. Le cas du développement local est symétrique. □

Remarque 2.2. *Les notions de développement multipôle et de développement local sont ici masquées par les polynômes de Legendre. La décomposition exacte de chaque polynôme de Legendre en harmoniques sphériques selon le théorème d'addition des harmoniques sphériques (théorème 1) met en lumière le type de développement utilisé suivant que $r > \rho$ ou que $r < \rho$.*

En reprenant les notations de la définition 2.1, avec pour simplifier $\mathbf{Z_0} = 0$, et en notant R_0 le rayon de la sphère centrée à l'origine et englobant toutes les particules (voir figure 2.7), on a alors la borne d'erreur suivante sur la convergence du développement multipôle fini, résultant de l'opérateur *P2M*, et évalué en $Z = (r, \theta, \phi) \in \mathbb{R}^3$:

$$\left| \Phi(\mathbf{Z}) - \sum_{j=0}^{P} \sum_{k=-j}^{j} M_j^k O_j^{-k}(\mathbf{Z}) \right| \leq \left(\frac{\sum_{i=1}^{m} |q_i|}{r - R_0} \right) \left(\frac{R_0}{r} \right)^{P+1}. \tag{2.17}$$

Idée de la preuve. Ce résultat se déduit directement du lemme 1 et de l'inégalité triangulaire. □

On note en particulier que la convergence du développement multipôle est assurée en tout point situé hors de la sphère de rayon R_0 et de centre $\mathbf{Z_0}$, c'est-à-dire la sphère englobant toutes les particules.

Comme expliqué dans [60] ou dans [51], l'opérateur *M2M* n'introduit pas d'erreur supplémentaire. En d'autres termes, l'erreur générée par un développement multipôle centré en $\mathbf{Z_1}$ résultant d'un appel à l'opérateur *P2M* en $\mathbf{Z_0}$ puis à l'opérateur *M2M* (pour passer de $\mathbf{Z_0}$ à $\mathbf{Z_1}$), est la même que celle générée par un développement multipôle construit directement grâce à l'opérateur *P2M* en $\mathbf{Z_1}$.

Pour la borne d'erreur d'un développement local, nous avons besoin de considérer deux sphères. Comme illustré en figure 2.7, la première, de centre $\mathbf{Z_1}$ et de rayon R_1, englobe toutes les particules, alors que la seconde, de centre $\mathbf{Z_2}$ et de rayon R_2, englobe tous les points où seront évalués le développement local. En notant $R = \|\mathbf{Z_1} - \mathbf{Z_2}\|$, et avec pour simplifier $\mathbf{Z_2} = 0$, on a alors la borne d'erreur suivante sur la convergence d'un développement local fini :

$$\left| \Phi(\mathbf{Z}) - \sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k I_j^k(\theta, \phi) \right| \leq \left(\frac{\sum_{i=1}^{m} |q_i|}{R - R_1 - R_2} \right) \left(\frac{R_2}{R - R_1} \right)^{P+1}. \tag{2.18}$$

Idée de la preuve. Ce résultat se déduit directement du lemme 1, et de l'inégalité triangulaire. □

La convergence du développement local est alors assurée dès que $R > R_1 + R_2$, c'est-à-dire dès que les deux sphères sont disjointes. Comme expliqué dans [60], et indiqué par l'expression exacte du théorème 4, l'opérateur *L2L* n'introduit tout simplement aucune erreur. En ce qui concerne l'opérateur *P2L* (voir définition 2.5), le développement local converge en chaque point $\mathbf{Z}$ vérifiant $\forall i : \ \|\mathbf{Z} - \mathbf{Z_2}\| \leq \|\mathbf{Q_i} - \mathbf{Z_2}\|$, et l'erreur induite vérifie l'équation (2.18).

Cependant, afin d'assurer une convergence de l'erreur suffisament rapide, nous allons imposer une condition plus stricte que $R > R_1 + R_2$ en définissant la notion de *bonne séparation* (*well-separateness* [65]) qui est au cœur de la FMM. Entre deux cellules *bien séparées* on pourra construire puis convertir le développement multipôle de l'une pour évaluer le potentiel en tout point de l'autre grâce à un développement local.

Afin de définir cette propriété et d'exposer les convergences correspondantes, on se concentre sur la partie des bornes d'erreur qui dépend de P, à savoir $\left(\frac{R_1}{r}\right)^{P+1}$ et $\left(\frac{R_2}{R-R_1}\right)^{P+1}$. Une fois appliquées à l'octree de la FMM, les sphères englobant tous les points d'une cellule correspondent aux sphères concentriques et circonscrites à chaque cellule. La taille du côté de chaque cellule étant à chaque fois éliminée dans ces expressions, la convergence des bornes d'erreur est indépendante du niveau de l'octree. Et comme l'erreur générée par le développement multipôle est plus faible que celle générée par le développement local (voir [68]), on examine d'abord la convergence de l'erreur du développement local.

Dans [65], Greengard & Rokhlin souhaitaient imposer une convergence en $\left(\frac{1}{2}\right)^{P+1}$. En $2D$, où la convergence de l'erreur suivant P est la même qu'aux équations (2.17) et (2.18), deux cellules sont considérées comme bien séparées dans [65] si elles ne sont pas des *voisins immédiats*, c'est-à-dire si elles ne partagent ni un sommet, ni une arête, ni une face (en $3D$). En fait, comme précisé dans [60] [62], la convergence alors obtenue n'est pas de $\left(\frac{1}{2}\right)^{P+1}$, mais de $\left(\frac{1}{2\sqrt{2}-1}\right)^{P+1} \approx (0,547)^{P+1}$. Celle-ci correspond bien à la borne de l'équation (2.18) avec $R_1 = R_2 = \frac{\sqrt{2}}{2}$ et $R \geq 2$. Et l'erreur dû au développement multipôle converge en $\left(\frac{\sqrt{2}}{3}\right)^{P+1} \approx (0,471)^{P+1}$ (équation (2.17) avec $r \geq \frac{3}{2}$) : la borne d'erreur de la FMM en $2D$ est donc fixée par celle du développement local.

Dans le cas $3D$ pour obtenir une convergence en au moins $\left(\frac{1}{2}\right)^{P+1}$, il faut exclure de la notion de bonne séparation, en plus des voisins immédiats, les *seconds voisins*, c'est-à-dire les voisins des voisins immédiats. On obtient alors une convergence en $\left(\frac{1}{2\sqrt{3}-1}\right)^{P+1} \approx (0,406)^{P+1}$ pour l'équation (2.18), car $R_1 = R_2 = \frac{\sqrt{3}}{2}$ et $R \geq 3$. En notant ws le *critère de bonne séparation* qui détermine le nombre de cellules entre deux cellules bien séparées, on a donc : $ws = 2$. Cependant, comme nous le verrons à la section 2.3.3.5, ceci accroît très sensiblement le coût de calcul. On préfère en pratique (voir en section 2.3.3.5) n'exclure que les voisins immédiats ($R \geq 2$), c'est-à-dire utiliser $ws = 1$ et obtenir ainsi une convergence en $\left(\frac{1}{\frac{4}{\sqrt{3}}-1}\right)^{P+1} \approx (0,764)^{P+1}$ pour l'équation (2.18). Là encore, on vérifie que la convergence du développement multipôle est plus rapide et que la borne d'erreur de la FMM en $3D$ est donc fixée par celle du développement local : avec les voisins immédiats et les seconds voisins on a en effet une convergence en $\left(\frac{\sqrt{3}}{5}\right)^{P+1} \approx (0,346)^{P+1}$ pour l'équation (2.17) car $r \geq \frac{5}{2}$, et avec les seuls voisins immédiats la convergence est en $\left(\frac{1}{\sqrt{3}}\right)^{P+1} \approx (0,577)^{P+1}$ car $r \geq \frac{3}{2}$.

Ainsi P peut être déterminé en fonction de la précision ϵ souhaitée : $P \approx -log_c(\epsilon)$, avec $c = 0,764$ en $3D$ lorsqu'on n'exclut que les voisins immédiats.

Comme mentionné par plusieurs auteurs [113] [124], il y a cependant un problème avec la borne d'erreur du développement local présentée ici. L'équation (2.18) traduit en effet

l'erreur due à un développement local certes fini, mais dont chaque terme serait exact. Cette borne d'erreur s'appliquerait à l'opérateur *P2L*, ou à un opérateur *M2L* agissant sur un développement multipôle infini. Or en pratique, l'opérateur *M2L* agit sur un développement multipôle fini, et les deux développements multipôle et local, sont donc finis « en même temps ». Deux expressions différentes de l'opérateur *M2L* fini peuvent alors être distinguées.

2.3.3.4 Noyaux de hauteurs simple et double

La première expression *M2L*, qui est aussi la plus courante, s'écrit :

$$L_j^k = \sum_{n=0}^{P} \sum_{l=-n}^{n} M_n^l O_{j+n}^{-k-l}(\rho, \alpha, \beta). \tag{2.19}$$

Nous utilisons ici tous les termes M_n^l du développement multipôle (ces termes sont nuls hors de $n \geq 0$, $|l| \leq n$) et nous utilisons aussi les termes O_j^k (de la fonction de transfert *M2L*) avec des degrés allant jusqu'à $2P$. Même si des travaux importants ont été réalisés afin d'estimer le comportement de l'erreur introduite par cette expression *M2L* (voir [98] [113]), aucune borne d'erreur précise n'a encore été prouvée. Nous nommons cette expression *M2L* : *noyau*[2] M2L *de hauteur double*, ou *UpTo2P*.

La seconde expression de l'opérateur *M2L* est :

$$L_j^k = \sum_{n=0}^{P-j} \sum_{l=-n}^{n} M_n^l O_{j+n}^{-k-l}(\rho, \alpha, \beta). \tag{2.20}$$

Nous limitons ici le degré maximum des termes O_j^k utilisés à P. Nous nommons donc cette expression *M2L* : *noyau* M2L *de hauteur simple*, ou *UpToP*. Celle-ci a par exemple été utilisée dans le code DPMTA, mais la preuve de la borne d'erreur donnée dans [51] est incorrecte[3]. Nous recommendons néanmoins la lecture de cette annexe car elle présente une analyse de l'erreur dans le pire des cas similaire à celle utilisée ici. En particulier, elle montre que les opérateurs *M2M* et *L2L* n'influent pas sur la borne d'erreur, comme expliqué en section 2.3.3.3.

Il peut être noté que cette borne d'erreur pour un noyau *M2L* de hauteur simple a été brièvement présentée dans [124], mais les auteurs ont utilisé un noyau de hauteur double dans leur implémentation de la FMM.

2.3.3.5 Algorithme uniforme

Avant de décrire l'algorithme de la FMM, nous avons encore besoin d'introduire la notion de *liste d'interaction* qui se déduit de celle de bonne séparation. La liste d'interaction d'une cellule c regroupe toutes les cellules qui sont bien séparées de c, mais dont le père n'était pas bien séparé du père de c. Une représentation de cette liste d'interaction en $2D$ est donnée en figure 2.8.

En d'autres termes, la liste d'interaction de c est formée, dans le cas $ws = 1$, des fils des voisins immédiats du père de c qui ne sont pas eux-mêmes des voisins immédiats de c. Dans

[2] Le terme *noyau* provient de [51].

[3] L' équation (C.23) dans l'annexe C de [51] est fausse car la composition d'opérateurs différentiels diffère du produit des dérivées.

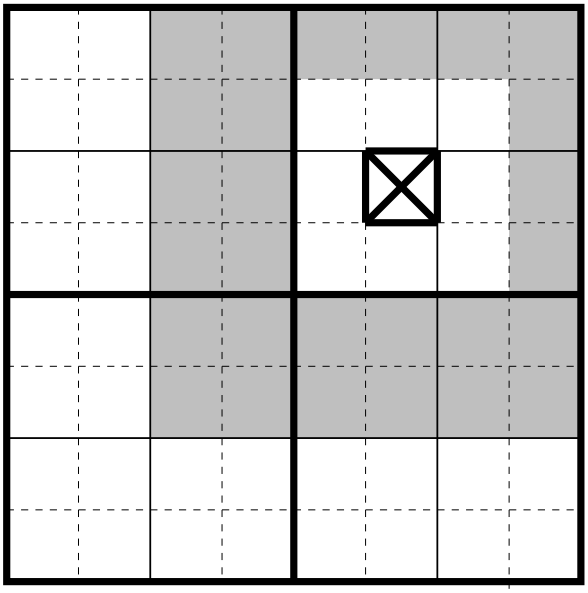


FIG. 2.8 – Liste d'interaction en $2D$ ($ws = 1$).

le cas $ws = 2$, il faut prendre les fils des voisins immédiats et des seconds voisins du père, et en exclure les voisins immédiats et les seconds voisins de c. Le nombre d'éléments dans cette liste est donc de 189 pour $ws = 1$, mais de 875 pour $ws = 2$.

L'ensemble des voisins immédiats (dans le cas $ws = 1$, auxquels on ajoute les seconds voisins dans le cas $ws = 2$) forme quant à lui la *liste du calcul direct*, ou *liste du champ proche*.

Description de l'algorithme. Nous allons décrire l'algorithme de la version séquentielle, en trois dimensions, de la FMM *uniforme* (ou *non adaptative*) destinée aux distributions uniformes de particules. Nous nous plaçons ici dans le cas de systèmes isolés dans l'espace : il n'y a aucune particule hors de la racine de l'octree (*conditions aux limites libres*). Nous commençons par préciser les notations utilisées :

- N désigne le nombre de corps (ou particules) étudiés ;
- P désigne le degré maximum des développements multipôle et locaux ;
- H désigne la hauteur de l'octree (complet) utilisé. Cette hauteur, qui correspond au degré de raffinement souhaité, est fixée par l'utilisateur ;
- $\mathcal{M}_{l,i}$ désigne le développement multipôle dû aux particules contenues dans la cellule i du niveau l ;
- $\mathcal{L}_{l,i}$ désigne le développement local, pour la cellule i du niveau l, décrivant le potentiel dû aux particules contenues dans les cellules *bien-séparées* de la cellule i.

La construction de l'octree complet est supposée avoir déjà été réalisée lors d'une étape initiale, et les particules sont stockées au niveau des feuilles comme représenté en figure 2.9.

Voici l'algorithme en langage de description algorithmique. Il est composé de quatre phases principales : une *remontée* de l'arbre depuis ses feuilles, suivie d'une *descente*, puis

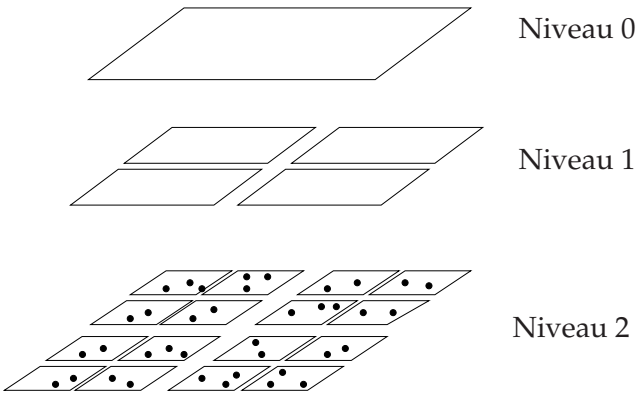


FIG. 2.9 – Situation initiale de la FMM.

une *phase de calcul direct* et enfin une *phase d'évaluation* du champ lointain.

Phase de remontée. La phase de remontée est constituée de deux étapes. La première effectue la construction du développement multipôle de chaque feuille grâce à l'opérateur *P2M* : voir figure 2.10. La seconde calcule, lors d'une remontée dans l'octree, le développement multipôle de chaque cellule interne grâce à l'opérateur *M2M* : voir figure 2.11.

Cette phase de remontée est décrite par l'algorithme 1.

Algorithme 1 FMM uniforme en $3D$: **PHASE DE REMONTEE**

Pour $i = 1$ à 8^H **faire**
  Calculer $\mathcal{M}_{H,i}$ grâce à l'opérateur *P2M* ;
Fin pour
Pour $l = H - 1$ à 0 **faire**
  Pour $i = 1$ à 8^l **faire**
    Pour $j = 0$ à 8 **faire**
      Translater $\mathcal{M}_{l+1,8.i+j}$ du $j^{\text{ième}}$ fils de la cellule i au centre de i grâce à l'opérateur *M2M*, et l'additionner à $\mathcal{M}_{l,i}$;
    Fin pour
  Fin pour
Fin pour

Phase de descente. Lors de la phase de descente, décrite par l'algorithme 2, on établit le développement local de chaque cellule en convertissant le développement multipôle de chaque membre de sa liste d'interaction, voir figure 2.12, et en les additionnant au résultat de la translation du développement local de son père : voir figure 2.13. Cette translation du développement local du père permet de prendre en compte toutes les cellules qui sont au-delà de la liste d'interaction. Dans le cas des conditions aux limites libres, les développements locaux des cellules des niveaux 0 et 1 sont considérés comme nuls : la phase de descente peut donc commencer au niveau 2.

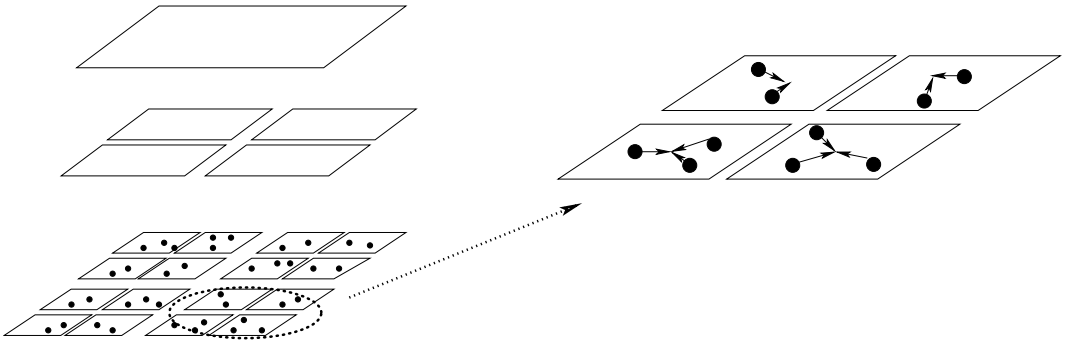


FIG. 2.10 – Remontée : opérateur *P2M*.

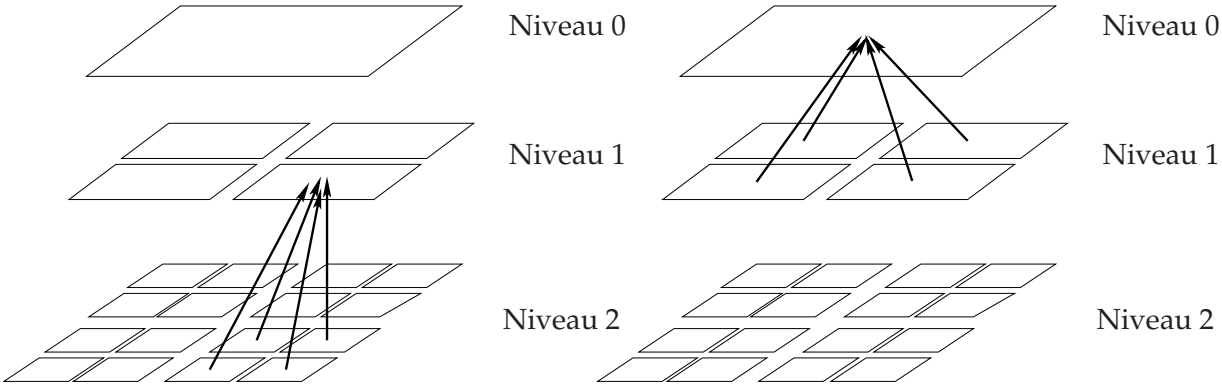


FIG. 2.11 – Remontée : opérateur *M2M*.

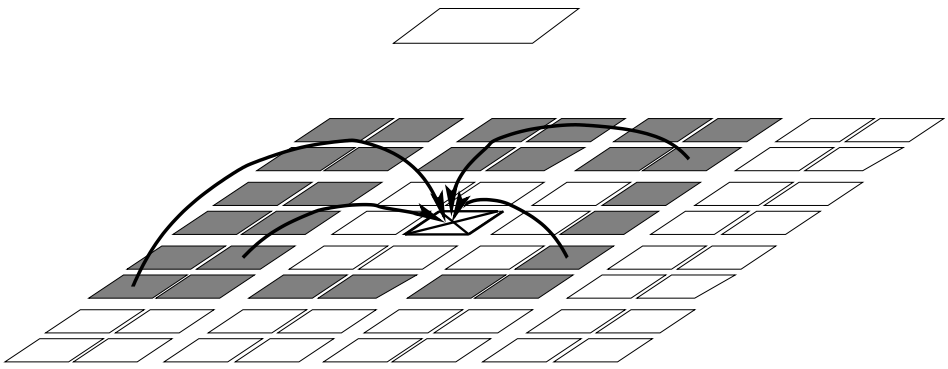


FIG. 2.12 – Descente : opérateur *M2L*.

Algorithme 2 FMM uniforme en $3D$: **PHASE DE DESCENTE**

1: **Pour** $l = 2$ à H **faire**
2: **Pour** $i = 1$ à 8^l **faire**
3: Translater $\mathcal{L}_{l-1,i/8}$ du père de la cellule i en son centre grâce à l'opérateur *L2L*, et le stocker dans $\mathcal{L}_{l,i}$;
4: **Pour** chacune des cellules j de la liste d'interaction de la cellule i **faire**
5: Convertir $\mathcal{M}_{l,j}$ en un développement local basé sur le centre de la cellule i grâce à l'opérateur *M2L*, et l'additionner à $\mathcal{L}_{l,i}$;
6: **Fin pour**
7: **Fin pour**
8: **Fin pour**

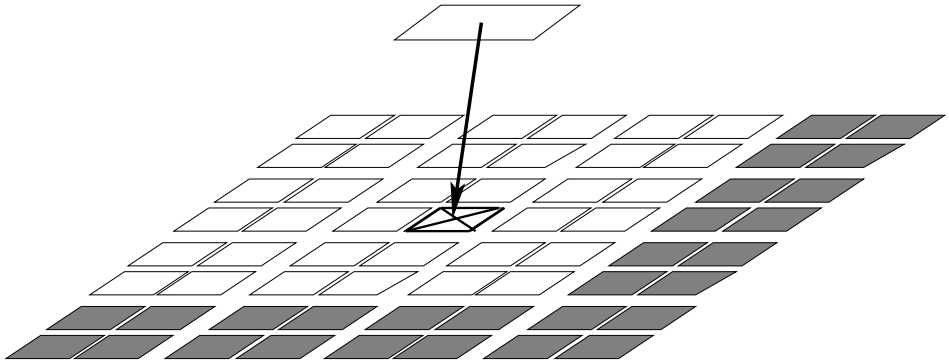


FIG. 2.13 – Descente : opérateur *L2L*.

Phase de calcul direct. Lors de la phase de calcul direct, décrite par l'algorithme 3, le champ proche est évalué par un calcul direct entre la cellule cible et chaque membre de sa liste du champ proche. On évalue aussi par calcul direct les interactions entre les particules de chaque cellule. Cette phase est représentée en figure 2.14.

Algorithme 3 FMM uniforme en $3D$: **PHASE DE CALCUL DIRECT**

1: **Pour** $i = 1$ à 8^H **faire**
2: **Pour** chacune des cellules j de la liste du calcul direct de la cellule i du niveau H **faire**
3: Calculer l'interaction entre toutes les particules de la cellule i et toutes les particules de la cellule j grâce à l'opérateur *P2P* ;
4: **Fin pour**
5: Calculer les interactions entre toutes les particules de la cellule i grâce à l'opérateur *P2P* ;
6: **Fin pour**

Phase d'évaluation. Enfin, lors de la phase d'évaluation, décrite par l'algorithme 4, le développement local de chaque feuille est évalué en chacune de ses particules, et le résultat est additionné au calcul du champ proche.

Une fois le potentiel et la force calculés, on en déduit l'accélération de chaque particule et

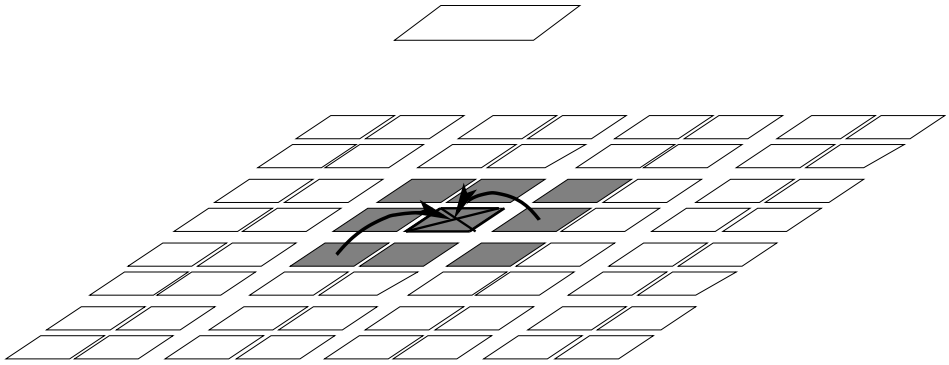


FIG. 2.14 – Calcul direct (opérateur *P2P*).

Algorithme 4 FMM uniforme en $3D$: **PHASE D'EVALUATION**

1: **Pour** $i = 1$ à 8^H **faire**
2: **Pour** chaque particule j localisée en z_j dans la cellule i du niveau H **faire**
3: Evaluer $\mathcal{L}_{l,i}(z_j)$ grâce à l'opérateur *L2P*, et additionner le résultat au potentiel et à la force obtenus par le calcul direct ;
4: **Fin pour**
5: **Fin pour**

on met à jour leur position avec un schéma d'intégration, par exemple du type Euler, *leap-frog* [20] [89] ou encore Runge-Kutta [114]. Il faut alors mettre à jour les cellules de l'octree qui ont perdu ou gagné des particules, puis passer au pas de temps suivant. Des pas de temps individuels pour les particules peuvent être utilisés afin de réduire sensiblement le temps de calcul : une particule isolée n'a pas besoin d'un pas de temps aussi court qu'une particule dans une zone à forte densité [58] [121]. De la même façon, dans les méthodes dites à pas multiples, un pas de temps différent peut être utilisé pour le champ proche (pas de temps court) et le champ lointain (pas de temps long) [22] [49] [51] [114] [119].

Complexité théorique. Dans cette section, nous allons présenter la complexité théorique de la FMM dans le cas d'une distribution uniforme de N particules. La hauteur de l'octree H est fixée par l'utilisateur. Elle impose directement le nombre de particules moyen au niveau des feuilles suivant les relations : $s_{moy} = \frac{N}{8^H}$ et $H \approx \log_8\left(\frac{N}{s_{moy}}\right)$ (H étant alors fixé comme l'entier le plus proche). La preuve de la complexité linéaire de la FMM repose sur le fait que s_{moy} puisse être considéré comme une constante, ou du moins qu'il puisse être considéré comme appartenant à un intervalle de valeurs dont les extrémités sont constantes. Ainsi lorsque N augmente, la hauteur H doit être adaptée afin que s_{moy} reste dans cet intervalle.

Plus précisément, pour une hauteur H donnée, le nombre de feuilles $\mathcal{F}$ vaut : $\mathcal{F} = 8^H = O(N)$, et le nombre total $\mathcal{N}$ de cellules dans l'octree vaut : $\mathcal{N} = \frac{8^{H+1}-1}{7} = O(N)$. Le tableau 2.1 donne alors le nombre théorique d'opérations associées à chaque opérateur. Au final, la complexité obtenue est bien linéaire avec un facteur constant en $\mathcal{O}(P^4)$.

Cependant, la complexité de la FMM à H constant reste quadratique par rapport à N, et ce à cause du calcul direct de l'opérateur *P2P*. Ceci est illustré en figure 2.15, où nous présen-

Opérateur	Nombre d'opérations
P2M	$\mathcal{F}.\mathcal{O}(P^2).s_{moy} = \mathcal{O}(P^2N)$
M2M	$\mathcal{N}.\mathcal{O}(P^4) = \mathcal{O}(P^4N)$
M2L (avec $ws = 1$)	$\mathcal{N}.189.\mathcal{O}(P^4) = \mathcal{O}(P^4N)$
L2L	$\mathcal{N}.\mathcal{O}(P^4) = \mathcal{O}(P^4N)$
L2P	$\mathcal{F}.s_{moy}.\mathcal{O}(P^2) = \mathcal{O}(P^2N)$
P2P (avec le principe des interactions réciproques)	$\mathcal{F}.27.\frac{s_{moy}^2}{2} = \mathcal{O}(N)$
Total	$\mathcal{O}(P^4N)$

TAB. 2.1 – Nombre théorique d'opérations associées à chaque opérateur de la FMM uniforme.

tons le coût théorique de la FMM pour trois valeurs de H. Ce coût théorique est estimé grâce à une analyse détaillée du nombre d'opérations effectuées par chaque opérateur (noyau de hauteur double pour l'opérateur *M2L*), et grâce à une pondération de chaque opération élémentaire (addition, multiplication, division, cosinus, racine carrée, *etc.*) par le nombre de cycles qu'elle requiert au niveau du processeur. La complexité linéaire masque donc bien un réajustement nécessaire de H lorsque N varie, qui permet de consverver s_{moy} dans un intervalle d'extrémités constantes. En fait, *le choix de H, et donc de s_{moy}, doit équilibrer le coût du calcul du champ proche et le coût de calcul du champ lointain*. En utilisant par exemple les outils de résolution d'équation du logiciel *Maple*, on peut aussi déduire de ce coût théorique les bornes de l'intervalle pour s_{moy} en fonction de P. Pour $P = 5$, s_{moy} doit être dans l'intervalle $[\![20, 153]\!]$, alors que pour $P = 15$ l'intervalle est $[\![118, 937]\!]$. La justesse de ces calculs a été confirmée en pratique par des mesures de temps CPU sur des exécutions de la FMM.

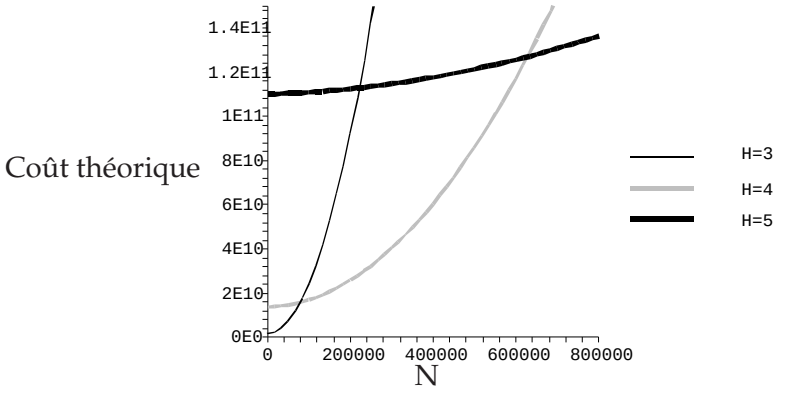


FIG. 2.15 – Coût théorique total de la FMM en fonction de la hauteur H de l'octree pour une distribution uniforme de N particules.

On note de plus que l'opérateur *M2L* représente la majeure partie du temps de calcul pour le champ lointain. Pour chaque cellule, nous dénombrons en effet 189 appels à l'opérateur *M2L* contre un seul appel à chacun des autres opérateurs du champ lointain (*P2M*, *M2M*, *L2L*, *L2P*). Dans le cas $ws = 2$, le nombre d'appels *M2L* par cellule monte à 875, ce qui accroît fortement le nombre d'opérations de la FMM : on préfère donc généralement

utiliser $ws = 1$ (voir [68] [98] par exemple), même si l'erreur induite est plus importante à P constant.

Opérateur *M2L*. Comme référence pour de prochaines comparaisons, nous présentons finalement la différence en termes de complexité entre les noyaux *M2L* de hauteurs simple et double pour le schéma de calcul original de l'opérateur *M2L*, aussi appelé *M2L classique*. Comme nous le verrons en section 2.3.3.6, seuls les termes d'ordre positif ou nul sont effectivement calculés dans les développements locaux. Pour un noyau de hauteur simple, nous avons donc :

$$M2L = \frac{1}{12}(P+3)(P+1)(P+2)^2,$$

alors que pour un noyau de hauteur double, nous obtenons :

$$M2L = \frac{1}{2}(P+2)(P+1)^3.$$

Retenons pour l'instant que le rapport entre les deux noyaux est environ 6.

2.3.3.6 Implémentation efficace de la FMM

Comme présenté en section 2.3.3.5, la hauteur de l'octree doit être choisie afin d'équilibrer le coût de calcul du champ proche avec celui du champ lointain.

Champ proche. Le calcul direct du champ proche représente donc, en moyenne, la moitié du coût total de la FMM. Son optimisation est donc cruciale. Elle est tout d'abord réalisée en ayant systématiquement recours au principe des interactions réciproques. Il suffit simplement pour cela de n'appeler l'opérateur *P2P* à la ligne 3 de l'algorithme 3 que lorsque $i < j$. De la même façon, on appliquera ce principe au calcul direct des interactions au sein d'une unique cellule (ligne 3 de l'algorithme 3) par des tests sur les indices des particules. L'optimisation des boucles et des instructions au sein de l'opérateur *P2P* sont aussi bien sûr décisives.

Champ lointain. Pour ce qui est du champ lointain, la majeure partie du temps de calcul est due aux appels à l'opérateur *M2L*, comme expliqué en section 2.3.3.5. L'optimisation du calcul de l'opérateur *M2L* fera l'objet de la partie II de ce document. Cependant nous pouvons d'ores et déjà présenter certains améliorations, relativement simples à mettre en place, et concernant tous les opérateurs qui agissent sur les développements.

Tout d'abord, comme pour le calcul direct, l'implémentation des boucles et des instructions doit être optimisée : ceci concerne l'accès aux tableaux où sont stockés les développements, la suppression de toute redondance dans les calculs, l'élimination des tests au sein des boucles de calcul, la minimisation des accès mémoire et l'utilisation d'options d'optimisation lors de la compilation. Mais deux optimisations d'ordre algorithmique sont aussi capitales.

1. La première est de ne calculer dans les développements multipôles et locaux que les termes d'ordre positif. Ceci est rendu possible par les équations (2.11) et (2.13) qui permettent de déduire aisément les termes d'ordre négatif. Le temps de calcul est alors presque diviser par 2, et l'espace mémoire utilisé pour le stockage des développements

est aussi réduit de moitié, car seuls les termes d'ordre positif sont effectivement stockés.

2. La seconde optimisation concerne le calcul des fonctions de transfert *M2M*, *M2L* et *L2L* (voir section 2.3.3.2). Ces fonctions de transfert ne dépendent que du vecteur $\mathbf{v}$ entre le centre de la cellule cible et le centre de la cellule source. Il est donc intéressant de les précalculer une seule fois pour chaque vecteur $\mathbf{v}$ possible, de les stocker dans une structure de données adéquate et d'y accéder lors de chaque opération *M2M*, *M2L* ou *L2L* en fonction du vecteur $\mathbf{v}$ utilisé. Il y a 8 vecteurs possibles (un pour chacun des 8 fils) pour les fonctions de transfert *M2M* et *L2L*, et $(2(2ws+1)+1)^3-(2*ws+1)^3=316$ vecteurs possibles pour les fonctions de transfert *M2L* (avec $ws=1$). Cependant les valeurs de ces vecteurs changent à chaque niveau en fonction de la taille du côté d'une cellule, et les fonctions de transfert doivent donc être précalculées pour chaque niveau. Ceci avait déjà été implémenté dans le code DPMTA [51] [102] par exemple.

Dans cette même dissertation [51], W.D. Elliott s'appuie sur l'ordre de Morton pour accéder rapidement aux éléments de la liste d'interaction d'une cellule c : à l'aide d'opérations arithmétiques et booléennes entre les bits de l'indice de Morton de c, ceux de l'indice de Morton du vecteur *M2L*, et ceux de masques de bits dépendants du niveau de l'octree, on peut calculer, avec un petit nombre constant d'instructions, l'indice de Morton de l'élément correspondant dans la liste d'interaction. De plus, le calcul reste correct pour des conditions aux limites périodiques, et pour des conditions aux limites libres des bits de débordement permettent de détecter si l'on a franchi les limites de l'octree. Le détail des opérations à effectuer est présenté dans [51] et dans le code de DPMTA. Ce mécanisme s'applique aussi à l'accès aux voisins immédiats d'une feuille pour le calcul direct du champ proche, et les accès aux indices du père ou des différents fils sont triviaux à mettre en œuvre pour l'ordre de Morton.

Pour clore cette partie sur l'implémentation, nous précisons que le calcul des harmoniques sphériques nécessaires est directement réalisé à partir des fonctions associées de Legendre, évaluées selon la méthode présentée dans [100].

Besoins mémoire de l'opérateur *M2L*. Enfin, en vue d'une comparaison avec les autres schémas de calcul qui vont être présentés en section 2.3.3.12 et étudiés au chapitre 4, nous détaillons ici les besoins mémoire pour le schéma de calcul original de l'opérateur *M2L* (*M2L classique*). Nous nous concentrons ici uniquement sur la mémoire utilisée par les développements. On note t_c la taille d'un nombre complexe, et en double précision nous avons donc : $t_c = 16$ octets. Comme dans les développements nous ne stockons uniquement que les termes d'ordre positif ou nul, la taille d'un développement multipôle ($\mathcal{M}$) ou d'un développement local ($\mathcal{L}$) s'écrit en fonction de P (voir les définitions 2.1 et 2.3) :

$$\mathcal{M}(P)=\mathcal{L}(P)=\sum_{j=0}^{P}\sum_{k=0}^{j}t_c=\frac{(P+1)(P+2)}{2}t_c.$$

Le nombre de cellules $\mathcal{N}$ dans un octree de hauteur H est :

$$\mathcal{N}(H)=\sum_{l=0}^{H}8^l=\frac{8^{H+1}-1}{7}.$$

Nous devons aussi déterminer l'espace mémoire nécessaire pour stocker les fonctions de transfert *M2L*, ces fonctions étant précalculées à chaque niveau. Leur nombre correspond au nombre de tous les vecteurs *M2L* possibles, soit $\mathcal{N}_\mathcal{T} = (2(2ws+1)+1)^3 - (2ws+1)^3$, et ainsi avec $ws = 1$:

$$\mathcal{N}_\mathcal{T} = 316.$$

Avec un noyau *M2L* de hauteur simple, l'espace mémoire (ou taille) $\mathcal{T}$ d'une fonction de transfert *M2L* est égale à $\mathcal{M}$, mais avec un noyau de hauteur double elle vaut : $\mathcal{T}(P) = \mathcal{M}(2P)$. Par conséquent, les besoins mémoire totaux *Mem* pour le calcul *M2L* classique sont :

$$Mem(P,H) = \mathcal{N}(H).(\mathcal{M}(P) + \mathcal{L}(P)) + \mathcal{N}_\mathcal{T}.\mathcal{T}(P).$$

Pour un noyau de hauteur simple, ceci donne :

$$Mem_{Sg}(P,H) = \mathcal{N}(H)(P+1)(P+2)t_c + \mathcal{N}_\mathcal{T}\frac{(P+1)(P+2)}{2}t_c, \tag{2.21}$$

et pour un noyau de hauteur double :

$$Mem_{Db}(P,H) = \mathcal{N}(H)(P+1)(P+2)t_c + \mathcal{N}_\mathcal{T}\frac{(2P+1)(2P+2)}{2}t_c. \tag{2.22}$$

2.3.3.7 Conditions aux limites

Afin de minimiser le nombre d'atomes et de réduire les effets de surface, les simulations en dynamique moléculaire sont habituellement réalisées avec des *conditions aux limites périodiques* (*Periodic Boundary Condition*, ou PBC) : le système moléculaire est répliqué à l'infini dans toutes les dimensions ; on obtient alors un réseau cristallin. Nous présentons ici une étude des différentes méthodes possibles pour traiter ces conditions aux limites périodiques.

Historiquement ces simulations étaient réalisées grâce à la sommation d'Ewald [7] [16], présentée pour la première fois en 1923, qui impose que la cellule originale soit neutre électriquement. De par la précision obtenue, cette méthode en $\mathcal{O}(N^2)$ sert de référence pour les calculs avec conditions aux limites périodiques. Des méthodes d'Ewald rapides (*fast Ewald methods*) ont été introduites plus récemment : elles permettent de réduire la complexité théorique à un $O(N \log N)$ pour la méthode Particle Mesh Ewald [38], inspirée de la méthode *Particle-Mesh* et qui a donc recours à la Transformée Rapide de Fourier (FFT), ou à un $O(N^{3/2})$ pour l'algorithme de Perram *et al.* [97].

La sommation d'Ewald peut être efficacement couplée avec la FMM. A la fin de la phase de remontée on dispose en effet du développement multipôle de la cellule à la racine de l'arbre, qui correspond à la cellule originale répliquée à l'infini, notée $\mathcal{C}_\prime$. La sommation d'Ewald appliquée aux développements de la FMM permet alors de construire le développement local de $\mathcal{C}_\prime$ dû à toutes les cellules bien séparées de $\mathcal{C}_\prime$: ces cellules, dont le nombre est infini, sont toutes des images périodiques de $\mathcal{C}_\prime$ et leur développement multipôle est donc égal à celui de $\mathcal{C}_\prime$. En deux dimensions, ce couplage a été évoqué dès le premier article de Greengard & Rokhlin en 1987 [65], et approfondi par la suite [37] [112]. En trois dimensions, la première tentative date de 1991 avec Schmidt & Lee [106] (attention, les équations (7)-(11) sont erronées). Des détails du calcul du développement local (en coordonnées cartésiennes) par sommation d'Ewald ont été apportées dans [108] (annexe C). Les formules complètes pour les coordonnées sphériques (c'est-à-dire pour des développement en harmoniques sphériques) ont quant à elles été présentées par Challacombe *et al.* [33]. La sommation

d'Ewald pouvant être précalculée (en fonction de la taille de $\mathcal{C}_l$) avant même le calcul FMM, le surcoût est faible. La phase de descente est alors similaire à l'algorithme donné pour des conditions aux limites libres, à la différence que désormais les cellules aux bords de l'octree ont une liste d'interaction et une liste du calcul direct toutes deux complètes. En effet, lorsqu'un voisin se trouve hors des frontières de l'octree, on utilise, du fait de la périodicité, l'image de la cellule située à l'opposé dans l'octree. Au final, le surcoût reste modéré par rapport au coût de la FMM pour des systèmes moléculaires isolés.

Cependant, une autre méthode permet aussi de traiter les conditions aux limites périodiques avec la FMM, mais sans recourir à la sommation d'Ewald : c'est la *Macroscopic Multipole Method* (MMM) [25] [80] (voir aussi [51] et [102] pour l'implémentation). Dans cette méthode, la phase de remontée est prolongée de k niveaux « au dessus » de $\mathcal{C}_l$ (ces k niveaux virtuels peuvent être aussi vus comme des niveaux négatifs dans l'octree), et la construction des développements locaux de la phase de descente débute alors par ces k niveaux. On prend alors en compte l'influence d'un nombre fini d'images périodiques de $\mathcal{C}_l$, et la valeur de k doit être assez grande ($k \in [\![3, 6]\!]$ en pratique) pour obtenir des précisions comparables à celles obtenues par la sommation d'Ewald. Le surcoût reste modéré (25 à 30 % d'après [16]), et les avantages de la MMM par rapport à la sommation d'Ewald, outre le fait que les bases mathématiques de la MMM se limitent à celles de la FMM, sont qu'elle permet de traiter des cellules $\mathcal{C}_l$ dont la charge totale n'est pas nulle, et qu'elle permet de ne répliquer $\mathcal{C}_l$ que dans deux des trois dimensions de l'espace : on peut ainsi étudier des surfaces (simulation de membranes en biologie par exemple).

Plusieurs comparaisons ont été réalisées entre ces différentes méthodes : [25], [33], [53], [80], [112], [106], et [16]. Pour des simulations de grande taille, le couplage FMM/Ewald ([33] et [112]) et la MMM ([25]) sont plus les efficaces que les méthodes d'Ewald rapides. Quant au choix entre le couplage FMM/Ewald et la MMM, il ne peut être réalisé qu'après une comparaison des surcoûts respectifs sur la FMM, ou en fonction du type de simulation visée (charge totale nulle ou pas, réplication à l'infini dans les trois dimensions ou uniquement sur une surface).

2.3.3.8 La méthode d'Anderson : une FMM sans multipôles

Dans [10], C. R. Anderson a présenté une méthode originale qui utilise le même algorithme que la FMM, mais où le champ lointain n'est plus représenté par un développement multipôle sous la forme d'harmoniques sphériques. A la place, l'auteur utilise l'équation de Poisson pour établir des approximations basées sur l'évaluation du potentiel en plusieurs points d'une sphère englobant les particules. Les formules ainsi obtenues sont plus simples, et le passage du $2D$ au $3D$ est aussi plus aisé.

Cette méthode a inspiré les travaux de Ying *et al.* [127] [128] qui reprennent cette idée afin d'obtenir un algorithme multipôle rapide directement adaptable à plusieurs potentiels. En plus des potentiels électrostatiques et gravitationnels, solutions de l'équation de Laplace (ou de Poisson), leur formulation permet de traiter les solutions de l'équation instationnaire de Laplace, de l'équation (stationnaire ou instationnaire) de Stokes, et de l'équation (stationnaire ou instationnaire) de Navier, et ce en deux et en trois dimensions. Au contraire, dans le cas de la FMM basée sur les harmoniques sphériques, le changement de potentiel impose de revoir l'ensemble des formules et des développements utilisés. Les comparaisons de performances n'ont cependant être pu réalisées qu'en reprenant les temps CPU donnés par Cheng *et al.* [34] sur une machine différente.

La méthode d'Anderson a aussi donné lieu à la seule utilisation des routines BLAS, à notre connaissance, dans un algorithme de type FMM, par Hu & Johnsson [74].

2.3.3.9 Algorithme non uniforme

Contrairement à l'algorithme de Barnes & Hut qui s'adapte naturellement aux distributions non uniformes, une version *adaptative* de la FMM doit être élaborée afin de traiter efficacement les distributions non uniformes de particules. Dans cette section, nous allons présenter deux versions de la FMM adaptative.

La version adaptative originale. La première version adaptative de la FMM a été conçue par J. Carrier, L. Greengard et V. Rokhlin [32] puis améliorée par Cheng, L. Greengard et V. Rokhlin [34]. L'algorithme en $\mathcal{O}(N \log N)$ présenté simultanément dans [45] est basé sur les mêmes idées. La construction de l'octree suit la procédure donnée en section 2.3.1.1 pour des distributions non uniformes : chaque feuille avec un nombre de particules supérieur à un s_{max} fixé est divisée en huits fils, et ses particules sont distribuées entre ces fils. L'octree, non complet, obtenu au final possède donc des feuilles à différents niveaux, et ne contient aucune cellule vide. Deux cellules voisines ne sont donc plus forcément de même taille, et on définit alors la notion de *collègue* qui désigne des voisins immédiats de même taille. Ce type d'octree impose dans l'algorithme de la FMM 4 listes différentes pour chaque feuille, au lieu de 2 dans le cas uniforme (la liste d'interaction et la liste du calcul direct) ; elles sont représentées en figure 2.16 et décrites ci-dessous pour une cellule b donnée.

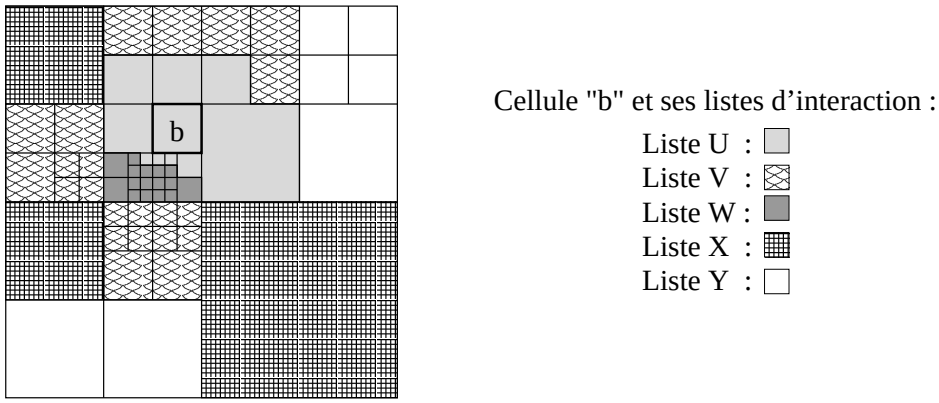


FIG. 2.16 – Listes d'interaction de la FMM adaptative pour un quadtree (d'après l'exemple donné dans [32]).

- La liste U (pour les feuilles uniquement) : si b est une feuille, la liste U contient b elle-même et toutes les feuilles qui sont des voisines immédiates de b. Cette liste correspond à la liste du calcul direct, et les interactions associées seront calculées par l'opérateur *P2P*.
- La liste V : que la cellule b soit une feuille ou une cellule interne, cette liste contient tous les fils des collègues du père de b qui sont bien séparés de b. Cette liste correspond à l'unique liste d'interaction définie dans le cas uniforme, et elle sera donc traitée par l'opérateur *M2L*.

- La liste W (pour les feuille uniquement) : si b est une feuille, cette liste contient tous les descendants des collègues de b dont les parents sont adjacents à b, mais qui ne sont pas eux-mêmes adjacents à b. b n'est pas bien séparée d'une cellule w de sa liste $W(b)$, mais on remarque que b est séparé de w par une distance supérieure ou égale à la longueur du côté de w. Le développement multipôle de w peut donc être évalué par l'opérateur *M2P* en chaque point de b en respectant la borne d'erreur de la FMM avec $ws = 1$ (voir section 2.3.3.3).
- La liste X qui est la liste duale de W : elle est formée de toutes les cellules c telles que $b \in W(c)$. Cette liste n'a donc pas besoin d'être construite explicitement : on se sert de la liste W. De façon réciproque par rapport à la liste W, on peut évaluer le champ créé par les particules d'une cellule $x \in X(b)$ à l'aide d'un développement local obtenu grâce à l'opérateur *P2L*. L'erreur alors générée en chaque point de b respecte la borne d'erreur de la FMM avec $ws = 1$.
- On peut enfin noter Y l'ensemble des cellules qui sont bien séparées du père de b. Les interactions associées aux éléments de Y sont déjà prises en compte dans le développement local du père de b que l'on translate par l'opérateur *L2L*, et il n'y a donc en pratique pas de liste Y à construire.

L'algorithme complet de la FMM adaptative s'obtient alors en intégrant ces nouvelles listes à la phase de descente et à celle du calcul direct. La valeur de s_{max} n'est pas précisée mais elle est généralement positionnée à quelques dizaines. Cette version originale a cependant été améliorée dans [34] : des tests supplémentaires permettent d'utiliser l'opérateur *P2P* à la place des opérateurs *M2P* ou *P2L* (pour les listes W et X) lorsque le nombre de particules est trop faible. En effet il est alors plus rapide d'effectuer le calcul direct que d'utiliser un développement multipôle ou local. La valeur du seuil pour ces tests est fixée à P^2, ce qui équilibre grossièrement le nombre théorique d'opérations de l'opérateur *P2P* et le nombre théorique d'opérations de l'opérateur *M2P* (ou *P2L*).

Le principal inconvénient de cette version originale est que, sans hypothèse sur la distribution des particules, trois des quatre listes ont des tailles non bornées : comme la hauteur de l'octree, ces tailles peuvent varier (en théorie) suivant un $\mathcal{O}(N)$, ce qui ne garantit plus la complexité linéaire de la FMM. Carrier, Greengard et Rokhlin s'appuyaient sur la précision machine (limitée) pour justifier une borne sur la hauteur de l'arbre et prouver ainsi la complexité linéaire de la FMM : cet argument a été critiqué par S. Aluru dans [8], et reconnu comme erroné par les auteurs dans [34]. A la place, la preuve de la complexité linéaire dans [34] a été renvoyée à l'article de Nabors *et al.* [93], mais comme nous le verrons au paragraphe suivant, l'algorithme utilisé par Nabors *et al.* n'est pas le même ! Cette méthode est néanmoins couramment utilisée [111] [115] [127].

La seconde version adaptative. Une approche différente a été proposée par W.T. Rankin [102] et Nabors *et al.* [93] : au lieu de fixer s_{max}, c'est la hauteur de l'octree qui est fixée. On considère donc que toutes les cellules de l'octree existent, mais il y a potentiellement de nombreuses cellules vides. La construction de l'octree suit la procédure donnée en section 2.3.1.1 pour des distributions uniformes, et la version uniforme de la FMM est alors appliquée, les cellules vides (cibles ou sources) étant omises. Comme exposé dans [92], cet algorithme peut effectuer, pour certaines distributions de particules, moins d'opérations que la version originale de la FMM adaptative présentée ci-dessus.

Cependant deux améliorations, apportées dans [93], rendent cet algorithme réellement

compétitif par rapport à la FMM adaptative de Cheng *et al.* [34].

– La notion de *cellule adaptative* (ou *adaptive cube* dans [93]) est tout d'abord introduite. Une cellule adaptative est soit une feuille, soit une cellule interne qui a plus d'un fils non vide (voir figure 2.17). Lorsqu'une cellule n'est pas adaptative, nous pouvons utiliser, sans aucune perte de précision, le développement multipôle de son unique fils pour représenter le potentiel dû aux particules qu'elle contient. Pendant la phase de remontée de la FMM, nous pouvons donc omettre l'opération *M2M* entre cette cellule et son fils. De façon similaire pendant la phase de descente, le développement local d'une cellule non adaptative n'a pas besoin d'être construit et nous pouvons établir directement, et sans perte de précision, le développement local de son unique fils, ce qui permet d'économiser une opération *L2L*. Plusieurs cellules non adaptatives consécutives forment une *chaîne* dans l'octree (voir figure 2.17), et plusieurs opérations *M2M* et *L2L* peuvent alors être évitées le long d'une telle chaîne.

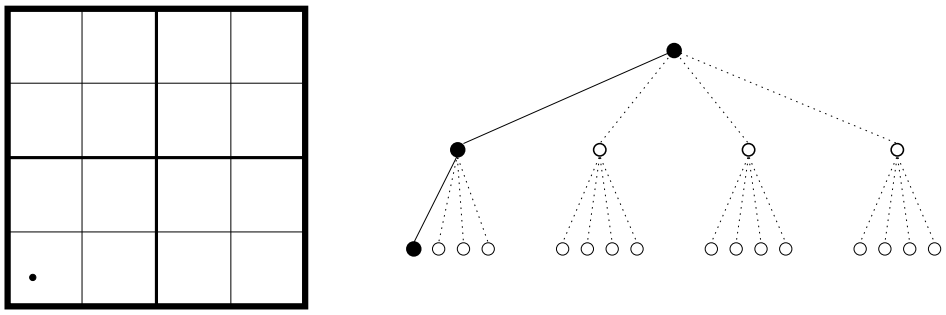


FIG. 2.17 – Exemple de cellules non adaptatives (en noir) et de chaîne. Les pointillés correspondent aux fils vides.

– La seconde amélioration concerne l'établissement d'un seuil sur le nombre de particules d'une cellule. Considérons une interaction entre deux cellules bien séparées ; si le nombre de particules dans la cellule source est inférieur à une valeur seuil, notée s_{min}, il est alors plus rapide d'utiliser directement, à la place de l'opérateur *M2L*, les particules de la cellule source afin de construire le développement local de la cellule cible, et ce grâce à l'opérateur *P2L*. De même, lorsque le nombre de particules de la cellule cible est inférieur à s_{min}, il est plus rapide d'évaluer directement le développement multipôle de la cellule source aux positions des particules de la cellule cible grâce à l'opérateur *M2P*, plutôt que d'utiliser l'opérateur *M2L*. Et quand les deux cellules cible et source ont moins de s_{min} particules, le calcul direct est utilisé. Dans [93], la valeur du seuil s_{min} est fixée au nombre de termes des développements. Attention, ces tests sur le nombre de particules par cellule ont ici bien plus d'impact que les tests inclus dans la version originale (améliorée par Cheng *et al.* [34]) : alors que dans la version de Cheng *et al.* le père de chaque feuille contient plus de s_{max} particules, ce qui limite en pratique la portée de ces tests, nous n'avons ici aucune contrainte sur le nombre de particules par feuille, et ce à tous les niveaux de l'octree.

Grâce à ces modifications, la complexité linéaire est garantie quelle que soit la distribution des particules, comme démontré dans [93].

Autres versions. On peut aussi noter que Hariharan, Aluru *et al.* [72] (voir aussi des mêmes auteurs : [9], [107], [71]) garantissent aussi un nombre théorique d'opérations linéaire quelle que soit la distribution grâce à une structure de données spéciale (*compressed octree*) qui compacte les chaînes. L'approche est donc similaire à celle de Nabors *et al.*, mais ici l'algorithme est intrinsèquement lié à la structure de données. Celle-ci est plus compliquée, de même que les accès aux cellules, leur création ou leur destruction. Son surcoût pour des distributions uniformes n'est en particulier pas étudié. De plus, la liste d'interaction est calculée de manière récursive, et conservée d'un pas de temps à l'autre : outre le surcoût mémoire, ceci n'est possible que pour des simulations où les particules sont immobiles (comme la simulation en électromagnétisme présentée dans [72]). En dynamique moléculaire et en astrophysique, la dynamicité des particules impose un recalcul de cette liste à chaque modification de l'octree.

2.3.3.10 Détermination de la liste d'interaction

La borne d'erreur de la FMM dépend de deux paramètres : le degré maximum P des développements et la « forme » de la liste d'interaction. Entre les deux extrêmes que constituent les critères de bonne séparation $ws = 1$ et $ws = 2$, des listes d'interactions intermédiaires ont été introduites : [27] [30] [51] [53] [78] [94]. Ces listes d'interactions, que nous qualifierons de *dynamiques*, sont généralement déterminées à partir du rayon r_A de la cellule cible A, du rayon r_B de la cellule source B, de la distance R entre les centres des deux cellules et d'un nouveau paramètre θ. La condition de bonne séparation proposée par exemple par W.D. Elliott dans [51] est alors : $\frac{r_A+r_B}{R} \leq \theta$. En pratique θ varie ici entre $0,5$ et $0,9$, et ce paramètre permet alors de contrôler plus finement la borne d'erreur de la FMM telle qu'elle est présentée dans [51].

On peut aussi remarquer que dans [30] ou [78] les rayons des cellules cubiques de l'octree ne sont pas ceux des sphères circonscrites, mais correspondent aux rayons des sphères englobant toutes les particules. Ceci permet de prendre en compte le contenu des cellules dans la condition de bonne séparation, à la manière de ce qui est réalisé dans les variantes de l'algorithme de Barnes & Hut (voir sections 2.3.2 et 2.3.3.13), même si ici les développements en harmoniques sphériques imposent toujours que la sphère soit concentrique à la cellule.

Deux autres améliorations ont été apportées au traitement de la liste d'interaction.

1. La première est le concept de *super-nodes* introduit par Zhao [129], aussi connu sous le nom de *parental conversion* dans la littérature, et repris par plusieurs auteurs [75] [27] [93]. Lorsqu'au sein de la liste d'interaction d'une cellule c au niveau l avec $ws = 2$, les huits fils d'une même cellule p du niveau $l - 1$ sont présents, on remplace les huits opérations *M2L* correspondantes par une seule opération *M2L* avec le développement multipole de p. Le nombre d'interactions passe alors de 875 ($ws = 2$) à 189. Cependant ceci accroît l'erreur engendrée par rapport au cas $ws = 2$ [51] [86], le rayon de la cellule du père étant plus important que celui de ses fils. A noter que le fait de considérer une liste d'interaction avec des cellules de niveaux différents se retrouve aussi dans les listes d'interaction dynamiques présentées ci-dessus.
2. Une autre amélioration, évoquée dans [106] et mise en œuvre dans [99], consiste à utiliser une plus petite valeur de P pour les cellules de la liste d'interaction qui sont éloignées de la cellule cible. Comme la distance entre la cellule cible et la cellule source

augmente, on peut se permettre d'utiliser de plus petites valeurs de P tout en conservant la même borne d'erreur. La relation entre la distance et la valeur de P doit cependant être déterminée de façon précise.

2.3.3.11 Etat de l'art de la parallélisation

Dans cette section, nous présentons un récapitulatif des différentes parallélisations de l'algorithme de la FMM. Outre une forte réduction des temps de calcul, ces parallélisations permettent aussi de traiter de plus grosses simulations : par exemple 512 millions de particules sur 512 processeurs en dynamique moléculaire dans [96] et 690 millions sur 3000 processeurs pour l'équation de Laplace dans [128]. L'algorithme de Barnes & Hut présentant certaines similarités avec la FMM, nous étudierons les techniques de parallélisation employées pour ces deux méthodes. Le principal problème consiste à obtenir une décomposition de l'espace en différents sous-domaines, chaque sous-domaine étant traité par un processeur donné, qui offre à la fois un bon équilibrage de charge et une bonne localité des données. La localité des données est en effet cruciale pour les opérations *M2M*, *M2L*, *L2L* et *P2P* (surtout avec le principe des interactions réciproques) qui nécessitent des données de la part de leur père, de leurs fils, ou de cellules situées dans leur « voisinage » (liste d'interaction et liste du calcul direct). On notera tout de même que dans l'algorithme de Barnes & Hut, l'entité élémentaire de parallélisme (c'est-à-dire la brique de base de la décomposition) est la particule, alors que pour la FMM c'est la cellule de l'octree.

Le schéma de communication de la FMM et de l'algorithme de Barnes & Hut ont été étudiés par Teng [118] : en théorie, le graphe des communications de la version adaptative de la FMM (proche de celle de Cheng *et al.* [34]), et donc a fortiori de la version uniforme, peut être partitionné de façon équilibrée et extensible. En pratique, si des parallélisation efficaces ont été réalisées dans le cas uniforme [86] [96] [102], la décomposition « optimale » est plus difficile à obtenir dans le cas non uniforme à cause de l'irrégularité de la distribution des particules, du caractère dynamique de la simulation sur plusieurs pas de temps (déplacement des particules), et éventuellement du caractère difficilement prévisible du schéma de communication [84] [110]. De plus, dans le cas de la FMM, les différentes phases de l'algorithme ont des coûts potentiellement différents, en fonction du nombre de particules par cellule et du nombre de développements, ce qui impliquerait une décomposition différente pour chaque phase [110].

Historiquement, les premières parallélisations de la FMM sont apparues peu de temps après son introduction [62] [129] [130], et étaient généralement destinées à des supercalculateurs parallèles spécifiques, tels le système Connection Machine (modèle CM-2) ou l'Encore Multimax 320. Parmi les premières parallélisations, on peut aussi citer [75], [95], puis [27] et [26]. Par la suite, de nouvelles techniques de parallélisation ont été conçues : parmi elles, deux types de décomposition ont été principalement utilisées.

Orthogonal Recursive Bisection Le premier type de décomposition, dit *Orthogonal Recursive Bisection* (ORB), a a été utilisé pour l'algorithme de Barnes & Hut [84] [120] et la FMM [78]. Comme représenté en figure 2.18, l'espace est découpé récursivement en deux suivant une dimension (on change de dimension à chaque fois) : on obtient ainsi un arbre binaire nommé *ORB-tree*. L'équilibrage de charge est réalisé en veillant à chaque fois à ce que la charge de calcul soit la même de chaque côté. Chaque processeur se voit ainsi attribuer une

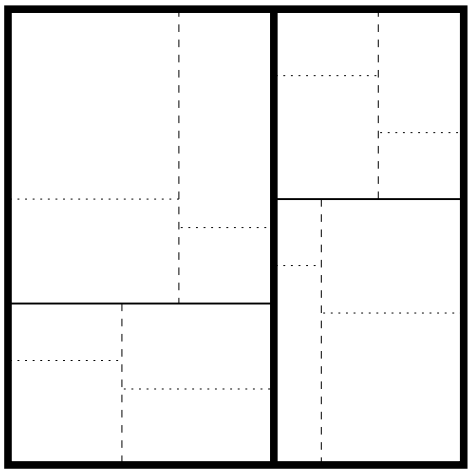


FIG. 2.18 – Exemple de décomposition de l'espace par la méthode dite *Orthogonal Recursive Bisection*.

portion de l'espace et on doit alors établir son *locally essential tree* (LET) [120] : cet octree, local à chaque processeur, contient toutes les données (développements et particules sources) permettant de calculer les interactions de toutes les particules cibles qu'il possède.

Liu & Bhatt [84] ont montré que la décomposition ORB pouvait être efficacement mise en œuvre (pour l'algorithme de Barnes & Hut) sur la Connection Machine CM-5 : ceci est notamment obtenu grâce à un modèle de communication à gros grain (les messages de petite taille vers un même destinataire sont rassemblés en un unique message afin d'amortir le coût de la latence du réseau) dirigé par l'émetteur (*sender-directed* ou *sender-driven*), selon lequel on envoie à l'avance les données partout où elles seront requises, plutôt que de demander ces données à chaque fois qu'elles sont nécessaires. Pour cela, le MAC utilisé (voir section 2.3.2) ne dépend que de la géométrie des cellules de l'octree, et non de leur contenu. Il est tout de même possible d'utiliser des MAC ou des listes d'interaction qui prennent en compte la position des particules dans chaque cellule (voir sections 2.3.2 et 2.3.3.10) avec la décomposition ORB, notamment grâce à la forme régulière des domaines obtenus (à savoir de forme rectangulaire $3D$, voir [78] [84].

Costzones et décomposition de Morton. Plutôt que de réaliser un nouveau partitionnement de l'espace comme dans la méthode ORB, on peut utiliser directement celui offert par l'octree. Ceci permet d'éviter d'avoir à gérer deux structures hiérarchiques différentes : l'octree et l'*ORB-tree*. Une telle décomposition a été introduite par Singh *et al.* [110] sous le nom de *costzones*, en utilisant une programmation pour des architectures à mémoire distribuée présentant un espace d'adressage commun (*CC-NUMA* pour *Cache Coherent - Non Uniform Memory Access*). Warren & Salmon [121] ont proposé une version utilisant une programmation par échanges de messages (*Message Passing Environment*) sur des architectures à mémoire distribuée, avec la structure de données *hashed octree* décrite en section 2.3.1.2.

Le principe de ces décompositions est le même : en se basant sur une *space-filling curve* (à savoir une courbe qui parcourt l'ensemble des cellules (ou des particules) de l'octree de manière à établir une représentation $1D$ de l'espace $3D$), on décompose l'espace en autant de

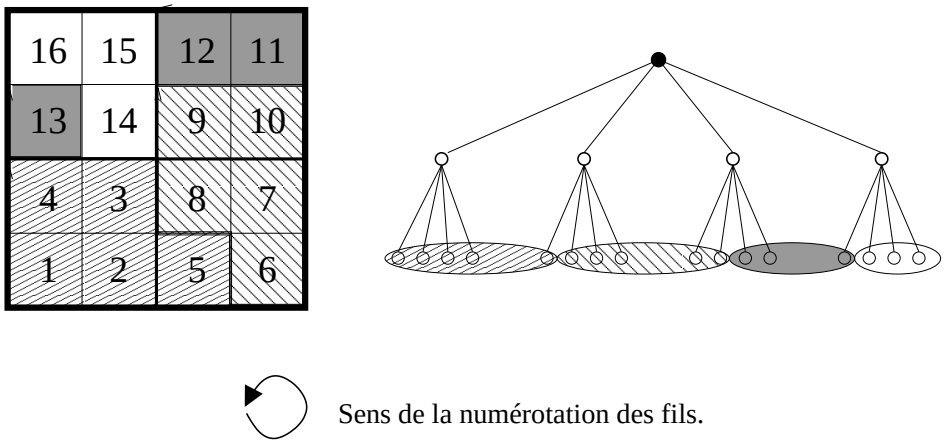
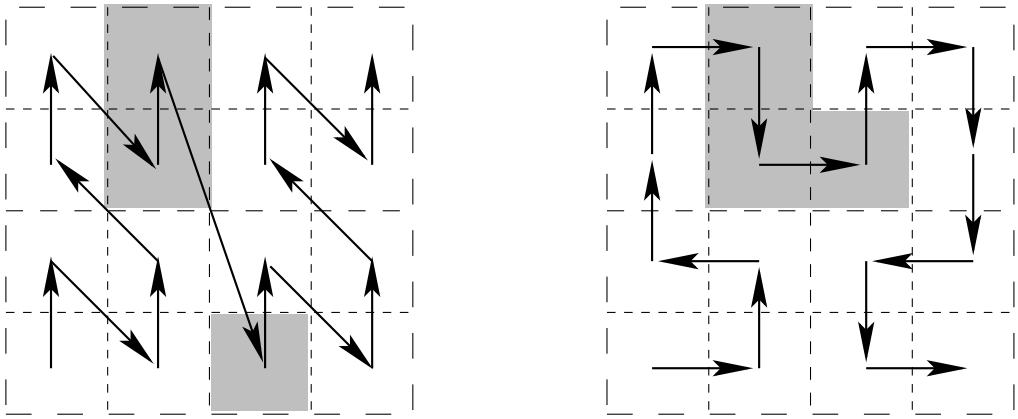


FIG. 2.19 – Exemple de décomposition de l'espace par la méthode *costzones* pour 4 processeurs (d'après le cours en ligne de James Demmel à l'Université de Berkeley : http://www.cs.berkeley.edu/~demmel/cs267/lecture27/lecture27.html).

paquets successifs qu'il y a de processeurs disponibles. Chaque paquet est ensuite affecté à un processeur qui traite toutes les cellules ou particules cibles qui s'y trouvent. La figure 2.19 donne un exemple de la décomposition *costzones* : la représentation plane de l'octree est ici induite par le sens de numérotation des fils d'une cellule interne. Mais on peut aussi utiliser l'ordre de Morton présenté en section 2.3.1.2 (voir par exemple [121]). Nous appellerons alors la décomposition ainsi obtenue *décomposition de Morton*.

Afin d'assurer l'équilibrage de charge entre les différents processeurs, la taille de chaque paquet est déterminée à l'aide d'une fonction de coût. Pour l'algorithme de Barnes & Hut (dans sa version améliorée par Warren & Salmon [121]), les particules cibles sont distribuées entre les différents processeurs : l'ordre de Morton porte donc sur les coordonnées des particules et un coût est associé à chaque particule. Dans la FMM, la fonction de coût porte d'abord sur les feuilles de l'octree, et l'ordre de Morton des indices des cellules est utilisé. Pour la distribution des cellules internes, il y a plusieurs possibilités : on peut attribuer chaque cellule interne au processeur qui possède son premier fils (comme dans DPMTA), au processeur qui possède la majorité de ses fils [111], ou encore effectuer un parcours infixe de l'octree après avoir attribué des coûts aux cellules internes [111]. La prise en compte des cellules internes vise en effet à améliorer l'équilibrage de charge sans introduire trop de communications supplémentaires pour les opérations *M2M* et *L2L*. Cette fonction de coût peut être basée sur des informations tirées du pas de temps précédent en prenant en compte le nombre d'interactions alors mises en jeu [110], ou les temps de calculs de chaque processeur [102]. En pratique, ces rééquilibrages n'ont pas lieu à chaque pas de temps, mais à intervalles réguliers, ou lorsque le déséquilibre introduit par la dynamicité de la simulation devient trop important.

Sing *et al.* ont ainsi montré que la décomposition *costzones* était moins coûteuse à établir et donnait de meilleurs résultats que la décomposition ORB sur des machines de type CC-NUMA avec des communications (implicites) à grain fin dirigées par le récepteur (*receiver-initiated* ou *request & reply*, par opposition aux communications dirigées par l'émetteur). De même, après avoir développé un premier code utilisant la décomposition ORB [120], Warren



(a) Morton. (b) Hilbert.

FIG. 2.20 – Décompositions ($2D$) de Morton et de Hilbert pour un quadtree de hauteur 2.

& Salmon ont adopté la décomposition de Morton [121] [122] avec les mêmes communications à grain fin dirigés par le récepteur, mais gérées cette fois de façon explicite. Cependant, Grama *et al.* [56] [57] ont montré qu'une variante de la décomposition de Morton (*Morton Ordering Based Scattered Decomposition*) pour l'algorithme de Barnes & Hut permet de décomposer des sous-domaines de l'espace en utilisant des communications à gros grain dirigées par l'émetteur. Enfin une autre variante de ces méthodes, où la décomposition est réalisée à un niveau intermédiaire de l'octree, a été proposée par Lu & Okunbor [87].

Décomposition de Hilbert. L'ordre de Morton, ou le sens de numérotation des fils présenté à la figure 2.19, peuvent cependant aboutir à un problème de localité des données. La figure 2.19 en donne un exemple pour le troisième paquet : les cellules de ce paquet ne sont en effet pas toutes contiguës, ce qui dégrade la localité des données au sein du processeur qui se voit attribuer un tel paquet. Afin d'assurer la contiguïté de toutes les cellules dans tous les paquets, et ce quelle que soit la distribution de particules et quel que soit le nombre de paquets (c'est-à-dire de processeurs), il faut utiliser une numérotation des fils non uniforme sur tout l'octree. Ceci a été proposé par Singh *et al.* [111], Warren & Salmon [121] et finalement implémenté pour la FMM dans le code DPMTA [101] [102] (et repris dans [79]) à l'aide de l'ordre de *Hilbert*. Comme le montre la figure 2.20(b), la *décomposition de Hilbert* qui est basée sur cet ordre garantit la contiguïté des cellules au sein de chaque paquet, ce qui n'est pas le cas de la décomposition de Morton représentée en figure 2.20(a). Dans l'ordre de Hilbert l'indice d'une cellule c n'est plus fonction des seules coordonnées du centre de c, mais aussi de l'indice du père de c. En d'autres termes, le sens de numérotation des fils d'une cellule c dépend du sens de numérotation utilisé au niveau de c, qui dépend lui-même du sens de numérotation utilisé au niveau du père de c, et ainsi de suite jusqu'à la racine.

Des tests sur une distribution uniforme de particules montrent alors que la décomposition de Hilbert appliquée à la FMM donne des résultats légèrement meilleurs pour un nombre de processeurs $\mathcal{P}$ supérieur à une douzaine, et une efficacité parallèle plus régulière lorsque $\mathcal{P}$ augmente [101]. A noter que cet ordre de Hilbert a aussi été utilisé pour l'algorithme de Barnes & Hut du code GADGET-2 [114] afin de distribuer les particules entre les processeurs (comme dans [121]).

Néanmoins, ces décompositions (*costzones*, décompositions de Morton et de Hilbert) nécessitent un parcours séquentiel de la *space-filling curve*. Si ceci est aisé en mémoire partagée (ou avec un espace d'adressage commun, comme pour *costzones*), ceci implique dans le cas d'une programmation par échanges de messages en mémoire distribuée une opération de réduction afin de rassembler sur un même processeur les coûts associés à toutes les feuilles (pour la FMM) ou à toutes les particules (pour l'algorithme de Barnes & Hut). Les communications ainsi requises peuvent être importantes. Dans [114], la liste des indices de Hilbert est découpée en segments afin de réduire le coût de cette décomposition.

Principe des interactions réciproques Lors de la phase de calcul direct, le principe des interactions réciproques impose un choix entre

- utiliser ce principe : deux communications non concomitantes sont alors nécessaires pour chaque opération *P2P* entre deux cellules distantes (envoi des particules à l'un des deux processeurs puis retour des résultats) [78] [59] ;
- ou renoncer à ce principe : il y a alors un surcoût par rapport à l'algorithme séquentiel (qui exploite toujours le principe) mais les deux communications nécessaires (une dans chaque sens) peuvent être effectuées en même temps [102].

Le choix dépend de la possibilité de recouvrir ou non les communications, et du réseau utilisé (vitesse, liaison *half-duplex* ou *full-duplex*).

Construction de l'octree La construction de l'octree, surtout lorsqu'elle est réalisée à chaque pas de temps, nécessite elle aussi une parallélisation. Chaque processeur construit son arbre local à partir des particules dont il dispose et une procédure de fusion est ensuite appelée [110] [121] [122]. Un pseudo octree global peut être utilisé : cette version compressée de l'octree permet d'obtenir le minimum d'informations distantes nécessaires [114] [128]. Cependant si l'octree est construit une seule fois au début, puis mis à jour entre deux pas de temps (à cause de la dynamicité de la simulation), cette phase devient moins critique.

Recouvrement des communications par le calcul Afin d'obtenir de hautes performances avec une programmation par échange de messages, il est impératif de recouvrir le temps nécessaire aux communications par des calculs (voir par exemple [79], [102], [121]). On utilise pour cela une programmation en mode *multi*-thread ou des communications non bloquantes [91]. Ceci nécessite aussi une réorganisation entre les différentes phases de l'algorithme [78] [102].

Pour de petites distributions de particules ($13K$ atomes, 4 niveaux dans l'octree) Kurzak & Pettitt [79] proposent de plus de scinder en deux la boucle pour le calcul *M2L* et le calcul direct : après avoir initialisé les communications, on effectue d'abord les opérations entre cellules locales, puis après la terminaison des communications, on traite les opérations mettant en jeu une cellule source distante et une cellule cible locale. A l'aide d'un ordonnancement des communications entre les processeurs pour éviter les congestions, les phases de calcul du champ proche et du champ lointain recouvrent ainsi leurs propres communications.

Autres techniques Des techniques d'équilibrage de charge dynamiques ont aussi été proposés : on pourra voir par exemple les travaux de Banicescu & Lu [17] avec la méthode dite de *fractiling* qui met en jeu un système de type maître-esclave, ou ceux de D.C. Gray [59] où

des régions de l'espace correspondant à un niveau intermédiaire de l'octree sont attribuées à la volée à un *pool* de processeurs.

Marzouk & Ghoniem [88] ont quant à eux présenté une décomposition originale, pour une variante de l'algorithme de Barnes & Hut, qui est basée sur la méthode dite de *k-means clustering* : une distribution irrégulière de particules est décomposée en k « grappes » (*cluster*) et pour chaque grappe, on construit un octree local qui est attribué à l'un des k processeurs. Des heuristiques sont utilisées pour obtenir un bon équilibrage de charge lors du choix des k grappes. Cette méthode améliore la localité des données et optimise le recours aux développements.

On peut aussi noter la parallélisation avec une technique *out-of-core* présentée dans [104] : cette technique consiste à stocker temporairement certaines données sur disque afin de pouvoir réaliser de très grosses simulations dont les besoins mémoire sont très importants.

Enfin, en ce qui concerne la méthode d'Anderson (voir section 2.3.3.8), Hu & Johnsson [74] présentent un parallélisme de données à l'aide du langage CMF (*Connection Machine Fortran*) sur le système Connection Machine CM-5/5E, et précisent notamment qu'il est préférable de distribuer le précalul des fonctions de transfert (voir section 2) et de communiquer les résultats, plutôt que de laisser chaque processeur les calculer toutes. Ying *et al.* [128] ont implémenté une version parallèle de leur méthode (basée sur celle d'Anderson, voir en section 2.3.3.8), dont la version adaptative est similaire à celle de Cheng *et al.* [34] : ils utilisent une décomposition de Morton (basée sur le nombre de particules), et des communications de type *all-to-all* afin de déterminer les différentes listes d'interactions. De très grosses simulations (690 millions de particules pour l'équation de Laplace) ont ainsi été réalisées avec un nombre de processeurs $\mathcal{P}$ atteignant 3000, mais pour $\mathcal{P} \geq 512$ l'équilibrage de charge n'est plus très bon, et les communications ainsi générées sont trop importantes. Le recouvrement des communications est lui efficacement réalisé d'une façon similaire à celle présentée dans [102] (voir ci dessus).

2.3.3.12 Améliorations existantes de l'opérateur multipôle-vers-local.

Comme remarqué en section 2.3.3.5, l'opérateur *M2L* représente la majeure partie du coût de calcul du champ lointain. Plusieurs améliorations ont donc été apportées afin d'accélérer le calcul de cet opérateur. Ces améliorations visent toutes à réduire sa complexité théorique en $\mathcal{O}(P^4)$. Elles sont généralement extensibles aux opérateurs *M2M* et *L2L*, dont les formulations sont proches, et dont la complexité théorique s'exprime aussi en $\mathcal{O}(P^4)$.

Nous allons ici présenter succintement ces améliorations, ainsi que leurs principales caractéristiques. Afin de la distinguer de ces améliorations, nous nommons l'implémentation originale de l'opérateur *M2L*, telle qu'elle est décrite en section 2.3.3.4 pour des noyaux de hauteurs simple et double, *M2L classique*. Cette implémentation prend bien sûr déjà en compte les optimisations présentées en section 2.3.3.6.

Transformée Rapide de Fourier. La première amélioration est due à l'utilisation d'une Transformée Rapide de Fourier (*Fast Fourier Transform*, ou *FFT*). Elle a été évoquée dans plusieurs papiers ([26], [52], [66], [106]), mise en œuvre à l'Université de Duke (Caroline Du Nord, USA) par William D. Elliott ([50] ou [51]), et implémentée dans le code DPMTA. En considérant les opérateurs *M2M*, *M2L* et *L2L* comme des convolutions (ou corrélations) entre deux suites périodiques, l'introduction de la FFT permet d'effectuer le calcul associé à

ces opérateurs en $\mathcal{O}(P^2 \log P)$. Cependant, afin de réduire les instabilités numériques introduites par la FFT, une version par blocs est nécessaire [50] : le coût du calcul s'exprime alors en $\mathcal{O}(P^3)$. Ce travail n'a été réalisé dans [50] que pour un noyau *M2L* de hauteur simple et la version par blocs implémentée dans DPMTA a restreint les valeurs possibles de P aux multiples de la taille des blocs.

Rotations. La seconde amélioration consiste à utiliser des *rotations* dans le calcul des opérateurs *M2M*, *M2L* et *L2L*. Ces rotations ont déjà été présentées dans plusieurs articles : voir [34], [36], [68], [70] et [125]. Cette amélioration, stable numériquement, permet d'effectuer le calcul des opérateurs *M2M*, *M2L* et *L2L* avec une complexité théorique en $\mathcal{O}(P^3)$. Nous présenterons en détails l'utilisation des rotations en section 4.3.

Ondes planes. Enfin, la dernière amélioration concerne l'introduction de développements en *ondes planes* (*plane wave expansions*), telle qu'elle a été présentée dans [68] pour la FMM uniforme, et dans [34] pour la FMM adaptative. Le gain en termes de complexité théorique est ici important, puisqu'on passe pour l'ensemble de la liste d'interaction de 189 opérations en $\mathcal{O}(P^4)$ pour le *M2L* classique, à 189 opérations en $\mathcal{O}(P^2)$ plus 20 opérations en $\mathcal{O}(P^3)$ dans le cas des ondes planes. De plus, la prise en compte de certaines symétries permet de n'effectuer que 40 des 189 opérations. Cependant il faut noter qu'en plus d'un surcoût mémoire, cette amélioration impose une corrélation forte entre l'erreur générée par les développements multipôles et locaux et l'erreur due à l'introduction des ondes planes : seules quelques précisions sont disponibles dans les articles [34] [68], et pour obtenir d'autres précisions il faut établir au préalable des tables de coefficients selon la procédure présentée par Yarvin, Rokhlin & Cheng [35] [126]. Si cette amélioration a été largement retenue dans le cadre des équations de Maxwell et de Helmholtz [40], où les valeurs de P sont très grandes (voir section 2.3.3.2), aucune autre publication et aucun autre code (excepté le code FMMPART3D développé par les auteurs de [34] et [68]), n'ont à notre connaissance repris l'usage des ondes planes pour la FMM dans le cadre de l'équation de Laplace (ou de Poisson). Ces ondes planes n'ont ainsi pas été comparées, en termes de performances, aux autres schémas de calcul présentés ci-dessus. L'utilisation des ondes planes constitue toujours un sujet de recherche très actif (voir [40]).

2.3.3.13 Méthodes hybrides

L'application de la FMM aux simulations astrophysiques n'est cependant pas satisfaisante : il apparaît même que la méthode de Barnes & Hut est alors plus efficace, comme ceci a été montré à partir d'estimations théoriques du nombre d'opérations dans [23], ou à partir de l'implémentation des deux méthodes dans [30] (avec toutefois un algorithme FMM particulier). Le treecode apparaît alors comme étant deux à quatre fois plus rapide que la FMM [30]. Ceci s'explique essentiellement par deux raisons : les développements en harmoniques sphériques de la FMM sont plus coûteux à manipuler que les développements multipôles utilisés dans la méthode de Barnes et Hut et dans ses variantes, et la FMM s'adapte moins efficacement aux distributions hautement non uniformes que l'algorithme de Barnes et Hut. Par contre, comme remarqué en section 2.3.2, il est tentant d'introduire dans l'algorithme de Barnes et Hut des interactions de type *cellule - cellule*, par l'intermédiaire de développements locaux et ainsi obtenir un algorithme de complexité linéaire.

Outre la méthode PMTA (*Parallel Multipole Tree Algorithm*) développée à l'Université de Duke [26] [27], ceci a été réalisé par W. Dehnen [42] [43] qui a ainsi élaboré une méthode hybride combinant les meilleurs aspects de l'algorithme de Barnes et Hut et de la FMM, et l'a implémenté dans un code nommé *falcON* (*Force Algorithm with Complexity* $\mathcal{O}(N)$). Ce code reprend l'algorithme de Barnes & hut amélioré par Warren & Salmon [122] en construisant des moments multipolaires basés sur le centre de masse de chaque cellule. Comme dans [122], des développements locaux sont introduits sous la forme de séries de Taylor en coordonnées cartésiennes, et comme pour les version adaptatives de la FMM, le calcul direct est préféré aux développements lorsqu'une cellule a trop peu de particules. L'ordre des développements[4] est fixé à 3, et pour les moments multipolaires le deuxième moment est nul par définition du centre de masse, alors que le dernier moment n'est pas construit car seule la force (c'est-à-dire l'accélération) est généralement calculée dans les simulations astrophysiques. Un paramètre θ contrôle la précision à travers un MAC (Multipole Acceptance Criterion) qui détermine si deux cellules sont bien séparées : en se basant sur le contenu des cellules (c'est-à-dire sur la distance maximale au centre de masse), on peut alors dans certains cas utiliser les développements entre deux voisins immédiats. La phase de descente est donc formulée de façon récursive (doublement récursive en fait, à la manière de l'algorithme proposé par Warren & Salmon [122]) afin de prendre en compte ce MAC et de traiter naturellement l'octree déséquilibré. Enfin le principe des interactions réciproques est intégré à l'expression des développements, ce qui permet de réduire sensiblement le coût de la construction des séries de Taylor.

En termes de performance, *falcON* est plus rapide d'un facteur au moins 10 par rapport au treecode original de Barnes et Hut, et en reprenant la même machine et les temps CPU donnés par Cheng *et al.* dans [34], *falcON* apparaît comme étant là aussi 10 fois plus rapide que la FMM pour une distribution uniforme de particules. Ce code semble donc être devenu la référence dans le domaine des simulations gravitationnelles, du moins pour une exécution séquentielle.

L'algorithme de Dehnen, les développements auquel il recourt, et son MAC basé sur le centre de masse, sont donc hautement optimisés pour les simulations astrophysiques et leurs faibles précisions. Comme pour l'algorithme de Barnes & Hut en section 2.3.2, les moments multipolaires et les séries de Taylor utilisés ne fournissent cependant pas d'expression analytique des développements en fonction de P : cette méthode hybride n'est donc pas extensible à d'autres types de simulation, comme la dynamique moléculaire, où les précisions requises sont plus importantes.

Enfin, une parallélisation de l'algorithme de Dehnen à l'aide de routines MPI a été réalisée par Londrillo *et al.* [85] sur un code en Fortran-90. Les résultats présentés sont cependant succints, et un doublement des communications est utilisé pour préserver le principe des interactions réciproques exploité dans l'algorithme de Dehnen. La gestion de la localité des données n'est pas précisée, de même que le schéma exact des communications, et il ne semble pas y avoir de recouvrement entre les calculs et les communications. Il est donc probable que, suivant le nombre de processeurs utilisés et suivant la distribution des particules visée, cette parallélisation ne soit pas toujours optimale. D'un point de vue plus général, il semble que l'algorithme utilisé dans *falcON*, en particulier le caractère dynamique du MAC et le parcours récursif qu'il induit, rende difficile la prévision des communications dans le

[4] Ceci correspond à $P = 3$ dans la FMM, et les équations (3) et (4) de [43] suggèrent une correspondance avec un noyau *M2L* de hauteur simple.

cadre d'une parallélisation sur des architectures à mémoire distribuée.

Chapitre 3

Positionnement et contributions

Notre étude porte sur l'algorithmique des problèmes à N-corps pour des simulations en astrophysique et en chimie moléculaire, et ce dans le cadre de l'équation de Poisson. Nous cherchons donc à concevoir et à implémenter une méthode capable de traiter efficacement ces deux domaines de simulations. Celle-ci devra permettre de traiter des distributions uniformes et non uniformes de particules, et ce pour des précisions basses (entre 10^{-2} et 10^{-3}) ou moyennes (entre 10^{-5} et 10^{-7}) ; les hautes précisions (de l'ordre de 10^{-12}) n'étant généralement pas considérées dans ces domaines. Chaque simulation correspond à un ensemble {distribution, précision} particulier qui devra dès lors être exécuté le plus efficacement possible. Comme ces simulations comportent de nombreux pas de temps, nous considérerons que la phase de la construction de l'octree est une étape initiale non critique car amortie sur l'ensemble des pas de temps : il suffit pour cela de pouvoir aisément mettre à jour l'octree après les (faibles) déplacements des particules. Nous nous intéressons ici uniquement au calcul des interactions entre les N corps, et par conséquent nous nous limiterons à l'étude d'un unique pas de temps.

Nous privilégions les méthodes hiérarchiques par rapport aux méthodes non hiérarchiques car elles sont plus adaptées aux distributions non uniformes. Au sein des méthodes hiérarchiques, la méthode retenue est la méthode « multipôle rapide » (*Fast Multipole Method*, ou FMM) car elle permet d'obtenir toutes les précisions possibles, alors que l'algorithme de Barnes & Hut, ses variantes ou la méthode hybride implémentée dans *falcON*, sont limités aux basses précisions.

Dans un premier temps, l'implémentation de la FMM uniforme suit la présentation qui en a été faite aux sections 2.3.3.2 à 2.3.3.6. Nous utilisons l'ordre de Morton pour tous les accès aux cellules, notamment ceux de la liste d'interaction et ceux de la liste du calcul direct. La structure de données proposée pour l'octree devra donc permettre l'accès à chaque cellule en fonction de son indice de Morton : pour cette FMM uniforme nous choisissons donc d'utiliser les tableaux unidimensionnels de pointeurs décrits en section 2.3.1.2. L'implémentation de la FMM que nous proposons est par ailleurs limitée aux conditions aux limites libres (c'est-à-dire sans conditions aux limites périodiques) et nous travaillons essentiellement avec un critère de bonne séparation $ws = 1$. A partir de cette première implémentation de la FMM, notre travail se décompose en trois partie.

Dans une première partie (partie II), nous allons comparer, pour des distributions uni-

formes calculées en séquentiel, plusieurs schémas de calcul pour accélérer l'opérateur *M2L* qui convertit les développements multipôles en développements locaux, et qui est de loin l'opérateur le plus utilisé dans le calcul du champ lointain.

Nous avons distingué en section 2.3.3.4 deux expressions différentes pour l'opérateur *M2L* : le noyau *M2L* de hauteur simple et celui de hauteur double. Nous allons tout d'abord prouver en section 4.1 qu'un opérateur *M2L* avec un noyau de hauteur simple respecte une borne d'erreur précise, similaire à celle présentée, mais non prouvée, dans [51].

Afin de réduire le coût de l'opérateur *M2L*, nous proposons de recourir aux routines BLAS (Basic Linear Algebra Subprograms) [46] qui sont des implémentations haute performance d'opérations en algèbre linéaire. Grâce à une prise en compte optimale des différentes couches de la hiérarchie mémoire, de sorte que les pipelines des unités de calcul flottant soient remplis au maximum, ces routines offrent en effet de substantiels gains en performance sur les architectures *superscalaires*. Cette accélération n'affecte que la constante du comportement asymptotique et nous conservons donc la complexité théorique en $\mathcal{O}(P^4)$ de l'opérateur *M2L*, mais comme par exemple en dynamique moléculaire P varie généralement de 3 à 15 (voir section 2.3.3.2), le gain offert par les routines BLAS peut dépasser celui obtenu par un schéma de calcul avec une plus faible complexité théorique.

Afin de valider notre approche par BLAS, nous avons aussi implémenté les améliorations par Transformée Rapide de Fourier (FFT) et par rotations, présentées brièvement en section 2.3.3.12. Ces deux améliorations seront donc détaillées aux section 4.2 et 4.3 pour des noyaux *M2L* de hauteurs simple et double, la FFT étant ainsi étendue au noyau *M2L* de hauteur double. Nous étudierons aussi précisément les instabilités numériques qui subsistent dans la version par blocs de la FFT.

L'introduction des routines BLAS nécessite une formulation matricielle de l'opérateur *M2L*, qui sera présentée, ainsi que son implémentation avec les routines BLAS, au chapitre 5 pour des noyaux *M2L* de hauteurs simple et double : les matrices creuses du noyau de hauteur simple nécessitent notamment un traitement particulier. Nous verrons de plus comment utiliser, grâce à des recopies supplémentaires, des routines BLAS de niveau 3 (produits matriciels) qui offrent de bien meilleures performances. Enfin, deux stockages de données spéciaux, inédits par rapport aux seuls travaux en la matière (Hu & Johnsson [74], voir section 2.3.3.8), permettront d'éviter le surcoût dû à ces recopies, en particulier pour les basses précisions.

Au chapitre 6, nous comparerons ensuite de façon détaillée les besoins mémoire, les précisions numériques et les temps de calcul de toutes ces approches pour des distributions uniformes calculées en séquentiel. Tous ces travaux étant réalisés pour des noyaux *M2L* de hauteurs simple et double, nous pourrons de plus établir une comparaison en pratique du rapport précision / temps de calcul pour ces deux noyaux, et déterminer ainsi lequel est le plus efficace.

Enfin au chapitre 7, après une brève description de l'introduction des ondes planes dans le calcul *M2L*, nous comparerons notre approche BLAS au code FMMPART3D qui implémente ces ondes planes. Ceci constitue la première comparaison de la FMM avec ondes planes avec un autre schéma de calcul *M2L*, les articles [68] et [34] se contentant de la comparer au calcul direct. Nous allons en particulier pouvoir étudier l'importance du facteur constant sous-jacent à la complexité théorique en $\mathcal{O}(P^3) + \mathcal{O}(P^2)$ du calcul *M2L* avec les ondes planes pour la gamme de valeurs de P qui nous intéressent en dynamique moléculaire.

A la fin de cette partie, nous disposerons donc d'une FMM uniforme particulièrement

efficace et adaptée aux distributions uniformes de particules souvent rencontrées en dynamique moléculaire. Le code implémentant cette FMM, ainsi que ses versions ultérieures, est appelé FMB pour *Fast Multipole with BLAS* (ou *Fast Multipole in Bordeaux* ...) ; ce code est écrit en langage C.

La partie III sera consacrée à la version adaptative de la FMM. En ce qui concerne le choix entre les deux versions présentées en section 2.3.3.9, nous choisissons de retenir celle de Nabors *et al.* [93]. La taille non bornée des listes d'interactions de la version originale de la FMM adaptative de Cheng *et al.* [34] est en effet problématique. Qu'on choisisse de les conserver d'un pas de temps à l'autre, ou de les (re)calculer à chaque pas de temps, ces listes non bornées induiront un surcoût, soit en termes d'espace mémoire, soit en termes de temps CPU[1]. De plus, et c'est un souci majeur, ces listes non bornées aboutiront à des schémas de communication difficilement prévisibles lors d'une implémentation parallèle de la FMM sur des architectures à mémoire distribuée. Dans le cas de la version de Nabors *et al.*, outre la garantie de la complexité linéaire, nous conservons les deux listes de l'algorithme uniforme, de tailles bornées et facilement calculables, ce qui induira, dans le cadre de calculs parallèles en mémoire distribuée, des schémas de communication plus réguliers et plus prévisibles.

De plus, dans le cas de la FMM adaptative originale de Cheng *et al.* [34], les listes d'interactions dynamiques présentées en section 2.3.3.10, conduisent à une formulation récursive de la phase de descente de la FMM : une formulation itérative imposerait de stocker, au cours de la descente, pour chaque cellule c, toutes les cellules qui n'ont pas été considérées comme bien séparées par rapport au père de c. Or ces formulations récursives ne sont pas adaptées à une parallélisation en mémoire distribuée. Ce n'est pas le cas avec la version adaptative de Nabors *et al.* [93] : comme dans le code DPMTA [51], et du fait que toutes les cellules existent (ce qui n'est pas le cas dans la version de Cheng *et al.* [34]), la liste d'interaction a la même forme à tous les niveaux, et donc en particulier lorsqu'on passe du niveau du père à celui du fils. Il est donc aisé de connaître les cellules qui n'ont pas été considérées comme bien séparées du père de c. Même si ces listes d'interactions dynamiques n'ont pas été implémentées (nous utilisons $ws = 1$), cette possibilité reste donc ouverte avec le choix de la version de Nabors *et al.*.

Pour ces mêmes raisons (formulation récursive de la liste d'interaction), l'algorithme et la structure de données de Hariharan, Aluru *et al.* [72] (voir section 2.3.3.9) ont aussi été écartés. De plus, la même garantie de complexité linéaire est apportée par l'algorithme de Nabors *et al.* sans recours à une structure de données plus compliquée, et donc potentiellement plus coûteuse.

Enfin, un autre avantage moins perceptible peut être évoqué. Lors de la construction de l'octree, nous ne contrôlons absolument pas l'orientation du découpage de l'espace par rapport à la distribution des particules : il est tout à fait possible qu'une légère translation ou rotation des coordonnées des particules modifie sensiblement la répartition des particules dans les cellules. Or ce problème est bien plus sensible dans la version adaptative de Cheng *et al.*, notamment en ce qui concerne la hauteur de chaque feuille et la taille des listes U, W et X. Dans la version de Nabors *et al.*, la charge de calcul par cellule devrait par contre rester beaucoup plus stable. Et là encore, ce problème est amplifié lorsqu'on considère la prévision et la gestion du coût des communications pour une parallélisation en mémoire distribuée.

Afin d'implémenter la version adaptative de Nabors *et al.*, nous commencerons par approfondir la notion de seuil s_{min} en section 8.1 et nous donnerons alors l'algorithme de notre

[1]Dans ce document nous désignons par « temps CPU » les temps réels d'exécution (*wallclock time*).

version adaptative de la FMM. Puis nous aurons besoin d'une structure de données adaptée pour l'octree. En effet, comme exposé en section 2.3.1.2, ni l'implémentation d'un octree par des tableaux unidimensionnels de pointeurs, ni l'implémentation à l'aide d'un *hashed octree* ne sont adaptées à tous les types de distributions. Nous introduirons donc en section 8.2 une nouvelle structure de données, *l'octree avec indirection*, qui permet de traiter efficacement les distributions uniformes comme les distributions non uniformes, et ce sans surcoût notable.

Nous adapterons ensuite notre formulation matricielle à la FMM adaptative ; celle-ci étant plus efficace à l'intérieur des zones uniformes, nous présenterons en section 8.3 un algorithme permettant de détecter les plus grandes zones uniformes au sein de distributions non uniformes de particules.

Après une validation en pratique de ces contributions en section 8.4, nous montrerons en section 8.5 l'intérêt, dans le cadre des distributions non uniformes, de notre version par BLAS par rapport aux autres schémas de calcul *M2L* (calcul *M2L* classique, par FFT, par rotations et par ondes planes). Enfin, dans le cas particulier des simulations astrophysiques, nous présenterons en section 8.6 une comparaison entre notre code FMB et des codes de référence dans le domaine : le *treecode* de Barnes & Hut, GADGET-2 et *faclON*.

Dans la dernière partie de ce document (partie IV), nous traiterons de la parallélisation de la FMM. Les architectures parallèles visées sont essentiellement les grappes de SMP (*symmetric multiprocessors*). Ces systèmes sont formés de nœuds multi-processeurs en mémoire partagée où tous les processeurs peuvent partager le même espace d'adressage avec des temps d'accès uniformes à la mémoire (*uniform memory access*). Ces nœuds sont interconnectés par un réseau rapide de type Myrinet, Quadrics, Infiniband, Federation *etc.*, qui apporte un aspect mémoire distribuée. Les gammes SP3, SP4 et SP5 d'IBM en sont des exemples.

Deux types de parallélisation sont alors possibles sur de telles architectures. La première est une approche multi-*thread* où différents *threads* s'exécutent en parallèle sur un même espace d'adressage, au sein d'un unique processus. Cette approche peut être mise en œuvre sur un nœud SMP en *mémoire partagée*, mais aussi sur des architectures à mémoire distribuée présentant un espace d'adressage commun (*CC-NUMA* pour *Cache Coherent - Non Uniform Memory Access*). La deuxième approche est une parallélisation multi-processus où différents processus s'exécutent en parallèle, sur un même nœud ou sur plusieurs nœuds, et s'échangent des données lors de phases de communication. Cette approche requiert une programmation par échange de messages (*Message Passing Environment*). Nous allons mettre en œuvre ces deux approches en utilisant les *threads* POSIX pour la première et la bibliothèque de communication MPI [91] pour la seconde. Comme nous nous sommes limités à l'étude d'un pas de temps, nous nous concentrons ici sur des équilibrages de charge au sein du pas de temps, et nous ne nous baserons donc pas sur des équilibrages de charge établis à partir du pas de temps précédent (voir section 2.3.3.11).

Pour la partie multi-*thread* présentée en section 9.1, nous mettrons en place, contrairement aux décompositions similaires déjà réalisées telle *costzones* (voir section 2.3.3.11), des décompositions spécifiques à chaque phase de l'algorithme. Ces décompositions seront basées sur l'ordre de Morton ou de Hilbert, et seront réalisées indépendamment à chaque niveau de l'octree afin d'optimiser l'équilibrage de charge. Nous ne retenons pas ici les équilibrages de charge dynamiques, et nous employons un équilibrage de charge statique car la seule connaissance de l'octree fournit suffisament d'information : la présence du développement multipôle et du développement local, ou le nombre de particules, pour une cellule c, ainsi que pour les membres de sa liste d'interaction et de sa liste du calcul direct,

doit suffire à estimer la charge de calcul requise par les différentes phases de la FMM pour cette cellule c. Nous verrons aussi comment exploiter le principe des interactions réciproques pour l'opérateur *P2P* en minimisant les accès concurrents aux données partagées, comment différer le traitement de ces accès concurrents et de dépendances temporelles, et comment mettre en œuvre une version multi-*thread* de notre schéma de calcul *M2L* par BLAS. Cette version multi-*thread* sera validée par des tests jusqu'à 16 ou 32 processeurs, et nous présenterons alors une comparaison entre notre code FMB multi-*thread* et le code séquentiel *falcON* qui prolongera la comparaison effectuée en section 8.6 dans le cadre des simulations astrophyiques.

Pour la partie multi-processus présentée en section 9.2, nous privilégions une décomposition de Morton ou de Hilbert à une décomposition ORB afin de conserver le découpage de l'espace réalisé par l'octree et de mieux s'adapter aux distributions astrophysiques hautement non uniformes. Pour obtenir de hautes performances avec une programmation par échange de messages, nous mettrons en place (d'après l'étude réalisée en section 2.3.3.11) les techniques suivantes : recouvrement des communications par du calcul, groupement des envois de petits messages, ordonnancement des communications entre les processeurs, et utilisation de communications dirigées par l'émetteur plutôt que par le récepteur. Le recouvrement des communications par du calcul sera obtenu grâce à un entrelacement des différentes phases de calcul et de communication et grâce à un *thread* supplémentaire dédié à la réception des messages. Le regroupement et l'ordonnancement des messages en fonction de leur destinataire sera réalisé grâce à une détection des cellules impliquées dans les communications. Enfin, la prévision des communications par l'émetteur, compliquée par la forme irréguliaire des domaines obtenus par les décompositions de Morton ou de Hilbert, ne peut être obtenue que par une liste d'interaction uniquement basée sur la géométrie des cellules de l'octree : les listes d'interaction basées sur le contenu des cellules (voir section 2.3.3.10) sont donc écartées. De plus, comme expliqué ci-dessus, le choix de la version adaptative de Nabors *et al.* [93] plutôt que celle de Cheng *et al.*, nous permet justement d'avoir au niveau de la liste d'interaction (et de celle du calcul direct) des schémas de communication réguliers et prévisibles, et donc adaptés à des communications dirigées par l'émetteur. Par ailleurs, nous détaillerons le traitement spécial des communications nécessaires aux opérations *M2M* et *L2L*, et nous indiquerons les conséquences de cette parallélisation en mode multi-processus sur notre schéma de calcul *M2L* par BLAS. Cette version multi-processus sera validée par des tests jusqu'à 128 processeurs, et nous présenterons une comparaison entre les efficacités parallèles de la décomposition de Morton et de celle de Hilbert pour diverses distributions de particules (la comparaison présentée dans [101] est uniquement basée sur une distribution uniforme). Les limites de cette version seront analysées et nous justifierons le choix d'un couplage MPI-*threads* comme perspective afin de lever ces limites.

En conclusion, nous présenterons un bilan de nos contributions et nous ouvrirons plusieurs perspectives permettant d'étendre ou d'améliorer nos résultats.

Deuxième partie

Schémas de calcul multipôle-vers-local pour la FMM uniforme

Chapitre 4

Opérateur Multipole vers Local : analyse de la borne d'erreur et améliorations existantes

Sommaire

Dans cette partie, nous allons comparer, pour des distributions uniformes calculées en séquentiel, les différents schémas de calcul permettant d'accélérer l'opérateur *M2L* qui représente la majeure partie du temps de calcul du champ lointain. Au préalable, nous prouverons qu'un opérateur *M2L* avec un noyau de hauteur simple respecte, au contraire d'un opérateur *M2L* avec un noyau de hauteur double, une borne d'erreur précise, ce qui justifie

son implémentation. Cette comparaison de méthodes a donc été effectuée pour des noyaux de hauteurs simple et double, et elle a nécessité l'implémentation complète des améliorations par Transformée Rapide de Fourier et par rotations. Après avoir présenté notre nouvelle approche utilisant les routines BLAS, nous comparerons ces différentes méthodes, ainsi que les noyaux de hauteurs simple et double. Pour terminer, nous décrirons brièvement l'amélioration basée sur les ondes planes et nous effectuerons une comparaison supplémentaire entre notre schéma de calcul *M2L* par BLAS et cette amélioration, en confrontant directement notre code FMB au code FMMPART3D.

Dans la suite, le calcul associé à l'opérateur *M2L* sera communément appelé *calcul* M2L.

4.1 Analyse de la borne d'erreur

Comme annoncé au chapitre 3, nous allons ici prouver qu'un opérateur *M2L* avec un noyau de hauteur simple, contrairement à un opérateur *M2L* avec un noyau de hauteur double, respecte une borne d'erreur précise.

Afin de prouver cette borne d'erreur, nous étudions le potentiel au point $\mathbf{Z}$ dû à une unique charge unitaire localisée au point $\mathbf{Q}$, comme représenté en figure 4.1. Comme expliqué en sections 2.3.3.4 et 2.3.3.3, et aussi par Petersen *et al.* [98], ce cas d'étude est suffisant pour étudier la borne d'erreur de la FMM car les opérateurs *M2M* et *L2L* n'influent pas sur cette borne d'erreur. Nous notons $\mathcal{B}_1$ (respectivement $\mathcal{B}_2$) la boule centrée en $\mathbf{X_1}$ de rayon r_1

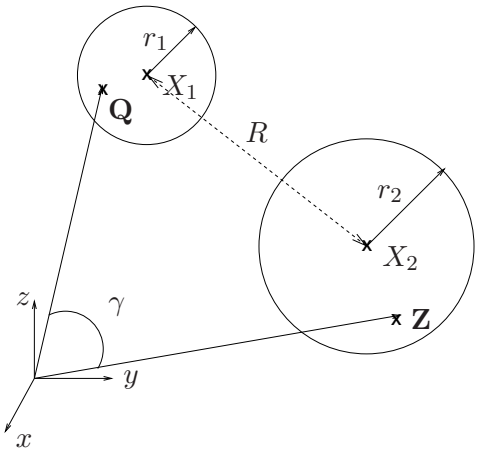


FIG. 4.1 – La configuration de notre cas d'étude.

(respectivement X_2 et r_2). On considère $\mathbf{Q}$ inclus dans $\mathcal{B}_1$, et $\mathbf{Z}$ dans $\mathcal{B}_2$. De plus, en notant $R = \|\mathbf{X_1} - \mathbf{X_2}\|$, nous supposons que : $R > r_1 + r_2$.

Nous commençons par procéder comme dans [60]. Soient γ l'angle entre les vecteurs $\mathbf{Q}$ et $\mathbf{Z}$, $r_>$ (respectivement $r_<$) le maximum (respectivement minimum) entre la norme de $\mathbf{Q}$ et celle de $\mathbf{Z}$, nous reprenons la décomposition du potentiel en polynômes de Legendre telle qu'elle est donnée à l'équation (2.3) :

$$\Phi(\mathbf{Z}) = \frac{1}{\|\mathbf{Z} - \mathbf{Q}\|} = \sum_{n=0}^{+\infty} \frac{r_<^n}{r_>^{n+1}} P_n(\cos\gamma). \tag{4.1}$$

Nous définissons $\Phi_P(\mathbf{Z})$ comme étant le potentiel tronqué à $n = P$, et nous avons alors :

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| = \sum_{n=P+1}^{+\infty} \frac{r_<^n}{r_>^{n+1}} P_n(\cos\gamma).$$

Comme pour tout $n \geq 0$ et pour tout $x \in [-1, 1]$, nous avons $|P_n(x)| \leq 1$ et nous pouvons écrire :

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \sum_{n=P+1}^{+\infty} \frac{r_<^n}{r_>^{n+1}} = \frac{1}{r_> - r_<} \left(\frac{r_<}{r_>}\right)^{P+1}. \tag{4.2}$$

Ceci nous amène à la borne d'erreur suivante :

Proposition 4.1. *$\forall\, \mathbf{Q} \in \mathcal{B}_1$ et $\forall\, \mathbf{Z} \in \mathcal{B}_2$ sous la condition $R > r_1 + r_2$, nous avons :*

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \frac{1}{R - (r_1 + r_2)} \left(\frac{r_1 + r_2}{R}\right)^{P+1}.$$

Démonstration. Comme $\|\mathbf{Z} - \mathbf{Q}\| = \|\left((\mathbf{Z} - \mathbf{X_2}) + (\mathbf{X_1} - \mathbf{Q})\right) + (\mathbf{X_2} - \mathbf{X_1})\|$, ceci découle directement de l'inégalité (4.2). □

Nous allons maintenant décomposer le potentiel de la même façon que pour la FMM, tout en maintenant cette borne d'erreur, et nous allons prouver le théorème suivant.

Théorème 5. *L'opérateur* M2L **avec un noyau *M2L* de hauteur simple** *respecte strictement la borne d'erreur de la proposition 4.1.*

Démonstration. Afin de prouver le théorème 5, nous considérons un développement multipôle centré en $\mathbf{X_1}$, et un développement local centré en $\mathbf{X_2}$, pour exprimer le potentiel en $\mathbf{Z}$ dû à la particule en $\mathbf{Q}$. Nous commençons par :

$$\Phi(\mathbf{Z}) = \frac{1}{\|\mathbf{Z} - \mathbf{Q}\|} = \frac{1}{\|\left(\mathbf{X_2} - \mathbf{X_1}\right) - \left((\mathbf{Q} - \mathbf{X_1}) - (\mathbf{Z} - \mathbf{X_2})\right)\|}.$$

Comme la condition $R > r_1 + r_2$, qui garantie à la fois la convergence des développements multipôles et celle des développements locaux, implique $\|(\mathbf{Q} - \mathbf{X_1}) - (\mathbf{Z} - \mathbf{X_2})\| < \|\mathbf{X_2} - \mathbf{X_1}\|$, nous pouvons écrire à l'aide du théorème 2 :

$$\Phi(\mathbf{Z}) = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} (-1)^n I_n^{-l}\left((\mathbf{Q} - \mathbf{X_1}) - (\mathbf{Z} - \mathbf{X_2})\right) O_n^l\left(\mathbf{X_2} - \mathbf{X_1}\right).$$

Et comme $I_n^l(-\mathbf{X}) = (-1)^n I_n^l(\mathbf{X})$, nous avons :

$$\Phi(\mathbf{Z}) = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} I_n^{-l}((\mathbf{Z} - \mathbf{X_2}) - (\mathbf{Q} - \mathbf{X_1}))O_n^l(\mathbf{X_2} - \mathbf{X_1}).$$

Nous utilisons alors le théorème de translation de Laplace pour *Inner-to-Inner* (théorème 4 présenté en section 2.3.3.2), qui ne requiert aucune condition sur $\mathbf{X}$ et $\mathbf{X}'$, et nous obtenons :

$$\Phi(\mathbf{Z}) = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} \sum_{j=0}^{n} \sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1}) I_{n-j}^{-l-k}(\mathbf{Z} - \mathbf{X_2}) O_n^l(\mathbf{X_2} - \mathbf{X_1}). \tag{4.3}$$

Nous soulignons ici que l'équation (4.3) n'est qu'une réécriture de l'équation (4.1) : c'est pourquoi, lorsqu'on tronque la série sur n dans l'équation (4.3) à $n = P$, nous obtenons la même borne d'erreur qu'à la proposition 4.1. Nous avons donc :

$$\Phi_P(\mathbf{Z}) = \sum_{n=0}^{P} \sum_{l=-n}^{n} \sum_{j=0}^{n} \sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1}) I_{n-j}^{-l-k}(\mathbf{Z} - \mathbf{X_2}) O_n^l(\mathbf{X_2} - \mathbf{X_1}).$$

Nous pouvons inverser les deux sommes finies sur les degrés n et j ainsi :

$$\Phi_P(\mathbf{Z}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} \sum_{n=j}^{P} \sum_{l=-n}^{n} (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1}) I_{n-j}^{-l-k}(\mathbf{Z} - \mathbf{X_2}) O_n^l(\mathbf{X_2} - \mathbf{X_1}).$$

En posant tout d'abord $n' = n - j$, nous obtenons :

$$\Phi_P(\mathbf{Z}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} \sum_{n'=0}^{P-j} \sum_{l=-(n'+j)}^{n'+j} (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1}) I_{n'}^{-l-k}(\mathbf{Z} - \mathbf{X_2}) O_{n'+j}^l(\mathbf{X_2} - \mathbf{X_1}),$$

puis avec $l' = l + k$:

$$\Phi_P(\mathbf{Z}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} \sum_{n'=0}^{P-j} \sum_{l'=-(n'+j)+k}^{n'+j+k} (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1}) I_{n'}^{-l'}(\mathbf{Z} - \mathbf{X_2}) O_{n'+j}^{l'-k}(\mathbf{X_2} - \mathbf{X_1}).$$

Nous rappellons ici que $I_{n'}^{-l'}(\mathbf{x})$ impose $-n' \leq l' \leq n'$. De la même façon, $I_j^k(\mathbf{x})$ impose $|k| \leq j$, c'est-à-dire $j + k \geq 0$, et $k - j \leq 0$. De plus :

$$\begin{array}{rcl} j + k \geq 0 & \Rightarrow & n' + j + k \geq n', \\ k - j \leq 0 & \Rightarrow & -(n' + j) + k = -n' + (k - j) \leq -n'. \end{array}$$

Dans ces conditions nous avons l'égalité suivante :

$$\sum_{l'=-(n'+j)+k}^{n'+j+k} = \sum_{l'=-n'}^{n'}.$$

Ce qui signifie : l'ensemble des termes indexés par la somme du membre droit de l'égalité est inclus dans l'ensemble des termes indexés par la somme du membre gauche, et tous les termes présents dans le membre gauche, mais pas dans le membre droit, sont nuls.

Avec $l = -l'$ et $n = n'$, nous avons finalement :

$$\Phi_P(\mathbf{Z}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} \sum_{n=0}^{P-j} \sum_{l=-n}^{n} (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1}) I_n^l(\mathbf{Z} - \mathbf{X_2}) O_{n+j}^{-l-k}(\mathbf{X_2} - \mathbf{X_1}).$$

En remarquant que :

$$\sum_{j=0}^{P} \sum_{n=0}^{P-j} \equiv \sum_{\substack{|j| \leq P \\ |n| \leq P, \\ 0 \leq n+j \leq P}} \equiv \sum_{n=0}^{P} \sum_{j=0}^{P-n}$$

nous obtenons :

$$\Phi_P(\mathbf{Z}) = \sum_{n=0}^{P} \sum_{l=-n}^{n} \left(\sum_{j=0}^{P-n} \sum_{k=-j}^{j} (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1}) O_{n+j}^{-l-k}(\mathbf{X_2} - \mathbf{X_1}) \right) I_n^l(\mathbf{Z} - \mathbf{X_2}).$$

Pour tout $\mathbf{Q} \in \mathcal{B}_1$, les termes du développement multipôle M_j^k, $\forall\, (j,k) \in \mathbb{N} \times \mathbb{Z}$ avec $0 \leq |k| \leq j$, étant définis par : $M_j^k = (-1)^j I_j^k(\mathbf{Q} - \mathbf{X_1})$, nous venons de prouver que

$$\Phi_P(\mathbf{Z}) = \sum_{n=0}^{P} \sum_{l=-n}^{n} L_n^l I_n^l(\mathbf{Z} - \mathbf{X_2}),$$

où L_n^l désignent les termes du développement local centré en $\mathbf{X_2}$ dû à une charge unitaire localisée en $\mathbf{Q}$, et obtenus grâce à l'opérateur *M2L*, avec un noyau *M2L* de hauteur simple, appliqué aux termes du développement multipôle M_j^k ainsi

$$L_n^l = \sum_{j=0}^{P-n} \sum_{k=-j}^{j} M_j^k O_{n+j}^{-l-k}(\mathbf{X_2} - \mathbf{X_1})$$

□

Il faut noter que la condition de bonne séparation, avec $ws = 1$ ou $ws = 2$, est en fait plus stricte que $R > r_1 + r_2$, et ce dans le but d'assurer une convergence assez rapide. Avec $ws = 1$ par exemple, nous avons pour une cellule de longueur l : $r_1 = r_2 = \frac{\sqrt{3}}{2}$ et $R \geq 2$. D'où :

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \frac{1}{R - (r_1 + r_2)} \left(\frac{\sqrt{3}}{2} \right)^{P+1} = \frac{1}{R - (r_1 + r_2)} (0,866)^{P+1}.$$

Avec $ws = 2$ ($R \geq 3$), nous obtenons :

$$\|\Phi(\mathbf{Z}) - \Phi_P(\mathbf{Z})\| \leq \frac{1}{R - (r_1 + r_2)} \left(\frac{\sqrt{3}}{3} \right)^{P+1} = \frac{1}{R - (r_1 + r_2)} (0,577)^{P+1}.$$

Cette borne d'erreur est plus stricte que celle présentée en section 2.3.3.3 qui était : $\left(\frac{r_2}{R - r_1}\right)^{P+1}$ ($ws = 1 \Rightarrow (0,764)^{P+1}$ et $ws = 2 \Rightarrow (0,406)^{P+1}$).

En conclusion, puisque l'erreur exprimée à la proposition 4.1 est plus grande que celle due au développement multipôle (voir en section 2.3.3.3), et puisqu'aucune erreur supplémentaire n'est introduite par les opérateurs *M2M* et *L2L* (voir là aussi en section 2.3.3.3), la borne d'erreur de la FMM avec un noyau de hauteur simple pour l'opérateur *M2L* est par conséquent celle de la proposition 4.1.

En conséquence, cette borne d'erreur peut être utilisée de façon sûre dans le cas d'un noyau de hauteur simple pour déterminer la valeur de P en fonction de la précision requise. Comme expliqué en section 2.3.3.4, une telle borne n'a pas été prouvée pour le noyau de hauteur double. Nous pouvons seulement affirmer que le noyau de hauteur double respecte au moins la borne d'erreur du noyau de hauteur simple, et qu'il est même plus précis car il ajoute des termes à l'expression de l'opérateur *M2L*, avec toutefois un surcoût au niveau

du temps de calcul. Le problème est que nous ne pouvons pas prévoir ce gain dans la précision de la FMM, et nous ne pouvons par conséquent pas comparer les noyaux de hauteurs simple et double (et donc choisir entre les deux) en se basant uniquement sur la relation entre précision théorique et temps de calcul. Seules des comparaisons avec les précisions obtenues en pratique seront possibles comme cela est fait au chapitre 6 : voir aussi [51] (section C.3.1) pour une brève comparaison mettant en relation le nombre théorique d'opérations et la précision.

4.2 Transformée Rapide de Fourier

Dans cette section, nous détaillons l'introduction de la Transformée Rapide de Fourier (*Fast Fourier Transform*, ou *FFT*) présentée brièvement en section 2.3.3.12. Cette amélioration de l'opérateur *M2L* est ici adaptée à nos formules, et nous détaillons aussi son implémentation dans notre code FMB. De plus, nous généralisons l'utilisation de la FFT à un noyau *M2L* de hauteur double et notre version par blocs, nécessaire pour réduire les instabilités numériques introduites par la FFT, peut être utilisée pour toutes les valeurs de P.

La FFT n'a été appliquée qu'à l'opérateur *M2L* puisque c'est l'opérateur le plus utilisé dans la FMM, mais son application aux opérateurs *M2M* (voir aussi [52]) et *L2L* est possible. Cependant, comme expliqué dans [51], la procédure complète de la FMM, obtenue en appliquant successivement les trois opérateurs *M2M*, *M2L* et *L2L*, ne peut être effectuée entièrement dans l'espace de Fourier.

Le principe sous-jacent à l'utilisation de la FFT dans la FMM est de considérer les opérateurs *M2M*, *M2L* et *L2L* comme des convolutions (ou corrélations) entre deux suites périodiques. Cette convolution est réalisée dans l'espace *réel* (aussi appelé espace des coefficients par Elliott [51]). Après avoir subi une Transformée de Fourier Discrète (TFD), les deux suites sont considérées comme étant dans l'espace de *Fourier*. La convolution dans l'espace réel correspond alors à un produit terme-à-terme dans l'espace de Fourier, et une Transformée de Fourier Discrète inverse fournit le même résultat (grâce à la propriété de périodicité) dans l'espace réel que la convolution originale. Avec la FFT, qui effectue le calcul d'une TFD pour un faible coût (à l'aide d'un algorithme de type *diviser pour régner*), le passage par l'espace de Fourier est au final moins coûteux que la convolution originale, ce qui justifie la méthode.

La théorie de la Transformée de Fourier Discrète est rappelée dans l'annexe B. Dans la suite de cette section, nous allons l'appliquer au calcul *M2L*.

4.2.1 La Transformée de Fourier Discrète appliquée à l'opérateur *M2L*

Nous considérons tout d'abord l'opérateur *M2L* avec un noyau de hauteur simple tel qu'il est défini dans l'équation (2.20). Le degré des termes des développements varie de 0 à P. On a donc :

$$L_j^k = \sum_{n=0}^{P-j} \sum_{l=-n}^{n} M_n^l O_{n+j}^{-(k+l)} \qquad \forall (j,k) \in \mathbb{N} \times \mathbb{Z}, 0 \leq j \leq P, |k| \leq j. \tag{4.4}$$

4.2.1.1 L'opérateur *M2L* vu comme une convolution $2D$

Nous allons réécrire l'opérateur *M2L* comme une convolution $2D$. Nous aurions pu choisir de le réécrire comme une corrélation $2D$ mais ceci aurait conduit à un produit terme-

à-terme « inversé » dans l'espace de Fourier. Nous avons préféré réaliser l'inversion dans l'espace réel ainsi :

$$\begin{array}{l} {L'}_j^k = L_{-j}^{-k}, \\ {O'}_j^k = O_{-j}^k \quad \text{(seuls les degrés sont inversés).} \end{array} \tag{4.5}$$

En reportant dans l'équation (4.4), on obtient :

$$L'^{-k}_{\ -j} = \sum_{n=0}^{P-j} \sum_{l=-n}^{n} M_n^l {O'}^{-(k+l)}_{\ \ -(n+j)},$$

et avec $j' = -j$ et $k' = -k$:

$$L'^{k'}_{\ j'} = \sum_{n=0}^{P-j} \sum_{l=-n}^{n} M_n^l {O'}^{k'-l}_{\ \ j'-n}.$$

Afin de simplifier la suite des explications, nous allons omettre (temporairement, voir section 4.2.3) cette inversion et redéfinir, pour tout ce qui suit, les termes L_j^k et O_j^k de sorte qu'ils correspondent aux termes ${L'}_j^k$ et ${O'}_j^k$. On étudie donc désormais la convolution $2D$ suivante :

$$L_j^k = \sum_{n=0}^{P-j} \sum_{l=-n}^{n} M_n^l O_{j-n}^{k-l}$$

et on considère que $\Gamma = \{(n,l) \in \mathbb{N} \times \mathbb{Z} \quad avec \quad 0 \leq n \leq P, |l| \leq n\}$, est l'ensemble de définition des termes M_j^k, O_j^k et L_j^k. Tous les termes hors de Γ sont considérés comme nuls.

Comme dans l'annexe B.6, nous définissons deux ensembles $\mathcal{J}$ et $\mathcal{K}$, par :

$$\mathcal{J} = [\![-P, P]\!],$$

$$\mathcal{K} = [\![-P, P]\!],$$

et nous étendons la définition des termes X_j^k ainsi :

$$\widetilde{X}_j^k = \begin{cases} X_j^k & \text{si } (j,k) \in \Gamma \\ 0 & \text{si } (j,k) \in (\mathcal{J} \times \mathcal{K}) \setminus \Gamma \end{cases} \qquad \forall (j,k) \in \mathcal{J} \times \mathcal{K}$$

où $X_j^k = M_j^k$ ou $X_j^k = O_j^k$.

On note $J = |\mathcal{J}|$ et $K = |\mathcal{K}|$, et on étend maintenant la définition des termes $\widetilde{M}_j^k$ et $\widetilde{O}_j^k$ à $\mathbb{Z}^2$ par J-périodicité pour les degrés et par K-périodicité pour les ordres.

On définit alors les termes $\widetilde{L}_j^k$ sur $\mathcal{J} \times \mathcal{K}$ par :

$$\widetilde{L}_j^k = \sum_{n \in \mathcal{J}} \sum_{l \in \mathcal{K}} \widetilde{M}_n^l \widetilde{O}_{j-n}^{k-l} \qquad \forall (j,k) \in \mathcal{J} \times \mathcal{K}.$$

On a désormais le lemme suivant.

Lemme 2.

$$\widetilde{L}_j^k = L_j^k \qquad \forall (j,k) \in \Gamma$$

Démonstration. Nous avons en effet :

$$M_j^k = 0 \qquad \forall (j,k)/|k| > j,$$

$$M_j^k = 0 \qquad \forall (j,k)/j < 0,$$

$$O_j^k = 0 \qquad \forall (j,k)/j > P.$$

□

Mais les termes $\widetilde{L}_j^k$ peuvent être non nuls pour $(j,k) \notin \Gamma$.

A nouveau, nous redéfinissons les termes L_j^k, M_j^k et O_j^k par respectivement $\widetilde{L}_j^k$, $\widetilde{M}_j^k$ et $\widetilde{O}_j^k$, et nous obtenons finalement :

$$L_j^k = \sum_{n \in \mathcal{J}} \sum_{l \in \mathcal{K}} M_n^l O_{j-n}^{k-l} \qquad \forall (j,k) \in \mathbb{Z}^2.$$

Nous avons donc obtenu une convolution discrète $2D$ entre deux suites périodiques : d'après le théorème 7 de l'annexe B, cette convolution peut être calculée dans l'espace de Fourier par le produit terme-à-terme des résultats des deux transformées de Fourier.

Remarque 4.1. *Le « zero-padding » (voir annexe B.6) utilisé dans la définition de $\mathcal{J}$ correspond à un ajout de 100 % de zéros dans la dimension des degrés. Nous ne pouvons pas ajouter moins de zéros à cause de la forme « non-centrée » de la convolution : n varie de 0 à $P - j$ dans (4.4) alors qu'une convolution centrée a des indices variant de $-N+1$ à N. Cependant, pour la définition de $\mathcal{K}$, à savoir dans la dimension des ordres, nous n'avons pas besoin de « zero-padding » (grâce à la nullité hors de Γ).*

4.2.1.2 Utilisation des propriétés de symétrie

La propriété (2.8) dans l'espace réel, valable aussi pour L_j^k et M_j^k, implique dans l'espace de Fourier pour une TFD $1D$ sur les ordres, que la moitié des termes soit égale au conjugué complexe des termes de l'autre moitié.

En effet, en opérant une TFD $1D$ sur une suite $(X)^{l \in \mathbb{Z}}$ de période N avec $X^{-l} = (-1)^l \overline{X^l}$, on obtient :

$$\widehat{X}^{-l} = \frac{1}{N} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} X^k e^{-\frac{i2\pi k(-l)}{N}},$$

$$\widehat{X}^{-l} = \frac{1}{N} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} (-1)^k \overline{X^{-k}} e^{\frac{i2\pi kl}{N}}.$$

On pose $X'^n = X^{-n}$, $\forall n \in \mathbb{Z}$.

$$\widehat{X}^{-l} = \frac{1}{N} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} \overline{X'^k} e^{\frac{i2\pi k}{N}\left(l+\frac{N}{2}\right)}$$

$$\widehat{X}^{-l} = \overline{\frac{1}{N} \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} X'^{k} e^{-\frac{i2\pi k}{N}\left(l+\frac{N}{2}\right)}}$$

Soit :

$$\widehat{X}^{-l} = \overline{\widehat{X'}^{l+\frac{N}{2}}}.$$

Comme $\widehat{X'}^{k} = \widehat{X}^{-k}$, on a :

$$\widehat{X}^{-l} = \overline{\widehat{X}^{-l-\frac{N}{2}}}.$$

$(\widehat{X}^{k})_{k\in\mathbb{Z}}$ étant périodique de période N (voir annexe B.2), on en déduit

$$\widehat{X}^{-l} = \overline{\widehat{X}^{\frac{N}{2}-l}}.$$

C'est pourquoi, dans l'espace de Fourier, seule la moitié des termes a besoin d'être effectivement stockée.

4.2.2 Transformée Rapide de Fourier par blocs

4.2.2.1 Instabilité numérique

D'après la définition 2.1 et les équations (2.6) (2.7), les termes O_j^k et M_j^k se comportent ainsi lorsque j et k augmentent :

$$O_j^k(r,\theta,\phi) \sim \frac{(j-|k|)!}{r^{j+1}} \tag{4.6}$$

$$I_j^k(r,\theta,\phi) \sim \frac{r^j}{(j+|k|)!}. \tag{4.7}$$

Comme mentionné dans [51], cette variation rapide de la norme des termes crée des instabilités numériques lorsqu'on utilise la FFT avec des valeurs de P trop importantes. En effet lorsqu'on réalise une TFD, les termes de faible amplitude sont additionnés aux termes de forte amplitude : ils perdent ainsi leur précision, et l'information contenue dans ces termes de faible amplitude n'est pas récupérée lors de la TFD inverse.

Ce problème dépend de la précision machine utilisée : dans nos calculs en virgule flottante, les instabilités numériques apparaissent pour des valeurs de P bien plus petites en simple précision (4 octets) qu'en double précision (8 octets).

En double précision, ce problème n'apparaît pas avec le *M2L* classique (c'est-à-dire sans FFT) pour les valeurs de P usuelles (du moins jusqu'à $P = 40$) car les termes sont alors multipliés avant d'être additionnés. Cependant, en simple précision des problèmes peuvent survenir avec le *M2L* classique, mais pour des valeurs de P bien plus grandes qu'avec la FFT.

Ce problème a été partiellement (voir sections 4.2.6 et 4.2.7) résolu par Elliott [51] grâce à une FFT par blocs : dans la dimension des degrés, les termes sont regroupés suivant leur degré, et donc suivant leur norme, et la TFD est réalisée sur chaque bloc séparément. Le produit terme-à-terme doit alors être effectué bloc par bloc tout en utilisant une *convolution à gros grain* [51] entre les blocs.

Nous allons maintenant détailler cette FFT par blocs.

4.2.2.2 Décomposition en blocs

On note :
- B : le nombre de blocs ;
- T : la « taille » de chaque bloc (dans la dimension des degrés, et sans zero-padding, voir plus loin).

On suppose tout d'abord que $P+1$ est un multiple de T, et on alors : $B = \frac{P+1}{T}$. La version par blocs de la FFT ne concerne que la dimension des degrés, la FFT dans la dimension des ordres demeurant inchangée. Nous allons donc considérer la corrélation $1D$ suivante :

$$L_j = \sum_{n=0}^{P-j} M_n O_{j+n},$$

les termes L_i, M_i et O_i étant définis sur $[\![0, P]\!]$.

Pour tout $I \in [\![0, B-1]\!]$, nous définissons le bloc $M(I)$ par :

$$M(I)_i = \begin{cases} M_{IT+i} & \forall i \in [\![0, T-1]\!], \\ 0 & \forall i \in [\![-(T-1), 0]\!]. \end{cases}$$

Nous étendons cette définition à $\mathbb{Z}$ par $(2T-1)$-périodicité. Les blocs $O(I)$ sont définis similairement et sont aussi étendus à $\mathbb{Z}$ par $(2T-1)$-périodicité. Pour tout $(I, J) \in [\![0, B-1]\!]^2$, et pour tout $i \in [\![-(T-1), T-1]\!]$, nous définissons le bloc $X(I, J)$ ainsi :

$$X(I, J)_i = \sum_{n=-(T-1)}^{T-1} M(I)_n O(J)_{i+n}. \tag{4.8}$$

D'après la définition des blocs $M(I)$ et $O(J)$, la somme peut être réduite à : $X(I, J)_i = \sum_{n=0}^{T-1-i} M(I)_n O(J)_{i+n}$.

Le bloc $X(I, J)$ est donc le résulat de la corrélation $1D$ de taille $2T-1$ entre le bloc $M(I)$ et le bloc $O(J)$. De par la définition de ces blocs (périodicité et zero-padding), on peut obtenir exactement le même résultat en procédant par l'enchaînement : TFD / produit terme-à-terme inversé dans l'espace de Fourier / TFD inverse.

Nous devons désormais récupérer les coefficients L_j dans les blocs $X(I, J)$. La FFT pour *M2L* n'est cependant pas parfaitement décomposable en blocs (comme l'est un produit matrice-vecteur par exemple) car son résultat se retrouve « dispersé » sur plusieurs blocs : chaque L_j doit alors être calculé en sommant plusieurs termes dans différents blocs $X(I, J)$. Afin de déterminer quels termes doivent être additionnés pour un L_j donné, nous soulignons tout d'abord que $\forall i \in [\![-(T-1), T-1]\!]$, $X(I, I+J)_i$ fait partie de la somme pour L_{JT+i}.

Plus précisément,

le bloc $X(I, I+J)$ contribue à

$L_{JT-(T-1)}$
$L_{JT-(T-2)}$
$\dots$
L_{JT-1}
L_{JT}
L_{JT+1}
$\dots$
$L_{JT+(T-1)}$

Du point de vue des termes L_j, comme :

$$\forall i \in [\![0, P]\!], \qquad \exists!(I, i') \in [\![0, B-1]\!] \times [\![0, T-1]\!] \qquad \text{tel que} \qquad i = IT + i',$$

nous avons, si $i' = 0$

$$L_i = \sum_{N=0}^{B-1-I} X(N, N+I)_{i'}, \tag{4.9}$$

et sinon

$$L_i = \sum_{N=0}^{B-1-I} X(N, N+I)_{i'} + \sum_{N=0}^{B-1-I-1} X(N, N+I+1)_{i'-T}. \tag{4.10}$$

Comme annoncé dans [51], une convolution à gros grain (ici une corrélation) apparaît au niveau des blocs dans les équations (4.9) et (4.10) : ceci est illustré par la figure 4.2.

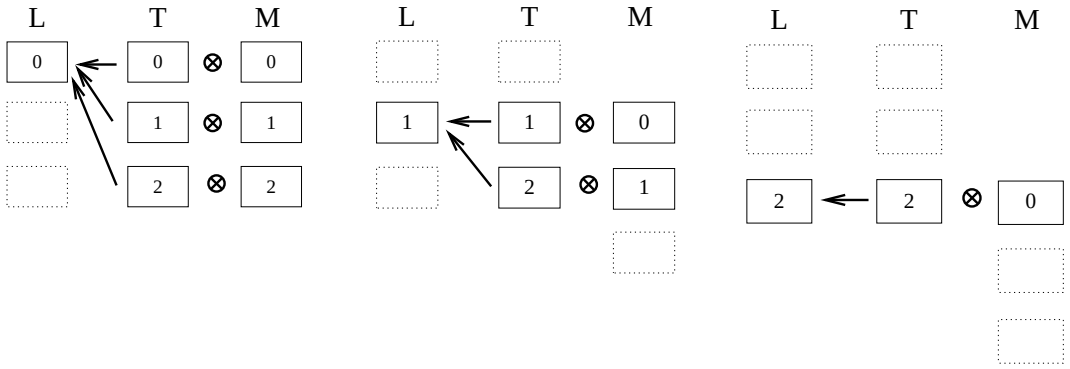


FIG. 4.2 – Corrélation à gros grain de la FFT par blocs. Ici M et L désignent les blocs contenant les développements multipôles et les développements locaux, et T les blocs contenant les fonctions de transfert *M2L*.

Quand $P+1$ n'est pas un multiple de T, le nombre de blocs utilisé est $\frac{P+1}{T} + 1$ (division entière). Le dernier bloc contient alors des lignes nulles en plus de celles utilisées pour le zero-padding.

Dans la version par blocs présentée dans [51], les premiers blocs sont plus petits dans la dimension des ordres : ceci n'a pas été utilisé ici car toutes les FFT sont précalculées (voir section 2), ce qui réduit le gain d'une telle amélioration.

4.2.3 Détails d'implémentation

FFTW [54] est utilisé comme implémentation efficace de la FFT. Ceci nous évite l'écriture de notre propre FFT, mais présente l'inconvénient de ne pas pouvoir exploiter directement les symétries sur les ordres (voir section 4.2.1.2) : avec FFTW nous avons besoin d'un tableau complet dans la dimension des ordres (c'est-à-dire contenant les ordres positifs et négatifs) alors qu'une FFT codée spécialement n'aurait eu besoin que d'un demi tableau.

Plus précisément, nous ne pouvons pas utiliser la FFT $2D$ de FFTW mais nous devons décomposer cette FFT $2D$ en deux ensembles de FFT $1D$: les FFT $1D$ dans la dimension des ordres sont d'abord exécutées sur un tableau complet contenant à la fois les ordres positifs et les ordres négatifs, puis seule une moitié de ce tableau est conservée pour la FFT $1D$ dans la dimension des degrés. Le produit terme-à-terme est alors réalisé sur les demi tableaux, et on inverse le processus pour la TFD inverse. Ceci implique des copies entre les tableaux complets et les demi tableaux, mais ce surcoût reste faible.

L'avantage principal de FFTW, outre son efficacité, est qu'il peut traiter des tailles arbitraires pour la FFT : nous ne somme pas ici limité aux puissances de 2.

4.2.3.1 Sans blocs

Avec FFTW, une FFT $1D$ de taille N calcule le tableau Y_k :

$$Y_k = \sum_{j=0}^{N-1} X_j e^{\frac{-2i\pi jk}{N}} \qquad \forall k \in [\![0, N-1]\!].$$

Jusqu'à présent, nous avons défini les termes de nos développements sur $\mathcal{J} \times \mathcal{K} = [\![-P, P]\!] \times [\![-P, P]\!]$: ces intervalles sont centrés sur 0. Cependant, pour fournir des tableaux corrects en entrée de FFTW, les $P+1$ premiers termes dans chacune des 2 dimensions doivent correspondre aux termes d'indices positifs (ou nuls), alors que les P termes suivants correspondent par périodicité aux termes d'indices négatifs. C'est ce qu'on appelle le *wrap-around order*.

Pour une disposition « théorique » du développement suivant, avec $P = 4$ (dimensions $2P+1 \times 2P+1$),

				X_0^0				
			X_1^{-1}	X_1^0	X_1^1			
		X_2^{-2}	X_2^{-1}	X_2^0	X_2^1	X_2^2		
	X_3^{-3}	X_3^{-2}	X_3^{-1}	X_3^0	X_3^1	X_3^2	X_3^3	
X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}	X_4^0	X_4^1	X_4^2	X_4^3	X_4^4

$k \in [\![4,4]\!]$ (ordres)

$j \in [\![4,4]\!]$ (degrés)

nous aurons donc le stockage suivant en mémoire

X_0^0								
X_1^0	X_1^1							X_1^{-1}
X_2^0	X_2^1	X_2^2					X_2^{-2}	X_2^{-1}
X_3^0	X_3^1	X_3^2	X_3^3			X_3^{-3}	X_3^{-2}	X_3^{-1}
X_4^0	X_4^1	X_4^2	X_4^3	X_4^4	X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}

$k \in [\![0,8]\!]$ (ordres)

$j \in [\![0,8]\!]$ (degrés)

Cependant, à cause de la redéfinition de nos termes O_j^k (voir l'équation (4.5)), les degrés de la fonction de transfert *M2L* sont inversés, ce qui donne

X_0^0								
X_4^0	X_4^1	X_4^2	X_4^3	X_4^4	X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}
X_3^0	X_3^1	X_3^2	X_3^3			X_3^{-3}	X_3^{-2}	X_3^{-1}
X_2^0	X_2^1	X_2^2					X_2^{-2}	X_2^{-1}
X_1^0	X_1^1							X_1^{-1}

$k \in [\![0,8]\!]$ (ordres)

$j \in [\![0,8]\!]$ (degrés)

(4.11)

Les ligne nulles peuvent évidemment être sautées lorsqu'on réalise les FFT $1D$ dans la dimension des ordres (avant les FFT $1D$ dans la dimension des degrés), mais cette disposition n'est pas propice à la « planification » des FFT avec FFTW (voir la notion de « plan » dans FFTW). C'est pourquoi nous avons choisi de ne pas utiliser le wrap-around order : ceci aboutit à un décalage dans les indices des 2 dimensions (ordres et degrés) dans l'espace réel. Avec des taille paires, le décalage est égal à la moitié de la dimension, et dans l'espace de Fourier, un tel décalage revient à multiplier chaque terme par $(-1)^{j+k}$. Or ces facteurs disparaissent ensuite lors du produit terme-à-terme ! Mais afin d'avoir des tailles paires, nous devons ajouter une ligne nulle et une colonne nulle à nos tableaux, les tailles de nos FFT devenant alors $2(P+1)$ dans chaque dimension.

Nous avons donc, sans wrap-around order et sans inversion des degrés (c'est-à-dire pour

les termes M_j^k)

					X_0^0				
				X_1^{-1}	X_1^0	X_1^1			
			X_2^{-2}	X_2^{-1}	X_2^0	X_2^1	X_2^2		
		X_3^{-3}	X_3^{-2}	X_3^{-1}	X_3^0	X_3^1	X_3^2	X_3^3	
	X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}	X_4^0	X_4^1	X_4^2	X_4^3	X_4^4

$k \in [\![0,9]\!]$ (ordres)

$j \in [\![0,9]\!]$ (degrés)

et avec inversion des degrés (c'est-à-dire pour les termes O_j^k)

	X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}	X_4^0	X_4^1	X_4^2	X_4^3	X_4^4
		X_3^{-3}	X_3^{-2}	X_3^{-1}	X_3^0	X_3^1	X_3^2	X_3^3	
			X_2^{-2}	X_2^{-1}	X_2^0	X_2^1	X_2^2		
				X_1^{-1}	X_1^0	X_1^1			
					X_0^0				

$k \in [\![0,9]\!]$ (ordres)

$j \in [\![0,9]\!]$ (degrés)

Les termes L_j^k obtenus par FFT inverse sont stockés avec wrap-around order et ont leurs degrés et leurs ordres inversés (voir équation (4.5)) : leur disposition en mémoire est donc la même qu'en (4.11) mais avec des ordres inversés. De plus, lors de la FFT inverse dans la dimension des ordres, nous n'exécutons de FFT $1D$ inverse que sur les lignes qui nous serons utiles pour récuperer les termes L_j^k, à savoir les lignes qui contiennent les degrés j vérifiant $0 \leq j \leq P$.

Remarque 4.2. *Le choix de ne pas utiliser le wrap-around order, pour les développements multipôles comme pour les fonctions de transfert* M2L*, a été fait dans le but de réduire le coût des FFT. Cependant, comme la plus grande partie du temps de calcul est due au produit terme-à-terme (voir section 4.2.4), il aurait aussi été possible de conserver le wrap-around order (et ainsi des tailles impairs en* $2P+1 \times 2P+1$*, au lieu de* $2(P+1) \times 2(P+1)$ *) afin d'avoir de plus petits tableaux pour le produit terme-à-terme.*

4.2.3.2 Avec blocs

Nous avons présenté dans la section 4.2.2 une décomposition en blocs d'une corrélation $1D$. Cette décomposition est appliquée à l'opérateur *M2L* mais uniquement pour les degrés : dans la dimension des ordres la procédure reste inchangée comparée à la version sans blocs. Dans la dimension des degrés par contre, nous considérons la corrélation de l'équation (4.4), qui aboutira donc à une corrélation à gros grain entre les blocs, mais pour les corrélations élémentaires entre blocs (voir équation (4.8)), nous réalisons la même inversion qu'à l'équation (4.5) au sein de chaque bloc : l'équation (4.8) est ainsi réécrite comme une convolution, ce qui aboutit à un produit terme-à-terme non inversé dans l'espace de Fourier.

Nous donnons maintenant un exemple de la disposition en mémoire de nos blocs pour $P = 7$: alors qu'une version sans blocs de la FFT utiliserait des tableaux de taille 16×16, la FFT par blocs, avec $T = 2$ et avec zero-padding ($T \to 2T$), utilise des blocs de taille 4×16. Nous représentons ici le bloc numero 2.

La disposition théorique serait :

				X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}	X_4^0	X_4^1	X_4^2	X_4^3	X_4^4			
			X_5^{-5}	X_5^{-4}	X_5^{-3}	X_5^{-2}	X_5^{-1}	X_5^0	X_5^1	X_5^2	X_5^3	X_5^4	X_5^5		

$k \in [\![-8, 7]\!]$ (ordres)

$j \in [\![-2, 1]\!]$ (degrés)

Pour les développements multipôles (c'est-à-dire sans wrap-around order, et sans inversion des degrés ou des ordres) :

				X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}	X_4^0	X_4^1	X_4^2	X_4^3	X_4^4			
			X_5^{-5}	X_5^{-4}	X_5^{-3}	X_5^{-2}	X_5^{-1}	X_5^0	X_5^1	X_5^2	X_5^3	X_5^4	X_5^5		

$k \in [\![0, 15]\!]$ (ordres)

$j \in [\![0, 3]\!]$ (degrés)

Pour les fonctions de transfert *M2L* (c'est-à-dire sans wrap-around order, et avec inversion des degrés) :

			X_5^{-5}	X_5^{-4}	X_5^{-3}	X_5^{-2}	X_5^{-1}	X_5^0	X_5^1	X_5^2	X_5^3	X_5^4	X_5^5		
				X_4^{-4}	X_4^{-3}	X_4^{-2}	X_4^{-1}	X_4^0	X_4^1	X_4^2	X_4^3	X_4^4			

$k \in [\![0, 15]\!]$ (ordres)

$j \in [\![0, 3]\!]$ (degrés)

Pour les développements locaux (c'est-à-dire avec wrap-around order, et avec inversion des degrés et des ordres) :

X_4^0	X_4^{-1}	X_4^{-2}	X_4^{-3}	X_4^{-4}								X_4^4	X_4^3	X_4^2	X_4^1
X_5^0	X_5^{-1}	X_5^{-2}	X_5^{-3}	X_5^{-4}	X_5^{-5}						X_5^5	X_5^4	X_5^3	X_5^2	X_5^1

$k \in [\![0, 15]\!]$ (ordres)

$j \in [\![0, 3]\!]$ (degrés)

Comme pour la version sans blocs de la FFT, les lignes nulles sont sautées lors de la FFT dans la dimension des ordres pour les blocs des développements multipôles et les blocs des fonctions de transfert. Cependant pour la FFT inverse dans la dimension des ordres (pour les blocs des développements locaux), il n'y a qu'une seule ligne « inutile » par bloc : toutes les autres lignes contiennent des termes utiles car la décomposition en blocs disperse les termes des développements locaux à l'intérieur de chaque bloc. En pratique, nous ne sautons aucune ligne dans ce cas.

Remarque 4.3 (Utilisation des routines BLAS)**.** *Dans la version par blocs comme dans la version sans blocs, le produit terme-à-terme, qui est la partie la plus coûteuse du calcul de l'opérateur $M2L$ avec FFT, ne correspond à aucune routine BLAS standard (voir section 5). En fait, on ne peut espérer obtenir beaucoup d'accélération d'un tel calcul qui ne présente pas de réutilisation des données dans les caches : une fois que la seule boucle du calcul a été écrite, l'obtention d'un code « pipeliné » sur des architectures superscalaires dépend essentiellement du compilateur.*

4.2.4 Complexité théorique

Nous étudions maintenant la complexité théorique des versions FFT avec et sans blocs de l'opérateur *M2L*. Tout d'abord, nous considérons que les fonctions de transfert *M2L* (les termes O_j^k) sont précalculées à chaque niveau l de l'octree, et que leur conversion dans l'espace de Fourier est aussi effectuée à l'avance. Pour des valeurs de l croissantes, le coût de ce précalcul devient rapidement insignifiant. Le calcul de *M2L* avec FFT se déroule ensuite ainsi : tous les développements multipôles sont d'abord convertis dans l'espace de Fourier, puis pour chaque cellule c, nous additionnons les résultats de tous les produits terme-à-terme de sa liste d'interaction, et ramenons cette somme dans l'espace réel afin d'obtenir le développement local de la cellule c. Comme la liste d'interaction a 189 éléments (pour $ws = 1$), nous considérons donc :

- 1 conversion dans l'espace de Fourier pour un développement multipôle,
- 189 produits terme-à-terme,
- 1 conversion inverse dans l'espace réel pour un développement local.

Ceci, multiplié par le nombre de cellules au niveau l, donne une estimation précise du nombre d'opérations nécessaires au calcul de toutes les opérations *M2L* avec FFT au niveau l. Le nombre d'opérations estimé pour une FFT $1D$ complexe de taille N avec FFTW est $5N\log_2(N)$ (voir la documentation en ligne de FFTW à l'URL http://www.fftw.org).

4.2.4.1 Sans blocs

Complexité théorique 4.1. *Pour* 189 *opérations* M2L*, la complexité théorique de la FFT sans blocs est :*

$$189\ M2L = 40(P+1)^2\log_2(2P+2) + 378 * (P+1)^2.$$

Démonstration. Avec l'utilisation de la symétrie sur les ordres (telle qu'elle est décrite en section 4.2.1.2), l'ajout d'une ligne et d'une colonne supplémentaires pour avoir des tailles paires (à savoir $2(P+1) \times 2(P+1)$, voir section 4.2.3.1), et des FFT $1D$ (non inverses et inverses) réalisées uniquement sur les $P+1$ lignes « utiles », nous avons pour la version sans blocs :

– pour les FFT sur le tableau du développement multipôle considéré, $P+1$ FFT de taille $2(P+1)$ dans la dimension des ordres (tableau complet avec lignes nulles sautées), suivies de $P+1$ FFT de taille $2(P+1)$ dans la dimension des degrés (demi tableau), soit :

$$(P+1)5(2P+2)\log_2(2P+2) + (P+1)5(2P+2)\log_2(2P+2) \\ = 20(P+1)^2\log_2(2P+2);$$

– pour les FFT inverses sur le tableau du développement local, $P+1$ FFT de taille $2(P+1)$ dans la dimension des degrés (demi tableau), puis $P+1$ FFT de taille $2(P+1)$ dans la dimension des ordres (tableau complet avec lignes inutiles sautées), soit :

$$20(P+1)^2\log_2(2P+2);$$

– 189 produits terme-à-terme entre demi tableaux (de taille $2(P+1)\times(P+1)$) :

$$189 * 2 * (P+1)^2 .$$

Le résultat final est obtenu en sommant ces valeurs. □

Pour les valeurs usuelles de P, la partie la plus coûteuse du calcul de l'opérateur *M2L* avec FFT est par conséquent le produit terme-à-terme.

4.2.4.2 Avec blocs

Complexité théorique 4.2. *Pour* 189 *opérations* M2L*, la complexité théorique de la FFT par blocs est :*

$$189\ M2L = 189\left(\frac{(P+1)^3}{T} + (P+1)^2\right) + 10\,(P+1)^2\log_2\left(32\,(P+1)^2\,T^2\right).$$

Démonstration. Nous comptons les mêmes opérations que pour la version sans blocs et nous supposons ici que $P+1$ est un multiple de T, d'où $B = \frac{P+1}{T}$. La taille d'un bloc avec zero-padding est alors $2T\times 2(P+1)$ pour la FFT dans la dimension des ordres (tableau complet), et $2T\times(P+1)$ pour la FFT dans la dimension des degrés et pour la corrélation à gros grain (demi tableau).

– Pour les FFT sur les B blocs du développement multipôle considéré, nous comptons T FFT de taille $2(P+1)$ dans la dimension des ordres (tableau complet avec lignes nulles sautées) et $P+1$ FFT de taille $2T$ dans la dimension des degrés (demi tableau), soit :

$$B\,(5T(2P+2)\log_2(2P+2) + (P+1)5(2T)\log_2(2T)) \\ = 10BT(P+1)\log_2(4(P+1)T);$$

– Pour les FFT inverses sur les B blocs du développement local, nous comptons $P+1$ FFT de taille $2T$ dans la dimension des degrés (demi tableau) et ensuite $2T$ FFT de taille $2(P+1)$ dans la dimension des ordres (tableau complet mais aucune ligne sautée), soit :

$$B\,(5(P+1)(2T)\log_2(2T) + (2T)5(2P+2)\log_2(2P+2)) \\ = 10BT(P+1)\log_2(8(P+1)T);$$

– 189 corrélations à gros grain avec des produits terme-à-terme entre blocs de taille $2T \times (P+1)$:

$$189 \left[\sum_{J=0}^{B-1} \sum_{I=0}^{B-1-J} (2T\,(P+1)) \right] = 189B(B+1)T(P+1).$$

Au final, nous avons donc :

$$189\ M2L = 10\,(P+1)^2 \log_2 \left(32\,(P+1)^2\,T^2\right) + 189\,(P+1)^2\,(B+1)\,.$$

□

Nous voyons ici que la version par blocs de la FFT pour l'opérateur *M2L* a un nombre d'opérations qui grandit en $\mathcal{O}(P^3)$, contre $\mathcal{O}(P^2 \log_2 P)$ pour la version sans blocs. Nous pouvons aussi remarquer que, comparé à la version sans blocs, la partie « corrélation à gros grain / produit terme-à-terme » est encore plus coûteuse que la partie « FFT / FFT inverse ».

4.2.5 Besoins mémoire

Comparons maintenant la mémoire requise par le calcul *M2L* avec FFT avec celle requise par le calcul *M2L* classique (voir section 2.3.3.6). Avec les mêmes notations qu'en section 2.3.3.6, nous calculons les besoins de la version sans blocs. La version par blocs a les mêmes besoins mémoire lorsque $P+1$ est un multiple de T. Sinon, la taille utilisée est la même que pour le premier multiple de T supérieur (moins 1).

Puisque les développements dans l'espace de Fourier sont stockés dans des demi-tableaux avec des nombres complexes, leur taille est donnée par $2(P+1)^2 t_c$. Ceci est valable pour les développements multipôles et les développements locaux, ainsi que pour les fonctions de transfert. Cependant nous n'avons besoin que d'un seul demi tableau pour les développements locaux : celui-ci est utilisé comme un tableau auxiliaire où tous les résultats des calculs *M2L* de la liste d'interaction sont additionnés pour la cellule courante. La taille de cet unique tableau est négligée.

Nous avons donc (ici pour un noyau *M2L* de hauteur simple)

$$Mem_{Sg}(P,H) = \mathcal{N}(H) \left(2(P+1)^2 + \frac{(P+1)(P+2)}{2}\right) t_c + \mathcal{N}_{\mathcal{T}} \left(2(P+1)^2\right) t_c. \qquad (4.12)$$

4.2.6 Instabilités résiduelles

Cependant des tests montrent que des instabilités numériques subsistent pour de grandes valeurs de P : ceci est illustré par la figure 4.3 pour un noyau *M2L* de hauteur simple. On peut l'expliquer par l'influence de la valeur des ordres (c'est-à-dire k) dans les équations (4.6) et (4.7) : la décomposition par blocs n'est réalisée que dans la dimension des degrés, mais pour de grandes valeurs de P des instabilités numériques peuvent aussi apparaître dans la dimension des ordres.

Cette instabilité peut cependant être réduite en utilisant des blocs de taille plus petite : alors que, pour un noyau *M2L* de hauteur simple, la FFT avec des blocs de taille 4 devient instable pour P supérieur à 14 (voir figure 4.3), une FFT avec des blocs de taille 3

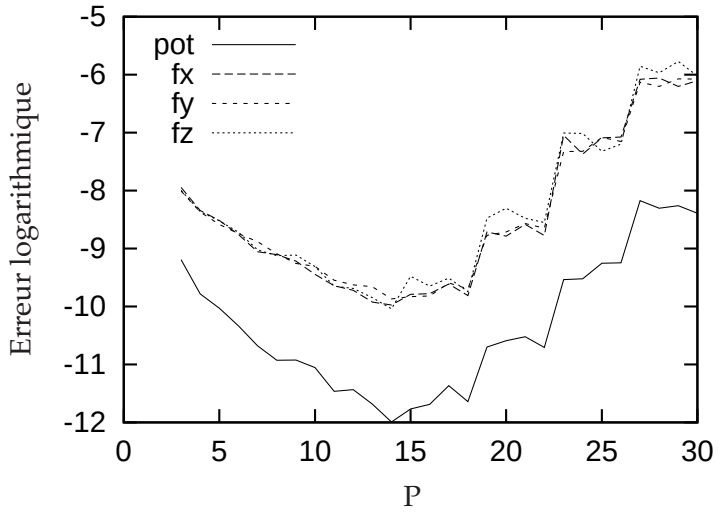


FIG. 4.3 – Erreur logarithmique pour la potentiel (pot) et les composantes de la force (fx, fy, fz) en fonction de P pour un noyau *M2L* de hauteur simple avec FFT par blocs de taille 4. Tests réalisés sur une distribution uniforme de 10 000 particules avec un octree de hauteur 4. L'erreur tracée représente l'erreur absolue maximale sur toutes les particules.

est stable jusqu'à 27, tout en étant bien sûr plus coûteuse en temps de calcul du fait du plus grand nombre de blocs utilisés.

Malheureusement, le risque d'instabilité numérique augmente aussi avec la hauteur de l'octree. En effet, lorsqu'on considère les équations (4.6) et (4.7), nous devons aussi prendre en compte l'influence de deux distances :

- la distance entre les centres des deux cellules considérées (pour les termes O_j^k dans l'opérateur *M2L* par exemple),
- et la distance entre le centre d'une cellule et la position d'une particule dans cette cellule (pour les termes M_j^k dans l'opérateur *P2M*).

Avec une hauteur croissante de l'octree, ces distances décroissent, car la taille d'une cellule est alors divisée par 2 entre deux niveaux consécutifs de l'octree. Ceci crée des instabilités numériques lorsque leur ordre d'amplitude devient trop petit. Comme le montre la figure 4.4, avec un octree de hauteur 6, la FFT par blocs de taille 4 est par exemple instable pour P supérieur à 10 (comparé à 14 pour une hauteur 4).

Les valeurs de P et de hauteur d'octree au-dessus desquelles des instabilités numériques apparaissent sont résumées dans le tableau[1] 4.1 pour un noyau *M2L* de hauteur simple (pour des FFT par blocs de taille 2, la hauteur de l'arbre n'influence pas l'apparition des instabilités dans les tests que nous avons effectués, mais ceci sera probablement le cas avec des hauteurs plus importantes qui ne peuvent pas être utilisées ici avec la mémoire d'un seul processeur). Ces valeurs dépendent naturellement de la taille de la boîte de calcul englobant toutes les particules : nos tests sont réalisés avec les coordonnées de toutes les particules comprises dans $[0\ ;\ 1,0]$, mais avec une boîte de calcul plus grande les instabilités numériques appa-

[1] Tests effectués sur des processeurs IBM Power3 (2 Go de mémoire) et Power4 (8 Go de mémoire) au LaBRI (Laboratoire Bordelais de Recherche en Informatique, Talence, France), ainsi que sur des processeurs IBM Power3 (16 Go de mémoire) and Power4 (32 Go de mémoire) au CINES (Centre Informatique National de l'Enseignement Supérieur, Montpellier, France). Voir section 5 pour plus d'informations sur ces machines.

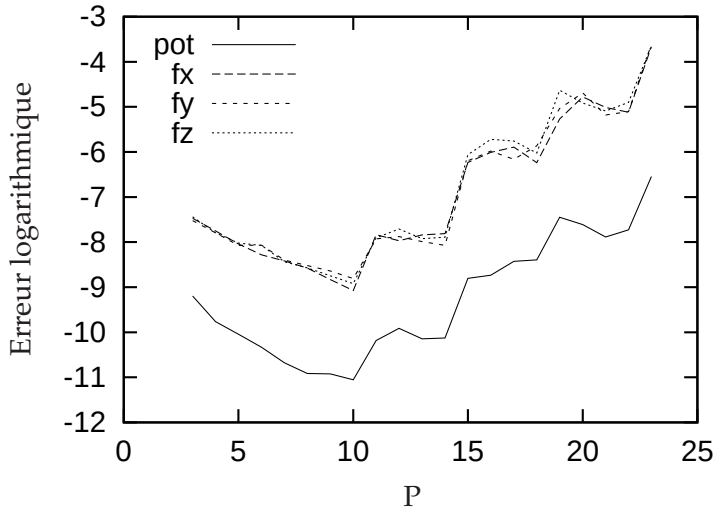


FIG. 4.4 – Erreur logarithmique pour le potentiel (pot) et les composantes de la force (fx, fy, fz) en fonction de P pour un noyau *M2L* de hauteur simple avec FFT par blocs de taille 4. Tests réalisés sur une distribution uniforme de 10 000 particules avec un octree de hauteur 6. L'erreur tracée représente l'erreur absolue maximale sur toutes les particules.

	$hauteur = 4$	$hauteur = 5$	$hauteur = 6$
FFT par blocs de taille 4	14	14	10
FFT par blocs de taille 3	27	23	20
FFT par blocs de taille 2	40	40	40

TAB. 4.1 – Valeurs critiques pour P, en fonction de la hauteur de l'octree et de la taille des blocs de la FFT, au dessus desquelles des instabilités numériques apparaissent.

raîtraient néanmoins pour des hauteurs supérieures.

Cependant, l'instabilité due à la hauteur croissante de l'arbre peut être évitée grâce à une technique simple : lorsqu'on multiplie les coordonnées cartésiennes de toutes les particules, de même que les coordonnées de la boîte de calcul, par un scalaire $\alpha > 1$, les deux distances mentionnées ci-dessus sont alors allongées, ce qui tend à réduire l'instabilité numérique. Le potentiel et les forces initiaux peuvent être obtenus après coup grâce à une multiplication par respectivement $\frac{1}{\alpha}$ et $\frac{1}{\alpha^2}$. Par exemple, avec $\alpha = 4$, les instabilités numériques avec une hauteur 6 correspondent à celles obtenues précédement (c'est-à-dire sans multiplication par α) avec une hauteur 4. Et avec $\alpha = 4$ et une hauteur 4, aucune instabilité numérique n'est detectée dans nos tests pour P allant de 3 à 40 avec des blocs de taille 4.

Toutefois, ces instabilités restent problématiques et limitent l'utilisation de la FFT pour le calcul de l'opérateur *M2L* : la taille critique d'une feuille n'est a priori pas connue et dépend de la précision machine utilisée. De plus, la nécessité d'utiliser ou non la technique de redimensionnement décrite ci-dessus dépend de la valeur de P, de la taille des blocs et de la taille des feuilles.

4.2.7 Comparaison avec le code DPMTA

Afin de vérifier la justesse et l'efficacité de notre implémentation de la FFT pour l'opérateur *M2L*, nous avons comparé notre code (FMB) avec le code DPMTA [51] [102] (voir section 2.3.3). Nous avons utilisé DPMTA avec la liste d'interaction « fixe » de Greengard & Rokhlin et $ws = 1$; aucun MAC (*Multipole Acceptance Criterion*, ou θ) n'a été utilisé. Comme dans notre code, toutes les fonctions de transfert *M2L* sont précalculées. La signification de P dans notre code n'est cependant pas la même que celle donnée au P_{DPMTA} utilisé dans DPMTA. On a en effet $P_{DPMTA} = P+1$. La comparaison a été réalisée sur des processeurs IBM Power3 et Power4, nécessitant donc quelques modifications des Makefiles afin de pouvoir utiliser le compilateur IBM C (xlc) pour DPMTA. Les mêmes options de compilation ont été utilisées pour DPMTA et pour FMB (à savoir pour Power3 : *xlc -O3 -qstrict -bmaxdata :0x80000000 -qarch=pwr3 -qtune=pwr3 -Q=150 -qstaticinline*). La figure 4.5 montre que dans ces conditions notre implémentation de la FFT pour l'opérateur *M2L* est aussi rapide que celle de DPMTA.

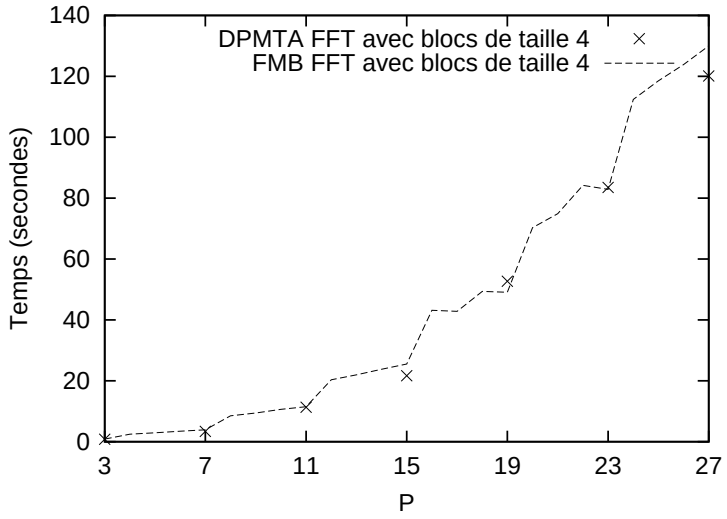


FIG. 4.5 – Temps CPU de la phase de descente pour DPMTA et FMB : la hauteur de l'octree est 4 et la distribution uniforme utilisée contient 100 000 particules, de sorte qu'il n'y ait pas de feuille vide. Les coordonnées de toutes les particules sont dans $[0, 1]$. Nous utilisons un noyau *M2L* de hauteur simple car c'est le seul disponible dans DPMTA, et des blocs de taille 4 pour la FFT. Les P utilisés en abscisse sont ceux de FMB ($P_{DPMTA} = P + 1$) et DPMTA n'accepte pour P_{DPMTA} que les multiples de la taille des blocs.

D'autre part, nous avons aussi trouvé des instabilités numériques en testant DPMTA sur un calcul FMM complet : celles-ci apparaissent pour les mêmes valeurs de P et de hauteur d'octree qu'avec FMB (voir section 4.2.6).

4.2.8 Transformée Rapide de Fourier pour un noyau *M2L* de hauteur double

Avec un noyau *M2L* de hauteur double, le degré maximum des termes O_j^k est $2P$. Nous devons donc définir $\mathcal{J}$ et $\mathcal{K}$ par : $\mathcal{J} = [\![-2P, 2P]\!]$ et $\mathcal{K} = [\![-2P, 2P]\!]$. Mais grâce à l'utilisation des symétries (voir section 4.2.1.2) nous utilisons en fait : $\mathcal{K} = [\![0, 2P]\!]$. Les mêmes $\mathcal{J}$ et $\mathcal{K}$ doivent être utilisés pour les termes M_j^k et L_j^k : ceci est obtenu par zero-padding. La FFT pour

un noyau *M2L* de hauteur double opère ensuite de la même façon que pour une hauteur simple.

Avec un noyau *M2L* de hauteur double, les instabilités numériques sont encore plus problématiques qu'avec une hauteur simple, et la décomposition en blocs de la FFT est d'autant plus nécessaire. En effet, comme les termes O_j^k ont des degrés allant jusqu'à $2P$, l'échelle des amplitudes est encore plus large et les calculs deviennent instables pour des valeurs de P d'autant plus petites. Comme pour la FFT sans blocs, la FFT par blocs pour un noyau de hauteur double se déduit directement de celle pour un noyau de hauteur simple une fois que $\mathcal{J}$ et $\mathcal{K}$ ont été redéfinis. Mais pour une même valeur de la taille des blocs T, nous avons environ deux fois plus de blocs avec une hauteur double qu'avec une hauteur simple.

Cependant, même avec la version par blocs, la plage de valeurs de P qui conduit à un calcul correct avec un noyau de hauteur double est plus petite que celle liée à un noyau de hauteur simple (à moins qu'un redimensionnement soit effectué comme indiqué en section 4.2.6). En comparaison avec le tableau 4.1, une FFT par blocs de taille 4 est ici stable uniquement pour $P \leq 8$ (dans un octree de hauteur 4 et une boîte de calcul de côté 1). En s'accordant plus de temps de calcul pour gagner en précision, nous pouvons atteindre $P = 12$ avec des blocs de taille 3, et $P = 19$ avec des blocs de taille 2. Et comme pour le noyau de hauteur simple, la situation empire avec des hauteurs croissantes de l'octree.

Afin d'établir la complexité théorique de l'utilisation de la FFT pour l'opérateur *M2L* avec un noyau de hauteur double, nous considérons les mêmes opérations qu'en section (4.2.4).

Pour la version sans blocs, nous avons juste à remplacer P par $2P$:

$$189\ M2L = 40\,(2P+1)^2 \log_2\left(2\,(2P+1)\right) + 378 * (2P+1)^2\,.$$

Remarque 4.4. *En pratique on peut sauter plus que $2P+1$ lignes lors des FFT sur les tableaux contenant les développements multipôles et lors des FFT inverses sur les tableaux contenant les développements locaux, car le degré maximum de ces développements reste P.*

Pour la version par blocs de la FFT, nous considérons que $2P+1$ est un multiple de T : ce n'est bien sûr jamais le cas pour $T = 4$ par exemple, mais l'estimation présentée ci-dessous est proche de la réalité. Nous remplaçons alors P par $2P$ et le nombre de blocs devient $B = \frac{2P+1}{T}$. La taille d'un bloc est désormais soit $2T \times 2(2P+1)$ (tableau complet), soit $2T \times (2P+1)$ (demi tableau), et on obtient :

$$189\ M2L = 10\,(2P+1)^2 \log_2\left(32\,(2P+1)^2\,T^2\right) + 189\left(\frac{(2P+1)^3}{T} + (2P+1)^2\right).$$

Remarque 4.5. *Comme pour la version sans blocs, plus de lignes peuvent être sautées en pratique.*

Enfin, nous donnons les besoins mémoire pour un noyau *M2L* de hauteur double en procédant de la même façon qu'en section 4.2.5. La taille de tous les demi tableaux est maintenant $2(2P+1)^2$, pour les développements multipôles et pour les fonctions de transfert, d'où :

$$Mem_{Db}(P,H) = \mathcal{N}(H)\left(2(2P+1)^2 + \frac{(P+1)(P+2)}{2}\right)t_c + \mathcal{N}_{\mathcal{T}}\left(2(2P+1)^2\right)t_c. \quad (4.13)$$

4.2.9 Comparaison entre noyaux de hauteurs simple et double

Nous rappellons tout d'abord que le rapport entre les coûts (théoriques) des noyaux de hauteurs simple et double est d'environ 6 dans le cas du calcul *M2L* classique (voir section 2.3.3.5).

Pour la FFT sans bloc, nous avons, sans tenir compte des logarithmes, un rapport de 4 qui favorise donc le noyau de hauteur double par rapport au calcul *M2L* classique. Cependant, à cause de sa plus gande instabilité numérique, le noyau de hauteur double requiert davantage l'utilisation de la version par blocs de la FFT. Pour cette version par blocs, le rapport reste à 4 pour la partie « FFT / FFT inverse » mais vaut environ 8 pour la partie « corrélation à gros grain / produit terme-à-terme » qui est la plus coûteuse. Ceci ne favorise clairement pas le noyau de hauteur double.

4.3 Rotations

Nous détaillons ici l'introduction des *rotations* dans la FMM, telle qu'elle a été présentée en section 2.3.3.12. Nous avons choisi d'utiliser les formules détaillées par Gumerov et Duraiswami dans [70] car :

- ils utilisent la même définition que nous pour leurs harmoniques sphériques,
- ils utilisent des symétries pour accélérer les calculs,
- ils se concentrent uniquement sur les rotations nécessaires,
- la récurrence est réalisée sur des nombres réels (et non sur des nombres complexes),
- et enfin cette récurrence est simple à démarrer.

Cependant aucune preuve n'est donnée sur la stabilité numérique des formules utilisées : cette stabilité devra donc être vérifiée expérimentalement (voir section 4.3.4).

Nous présentons ici le principe de cette amélioration uniquement pour l'opérateur *M2L*, et nous détaillons les formules utilisées pour des noyaux *M2L* de hauteurs simple et double. Nous donnons aussi les formules adaptées aux fonctions *Outer* (O_j^k) et *Inner* (I_j^k). Finalement, nous insistons sur le fait que le calcul *M2L* avec rotations ne correspond à aucune routine BLAS standard, et nous vérifions la stabilité numérique de nos calculs.

4.3.1 Formules

Le principe général du calcul *M2L* avec rotations est illustré par la figure 4.6. Nous allons ici introduire les formules mathématiques requises et détailler cette amélioration.

4.3.1.1 Principe de la méthode

Notations. On note respectivement $\mathcal{R}_x(\alpha)$, $\mathcal{R}_y(\alpha)$, $\mathcal{R}_z(\alpha)$ les rotations d'angle α du système cartésien de coordonnées autour des axes $\mathbf{x}$, $\mathbf{y}$ et $\mathbf{z}$. Ces rotations sont considérées comme étant réalisées dans le sens contraire des aiguilles d'une montre lorsqu'on regarde dans la direction de l'origine depuis une position de coordonnée positive sur l'axe de rotation.

Soit un point $\mathbf{P}$ de coordonnées sphériques (r, θ, ϕ) dans le système initial de coordonnées, ses coordonnées sphériques dans le système résultant de la rotation sont notées $(r, \widehat{\theta}, \widehat{\phi})$.

De manière générale, nous allons d'abord considérer une rotation (du système cartésien de coordonnées) d'angle χ autour de l'axe $\mathbf{z}$, suivie d'une rotation d'angle γ autour de l'axe

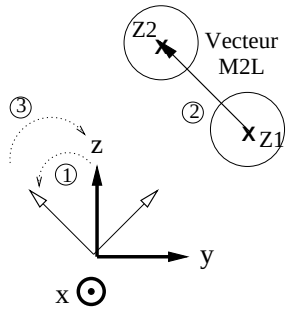


FIG. 4.6 – Calcul *M2L* avec rotations. 1 : rotation du système de coordonnées afin que le vecteur *M2L* soit aligné sur l'axe **z**. 2 : calcul *M2L* réalisé le long de l'axe **z**. 3 : rotation inverse afin de retrouver le système initial de coordonnées.

y, et enfin une rotation d'angle $\pi - \omega$ autour du *nouvel* axe **z**, soit :

$$\mathcal{R}_z(\pi - \omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(\chi). \tag{4.14}$$

Remarque 4.6. *Dans [70] la deuxième rotation est réalisée autour de l'axe* **x**.

Rotation des harmoniques sphériques. Comme indiqué dans les papiers cités précédement, pour les rotations $\mathcal{R}_z(\pi - \omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(\chi)$ il existe des coefficients $T_j^{\nu,k}(\omega, \gamma, \chi)$, avec $j \geq 0, |k| \leq j, |\nu| \leq j$, tels qu'une harmonique sphérique $Y_j^k(\theta, \phi)$, exprimée dans le système initial de coordonnées, peut s'écrire en fonction d'harmoniques sphériques, exprimées dans le système résultant de la rotation, ainsi :

$$Y_j^k(\theta, \phi) = \sum_{\nu=-j}^{j} T_j^{\nu,k}(\omega, \gamma, \chi) Y_j^\nu(\widehat{\theta}, \widehat{\phi}). \tag{4.15}$$

Les mêmes $T_j^{\nu,k}$ s'appliquent à la fois aux harmoniques sphériques normalisées et aux harmoniques sphériques non normalisées.

Les fonctions *Outer* et *Inner*, définies aux équations (2.6) et (2.7), étant basées sur ces harmoniques sphériques, il existe aussi des coefficients $T_{O_j}^{\nu,k}(\omega, \gamma, \chi)$, et $T_{I_j}^{\nu,k}(\omega, \gamma, \chi)$ tels que

$$O_j^k(r, \theta, \phi) = \sum_{\nu=-j}^{j} T_{O_j}^{\nu,k}(\omega, \gamma, \chi) O_j^\nu(r, \widehat{\theta}, \widehat{\phi}), \tag{4.16}$$

et

$$I_j^k(r, \theta, \phi) = \sum_{\nu=-j}^{j} T_{I_j}^{\nu,k}(\omega, \gamma, \chi) I_j^\nu(r, \widehat{\theta}, \widehat{\phi}). \tag{4.17}$$

Ces coefficients $T_j^{\nu,k}$, $T_{O_j}^{\nu,k}$ et $T_{I_j}^{\nu,k}$ sont considérés comme nuls pour $|\nu| > j$ ou $|k| > j$. Par la suite nous les appellerons *coefficients de rotation*.

Rotation des développements multipôles. Soit $\mathbf{P}$ le point de coordonnées sphériques (r, θ, ϕ) dans le système initial de coordonnées. Le potentiel évalué en $\mathbf{P}$ avec un développement multipôle est donné par

$$\Phi(\mathbf{P}) = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} M_n^l O_n^{-l}(r, \theta, \phi).$$

Dans le système de coordonnées résultant de la rotation, le potentiel s'écrit

$$\Phi(\mathbf{P}) = \sum_{n=0}^{+\infty} \sum_{\nu=-n}^{n} \widehat{M}_n^\nu O_n^{-\nu}(r, \widehat{\theta}, \widehat{\phi}),$$

avec les nouveaux termes du développement multipôle définis par

$$\widehat{M}_n^\nu = \sum_{l=-n}^{n} M_n^l T_{O_n}^{-\nu,-l}(\omega, \gamma, \chi). \tag{4.18}$$

Rotation des développements locaux. Avec un développement local, le potentiel en $\mathbf{P}$ s'écrit dans le système initial de coordonnées

$$\Phi(\mathbf{P}) = \sum_{n=0}^{+\infty} \sum_{l=-n}^{n} L_n^l I_n^l(r, \theta, \phi).$$

Dans le système de coordonnées résultant de la rotation, le potentiel est donné par

$$\Phi(\mathbf{P}) = \sum_{n=0}^{+\infty} \sum_{\nu=-n}^{n} \widehat{L}_n^\nu I_n^\nu(r, \widehat{\theta}, \widehat{\phi}),$$

où les nouveaux termes du développement local sont définis par

$$\widehat{L}_n^\nu = \sum_{l=-n}^{n} L_n^l T_{I_n}^{\nu,l}(\omega, \gamma, \chi). \tag{4.19}$$

Opérateur *M2L* le long de l'axe z. Le gain dans l'utilisation des rotations pour l'opérateur *M2L* est basé sur le fait qu'un calcul *M2L* le long de l'axe $\mathbf{z}$ a un coût en $\mathcal{O}(P^3)$, contre $\mathcal{O}(P^4)$ pour un calcul *M2L* classique. Les fonctions associées de Legendre vérifient en effet

$$P_n^m(1) = \delta_{m,0}$$

où $\delta_{i,j}$ est le symbole de Kronecker. Nous avons par conséquent

$$O_j^k(r, 0, 0) = \begin{cases} \frac{(-1)^j}{A_j^0 r^{j+1}} = \frac{j!}{r^{j+1}} & \text{si } k = 0, \\ 0 & \text{sinon,} \end{cases}$$

et

$$I_j^k(r, 0, 0) = \begin{cases} A_j^0 r^j = \frac{(-1)^j r^j}{j!} & \text{si } k = 0, \\ 0 & \text{sinon.} \end{cases}$$

Quand le vecteur *M2L* (défini par $\mathbf{z_2}-\mathbf{z_1}$ dans la définition 2.3) a des coordonnées sphériques de la forme $(r,0,0)$, l'opérateur *M2L* peut être calculé ainsi

$$L_j^k = \sum_{n=0}^{+\infty} M_n^{-k} O_{j+n}^0(r,0,0) = \sum_{n=0}^{+\infty} M_n^{-k} \frac{(-1)^{j+n}}{A_{j+n}^0 r^{j+n+1}}. \tag{4.20}$$

Comme M_n^{-k} impose $n \geq |k|$, nous avons pour un noyau *M2L* de hauteur simple

$$L_j^k = \sum_{n=|k|}^{P-j} M_n^{-k} \frac{(-1)^{j+n}}{A_{j+n}^0 r^{j+n+1}}, \tag{4.21}$$

et pour un noyau *M2L* de hauteur double

$$L_j^k = \sum_{n=|k|}^{P} M_n^{-k} \frac{(-1)^{j+n}}{A_{j+n}^0 r^{j+n+1}}. \tag{4.22}$$

Calcul *M2L* quelconque avec rotations. Nous pouvons désormais détailler le principe illustré en figure 4.6 pour un calcul *M2L* quelconque avec un centre de cellule $\mathbf{z_1}$ pour le développement multipôle et un centre de cellule $\mathbf{z_2}$ pour le développement local. Le vecteur *M2L* $\mathbf{V} = \mathbf{z_2} - \mathbf{z_1}$ a pour coordonnées sphériques (r,θ,ϕ). Nous effectuons tout d'abord une rotation du système de coordonnées de sorte que le vecteur $\mathbf{V}$ soit aligné le long de l'axe $\mathbf{z}$. Ceci peut être réalisé avec les 2 rotations suivantes : $\mathcal{R}_y(\theta).\mathcal{R}_z(\phi)$. Cependant nous utilisons une formulation plus générale avec 3 rotations qui pourra aussi être utilisée pour la rotation inverse comme exposé plus loin. Ainsi nous effectuons $\mathcal{R}_z(\pi-\omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(\chi)$ avec $\omega=\pi$, $\gamma=\theta$, $\chi=\phi$, soit :

$$\mathcal{R}_z(0).\mathcal{R}_y(\theta).\mathcal{R}_z(\phi).$$

Les coordonnées de $\mathbf{V}$ dans le nouveau système de coordonnées sont $(r,0,0)$. Le développement multipôle dans le nouveau système est lui donné par l'équation (4.18). Grâce à l'équation (4.20) nous calculons le développement local dans le nouveau système de coordonnées. Afin d'obtenir l'expression du développement local dans le système initial, nous devons utiliser l'équation (4.19) avec la rotation inverse :

$$\mathcal{R}_z(\pi-\phi).\mathcal{R}_y(\theta).\mathcal{R}_z(\pi).$$

En effet, dans le cadre général de l'équation (4.14), la rotation inverse s'écrit :

$$\mathcal{R}_z(-\chi).\mathcal{R}_y(-\gamma).\mathcal{R}_z(-(\pi-\omega)) = \mathcal{R}_z(\pi-\chi).\mathcal{R}_y(\gamma).\mathcal{R}_z(\omega). \tag{4.23}$$

En conclusion, nous avons besoin de calculer les coefficients de rotation suivants : ${T_O}_j^{\nu,k}(\pi,\theta,\phi)$ et ${T_I}_j^{\nu,k}(\phi,\theta,\pi)$, $\forall j \geq 0, |\nu| \leq j, |k| \leq j$.

4.3.1.2 Calcul des coefficients de rotation

La plupart des formules qui suivent sont une simple réécriture de celles présentées par Gumerov et Duraiswami dans [70]. La méthode complète utilisée pour calculer les coefficients de rotation est donnée à la fin de la section : toutes les bases mathématiques sont d'abord présentées.

Matrice de rotation. Soit $\mathbf{V}$ un vecteur de coordonnées cartésiennes (x, y, z) dans le système de coordonnées $(\mathbf{i}_x, \mathbf{i}_y, \mathbf{i}_z)$. Après rotation du système de coordonnées, nous notons $\widehat{\mathbf{V}} = (\widehat{x}, \widehat{y}, \widehat{z})$ ses nouvelles coordonnées dans le système de coordonnées $(\mathbf{i}_{\widehat{x}}, \mathbf{i}_{\widehat{y}}, \mathbf{i}_{\widehat{z}})$ résultant des rotations. Nous notons alors $\mathbf{Q}$ la matrice de rotation suivante $\widehat{\mathbf{V}} = \mathbf{Q}\mathbf{V}$, à savoir :

$$\mathbf{Q} = \begin{bmatrix} \mathbf{i}_{\widehat{x}}.\mathbf{i}_x & \mathbf{i}_{\widehat{x}}.\mathbf{i}_y & \mathbf{i}_{\widehat{x}}.\mathbf{i}_z \\ \mathbf{i}_{\widehat{y}}.\mathbf{i}_x & \mathbf{i}_{\widehat{y}}.\mathbf{i}_y & \mathbf{i}_{\widehat{y}}.\mathbf{i}_z \\ \mathbf{i}_{\widehat{z}}.\mathbf{i}_x & \mathbf{i}_{\widehat{z}}.\mathbf{i}_y & \mathbf{i}_{\widehat{z}}.\mathbf{i}_z \end{bmatrix} \tag{4.24}$$

Comme présenté en section 4.3.1.1, nous considérons les rotations suivantes : $\mathcal{R}_z(\pi - \omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(\chi)$. Dans ce cas, nous avons

$$\mathbf{Q}(\omega, \gamma, \chi) = \begin{bmatrix} -\cos\omega & \sin\omega & 0 \\ -\sin\omega & -\cos\omega & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{bmatrix} \begin{bmatrix} \cos\chi & \sin\chi & 0 \\ -\sin\chi & \cos\chi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

soit

$$\mathbf{Q}(\omega, \gamma, \chi) = \begin{bmatrix} -\cos\omega.\cos\gamma.\cos\chi - \sin\omega.\sin\chi & -\cos\omega.\cos\gamma.\sin\chi + \sin\omega.\cos\chi & \cos\omega.\sin\gamma \\ -\sin\omega.\cos\gamma.\cos\chi + \cos\omega.\sin\chi & -\sin\omega.\cos\gamma.\sin\chi - \cos\omega.\cos\chi & \sin\omega.\sin\gamma \\ \sin\gamma.\cos\chi & \sin\gamma.\sin\chi & \cos\gamma \end{bmatrix} \tag{4.25}$$

$\mathbf{Q}(\omega, \gamma, \chi)$ est une matrice orthogonale et de plus, nous avons (voir équation (4.23)) :

$$\mathbf{Q}^{-1}(\omega, \gamma, \chi) = \mathbf{Q}(\chi, \gamma, \omega).$$

Calcul récursif des coefficients $T_j^{\nu,k}$. Nous utilisons la même définition des harmoniques sphériques Y_j^k (voir équation (A.2)) que dans [70]. Dans ce papier, Gumerov & Duraiswami utilisent l'équation (5.39) (avec des degrés j croissants pour $T_j^{\nu,k}$) comme exemple d'un calcul récursif des coefficients $T_j^{\nu,k}$. Nous préférons utiliser ici leur équation (5.28) avec des degrés décroissants.

Avec la matrice $\mathbf{W}$ definie (avec $i = \sqrt{-1}$) par

$$\mathbf{W} = \begin{bmatrix} \mathbf{i}_x.\left(\mathbf{i}_{\widehat{x}} - i\mathbf{i}_{\widehat{y}}\right) & \mathbf{i}_x.\left(\mathbf{i}_{\widehat{x}} + i\mathbf{i}_{\widehat{y}}\right) & -2\mathbf{i}_x.\mathbf{i}_{\widehat{z}} \\ i\mathbf{i}_y.\left(\mathbf{i}_{\widehat{x}} - i\mathbf{i}_{\widehat{y}}\right) & i\mathbf{i}_y.\left(\mathbf{i}_{\widehat{x}} + i\mathbf{i}_{\widehat{y}}\right) & -2i\mathbf{i}_y.\mathbf{i}_{\widehat{z}} \\ -\frac{1}{2}\mathbf{i}_z.\left(\mathbf{i}_{\widehat{x}} - i\mathbf{i}_{\widehat{y}}\right) & -\frac{1}{2}\mathbf{i}_z.\left(\mathbf{i}_{\widehat{x}} + i\mathbf{i}_{\widehat{y}}\right) & \mathbf{i}_z.\mathbf{i}_{\widehat{z}} \end{bmatrix}$$

l'équation (5.28) dans [70] s'écrit :

$$2b_{n+1}^{-m-1}T_{n+1}^{\nu,m+1} = (W_{1,1}+W_{2,1})b_{n+1}^{-\nu}T_n^{\nu-1,m} + (W_{1,2}+W_{2,2})b_{n+1}^{\nu}T_n^{\nu+1,m} + (W_{1,3}-W_{2,3})a_n^{\nu}T_n^{\nu,m}.$$

Avec la convention $T_n^{\nu,m} = 0, \forall (n,m)/|m| > n$, nous avons les équations suivantes (qui correspondent à (5.30) et (5.33) dans [70]) :

$$2b_{n+1}^{-m-1}T_{n+1}^{n+1,m+1} = (W_{1,1} + W_{2,1})b_{n+1}^{-n-1}T_n^{n,m},$$

$$2b_{n+1}^{-m-1}T_{n+1}^{-n-1,m+1} = (W_{1,2} + W_{2,2})b_{n+1}^{-n-1}T_n^{-n,m}.$$

Avec nos équations (4.24) et (4.25), nous obtenons les relations suivantes entre $\mathbf{W}$ et $\mathbf{Q}$:

$$W_{1,1} + W_{2,1} = \mathbf{Q}_{1,1} + \mathbf{Q}_{2,2} + i(\mathbf{Q}_{1,2} - \mathbf{Q}_{2,1}) = (\cos\gamma + 1)e^{i\chi}e^{i(\pi-\omega)},$$

$$W_{1,2} + W_{2,2} = \mathbf{Q}_{1,1} - \mathbf{Q}_{2,2} + i(\mathbf{Q}_{1,2} + \mathbf{Q}_{2,1}) = (\cos\gamma - 1)e^{i\chi}e^{-i(\pi-\omega)},$$
$$W_{1,3} + W_{2,3} = -2\sin\gamma.e^{i\chi}.$$

D'où :

$$T_{n+1}^{\nu,m+1} = \frac{e^{i\chi}}{b_{n+1}^{-m-1}} \left\{ \frac{1}{2}\left[(\cos\gamma + 1)\, e^{i(\pi-\omega)} b_{n+1}^{-\nu} T_n^{\nu-1,m} + (\cos\gamma - 1) e^{-i(\pi-\omega)} b_{n+1}^{\nu} T_n^{\nu+1,m} \right] - \sin\gamma a_n^{\nu} T_n^{\nu,m} \right\}.$$

Cette récursion est réalisée dans $\mathbb{C}$; nous allons maintenant la simplifier afin qu'elle ait lieu dans $\mathbb{R}$.

Calcul récursif dans $\mathbb{R}$. En effectuant la rotation $\mathcal{R}_z(\alpha)$ sur une harmonique sphérique $Y_j^k(\theta,\phi)$, nous avons : $Y_j^k(\theta,\phi) = e^{ik\alpha} Y_j^k(\widehat{\theta},\widehat{\phi})$. Nous pouvons donc décomposer $T_n^{\nu,m}(\omega,\gamma,\chi)$ en (encore une fois, nous renvoyons à l'article [70] pour les détails et la preuve explicite) :

$$T_n^{\nu,m}(\omega,\gamma,\chi) = e^{im\chi} e^{-i\nu\omega} H_n^{\nu,m}(\gamma) \tag{4.26}$$

avec $H_n^{\nu,m}(\gamma) \in \mathbb{R}$.

L'équation récursive portant sur les coefficients $H_n^{\nu,m}(\gamma)$ est :

$$H_{n+1}^{\nu,m+1} = \frac{1}{b_{n+1}^{-m-1}} \left\{ \frac{1}{2}\left[-(\cos\gamma + 1)\, b_{n+1}^{-\nu} H_n^{\nu-1,m} - (\cos\gamma - 1) b_{n+1}^{\nu} H_n^{\nu+1,m} \right] - \sin\gamma a_n^{\nu} H_n^{\nu,m} \right\}. \tag{4.27}$$

Cette récursion démarre avec $H_n^{\nu,0}(\gamma)$ grâce au lemme suivant :

Lemme 3.

$$H_n^{\nu,0}(\gamma) = (-1)^{\nu} \sqrt{\frac{(n-|\nu|)!}{(n+|\nu|)!}} P_n^{|\nu|}(\cos\gamma).$$

Démonstration. Nous procédons comme dans [70] pour le calcul des coefficients $H_n^{\nu,0}(\gamma)$. Plus précisément, nous considérons $\mathcal{R}_z(\pi-\omega).\mathcal{R}_y(\gamma).\mathcal{R}_z(0)$ avec $\mathbf{V} = (r,\theta,\phi)$. θ désigne ici l'angle entre $\mathbf{V}$ et $\mathbf{i}_z$: c'est aussi l'angle entre $\widehat{\mathbf{V}}$ et $\mathbf{i}_{\widehat{z}}$. De plus, les coordonnées sphériques de l'axe $\mathbf{z}$ dans le système de coordonnées $(\mathbf{i}_{\widehat{x}}, \mathbf{i}_{\widehat{y}}, \mathbf{i}_{\widehat{z}})$ résultant de la rotation sont : $(1,\gamma,\omega)$. En appliquant le théorème d'addition des harmoniques sphériques (théorème 1 présenté en section 2.3.3.2) entre $\mathbf{V}$ et l'axe $\mathbf{z}$ dans le système résultant de la rotation, on obtient alors

$$P_n(\cos\theta) = \sum_{\nu=-n}^{n} Y_n^{-\nu}(\gamma,\omega) Y_n^{\nu}(\widehat{\theta},\widehat{\phi}).$$

Nous identifions cette équation avec celle-ci (obtenue à partir de l'équation (4.15)) :

$$P_n(\cos\gamma) = Y_n^0(\theta,\phi) = \sum_{\nu=-n}^{n} T_n^{\nu,0}(\omega,\gamma,0) Y_n^{\nu}(\widehat{\theta},\widehat{\phi})$$

ce qui donne $\forall\chi$

$$T_n^{\nu,0}(\omega,\gamma,\chi) = Y_n^{-\nu}(\gamma,\omega).$$

Nous en déduisons :

$$H_n^{\nu,0}(\gamma) = Y_n^{-\nu}(\gamma,0) = Y_n^{\nu}(\gamma,0).$$

□

Finalement, d'après l'équation (A.3) et comme les harmoniques sphériques normalisées forment une base orthonormale, nous avons

$$T_n^{-\nu,-m}(\gamma,\chi) = \overline{T_n^{\nu,m}(\gamma,\chi)}, \tag{4.28}$$

et par conséquent d'après l'équation (4.26)

$$H_n^{-\nu,-m}(\gamma) = H_n^{\nu,m}(\gamma). \tag{4.29}$$

De plus (voir équation (5.52) in [70]) nous avons aussi :

$$H_n^{\nu,m}(\gamma) = H_n^{m,\nu}(\gamma). \tag{4.30}$$

Ces deux équations, (4.29) et (4.30), sont utiles pour accélérer le calcul des coefficients $H_n^{\nu,m}(\gamma)$ (voir ci-dessous).

Calcul des coefficients $T_{O_j}^{\nu,k}$ et $T_{I_j}^{\nu,k}$. D'après la définition des fonctions *Outer* et *Inner*, équations (2.4) et (2.5),et d'après la définition de leurs coefficients de rotation, équations (4.16) et (4.17), nous pouvons écrire avec l'équation (4.15)

$$T_{O_j}^{\nu,k} = i^{|k|-|\nu|}\frac{A_j^\nu}{A_j^k}T_j^{\nu,k}, \tag{4.31}$$

et

$$T_{I_j}^{\nu,k} = i^{|\nu|-|k|}\frac{A_j^k}{A_j^\nu}T_j^{\nu,k}. \tag{4.32}$$

Avec l'équation (4.28), nous avons aussi

$$T_{O_j}^{-\nu,-k} = (-1)^{k+\nu}\overline{T_{O_j}^{\nu,k}},$$

$$T_{I_j}^{-\nu,-k} = (-1)^{k+\nu}\overline{T_{I_j}^{\nu,k}}.$$

Procédure de calcul. Les coefficients $H_n^{\nu,0}(\gamma)$ étant donnés par le lemme 3 et l'équation (4.29) pour $n = [\![0..P]\!]$ et $|\nu| \leq n$, nous utilisons les équations (4.27) et (4.30) pour calculer les $H_n^{\nu,m}(\gamma)$ pour $n = [\![0..P]\!]$, $|\nu| \leq n$ et $|m| \leq n$. Les coefficients $T_{O_j}^{\nu,k}$ et $T_{I_j}^{\nu,k}$ se déduisent alors des coefficients $H_n^{\nu,m}$ grâce aux équations (4.26), (4.31) et (4.32).

Il n'y a cependant pas besoin d'accélérer le calcul des coefficients $T_{O_j}^{\nu,k}$ et $T_{I_j}^{\nu,k}$: ceci fait en effet partie du précalcul des fonctions de transfert *M2L* à chaque niveau de l'octree (voir section 2), et le temps de calcul associé est marginal. Il est ici plus important de se concentrer sur la stabilité numérique du schéma de calcul utilisé pour ces coefficients de rotation (voir section 4.3.4).

4.3.1.3 Complexité théorique

Comme pour l'étude de la complexité théorique de la version FFT de l'opérateur *M2L*, nous ne considérons pas ici le précalcul réalisé une seule fois à chaque niveau de l'octree. Les coefficients $T_{O_j}^{\nu,k}(\pi,\theta,\phi)$, $T_{I_j}^{\nu,k}(\phi,\theta,\pi)$ et $O_j^k(r,0,0)$ ne sont par conséquent pas pris en compte.

Pour un calcul *M2L*, nous considérons donc :

- une rotation pour le développement multipôle,
- un calcul *M2L* le long de l'axe **z**,
- une rotation pour le développement local.

Complexité théorique 4.3. *La complexité théorique d'un calcul* M2L *avec rotations dans le cas d'un noyau de hauteur simple est :*

$$M2L_{ROT} = \frac{(P+2)(14P^2+41P+36)}{24}.$$

Démonstration. Nous avons, avec $Q = P/2$ (division entière) :

$$ROT(\text{Multipôle}) = \sum_{\nu=0}^{Q}\sum_{j=\nu}^{P-\nu}\sum_{k=-j}^{j} = \frac{(P+1)(P+2)^2}{4} \approx \frac{1}{4}P^3,$$

$$M2L_z = \sum_{j=0}^{P}\sum_{k=0}^{\min(j,P-j)}\sum_{n=k}^{P-j} = \frac{(2P+3)(P+4)(P+2)}{24} \approx \frac{1}{12}P^3,$$

$$ROT(Local) = \sum_{j=0}^{P}\sum_{\nu=0}^{j}\sum_{k=-Min(j,P-j)}^{Min(j,P-j)} = \frac{(P+2)(P^2+2P+2)}{4} \approx \frac{1}{4}P^3,$$

et $M2L_{ROT}$ est la somme de ces trois grandeurs. □

Complexité théorique 4.4. *La complexité théorique d'un calcul* M2L *avec rotations dans le cas d'un noyau de hauteur double est :*

$$M2L_{ROT2P} = \frac{(10P+9)(P+2)(P+1)}{6}.$$

Démonstration. Nous avons :

$$ROT(\text{Multipôle}) = \sum_{\nu=0}^{P}\sum_{j=\nu}^{P}\sum_{k=-j}^{j} = \frac{(4P+3)(P+2)(P+1)}{6} \approx \frac{2}{3}P^3,$$

$$M2L_z = \sum_{j=0}^{P}\sum_{k=0}^{j}\sum_{n=k}^{P} = \frac{(2P+3)(P+2)(P+1)}{6} \approx \frac{1}{3}P^3,$$

$$ROT(Local) = \sum_{j=0}^{P}\sum_{\nu=0}^{j}\sum_{k=-j}^{j} = \frac{(4P+3)(P+2)(P+1)}{6} \approx \frac{2}{3}P^3,$$

et on obtient $M2L_{ROT2P}$ en sommant ces trois grandeurs. □

Le rapport entre les coûts (théoriques) des noyaux de hauteurs simple et double avec rotations est donc :

$$\frac{M2L_{ROT2P}}{M2L_{ROT}} = \frac{4(10P+9)(P+1)}{14P^2+41P+36} \approx \frac{20}{7}$$

en ne conservant que les termes en P^2. Ce rapport est plus faible que ceux du calcul *M2L* classique (rapport de 6, voir section 2.3.3.5) et du calcul *M2L* avec FFT par blocs (rapport de 8, voir section 4.2.9). A priori, le noyau de hauteur double gagnera donc en efficacité par rapport à celui de hauteur simple dans le cas des rotations.

4.3.2 Besoins mémoire

L'implémentation de l'utilisation des rotations pour le calcul *M2L* ne pose pas de problèmes particuliers, excepté une attention particulière lors de l'écriture des boucles dans le cas d'un noyau de hauteur simple. Les coefficients de rotation sont simplement stockés dans des tableaux à trois dimensions. Nous allons maintenant détailler les besoins en mémoire supplémentaire.

Pour chaque calcul *M2L* avec rotations, nous avons premièrement besoin de deux tableaux auxiliaires où sont stockés le développement multipôle et le développement local le long de l'axe **z**. Dans le cas d'un noyau de hauteur double, ces tableaux ont la même taille qu'un développement classique (multipôle ou local), et dans le cas d'un noyau de hauteur simple, la taille requise est même plus petite. Dans les deux cas, cet espace mémoire est négligé. A la place d'un développement, avec des degrés allant jusqu'à P ou $2P$, pour le calcul *M2L* classique, nous avons maintenant besoin pour chaque vecteur *M2L* :

- de la fonction de transfert *M2L* correspondante le long de l'axe **z** qui ne contient que les termes d'ordre nul : sa taille est par conséquent $P+1$ ou $2P+1$ et chaque terme est un nombre réel ;
- de deux tableaux contenant les termes complexes $T_{O_j}^{\nu,k}$ et $T_{I_j}^{\nu,k}$ correspondants au vecteur *M2L*, pour $0 \leq j \leq P$, $|k| \leq j$ et $0 \leq \nu \leq j$. En effet nous n'avons pas besoin des termes avec $-j \leq \nu < 0$. La taille de chaque tableau est : $\sum_{n=0}^{P} \sum_{\nu=0}^{n} \sum_{k=-n}^{n} = \frac{(4P+3)(P+2)(P+1)}{6}$. Ceci s'applique à la fois au noyau de hauteur simple et au noyau de hauteur double.

Finalement, nous avons donc, pour un noyau *M2L* de hauteur simple (avec les mêmes notations qu'en section 2.3.3.6)

$$Mem_{Sg}(P,H) = \mathcal{N}(H)(P+1)(P+2)t_c + \mathcal{N}_{\mathcal{T}} \left(\frac{P+1}{2} + \frac{(4P+3)(P+2)(P+1)}{3} \right) t_c, \quad (4.33)$$

et pour un noyau *M2L* de hauteur double

$$Mem_{Db}(P,H) = \mathcal{N}(H)(P+1)(P+2)t_c + \mathcal{N}_{\mathcal{T}} \left(\frac{2P+1}{2} + \frac{(4P+3)(P+2)(P+1)}{3} \right) t_c. \quad (4.34)$$

Comparé aux équations (2.21) et (2.22), il est clair qu'à moins d'une très faible hauteur d'octree et d'une très grande valeur de P, les besoins mémoire pour l'utilisation des rotations sont faibles.

4.3.3 Etude d'une utilisation des routines BLAS

Nous considérons ici l'utilisation de routines BLAS standards afin d'accélérer le calcul *M2L* avec rotations (voir section 5 pour une introduction sur les BLAS). Nous commençons par un noyau *M2L* de hauteur double car c'est le cas le plus commode.

4.3.3.1 Noyau de hauteur double

Lorsqu'on emploie les rotations pour l'opérateur *M2L*, nous utilisons 2 types différents d'opérations : les rotations des développements multipôles et locaux d'une part, et le calcul *M2L* le long de l'axe **z** d'autre part.

Rotations des développements. Nous nous concentrons sur la rotation d'un développement multipôle : la rotation d'un développement local est semblable. Nous avons besoin de calculer les ordres négatifs (voir équation (4.20)) ainsi

$$\widehat{M_n^{-\nu}} = \sum_{l=-n}^{n} M_n^l T_{O_n}^{\nu,-l}(\omega,\gamma,\chi)$$

avec $0 \leq j \leq P$ et $0 \leq |\nu| \leq j$ pour un noyau de hauteur double. Ce calcul ne se prête pas à un produit matrice-vecteur, puisqu'il ressemble à

$$c_{\nu,n} = \sum_{l=\dots}^{\dots} a_{n,\nu,l}.b_{l,n}$$

alors que nous aurions besoin d'une expression de la forme

$$c_{\nu,n} = \sum_{l=\dots}^{\dots} a_{\nu,l}.b_{l,n}$$

pour $\mathbf{C} = \mathbf{A}.\mathbf{B}$, avec $\widehat{M_n^{-\nu}}$ stocké dans $\mathbf{C}$, $T_{O_n}^{\nu,-l}$ dans $\mathbf{A}$ et M_n^l dans $\mathbf{B}$. La dépendance en n des $T_{O_n}^{\nu,-l}$ nous empêche de considérer la rotation d'un développement multipôle comme un produit matrice-vecteur, et ainsi d'utiliser des routines BLAS de niveau 2, et a fortiori celles de niveau 3. Il est cependant possible d'utiliser des routines BLAS de niveau 1 pour le calcul d'un $\widehat{M_n^{-\nu}}$ donné, mais le gain offert par ces BLAS de niveau 1 est limité.

Calcul *M2L* le long de l'axe z. L'équation (4.22) peut être réécrite sous la forme du produit matrice-vecteur suivant (ici dans le cas $P = 3$) :

$$\begin{bmatrix} \widehat{L}_0^0 & X & X & X \\ \widehat{L}_1^0 & \widehat{L}_1^1 & X & X \\ \widehat{L}_2^0 & \widehat{L}_2^1 & \widehat{L}_2^2 & X \\ \widehat{L}_3^0 & \widehat{L}_3^1 & \widehat{L}_3^2 & \widehat{L}_3^3 \end{bmatrix} = \begin{bmatrix} O_0^0 & O_1^0 & O_2^0 & O_3^0 \\ O_1^0 & O_2^0 & O_3^0 & O_4^0 \\ O_2^0 & O_3^0 & O_4^0 & O_5^0 \\ O_3^0 & O_4^0 & O_5^0 & O_6^0 \end{bmatrix} \begin{bmatrix} \widehat{M_0^0} & 0 & 0 & 0 \\ \widehat{M_1^0} & \widehat{M_1^{-1}} & 0 & 0 \\ \widehat{M_2^0} & \widehat{M_2^{-1}} & \widehat{M_2^{-2}} & 0 \\ \widehat{M_3^0} & \widehat{M_3^{-1}} & \widehat{M_3^{-2}} & \widehat{M_3^{-3}} \end{bmatrix}$$

Mais tous les termes X dans la matrice du développement local sont inutilement calculés, et les 0 de la matrice du développement multipôle ajoutent des opérations inutiles au calcul. Nous pouvons éviter de prendre en compte les 0 à l'aide des BLAS triangulaires (comme la routine $ZTRMM$, voir [46]) et d'une transposition :

$$\begin{bmatrix} \widehat{L}_0^0 & \widehat{L}_1^0 & \widehat{L}_2^0 & \widehat{L}_3^0 \\ X & \widehat{L}_1^1 & \widehat{L}_2^1 & \widehat{L}_3^1 \\ X & X & \widehat{L}_2^2 & \widehat{L}_3^2 \\ X & X & X & \widehat{L}_3^3 \end{bmatrix} = \begin{bmatrix} \widehat{M_0^0} & \widehat{M_1^0} & \widehat{M_2^0} & \widehat{M_3^0} \\ 0 & \widehat{M_1^{-1}} & \widehat{M_2^{-1}} & \widehat{M_3^{-1}} \\ 0 & 0 & \widehat{M_2^{-2}} & \widehat{M_3^{-2}} \\ 0 & 0 & 0 & \widehat{M_3^{-3}} \end{bmatrix} \begin{bmatrix} O_0^0 & O_1^0 & O_2^0 & O_3^0 \\ O_1^0 & O_2^0 & O_3^0 & O_4^0 \\ O_2^0 & O_3^0 & O_4^0 & O_5^0 \\ O_3^0 & O_4^0 & O_5^0 & O_6^0 \end{bmatrix} \tag{4.35}$$

Les termes X de la matrice du développement local sont cependant toujours inutilement calculés. De plus, le gain offert par les BLAS est nettement meilleur avec des routines BLAS de niveau 3 : ceci nécessiterait la concaténation de plusieurs matrices de développements multipôles d'une part, et de plusieurs matrices de développements locaux d'autre part, au

sein de l'équation (4.35) (comme en section 5.3). Mais la matrice contenant les développements multipôles ne serait alors plus triangulaire : l'utilisation des BLAS de niveau 3 n'est donc pas appropriée à ce calcul. De plus, comme le montre la section 4.3.1.3, le calcul *M2L* le long de l'axe **z** représente la partie la moins coûteuse du calcul *M2L* avec rotations. Toutes ces raisons nous amènent à penser que le gain prévisible de l'introduction de routines BLAS dans le calcul *M2L* avec rotations est trop faible pour justifier son implémentation.

La seule façon d'accélérer les calculs *M2L* avec rotations semble donc résider dans l'écriture de nos propres routines BLAS dédiées. Cependant cette réécriture des boucles en tenant compte de la hiérarchie mémoire de l'architecture sous-jacente et en incluant de la programmation assembleur, à la manière des techniques utilisées dans l'implémentation des routines BLAS standards, se devrait d'être hautement spécialisée (et donc ardue) et ne serait de toute façon pas portable : cette solution n'a donc pas été essayée.

4.3.3.2 Noyau de hauteur simple

Puisque dans le cas d'un noyau de hauteur simple, les calculs nécessaires se prêtent encore moins à un produit matrice-vecteur (les termes des matrices triangulaires du noyau de hauteur double de l'équation (4.35) étant désormais aussi nuls « sous » la diagonale secondaire), un calcul avec des routines BLAS serait encore moins efficace.

Au final, nous avons donc choisi de ne pas introduire de routines BLAS dans nos calculs *M2L* avec rotations car nous considérons que celles-ci n'apporteront pas de gain significatif.

4.3.4 Stabilité numérique

La stabilité numérique du calcul récursif des coefficients de rotation a par exemple été étudiée dans [36] et [125], mais pour d'autres formules de récursion que celles de l'équation (4.27). Afin de vérifier nos formules, nous effectuons des tests numériques sur le calcul FMM complet : comme le montre la figure 4.7, aucune instabilité numérique n'est introduite par notre utilisation des rotations puisque les deux graphiques sont identiques. Ceci est aussi valable pour un noyau *M2L* de hauteur double et pour de plus grandes hauteurs de l'octree.

En conclusion de ce chapitre, les deux améliorations par FFT (par blocs) et par rotations réduisent donc la complexité théorique du calcul *M2L*, mais l'utilisation d'une FFT par blocs, contrairement aux rotations, requiert d'importants espaces mémoire supplémentaires et peut engendrer des instabilités numériques. Nous allons désormais proposer une approche alternative pour accélérer le calcul de l'opérateur *M2L*.

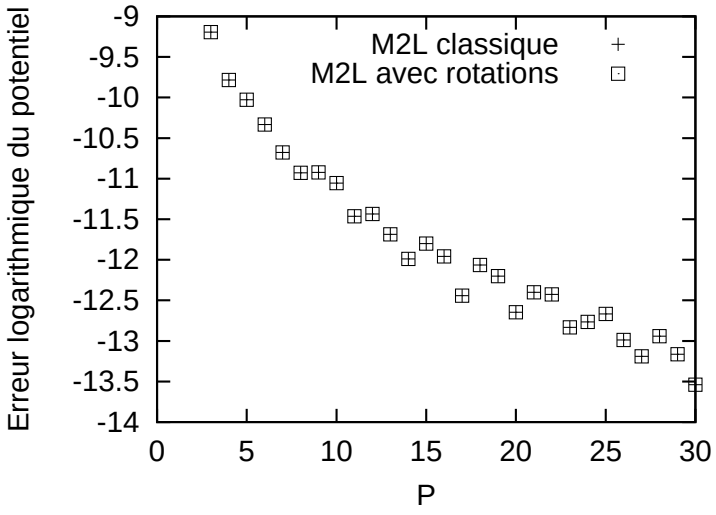


FIG. 4.7 – Erreur logarithmique pour le potentiel en fonction de P, pour un noyau *M2L* de hauteur simple, pour le calcul *M2L* avec et sans rotations. Tests réalisés sur une distribution uniforme de 10 000 particules avec un octree de hauteur 4. L'erreur tracée représente l'erreur absolue maximale sur toutes les particules.

Chapitre 5

Formulation matricielle et implémentation avec les routines BLAS

Sommaire

Les BLAS (Basic Linear Algebra Subprograms, voir [46] [47] [83]) sont une interface standard pour des opérations d'algèbre linéaire usuelles telles un produit scalaire de 2 vecteurs (BLAS de niveau 1), un produit matrice-vecteur (BLAS de niveau 2) ou un produit matrice-matrice (BLAS de niveau 3). Optimisées pour les pipelines d'exécution des unités de calcul flottant et pour la hiérarchie mémoire des architectures modernes, les routines BLAS offrent une implémentation efficace de ces opérations. Et plus le niveau des BLAS est élévé, plus l'accélération apportée est importante. Les BLAS sont de plus portables et aisés à obtenir.

Ces routines BLAS ont déjà été utilisées pour des algorithmes hiérarchiques en $\mathcal{O}(N)$ dédiés au problème à N-corps par Hu & Johnsson [74] avec la méthode d'Anderson. Cette méthode, présentée en section 2.3.3.8, a recours à des développements différents et donc à d'autres opérateurs de translation/conversion, mais présente le même algorithme pour les phases de remontée et de descente.

Nous proposons ici la première formulation matricielle, et son implémentation avec les routines BLAS, de l'opérateur *M2L* de la FMM pour des noyaux de hauteurs simple et double. Afin d'obtenir la meilleure efficacité possible, nous présentons aussi plusieurs façons d'utiliser les BLAS de niveau 3 pour l'opérateur *M2L*, dont deux stockages de données inédits qui permettent d'éviter des copies.

5.1 L'opérateur *M2L* vu comme un produit matrice-vecteur.

Alors que les formules originales de la FMM établies par Greengard & Rokhlin (voir [67]) ne permettent pas une réécriture de leurs opérateurs *M2M*, *M2L* et *L2L* en produits matrice-vecteur, ceci s'effectue aisément avec les formules plus simples présentées dans [51], [52] ou [124]. L'algorithme complet de la FMM a aussi été réécrit sous forme matricielle dans [116] (avec toutefois des notations différentes) : nous nous concentrons ici uniquement sur la réécriture de l'opérateur *M2L* en un produit matrice-vecteur.

Nous considérons tout d'abord le noyau *M2L* de hauteur double, et contrairement à [116], nous ne considérons que les termes non nuls dans les développements multipôles et locaux, à savoir les termes de degré j et d'ordre k tels que $0 \leq j$ et $|k| \leq j$. Nous insistons aussi sur le fait que lors d'une opération *M2L*, seuls les termes d'ordre positif du développement local sont calculés (voir section 2.3.3.6). Des relations de symmétrie dans la matrice de transfert (sa définition est donnée plus loin), ou dans ses sous-matrices, pourraient certes être repérées lorsque les termes d'ordre négatif sont aussi calculés, mais ceci résulterait au final en un calcul plus lent que celui des termes d'ordre positif uniquement (les termes d'ordre négatif étant alors obtenus grâce à l'équation (2.13)). Par ailleurs, nous ne considérons désormais que des opérations d'algèbre linéaire (celles implémentées dans les BLAS) : c'est pourquoi nous devons stocker dans les développements multipôles aussi bien les termes d'ordre positif que les termes d'ordre négatif. Malgré la propriété de symmétrie présentée à l'équation (2.11), nous ne pouvons en effet pas écrire l'opérateur *M2L* comme un produit matrice-vecteur tout en ne stockant que les termes d'ordre positif dans les développements multipôles.

Soit un développement local L_j^k avec $0 \leq j \leq P$, $0 \leq k \leq j$, nous définissons le *vecteur développement local* correspondant, ou *vecteur local*, de taille $\frac{(P+1)(P+2)}{2}$ par :

$$\forall i_1 \in [\![0, \tfrac{(P+1)(P+2)}{2} - 1]\!], \quad \begin{cases} \mathbf{v}^L(i_1) = L_j^k, \\ i_1 = \frac{j(j+1)}{2} + k. \end{cases}$$

Avec $P = 2$, nous avons ainsi :

$$\mathbf{v}^L = \begin{bmatrix} L_0^0 \\ \hline L_1^0 \\ L_1^1 \\ \hline L_2^0 \\ L_2^1 \\ L_2^2 \end{bmatrix} .$$

Soit un développement multipôle M_n^l avec $0 \leq n \leq P$, $|l| \leq j$, nous définission le *vecteur développement multipôle* correspondant, ou *vecteur multipôle* de taille $(P+1)^2$ par :

$$\forall i_2 \in [\![0, (P+1)^2 - 1]\!], \quad \begin{cases} \mathbf{v}^M(i_2) = M_n^l, \\ i_2 = n(n+1) + l. \end{cases}$$

Avec $P = 2$, nous avons ainsi :

$$\mathbf{v}^M = \begin{bmatrix} M_0^0 \\ \hline M_1^{-1} \\ M_1^0 \\ M_1^1 \\ \hline M_2^{-2} \\ M_2^{-1} \\ M_2^0 \\ M_2^1 \\ M_2^2 \end{bmatrix} .$$

Soit la fonction de transfert *M2L* O_j^k avec $0 \leq j \leq 2P$ et $|k| \leq j$, nous définissons la *matrice de transfert* M2L de taille $\frac{(P+1)(P+2)}{2} \times (P+1)^2$ par :

$$\forall (i_1, i_2) \in [\![0, \tfrac{(P+1)(P+2)}{2} - 1]\!] \times [\![0, (P+1)^2 - 1]\!], \quad \begin{cases} \mathbf{T}_{M2L}(i_1, i_2) = O_{j+n}^{-k-l}, \\ i_1 = \frac{j(j+1)}{2} + k, \\ i_2 = n(n+1) + l. \end{cases}$$

Avec $P = 2$, nous avons ainsi :

$$\mathbf{T}_{M2L} = \left[\begin{array}{ccc|ccccc} O_0^0 & O_1^1 & O_1^0 & O_1^{-1} & O_2^2 & O_2^1 & O_2^0 & O_2^{-1} & O_2^{-2} \\ \hline O_1^0 & O_2^1 & O_2^0 & O_2^{-1} & O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} \\ O_1^{-1} & O_2^0 & O_2^{-1} & O_2^{-2} & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} \\ \hline O_2^0 & O_3^1 & O_3^0 & O_3^{-1} & O_4^2 & O_4^1 & O_4^0 & O_4^{-1} & O_4^{-2} \\ O_2^{-1} & O_3^0 & O_3^{-1} & O_3^{-2} & O_4^1 & O_4^0 & O_4^{-1} & O_4^{-2} & O_4^{-3} \\ O_2^{-2} & O_3^{-1} & O_3^{-2} & O_3^{-3} & O_4^0 & O_4^{-1} & O_4^{-2} & O_4^{-3} & O_4^{-4} \end{array}\right] . \tag{5.1}$$

L'opérateur *M2L* peut alors être calculé comme le produit matrice-vecteur suivant :

$$\mathbf{v}^L \;=\; \mathbf{T}_{M2L} \;.\; \mathbf{v}^M.$$

Avec un noyau *M2L* de hauteur simple, les termes O_j^k de la matrice de transfert *M2L* doivent être considérés comme nuls pour $j > P$, ce qui donne pour $P = 2$:

$$\mathbf{T}_{M2L} = \left[\begin{array}{c|ccc|ccccc} O_0^0 & O_1^1 & O_1^0 & O_1^{-1} & O_2^2 & O_2^1 & O_2^0 & O_2^{-1} & O_2^{-2} \\ \hline O_1^0 & O_2^1 & O_2^0 & O_2^{-1} & 0 & 0 & 0 & 0 & 0 \\ O_1^{-1} & O_2^0 & O_2^{-1} & O_2^{-2} & 0 & 0 & 0 & 0 & 0 \\ \hline O_2^0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_2^{-1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ O_2^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] . \tag{5.2}$$

Remarque 5.1. *Nous pouvons aussi bien sûr écrire les opérateurs* M2M *et* L2L *comme des produits matrice-vecteur.*

5.2 BLAS de niveau 2

Maintenant que nous avons réécrit l'opérateur *M2L* en un produit matrice-vecteur, il est facile d'utiliser des routines BLAS de niveau 2 pour effectuer son calcul : l'accélération apportée par les BLAS est alors directement exploitée.

Nos matrices sont stockées par colonnes, et nous utilisons la matrice de transfert sous sa forme transposée, car il est généralement plus efficace de calculer $C \leftarrow A^T.B$ avec A stockée par lignes, que $C \leftarrow A.B$ avec A stockée par colonnes ; en effet, lorsqu'on étudie l'implémentation des BLAS (voir [2] pour une implémentation par défaut) la première solution conduit à moins d'écritures, avec certes plus de lectures, que la seconde.

5.2.1 Noyau *M2L* de hauteur double

La matrice $\mathbf{T}_{M2L}$ est dense : l'utilisation de la routine $ZGEMV$ (BLAS de niveau 2) est donc évidente, et cette routine va automatiquement optimiser le calcul du produit matrice-vecteur pour l'architecture super-scalaire sous-jacente. Afin de la distinguer des autres méthodes avec BLAS que nous allons utiliser par la suite, nous la nommons *BLAS_plein*.

Remarque 5.2. *Une autre possibilité est de séparer* $\mathbf{T}_{M2L}$ *en une matrice carrée* $\mathbf{T'}_{M2L}$ *de taille* $\frac{(P+1)(P+2)}{2} \times \frac{(P+1)(P+2)}{2}$ *qui correspond aux termes d'ordre positif ou nul dans le vecteur multipôle, et une matrice* $\mathbf{T''}_{M2L}$ *de taille* $\frac{(P+1)(P+2)}{2} \times \frac{P(P+1)}{2}$ *qui correspond aux termes d'ordre strictement négatif dans le vecteur multipôle. L'intérêt réside ici dans la symmétrie de la matrice* $\mathbf{T'}_{M2L}$*, mais l'utilisation d'une « routine BLAS symmétrique », comme les routines* $ZSYMV$ *ou* $ZSYMM$*, n'accélère pas plus le calcul que des appels aux routines* $ZGEMV$ *ou* $ZGEMM$ *: l'intérêt principal lorsqu'on utilise ces routines BLAS symmétriques est que seule la moitié de la matrice doit être écrite, ce qui ne nous intéresse pas ici puisque la construction des matrices de transfert* M2L *est précalculée (voir section 2) et a donc très peu d'impact sur les temps CPU. Des tests réalisés avec des BLAS de niveau* 3 *(voir section 5.3) et avec le découpage adapté pour* $\mathbf{T'}_{M2L}$ *et* $\mathbf{T''}_{M2L}$ *(voir section 5.3.3), ont confirmé qu'une telle alternative n'améliore pas les performances.*

5.2.2 Noyau *M2L* de hauteur simple

Dans le cas d'un noyau de hauteur simple, la matrice $\mathbf{T}_{M2L}$ est désormais creuse : le recours à un appel BLAS de niveau 2 comme la routine $ZGEMV$ ne serait clairement pas

efficace, vus les nombreux zéros inutilement intégrés au calcul. Nous devons donc décomposer le produit matrice-vecteur creux en un produit par blocs pleins.

Nous rappellons tout d'abord les techniques utilisées dans les implémentations haute performance des BLAS (voir par exemple [48]). Le but final est ici de remplir au mieux les pipelines d'exécution des unités de calcul flottant afin d'atteindre la performance crête du processeur. Lors de l'écriture des boucles correspondant aux opérations d'algèbre linéaire des BLAS, l'ordre des différentes boucles est premièrement choisi avec soin afin de maximiser la localité spatiale et temporelle entre les données. Ensuite, un premier niveau de blocage est introduit afin de permettre une réutilisation du cache (*cache reuse*) qui évite les chargements multiples d'une même donnée dans le cache. Si l'on considère une mémoire avec plusieurs niveaux de cache, et/ou la présence d'un TLB (*Translation Lookaside Buffer*), des niveaux supplémentaires de blocage peuvent être introduits. Une *dimension principale* (*leading dimension*)[1] inappropriée pour certaines matrices peut causer des transferts inutiles au sein de la hiérarchie mémoire et une sous-utilisation du cache : des tableaux locaux sont alors utilisés pour stocker temporairement des sous-matrices. D'autres techniques telles que le déroulage de boucles (*loop unrolling*), le blocage de registres (*register blocking*), le préchargement de données (*data prefetch*), etc. sont aussi utilisées en fonction de la machine. Toutes ces techniques sont contrôlées par des paramètres dépendants de la machine utilisée. Une présentation détaillée du principe des routines BLAS est disponible en annexe de la thèse de G. Latu [81].

Dans [76], il a été montré que les routines BLAS de niveau 3 peuvent être efficacement implémentées avec la seule routine BLAS de niveau 3 $GEMM$ et quelques routines BLAS de niveau 2. Par souci de portabilité, mais aussi de simplicité, nous préférons adopter la même approche. Nous allons donc décomposer, de la façon la plus efficace possible, le produit matrice-vecteur creux correspondant à l'opérateur *M2L* en plusieurs produits de blocs pleins, chacun de ces produits étant efficacement effectué par la routine $ZGEMV$. La routine $ZGEMV$ est utilisée ici car nous traitons de produits matrice-vecteur, mais en section 5.3 nous verrons comment utiliser des produits matrice-matrice, et la routine $ZGEMM$ sera alors utilisée comme dans [76]. Toutes les techniques rappelées ci-dessus seront mises en œuvre mais laissées autant que possible aux appels BLAS $ZGEMV$ / $ZGEMM$ sous-jacents (et dont l'implémentation est optimisée pour la machine utilisée). Nous souhaitons insister sur un point important : dans notre implémentation de la FMM, nous sommes libres dans le stockage en mémoire des blocs de $\mathbf{T}_{M2L}$ car leur construction est précalculée à chaque niveau de l'octree lors de la phase de descente de la FMM (voir section 2). Par conséquent, la plupart des problèmes dus à une dimension principale inappropriée pour les matrices utilisées ne concerneront pas nos découpages par blocs. Par ailleurs, nous rappellons aussi que les valeurs de P sont limitées : en pratique elles varient généralement de 3, pour les basses précisions, à 15 pour les hautes précisions. Nous ne pouvons donc pas inclure trop de zéros supplémentaires dans la décomposition en blocs de la matrice de transfert *M2L* avec un noyau de hauteur simple car chaque surcoût est amplifié par le très grand nombre d'appels à l'opérateurs *M2L*, et particulièrement coûteux pour les petites valeurs de P.

Avant de présenter les trois décompositions par blocs utilisées, nous détaillons la structure sous-jacente de $\mathbf{T}_{M2L}$. Comme suggéré dans sa représentation à l'équation (5.1), cette

[1] Cette *dimension principale* correspond à la taille de la colonne (avec un stockage par colonnes) de la matrice englobante lorsque le calcul BLAS ne porte que sur une sous-matrice (voir l'interface des routines BLAS, par exemple [2]).

matrice peut en fait être vue comme une « concaténation » de plusieurs sous-matrices que nous allons noter $\mathbf{B}_{I_1,I_2}$.

5.2.2.1 Structure par blocs de $\mathbf{T}_{M2L}$.

Pour tous $(I_1, I_2) \in [\![0, P]\!]^2$, nous notons $\mathbf{B}_{I_1,I_2}$ la matrice de taille $(I_1 + 1) \times (2I_2 + 1)$ définie par :

$$\mathbf{B}_{I_1,I_2}(i_1, i_2) = O_{I_1+I_2}^{-i_1+I_2-i_2} \qquad \forall (i_1, i_2) \in [\![0, I_1]\!] \times [\![0, 2I_2]\!].$$

Dans le cas d'un noyau de hauteur double la matrice $\mathbf{T}_{M2L}$ aurait la structure par blocs suivante :

$$\mathbf{T}_{M2L} = \left[\begin{array}{c|c|c|c} \mathbf{B}_{0,0} & \mathbf{B}_{0,1} & \dots & \mathbf{B}_{0,P} \\ \hline \mathbf{B}_{1,0} & \mathbf{B}_{1,1} & \dots & \mathbf{B}_{1,P} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline \mathbf{B}_{P,0} & \mathbf{B}_{P,1} & \dots & \mathbf{B}_{P,P} \end{array}\right] .$$

Comme nous nous plaçons ici dans le cas d'un noyau de hauteur simple, les blocs $\mathbf{B}_{I_1,I_2}$ sont considérés comme nuls pour $I_1 + I_2 > P$.

Pour $P = 2$, nous avons donc :

$$\mathbf{T}_{M2L} = \left[\begin{array}{c|c|c} \mathbf{B}_{0,0} & \mathbf{B}_{0,1} & \mathbf{B}_{0,2} \\ \hline \mathbf{B}_{1,0} & \mathbf{B}_{1,1} & 0_{3\times 5} \\ \hline \mathbf{B}_{2,0} & 0_{5\times 3} & 0_{5\times 5} \end{array}\right] \tag{5.3}$$

où $0_{M\times N}$ désigne la matrice nulle de taille $M \times N$.

Il peut être noté qu'en considérant aussi une décomposition par blocs du vecteur multipôle, nous obtenons une version simple d'un produit par blocs pour l'opérateur *M2L* avec un noyau de hauteur simple. En effet en notant $\mathbf{v}_I^M$, $\forall I \in [\![0, P]\!]$, le vecteur suivant de taille $2I + 1$

$$\mathbf{v}_I^M(i) = M_I^{i-I} \qquad \forall i \in [\![0, 2I]\!],$$

nous avons (pour $P = 2$)

$$\mathbf{v}^M = \left[\begin{array}{c} \mathbf{v}_0^M \\ \hline \mathbf{v}_1^M \\ \hline \mathbf{v}_2^M \end{array}\right]$$

et

$$\mathbf{T}_{M2L} \ . \ \mathbf{v}^M = \left[\begin{array}{c} \mathbf{B}_{0,0}\mathbf{v}_0^M + \mathbf{B}_{0,1}\mathbf{v}_1^M + \mathbf{B}_{0,2}\mathbf{v}_2^M \\ \hline \mathbf{B}_{1,0}\mathbf{v}_0^M + \mathbf{B}_{1,1}\mathbf{v}_1^M \\ \hline \mathbf{B}_{2,0}\mathbf{v}_0^M \end{array}\right] .$$

A cause des grandes différences entre les tailles des différents blocs $\mathbf{B}_{I_1,I_2}$, cette décomposition par blocs, avec un appel BLAS (routine $ZGEMV$) pour chaque produit entre 2 blocs, ne serait cependant pas la plus efficace. Nous présentons donc maintenant 3 décompositions qui peuvent être efficacement couplées avec des appels BLAS.

5.2.2.2 Décomposition BLAS_bande.

Notre première décomposition est strictement basée sur la structure sous-jacente de $\mathbf{T}_{M2L}$ telle qu'elle est présentée à l'équation (5.3) : cette structure ressemble à un « escalier renversé », chaque marche ayant une taille différente (hauteur et profondeur). A chaque marche correspond une unique *bande* (*band*).

Le premier choix à considérer porte sur une décompoition de $\mathbf{T}_{M2L}$ en *bandes* horizontales (c'est-à-dire dans la direction des lignes) ou verticales (dans la direction des colonnes). Nos matrices sont stockées par colonnes pour les appels BLAS, mais comme nous utilisons la transposée de la matrice de transfert (voir l'introduction de la section 5.2), celle-ci est stockée par lignes. Il semble qu'il vaille donc mieux choisir des *bandes* dans la direction des lignes. De plus, des *bandes* dans la direction des lignes imposent plusieurs parcours du vecteur multipôle (un pour chaque *bande*) contre un seul parcours du vecteur local. En d'autres termes, avec des *bandes* dans la direction des lignes, les blocs résultants du vecteur local sont mis à jour l'un après l'autre, alors que les données correspondantes du vecteur multipôle peuvent être rechargées plusieurs fois au cours du produit matrice-vecteur par blocs. D'un autre côté, des *bandes* dans la direction des colonnes imposent plusieurs parcours du vecteur local contre un seul parcours du vecteur multipôle. Puisque le vecteur local est parcouru pour une mise à jour (c'est-à-dire à la fois en lecture et en écriture) et le vecteur multipôle en lecture seule, il est plus efficace d'utiliser des *bandes* dans la direction des lignes.

Il y a donc $P+1$ *bandes* et la première bande est constituée de la concaténation des blocs $\mathbf{B}_{0,0},\ \mathbf{B}_{0,1}\ \ldots\ \mathbf{B}_{0,P}$, la seconde de la concaténation des blocs $\mathbf{B}_{1,0},\ \mathbf{B}_{1,1}\ \ldots\ \mathbf{B}_{1,P-1}$, et ainsi de suite jusqu'à la dernière *bande* qui correspond au seul bloc $\mathbf{B}_{P,0}$ (voir figure 5.1).

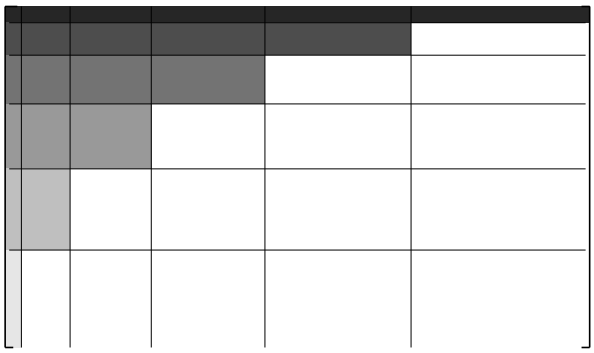


FIG. 5.1 – Décomposition *BLAS_bande* pour $P = 5$: les différents niveaux de gris identifient les différentes *bandes*.

Chaque *bande* est ensuite stockée séparément et traitée par un appel à la routine $ZGEMV$. Nous stockons chaque *bande* par ligne et utilisons sa transposée lors de l'appel BLAS. La dimension principale est ici la hauteur de la *bande*. Il peut être noté que la première *bande* amène à un simple produit scalaire alors que la dernière *bande* conduit à un produit terme à terme entre 2 vecteurs : ces deux bandes ne permettent pas d'obtenir beaucoup d'accélération. Cette décomposition sera appelée par la suite décomposition *BLAS_bande*.

Un problème évident avec une telle décomposition est qu'à mesure que P augmente les premières *bandes* sont trop minces et trop longues alors que les dernières sont trop hautes et trop courtes. Ceci empêche la routine BLAS de décomposer pleinement, en interne, ce

type de *bande* en fonction de la hiérarchie mémoire du processeur. Nous allons donc essayer d'utiliser des *bandes* avec une hauteur plus importante (c'est-à-dire un plus grand nombre de lignes) afin de relâcher ces contraintes sur l'efficacité des BLAS.

5.2.2.3 Décomposition BLAS_cbande.

Dans [76], le produit matrice-matrice triangulaire était effectué par des routines $ZGEMM$ grâce à une décomposition de la matrice triangulaire en « bandes » (« strips »), que nous désignerons ici plutôt par « lanières » afin de les distinguer des *bandes* de la section précédente. Ces lanières pouvaient être orientées verticalement ou horizontalement, impliquant respectivement des blocs colonnes ou des blocs lignes. Entre ces 2 orientations, celle retenue minimisait le nombre de mises à jour (c'est-à-dire de lectures-écritures) par rapport au nombre de lectures seules. Comme expliqué à la section précédente, ceci est réalisé dans notre cas avec des lanières dans la direction horizontale (à savoir des blocs lignes). Nous disposons ainsi d'une décomposition de $\mathbf{T}_{M2L}$ en lanières horizontales dont la « hauteur » est paramètrable, toutes les lanières ayant la même hauteur. Nous désignons de telles lanières *cbandes* pour « *bandes* constantes » (« constant bands ») en référence aux *bandes* utilisées dans la décomposition *BLAS_bande*, et leur hauteur commune est désormais nommée dim_{cbande}. Cette nouvelle décomposition sera appelée décomposition *BLAS_cbande*.

Lorsqu'une *cbande* recouvre, totalement ou partiellement, 2 *bandes* consécutives (telles que définies pour la décomposition *BLAS_bande*), des zéros doivent être ajoutés à la fin de la *bande* la plus basse de sorte que la longueur de la *cbande* soit égale à celle de la *bande* la plus haute. Nous avons donc implémenté une version plus efficace de la décomposition *BLAS_cbande*, qui saute ces zéros en utilisant deux appels BLAS : un premier appel pour la *cbande* de longueur égale à la *bande* la plus basse et un deuxième appel pour les données restantes à la fin de la *bande* la plus haute. Des tests de performance ont confirmé que ceci est plus efficace qu'un unique appel BLAS sur la *cbande* de longueur égale à la *bande* la plus haute (et incluant donc les zéros supplémentaires). Lorsqu'une *cbande* recouvre 3 *bandes* ou plus, seuls les zéros entre la première et la deuxième *bande* sont sautés : ceci élimine la plupart des zéros tout en évitant de nombreux appels BLAS avec peu d'opérations. Le bloc supérieur droit de $\mathbf{T}_{M2L}$, à savoir le bloc $\mathbf{B}_{0,P}$ dont la hauteur est 1, et le bloc inférieur gauche $\mathbf{B}_{P,0}$, dont la longueur est 1, sont traités séparément : ils impliqueraient en effet trop de zéros supplémentaires s'ils étaient directement inclus dans la décomposition *BLAS_cbande*. Tout ceci est représenté à la figure 5.2.

Comme pour la décomposition *BLAS_bande*, les *cbandes* sont stockées séparément, avec une dimension principale égale à leur hauteur (constante), et considérées comme transposées pour la routine $ZGEMV$.

Le problème avec cette décomposition *BLAS_cbande* est qu'avec de grandes dim_{cbande} nous introduisons des zéros inutiles dans le calcul, alors qu'avec de petites dim_{cbande} nous obtenons des blocs rectangulaires longs et fins, mal adaptés aux appels BLAS. C'est pourquoi la hauteur optimale de nos *cbandes* devra être déterminée expérimentalement en fonction de P et de la machine utilisée.

5.2.2.4 Décomposition BLAS_bloc.

Notre dernière décomposition est basée sur le caractère récursif de la structure sous-jacente de $\mathbf{T}_{M2L}$, dans le cas d'un noyau de hauteur simple imposant $O_j^k = 0, \forall j > P$. Plus

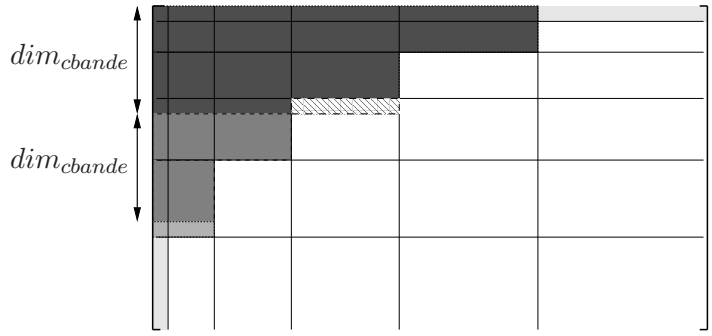


FIG. 5.2 – Décomposition *BLAS_cbande* pour $P = 5$: les différents niveaux de gris identifient les différentes *cbandes*. Les zéros supplémentaires sont marqués par des rayures. Le bloc supérieur droit et le blocs inférieur gauche sont traités séparément.

précisément, nous notons désormais $\mathbf{T}_{M2L}$ par $\mathbf{T}_{0,0}[P]$ où $\mathbf{T}_{I_1,I_2}[P]$, $0 \leq I_1 \leq P$, $0 \leq I_2 \leq P$, désigne la matrice de taille $\frac{(P+1)(P+2)}{2} \times (P+1)^2$ définie par :

$$\mathbf{T}_{I_1,I_2}[P] = \left[\begin{array}{c|c|c|c|c} \mathbf{B}_{I_1,I_2} & \mathbf{B}_{I_1,I_2+1} & \dots & \mathbf{B}_{I_1,P-(I_1+1)} & \mathbf{B}_{I_1,P-I_1} \\ \hline \mathbf{B}_{I_1+1,I_2} & \mathbf{B}_{I_1+1,I_2+1} & \dots & \mathbf{B}_{I_1+1,P-(I_1+1)} & 0 \\ \hline \vdots & \vdots & \vdots & \vdots & \vdots \\ \hline \mathbf{B}_{P-(I_2+1),I_2} & \mathbf{B}_{P-(I_2+1),I_2+1} & \dots & 0 & 0 \\ \hline \mathbf{B}_{P-I_2,I_2} & 0 & \dots & 0 & 0 \end{array}\right] .$$

Lorsque $I_1 + I_2 \leq P$, cette matrice est pleine : nous la notons alors $D_{I_1,I_2}[P]$. Plus précisément, nous avons :

$$\mathbf{D}_{I_1,I_2}[P] = \left[\begin{array}{c|c|c|c} \mathbf{B}_{I_1,I_2} & \mathbf{B}_{I_1,I_2+1} & \dots & \mathbf{B}_{I_1,P} \\ \hline \mathbf{B}_{I_1+1,I_2} & \mathbf{B}_{I_1+1,I_2+1} & \dots & \mathbf{B}_{I_1+1,P} \\ \hline \vdots & \vdots & \vdots & \vdots \\ \hline \mathbf{B}_{P,I_2} & \mathbf{B}_{P,I_2+1} & \dots & \mathbf{B}_{P,P} \end{array}\right] .$$

Nous avons alors la décomposition récursive suivante pour $0 \leq I_1 \leq P$, $0 \leq I_2 \leq P$:
– pour les valeurs impaires de $P = 2Q + 1$:

$$\mathbf{T}_{I_1,I_2}[P] = \mathbf{T}_{I_1,I_2}[2Q+1] = \left[\begin{array}{c|c} \mathbf{D}_{I_1,I_2}[Q] & \mathbf{T}_{I_1,I_2+Q+1}[Q] \\ \hline \mathbf{T}_{I_1+Q+1,I_2}[Q] & 0 \end{array}\right] , \qquad (5.4)$$

– et pour les valeurs paires de $P = 2Q$:

$$\mathbf{T}_{I_1,I_2}[P] = \mathbf{T}_{I_1,I_2}[2Q] = \left[\begin{array}{c|c} \mathbf{D}_{I_1,I_2}[Q] & \mathbf{T}_{I_1,I_2+Q+1}[Q-1] \\ \hline \mathbf{T}_{I_1+Q+1,I_2}[Q-1] & 0 \end{array}\right] .$$

Nous pouvons à nouveau décomposer récursivement $\mathbf{T}_{I_1,I_2+Q+1}[\ldots]$ et $\mathbf{T}_{I_1+Q+1,I_2}[\ldots]$, qui sont aussi des matrices creuses, et dont la disposition des sous-blocs présente exactement la même structure récursive que $\mathbf{T}_{I_1,I_2}$ mais avec un P deux fois plus petit. Le cas terminal de la récursion est déterminé par la valeur de Q.

Considérons par exemple le cas $P = 3$. Nous avons alors :

$$\mathbf{T}_{M2L} = \mathbf{T}_{0,0}[3] = \mathbf{T}_{0,0}[2 \times 1 + 1] =$$

$$\left[\begin{array}{ccccc|ccccc|ccccccc}
O_0^0 & O_1^1 & O_1^0 & O_1^{-1} & & O_2^2 & O_2^1 & O_2^0 & O_2^{-1} & O_2^{-2} & O_3^3 & O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} \\
O_1^0 & O_2^1 & O_2^0 & O_2^{-1} & & O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
O_1^{-1} & O_2^0 & O_2^{-1} & O_2^{-2} & & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
O_2^0 & O_3^1 & O_3^0 & O_3^{-1} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
O_2^{-1} & O_3^0 & O_3^{-1} & O_3^{-2} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
O_2^{-2} & O_3^{-1} & O_3^{-2} & O_3^{-3} & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
O_3^0 & 0 & 0 & 0 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
O_3^{-1} & 0 & 0 & 0 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
O_3^{-2} & 0 & 0 & 0 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
O_3^{-3} & 0 & 0 & 0 & & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right]$$

qui peut être décomposé d'après l'équation (5.4) en $\mathbf{D}_{0,0}[1]$

$$\mathbf{D}_{0,0}[1] = \left[\begin{array}{ccc|ccc}
O_0^0 & O_1^1 & O_1^0 & O_1^{-1} \\
\hline
O_1^0 & O_2^1 & O_2^0 & O_2^{-1} \\
O_1^{-1} & O_2^0 & O_2^{-1} & O_2^{-2}
\end{array}\right] \quad ,$$

et en $\mathbf{T}_{0,2}[1]$ et $\mathbf{T}_{2,0}[1]$

$$\mathbf{T}_{0,2}[1] = \left[\begin{array}{ccccc|ccccccc}
O_2^2 & O_2^1 & O_2^0 & O_2^{-1} & O_2^{-2} & O_3^3 & O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} \\
\hline
O_3^2 & O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
O_3^1 & O_3^0 & O_3^{-1} & O_3^{-2} & O_3^{-3} & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}\right]$$

$$\mathbf{T}_{2,0}[1] = \left[\begin{array}{ccc|ccc}
O_2^0 & O_3^1 & O_3^0 & O_3^{-1} \\
O_2^{-1} & O_3^0 & O_3^{-1} & O_3^{-2} \\
O_2^{-2} & O_3^{-1} & O_3^{-2} & O_3^{-3} \\
\hline
O_3^0 & 0 & 0 & 0 \\
O_3^{-1} & 0 & 0 & 0 \\
O_3^{-2} & 0 & 0 & 0 \\
O_3^{-3} & 0 & 0 & 0
\end{array}\right] \quad .$$

L'avantage de cette décomposition, appelée *BLAS_bloc*, est que nous utilisons autant que possible des matrices pleines (à savoir $\mathbf{D}_{I_1,I_2}$, qui peut être vu comme le « plus gros carré supérieur gauche » en nombre de blocs $\mathbf{B}_{I_1,I_2}$), qui sont efficacement traitées par la routine $ZGEMV$. Nous ne restreignons pas l'efficacité des BLAS par une hauteur de *bande* comme dans la décomposition *BLAS_bande*, et nous n'introduisons pas de zéros inutiles dans le calcul comme dans la décomposition *BLAS_cbande*. Le cas terminal de la récursion est lui traité comme dans *BLAS_bande*[2], et la valeur optimale s_{bloc} déterminant l'arrêt de la récursion devra être déterminée expérimentalement en fonction de P et de la machine utilisée. Pour les

[2]Les problèmes qui apparaissaient avec les premières et les dernières *bandes* dans la décomposition *BLAS_bande* sont ici minimisés car on réserve le découpage par *bandes* aux cas terminaux de la récursion.

petites valeurs de P, il est probable qu'il ne vaille mieux pas procéder à ce découpage récursif : s_{bloc} sera alors égal à P et la décomposition *BLAS_bande* sera utilisée pour toute la matrice.

Il doit être noté que l'appel récursif sur la sous-matrice « à droite » est réalisé avant l'appel récursif sur la sous-matrice « en bas » : ceci conduit en effet à moins de parcours du vecteur local (accédé en lecture-écriture) que du vecteur multipôle (accédé en lecture seule), de la même façon qu'en page 103.

5.3 BLAS de niveau 3

Un produit matrice-vecteur nécessite un espace mémoire en $\mathcal{O}(N^2)$ (en supposant que la matrice soit carrée) pour un nombre d'opératons en $\mathcal{O}(N^2)$. Comme la vitesse de calcul des processeurs modernes est bien supérieure à la bande passante de la mémoire qui leur est associée, il n'est généralement pas possible de fournir continûment les unités de calcul flottant avec les données requises : ces unités d'exécution restent alors régulièrement inactives dans l'attente du chargement des données manquantes depuis les caches de bas niveau ou depuis la mémoire principale. La performance des BLAS de niveau 2 est par conséquent limité par la vitesse de transfert des données à travers la hiérarchie mémoire, et les BLAS de niveau 2 n'atteignent généralement pas la performance crête du processeur. Par contre, un produit matrice-matrice nécessite un espace mémoire en $\mathcal{O}(N^2)$ (en supposant que les matrices soient carrées) pour un nombre d'opérations en $\mathcal{O}(N^3)$. Il est maintenant plus facile de recouvrir la latence mémoire avec du calcul sur les unités de calcul flottant, et ainsi d'atteindre la performance crête du processeur. Plus précisément, un niveau de blocage est introduit dans les boucles qui effectuent le produit matrice-matrice $C \leftarrow A^T.B$, ce qui correspond à une version par blocs du produit matriciel : tous les calculs faisant intervenir un bloc donné de A sont ainsi effectués avant de charger le bloc suivant de A. Cette réutilisation du cache permet d'éviter les multiples chargements des données de A qui interviendraient dans une version sans blocs du produit matriciel. Nous allons donc essayer de regrouper plusieurs opérations *M2L* en un seul produit matrice-matrice.

Pendant la phase de descente, à un niveau donné de l'octree, toutes les opérations *M2L* qui ont le même vecteur *M2L* entre le centre de la cellule source (contenant le développement multipôle) et le centre de la cellule cible (contenant le développement local) partagent les mêmes fonctions *Outer* (à savoir les O_j^k), et donc la même matrice de transfert *M2L*. En considérant toutes les paires de développements multipôles et locaux qui partagent le même vecteur *M2L*, il est alors possible de concaténer tous leurs vecteurs multipôles afin de former les colonnes d'une unique *matrice développement multipôle* $\mathbf{M}^M$, aussi appelée *matrice multipôle*. Les vecteurs locaux correspondants sont aussi concaténés selon le même ordre en une unique *matrice développement local* $\mathbf{M}^L$, aussi appelée *matrice locale*. Le produit matriciel $\mathbf{M}^L = \mathbf{T}_{M2L} \,.\, \mathbf{M}^M$ calcule alors toutes les opérations *M2L* correspondantes d'un coup comme le montre la figure 5.3. Il peut être noté que le stockage par colonnes de nos matrices pour les appels BLAS est clairement adapté à la concaténation des vecteurs en colonnes d'une même matrice.

Dans une première approche, nous allons considérer que la meilleure efficacité pour les BLAS de niveau 3 est obtenue en concaténant le plus possible de développements multipôles et locaux à chaque fois : nous réviserons cette assertion en section 5.3.3. Nous allons donc voir désormais comment concaténer un nombre maximum de développements mul-

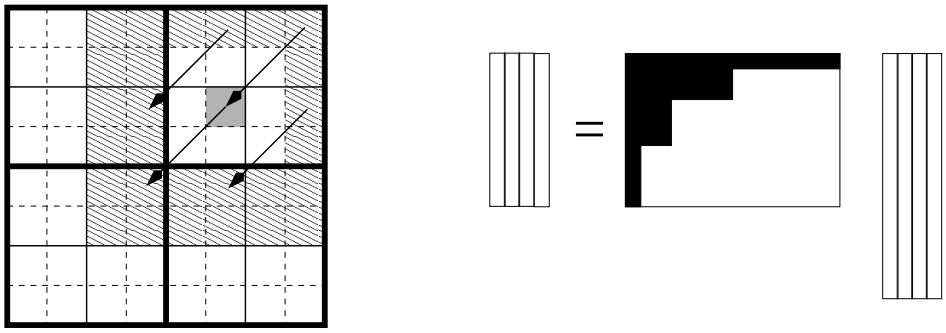


FIG. 5.3 – Concaténation de développements multipôles et locaux afin d'utiliser des BLAS de niveau 3. La liste d'interaction de la cellule grisée est représentée par des rayures. Chacune de ses opérations *M2L* est calculée dans cet exemple avec 3 autres opérations *M2L* qui partagent le même vecteur *M2L*, dans un unique produit matrice-matrice : les vecteurs multipôles et locaux correspondants sont par conséquent concaténés en une matrice multipôle et en une matrice locale, toutes deux ayant 4 colonnes. La matrice de transfert représentée ici correspond à un noyau de hauteur simple.

tipôles et locaux pour une matrice de transfert *M2L* donnée, et nous présenterons ensuite l'implémentation de ce produit matrice-matrice avec des appels BLAS de niveau 3.

Lorqu'on regroupe les développements suivant les vecteurs *M2L* de la liste d'interaction, nous devons aussi prendre en compte le fait que certaines cellules ont une liste d'interaction incomplète : avec des conditions aux limites libres (*free-space boundary conditions* (FBC)), où tout l'espace hors de la boîte de calcul est considéré comme vide, ceci concerne les « cellules frontières » (« border cells ») de chaque niveau de l'octree, à savoir les cellules situées aux bords de la boîte de calcul. Afin de faciliter leur calcul, et par souci d'efficacité, ces cellules avec liste d'interaction incomplète sont traitées à part lorsqu'on utilise des BLAS de niveau 3 pour le calcul *M2L*. Dans le cas des conditions aux limites périodiques (*Periodic Boundary Conditions* (PBC)), où toute la boîte de calcul est dupliquée périodiquement dans chaque dimension, toutes les cellules ont une liste d'interaction complète à chaque niveau et une telle disctinction est inutile.

5.3.1 Calcul des développements locaux des cellules avec liste d'interaction incomplète

Avec des conditions aux limites, les « cellules avec liste d'interaction incomplète » sont toutes les cellules dont le père a au moins un voisin situé hors de l'octree (voir figure 5.4 par exemple).

Afin de traiter toutes les opérations *M2L* portant sur les développements locaux de ces cellules, nous utilisons des *recopies* des vecteurs multipôles et locaux. Concrètement, les développements locaux courants sont copiés dans les colonnes de la matrice locale, les développements multipôles correspondants sont copiés dans les colonnes de la matrice multipôle, et le calcul *M2L* est effectué ainsi : $\mathbf{M}^L \leftarrow \mathbf{T}_{M2L} \,.\, \mathbf{M}^M + \mathbf{M}^L$. Les colonnes de la matrice locale sont ensuite *recopiées* à la place des développements locaux initiaux. Il aurait été possible d'utiliser une matrice locale nulle et d'ajouter le résulat aux développements locaux initiaux, mais sur les processeurs modernes, tels le PowerPC ou l'Itanium (IA-64), l'instruc-

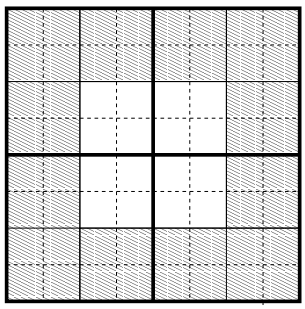


FIG. 5.4 – Les rayures correspondent aux cellules avec liste d'interaction incomplète, pour un quadtree de hauteur 3 et un critère de bonne séparation $ws = 1$.

tion « fused multiply-add » (FMA, ou FMADD) permet d'effectuer le calcul de $c = a.b + c$ avec le même nombre de cycles que pour $c = a.b$. Nous préférons donc deux copies plutôt qu'une mise à zéro et une addition.

Pour un critère de bonne séparation donné ws (voir section 2.3.3.3), les vecteurs *M2L* de la liste d'interaction d'une cellule donnée sont déterminés en fonction du *type de fils* de la cellule : le *type de fils* décrit ici la position du centre de la cellule par rapport au centre de son père. En $3D$, les huit *types de fils* différents sont : BDL, BUL, BDR, BUR, FDL, FUL, FDR, FUR, avec F, B, U, D, L, R correspondant respectivement à « Front » (avant), « Back » (arrière), « Up » (haut), « Down » (bas), « Left » (gauche) et « Right » (droite). Deux *types de fils* différents partagent des vecteurs *M2L*. Certains vecteurs *M2L* sont aussi partagés par tous les *types de fils* ; par exemple, avec $ws = 1$, sur les 189 vecteurs *M2L* de la liste d'interaction $5^3 - 3^3 = 98$ sont communs à tous les *types de fils*.

Lorsqu'on regroupe les opérations *M2L*, nous traitons les vecteurs *M2L* l'un après l'autre. Deux boucles sont donc nécessaires : une pour le *type de fils* et l'autre pour le vecteur *M2L*. On note $\mathcal{V}_{M2L}$ l'ensemble de tous les vecteurs *M2L* possibles, $\mathcal{T}_C$ l'ensemble de tous les *types de fils* et $\mathbf{T}_{M2L}(v)$ la matrice de transfer *M2L* correspondant au vecteur *M2L* v. La première possibilité est alors :

Algorithme 5 Liste d'interaction incomplète : première boucle sur les vecteurs *M2L*

1: **Pour tout** $v \in \mathcal{V}_{M2L}$ **faire**
2: **Pour tout** $t \in \mathcal{T}_C$ correspondant à v **faire**
3: Copier les vecteurs locaux dans $\mathbf{M}^L$;
4: Copier les vecteurs multipôles correspondants dans $\mathbf{M}^M$;
5: **Fin pour**
6: Effectuer $\mathbf{M}^L = \mathbf{T}_{M2L}.\mathbf{M}^M$;
7: Recopier tous les vecteurs locaux copiés dans la boucle sur *t* ;
8: **Fin pour**

Et la deuxième possibilité :

Le nombre de *recopies* est le même dans les deux cas, mais dans le premier algorithme le nombre de vecteurs locaux traités par produit matrice-matrice est plus grand que dans le second. En effet dans le premier algorithme, pour chaque vecteur *M2L* il n'y a qu'un

Algorithme 6 Liste d'interaction incomplète : première boucle sur les *types de fils*

1: **Pour tout** $t \in \mathcal{T}_C$ **faire**
2: **Pour tout** $v \in \mathcal{V}_{M2L}$ correspondant à t **faire**
3: Copier les vecteurs locaux dans $\mathbf{M}^L$;
4: Copier les vecteurs multipôles correspondants dans $\mathbf{M}^M$;
5: Effectuer $\mathbf{M}^L = \mathbf{T}_{M2L}.\mathbf{M}^M$;
6: Recopier les vecteurs locaux correspondants ;
7: **Fin pour**
8: **Fin pour**

seul appel BLAS de niveau 3 qui calcule tous les vecteurs locaux concernés par ce vecteur *M2L*, peu importe le *type de fils* de la cellule à laquelle ils appartiennent. Alors que dans le second algorithme, pour chaque vecteur *M2L* il y a 8 appels BLAS de niveau 3 : un pour chaque *type de fils*. Comme l'accélération obtenue avec les BLAS de niveau 3 augmente avec le nombre de vecteurs locaux (c'est-à-dire de colonnes) traités, nous préférons utiliser le premier algorithme. De plus le premier algorithme permet de réutiliser la même zone mémoire pour stocker les différentes matrices de transfert *M2L* successives : il n'est pas nécessaire de conserver en même temps en mémoire toutes les matrices de transfert *M2L*.

Comme nous le verrons par la suite (voir section 8.3.1), ce même algorithme peut aussi être utilisé dans la version adaptative de la FMM.

5.3.2 Calcul des développements locaux des cellules avec liste d'interaction complète

Toutes les cellules avec liste d'interaction complète ont une liste d'interaction de même taille, et toutes les cellules d'un même *type de fils* partagent exactement les mêmes vecteurs *M2L*. Cette régularité permet d'utiliser des algorithmes plus efficaces. Nous étudions donc deux méthodes différentes pour concaténer nos vecteurs multipôles et locaux :

- une méthode simple qui utilise à chaque fois des *recopies* afin de traiter ensemble le nombre maximum de paires de développements multipôles et locaux ;
- une autre méthode qui évite le surcoût des *recopies* grâce à un *stockage des données* spécial pour nos vecteurs multipôles et locaux. Ceci implique un « réarrangement » du stockage de ces vecteurs en mémoire qui est réalisé avant la phase de descente de la FMM. Les vecteurs multipôles et locaux peuvent être stockés ligne par ligne, tranche par tranche ou même pour tout le niveau à la fois, mais ceci implique (sauf dans le cas du *stockage par ligne*) le calcul inutile de certains développements locaux appartenant à ce que nous appelerons des *boîtes blanches*.

Dans l'octree, les niveaux inférieurs ou égaux à 2 ne comportent pas de cellules avec liste d'interaction complète.

5.3.2.1 Schéma avec recopies.

Comme dans [74], nous considérons un vecteur *M2L* après l'autre, et pour chacun nous copions toutes les paires correspondantes de vecteurs multipôles et locaux dans les matrices multipôle et locale, puis nous effectuons un seul produit matrice-matrice. Il y a trois possibilités présentées aux algorithmes 7, 8 et 9.

Algorithme 7 Schéma avec *recopies* : première boucle sur les *types de fils*

Pour tout $t \in \mathcal{T}_C$ **faire**
  Copier tous les vecteurs locaux dans $\mathbf{M}^L$;
  Pour tout $v \in \mathcal{V}_{M2L}$ correspondant à t **faire**
    Copier les vecteurs multipôles correspondants dans $\mathbf{M}^M$;
    Effectuer $\mathbf{M}^L = \mathbf{T}_{M2L}.\mathbf{M}^M$;
  Fin pour
  Recopier tous les vecteurs locaux ;
Fin pour

Algorithme 8 Schéma avec *recopies* : première boucle sur les vecteurs *M2L*

Pour tout $v \in \mathcal{V}_{M2L}$ **faire**
  Pour tout $t \in \mathcal{T}_C$ correspondant à v **faire**
    Copier tous les vecteurs locaux dans $\mathbf{M}^L$;
    Copier les vecteurs multipôles correspondants dans $\mathbf{M}^M$;
    Effectuer $\mathbf{M}^L = \mathbf{T}_{M2L}.\mathbf{M}^M$;
    Recopier tous les vecteurs locaux ;
  Fin pour
Fin pour

Algorithme 9 Schéma avec *recopies* : avec les vecteurs *M2L* communs

Copier tous les vecteurs locaux dans $\mathbf{M}^L$;
Pour tout $v \in \mathcal{V}_{M2L}$ commun à tous les *types de fils* **faire**
  Copier les vecteurs multipôles correspondants dans $\mathbf{M}^M$;
  Effectuer $\mathbf{M}^L = \mathbf{T}_{M2L}.\mathbf{M}^M$;
Fin pour
Recopier tous les vecteurs locaux ;
Pour tout $t \in \mathcal{T}_C$ **faire**
  Copier tous les vecteurs locaux dans $\mathbf{M}^L$;
  Pour tout $v \in \mathcal{V}_{M2L}$ correspondant à t (et non commun à tous les *types de fils*) **faire**
    Copier les vecteurs multipôles correspondants dans $\mathbf{M}^M$;
    Effectuer $\mathbf{M}^L = \mathbf{T}_{M2L}.\mathbf{M}^M$;
  Fin pour
  Recopier tous les vecteurs locaux ;
Fin pour

Pour l'algorithme 9 nous rappellons que parmi les 189 vecteurs *M2L* de la liste d'interaction, 98 sont communs à tous les *types de fils*. Les 91 restants sont utilisés dans la seconde boucle (sur t) de cet algorithme.

L'algorithme 8, comparé à l'algorithme 7, a l'inconvénient d'effectuer plus de copies pour les vecteurs locaux, tout en ayant le même nombre de vecteurs traités à chaque produit matrice-matrice. L'algorithme 9, comparé à l'algorithme 7, a l'avantage de traiter 8 fois plus de vecteurs locaux, pour 98 vecteurs *M2L* sur 189. Mais il a l'inconvénient d'effectuer deux fois plus de copies pour les vecteurs locaux que l'algorithme 7. De plus, nous pensons que le nombre de vecteurs traités dans l'algorithme 7 est déjà suffisant pour obtenir assez d'effi-

cacité dans les BLAS de niveau 3. En effet pour un niveau l de l'octree avec des conditions aux limites libres, il y a dans chaque dimension $2^{l-1} - 2ws$ cellules du même *type de fils* et avec liste d'interaction complète. $(2^{l-1} - 2ws)^3$ vecteurs sont donc traités ensemble avec l'algorithme 7, ce qui donne par exemple 2744 vecteurs pour le niveau 5, et est par conséquent assez grand pour la plupart des valeurs de P. C'est pourquoi nous choisissons l'algorithme 7.

Cependant, cet algorithme 7 impose de conserver en mémoire toutes les matrices de transfert *M2L* durant le calcul *M2L* d'un niveau de l'octree.

5.3.2.2 Stockage des données par ligne.

La première possibilité afin d'éviter les *recopies* est de choisir une dimension donnée et de stocker consécutivement en mémoire tous les vecteurs locaux qui appartiennent aux cellules d'un même *type de fils* le long d'une ligne dans cette dimension (voir figure 5.5). Ceci est réalisé pour chaque ligne du niveau de l'octree, et le même stockage est aussi mis en place pour les vecteurs multipôles. Il faut cependant noter que pour les vecteurs locaux, seules les cellules avec liste d'interaction complète ont leurs développements locaux réarrangés en ligne, alors que pour les développements multipôles le réarrangement est effectué pour toutes les cellules du niveau : ceci est dû au fait que le calcul *M2L* du développement local d'une cellule avec liste d'interaction complète peut avoir besoin de développements multipôles de cellules avec liste d'interaction incomplète.

Avec un tel stockage des données, nous pouvons appeler une routine BLAS de niveau 3 avec comme point de départ le premier vecteur local de la première cellule de chaque ligne, ainsi qu'avec la matrice de transfert *M2L* et le vecteur multipôle correspondants. Comme le vecteur local de la prochaine cellule de même *type de fils* dans la ligne est consécutif en mémoire et comme il en est de même pour les vecteurs multipôles, nous pouvons ici utiliser des BLAS de niveau 3 avec des matrices locale et multipôle stockées par colonnes.

Avec des conditions aux limites libres, au niveau l de l'octree, il y a 8^{l-1} cellules d'un *type de fils* donné. Pour chaque *type de fils*, les vecteurs multipôles sont réarrangés en $(2^{l-1})^2$ lignes de taille 2^{l-1}. Il y a $(2^{l-1} - 2.ws)^3$ cellules d'un *type de fils* donné avec liste d'interaction complète : les développements locaux sont donc réarrangés en $(2^{l-1} - 2.ws)^2$ lignes de taille $2^{l-1} - 2.ws$. Le nombre de développements locaux traités dans chaque produit matrice-matrice est par conséquent de $2^{l-1} - 2.ws$.

Remarque 5.3. *Afin d'implémenter le stockage par* ligne*, nous devons être capable d'identifier (et d'accéder au contenu de) chaque cellule en fonction des coordonnées de son centre. Ceci est rendu possible dans notre implémentation grâce à l'ordre de Morton (voir section 2.3.1), et ceci est aussi impératif pour les stockages par* tranche *et par* niveau*.*

5.3.2.3 Stockage des données par tranche.

Le principal inconvénient du stockage des données par *ligne* est que le nombre de vecteurs locaux calculés à chaque fois peut être assez faible pour les premiers niveaux de l'octree : au niveau $l = 4$, ce nombre vaut 6 ce qui n'est pas suffisant pour obtenir assez d'efficacité dans les BLAS de niveau 3.

C'est pourquoi nous proposons ici de stocker les lignes consécutivement en mémoire afin de former une « tranche » (« slice »), et de traiter ainsi plusieurs lignes avec un appel BLAS

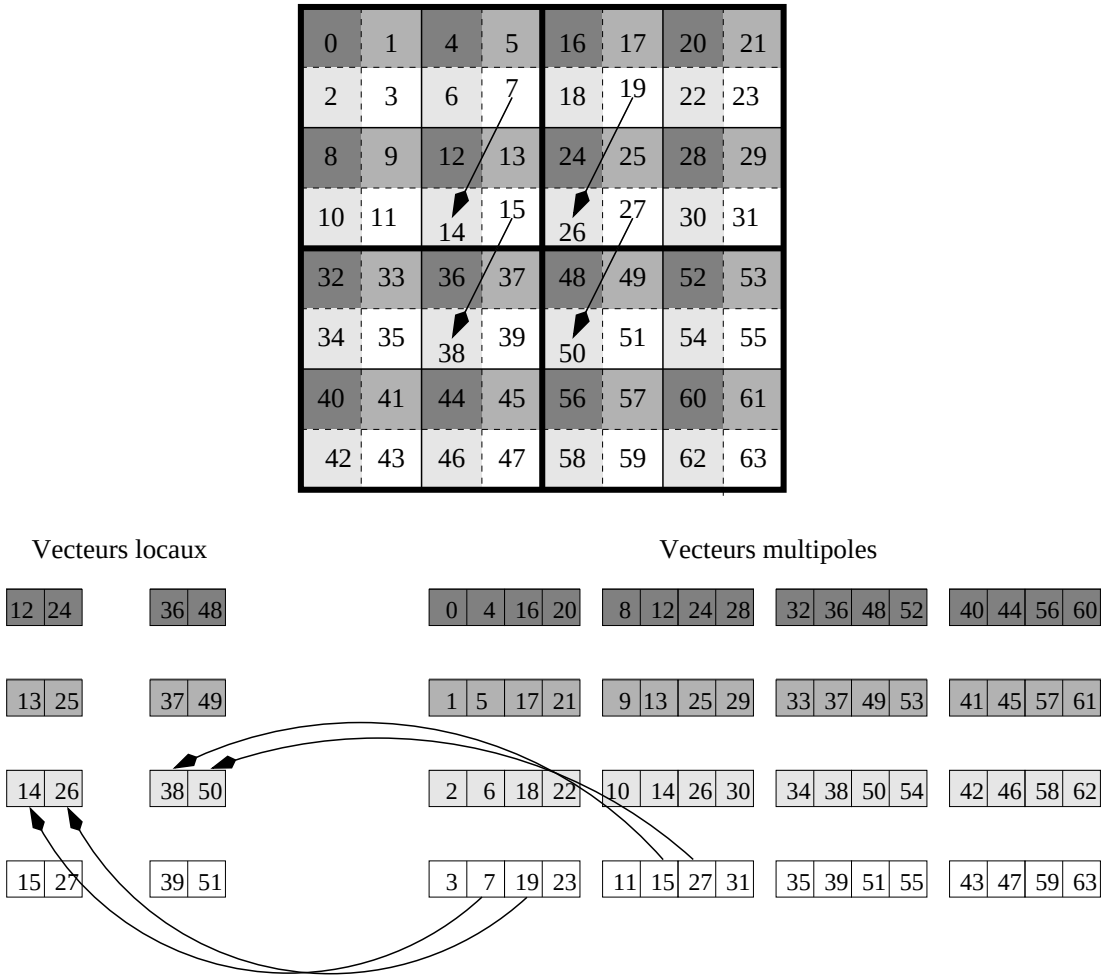


FIG. 5.5 – Stockage des données par *ligne* ($2D$, $ws = 1$, niveau 3). Les cellules sont indexées suivant l'ordre de Morton. Chacun des 4 *types de fils* a un niveau de gris différent. Les développements des cellules d'un même *type de fils* le long d'une ligne donnée sont concaténés : les quatre opérations *M2L*, dont les vecteurs *M2L* sont représentés sur le quadtree, peuvent alors être directement calculées avec des produits matrice-matrice (BLAS de niveau 3) sans *recopies*.

de niveau 3. Pour avoir une correspondance correcte entre les tranches des développements locaux et les tranches des développements multipôles, nous devons cependant insérer des *boîtes blanches* (*blank boxes* ou *blank cells*) entre les lignes des développements locaux : ces *boîtes blanches* correspondent à des cellules n'appartenant pas à l'octree dont les vecteurs locaux sont calculés inutilement. Elles servent en fait à combler l'abscence des cellules avec liste d'interaction incomplète qui sont traitées séparément. Avec $ws = 1$, il y a une *boîte blanche* au début et une à la fin de chaque ligne (voir figure 5.6). Nous pouvons néanmoins sauter la première *boîte blanche* de la tranche (c'est-à-dire, la première *boîte blanche* de la première ligne), ainsi que la dernière.

Comme pour le stockage par *ligne*, le réarrangement des vecteur locaux ne concerne que

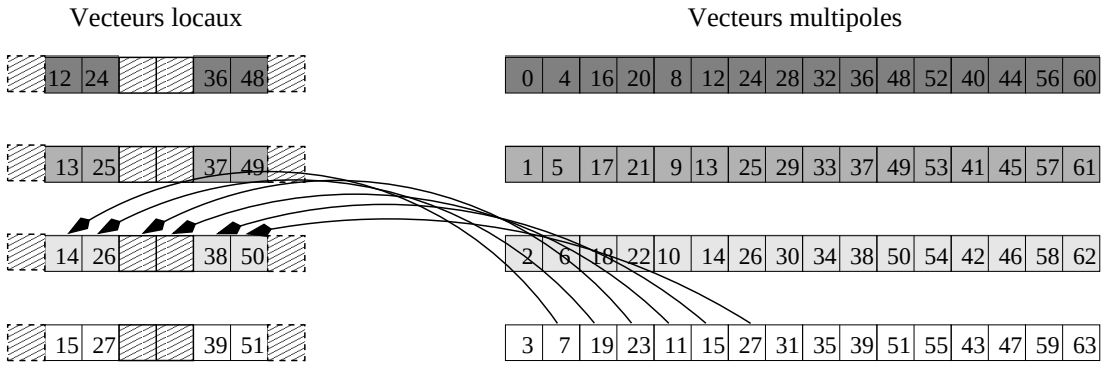


FIG. 5.6 – Stockage des données par *tranche* ($2D$, $ws = 1$, niveau 3) : voir aussi figure 5.5. Chaque *type de fils* a un niveau de gris différent. Les lignes du mode de stockage des données par *ligne* sont concaténées, et des *boîtes blanches* (marquées par des rayures) sont insérées entre chaque ligne : le produit matrice-matrice comprend désormais des matrices multipôle et locale avec 6 colonnes, contre 2 avec le stockage par *ligne* (voir figure 5.5).

les cellules avec liste d'interaction complète alors que le réarrangement des vecteurs multipôles concerne toutes les cellules. De plus, il n'y a des *boîtes blanches* qu'entre les lignes des vecteurs locaux concaténés. Il n'y a pas besoin de telles *boîtes blanches* pour les lignes des vecteurs multipôles : il suffit en effet de concaténer tous les développements multipôles d'une tranche complète de l'octree pour obtenir une correspondance correcte entre cette tranche et celle formée par les développements locaux et les *boîtes blanches*.

Avec des conditions aux limites libres, au niveau l de l'octree, et pour chaque *type de fils*, les vecteurs multipôles sont réarrangés en 2^{l-1} tranches de taille $(2^{l-1})^2$. Les vecteurs locaux sont réarrangés en $2^{l-1}-2.ws$ tranches de taille $(2^{l-1}-2.ws)^2+2.(2^{l-1}-2.ws)-2$ (à savoir le nombre de cellules d'un *type de fils* donné avec liste d'interaction complète dans 1 tranche, plus 1 *boîte blanche* à la fin et au début de chaque ligne, moins la première et la dernière *boîte blanche* de la tranche), ce qui donne le nombre de développements locaux traités à chaque produit matrice-matrice.

Si le stockage des données par *tranche* offre une meilleure efficacité dans les BLAS de niveau 3, il présente aussi clairement l'inconvénient d'effectuer un calcul supplémentaire et inutile au niveau des *boîtes blanches*.

5.3.2.4 Stockage des données par niveau.

De la même façon que nous avons concaténé des lignes pour former des tranches afin d'obtenir une meilleure efficacité dans les BLAS de niveau 3, nous pouvons concaténer des tranches dans le but d'avoir tout le niveau stocké dans un seul bloc mémoire et donc traité par un seul appel BLAS de niveau 3. C'est le stockage des données par *niveau*.

En plus des *boîtes blanches* au début et à la fin de chaque ligne, nous devons ajouter pour le stockage par *niveau* une ligne de *boîtes blanches* entre chaque tranche de vecteurs locaux (inutile pour les vecteurs multipôles). La première ligne de *boîtes blanches* du niveau, ainsi que la dernière, peuvent être omises.

L'efficacité des BLAS de niveau 3 est plus grande, surtout pour les premiers niveaux de l'octree, mais le nombre de *boîtes blanches* inutilement calculées est aussi clairement plus

important. Ce mode de stockage n'a pas été implémenté (voir la section 5.4.2 pour une justification expérimentale).

5.3.3 Implémentation avec routines BLAS de niveau 3

Comme dans la section 5.2, le produit matrice-matrice pour un noyau *M2L* de hauteur double peut être directement implémenté avec un unique appel BLAS, alors que pour un noyau *M2L* de hauteur simple la matrice de transfert creuse doit être décomposée en blocs. La discussion de la section 5.2 s'applique aussi ici en replaçant la routine $ZGEMV$ par la routine $ZGEMM$.

Cependant, avec un noyau *M2L* de hauteur simple, nous devons faire face à une contrainte supplémentaire lors du découpage du produit matrice-matrice en plusieurs appels BLAS de niveau 3. Nous avons en effet supposé au début de cette section (voir page 107) que plus le nombre de colonnes dans les matrices multipôle et locale était grand, meilleure était l'efficacité obtenue dans les appels BLAS de niveau 3. Si le produit matrice-matrice est effectué par l'intermédiaire d'un unique appel BLAS de niveau 3, comme dans le cas d'un noyau *M2L* de hauteur double (voir section 5.2.1), ceci est essentiellement vrai : des tests ont confirmé que même si de meilleures performances peuvent être obtenues en découpant les matrices multipôle et locale (comme ci-dessous), le gain en performance est relativement faible car l'unique appel BLAS de niveau 3 effectue son propre découpage.

Mais avec un noyau de hauteur simple, nous avons plusieurs appels BLAS de niveau 3 par produit matrice-matrice. Si toutes les colonnes de la matrice multipôle ou de la matrice locale ne peuvent être stockées dans un des niveaux de la hiérarchie mémoire (niveau 1, 2 ou même 3, du cache), ou requièrent plus de pages mémoire que le TLB (Translation Lookaside Buffer) de la table des pages ne peut adresser, toute la matrice devrait être rechargée à chaque appel BLAS. Et dans le schéma original avec *recopies* ou dans le stockage par *tranche* le nombre de colonnes augmente fortement avec la hauteur de l'octree. C'est pourquoi les matrices multipôle et locale devront aussi être traitées par blocs : nous les découpons en sous-matrices avec un nombre constant de colonnes, désigné par $NbExp_{max}$, et le même nombre de lignes que la matrice originale. Comme représenté en figure 5.7, la $i^{\text{ème}}$ sous-matrice de la matrice locale est calculée par un produit matrice-matrice entre la matrice de transfert *M2L* et la $i^{\text{ème}}$ sous-matrice de la matrice multipôle. Un produit matrice-matrice avec $NbExp$ développements (*expansions*) est donc calculé par $NbExp/NbExp_{max}$ produits matrice-matrice avec $NbExp_{max}$ développements, plus un éventuel produit matrice-matrice supplémentaire pour les dernières colonnes correspondant au reste de la division entière. Cette décomposition du produit matrice-matrice est appelée par la suite « décomposition $NbExp_{max}$ ».

Les valeurs optimales pour $NbExp_{max}$ seront déterminées expérimentalement (voir section 5.4.1).

Remarque 5.4. *On pourrait objecter que le choix de parcourir une fois les développements locaux et plusieurs fois les développements multipôles, comme expliqué page 103 pour les produits matrice-vecteur, combiné à un mode de stockage par colonnes de nos matrices, conduit ici dans le cas de produits matrice-matrice à un problème de dimension principale (voir section 5.2.2 pour une définition de cette notion) pour les matrices multipôle et locale : nous accédons à ces matrices principalement suivant leurs lignes alors qu'elles sont stockées par colonnes. Il est certainement possible de résoudre ce problème en considérant le produit matriciel transposé, avec une dimension principale*

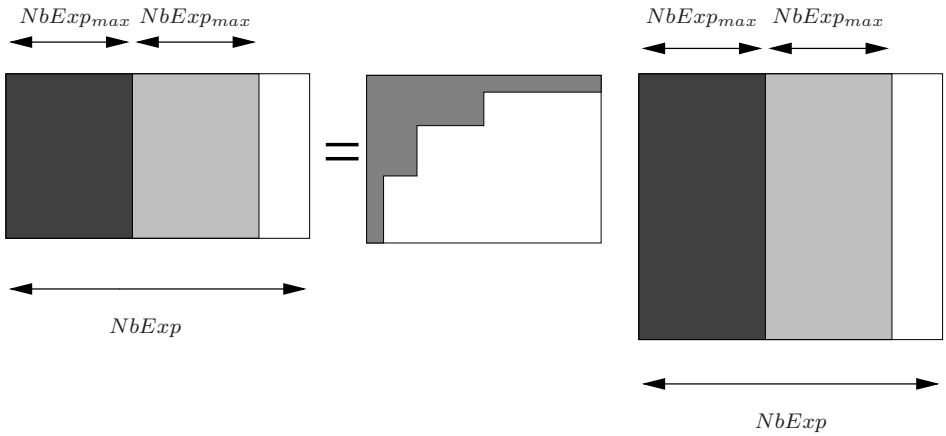


FIG. 5.7 – « Décomposition $NbExp_{max}$ ». Ce produit matrice-matrice (avec $NbExp$ colonnes dans les matrices multipôle et locale) est calculé par plusieurs produits matrice-matrice avec au plus $NbExp_{max}$ colonnes dans les matrices multipôle et locale.

égale à $NbExp_{max}$ pour les matrices multipôle et locale, mais ceci n'a pas été mis en place car il empêche de concaténer facilement les développements en colonnes de ces matrices. De plus ce problème de dimension principale ne devrait pas avoir beaucoup d'impact sur l'efficacité de nos appels BLAS (notamment grâce à la décomposition $NbExp_{max}$).

Remarque 5.5 (Appels BLAS pour les opérateurs *M2M* et *L2L*)**.** *Comme présenté dans [74], les opérateurs* M2M *et* L2L *peuvent être facilement calculés par des appels BLAS de niveau* 3 *: en écrivant* M2M *et* L2L *comme des produits matrice-vecteur entre le développement d'une cellule et le développement de son père, toutes les cellules d'un même* type de fils *partagent les mêmes fonctions de transfert* M2M *et* L2L*. Plusieurs développements peuvent ainsi être calculés d'un coup avec des appels BLAS de niveau* 3 *en les regroupant soit avec des* recopies*, soit avec un stockage en mémoire approprié qui évite ces* recopies *(le stockage par* ligne *ainsi que les autres modes de stockage des données utilisé pour l'opérateur* M2L *ne peuvent pas être utilisés ici, car le stockage ne peut pas être le même au niveau du père et au niveau des fils).*

5.4 Tests et comparaisons

Des tests de performance ont été réalisés afin de valider l'implémentation avec BLAS et trouver les paramètres qui offrent le calcul *M2L* le plus rapide. Ces tests ont été effectués sur un seul processeur : soit un processeur IBM Power3-II WH2+ (375 MHz, 1,5 GFLOPS, cache L1 cache de taille 64 Ko, cache L2 de taille 4 Mo, et 2 Go de mémoire), soit un processeur IBM Power4+ (1454 MHz, $\approx$ 6 GFLOPS, cache L1 de taille 32x2 Ko, cache L2 de taille 1,41 Mo, cache L3 de taille 32 Mo, et 8 Go de mémoire, nécessitant donc une compilation en mode 64 bits). Ces machines sont localisées au LaBRI[3]. Nous avons aussi utilisé des processeurs Power3 NH2 (cache L1 de taille 128 Ko, cache L2 de taille 8 Mo, et 16 Go de mémoire) localisés au CINES[4]. La bibliothèque BLAS utilisée est la bibliothèque IBM ESSL [4].

Le nombre de particules utilisé pour les simulations n'est généralement pas précisé car

[3]Laboratoire Bordelais de Recherche en Informatique, Talence, FRANCE.
[4]Centre Informatique National de l'Enseignement Supérieur, Montpellier, FRANCE.

les temps de calcul mesurés ici ne dépendent que de la hauteur de l'octree uniforme utilisé et de P, mais cette hauteur implique bien sûr un intervalle de valeurs adaptées pour le nombre de particules (les valeurs comprises dans cet intervalle étant celles qui équilibrent, pour cette hauteur d'octree, le coût du calcul du champ proche avec celui du champ lointain).

Tout d'abord nous justifions la préférence donnée aux routines BLAS de niveau 3 par rapport à celles de niveau 2 avec la figure 5.8 : pour un octree de hauteur 5 et un noyau *M2L* de hauteur double, l'utilisation du schéma avec *recopies* (BLAS de niveau 3) améliore clairement les performances (gain d'environ 50% pour $P \geq 7$). Ceci est aussi valable dans le cas d'un noyau *M2L* de hauteur simple et avec d'autres hauteurs d'octree. Pour les faibles valeurs de P, aucun gain significatif n'est cependant obtenu mais ceci sera amélioré grâce aux stockages par *ligne* et par *tranche*, comme présenté en section 5.4.2.

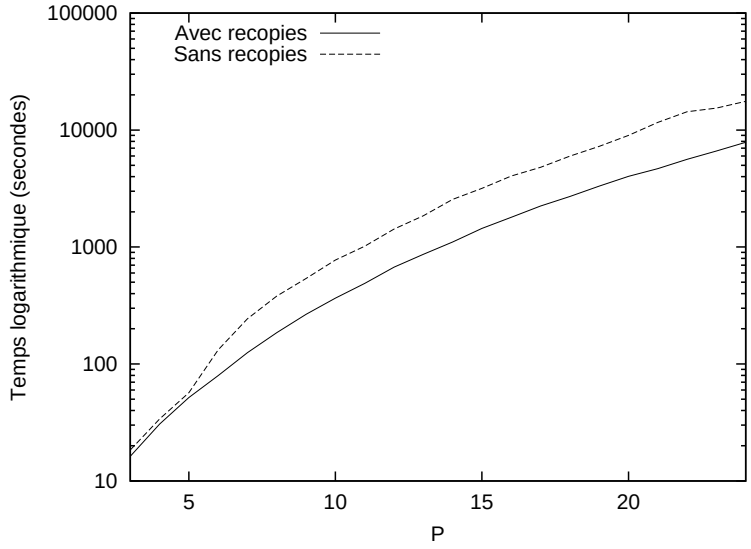


FIG. 5.8 – Temps logarithmique CPU pour la phase de descente pour un octree de hauteur 5, avec la méthode *BLAS_plein* (noyau de hauteur double), avec et sans *recopies*.

Nous allons maintenant déterminer la meilleure décomposition pour la matrice de transfert *M2L* creuse (cas d'un noyau *M2L* de hauteur simple), ainsi que les valeurs optimales pour $NbExp_{max}$, et ensuite nous comparerons les différentes schémas qui permettent d'utiliser les BLAS de niveau 3.

5.4.1 Décomposition pour noyau *M2L* de hauteur simple

Dans cette section, nous allons tout d'abord voir comment nous avons établi, pour chaque architecture (Power3 ou Power4), la meilleure valeur pour $NbExp_{max}$ (voir section 5.3.3) pour les routines *BLAS_bande*, *BLAS_cbande* et *BLAS_bloc* qui implémentent les décompositions correspondantes. Pour la routine *BLAS_cbande*, la meilleure valeur pour dim_{cbande} doit aussi être déterminée, alors que pour la routine *BLAS_bloc* nous devons aussi établir la meilleure valeur pour le seuil, noté s_{bloc}, qui détermine le cas terminal de la récursion (voir section 5.2.2). Avec ces paramètres, nous serons alors capables de comparer les routines *BLAS_bande*, *BLAS_cbande* et *BLAS_bloc* entre elles, et finalement de valider l'intérêt de la « décomposition $NbExp_{max}$ ». Tous ces tests ont été réalisés sur un produit matrice-matrice, correspondant à l'opérateur *M2L*, pour différentes valeurs de P.

	$niveau = 4$	$niveau = 5$	$niveau = 6$	$niveau = 7$
recopies	216	2744	27000	238328
stockage par *ligne*	6	14	30	62
stockage par *tranche*	46	222	958	3966

TAB. 5.1 – Valeurs de $NbExp$ pour nos différents schémas (avec des conditions aux limites libres, et $ws = 1$, voir section 5.3.2).

Premièrement nous devons déterminer les valeurs optimales pour $NbExp_{max}$. Celles-ci dépendent bien sûr de la machine utilisée, mais aussi de P puisque P fixe le nombre de lignes dans les matrices multipôle et locale. En fonction du schéma de BLAS niveau 3 utilisé (avec *recopies*, avec stockage par *ligne*, ou avec stockage par *tranche*) et du niveau de l'octree, le nombre $NbExp$ de développements multipôles et locaux concaténés varie sensiblement. Pour les tests de performance avec différentes valeurs de $NbExp_{max}$, nous n'avons considéré que les valeurs de $NbExp$ utilisées pour les cellules avec liste d'interaction complète : les produits matrice-matrice pour les cellules avec liste d'interaction incomplète ont des valeurs très disparates pour $NbExp$, mais ils sont surpassés en nombre par les produits matrice-matrice pour les cellules avec liste d'interaction complète. Le tableau 5.1 montre les valeurs de $NbExp$ pour nos différents schémas mis en place pour les cellules avec liste d'interaction complète.

En pratique, les tests pour une valeur optimale de $NbExp_{max}$ seront réalisés, en fonction de P, pour une unique valeur « assez grande » de $NbExp$, notée $NbExp_{ref}$. En effet, comme confirmé par des tests plus complets, ces valeurs optimales pour $NbExp_{ref}$ seront aussi parmi les valeurs optimales pour $NbExp > NbExp_{ref}$ (correspondant à des hauteurs d'octree plus grandes), alors que pour $NbExp < NbExp_{ref}$ la décomposition $NbExp_{max}$ ne sera pas nécessaire (à moins que P soit très grand, et les meilleurs valeurs de $NbExp_{max}$ trouvées pour $NbExp_{ref}$ seront alors aussi optimales). Pour une architecture Power3, la recherche peut donc être effectuée pour $NbExp_{ref} = 958$ ($niveau = 6$ et stockage par *tranche*) alors que pour une architecture Power4, qui dispose d'un cache supplémentaire de taille importante (cache L3), des valeurs optimales pour $NbExp_{max}$ plus grandes que 958 peuvent être trouvées, ce qui impose un plus grand $NbExp_{ref}$, à savoir 2744 ($niveau = 5$ et schéma avec *recopies*). En pratique, quand P varie de 3 à 30, les valeurs optimales trouvées pour $NbExp_{max}$ varient généralement de plusieurs centaines pour les petites valeurs de P, jusqu'à quelques dizaines pour les grandes valeurs de P : voir par exemple le tableau en figure 5.9.

Pour les routines *BLAS_cbande* et *BLAS_bloc*, cette recherche est couplée avec celle des meilleures valeurs pour respectivement dim_{cbande} et s_{bloc} : tous les couples possibles sont testés afin de déterminer le meilleur. Toutefois des tests (non présentés ici) ont montré que la valeur optimale $NbExp_{max}$ est plutôt indépendante de dim_{cbande} et de s_{bloc} : lorsqu'on découpe les matrices multipôle et locale selon $NbExp_{max}$, les tailles des sous-matrices dépendent en effet seulement de P (pour le nombre de lignes) et $NbExp_{max}$ (pour le nombre de colonnes), voir figure 5.7. Ceci sous-tend l'idée d'une taille (ou aire), de nos sous-matrices, optimale pour l'efficacité des BLAS de niveau 3 et vraisemblablement liée à la taille des caches mémoire.

Pour dim_{cbande}, d'autres tests ont aussi montré qu'une ou deux (proches) valeurs sont optimales, et qu'elles sont indépendantes de P. Une fois que la hauteur optimale d'une *bande*

est déterminée, elle correspond en effet à la taille optimale des blocs carrés utilisés par la routine $ZGEMM$ pour découper cette *bande* rectangulaire, et ceci reste indépendant de la taille globale de la matrice de transfert *M2L*.

Pour les premières valeurs de P (à savoir environ $P \leq 7$ dans nos tests) les meilleures valeurs pour s_{bloc} sont égales à P ce qui signifie qu'il vaut mieux ne pas utiliser de décomposition récursive : nous sommes alors ramenés à une décomposition de type *BLAS_bande*. Mais lorsque P est plus grand, il est plus efficace d'utiliser la décomposition récursive et des calculs plus rapides sont alors possibles avec des petites valeurs s_{bloc} (entre 2 et 5 dans nos tests) : voir le tableau de la figure 5.9.

Maintenant que les valeurs optimales de ces paramètres ont été trouvées pour chaque valeur de P, nous pouvons comparer les routines *BLAS_bande*, *BLAS_cbande* et *BLAS_bloc*. Il apparaît que *BLAS_cbande* est plus rapide que *BLAS_bande* seulement pour de grandes valeurs de P ($P \geq 14$ sur l'IBM Power3) : les zéros ajoutés dans la décomposition *BLAS_cbande* sont trop coûteux lorsque la matrice de transfert *M2L* n'est pas très grande. Pour la routine *BLAS_bloc*, les valeurs optimales trouvées pour s_{bloc} rendent son calcul aussi rapide que la routine *BLAS_bande* pour les petites valeurs de P, et plus rapide pour les plus grandes valeurs, comme expliqué précédemment. Et pour ces plus grandes valeurs de P, la routine *BLAS_bloc* est plus rapide que la routine *BLAS_cbande*.

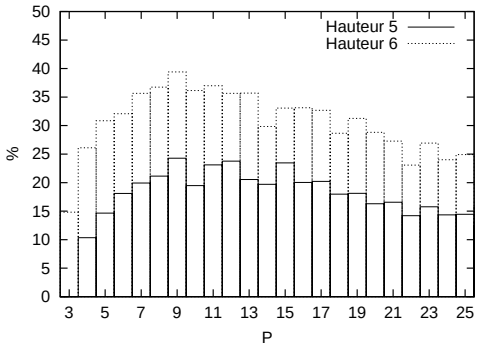
En conclusion nous choisissons donc la routine *BLAS_bloc* pour le calcul par BLAS de l'opérateur *M2L* avec un noyau de hauteur simple. Cependant les gains et les pertes mentionnés ci-dessus sont tous inférieurs à 5 % du temps de calcul de la routine *BLAS_bande* : ceci ne justifie pas vraiment a posteriori l'implémentation, plus complexe, des routines *BLAS_cbande* et *BLAS_bloc*.

Avant de comparer les différents schémas utilisés avec les BLAS de niveau 3 dans la section suivante, la pertinence de la décomposition $NbExp_{max}$ est montrée en figure 5.9 pour le schéma avec *recopies* : pour chaque valeur de P, nous montrons le gain sur les temps CPU de la phase de descente dû à la décomposition $NbExp_{max}$. Avec des hauteurs d'octree croissantes, le nombre de développements $NbExp$ traités avec un seul produit matrice-matrice augmente, et le gain apporté par la décomposition $NbExp_{max}$ augmente donc aussi. Avec les stockages par *ligne* et par *tranche*, la décomposition $NbExp_{max}$ sera pareillement efficace lorsque la valeur de $NbExp$ excédera la valeur optimale de $NbExp_{max}$.

5.4.2 Recopies et stockage des données spéciaux

Afin de comparer le schéma avec *recopies* et les stockages par *ligne* et par *tranche*, les temps CPU ont été mesurés sur la phase de descente complète de la FMM. Deux effets conjoints apparaissent.

– Comme indiqué dans [74], lorsque P augmente, le rapport entre le coût des copies de vecteurs et le coût du produit matrice-matrice décroît : les copies des développements multipôles et locaux sont en effet effectués en $\mathcal{O}(NbExp \times P^2)$ alors que le produit matrice-matrice requiert $\mathcal{O}(NbExp \times P^4)$ opérations. Ce phénomène est plus marqué avec un noyau de hauteur double qu'avec un noyau de hauteur simple car la quantité de calcul pour le produit matriciel est bien plus importante avec une hauteur double, alors que le coût des *recopies* reste le même. Les figures 5.10 et 5.11 montrent le gain des stockages par *ligne* et par *tranche* par rapport au schéma avec *recopies*, en fonction de différentes valeurs de P et pour une hauteur d'octree de 6 : les stockages par *ligne*



Gain en pourcentage.

P	s_{bloc}	$NbExp_{max}$	P	s_{bloc}	$NbExp_{max}$
3	3	921	15	5	102
4	4	936	16	3	96
5	5	463	17	3	84
6	5	403	18	3	84
7	5	307	19	4	102
8	3	260	20	7	126
9	7	210	21	3	120
10	2	168	22	2	120
11	10	156	23	5	132
12	10	126	24	6	120
13	1	138	25	11	120
14	1	120			

Valeurs optimales de s_{bloc} et de $NbExp_{max}$ (Power4).

FIG. 5.9 – Gain en pourcentage pour les temps CPU de la phase de descente complète, entre un calcul avec la décomposition $NbExp_{max}$ et un calcul sans la décomposition $NbExp_{max}$. Les tests ont été réalisés sur un IBM Power4, en utilisant la routine *BLAS_bloc* (avec un s_{bloc} optimal) et le schéma avec *recopies* pour des hauteurs d'octree égales à 5 ($NbExp = 2744$) et 6 ($NbExp = 27000$). Les valeurs correspondantes pour s_{bloc} et $NbExp_{max}$ sont aussi fournies.

et par *tranche* offrent donc de meilleurs gains par rapport aux *recopies* pour les petites valeurs de P, et ces gains sont toujours plus importants dans le cas d'un noyau de hauteur simple.

– Ces gains sont aussi influencés par la hauteur de l'octree : avec une hauteur croissante de l'octree, le nombre de développements traités par un produit matrice-matrice augmente pour les stockages par *ligne* et par *tranche* (voir le tableau 5.1), et la proportion de *boîtes blanches* décroît pour le stockage par *tranche*. Pour la figure 5.11, nous avons seulement $NbExp = 30$ pour le stockage par *ligne* ce qui conduit pour un noyau de hauteur double et pour $P \geq 11$ à de moins bonnes performances qu'avec les *recopies* (qui impliquent $NbExp = 27000$). Pour de petites hauteurs de l'octree telles que 4, l'usage des *recopies* est même légèrement plus rapide pour toutes les valeurs de P puisque ce schéma est le seul à offrir assez de colonnes dans les matrices multipôle et locale.

Le stockage par *tranche* est ici plus efficace que le stockage par *ligne*, mais ceci est principalement dû à la petite valeur de $NbExp$ pour le stockage par *ligne* à la hauteur 6 (c'est-à-dire $NbExp = 30$), alors que pour le stockage par *tranche* $NbExp = 958$ offre une meilleure efficacité dans les BLAS de niveau 3 : en effet, des tests réalisés avec une hauteur de 7 sur un processeur IBM Power3 avec 16Go de mémoire (CINES) montrent que le stockage par *ligne* est un peu plus rapide pour cette hauteur. En fait, le stockage par *tranche* n'offre généralement pas de gain par rapport au stockage par *ligne* et de plus, le stockage par *tranche* nécessite de très longues zones mémoires consécutives qui ne peuvent être allouées pour de trop grandes valeurs de P ou de trop grandes hauteurs d'octree ; l'allocation mémoire pour le stockage par *ligne* ne pose elle pas de problème. Pour ces raisons nous préférons le stockage par *ligne* et comme les valeurs de $NbExp$ offertes par le stockage par *tranche* semblent suffisantes, il est inutile d'implémenter le stockage par *niveau*.

En conclusion les stockages par *ligne* et par *tranche* offrent tous deux des gains importants sur le schéma avec *recopies* pour des petites et moyennes valeurs de P et des hauteurs suf-

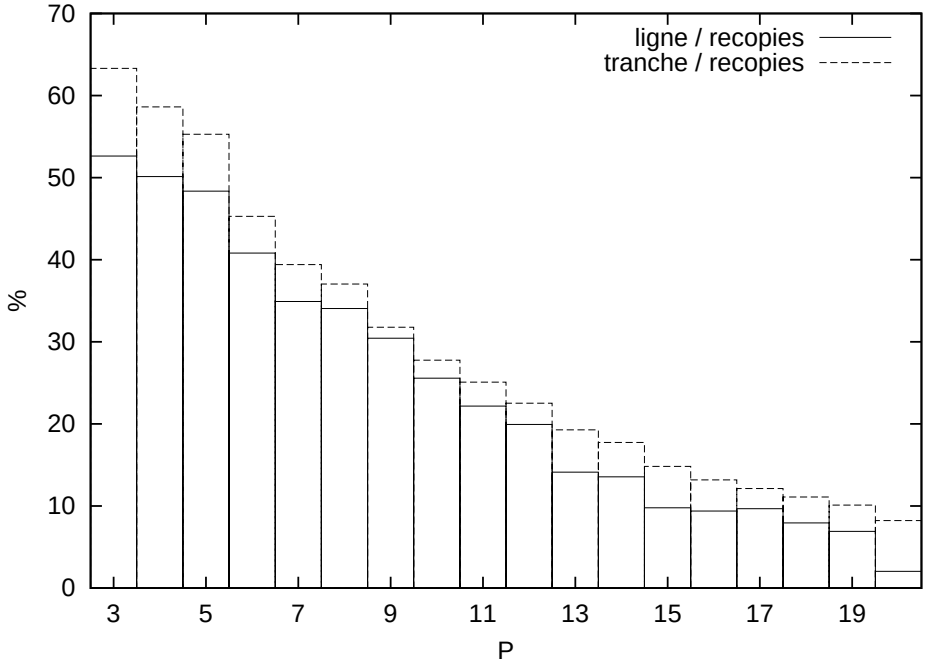


FIG. 5.10 – Gains en pourcentage offerts par les stockages par *ligne* et par *tranche*, par rapport au schéma avec *recopies*, pour les temps CPU de la phase de descente, avec un octree de hauteur 6 et la routine *BLAS_bloc* (noyau de hauteur simple). Tests réalisés sur un processeur IBM Power4.

fisantes. Nous choisirons généralement le stockage des données par *ligne*. Nous conservons néanmoins l'implémentation avec *recopies* (pour les octrees de petite hauteur).

En résumé, les meilleurs implémentations BLAS sont donc les routines *BLAS_bloc* et *BLAS_plein* (suivant la hauteur du noyau *M2L* utilisée), avec un stockage des données par *ligne*.

5.5 Besoins mémoire

Nous présentons ici les besoins mémoire pour le calcul *M2L* par BLAS. Tout d'abord, les développements multipôles doivent être convertis en vecteurs multipôles dont la taille est : $\sum_{j=0}^{P}\sum_{k=-j}^{j} t_c = (P+1)^2 t_c$, où t_c désigne la taille d'un nombre complexe (voir section 2.3.3.6). Ensuite les fonctions de transfert *M2L* sont converties en matrices de transfert *M2L*. Pour un noyau *M2L* de hauteur double, la taille de ces matrices est

$$\mathcal{T}_{matrice} = \frac{(P+1)(P+2)}{2}(P+1)^2 t_c.$$

Cette taille peut devenir assez importante pour de grandes valeurs de P puisqu'elle croît en $\mathcal{O}(P^4)$, contre $\mathcal{O}(P^2)$ pour les fonctions de transfert du calcul *M2L* classique. Pour un noyau de hauteur simple, nous considérons ici les matrices découpées suivant la décomposition *BLAS_bloc*. Leur espace mémoire est cependant le même qu'avec la décomposition *BLAS_bande*, et peut donc être évalué ainsi : les *bandes* étant numérotées du

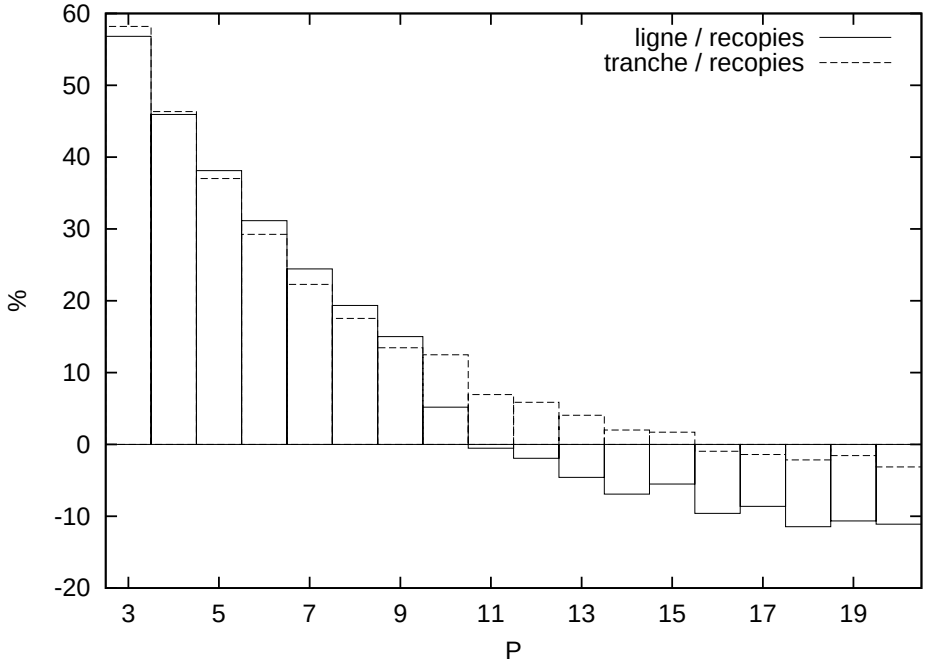


FIG. 5.11 – Gains en pourcentage offerts par les stockages par *ligne* et par *tranche*, par rapport au schéma avec *recopies* pour les temps CPU de la phase de descente avec un octree de hauteur 6 et la routine *BLAS_plein* (noyau de hauteur double). Tests réalisés sur un processeur IBM Power4.

haut vers le bas de la matrice, la *bande* de numéro M, $M \in [\![0, P]\!]$, a $M+1$ lignes et $\sum_{N=0}^{P-M}(2N+1) = (P-M+1)^2$ colonnes, et l'espace mémoire de toutes ces *bandes* est par conséquent :

$$\mathcal{T}_{matrix} = \sum_{M=0}^{P}(M+1)(P-M+1)^2 t_c = \frac{(P+1)(P+2)^2(P+3)}{12} t_c.$$

L'espace mémoire nécessaire à la décomposition *BLAS_cbande* est un peu plus élevé à cause de la présence des zéros supplémentaires dans les *bandes*.

Avec les BLAS de niveau 2, aucun autre espace mémoire n'est nécessaire. Mais avec les BLAS de niveau 3, nous avons besoin de mémoire supplémentaire. Tout d'abord, pour les cellules avec liste d'interaction incomplète, nous avons besoin de deux tampons temporaires : un pour la matrice multipôle et un autre pour la matrice locale. Le nombre de développements copiés dans de tels tampons varie énormément et peut devenir très grand pour des hauteurs de l'octree croissantes : par exemple avec une hauteur de 6, il varie de 1 à 37 952. Cependant le calcul peut être effectué tout aussi efficacement avec de plus petits tampons : si la taille des deux tampons ne permet de traiter que $\mathcal{N}_{Buf}$ développements ensemble dans un seul produit matrice-matrice, et si n paires de développements multipôles et locaux partagent le même vecteur *M2L*, avec $n > \mathcal{N}_{Buf}$, ces n paires sont traitées avec $\frac{n}{\mathcal{N}_{Buf}}$ appels consécutifs, plus un éventuel appel supplémentaire pour les développements restants. A condition que $\mathcal{N}_{Buf}$ soit assez grand pour obtenir une efficacité optimale dans les BLAS de niveau 3, les performances seront les mêmes qu'avec des tampons pouvant conte-

H	3	4	5	6	7
$\mathcal{N}_{\mathcal{B}}(H)$	32	512	3424	17344	77856
$\mathcal{N}_{\mathcal{B}}(H)/\mathcal{N}(H)$	5,5 %	10,9 %	9,1 %	5,8 %	3,2 %

TAB. 5.2 – Nombre de *boîtes blanches* $\mathcal{N}_{\mathcal{B}}(H)$ en fonction de la hauteur H de l'octree. Nous donnons aussi la proportion par rapport au nombre total de cellules $\mathcal{N}(H)$ dans l'octree.

nir les n développements. En pratique, les valeurs de $\mathcal{N}_{Buf}$ sont les mêmes que les valeurs optimales de $NbExp_{max}$ déterminées en section 5.4.1 : elles sont donc choisies en fonction de P et sont de plus les mêmes pour des noyaux *M2L* de hauteurs simple et double.

Pour les cellules avec liste d'interaction complète, nous devons distinguer le schéma avec les *recopies*, le stockage par *ligne* et le stockage par *tranche*. Avec les *recopies* les besoins mémoire sont les mêmes que pour les cellules avec liste d'interaction incomplète : nous avons besoin de deux tampons pour les *recopies* des matrices multipôle et locale. Pour une efficacité des BLAS optimale, nous imposons seulement qu'ils puissent contenir autant de développements que les valeurs optimales de $NbExp_{max}$ (déterminées en section 5.4.1 et fonctions de P) . Nous pouvons de plus considérer que ces tampons peuvent être les mêmes que ceux utilisés pour les cellules avec liste d'interaction incomplète, et le schéma avec *recopies* ne requiert donc pas d'espace mémoire supplémentaire.

Le stockage par *ligne* n'utilise pas de tampon temporaire, mais avec des hauteurs croissantes de l'octree, il requiert de plus longues zones mémoire continues pour chaque ligne. Ceci devient encore plus critique avec le stockage par *tranche* puisque celui-ci requiert des zones mémoire continues encore plus longues pour chaque tranche. De plus les *boîtes blanches* introduites avec le stockage par *tranche* augmentent les besoins mémoire. Concrètement, chaque *boîte blanche* correspond à un vecteur de même taille qu'un vecteur local, et donc qu'un développement local, et leur nombre peut être calculé comme suit (voir aussi la section 5.3.2.3) : au niveau l de l'octree, pour les cellules avec liste d'interaction complète et pour chacun des huit *types de fils*, il y a $2^{l-1} - 2.ws$ tranches avec $2^{l-1} - 2.ws$ lignes dans chaque tranche. Nous avons alors une *boîte blanche* au début et une à la fin de chaque ligne, moins la première et la dernière *boîte blanche* de la tranche, ce qui donne $2.(2^{l-1} - 2.ws) - 2$ « développements blancs » par tranche. Le nombre de tous les développements blancs $\mathcal{N}_{\mathcal{B}}(H)$ pour un octree de hauteur H est par conséquent $\sum_{l=3}^{H} 8(2.(2^{l-1} - 2.ws) - 2).(2^{l-1} - 2.ws)$ (il n'y a pas de cellules avec liste d'interaction complète pour les niveaux $l \leq 2$), et avec $ws = 1$ on obtient :

$$\mathcal{N}_{\mathcal{B}}(H) = \sum_{l=3}^{H} 16(2^{l-1} - 3)(2^{l-1} - 2) = \frac{16}{3}.4^H - 80.2^H + 96.H + \frac{128}{3}.$$

Le tableau 5.2 présente la proportion de *boîtes blanches*, par rapport au nombre total de cellules, en fonction de H. Cette proportion reste modérée, et elle décroît pour les H croissants (à partir de $H \geq 4$).

Pour résumer, le schéma avec *recopies* et le stockage par *ligne* ont les mêmes besoins mémoire, à savoir

$$\begin{aligned} Mem_{Sg}(P,H) = \quad & \mathcal{N}(H)\left((P+1)^2 + \tfrac{(P+1)(P+2)}{2}\right)t_c + \mathcal{N}_{\mathcal{T}}\mathcal{T}_{matrix} \\ & + \mathcal{N}_{Buf}\left((P+1)^2 + \tfrac{(P+1)(P+2)}{2}\right)t_c, \end{aligned} \tag{5.5}$$

et pour le stockage par *tranche*, nous avons

$$\begin{aligned} Mem_{Db}(P,H) = \quad & \mathcal{N}(H)\left((P+1)^2 + \tfrac{(P+1)(P+2)}{2}\right)t_c + \mathcal{N}_{\mathcal{T}}\mathcal{T}_{matrix} \\ & + \mathcal{N}_{\mathcal{B}}(H)\tfrac{(P+1)(P+2)}{2}t_c + \mathcal{N}_{Buf}\left((P+1)^2 + \tfrac{(P+1)(P+2)}{2}\right)t_c. \end{aligned} \tag{5.6}$$

5.6 Etude des recopies et des stockages de données spéciaux pour les autres améliorations *M2L*

Il est légitime de penser pouvoir obtenir une meilleure efficacité, pour les versions avec FFT et avec rotations de l'opérateur *M2L*, en utilisant les *recopies* ou les stockages de données spéciaux qui viennent d'être présentés.

Nous avons donc implémenté le schéma avec *recopies* et le stockage par *ligne* pour la FFT. Les tests ont été réalisés pour une version par blocs de la FFT avec la taille de bloc usuelle de 4. Il apparaît tout d'abord que l'utilisation des *recopies* dégrade sérieusement les performances : en effet le coût des *recopies* pour la FFT par blocs est au moins deux fois plus important que celui pour le calcul par BLAS puisque la taille des tableaux pour la FFT est $2(P+1)^2$ (noyau de hauteur simple), ce qui doit être comparé à $(P+1)^2$ pour les vecteurs multipôles et $\frac{(P+1)(P+2)}{2}$ pour les vecteurs locaux. De plus, ce coût n'est pas autant amorti par la corrélation à gros grain en $\mathcal{O}(P^3)$ qu'il l'était par le calcul par BLAS et son produit matrice-matrice en $\mathcal{O}(P^4)$. Par ailleurs, à part de plus longs flots de données pour remplir les pipelines d'exécution des unités de calcul flottant, le seul gain auquel on peut s'attendre en regroupant plusieurs paires *M2L* est d'éviter le rechargement du tableau contenant les fonctions de transfert *M2L* (dans l'espace de Fourier) entre deux calculs *M2L* : il n'y a en effet pas de réutilisation des données à exploiter au niveau des tableaux contenant les développements multipôles et les développements locaux (dans l'espace de Fourier). Ce gain potentiel, portant uniquement sur les fonctions de transfert *M2L*, n'est donc pas important. Enfin, il faut noter que le schéma avec *recopies* pour la FFT implique un tableau distinct pour le développement local (dans l'espace de Fourier) de chaque cellule du niveau de l'octree, ce qui accroît sérieusement les besoins mémoire.

Néanmoins, nous avons aussi testé la FFT avec le stockage par *ligne* qui permet d'éviter les coûteuses *recopies*. Les cellules avec liste d'interaction incomplète, qui requièrent plus de *recopies* pour leurs développement locaux, restaient quant à elles traitées avec le calcul *M2L* classique. Dans ces conditions aucun ralentissement n'était mesuré, mais aucun gain non plus.

Comme expliqué en section 4.3.3, nous ne pouvons pas utiliser de routines BLAS pour accélérer le calcul *M2L* avec rotations. Sans appels BLAS, il est alors fortement vraisemblable qu'aucune accélération ne sera obtenue pour les rotations en regroupant plusieurs paires *M2L*, que ce soit avec des *recopies* ou avec un stockage par *ligne*.

Maintenant que nous avons déterminé les meilleures implémentations BLAS, à savoir *BLAS_bloc* et *BLAS_plein* (suivant la hauteur du noyau *M2L* utilisée) avec un stockage des données par *ligne*, nous pouvons les comparer en pratique avec les autres améliorations apportées à l'opérateur *M2L*.

Chapitre 6

Comparaison des différents schémas de calcul *M2L*

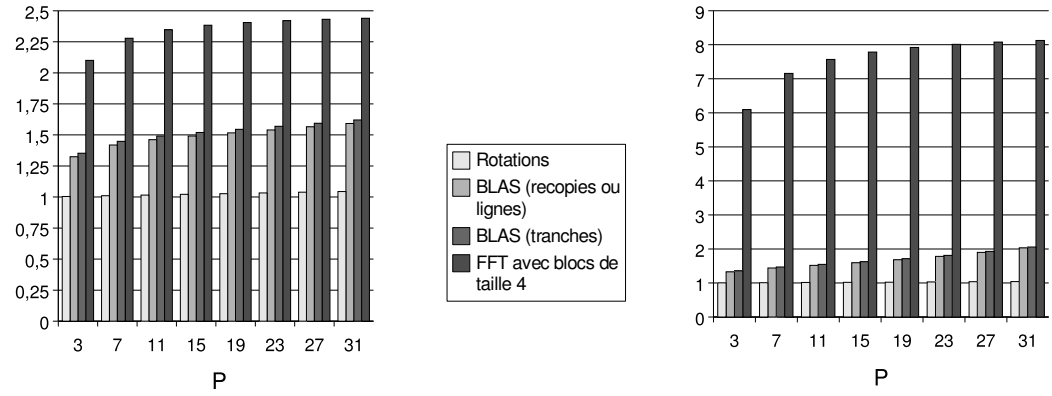
Sommaire

6.1 Besoins mémoire

Nous comparons tout d'abord les besoins mémoire des différentes schémas de calcul de l'opérateur *M2L* car ceci peut constituer le premier critère de choix. Nous considérons ici uniquement la mémoire utilisée pour les développements : la mémoire utilisée pour les particules reste en effet constante d'un schéma à l'autre. En utilisant les estimations théoriques établies aux équations (2.21) et (2.22) pour le calcul *M2L* classique, (4.12) et (4.13) pour le calcul *M2L* avec FFT, (4.33) et (4.34) pour celui avec rotations, ainsi que (5.5) et (5.6) pour le calcul par BLAS, nous avons tracé à la figure 6.1 pour chaque méthode le rapport entre ses besoins mémoire et les besoins mémoire du *M2L* classique ; nous considérons un octree de hauteur 6 et uniquement les valeurs de P qui sont des multiples de la taille de bloc de la FFT (ici 4). Pour le calcul par BLAS, nous avons utilisé $\mathcal{N}_{Buf} = 5000$ (voir section 5.5), ce qui est largement suffisant pour obtenir une efficacité optimale dans les routines BLAS de niveau 3.

Alors que les besoins mémoire du schéma avec rotations sont négligeables, et que ceux du schéma avec BLAS restent modérés, la mémoire supplémentaire requise par la FFT est problématique, surtout pour un noyau *M2L* de hauteur double. On constate les mêmes rapports quand on considère d'autres tailles de blocs pour la FFT. De plus, le rapport de 2 pour

les BLAS, dans le cas d'un noyau de hauteur double avec de grandes valeurs de P, est essentiellement dû aux matrices de transfert *M2L* pleines (voir section 5.5) : pour de plus grandes hauteurs d'octree ce rapport reste proche de $1,5$ car le nombre de ces matrices de transfert reste constant alors que le nombre de cellules (et donc de développements) dans l'octree croît de façon exponentielle. Enfin, le surcoût des *boîtes blanches* entre le stockage par *tranche* et celui par *ligne* est très faible.



Noyau de hauteur simple

Noyau de hauteur double

FIG. 6.1 – Besoins mémoire des différents schémas de calcul *M2L* (rapportés aux besoins du calcul *M2L* classique) avec un octree de hauteur 6.

6.2 Temps CPU pour un noyau *M2L* de hauteur simple

La figure 6.2 compare les différents schémas de calcul *M2L* pour un octree de hauteur 5. Si la version par BLAS surclasse le calcul *M2L* classique et celui avec rotations, la FFT par blocs est plus rapide, essentiellement pour les valeurs de P supérieures à 10. Cependant nous rappellons que, sans redimensionnement des coordonnées des particules (voir section 4.2.6), la FFT par blocs de taille 4 est instable, avec une telle hauteur, pour $P \geq 14$, et celle avec des blocs de taille 3 l'est pour $P \geq 23$: voir le tableau 4.1.

En ce qui concerne les basses et moyennes précisions, nous rappellons aussi que pour des hauteurs croissantes de l'octree, le calcul par BLAS avec stockage par *ligne* ou par *tranche* devient de plus en plus efficace puisque la longueur des lignes et des tranches augmente au niveau des feuilles (qui représente la plus grande partie du temps de calcul) : voir section 5.4.2. La figure 6.3 montre que les BLAS sont clairement compétitifs vis à vis de la FFT par blocs de taille 4, dans un octree de hauteur 7, pour des basses et moyennes précisions. De plus, les besoins mémoire du calcul avec FFT sont ici problématiques : sur la figure 6.3, la brutale augmentation du temps de calcul pour la FFT à $P = 10$ est due à de la pagination sur disque par manque de mémoire, et ce en dépit des 16 Go disponibles.

6.3 Temps CPU pour un noyau *M2L* de hauteur double

Avec un noyau de hauteur double, le calcul par BLAS surclasse clairement tous les autres schémas de calcul. La figure 6.4 compare les différents schémas pour un octree de hauteur 5.

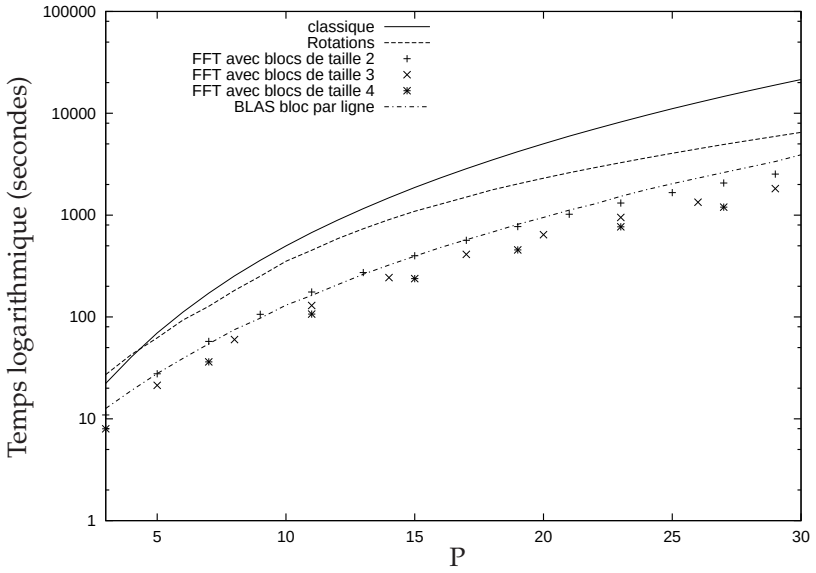


FIG. 6.2 – Temps CPU logarithmiques de la phase de descente pour les différents schémas de calcul *M2L*, avec un octree de hauteur 5 et un noyau *M2L* de hauteur simple. Tests réalisés sur un processeur IBM Power3 avec 2 Go de mémoire.

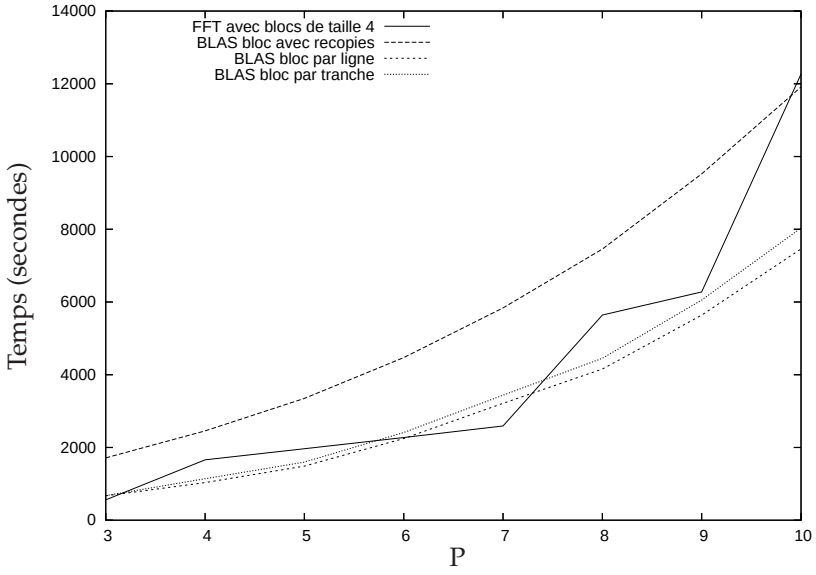


FIG. 6.3 – Temps CPU de la phase de descente pour les différents schémas de calcul *M2L*, avec un octree de hauteur 7 et un noyau *M2L* de hauteur simple. Tests réalisés sur un processeur IBM Power3 avec 16 Go de mémoire (CINES).

Pour les BLAS nous n'avons tracé que les temps du stockage des données par *ligne*, mais le stockage par *tranche* et le schéma avec *recopies* ont des performances similaires. Pour la FFT, la version par blocs de taille 4 est présentée ; dans ces tests, elle n'est stable que pour $P \leq 8$ (voir section 4.2.8), et des FFT plus stables, avec de plus petites tailles de blocs, sont encore moins

rapides. De plus, nous avons dû nous restreindre à $P \leq 19$ car avec la machine utilisée, les 2 Go de mémoire disponible étaient insuffisants pour la FFT avec $P > 19$, et ce quelle que soit la taille des blocs utilisée. Cependant il faut noter qu'avec cette hauteur d'octree, le calcul par rotations devient plus rapide que tous les calculs par BLAS pour $P \geq 23$: c'est la conséquence naturelle de la plus faible complexité théorique du calcul avec rotations. Enfin, comme pour le noyau de hauteur simple, le calcul par BLAS avec des stockages par *ligne* ou par *tranche* devient de plus en plus efficace avec des hauteurs croissantes de l'octree.

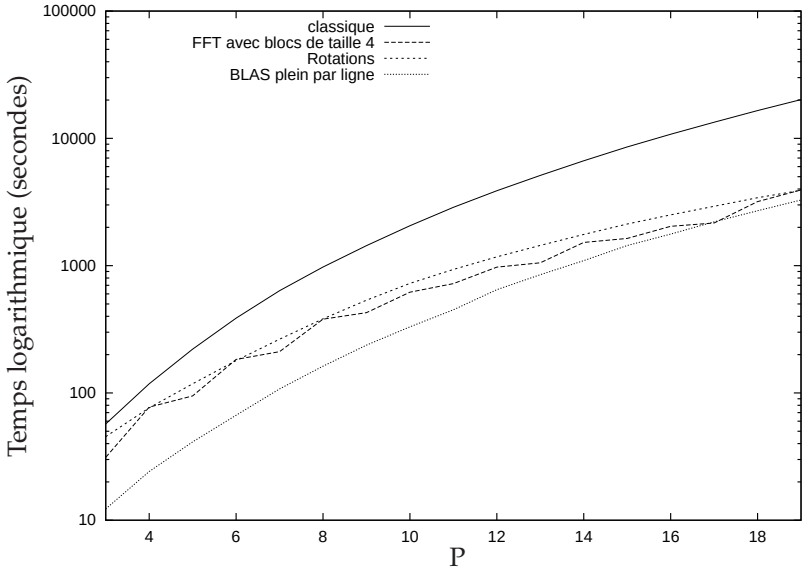


FIG. 6.4 – Temps CPU logarithmiques de la phase de descente pour les différents schémas de calcul *M2L*, avec un octree de hauteur 5 et un noyau *M2L* de hauteur double. Tests réalisés sur un processeur IBM Power3 avec 2 Go de mémoire.

6.4 Efficacité de calcul

Afin d'illustrer l'efficacité des versions BLAS qui les rendent plus rapides que les autres méthodes, nous présentons le tableau 6.1 qui montre les MFLOPS (Millions of Floating Point Operations per Seconds) atteints lors du calcul *M2L*. Les résultats sont donnés sous la forme de pourcentage par rapport à la performance crête du processeur IBM Power3 utilisé, à savoir 1500 MFLOPS. Ils ont été établis grâce au *Hardware Performance Monitor (HPM) Toolkit* (version 2.4.3), et mesurés comme la moyenne des MFLOPS obtenus sur tous les calculs *M2L* lors d'une exécution complète de la FMM. Pour les BLAS de niveau 3, nous avons utilisé le schéma par *recopies* avec les routines *BLAS_bloc* et *BLAS_plein* pour des noyaux de hauteurs simple et double (respectivement) ; l'octree est de hauteur 5. Le surcoût dû aux copies des développements n'est pas pris en compte ici puisqu'on traite de l'efficacité des routines BLAS. Nous avons de plus considéré toutes les cellules, c'est-à-dire avec liste d'interaction complète et incomplète. Par ailleurs, on rappelle que la hauteur de l'octree n'intervient pas dans l'efficacité du calcul *M2L* classique, de même que dans celles des calculs avec rotations et avec FFT.

	Noyau *M2L* de hauteur simple		Noyau *M2L* de hauteur double	
	$P = 7$	$P = 15$	$P = 7$	$P = 15$
classique	9,4 %	16,3 %	14,5 %	18,3 %
rotations	5,4 %	7,8 %	8,4 %	10,9 %
FFT avec blocs de taille 4	4,5 %	13 %	12,4 %	18,4 %
BLAS de niveau 3	46,4 %	74 %	85,9 %	89,2 %

TAB. 6.1 – Efficacité de calcul (en pourcentage d'utilisation du processeur) pour les différents schémas *M2L*.

Ces résultats ne peuvent pas être directement comparés d'une ligne à l'autre car le nombre d'opérations varie ! Néanmoins, ce tableau illustre clairement avec quel degré d'efficacité le processeur est utilisé dans les différents schémas de calcul, et pourquoi les calculs par BLAS surclassent toutes les autres méthodes.

Ce tableau indique aussi que notre décomposition du produit matrice-matrice dans le cas d'un noyau de hauteur simple par la routine *BLAS_bloc*, bien que satisfaisante, n'apparaît pas comme optimale lorsqu'on compare son efficacité à celle de la routine *BLAS_plein* du noyau de hauteur double, particulièrement pour les petites valeurs de P.

6.5 Noyau de hauteur simple contre noyau de hauteur double.

Nous avons déjà vu (voir théorème 5) qu'une borne d'erreur précise a été prouvée uniquement dans le cas d'un noyau *M2L* de hauteur simple. Un noyau *M2L* de hauteur double est certainement plus précis pour une même valeur de P, et pour une précision donnée, il devrait par conséquent imposer une plus petite valeur pour P que celui de hauteur simple. Mais comme aucune borne d'erreur précise n'est disponible pour le noyau de hauteur double, nous ne pouvons a priori pas savoir quelle sera cette plus « petite valeur de P ».

Nous comparons donc ici les temps CPU avec les précisions obtenues en pratique pour les deux hauteurs de noyaux. Comme cela a déjà été montré dans par exemple [124] pour un noyau de hauteur double, et dans [102] pour un noyau de hauteur simple, ces précisions pratiques sont meilleures que les précisions théoriques qui sont établies dans le pire des cas. Ces erreurs dans le pire des cas sont en effet obtenues pour des régions sphériques (voir section 4.1 par exemple), alors qu'en fait nous utilisons à cause de l'octree des cellules cubiques plus petites. Par ailleurs, nous utilisons des erreurs relatives au lieu des erreurs absolues afin que les résultats soient indépendants de la simulation effectuée, et ces erreurs relatives sont calculées pour le potentiel uniquement, et non pour les composantes des forces, car ces composantes peuvent être quasiment nulles. Comme certaines discontinuités apparaissent dans l'erreur du potentiel pour les particules franchissant les frontières des cellules (voir [102]), nous choisissons d'étudier l'erreur moyenne RMS (*Root Mean Square*) qui est plus stable que l'erreur maximale sur toutes les particules. Cette erreur moyenne RMS notée dans la suite ε_{rms} est calculée ainsi :

$$\varepsilon_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\frac{\Phi_{Dir}(\mathbf{X}_i) - \Phi_{FMM}(\mathbf{X}_i)}{\Phi_{Dir}(\mathbf{X}_i)}\right)^2}.$$

Nous considérons ici un potentiel gravitationnel calculé pour une distribution uniforme de 100 000 particules et un octree de hauteur 4, ce qui donne 25 particules par feuille en moyenne. Nous rappellons que lorsque le nombre de particules par feuille augmente, la part du champ proche (calculé directement) dans le potentiel devient plus importante, et le potentiel est alors automatiquement plus précis ; une moyenne de 25 particules par feuille garantit un minimum d'influence du champ lointain sur le potentiel, même pour de petites valeurs de P. La figure 6.5 présente, pour chaque schéma de calcul *M2L*, la relation entre les erreurs obtenues en pratique et les temps CPU correspondants, pour la phase de descente uniquement, avec des noyaux de hauteurs simple et double. Les échelles sont logarithmiques ; les valeurs de P utilisées pour le noyau de hauteur double vont de 3 à 14 par pas de 1, alors que pour le noyau de hauteur simple, elles vont de 3 à 29 avec un pas de 2 : $P = 14$ en hauteur double et $P = 29$ en hauteur simple correspondent en effet à la même précision (à savoir la première précision obtenue en dessous de 10^{-9} dans notre simulation).

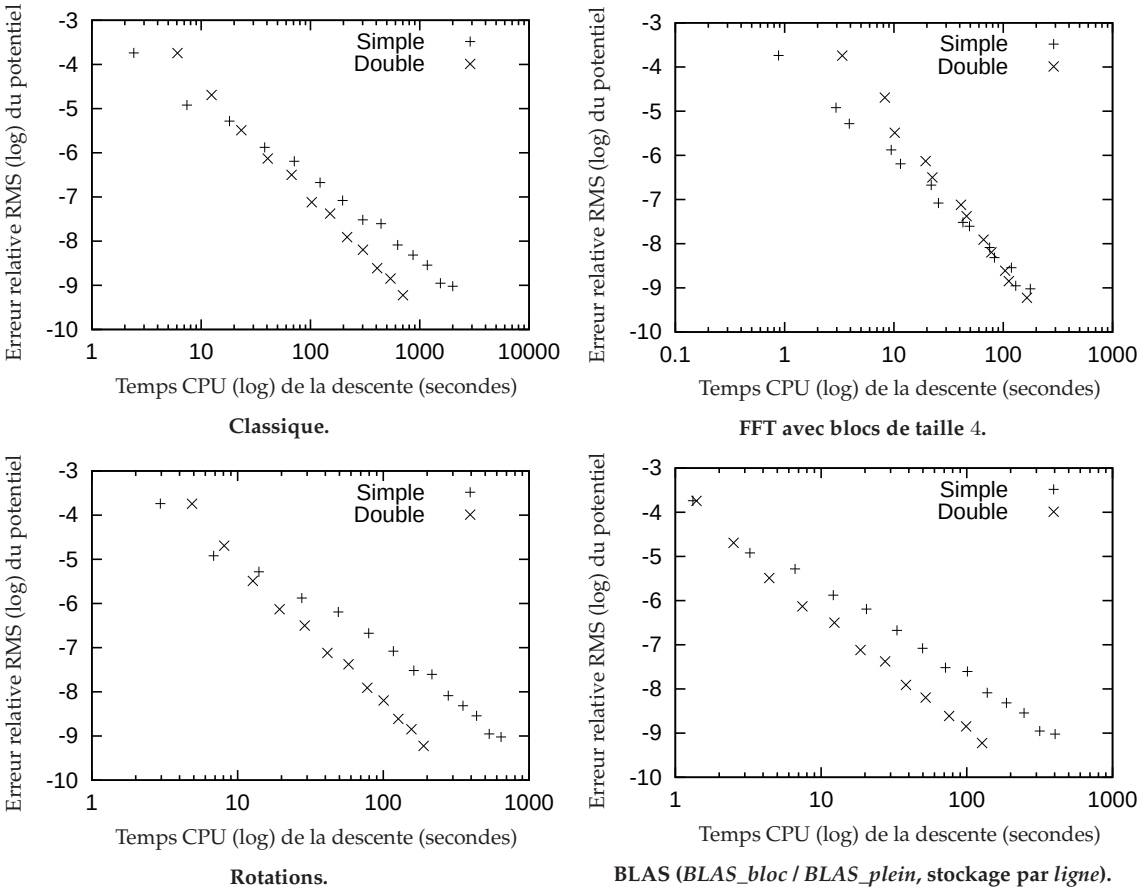


FIG. 6.5 – Relation entre les erreurs obtenues en pratique et les temps CPU pour des noyaux de hauteurs simple et double.

En ce qui concerne le calcul *M2L* classique, le noyau de hauteur simple est plus efficace pour les basses précisions, alors que c'est celui de hauteur double qui est plus efficace pour les hautes précisions. Ceci diffère légèrement des résultats présentés dans [51] (section C.3.1) où une brève comparaison, mettant en relation le nombre théorique d'opérations et la précision, montrait que les deux noyaux étaient théoriquement aussi efficaces l'un que l'autre.

Et, comme justifié théoriquement aux section 2.3.3.5, 4.2, 4.3 et 6.4, alors que les calculs avec rotations et avec BLAS favorisent le noyau de hauteur double, excepté pour les premières valeurs de P où le noyau de hauteur simple est plus efficace, celui avec FFT est généralement plus efficace avec le noyau de hauteur simple.

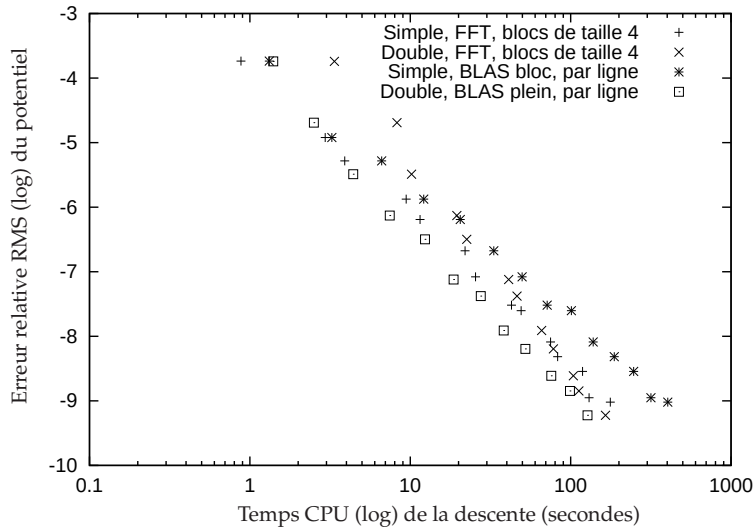


FIG. 6.6 – FFT avec blocs de taille 4 contre BLAS (*BLAS_bloc / BLAS_plein*, stockage par *ligne*).

Finalement, il nous reste à comparer entre eux les meilleurs schémas de calcul des deux noyaux : ceci est fait à la figure 6.6 où nous rassemblons les versions avec FFT et avec BLAS sur le même graphique, le calcul classique et celui avec rotations ayant déjà été écartés aux sections 6.2 et 6.3. Les BLAS avec un noyau de hauteur double sont alors clairement plus efficaces que la FFT avec un noyau de hauteur simple : par exemple, pour une erreur en pratique inférieure à 10^{-6} (respectivement 10^{-8}) le gain est de 35% (respectivement 30%). Ceci jusitifie donc pleinement l'intérêt de notre nouvelle version par BLAS, en comparaison des précédentes améliorations apportées au calcul *M2L*.

Chapitre 7

Comparaison avec les ondes planes

Sommaire

Pour clore cette partie sur les différents schémas de calcul *M2L* dans le cas uniforme, nous allons finalement comparer notre approche matricielle et ses routines BLAS avec la dernière amélioration apportée au calcul *M2L* par l'utilisation des ondes planes. L'intérêt principal des ondes planes est que le produit de deux ondes planes est une onde plane : la translation du centre d'un développement en ondes planes va donc pouvoir être effectuée plus rapidement que pour un développement en harmoniques sphériques.

7.1 Présentation des ondes planes

Nous présentons tout d'abord brièvement les formules et le principe de l'utilisation des ondes planes dans la FMM, tels qu'ils ont été décrits par Greengard & Rokhlin dans [68] pour une FMM uniforme, et par Cheng, Greengard & Rokhlin dans [34] pour une FMM adaptative.

On commence par exprimer le potentiel en un point $\mathbf{X}$, de coordonnées cartésiennes (x, y, z), dû à une charge unitaire placée en $\mathbf{X}' = (x', y', z')$, par l'intégrale double suivante (d'après [90], p.1256) :

$$\frac{1}{\|\mathbf{X} - \mathbf{X}'\|} = \frac{1}{2\pi} \int_0^\infty e^{-\lambda(z-z')} \int_0^{2\pi} e^{i\lambda((x-x')\cos(\alpha)+(y-y')\sin(\alpha))} d\alpha \, d\lambda.$$

En fonction de la précision ϵ souhaitée, cette intégrale est évaluée par des formules de quadrature dont l'expression générale vérifie :

$$\left| \frac{1}{\|\mathbf{X}-\mathbf{X}'\|} - \sum_{k=1}^{s(\epsilon)} \frac{\omega_k}{\mathcal{M}_k} \sum_{j=1}^{\mathcal{M}_k} e^{-\lambda_k(z-z')} e^{i\lambda_k((x-x')\cos(\alpha_{j,k})+(y-y')\sin(\alpha_{j,k}))} \right| < \epsilon$$

avec $\alpha_{j,k} = 2\pi j/\mathcal{M}_k$, et sous les hypothèses :

$$1 \leq z - z' \leq 4, \quad \text{et} \quad 0 \leq \sqrt{(x-x')^2 + (y-y')^2} \leq 4\sqrt{2}. \tag{7.1}$$

L'entier $s(\epsilon)$ dépend de la précision souhaitée et il en va de même pour les « poids » $\{\omega_k, k \in [\![1, s(\epsilon)]\!]\}$, les points de quadrature (ou « nœuds ») $\{\lambda_k, k \in [\![1, s(\epsilon)]\!]\}$, et les entiers $\{\mathcal{M}_k, k \in [\![1, s(\epsilon)]\!]\}$. Seules trois précisions sont disponibles dans les articles [68] et [34]. Ces précisions correspondent à des bornes d'erreur théoriques qui sont garanties par la méthode. Les formules de quadrature ayant été légèrement améliorées dans [34], ce sont celles que nous retenons et nous donnons dans le tableau 7.1 pour chacune des précisions les valeurs de $s(\epsilon)$ correspondantes, ainsi que celles du nombre total d'ondes planes utilisées : $S_{exp} = \sum_{k=1}^{s(\epsilon)} \mathcal{M}_k$. Les tableaux contenant les « poids » ω_k, les « nœuds » λ_k et les entiers $\mathcal{M}_k$ sont donnés en annexe de l'article [34]. Les détails de la procédure permettant d'obtenir ces quadratures, et d'en construire de nouvelles pour d'autres précisions, sont donnés par Yarvin, Rokhlin & Cheng dans [35] [126].

	Précision	$s(\epsilon)$	S_{exp}	P
$iprec = 0$	$1,6 \times 10^{-3}$	8	52	10
$iprec = 1$	$1,3 \times 10^{-6}$	17	258	19
$iprec = 2$	$1,1 \times 10^{-9}$	26	670	29

TAB. 7.1 – Valeurs de $s(\epsilon)$ et S_{exp}, ainsi que du P correspondant, pour les trois précisions disponibles. Chaque précision est identifiée par une valeur de la variable $iprec$ (voir section 7.2).

Quant aux hypothèses (7.1) spécifiées dans la majoration d'erreur, elles correspondent dans notre cas à une partie de la liste d'interaction, nommée *Downlist* dans [34], et qui englobe tous les éléments de la liste d'interaction d'une cellule b qui sont situées « sous » b, et séparés d'au moins une cellule dans la direction $-z$: voir figure 7.1. Dans [34], ces hypothèses correspondent à la *Uplist* (définie par opposition à la *Downlist*) car les auteurs utilisent la liste d'interaction inverse, où les cellules sources et les cellules cibles sont interverties, ce qui revient au même au final.

Dans [34] et [68], ces formulations en ondes planes sont utilisées au niveau du calcul de l'opérateur *M2L* de la FMM, les opérateurs *M2M* et *L2L*, dont l'importance est moindre, étant calculés à l'aide d'un schéma par rotations similaire à celui présenté en section 4.3. Le but est là encore d'accélérer le calcul de cet opérateur *M2L* : les développements en ondes planes, correspondant aux formules de quadrature, étant basés sur les coordonnées cartésiennes, la translation de leur centre pourra en effet être effectuée en $\mathcal{O}(P^2)$, contre $\mathcal{O}(P^4)$ pour les harmoniques sphériques basées sur les coordonnées sphériques.

Afin de détailler ce nouveau schéma de calcul *M2L*, nous écartons par souci de clarté les majorations d'erreur associées aux différents développements, et nous commençons par

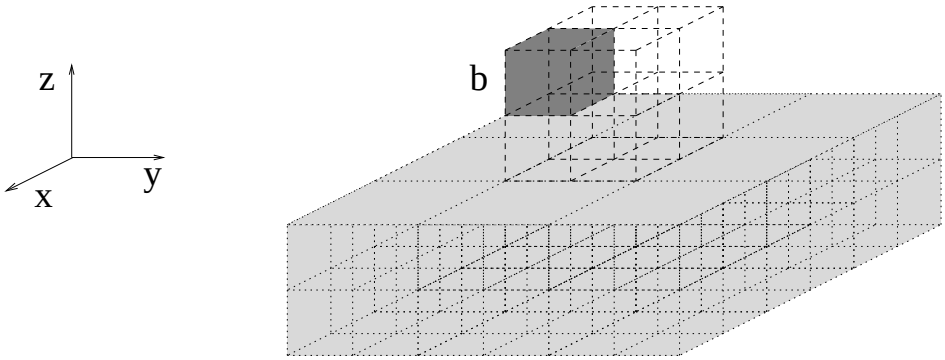


FIG. 7.1 – Représentation de la *Downlist* (en gris clair) de la cellule b (en gris foncé).

reprendre la définition des harmoniques sphériques utilisée dans [68] et [34] :

$$Y_n^m(\theta,\phi) = \sqrt{\frac{(n-|m|)!}{(n+|m|)!}} P_n^{|m|}(\cos\theta) e^{im\phi}, \quad \forall n \geq 0, \forall |m| \leq n.$$

Pour introduire les ondes planes dans le calcul *M2L*, celui-ci va être décomposé en plusieurs étapes. Le développement multipôle d'une cellule donnée est tout d'abord converti en un développement en ondes planes.

Soit une cellule cible b et une cellule source $c \in \mathit{Downlist}(b)$, toutes deux de côté d, c étant centrée à l'origine, et soient M_n^m les termes du développement multipôle de c. Le potentiel en un point $\mathbf{X} = (x,y,z)$ de coordonnées sphériques (r,θ,ϕ) appartenant à b est évalué à l'aide de ce développement multipôle sous la forme

$$\Phi(\mathbf{X}) \approx \sum_{n=0}^{P} \sum_{m=-n}^{n} \frac{M_n^m}{r^{n+1}} Y_n^m(\theta,\phi).$$

A partir de ce développement multipôle, et grâce à l'expression des harmoniques sphériques sous la forme de dérivées partielles de $1/r$ (voir théorème 7.1 dans [68]), on construit le développement en ondes planes suivant

$$\Phi(\mathbf{X}) \approx \sum_{k=1}^{s(\epsilon)} \sum_{j=1}^{\mathcal{M}_k} W(k,j) e^{-(\lambda_k/d)z} e^{i(\lambda_k/d)(x\cos(\alpha_{j,k})+y\sin(\alpha_{j,k}))}$$

où les coefficients $W(k,j)$ sont définis $\forall k \in [\![1, s(\epsilon)]\!]$, $\forall j \in [\![1, \mathcal{M}_k]\!]$ par

$$W(k,j) = \frac{\omega_k/d}{\mathcal{M}_k} \sum_{m=-P}^{P} (-i)^{|m|} e^{im\alpha_{j,k}} \sum_{n=|m|}^{P} \frac{M_n^m}{\sqrt{(n-m)!(n+m)!}} (\lambda_k/d)^n.$$

Le développement en ondes planes centré sur la cellule source c est alors translaté au centre $\mathbf{X}'' = (x'',y'',z'')$ de la cellule cible b. La cellule source étant centrée à l'origine, le développement translaté s'exprime ainsi :

$$\Phi(\mathbf{X}) \approx \sum_{k=1}^{s(\epsilon)} \sum_{j=1}^{\mathcal{M}_k} V(k,j) e^{-\lambda_k(z-z'')} e^{i\lambda_k((x-x'')\cos(\alpha_{j,k})+(y-y'')\sin(\alpha_{j,k}))}$$

où

$$V(k,j) = W(k,j)e^{-\lambda_k z''}e^{i\lambda_k(x''\cos(\alpha_{j,k})+y''\sin(\alpha_{j,k}))}, \quad \forall k \in [\![1, s(\epsilon)]\!], \forall j \in [\![1, \mathcal{M}_k]\!].$$

Enfin, il reste à convertir le développement en ondes places ainsi obtenu en un développement local pour la cellule cible b. En supposant pour simplifier que la cellule cible est désormais centrée à l'origine, les termes de ce développement local sont alors calculés ainsi

$$L_n^m = \frac{(-i)^{|m|}}{\sqrt{(n-m)!(n+m)!}} \sum_{k=1}^{s(\epsilon)} (-\lambda_k/d)^n \sum_{j=1}^{\mathcal{M}_k} W(k,j)e^{im\alpha_{j,k}}, \quad \forall n \in [\![0, P]\!], \forall m \in [\![-n, n]\!],$$

et le potentiel est évalué au point $\mathbf{X}$ à l'aide de ce développement local :

$$\Phi(\mathbf{X}) \approx \sum_{n=0}^{P} \sum_{m=-n}^{n} L_n^m Y_n^m(\theta,\phi) r^n.$$

Pour revenir à la majoration de l'erreur, l'erreur due aux développements en ondes planes doit concorder avec celle introduite par les développements multipôles, ce qui impose selon [34] :

$$s(\epsilon) \sim P, \tag{7.2}$$

et

$$S_{exp} \sim P^2. \tag{7.3}$$

Le tableau 7.1 présente les valeurs de P utilisées en pratique pour chacune des précisions, et nous renvoyons le lecteur aux articles [34] et [68] pour la justification théorique de la majoration précise de l'erreur.

En factorisant certains termes, et grâce aux relations (7.2) et (7.3), le coût de la conversion du développement multipôle en ondes planes s'effectue alors en $\mathcal{O}(P^3)$. Il en va de même de la conversion des ondes planes en développement local. La translation du centre du développement en ondes planes s'effectue quant à elle en $\mathcal{O}(P^2)$, ce qui justifie l'intérêt de l'introduction des ondes planes.

Tout ceci est valable dans le cas des cellules de la *Downlist*. Le cas des cellules appartenant à la *Uplist* est traité similairement. Par contre pour les autres cellules de la liste d'interaction, on se ramène au cas de la *Downlist* à l'aide de rotations du système de coordonnées en $\mathcal{O}(P^3)$ similaires à celles présentées en section 4.3. Au final les 189 opérations *M2L* (du cas $ws = 1$) sont ramenées à 189 opérations en $\mathcal{O}(P^2)$ plus 20 opérations en $\mathcal{O}(P^3)$.

D'après [68], les 189 opérations en $\mathcal{O}(P^2)$ peuvent de plus être ramenées à 40. A la manière des *super-nodes* (voir section 2.3.3.10), les ondes planes correspondant aux développements multipôles de 8 fils peuvent être accumulées au niveau de leur père et ainsi translatées ensemble au centre d'une même cellule cible. Et contrairement aux développements multipôles et locaux, ceci est réalisé sans perte de précision avec les ondes planes.

7.2 Comparaison avec le schéma de calcul par BLAS

Après cette description du schéma de calcul *M2L* avec ondes planes, nous nous proposons de le comparer avec notre version par BLAS (code FMB). Concrètement, nous allons

étudier les temps CPU en fonction de l'erreur engendrée en pratique par les deux algorithmes.

On peut tout d'abord remarquer que Greengard *et al.* n'utilisant pas les fonctions *Outer* et *Inner* (voir section 2.3.3.2), leurs expressions des opérations *M2M*, *L2L* et *M2L* sont plus complexes, et donc plus coûteuses en termes d'opérations que notre formulation.

Pour effectuer la comparaison, nous utilisons le code FMMPART3D (version 1.0), distribué par la société MadMax Optics[1] sous la forme d'une librairie précompilée en FORTRAN. Les comparaisons sont effectuées sur un PC Linux équipé d'un processeur Pentium 4 cadencé à 2,6 Ghz disposant de 2 Go de mémoire, d'un cache L1 de données de 8 Ko, et d'un cache L2 de 512 Ko. Le code FMB est compilé avec gcc 4.0.4 et utilise la librairie BLAS ATLAS (version 3.6.0) [1] [123]. Le code FMMPART3D est compilé avec le compilateur Lahey/Fujitsu Fortran 95 (version 6.2).

Nous restons ici dans le cas de simulations uniformes de particules. Comme dans l'exemple accompagnant FMMPART3D, et comme dans les articles [34] et [68], nous utilisons ici comme mesure de l'erreur pratique la norme L_2 définie ainsi :

$$\varepsilon_{L_2} = \sqrt{\frac{\sum_{i=1}^{M} \left(\Phi_{Dir}(\mathbf{X}_i) - \Phi_{FMM}(\mathbf{X}_i)\right)^2}{\sum_{i=1}^{M} \Phi_{Dir}(\mathbf{X}_i)^2}}.$$

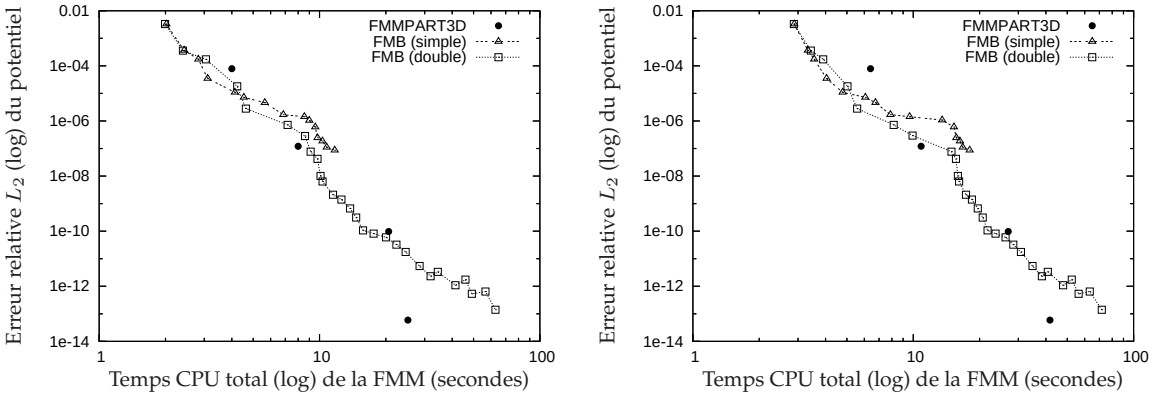
La méthode directe est alors utilisée pour calculer le potentiel exact Φ_{Dir} de $M = 1000$ corps (choisis aléatoirement parmi les N corps), et l'erreur engendrée par FMMPART3D et par FMB pour ces M corps est calculée avec la norme L_2. Comme pour la FMM classique, on constate que l'erreur engendrée en pratique par FMMPART3D pour chaque précision $iprec$ est bien plus faible que celle garantie en théorie (voir tableau 7.1).

Le code FMMPART3D propose les trois précisions présentées dans le tableau 7.1 plus une précision supplémentaire à 10^{-12} (brièvement présentée dans [34]), nécessitant $S_{exp} = 1292$ ondes planes et $P = 40$, désignée par $iprec = 3$. Pour le code FMB, les valeurs de P varient de 1 à 28, de 1 en 1, afin de couvrir tout l'intervalle de précisions possibles jusqu'à atteindre la plus haute précision offerte par FMMPART3D. Nous utilisons ici le schéma de calcul avec les routines BLAS de niveau 3 et le stockage des données par *ligne* (lorsque le niveau l de l'octree est suffisamment grand ($l \geq 5$), sinon le schéma avec *recopies* est utilisé).

Il y a cependant des différences d'ordre algorithmique entre les deux codes qu'il est important de préciser.

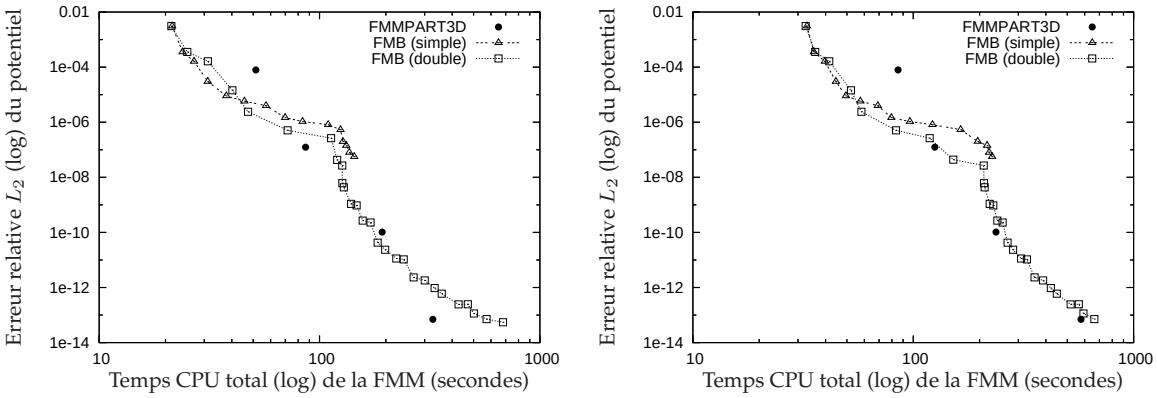
- Dans FMB nous utilisons des noyaux *M2L* de hauteurs simple et double, et comme nous l'avons fait remarqué en section 6.5, c'est uniquement pour les petites valeurs de P que, dans le cas des BLAS, le noyau *M2L* de hauteur simple est plus efficace que celui de hauteur double. Le noyau de hauteur simple n'est donc utilisé en pratique que jusqu'à $P = 15$. Dans FMMPART3D et dans les articles [34] [68], le choix du noyau n'est pas précisé, ce qui suggère un noyau de hauteur double (plus naturel et plus courant).
- En ce qui concerne la hauteur de l'octree, elle est reglée de façon optimale par l'utilisateur dans notre code FMB afin d'équilibrer le coût du calcul (direct) du champ proche et le coût du calcul (avec les développements) du champ lointain. Lorsque P augmente, le coût du calcul du champ lointain augmente aussi, et nous sommes

[1] http://www.madmaxoptics.com/technology/products/FMMPART3D.html



(a) Potentiel uniquement. (b) Force et potentiel.

FIG. 7.2 – Comparaison entre FMMPART3D et FMB pour une distribution uniforme de 100 000 particules.



(a) Potentiel uniquement. (b) Force et potentiel.

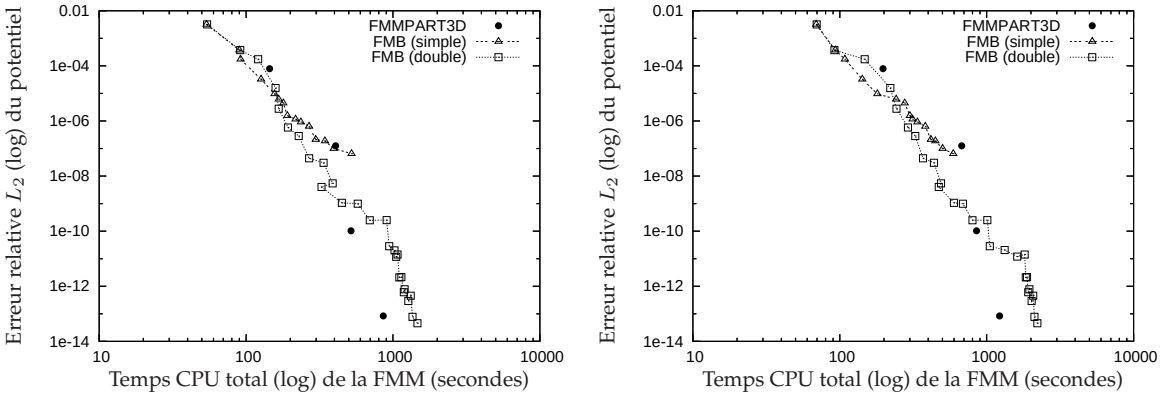
FIG. 7.3 – Comparaison entre FMMPART3D et FMB pour une distribution uniforme de 1 000 000 particules.

alors éventuellement amenés à réduire la hauteur de l'octree. Par contre FMMPART3D est un code FMM adaptatif, dont l'algorithme est décrit dans [32] et [34] (voir section 2.3.3.9) : le code impose un nombre maximal de particules par feuilles (40 pour $1,6 \times 10^{-3}$, 100 pour $1,3 \times 10^{-6}$, et 180 pour $1,1 \times 10^{-9}$, d'après [34]).

– Dans FMMPART3D, les opérations *M2M* et *L2L* sont effectuées à l'aide d'un schéma de calcul par rotations en $\mathcal{O}(P^3)$, alors que dans FMB nous utilisons le calcul classique en $\mathcal{O}(P^4)$. Ces opérations sont cependant nettement minoritaires dans le coût de calcul global de la FMM pour des distributions uniformes.

Les figures 7.2, 7.3 et 7.4 présentent l'évolution entre le temps CPU et l'erreur ε_{L_2} pour les deux codes, et ce pour des distributions uniformes de 100 000, 1 million, et 3 millions de particules. Dans chaque figure, les quatres points pour FMMPART3D correspondent aux quatres valeurs possibles de la variable $iprec$ (de 0 à 3).

Plusieurs conclusions peuvent en être tirées.



(a) Potentiel uniquement. (b) Force et potentiel.

FIG. 7.4 – Comparaison entre FMMPART3D et FMB pour une distribution uniforme de 3 000 000 particules.

– On remarque tout d'abord à l'aide du tableau 7.2 que pour chaque précision offerte en pratique par FMMPART3D, la valeur de P nécessaire pour atteindre (voire dépasser) cette précision dans FMB est bien plus faible que celle utilisée dans FMMPART3D (voir les valeurs de P présentées au tableau 7.1). Un tel écart est probablement dû à une perte de précision de la FMM à cause de l'introduction des ondes planes, l'erreur générée en pratique par les développements en ondes planes étant supérieure à celle introduite par les développements en harmoniques sphériques[2]. Cette erreur supplémentaire due aux ondes planes doit pouvoir être réduite en augmentant $s(\epsilon)$ et S_{exp} mais ceci augmentera aussi le nombre d'opérations du schéma de calcul par ondes planes.

P	$iprec = 0$	$iprec = 1$	$iprec = 2$	$iprec = 3$
FMMPART3D	10	19	29	40
FMB (noyau *M2L* de hauteur double)	4	8	16	26-28
FMB (noyau *M2L* de hauteur simple)	4	14	-	-

TAB. 7.2 – Pour chacune des quatres précisions offerte en pratique par FMMPART3D : valeurs de P utilisées dans FMMPART3D et valeurs de P permettant d'atteindre (ou de dépasser) cette précision dans FMB.

– Il apparaît aussi que FMB est plus performant, par rapport à FMMPART3D, lorsqu'on calcule à la fois le potentiel et la force, que lorsqu'on calcule uniquement le potentiel. Ceci peut s'expliquer par une implémentation plus efficace de notre calcul direct (qui représente en moyenne la moitié du temps de calcul) au niveau de la force, ou par une version plus efficace de l'opérateur *L2P*, qui évalue la force à partir des développements locaux. Ceci étant dans les deux cas étranger au calcul *M2L*, nous nous

[2]L'erreur générée en pratique par FMMPART3D reste bien sûr conforme aux bornes théoriques du tableau 7.1.

concentrons sur les temps où seul le potentiel est calculé.

- Comme annoncé, le noyau *M2L* de hauteur simple est plus efficace que celui de hauteur double uniquement pour les petites valeurs de P (en pratique pour $2 \leq P \leq 5-6$).
- Enfin, si on compare les temps CPU pour une précision donnée et pour le calcul du seul potentiel, on remarque que FMB est toujours plus performant pour la première précision ($iprec = 0$), avec par exemple un gain de 39% avec $P = 4$ (noyau *M2L* de hauteur simple) dans le cas du million de particules. Pour les deux précisions suivantes ($iprec = 1, 2$), FMB est aussi performant que FMMPART3D, mais moins performant pour la plus haute précision ($iprec = 3$).
  Ceci illustre parfaitement le gain en performance des BLAS qui conserve la complexité théorique du calcul classique mais font chuter le facteur constant : pour les petites et moyennes valeurs de P, le gain offert par les BLAS est prépondérant, mais pour les plus grandes valeurs de P le gain en complexité théorique apporté par les ondes planes devient décisif. Les cassures dans les courbes de FMB sont quant à elles dues aux changements de hauteur de notre octree uniforme. Les hauteurs d'octree utilisées étant plus faibles dans le cas de 100 000 particules, notre schéma de calcul par BLAS est moins efficace, mais les mêmes phénomènes sont néanmoins observés. Dans les cas d'un million et de trois millions de particules, les hauteurs d'octree sont les mêmes (si l'on excepte le fait que le changement de hauteur arrive plus tard avec trois millions de particules) et les différences sont ici essentiellement dues à la part plus importante du calcul direct dans le cas de trois millions de particules. La mémoire disponible sur la machine ne permettait pas de réaliser de tests offrant un niveau supplémentaire dans l'octree : avec une hauteur d'octree plus grande, notre version par BLAS gagnerait pourtant davantage en efficacité.

En dynamique moléculaire, les précisions qui nous intéressent se situent entre 10^{-5} et 10^{-7} (voir en section 2.3.3.2) : ceci correspond donc à la gamme de valeurs de P où notre code FMB est soit meilleur, soit aussi rapide, que le code FMMPART3D. Ceci explique par ailleurs pourquoi les ondes planes n'ont pas été reprises, depuis [34], dans le cadre des équations de Poisson : contrairement au cas des équations de Maxwell et de Helmholtz où les valeurs de P sont grandes, les valeurs de P requises en dynamique moléculaire limitent le gain apporté par la complexité théorique en $\mathcal{O}(P^2) + \mathcal{O}(P^3)$, et le facteur constant sous-jacent à cette complexité n'est donc pas négligeable.

Conclusion

En conclusion, nous avons présenté dans cette partie II une étude générale de la meilleure implémentation de la FMM pour des distributions uniformes de particules souvent rencontrées en dynamique moléculaire.

Une étude détaillée de la borne d'erreur nous a tout d'abord conduit à deux expressions de l'opérateur *M2L* qui convertit un développement multipôle en un développement local : alors que le *noyau* M2L *de hauteur double* est généralement plus efficace, seul le *noyau* M2L *de hauteur simple* respecte une borne d'erreur précise.

Nous avons ensuite étudié plusieurs schémas de calcul *M2L* dont nous rappelons les différentes complexités théoriques au tableau 7.3. Pour chaque hauteur de noyau *M2L*, nous avons ainsi efficacement implémenté les schémas basés sur la Transformée Rapide de Fourier (FFT) par blocs et sur les rotations. La première implémentation du schéma par FFT pour un noyau *M2L* de hauteur double a ainsi été effectuée, et nous avons montré que des instabilités numériques subsistaient dans la version par blocs de la FFT. Le schéma par rotations a lui aussi été détaillé et adapté aux formules utilisées dans notre implémentation de la FMM.

	calcul *M2L* classique	FFT par blocs	Rotations	Ondes planes	BLAS
Complexité théorique	$\mathcal{O}(P^4)$	$\mathcal{O}(P^3)$	$\mathcal{O}(P^3)$	$\mathcal{O}(P^2)$	$\mathcal{O}(P^4)$

TAB. 7.3 – Complexité théorique de l'opérateur *M2L* en fonction des différents schémas de calcul *M2L* étudiés. Le nombre précis d'opérations est donné aux sections 2.3.3.5, 4.2.4, 4.2.8, 4.3.1.3 et 7.1. Dans le cas des ondes planes, il faut néanmoins préciser que pour l'ensemble des 189 opérations *M2L* d'une liste d'interaction donnée, 20 opérations supplémentaires en $\mathcal{O}(P^3)$ sont nécessaires.

Nous avons alors introduit pour l'opérateur *M2L* un nouveau schéma de calcul à base de routines BLAS (Basic Linear Algebra Subprograms), inédit pour les développements en harmoniques sphériques de la FMM. Ce schéma a été élaboré pour les matrices pleines du noyau de hauteur double comme pour les matrices creuses du noyau de hauteur simple. Bien que ce schéma de calcul conserve la complexité théorique originale en $\mathcal{O}(P^4)$ du calcul *M2L* classique, les gains offerts par l'implémentation haute performancees des routines BLAS sont considérables. Un schéma avec *recopies* a d'abord permis d'employer des routines BLAS de niveau 3 et ainsi d'obtenir de bien meilleurs performances. Des stockages de données spéciaux, par *ligne* ou par *tranche* nous ont ensuite permis d'éviter le surcoût dû à ces recopies, en particulier pour les petites valeurs de P requises pour les basses précisions. Plus la hauteur de l'octree est grande, plus la taille de nos produits matriciels est importante, et meilleure est l'efficacité de nos schémas par BLAS.

Les complexités théoriques précises et les besoins mémoires de chaque schéma de calcul ont aussi été donnés pour les deux hauteurs de noyaux *M2L*. A partir de ces résultats, et en s'appuyant sur les temps CPU de diverses exécutions de la FMM, nous avons pu précisément comparer ces différents schémas. Il apparaît que la version par BLAS et l'amélioration basée sur la FFT par blocs sont les plus efficaces. Alors que la version par BLAS est tou-

jours plus rapide dans le cas du noyau de hauteur double, la FFT par blocs est plus rapide dans le cas du noyau de hauteur simple avec des hautes précisions. Cependant les besoins mémoire de la FFT par blocs, ainsi que les instabilités numériques résiduelles, précisément dans le cas des hautes précisions, limite fortement le gain offert par ce schéma de calcul au niveau des temps CPU dans ce cas. En prenant en compte les précisions obtenues en pratique et les temps CPU de toutes les versions, le schéma par BLAS avec un noyau *M2L* de hauteur double est alors le plus efficace, tout en introduisant aucune instabilité numérique et en ayant des besoins mémoires modérés.

A l'aide du code FMMPART3D, nous avons de plus pu procéder à une comparaison entre notre schéma par BLAS et le schéma par ondes planes. Nous avons alors montré que le facteur constant sous-jacent à la complexité en $\mathcal{O}(P^2) + \mathcal{O}(P^3)$ de l'opérateur *M2L* avec les ondes planes n'est pas négligeable, et que dans le cadre des valeurs de P qui nous intéressent, le gain apporté par les routines BLAS est soit supérieur, soit égal, à celui obtenu avec les ondes planes.

L'intérêt de notre approche matricielle et de ses routines BLAS est donc totalement justifié dans le cas uniforme. Afin d'optimiser l'efficacité de la FMM, nous utiliserons désormais le noyau *M2L* de hauteur double pour toutes les valeurs de P, excepté les premières ($P \leq 3$) pour lesquelles on préférera un noyau *M2L* de hauteur simple. Nous allons maintenant considérer des distributions non uniformes de particules et voir comment cette nouvelle approche peut y être adaptée.

Troisième partie

Formulation BLAS pour la FMM adaptative

Chapitre 8

Formulation BLAS pour la FMM adaptative

Sommaire

Nous avons montré dans la partie II l'intérêt de notre schéma de calcul par BLAS pour des distributions uniformes de particules. Nous allons désormais désormais étendre cette formulation aux distributions non uniformes.

Nous présentons tout d'abord une amélioration d'ordre algorithmique concernant la version adaptative de la FMM. Puis, comme annoncé au chapitre 3, nous introduisons une

nouvelle structure de données, *l'octree avec indirection*, qui permet de traiter efficacement les distributions uniformes et les distributions non uniformes, ce qui n'est pas le cas pour les structures actuellement utilisées (tableaux unidimensionnels de pointeurs et *hashed octree* : voir en section 2.3.1.2). Nous adaptons alors notre formulation BLAS aux distributions non uniformes, et nous décrivons comment détecter les plus grandes zones uniformes possibles dans ces distributions non uniformes : ces zones uniformes permettent en effet d'améliorer l'efficacité de la version par BLAS. Tous ces points sont ensuite validés en pratique sur différentes distributions, et nous revenons sur l'une des améliorations algorithmique proposée par Nabors *et al.* pour la FMM adaptative. Nous terminons par deux comparaisons : la première valide à nouveau notre schéma de calcul par BLAS face aux autres schémas de calcul *M2L*, cette fois dans le cas non uniforme, et la deuxième présente les résultats de notre code FMM (FMB) dans le cadre de simulations astrophysiques, où il est comparé aux codes de référence dans le domaine.

8.1 Algorithme de la FMM adaptative

Nous avons justifié au chapitre 3 le choix de la version adaptative de Nabors *et al.* [93] par rapport à celle de Cheng *et al.* [34]. Nous allons désormais montrer que l'étude de la valeur optimale pour le seuil s_{min} peut être approfondie. On rappelle que, dans la version adaptative de Nabors *et al.* [93], s_{min} désigne le nombre minimal de particules qui justifie la construction des développements multipôle et local dans une cellule donnée.

8.1.1 Estimation des seuils pour les développements multipôles et locaux

Dans [93] la même valeur de s_{min} est utilisée comme seuil pour les cellules cibles et pour les cellules sources, c'est-à-dire à la fois pour l'opérateur *P2L* et pour l'opérateur *M2P*. Mais en pratique, il apparaît que le coût en termes de temps CPU pour une opération *M2P* est environ deux fois plus important que pour une opération *P2L*. Ceci suggère le recours à deux seuils différents : s^M_{min} et s^L_{min}. Concrètement, en notant s le nombre de particules dans une cellule donnée, cette cellule aura un développement multipôle (respectivement local) uniquement si $s \geq s^M_{min}$ (respectivement $s \geq s^L_{min}$). Par conséquent une cellule peut avoir soit un développement multipôle, soit un développement local, soit les deux, soit aucun, comme résumé au tableau 8.1.

	$s \geq s^M_{min}$		$s < s^M_{min}$	
	$s \geq s^L_{min}$	$s < s^L_{min}$	$s \geq s^L_{min}$	$s < s^L_{min}$
Développement multipôle	✓	✓	-	-
Développement local	✓	-	✓	-

TAB. 8.1 – Utilisation ou non du développement multipôle et du développement local en fonction du nombre s de particules dans la cellule et des seuils s^M_{min} et s^L_{min}.

Afin de pouvoir estimer, a priori et pour chaque valeur de P, les meilleurs valeurs pour s^M_{min} et s^L_{min}, nous allons considérer l'interaction entre une cellule source et une cellule cible : les opérations possibles pour évaluer cette interaction sont énumérées au tableau 8.2. Il doit être noté que si (et seulement si) les deux cellules n'ont aucun développement, le calcul direct de cette interaction est effectué en même temps que celui de l'interaction inverse, grâce au principe des interactions réciproques. Cependant ceci est considéré uniquement comme une amélioration supplémentaire, et dans les estimations suivantes l'opération *P2P* correspond au calcul direct sans recours au principe des interactions réciproques.

cellule source	cellule cible	opération
$s \geq s^M_{min}$	$s \geq s^L_{min}$	*M2L*
$s < s^M_{min}$	$s \geq s^L_{min}$	*P2L*
$s \geq s^M_{min}$	$s < s^L_{min}$	*M2P*
$s < s^M_{min}$	$s < s^L_{min}$	*P2P*

TAB. 8.2 – Opérations possibles pour l'évaluation d'une interaction entre une cellule source et une cellule cible en fonction de s^M_{min} et de s^L_{min} : s désigne le nombre de particules dans la cellule.

Afin d'estimer les deux seuils, nous allons procéder comme suit. Comme le coût de l'opérateur *P2L* croît linéairement avec le nombre de particules utilisées, nous commençons par déterminer la plus petite valeur de s^M_{min} qui vérifie :

$$\mathcal{C}_{M2L}(P) \leq \mathcal{C}_{P2L}(P, s^M_{min}), \tag{8.1}$$

où $\mathcal{C}_{op}$ désigne le coût de l'opération *op*.

Pareillement, nous déterminons la plus petite valeur de s^L_{min} qui vérifie

$$\mathcal{C}_{M2L}(P) \leq \mathcal{C}_{M2P}(P, s^L_{min}). \tag{8.2}$$

puisque $\mathcal{C}_{M2P}(P, s)$ croît aussi linéairement en fonction de s.

Afin de vérifier ces premières estimations, nous allons maintenant prendre en compte le coût du calcul direct (*P2P*). Nous allons ainsi vérifier que[1] :

$$\forall s_m < s^M_{min}, \qquad \mathcal{C}_{M2P}(P, s_l) > \mathcal{C}_{P2P}(s_m, s_l) \qquad \forall\, s_l \in [\![1, s^L_{min} - 1]\!]. \tag{8.3}$$

Tant que ce test échoue, nous décrémentons s^M_{min}, ($s^M_{min} \leftarrow s^M_{min} - 1$), et nous testons à nouveau.

[1]Nous aurions aussi pu choisir ici la vérification opposée qui amène à des valeurs croissantes de s^M_{min}, mais comme l'introduction de ce seuil est vu comme une optimisation par rapport à l'algorithme original, nous préférons minimiser la valeur de s^M_{min} (et ceci s'applique aussi à s^L_{min}).

Pareillement, nous vérifions aussi que :

$$\forall s_l < s_{min}^L, \qquad \mathcal{C}_{P2L}(P, s_m) > \mathcal{C}_{P2P}(s_m, s_l) \qquad \forall\, s_m \in [\![1, s_{min}^M - 1]\!]. \tag{8.4}$$

Et tant que ce test échoue, nous décrémentons s_{min}^L, ($s_{min}^L \leftarrow s_{min}^L - 1$), et nous testons à nouveau.

Cependant, comme $\mathcal{C}_{P2P}(s_m, s_l)$ croît linéairement avec soit s_m soit s_l, nous avons juste à tester, à la place de (8.3) et de (8.4), que

$$\mathcal{C}_{M2P}(P, s_l) > \mathcal{C}_{P2P}(s_{min}^M - 1, s_l) \qquad \forall\, s_l \in [\![1, s_{min}^L - 1]\!],$$

et que

$$\mathcal{C}_{P2L}(P, s_m) > \mathcal{C}_{P2P}(s_m, s_{min}^L - 1) \qquad \forall\, s_m \in [\![1, s_{min}^M - 1]\!].$$

Concrètement, tous les coûts $\mathcal{C}_{op}$ peuvent être évalués soit par des mesures pratiques de temps CPU, soit par des évaluations théoriques du nombre d'opérations. Nous utilisons en fait les mesures de temps CPU car elles sont plus précises que les estimations théoriques du nombre d'opérations, en particulier pour la prise en compte de la machine, des optimisations de compilation et de l'implémentation, ce qui ne peut être négligée ici (surtout pour l'opérateur *P2P* dont le grain de calcul peut être très fin). De plus, le gain offert par les routines BLAS est difficilement modélisable (et est aussi dépendant de la machine).

Remarquons finalement que $\mathcal{C}_{M2L}(P)$ dans (8.1) et (8.2) dépend du schéma de calcul *M2L* utilisé : des tests indépendants devront être réalisés en fonction de la hauteur du noyau *M2L* pour le calcul *M2L* classique, pour le calcul avec BLAS, et pour les autres améliorations *M2L* (rotations et FFT par blocs). Cependant, pour le calcul par BLAS, seules des estimations avec les routines BLAS de niveau 2 peuvent être réalisées, car l'efficacité des routines BLAS de niveau 3 dans le calcul *M2L* dépend du nombre (a priori inconnu) de colonnes utilisées dans les produits matrices-matrices (voir section 5.3) : les routines BLAS de niveau 3 vont donc être considérées ici comme une éventuelle amélioration supplémentaire par rapport à celles de niveau 2.

8.1.2 Description de l'algorithme avec les seuils pour les développements multipôles et locaux

Maintenant que nous avons introduit les seuils s_{min}^M et s_{min}^L, nous présentons l'algorithme de notre version de la FMM adaptative. Par souci de clarté, nous ne distinguons pas ici les cellules adaptatives des cellules non adaptatives (voir section 2.3.3.9). Nous reprenons les différentes phases de l'algorithme uniforme ainsi que les notations utilisées en section 2.3.3.5. Nous notons de plus s_i le nombre de particules contenues dans la cellule i.

La phase de remontée de la version adaptative est décrite par l'algorithme 10, et celle de descente par l'algorithme 11, La phase de calcul direct reste quant à elle inchangée par rapport à la FMM uniforme, et celle d'évaluation est décrite par l'algorithme 12.

8.1.3 Premières conséquences sur la structure de données

Les changements algorithmiques de la version adaptative imposent des modifications au niveau des structures de données utilisées jusqu'à présent dans la FMM uniforme.

Algorithme 10 FMM adaptative en $3D$: **PHASE DE REMONTEE**

Pour $i = 1$ à 8^H **faire**
  Si $s_i \geq s_{min}^M$ **Alors**
    Calculer $\mathcal{M}_{H,i}$ grâce à l'opérateur *P2M* ;
  Fin si
Fin pour
Pour $l = H - 1$ à 0 **faire**
  Pour $i = 1$ à 8^l **faire**
    Pour $j = 0$ à 8 **faire**
      Si $s_{j^{\text{ième}} \text{ fils de } i} \geq s_{min}^M$ **Alors**
        Translater $\mathcal{M}_{l+1,8.i+j}$ du $j^{\text{ième}}$ fils de la cellule i au centre de i grâce à l'opérateur *M2M*, et l'additionner à $\mathcal{M}_{l,i}$;
      Sinon
        Construire $\mathcal{M}_{l+1,8.i+j}$ au centre de i grâce à l'opérateur *P2M*, et l'additionner à $\mathcal{M}_{l,i}$;
      Fin si
    Fin pour
  Fin pour
Fin pour

- Contrairement au cas uniforme où l'accès aux particules ne se faisait qu'au niveau des feuilles, l'introduction des seuils s_{min}^M et s_{min}^L requiert de pouvoir accéder à toutes les particules contenues dans une cellule interne (pour les opérations *P2L*, *M2P* et *P2P*). Nous disposons donc désormais dans chaque cellule d'un pointeur permettant d'accéder aux particules situées à l'intérieur de cette cellule. Si la cellule c est une feuille, le pointeur correspond directement au tableau contenant ses particules, mais si c est une cellule interne le pointeur correspond à une liste intermédiaire de pointeurs vers les tableaux de particules de chaque feuille (non vide) descendant de c.
- La distinction entre cellules adaptatives et cellules non adaptatives, présentée en section 2.3.3.9, impose quant à elle de repérer les cellules non adaptatives, qui correspondent aux cellules à l'intérieur d'une chaîne (voir figure 2.17). Pour cela, chaque cellule non adaptative c contient l'indice, et le niveau, de la cellule adaptative e située à l'extrémité de la chaîne (dans la direction des feuilles). En pratique, nous ne construirons donc pas de développements pour c et nous utiliserons directement ceux de e.

8.2 Octree avec indirection

Nous introduisons ici une structure de données, *l'octree avec indirection*, qui est efficace pour les distributions uniformes et qui permet de représenter, dans le cas des distributions non uniformes, des octrees déséquilibrés avec des hauteurs supérieures à 10. Nous vérifierons par la suite que cette nouvelle structure de données ne génère pas de surcoût dans les accès aux cellules.

La structure d'octree avec indirection est basée sur les tableaux unidimensionnels de pointeurs présentés en section 2.3.1.2, et dans le cas des distributions uniformes elle se ramène à ces tableaux.

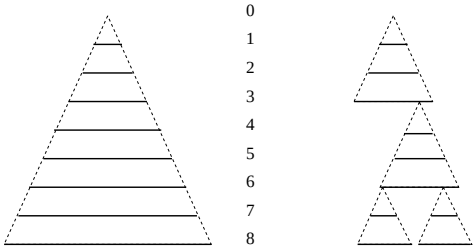
Algorithme 11 FMM adaptative en $3D$: **PHASE DE DESCENTE**

Pour $l = 2$ à H **faire**
  Pour $i = 1$ à 8^l **faire**
    Si $s_{\text{père de } i} \geq s^L_{min}$ **Alors**
      Si $s_i \geq s^L_{min}$ **Alors**
        Translater $\mathcal{L}_{l-1,i/8}$ du père de la cellule i en son centre grâce à l'opérateur *L2L*, et le stocker dans $\mathcal{L}_{l,i}$;
      Sinon
        Evaluer $\mathcal{L}_{l-1,i/8}$ en chacune des particules de la cellule i grâce à l'opérateur *L2P* ;
      Fin si
    Fin si
    Pour chacune des cellules j de la liste d'interaction de la cellule i **faire**
      Si $s_i \geq s^L_{min}$ **Alors**
        Si $s_j \geq s^M_{min}$ **Alors**
          Convertir $\mathcal{M}_{l,j}$ en un développement local basé sur le centre de la cellule i grâce à l'opérateur *M2L*, et l'additionner à $\mathcal{L}_{l,i}$;
        Sinon
          Construire au centre de la cellule i le développement local dû aux particules de la cellule j grâce à l'opérateur *P2L*, et l'additionner à $\mathcal{L}_{l,i}$;
        Fin si
      Sinon
        Si $s_j \geq s^M_{min}$ **Alors**
          Evaluer $\mathcal{M}_{l,j}$ en chacune des particules de la cellule i grâce à l'opérateur *M2P* ;
        Sinon
          Evaluer les interactions des particules de la cellule j sur les particules de la cellule i grâce à l'opérateur *P2P* ;
        Fin si
      Fin si
    Fin pour
  Fin pour
Fin pour

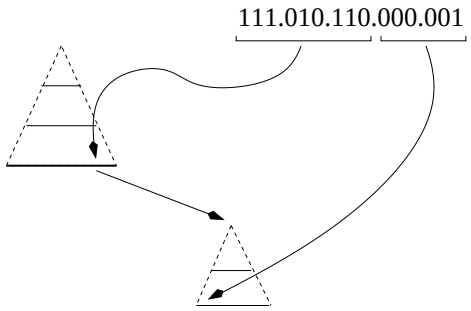
Algorithme 12 FMM adaptative en $3D$: **PHASE D'EVALUATION**

Pour $i = 1$ à 8^H **faire**
  Si $s_i \geq s^L_{min}$ **Alors**
    Pour chaque particule j localisée en z_j dans la cellule i du niveau H **faire**
      Evaluer $\mathcal{L}_{l,i}(z_j)$ grâce à l'opérateur *L2P*, et additionner le résultat au potentiel et à la force obtenus par le calcul direct ;
    Fin pour
  Fin si
Fin pour

Dans le cas des distributions non uniformes, l'octree déséquilibré est divisé en plusieurs sous-octrees de hauteur constante h. Cette hauteur h sera appelée dans la suite *seuil d'indirection*. Le premier sous-octree est enraciné à la racine de l'octree global. Si la hauteur totale de l'octree est plus grande que h, il y aura d'autres sous-octrees enracinés au niveau h : nous ne stockerons alors uniquement que les sous-octrees non vides. Si la hauteur totale est aussi plus grande que $2h$, d'autres sous-octrees seront utilisés au niveau $2h$ et ainsi de suite ; ceci est représenté en figure 8.1(a).



(a) Octree classique (à gauche) et octree avec indirection (à droite).



(b) Accès à la cellule d'indice de Morton 111.010.110.000.001 au niveau 5.

FIG. 8.1 – Octree avec indirection (seuil d'indirection $h = 3$). A chaque niveau sont représentés les tableaux unidimensionnels de pointeurs qui permettent d'accéder au contenu de n'importe quelle cellule.

A chacun de ces *niveaux d'indirection* h, $2h$, $3h$, ..., nous utilisons des tableaux unidimensionnels de pointeurs pour stocker les adresses des sous-octrees non vides : les sous-octrees sont indexés dans ces tableaux en fonction de l'indice de Morton de leur racine. L'intérêt d'une telle structure de données est que tous les tableaux sont désormais locaux aux sous-octrees, et leur taille ne dépassera par conséquent pas 8^h.

L'accès au contenu d'une cellule est décrit à la figure 8.1(b) et le nombre d'opérations requises est proportionnel au nombre d'indirections rencontrées. Nous allons montrer en section 8.4 que ce surcoût est indiscernable en pratique. Et pour les niveaux de l'octree inférieurs à h, notre structure d'octree avec indirection est en fait identique aux tableaux de pointeurs de la section 2.3.1.2.

Il faut noter que plusieurs fois au cours de l'algorithme de la FMM, nous allons devoir parcourir toutes les cellules de l'octree à un niveau donné : dans le cas d'octrees hautement déséquilibrés, nous allons alors devoir traverser de nombreuses cellules vides aux niveaux les plus profonds de l'octree, ces cellules vides étant de plus en plus nombreuses lorsqu'on approche du niveau des feuilles. Afin de limiter cette perte de temps CPU, nous construisons au préalable à chaque niveau de l'octree, une liste d'intervalles contenant les indices des cellules non vides. Seuls les indices de ces intervalles seront alors parcourus à chaque niveau dans l'algorithme de la FMM. Ces listes d'intervalles sont établies juste après la construction de l'octree (une fois que toutes les particules y ont été insérées) selon l'algorithme 13 : nous parcourons chaque niveau de l'octree depuis la racine jusqu'aux feuilles, et pour chaque cellule vide c rencontrée au niveau l, nous scindons en deux un intervalle au niveau $l + l_0$, en éliminant les descendants de c au niveau $l + l_0$. En pratique la valeur de l_0 est un compromis entre le coût de ce parcours et la finesse des intervalles contruits : $l_0 = 3$ est satisfaisante, et

$8^3 = 512$ descendants sont ainsi écartés pour chaque cellule vide.

Algorithme 13 Construction des intervalles écartant les cellules vides. H est la hauteur de l'octree et l_0 le paramètre permettant de régler la finesse et le coût de la construction de ces intervalles. Les indices des cellules sont issus de l'ordre de Morton ou de l'ordre de Hilbert, et nous sommes dans le cas d'un octree $3D$.

1: **Pour** $l = 0$ à $H - l_0$ **faire**
2: Initialiser les intervalles au niveau $l + l_0$ en tranposant les intervalles établis au niveau $l + l_0 - 1$.
3: **Pour** toute cellule c_k d'indice k dans les intervalles au niveau l **faire**
4: **Si** c_k est vide **Alors**
5: Scinder au niveau $l + l_0$ l'intervalle $[a, b]$ contenant les indices $8^{l_0}.k$ à $8^{l_0}.(k + 1) - 1$ en deux intervalles $[a, 8^{l_0}.k - 1]$ et $[8^{l_0}.(k + 1), b]$.
6: **Fin si**
7: **Fin pour**
8: **Fin pour**

8.3 Formulation BLAS

Dans le cas des distributions non uniformes de particules, la formulation BLAS du calcul *M2L* est élaborée à partir de celle présentée pour les distributions uniformes.

8.3.1 Cas d'une distribution quelconque de particules

Sans hypothèse sur la distribution des particules, le calcul *M2L* avec des routines BLAS peut être réalisé de deux façons différentes dans la FMM adaptative.

- Nous pouvons tout d'abord utiliser des routines BLAS de niveau 2 qui permettent de calculer chaque opération *M2L* sous la forme d'un produit matrice-vecteur, comme cela a été décrit en section 5.2.
- Mais nous pouvons aussi utiliser les routines BLAS de niveau 3, plus efficaces, en groupant plusieurs opérations *M2L* ensemble en un unique produit matrice-matrice comme cela a été décrit en section 5.3 (schéma avec recopies). Ceci implique cependant un surcoût dû aux copies des développements, vers et depuis ces matrices, qui doit être compensé par le gain en efficacité apporté par les BLAS de niveau 3. Plus précisément, nous reprenons l'algorithme retenu en section 5.3.1 pour les cellules avec liste d'interaction incomplète. La boucle extérieure porte sur les vecteurs *M2L*, alors que la boucle intérieure parcourt toutes les cellules qui correspondent au vecteur *M2L* courant. Le nombre de cellules étant ici potentiellement très grand, le produit matriciel est effectué à chaque fois que les matrices ont atteint un nombre de colonnes suffisant. Nous utilisons la valeur $NbExp_{max}$ (voir section 5.3.3) pour déterminer en pratique ce nombre de colonnes suffisant (aussi bien pour un noyau *M2L* de hauteur simple que pour un noyau *M2L* de hauteur double).

Nous verrons en section 8.4.3.1 que le choix entre ces deux possibiltés dépend de la valeur de P.

8.3.2 Détection des zones uniformes

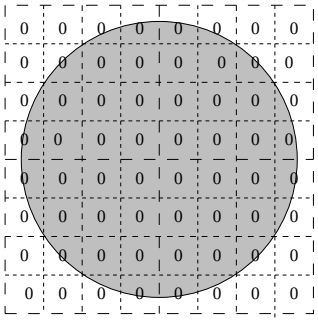
Cependant, sous certaines conditions sur la distribution des particules, nous pouvons améliorer l'efficacité du schéma avec les BLAS de niveau 3. Nous allons ainsi considérer des zones cubiques ou de forme rectangulaire (tel un parallélépipède rectangle en $3D$) où toutes les cellules ont à la fois un développement multipôle et un développement local : ces zones seront appelées *zones uniformes*. De telles zones cubiques correspondent en fait localement à des octrees uniformes (ou complets). Dans ces zones uniformes, nous n'avons pas à éviter les éventuelles cellules vides qui n'ont pas de développements, ou les cellules avec trop peu de particules pour avoir à la fois un développement multipôle et un développement local. Le regroupement des développements en matrices peut donc être simplifié, et ainsi accéléré. De plus, comme présenté en section 5.3.2 pour les octrees uniformes, le surcoût des copies peut même être évité dans ces zones : grâce aux stockages par *ligne* ou par *tranche*, nous pouvons en effet agencer les développements en mémoire de sorte que les matrices soient déjà construites. Comme montré en section 5.4.2, ceci réduit encore les temps CPU du schéma avec BLAS de niveau 3, en particulier pour les petites valeurs de P car les copies sont plus difficilement compensées dans ce cas.

Les meilleures efficacités BLAS étant obtenues pour des matrices avec un nombre suffisant de lignes et de colonnes, ceci requiert d'assez grandes zones uniformes, ce qui signifie un octree uniforme « assez haut » : en pratique, il faut au moins une hauteur d'octree uniforme supérieure ou égale à 5 pour obtenir des gains significatifs avec nos stockages de données spéciaux par *ligne* ou par *tranche*. Par conséquent, nous allons devoir détecter dans les distributions non uniformes les zones uniformes disponibles, et les marquer en fonction de leur taille, afin d'améliorer l'efficacité des BLAS à ces endroits. Ceci peut être aisément réalisé en s'appuyant sur la décomposition spatiale fournie par l'octree. En pratique, nous allons utiliser deux parcours de l'arbre.

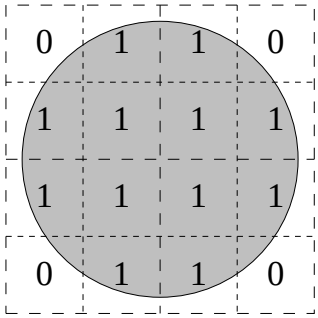
Le premier parcours est une remontée qui peut être effectuée en même temps que la phase de remontée de la FMM. Durant ce parcours, nous étiquetons les cellules dont les 8 fils ont leurs deux développements (multipôle et local). Plus précisément, l'étiquette utilisée pour une telle cellule c correspond à la hauteur du sous-octree uniforme enraciné en c. Les feuilles et les cellules internes dont certains fils n'ont pas leurs deux développements sont ainsi étiquetées par 0, alors que les cellules dont tous les fils ont leurs deux développements sont étiquetées par la plus petite étiquette parmi leurs fils, incrémentée de 1. Un exemple est donné en figure 8.2.

Remarque 8.1. *A cause des seuils mis en place pour la construction des développements multipôles et locaux en section 8.1.1, nous tolérons en pratique un quota de fils non vides auxquels il ne manquerait qu'un ou deux développements. Nous forçons alors la construction des développements de façon à obtenir une zone uniforme, le gain offert par les BLAS permettant de compenser la perte issue du non respect de s^M_{min} ou de s^L_{min}.*

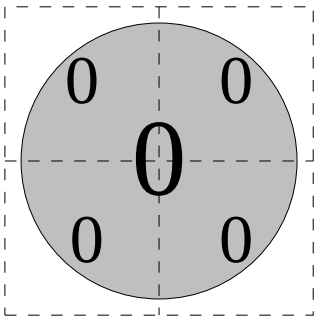
Après cette phase de remontée, nous avons seulement détecté les plus grandes zones uniformes qui correspondent à une cellule de l'octree. De plus grandes zones uniformes peuvent cependant exister : par exemple au centre de l'octree, parmi les 8 cellules qui n'ont que la racine comme ancêtre commum. C'est pourquoi nous parcourons ensuite l'octree de la racine jusqu'aux feuilles au cours d'une phase de descente similaire, mais non confondue, à celle de la FMM, en recherchant d'éventuels voisins parmi les zones uniformes déjà détectées, afin de les regrouper en zones plus grandes. La figure 8.2 en donne un exemple.



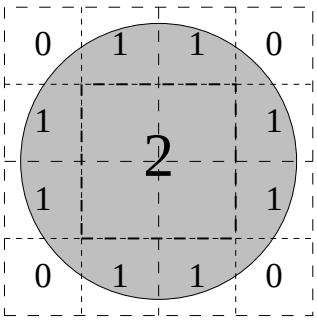
Remontée : étiquettes au niveau des feuilles ($l = 3$).



Remontée : étiquettes au niveau $l = 2$.



Remontée : étiquettes au niveaux $l = 1$ et $l = 0$.



Descente : étiquettes finales.

FIG. 8.2 – Détection des zones uniformes : phases de remontée et de descente avec une distribution $2D$ uniforme sous la forme d'un disque plein, et avec un octree de hauteur 3. La généralisation à une sphère $3D$ pleine est directe.

Après ce second parcours, la phase de descente de la version par BLAS de la FMM peut être effectuée en utilisant les zones uniformes détectées afin d'atteindre une meilleure efficacité dans les routines BLAS de niveau 3. En pratique, seules les zones uniformes avec une hauteur supérieure ou égale à 5 sont détectées et traitées spécifiquement (avec un stockage par *ligne* ou par *tranche*). Comme dans le cas uniforme, les cellules aux bords des zones uniformes sont quant à elles traitées spécialement : nous renvoyons ici au traitement des cellules avec liste d'interaction incomplète présenté à la section 5.3.1. Hors de ces zones uniformes suffisament grandes, nous utilisons les algorithmes présentés à la section précédente (section 8.3.1).

8.4 Validation experimentale

Afin de valider l'intérêt des seuils s^M_{min} et s^L_{min}, de l'octree avec indirection, et de la formulation BLAS pour la FMM adaptative, avec ou sans zones uniformes, nous allons considérer les distributions non uniformes suivantes.

1. Pour introduire un premier degré d'irrégularité par rapport au cas entièrement uniforme, nous utiliserons comme dans [102] une distribution un peu artificielle, où seuls 2 des 8 premiers octants sont uniformément remplis de particules, comme représenté en figure 8.7 (distribution dite des « deux octants ») ;

2. Puis nous considérerons d'autres distributions uniformes au sein d'un volume géométrique, comme une balle ou une barre, voir les figures 8.5 et 8.6 ;

3. Nous étudierons aussi des distributions uniformes sur une surface $2D$ dans un espace $3D$, comme le cylindre et la sphère représentés aux figures 8.3 et 8.4. Ce sont des exemples de référence utilisés dans [34], [93] et [127]. A noter dans le cas de la sphère que la concentration des particules est plus élevée au niveau des pôles : ceci est logiquement dû à la distribution aléatoire sur les angles θ et ϕ des coordonnées sphériques ;

4. Enfin, nous allons aussi considérer des exemples classiques de distributions gravitationnelles. Le premier est un modèle de galaxie de Plummer (voir [5]), où la densité est très importante près du centre de la galaxie mais décroît au fur et à mesure que l'on s'éloigne de ce centre : voir figure 8.9. Le second exemple, nommé *hcg027* et présenté par E. Athanassoula dans [14], est un cas test spécifique composé de $1,5\ 10^6$ particules groupées en 50 galaxies : une capture d'écran est donnée[2] en figure 8.8. Ces simulations gravitationnelles ne nécessitant que des basses précisions, seuls $P = 2$ ou $P = 3$ seront utilisés dans ces cas.

Sauf indication contraire, tous les tests ont été réalisés sur un processeur IBM Power3-II WH2+ décrit en section 5.4. En ce qui concerne le choix de la hauteur du noyau *M2L*, nous utilisons ici le noyau *M2L* de hauteur double, sauf pour les petites valeurs de P ($P = 2$ ou $P = 3$) où le noyau de hauteur simple est plus efficace (voir section 6.5).

[2]Les captures d'écran présentées pour le modèle de Plummer et pour le cas *hcg027* ont été obtenues grâce au logiciel *Glnemo* (version 0.7) écrit par Jean-Charles Lambert : `Jean-Charles.Lambert@oamp.fr`.

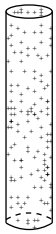


FIG. 8.3 – Cylindre.

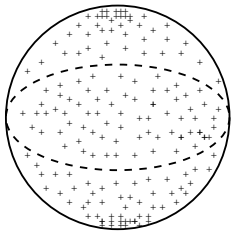


FIG. 8.4 – Sphère.

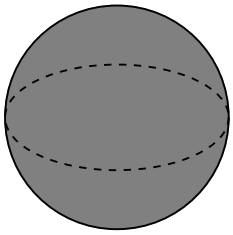


FIG. 8.5 – Balle.

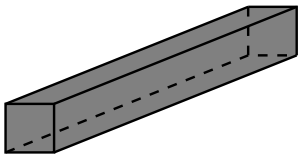


FIG. 8.6 – Barre.

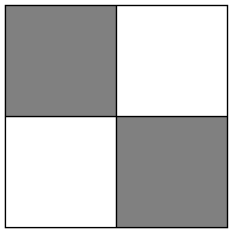


FIG. 8.7 – Deux octants.

FIG. 8.8 – *hcg027*.

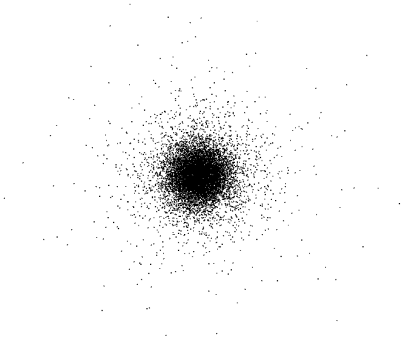


FIG. 8.9 – Modèle de Plummer.

8.4.1 Seuils pour les développements multipôles et locaux

En pratique, les coûts $\mathcal{C}_{M2L}(P)$, $\mathcal{C}_{P2L}(P, s_m)$, $\mathcal{C}_{M2P}(P, s_l)$, $\mathcal{C}_{P2P}(s_m, s_l)$ qui apparaissent dans les équations (8.1), (8.2), (8.3) et (8.4), sont déterminés à partir des temps CPU mesurés (et moyennés sur une centaine d'exécutions) pour chaque valeur de P, s_m et s_l. Grâce à ces coûts, et grâce au processus d'évaluation basé sur les équations (8.1) (8.2) (8.3) (8.4), nous pouvons estimer les meilleures valeurs pour s^M_{min} et s^L_{min}.

Dans le tableau 8.3, nous comparons alors nos estimations avec les valeurs effectivement optimales : ces valeurs optimales ont été déterminées en comparant les temps CPU d'une exécution totale de la FMM pour toutes les valeurs possibles de s^M_{min} et s^L_{min}. Seul le calcul *M2L* classique est utilisé ici. Nous pouvons tout d'abord remarquer que pour diverses distributions non uniformes[3], ces valeurs optimales confirment la pertinence de la distinction entre s^M_{min} ets^L_{min}. Nos valeurs estimées sont très proches des valeurs optimales, et lorsqu'elles diffèrent, la différence sur le temps CPU de la FMM totale est elle insignifiante. Plus précisément, ces résultats indiquent qu'il y a en fait un intervalle de valeurs autour de chaque valeur optimale, pour s^M_{min} et s^L_{min}, à l'intérieur duquel les temps CPU sont très proches du temps optimal. Les tailles de ces intervalles augmentent pour des P croissants, mais hors de ces intervalles les temps CPU peuvent se détériorer fortement : le tableau 8.3 montre bien que notre processus d'évaluation donne toujours des valeurs qui sont effectivement dans ces intervalles.

	N	P	Estimés		Optimaux		Temps CPU total	Perte / optimal
			s^M_{min}	s^L_{min}	s^M_{min}	s^L_{min}		
Sphère	10^5	3	1	1	2	1	10	0,5 %
Sphère	10^5	7	13	7	16	9	30	0,4 %
Sphère	10^5	15	68	30	80	40	87	0,3 %
Cylindre	10^5	3	1	1	2	1	7	1,5 %
Cylindre	10^5	7	13	7	17	9	22	0,2 %
Cylindre	10^5	15	68	30	50	24	71	0,1 %
Plummer	10^4	2	1	1	2	2	2	0,2 %
hcg027	$1,5\ 10^6$	2	1	1	2	2	397	0,2 %

TAB. 8.3 – Valeurs estimées contre valeurs optimales pour les seuils s^M_{min} et s^L_{min} avec diverses distributions non uniformes de N particules. Nous utilisons ici le calcul *M2L* classique. Nous donnons aussi les temps CPU (en secondes) d'une exécution totale de la FMM avec les valeurs estimées de s^M_{min} et s^L_{min}, et la perte par rapport au temps CPU avec les valeurs optimales.

Pour le calcul *M2L* avec les BLAS de niveau 2, les valeurs estimées de s^M_{min} et de s^L_{min} (qui seront aussi utilisées pour les BLAS de niveau 3) sont plus petites que celles présentées au tableau 8.3 puisque le calcul *M2L* est alors généralement plus rapide (voir les équations (8.1) et (8.2)), mais notre processus d'évaluation est aussi satisfaisant : voir le tableau 8.4 (les tests avec $P = 2$ ou $P = 3$ ne sont pas présentés car les BLAS n'accélèrent pas le calcul *M2L* dans ce cas comme nous le verrons en section 8.4.3.1). Il est plus difficile d'obtenir des estimations

[3]Les distributions non uniformes comme la balle, la barre ou les deux octants, ne sont pas étudiées ici car elles sont composées soit d'espaces vides, soit de zones remplies uniformément de particules. Comme les distributions uniformes, leur seul paramètre significatif est alors la hauteur de l'octree.

précises pour les routines BLAS car leurs performances dépendent de l'état des caches et de la mémoire au moment de l'appel BLAS : cet état est bien sûr différent lors des mesures (pour déterminer s^M_{min} et s^L_{min}) et lors d'une exécution de notre code FMB. De plus, le cas de la sphère est plus compliqué du fait de la concentration des particules au niveau des pôles. Pour le calcul *M2L* avec FFT par blocs ou avec rotations, les valeurs estimées de s^M_{min} et de s^L_{min} sont aussi plus petites pour la même raison.

	N	P	Estimés		Optimaux		Temps CPU total	Perte / optimal
			s^M_{min}	s^L_{min}	s^M_{min}	s^L_{min}		
Sphère	10^5	7	2	1	6	3	20	3,1 %
Sphère	10^5	15	14	6	30	16	65	4,8 %
Cylindre	10^5	7	2	1	2	2	15	0,7 %
Cylindre	10^5	15	14	6	16	15	55	0,1 %

TAB. 8.4 – Valeurs estimées contre valeurs optimales pour les seuils s^M_{min} et s^L_{min} avec diverses distributions non uniformes de N particules. Nous utilisons ici le schéma de calcul *M2L* avec les routines BLAS de niveau 2. Nous donnons aussi les temps CPU (en secondes) d'une exécution totale de la FMM avec les valeurs estimées de s^M_{min} et s^L_{min}, et la perte par rapport au temps CPU avec les valeurs optimales.

8.4.2 Avantages de l'octree avec indirection

Nous devons maintenant vérifier que l'octree avec indirection présenté en section 8.2 offre des gains conséquents en termes de consommation mémoire, tout en ne ralentissant pas les temps d'accès aux cellules. Ceci est effectué au tableau 8.5 pour différentes distributions non uniformes[4] et avec un seuil d'indirection $h = 6$, ce qui est une valeur appropriée en pratique puisque le nombre d'indirections est au plus 2 (pour des hauteurs d'octree allant jusqu'à 17) alors que la taille des sous-octrees reste modérée.

Les gains mémoire sont ainsi considérables et la perte en temps CPU est négligeable. On rappelle qu'avec des valeurs croissantes de P, le coût de calcul de la FMM augmente fortement, et l'éventuel surcoût dans les temps d'accès devient donc encore moins discernable. De plus, des distributions hautement non uniformes, comme *hcg027*, ne peuvent tout simplement pas être simulées sans l'octree avec indirection.

8.4.3 Formulation BLAS

Nous allons maintenant présenter les résultats de l'utilisation des routines BLAS avec des distributions non uniformes.

8.4.3.1 Sans zones uniformes

Comme montré dans le tableau 8.6, la formulation BLAS de la FMM dans le cas non uniforme, et sans zone uniforme disponible, n'est pas toujours plus rapide que le calcul *M2L* classique. Plus précisément, on a les comportements suivants en fonction des valeurs de P.

[4]Les distributions du type balle, barre ou deux octants ne sont pas présentées ici car les hauteurs d'octree nécessaires sont modestes et ne nécessitent donc pas d'octree avec indirection.

	N	P	Hauteur de l'octree H	Octree original ($h = H$)	Octree avec indirection ($h = 6$)	Δt
Cylindre	$2\ 10^6$	3	9	593 Mo	12 Mo	0,3 %
Cylindre	$3\ 10^6$	7	8	77 Mo	5 Mo	0,0 %
Sphère	$2\ 10^6$	3	8	89 Mo	20 Mo	0,2 %
Plummer	10^5	2	9	588 Mo	11 Mo	1 %
hcg027	$1,5\ 10^6$	2	13	-	326 Mo	-

TAB. 8.5 – Besoins mémoire et différence dans les temps CPU, pour l'octree original (sans indirection) et pour l'octree avec indirection. Δt donne le surcoût pour l'octree avec indirection sur l'octree original et est calculé comme : *(CPU Time(h=6) - CPU Time(h=H))/CPU Time(h=H)*. La simulation *hcg027* ne peut pas être exécutée sur les 2 Go disponibles avec l'octree original car la hauteur d'octree nécessaire est 13, ce qui impliquerait des besoins supérieurs à 2000 Go avec une telle structure de données.

- Pour les petites valeurs de P ($P = 2$ ou $P = 3$), le calcul classique est plus rapide.
- Pour les valeurs intermédiaires de P, le recours aux BLAS de niveau 2 est le plus efficace ($P = 5$), mais au fur et à mesure que P augmente, le schéma avec les BLAS de niveau 3 devient pareillement efficace ($P = 7$).
- Et pour les valeurs de P supérieures ($P = 10$ et $P = 15$ ici), les BLAS de niveau 3 surpassent ceux de niveau 2 en étant jusqu'à 5 fois plus rapides que le calcul *M2L* classique.

Dans le cas de simulations gravitationnelles avec de petites valeurs de P, le calcul *M2L* classique est aussi le plus rapide. Ceci s'explique par l'influence de P sur l'efficacité des BLAS : pour des valeurs de P trop petites, les matrices correspondantes sont trop petites pour que le calcul qui leur est associé soit efficacement accéléré par les routines BLAS de niveau 2. De plus, comme détaillé en section 8.3, les copies supplémentaires requises par les BLAS de niveau 3 sont particulièrement coûteuses pour les petites valeurs de P, et l'irrégularité de la distribution des développements dans l'octree empêche ici tout stockage de données spécial (par *ligne* ou *tranche*).

8.4.3.2 Détection des zones uniformes

Cependant, lorsque des zones uniformes sont disponibles dans les distributions non uniformes, nous pouvons les détecter et les exploiter (voir section 8.3) afin d'améliorer l'efficacité des BLAS de niveau 3 ; cela est montré dans le tableau 8.7.

Sur l'exemple avec les deux octants pleins, la phase de remontée de l'algorithme de détection des zones uniformes découvre toutes les régions uniformes disponibles puisque celles-ci correspondent à la décomposition spatiale de l'octree. Mais pour les deux autres exemples, la balle et la barre, la phase de descente est nécessaire pour détecter les plus grandes zones existantes. Dans le cas des petites valeurs de P, il peut être plus rapide d'utiliser des BLAS de niveau 3 à l'intérieur des zones uniformes, et d'avoir recours aux BLAS de niveau 2 ou au calcul *M2L* classique hors de ces zones uniformes (voir section précédente) : c'est ici le cas avec $P = 3$.

Pour toutes ces distributions, la détection des zones uniformes améliore clairement l'efficacité des BLAS de niveau 3 avec un gain pouvant atteindre 50%. De plus, le gain augmente généralement avec des valeurs décroissantes de P : ceci améliore donc les résultats de la

	N	P	H	*M2L* classique	*M2L* avec BLAS de niveau 2	*M2L* avec BLAS de niveau 3
Sphère	10^6	3	8	**52**	69	128
Sphère	10^6	5	7	108	**45**	67
Sphère	10^6	7	7	254	**125**	**127**
Sphère	10^6	10	6	248	135	**91**
Sphère	10^6	15	6	794	517	**309**
Cylindre	10^6	3	9	**22**	29	51
Cylindre	10^6	5	8	43	**14**	20
Cylindre	10^6	7	8	118	**35**	**36**
Cylindre	10^6	10	7	97	36	**21**
Cylindre	10^6	15	7	387	160	**78**
Plummer	10^6	2	10	**141**	244	783
hcg027	$1,5\ 10^6$	2	13	**117**	196	862

TAB. 8.6 – Temps CPU de la phase de descente (en secondes) pour le calcul *M2L* classique (sans BLAS), avec les BLAS de niveau 2 et avec les BLAS de niveau 3. Les meilleurs temps sont donnés en **gras**.

section précédente où les schémas par BLAS n'étaient pas efficaces pour les petites valeurs de P. En effet, comme indiqué en section 8.3, notre détection permet d'éviter les copies requises par les BLAS de niveau 3 ; or ces copies sont justement plus coûteuses pour les petites valeurs de P. Et pour les grandes valeurs de P, les routines BLAS de niveau 3 étaient déjà les plus rapides sans détection des zones uniformes.

	N	P	*M2L* classique	BLAS de niveau 3		
				Sans détection ZU	Avec détection ZU	Gain avec détection ZU
Deux Octants	$3\ 10^6$	7	1276	247	204	17,4 %
Balle	$2\ 10^6$	3	83	102	73†	12 %
Balle	$2\ 10^6$	5	759	216	196	9,3 %
Barre	$3\ 10^6$	3	101	107	51	49,5 %
Barre	$3\ 10^6$	5	928	243	161	33,7 %
Barre	$3\ 10^6$	7	2650	504	399	20,8 %

TAB. 8.7 – Temps de la phase de descente (en secondes) pour le calcul *M2L* classique (sans BLAS), et pour le calcul *M2L* avec BLAS de niveau 3, avec et sans détection des zones uniformes (ZU). Les temps marqués d'un † indiquent l'utilisation du calcul *M2L* classique hors des zones uniformes. Le gain offert par la détection des zones uniformes est toujours calculé par rapport au meilleur des deux temps sans détection (*M2L* classique et BLAS sans détection).

Remarque 8.2. *Les distributions gravitationnelles, comme le modèle de Plummer et* hcg027*, offrent aussi des zones uniformes, mais celles-ci sont trop petites pour être efficacement exploitées. Il faut aussi noter que la détection des zones uniformes dans ces distributions gravitationnelles est plus*

compliquée que pour les distributions présentées au tableau 8.7. La fonction de densité des particules peut en effet varier fortement dans le premier cas : si une zone uniforme est détectée au niveau l, il n'est alors pas garanti que ses descendants à un niveau $l' > l$ forment aussi une zone uniforme, et il est alors plus compliqué de regrouper des zones uniformes lors de la phase de descente. Par contre, dans le second cas, la fonction de densité est soit constante, soit nulle, ce qui implique qu'une zone uniforme à un niveau donné s'étend toujours jusqu'au niveau des feuilles (si la hauteur de l'octree est correctement choisie) : le regroupement des zones uniformes est alors aisé.

8.4.4 Efficacité du traitement spécial des cellules non adaptatives

Dans les tests effectués jusqu'à présent, nous avons toujours utilisé le traitement spécial des cellules non adaptatives, présenté en section 2.3.3.9. En réalisant ces tests, nous avons cependant constaté un résultat a priori surprenant : ce traitement allonge les temps de calculs. Le tableau 8.8 reprend ainsi les tests présentés au tableau 8.6 de la section 8.4.3.1 en traitant les cellules non adaptatives (à savoir, les cellules à l'intérieur d'une chaîne) comme les autres cellules. Le cas du cylindre présentant en fait peu de cellules non adaptatives, peu de différences sont observées dans ce cas. Par contre pour la sphère, où les cellules non adaptatives sont plus nombreuses, les gains sont importants si on renonce à traiter spécialement ces cellules non adaptatives.

	N	P	H	*M2L* classique	*M2L* avec BLAS de niveau 2	*M2L* avec BLAS de niveau 3
Sphère	10^6	3	8	**49**	62	118
Sphère	10^6	5	7	107	**30**	51
Sphère	10^6	7	7	254	**89**	**87**
Sphère	10^6	10	6	248	107	**58**
Sphère	10^6	15	6	795	420	**206**
Cylindre	10^6	3	9	**22**	27	51
Cylindre	10^6	5	8	43	**11**	18
Cylindre	10^6	7	8	119	**32**	**32**
Cylindre	10^6	10	7	97	36	**21**
Cylindre	10^6	15	7	388	160	**77**

TAB. 8.8 – Temps CPU de la phase de descente (en secondes) pour le calcul *M2L* classique (sans BLAS), avec les BLAS de niveau 2 et avec les BLAS de niveau 3. Les cellules non adaptatives ne sont pas traitées spécialement. Les meilleurs temps sont donnés en **gras**.

Ceci s'explique par le précalcul des fonctions de transfert *M2L* présenté en section 2. Ce précalcul n'est en effet effectué que pour les vecteurs *M2L* de la liste d'interaction originale de la FMM : si une cellule c (source ou cible) n'est pas une cellule adaptative, on utilise le développement (multipôle ou local suivant le cas) de son descendant adaptatif. Le centre du développement n'étant plus celui de c, le vecteur *M2L* ne correspond plus à l'ensemble des fonctions de transfert *M2L* précalculées. Il faut donc calculer spécialement, pour chaque opération *M2L* faisant intervenir une cellule non adaptative, la fonction de transfert *M2L* correspondante. Et cette fonction de transfert sera alors calculée plusieurs fois si le même vecteur se représente. Ce surcôut n'est pas forcément compensé par les *M2M* et les *L2L* évités pour les cellules non adaptatives, d'autant plus qu'il y a potentiellement 189 opérations *M2L*

pour 1 opération *M2M* et 1 opération *L2L*. Le phénomène est flagrant pour le schéma de calcul avec BLAS, car les matrices de transfert *M2L* sont plus coûteuses à calculer que les simples fonctions de transfert *M2L* utilisées dans le calcul *M2L* classique. Mais ceci reste aussi sensible dans le cas du calcul *M2L* classique, comme le montre le tableau 8.9 dans le cas des distributions gravitationnelles : là encore il vaut mieux éviter de traiter spécialement les cellules non adaptatives.

	N	P	*M2L* classique	
			Avec cellules non adaptatives	Sans cellule non adaptatives
Plummer	10^6	2	141	111
hcg027	$1,5\ 10^6$	2	117	90

TAB. 8.9 – Temps CPU de la phase de descente (en secondes) pour des distributions gravitationnelles avec le calcul *M2L* classique. Les cellules non adaptatives ne sont pas traitées spécialement.

On pourrait cependant concevoir d'étendre la structure de données contenant les fonctions de transfert *M2L* précalculées aux vecteurs *M2L* utilisés avec les cellules non adaptatives. Mais une telle structure devrait permettre d'accéder potentiellement à un nombre exponentiel d'entrées : alors que sans les cellules non adaptatives le nombre de fonctions de transfert est 316 (voir section 2), ce nombre est multiplié par 8^2 (cellules sources et cellules cibles) à chaque niveau supplémentaire pris en compte lorsqu'on considère les cellules non adaptatives potentielles. Ce qui donne pour des chaînes contenant respectivement 1, 2 et 3 cellules non adaptatives : 20 224, 1 294 336 et 82 837 504. Par ailleurs il semble que les distributions que nous étudions ne présentent pas suffisament de cellules non adaptatives : le cas du cylindre ne présente que 1% de cellules non adaptatives, et la sphère 4,5%. Et même dans le cas des distributions gravitationnelles qui présentent plus de cellules non adaptatives (jusqu'à 30%), les gains, alors plus importants au niveau des opérations *M2M* et *L2L*, restent insignifiants une fois ramenés au niveau du temps de calcul global, comme le montre le tableau 8.10. Même avec un précalcul des fonctions de transfert dans le cas des cellules non adaptatives, les gains escomptables au final sont bien trop faibles.

	N	P	H	% cellules non adaptatives	Gain (%)		
					M2M	*L2L*	Temps total
Sphère	10^6	3	8	4,5	12,7	4,5	0,15
Plummer	10^6	3	10	28,1	36,4	24,6	0,36
hcg027	$1,5\ 10^6$	2	13	26,3	9,3	5,9	0,02

TAB. 8.10 – Pourcentage de cellules non adaptatives par rapport au nombre total de cellules non vides pour diverses distributions et gains correspondants (en pourcentage du temps CPU) au niveau des opérations *M2M* et *L2L*, ainsi qu'au niveau du temps total. Mesures de temps CPU réalisées sur le PC Linux décrit en section 7.2, à l'aide de l'outil de profilage `gprof` (version 2.16.91).

On préfère donc dans la suite, en particulier pour l'étude des simulations astrophysiques et pour la parallélisation de l'algorithme, renoncer à utiliser les cellules non adaptatives : la complexité linéaire de l'algorithme n'est plus garantie sur le plan théorique (voir section

2.3.3.9), mais les performances sont en pratique meilleures grâce au précalcul des fonctions de transfert.

8.5 Comparaison avec les autres schémas de calcul *M2L*

Maintenant que nous avons établi notre schéma de calcul *M2L* par BLAS dans le cas non uniforme, nous nous proposons de le comparer aux autres schémas de calcul *M2L* présentés à la partie II.

Dans les distributions présentant des zones uniformes, comme la balle, la barre ou les deux octants, la majeure partie du temps de calcul correspond à ces zones uniformes : la mise en place de la détection des zones uniformes nous ramène alors au cas de la FMM uniforme, et les mêmes conclusions qu'à la partie II peuvent être tirées concernant l'intérêt de notre version par BLAS. Nous nous concentrons donc sur les distributions ne présentant pas de zones uniformes, afin de vérifier l'efficacité des BLAS dans ce cas. Les faibles valeurs de P utilisées dans les simulations astrophyiques ($P = 2$ ou $P = 3$ avec un noyau *M2L* de hauteur simple) ne bénéficiant pas ou peu des schémas de calcul par FFT, BLAS (voir section 8.4.3.1), rotations ou ondes planes, les distributions correspondantes ne sont pas étudiées ici : nous reviendrons sur ce point en section 8.6.

Nous nous intéressons donc au cas du cylindre.

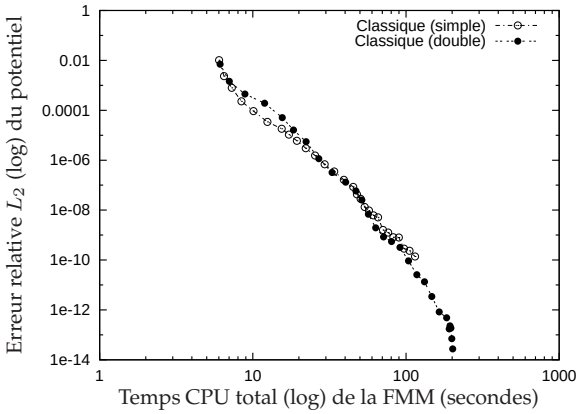
8.5.1 Comparaison avec le calcul classique, la FFT et les rotations

Nous présentons tout d'abord une comparaison des différents schémas de calcul implémentés dans notre code FMB : le calcul *M2L* classique, le calcul *M2L* avec une FFT[5] par blocs, le calcul *M2L* par rotations et le calcul *M2L* avec les routines BLAS.

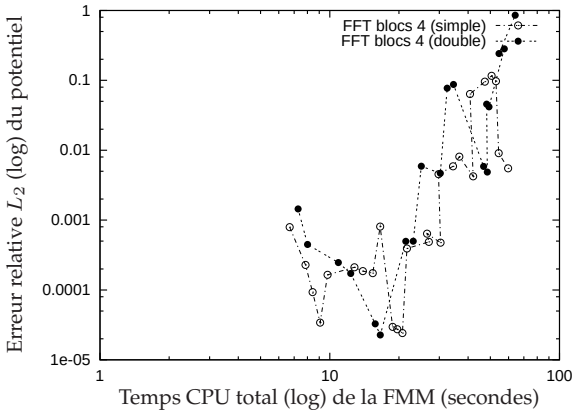
De la même façon qu'en section 6.5, nous déterminons en premier lieu, grâce aux résultats d'expérimentations reportées aux figures 8.10 et 8.11, la hauteur de noyau *M2L* la plus efficace en fonction de P et du schéma de calcul *M2L* utilisé. En ce qui concerne le schéma avec BLAS, nous nous basons sur les résultats de la section 8.4.3.1 et nous n'utilisons les routines BLAS de niveau 3 (c'est-à-dire le schéma avec recopies pour les zones non uniformes présenté en section 8.3.1) que pour les plus grandes valeurs de P ($P \geq 7$) ; le schéma de calcul avec les BLAS de niveau 2 (présenté lui aussi en section 8.3.1) est utilisé pour les valeurs de P inférieures. Et pour les toutes premières valeurs de P ($P \leq 3$), c'est même le schéma de calcul *M2L* classique qui est utilisé, car il est alors plus rapide (voir section 8.4.3.1). Comme pour les distributions uniformes (voir section 6.5), le noyau de hauteur simple est plus efficace que celui de hauteur double pour les petites valeurs de P dans le cas du calcul *M2L* classique (figure 8.10(a)) et du calcul *M2L* par rotations (figure 8.10(c)). Dans le cas de la FFT (figure 8.10(b)), le noyau de hauteur simple est toujours plus efficace, alors que pour les BLAS (figure 8.11) le noyau de hauteur double (et ses matrices pleines, plus faciles à traiter que les matrices creuses du noyau de hauteur simple) est aussi rapide que celui de hauteur simple pour les petites valeurs de P, et plus rapide pour les plus grandes valeurs.

D'autre part, les instabilités numériques de la FFT, même par blocs, apparaissent à la figure 8.10(b) dès $P = 7$ (c'est-à-dire bien plus tôt qu'en section 4.2.6), rendant le schéma

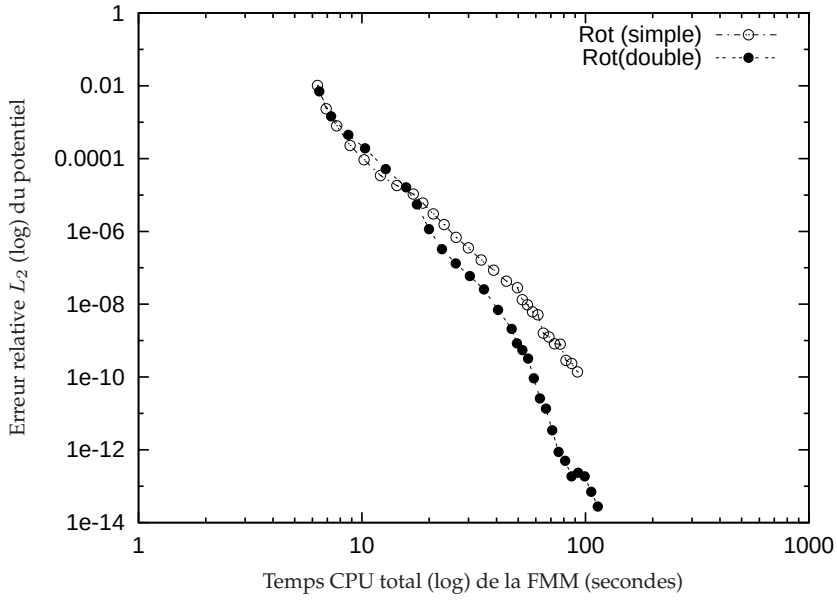
[5] Dans le cas de la FFT, les seuils s_{min}^M et s_{min}^L sont fixés à 1 car l'opérateur *M2P* n'a pas été mis en place pour les développements multipôles dans le domaine de Fourier : ceci amènerait en effet un surcoût non négligeable, soit en espace mémoire, soit en temps de calcul.



(a) Classique.



(b) FFT avec blocs de taille 4.



(c) Rotations.

FIG. 8.10 – Relation entre les erreurs obtenues en pratique et les temps CPU avec des noyaux de hauteurs simple et double pour un cylindre de 100 000 particules. L'erreur mesurée ici est l'erreur L_2, définie en section 7.2, qui est calculée pour un échantillon de 1000 particules. Les valeurs de P affichées ici pour les deux hauteurs de noyaux vont de 1 à 28 par pas de 1 (sauf pour la FFT par blocs de taille 4 où la première valeur est $P = 3$ à cause de la taille des blocs). Pour chaque valeur de P, la hauteur de l'octree utilisée est celle qui minimise le temps de calcul (en équilibrant le coût de calcul du champ proche et celui du champ lointain). Tests effectués sur un processeur IBM Power3-II WH2+ décrit en section 5.4.

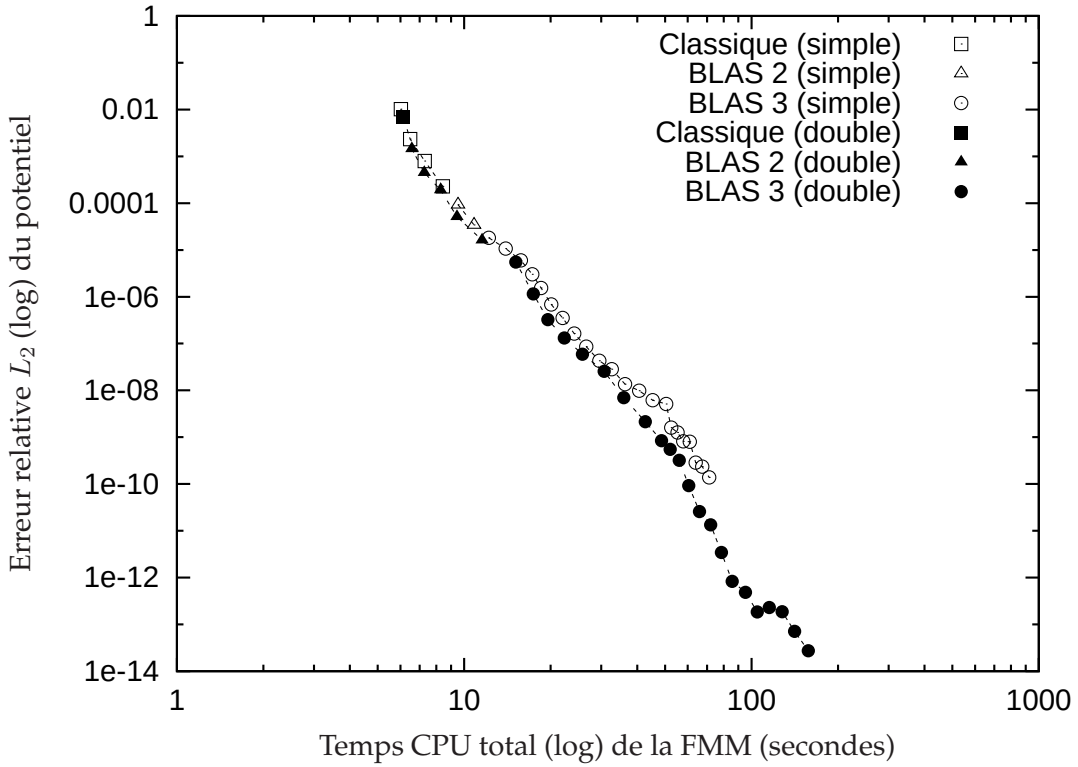


FIG. 8.11 – Relation entre les erreurs obtenues en pratique et les temps CPU avec le schéma par BLAS pour un cylindre de 100 000 particules. Tests réalisés dans les mêmes conditions qu'à la figure 8.10.

de calcul *M2L* quasiment inutilisable pour la FMM adaptative : ceci est dû à la hauteur de l'octree qui est plus importante dans le cas des distributions non uniformes que dans le cas des distributions uniformes (hauteur 7 pour les premières valeurs de P dans le cas de la figure 8.10(b)).

En sélectionnant à chaque fois la hauteur de noyau la plus efficace, nous pouvons comparer à la figure 8.12 les différents schémas de calcul. Alors que le schéma de calcul classique est rapidement trop coûteux, et que celui avec FFT est rapidement instable, le schéma de calcul par BLAS est bien le plus rapide, sauf pour les grandes valeurs de P où il est supplanté par le calcul avec rotations, grâce à la plus faible complexité théorique de ce dernier (en $\mathcal{O}(P^3)$). Mais pour les précisions qui nous intéressent en dynamique moléculaire (entre 10^{-5} et 10^{-7}) c'est bien le schéma de calcul par BLAS qui est le plus rapide. De plus, si l'on grossit la taille de la distribution, comme en figure 8.13, le schéma par BLAS est encore plus efficace et le point de *cross-over* entre les rotations et les routines BLAS a tendance à reculer, le grain de calcul étant plus important pour les produit matriciels de la formulation par BLAS (routines BLAS de niveau 3).

8.5.2 Comparaison avec les ondes planes

De la même façon qu'en section 7.2 et sur le même PC Linux, nous allons comparer le schéma de calcul par BLAS au schéma de calcul avec ondes planes à l'aide de notre code

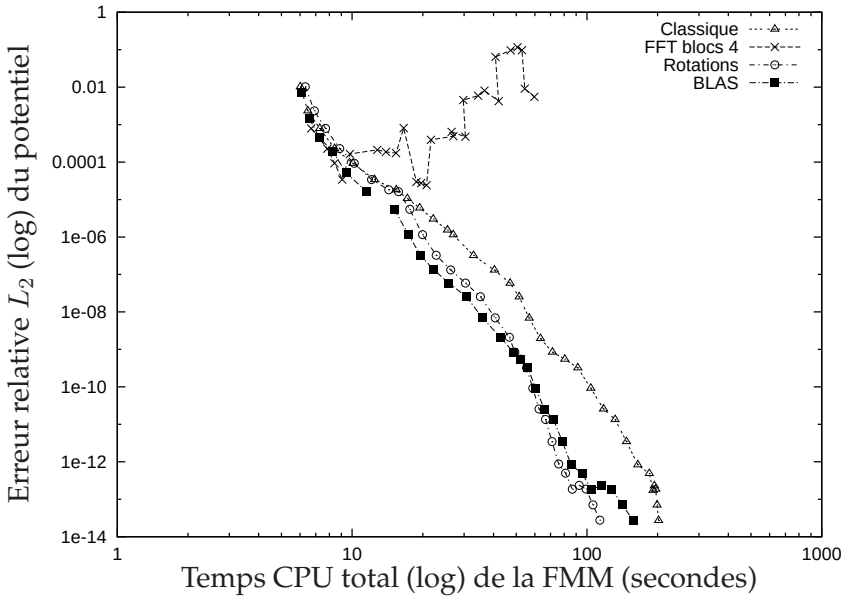


FIG. 8.12 – Relation entre les erreurs obtenues en pratique et les temps CPU avec une hauteur de noyau *M2L* optimale (d'après les figures 8.10 et 8.11) pour un cylindre de 100 000 particules.

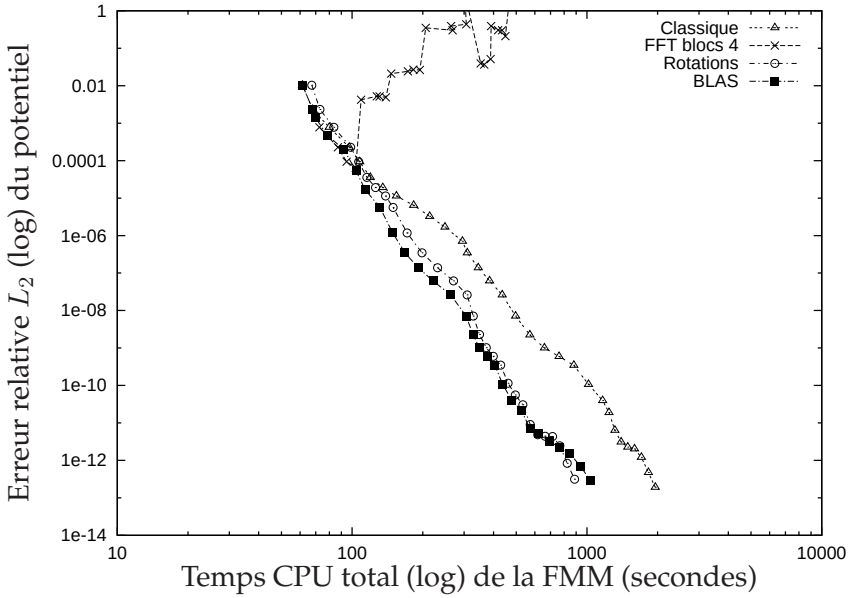
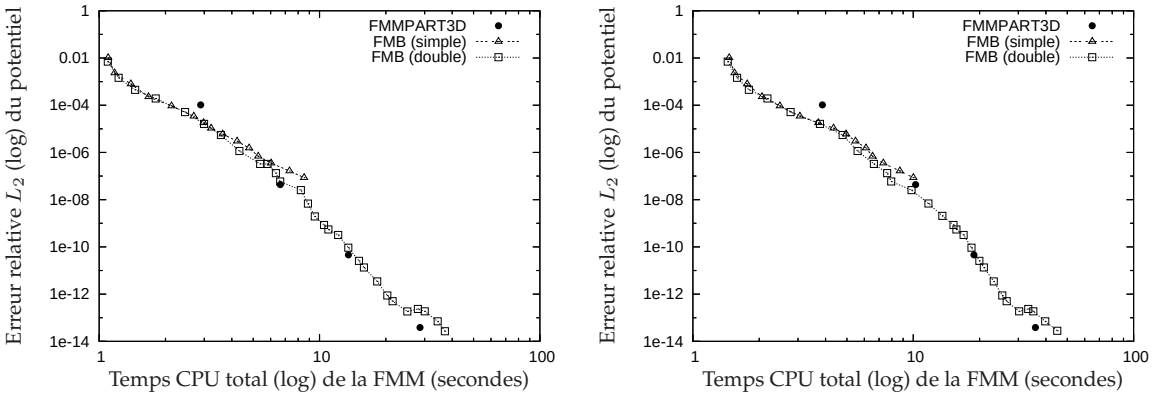
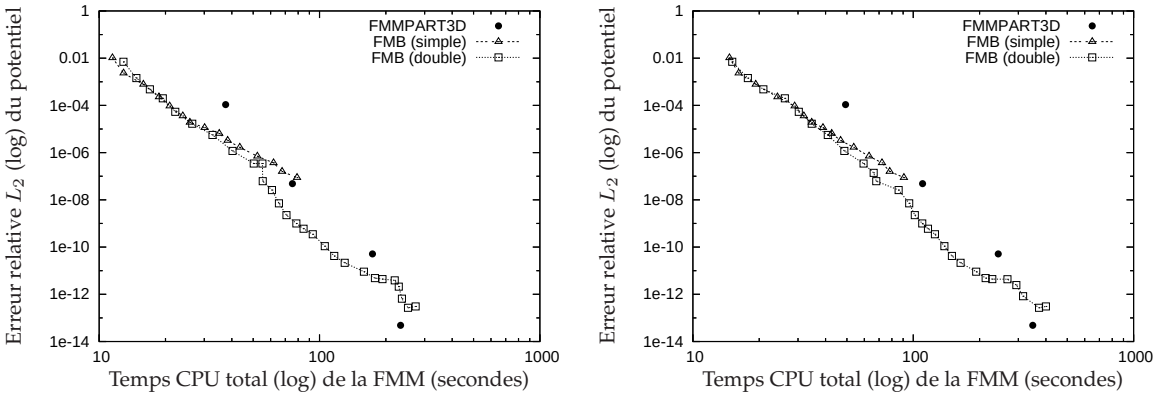


FIG. 8.13 – Relation entre les erreurs obtenues en pratique et les temps CPU avec une hauteur de noyau *M2L* optimale (de la même façon qu'en figure 8.12) pour un cylindre de 1 million de particules. Les valeurs de P sont ici comprises entre 1 et 27.



(a) Potentiel uniquement. (b) Force et potentiel.

FIG. 8.14 – Comparaison entre FMMPART3D et FMB pour un cylindre de 100 000 particules.



(a) Potentiel uniquement. (b) Force et potentiel.

FIG. 8.15 – Comparaison entre FMMPART3D et FMB pour un cylindre d'un million de particules.

FMB et du code FMMPART3D. Le schéma de calcul par BLAS est le même que celui utilisé à la section 8.5.1 : BLAS de niveau 2 (voire calcul classique) pour les petites valeurs de P et BLAS de niveau 3 pour les autres valeurs de P. Ces tests, réalisés dans les mêmes conditions que pour les distributions uniformes (voir section 7.2), sont présentés aux figures 8.14 et 8.15.

Les mêmes conclusions peuvent être tirées que dans le cas uniforme. Notre code est ainsi plus efficace lorsque le potentiel et la force sont calculés, et lorsque seul le potentiel est calculé, FMB est au moins aussi rapide que FMMPART3D pour les trois premières précisions de FMMPART3D, et est toujours plus rapide pour la première précision. Le gain en complexité théorique apporté par les ondes planes n'est donc significatif que pour les plus grandes valeurs de P. Il faut aussi remarquer que, comme pour les distributions uniformes et comme remarqué lors des comparaisons avec les rotations (voir section précédente), lorsque la taille de la distribution augmente, le calcul par BLAS devient sensiblement plus efficace, le grain de calcul étant plus important pour les routines BLAS de niveau 3. Pour les trois premières précisions de FMMPART3D, et pour une précision supérieure ou égale dans FMB, on obtient

alors, dans le cas du cylindre d'un million de particules, des gains conséquents de (respectivement) 44,3%, 19,2% et 32,9% (cas où seul le potentiel est calculé).

Pour la gamme de précision requise en dynamique moléculaire (10^{-5} à 10^{-7}), notre schéma de calcul *M2L* par BLAS est donc là encore soit meilleur, soit aussi rapide que le calcul *M2L* avec ondes planes.

8.6 Comparaisons en astrophysique avec *falcON*, le *treecode* et GADGET-2

Nous nous intéressons finalement aux cas des distributions gravitationnelles. Les simulations astrophysiques nécessitant de faibles précisions (voir section 2.3.3.2), les valeurs de P utilisées en pratique sont $P = 2$ ou $P = 3$. Nous avons déjà vu, aux sections 6.5 et 8.5.1, que pour ces valeurs de P, et avec un schéma de calcul *M2L* classique, le noyau *M2L* de hauteur simple est le plus efficace. De plus, pour ces simulations astrophysiques, il est inutile de recourir à un schéma de calcul spécial pour l'opérateur *M2L* :

- le schéma de calcul *M2L* avec une FFT par blocs réduit certes les temps CPU mais les hauteurs d'octree rencontrées dans ces simulations, notamment celles supérieures ou égales à 13, engendrent alors des instabilités numériques ;
- le schéma de calcul avec rotations pour les petites valeurs de P (ici $P = 2$ ou $P = 3$), avec un noyau *M2L* de hauteur simple, est en fait moins rapide que le calcul *M2L* classique, comme l'indique la figure 6.2 en section 6.2 ;
- sans zones uniformes, le schéma de calcul par BLAS est lui aussi moins rapide que le calcul *M2L* classique pour ces valeurs de P avec un noyau de hauteur simple (voir section 8.4.3.1), et malheureusement les zones uniformes détectées dans les distributions astrophysiques sont trop petites pour être exploitées efficacement (voir remarque 8.2) ;
- enfin, en ce qui concerne les ondes planes, la première précision offerte par le code FMMPART3D donne en pratique une erreur relative (erreur L_2) de l'ordre de 10^{-4} ou 10^{-5} ce qui est excessif pour des simulations astrophysiques.

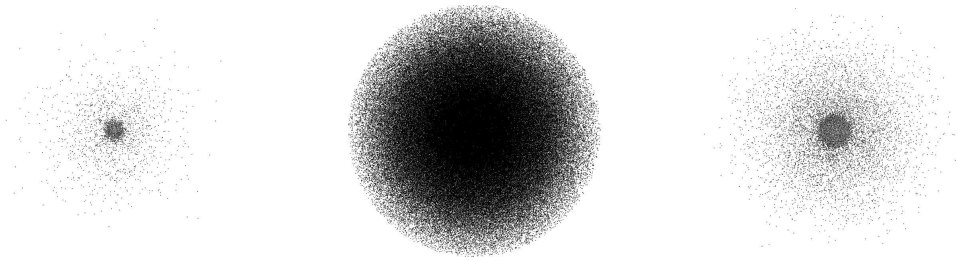
Par ailleurs, afin de pouvoir utiliser notre code FMB dans le cadre de l'astrophysique, nous avons dû inclure un paramètre de *softening* ϵ [41] [43] [89] qui modifie l'expression du calcul direct du potentiel (à une distance r d'une masse m) ainsi

$$\Phi(r) = -\mathcal{G}\frac{m}{\sqrt{r^2 + \epsilon^2}},$$

$\mathcal{G}$ étant la constante de gravitation universelle. Ceci permet d'éviter une divergence dans le calcul du potentiel lorsque deux particules se rapprochent l'une de l'autre, et le calcul direct de la force est modifié pareillement. Le calcul du champ lointain n'est pas modifié car l'effet de ce *softening* n'est volontairement sensible que sur de courtes distances : des valeurs de l'ordre de 10^{-2} ou 10^{-1} sont utilisées en pratique pour ϵ.

Nous allons donc comparer les performances (en termes de temps CPU) de différents codes pour diverses simulations astrophysiques. Les codes comparés sont les suivants :

- le *treecode* de Joshua E. Barnes (version 1.4) [18], implémentant l'algorithme de Barnes & Hut décrit en section 2.3.2 ;
- le code GADGET-2 de Volker Springel (version 2.0) [114], qui combine l'algorithme de Barnes & Hut avec un algorithme de type Particle-Mesh pour former une méthode TreePM (voir sections 2.2.5 et 2.3.2) ;



(a) *wde064* (t=900). (b) *wde475* (t=0). (c) *wde475* (t=600).

FIG. 8.16 – Distributions astrophysiques supplémentaires.

- le code *falcON* de Walter Dehnen (version du 23 septembre 2004), qui implémente une méthode hybride, entre l'algorithme de Barnes & Hut et la FMM, présentée en section 2.3.3.13 ;
- et enfin notre code FMB implémentant la FMM.

Les comparaisons sont effectuées entre les versions séquentielles de chaque code sur un PC Linux disposant d'un processeur Intel Pentium 4 à 2,6 Ghz, disposant de 1 Go de mémoire, d'un cache L1 de données de 8 Ko et d'un cache L2 de 512 Ko. Tous les calculs sont effectués en simple précision.

En plus du modèle de Plummer et du cas *hcg027* présentés en section 8.4, nous utiliserons aussi le cas d'une galaxie à disque avec une barre (1 163 030 particules au total), nommé *wde064* (au temps $t = 900$), et deux états d'une même galaxie, *wde475*, de 1 163 034 particules, aux temps $t = 0$ et $t = 600$. Ces distributions sont représentées à la figure 8.16. Dans le cas *wde475*, les particules sont beaucoup plus dispersées au temps $t = 600$ qu'au temps $t = 0$: la hauteur d'octree nécessaire est donc plus importante au temps $t = 600$ ($H = 13$ pour $P = 2$ dans FMB) qu'au temps $t = 0$ ($H = 8$). A titre complémentaire, nous présenterons aussi les résultats pour une distribution uniforme.

En ce qui concerne la précision des simulations pour *falcON*, le *treecode* et GADGET-2, le critère d'acceptation utilisé est de $0,7$. Le *treecode* permet par ailleurs d'utiliser des développements monopolaire (`mono`), ou des développements quadripolaires (`quad`), plus précis mais plus coûteux (voir section 2.3.2). Dans le cas de GADGET-2, nous utilisons le MAC (*Multipole Acceptance Criterion*) standard de Barnes & Hut, et comme nous restreignons la comparaison aux méthodes hiérarchiques, le TreePM n'est pas activé. Mais, les conditions aux limites étant libres, l'utilisation du TreePM impliquerait des FFT de taille deux fois plus grande que pour des conditions aux limites périodiques (à cause du « zero-padding » : voir section B.6 et [114]). Pour FMB, le degré maximum des développements est $P = 2$ ou $P = 3$ ($P = 2$, qui implique 3 termes dans les développements utilisés, semble suffisant dans la majorité des cas, et $P = 3$ sera juste donné à titre indicatif) avec un noyau *M2L* de hauteur simple. Et comme justifié en section 8.4.4, nous n'effectuons pas de traitement spécial pour les cellules non adaptatives.

Le tableau 8.11 présente un récaptitulatif des temps CPU obtenus lorsqu'on calcule à la fois la force (ou l'accélération) et le potentiel.

Les codes GADGET-2 et FMB permettent de plus de ne calculer que la force, ce qui réduit sensiblement les temps de calculs. Dans le cas de FMB, outre le calcul direct qui est naturellement plus rapide si on ne calcule que la force, le calcul du champ lointain est lui

	treecode		GADGET-2	*falcON*	FMB	
	`mono`	`quad`			$P=2$	$P=3$
Plummer 100k	15,84	26,88	11,74	0,77	5,07	7,80
Plummer 500k	96,6	169,32	67,61	3,85	26,51	42,13
Plummer 1M	202,32	363,54	142,69	7,68	52,75	77,81
Plummer 2M	-	-	304,29	15,22	126,34	163,34
Plummer 3M	-	-	464,20	22,10	181,39	275,34
hcg027	-	-	174,75	10,63	98,72	118,67
wde064 (t=900)	275,94	447,42	217,69	8,96	110,67	147,61
wde475 (t=0)	152,58	253,08	98,17	8,67	70,99	120,48
wde475 (t=600)	291,96	494,76	193,09	8,95	107,68	159,96
Uniforme (500k)	32,70	47,76	27,01	3,99	11,50	14,82

TAB. 8.11 – Comparaison de différents codes pour diverses simulations astrophysiques : calcul de la force et du potentiel. On considère ici les temps CPU totaux (en secondes) englobant la construction de l'octree et le calcul des forces sur un seul pas de temps (*softening* $\epsilon = 0$). Pour le *treecode*, le cas *hcg027* donne une erreur dans la construction de l'octree, alors que les cas Plummer 2M et Plummer 3M débouchent sur des erreurs de débordement dans les calculs flottants.

aussi accéléré. En effet, comme indiqué dans [43], la force résulte de la dérivée du potentiel exprimé par le développement local : le premier terme de ce développement L_0^0 disparaît lors de la dérivation, et il n'a donc pas besoin d'être calculé[6] lors de l'opération *M2L*. Nous présentons donc au tableau 8.12 les temps CPU obtenus en ne calculant que la force, pour GADGET-2 et pour FMB.

	GADGET-2	FMB	
		$P=2$	$P=3$
Plummer 100k	6,37	4,09	6,22
Plummer 500k	36,38	21,92	34,14
Plummer 1M	76,92	41,95	61,96
Plummer 2M	163,92	97,46	127,12
Plummer 3M	254,14	153,26	212,10
hcg027	94,39	75,20	91,22
wde064 (t=900)	113,35	84,33	113,64
wde475 (t=0)	55,87	56,89	94,82
wde475 (t=600)	106,67	91,56	126,12
Uniforme (500k)	15,32	8,85	11,51

TAB. 8.12 – Comparaison de différents codes pour diverses simulations astrophysiques : calcul de la force uniquement. On considère ici les temps CPU totaux (en secondes) englobant la construction de l'octree et le calcul des forces sur un seul pas de temps (*softening* $\epsilon = 0$).

[6] Dans le cas d'un noyau *M2L* de hauteur simple, on pourrait aussi ne pas calculer les termes du développement multipôle $M_P^k, \forall k/|k| < P$ inutiles lors de l'opération *M2L*, mais ceux-ci sont nécessaires pour l'opération *M2P* et sont donc bien calculés.

D'après ces deux tableaux, FMB est clairement plus rapide que le *treecode*, et il est aussi plus rapide que GADGET-2 : il n'y a que dans le cas *wde475* ($t = 0$, avec calcul de la force seule) que les deux codes présentent des temps similaires. Ceci contredit donc les résultats de [30] (voir section 2.3.3.13) où la FMM était annoncée comme deux à quatre fois plus lente que l'algorithme de Barnes & Hut. De plus, alors que le temps de calcul est multiplié par deux entre le temps $t = 0$ et le temps $t = 600$ pour le cas *wde475*, avec le *treecode* comme avec GADGET-2, la différence est moins important avec FMB.

Par contre entre FMB et *falcON*, le rapport varie entre 7 et 12 suivant les simulations gravitationnelles étudiées, mais tombe à moins de 3 pour la distribution uniforme. La différence entre le temps $t = 0$ et le temps $t = 600$ pour le cas *wde475* est par ailleurs très faible. Comme expliqué en section 2.3.3.13, ceci est principalement dû à la dynamicité du MAC (à savoir du critère d'acceptation entre le calcul direct et le calcul approché (avec les développements)) utilisée dans *falcON* : en fonction de la disposition des particules dans la cellule cible et dans la cellule source, *falcON* détermine très efficacement s'il est possible (en respectant la borne d'erreur) d'utiliser les développements pour évaluer l'interaction entre les deux cellules, et ce même si elles sont voisines. Par contre, dans FMB, ce critère n'est basé que sur des considérations géométriques : on choisit à chaque fois le calcul direct pour les voisins immédiats de la cellule cible (cas $ws = 1$). Concrètement, cette différence influe sur le nombre de racines carrées utilisées (une pour chaque interaction calculée de façon exacte par la méthode directe) ; cela est bien illustré dans la table 8.13 où l'on retrouve un rapport d'au moins 10 entre faclON et FMB.

	falcON	FMB
Plummer 10k	$3,37 \times 10^5$	$5,18 \times 10^6$
Uniforme (1M)	$3,6 \times 10^7$	$3,86 \times 10^8$

TAB. 8.13 – Nombre d'appels à la fonction de calcul de racine carrée pour les codes *falcON* et FMB.

Le coût du calcul direct est donc divisé d'autant, et les développements utilisés dans *falcON* étant moins coûteux à manipuler que ceux de la FMM (car optimisés pour les faibles précisions, voir section 2.3.3.13), le résultat est au final plus performant pour ce type de simulations.

Cependant on remarque que dans le cas d'une distribution uniforme, notre rapport de 3 avec un calcul *M2L* classique est bien meilleur que celui de 10 annoncé dans [43] entre les temps CPU de *falcON* et les temps CPU extraits de l'article de Cheng *et al.* [34] (FMM avec un calcul *M2L* par ondes planes). Ce rapport devrait encore baisser en tirant partie de l'uniformité de la distribution avec un schéma BLAS sans recopies (par *ligne* ou par *tranche*). Nous reviendrons sur ce point dans le chapitre traitant du parallélisme (section 9.1.4).

8.7 Conclusion

Dans cette partie, nous avons donc étendu notre formulation BLAS à la FMM adaptative de Nabors *et al.* [93] qui a été améliorée au niveau algorithmique. En fonction des seuils s^M_{min} et s^L_{min} sur le nombre de particules d'une cellule, celle-ci peut désormais avoir soit un développement multipôle, soit un développement local, soit les deux ou soit aucun. Une procédure a été mise en place, puis validée, afin d'estimer les valeurs optimales pour ces

seuils. Toujours sur le plan algorithmique, nous avons abandonné le traitement spécial des cellules non adaptatives : les performances sont ainsi meilleures en pratique grâce au précalcul des fonctions de transfert. Une nouvelle structure de données, l'octree avec indirection, a de plus été conçue. Cette structure offre une implémentation efficace des distributions uniformes tout en étant capable de traiter des octrees hautement déséquilibrés avec des hauteurs supérieures à 10. L'utilisation des routines BLAS a ensuite permis d'accélérer le calcul *M2L* pour les valeurs moyennes de P (BLAS de niveau 2) et pour les grandes valeurs de P (BLAS de niveau 3). Grâce à la détection des zones uniformes, l'efficacité du schéma de calcul par BLAS est encore améliorée, notamment pour les petites valeurs de P.

Pour la gamme de précision qui nous intéresse, notre schéma de calcul combinant les BLAS de niveaux 3, les BLAS de niveau 2 et le calcul *M2L* classique, apparaît alors comme le plus performant face aux autres schémas de calcul *M2L*, ce qui justifie de son intérêt dans le cas non uniforme (avec ou sans zone uniforme). Dans le cas des simulations astrophysiques, si notre code FMB est plus lent que le code *falcON*, spécialisé exclusivement dans ce type de simulation, nous avons montré qu'une implémentation efficace de la FMM s'avère plus rapide que l'algorithme de Barnes & Hut. De plus, comme remarqué en section 2.3.3.13, une parallélisation efficace de *falcON* semble plus difficile à mettre en œuvre que celle de FMB : le rapport entre les deux codes est donc susceptible de s'inverser avec une parallélisation efficace de la FMM lorsqu'il s'agira de passer à l'échelle pour de très grandes simulations.

Nous allons maintenant montrer que notre FMM peut être efficacement parallélisée, en mémoire partagée, comme en mémoire distribuée.

Quatrième partie

Parallelisation

Chapitre 9

Parallélisation

Sommaire

Nous présentons ici deux parallélisations de notre version de la FMM : la première en mémoire partagée et en mode multi-*thread*, et la seconde en mémoire distribuée et en mode multi-processus. Comme expliqué en section 8.4.4, nous n'effectuons a priori plus de traitement spécial pour les cellules non adaptatives (présentes dans les chaînes de l'octree) ; outre les gains de performances, ceci simplifie la parallélisation de l'algorithme.

Au préalable une remarque est nécessaire sur l'implémentation de l'ordre de Hilbert décrit en section 2.3.3.11. Cet ordre offre une meilleure localité des données que l'ordre de Morton lors de la distribution des cellules sur différents processeurs, mais il est en contrepartie plus coûteux de calculer l'indice de Hilbert d'une cellule à partir des coordonnées de son centre. Ceci implique, lorsque la décomposition de Hilbert est utilisée, des conversions entre l'indice de Morton et l'indice de Hilbert d'une cellule ce qui se traduit par un surcoût soit en termes de temps CPU, soit en termes d'espace mémoire si on stocke toutes les conversions. Nous avons ici choisi de stocker les conversions dans le sens « Hilbert vers Morton », et de calculer les conversions dans l'autre sens car ces dernières sont moins fréquentes. Pour une implémentation efficace de ces conversions, nous renvoyons à [82].

9.1 Mémoire partagée en mode multi-*thread*

Nous nous intéressons ici à une exécution parallèle au sein d'un unique processus. Ce processus s'exécute sur plusieurs processeurs (de 2 à 16, voire 32) partageant le même espace d'adressage par l'intermédiaire de plusieurs *threads* : il y a autant de *threads* de calcul que de processeurs disponibles, et chacun de ces *threads* se doit d'être un *thread noyau* (à savoir indépendant des autres *threads* du point de vue de l'ordonnanceur). Les deux principales techniques de programmation en mode multi-*thread* sont l'introduction de directives de compilation, telles OpenMP [3], ou le recours à une bibliothèque de gestion explicite des *threads*. Nous choisissons ici la seconde technique qui permet de contrôler plus finement les *threads*, et nous utilisons le standard des *threads* POSIX. Le contexte de mémoire partagée permet à tous les *threads* d'accéder à toutes les données de l'octree (cellules, particules, développements) sans aucune communication. En contre partie, les accès concurrents à ces données partagées doivent être contrôlés : aucune lecture ou écriture ne peut avoir lieu sur une donnée en cours de modification.

Comme annoncé au chapitre 3, l'équilibrage de charge entre les différents *threads* est obtenu grâce à une décomposition statique de chaque niveau de l'octree suivant l'ordre de Morton ou l'ordre de Hilbert. Chaque *thread* se voit ainsi attribué, à un niveau donné de l'octree, un intervalle continu d'indices de Morton ou de Hilbert. Au sein de cet intervalle, qui représente le *domaine* du *thread*, chaque indice correspond à une cellule « cible » pour les diverses opérations de la FMM (*M2M*, *M2L*, *M2P*, *L2P*, *L2L*, *P2P*), et chaque *thread* est alors chargé d'effectuer toutes ces opérations pour toutes les cellules de son domaine. Par la suite, nous écrirons qu'une cellule *appartient* à un *thread* si l'indice de cette cellule est compris dans l'intervalle dédié à ce *thread*.

Contrairement aux décompositions similaires déjà réalisées, telle que *costzones* (voir section 2.3.3.11), nous choisissons de distinguer la décomposition utilisée pour le calcul du champ proche et la décomposition utilisée pour le calcul du champ lointain. En effet, la charge de calcul n'est pas la même : alors que le calcul du champ lointain est essentiellement basé sur les développements multipôles et locaux des cellules cibles et sources, le calcul du champ proche est lui fonction du nombre de particules dans chaque cellule. De plus, alors que le calcul du champ lointain fait intervenir les différents niveaux de l'octree, le calcul du champ proche n'est effectué qu'au niveau des feuilles.

Nous allons désormais présenter ces deux décompositions ainsi que les méchanismes de synchronisation entre *threads* qui leur sont associées.

9.1.1 Mise en place pour le calcul du champ proche

9.1.1.1 Principe des interactions réciproques

Une contrainte importante pour la parallélisation du champ proche en mémoire partagée est une dépendance temporelle : le recours au principe des interactions réciproques a pour conséquence de mettre à jour, lors d'une opération *P2P*, le potentiel et la force des particules de la feuille cible mais aussi de celles de la feuille source. Ainsi lors d'une opération *P2P* effectuée par le *thread* t_1, si la feuille source appartient au domaine d'un *thread* t_2 ($t_2 \neq t_1$), nous avons un rique de conflit en écriture si cette feuille source est concurremment utilisée pour une autre opération *P2P* effectuée par t_2. L'exemple d'un tel conflit est donné à la figure 9.1.

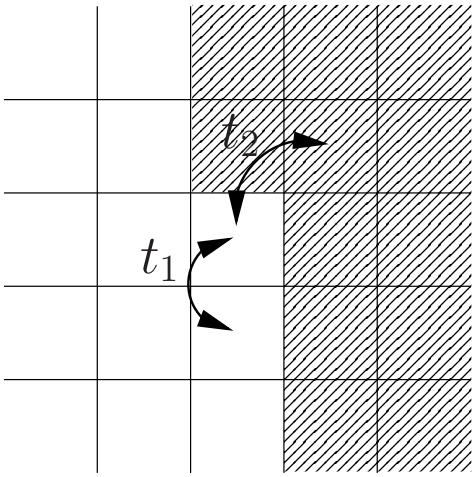


FIG. 9.1 – Exemple de conflit entre deux opérations *P2P* effectuées avec le principe des interactions réciproques par deux *threads* t_1 et t_2 différents. Les deux opérations *P2P* sont représentées par les flèches. Le domaine de t_1 est laissé en blanc alors que celui de t_2 est rayé.

La première solution consiste à renoncer au principe des interactions réciproques entre deux feuilles qui n'appartiennent pas au même *thread*. Ceci a été implémenté dans un premier temps, mais l'efficacité parallèle ainsi obtenue n'est pas optimale à cause de la redondance introduite dans les calculs.

Nous allons donc chercher à conserver le principe des interactions réciproques utilisé dans la version séquentielle. Nous pouvons tout d'abord chercher pour le parcours des différentes feuilles du domaine de chaque *thread*, un ordre particulier qui garantisse qu'il n'y a aura jamais d'accès concurrents à des particules (ou qui minimise le nombre de ces accès concurrents). Une telle solution est envisageable pour des domaines réguliers, tels ceux issus de la décomposition ORB (voir section 2.3.3.11) : par exemple si 8 *threads* se partagent les feuilles de l'octree à l'aide de 8 cubes de même taille, il suffit que chaque *thread* commence par « le même coin » de son domaine, puis progresse dans « la même direction » pour éviter tout conflit. Mais avec la décomposition de Morton ou de Hilbert, nous n'avons pas une idée précise de la « forme » géométrique du domaine de chaque *thread* (voir figures 2.20(a) et 2.20(b)). On pourrait certes revoir la décomposition pour les *threads* mais nous ne souhaitons formuler aucune hypothèse sur la forme du *domaine du processus* (ensemble des cellules cibles à traiter par le processus, on reverra plus en détail cette notion dans la section 9.2). Nous avons en effet déjà choisi d'utiliser la décomposition de Morton (ou de Hilbert) pour le découpage de l'espace entre processus en mémoire distribuée (voir chapitre 3) : par conséquent, dans la perspective d'un couplage entre le mode multi-*thread* et le mode multi-processus, aucune hypothèse sur la forme des domaines des *threads* au sein de chaque processus ne peut être formulée. S'il n'est donc pas possible d'établir une numérotation qui évite tout conflit, le simple fait de parcourir les intervalles de chaque *thread* dans l'ordre croissant de leurs indices (c'est-à-dire en suivant l'ordre de Morton ou de Hilbert avec lequel ils ont été construits) restreint cependant naturellement le nombre de conflits. Deux cellules dont les indices de de Morton sont proches, sont susceptibles d'être proches spatialement, et ceci est encore plus probable avec l'ordre de Hilbert. Ainsi les premiers indices de chaque inter-

valle devrait être suffisamment différents pour ne pas correspondre à des cellules voisines, et il en ira de même en progressant dans l'intervalle.

Mais des accès concurrents aux mêmes particules pourront tout de même avoir lieu. Nous allons donc mettre en place un système de verrous qui interdit deux opérations *P2P* simultanées sur la même feuille. Le traitement d'une de ces deux opérations sera alors différé.

Comme il a été montré dans [59], il est trop coûteux d'utiliser un verrou par particule : nous choisissons donc de bloquer l'accès à toutes les particules d'une feuille à la fois, ce qui réduit le nombre de verrous tout en maintenant suffisamment court le temps pendant lequel chaque verrou sera fermé. Et plutôt que d'avoir recours à un véritable verrou pour chaque feuille, nous proposons de n'utiliser qu'un seul bit qui sera positionné à 1 lorsque la feuille prendra part à une opération *P2P* (et 0 autrement). En choisissant un bit inutilisé au sein de la structure implémentant le type cellule, le surcoût mémoire est nul. Ainsi le système de verrous revient simplement à tester la valeur précédente du bit et à le positionner à 1 avant chaque opération *P2P* : si le bit était déjà à 1, cela signifie que la feuille est en cours d'utilisation par une autre opération *P2P* et la nouvelle opération *P2P* ne peut pas avoir lieu. Si le bit était à 0, l'opération peut s'effectuer, et le bit est remis à zéro à la fin. Cependant, le test et la mise à 1 du bit doivent être faits de manière *atomique* afin d'éviter tout conflit entre *threads*. Il existe certes sur certaines architectures l'instruction `test_and_set` qui réalise exactement ceci, mais une telle opération n'est pas standard, et elle est généralement réservée au code du noyau du système d'exploitation. De plus, cette instruction a l'inconvénient de bloquer l'ensemble du bus mémoire sur des machines multi-processeurs. Nous préférons donc utiliser de simples verrous, tels les *mutexes* fournis avec la bibliothèque de *threads* POSIX. Comme le temps pendant lequel chaque verrou sera fermé sera très court (juste le temps de lire et positionner le bit), il est préférable d'éviter que le *thread* change d'état au niveau de l'ordonnanceur du système (en passant de l'état « en cours d'exécution » à l'état « bloqué ») : il vaut mieux que le *thread* « insiste » (en attente *active*) auprès du système pour obtenir le verrou. Ceci est naturellement mis en place pour les machines IBM sous AIX 5L (voir la variable d'environnement SPINLOOPTIME, dans [21] par exemple, qui contrôle le temps d'attente active) ; pour les machines sous Linux, nous utilisons les *spin locks* qui sont eux aussi implémentés dans la bibliothèque de *threads* POSIX et pour lesquels le *thread* effectue toujours de l'attente active pour obtenir le verrou.

Comme il ne peut y avoir un verrou par bit (surcoût mémoire), nous ne pouvons pas non plus nous contenter d'un unique verrou pour tous les bits : l'accès à ce verrou deviendrait rapidement un « goulot d'étranglement » pour tout le calcul. Nous choisissons donc d'utiliser un verrou par *thread*. Ce verrou est affecté au domaine du *thread*. Ainsi lorsqu'un *thread* veut effectuer une opération *P2P* avec une feuille f (cible ou source), la vérification de l'état du bit est assurée de manière atomique grâce au verrou associé au domaine contenant f.

En pratique, chaque *thread* parcourt donc son domaine et commence par positionner à 1 le bit de la feuille (cible) courante f_c. Puis pour chaque feuille source f_s, membre de la liste du calcul direct de f_c, il positionne le bit de f_s et effectue l'opération *P2P*. Si f_s était déjà marquée, nous stockons alors son indice dans une *file*. Cette file est défilée à la fin du parcours de la liste d'interaction de f_c : nous réessayons alors l'opération *P2P* avec f_s, et nous enfilons à nouveau l'indice de f_s en cas d'échec. Nous continuons ainsi jusqu'à ce que la file soit vide. De la même façon, si le bit de f_c était déjà positionné au début, nous stockons l'indice de f_c dans une seconde file, dédiée aux feuilles cibles, et nous défilons cette

file (éventuellement en plusieurs fois) à la fin du parcours du domaine.

9.1.1.2 Localité des données et équilibrage de charge

Du fait de l'abscence de communications en mode multi-*thread* en mémoire partagée, la localité des données n'intervient qu'au niveau du principe des interactions réciproques. Comme expliqué ci-dessus, notre décomposition statique suivant l'ordre de Morton ou de Hilbert devrait minimiser le nombre de conflits entre *threads* pour l'opération *P2P* du calcul du champ proche. L'ordre de Hilbert peut être a priori plus intéressant malgré le surcoût qu'il induit : nous reviendrons sur ce point en section 9.1.3.

En ce qui concerne l'équilibrage de charge, nous devons estimer le coût de chaque feuille lors de notre décomposition statique afin que chaque *thread* dispose de la même quantité de travail. Si la seule prise en compte de la présence ou non de particules dans une feuille (à savoir un coût de 0 pour une feuille vide, et un coût de 1 pour une feuille non vide) donne des résultats satisfaisants dans le cas de distributions uniformes de particules, l'équilibrage de charge ainsi obtenu n'est pas bon pour des distributions hautement non uniformes telles celles rencontrées en astrophysique. Nous avons donc choisi une estimation plus fine du coût associé à chaque feuille pour le calcul direct. En se basant sur le nombre théorique d'opérations associé à l'opération *P2P* (voir tableau 2.1), nous prenons ainsi en compte le nombre N_i de particules de la feuille d'indice i et le nombre N_j de particules de chacun des membres (d'indice j) de sa liste du champ proche. Et afin de ne pas ignorer le coût associé aux parcours des cellules vides, nous leur attribuons un coût minimal mais non nul (égal à 1 en pratique). Le coût $\mathcal{C}_{dir}(i)$ associé à la feuille d'indice i pour le calcul direct est donc calculé ainsi :

$$\mathcal{C}_{dir}(i) = \begin{cases} 1 & \text{si la feuille } i \text{ est vide,} \\ \frac{N_i^2}{2} + \sum\limits_{i<j} c_{ij} & \text{sinon,} \end{cases}$$

avec

$$c_{ij} = \begin{cases} 1 & \text{si la feuille } j \text{ est vide,} \\ N_i.N_j & \text{sinon.} \end{cases}$$

9.1.2 Mise en place pour le calcul du champ lointain

Dans le cas du calcul du champ lointain, nous sommes confrontés à des dépendances de données : certaines opérations ne peuvent être effectuées que si d'autres sont terminées. C'est le cas des opérations *M2M*, *M2L*, *M2P*, *L2L* et *L2P* : le développement « source » doit avoir été établi avant chacune de ces opérations.

De plus, dans le cas du champ lointain nous devons aussi décomposer entre les différents *threads* les niveaux internes de l'octree, et non plus uniquement le niveau des feuilles.

9.1.2.1 Coût associé aux cellules internes

Il existe deux façons de décomposer les niveaux internes de l'octree.

1. On peut tout d'abord se baser sur la décomposition obtenue au niveau des feuilles. Cette décomposition est obtenue à l'aide de l'ordre de Morton ou de Hilbert après avoir attribué un coût à chacune des feuilles, de la même façon que dans le cas du

champ proche. En notant H le niveau des feuilles, on construit ensuite la décomposition au niveau $H-1$ ainsi : chaque cellule au niveau $H-1$ est attribué au *thread* qui possède son fils de numéro 0 au niveau H. Dans le cas de l'ordre de Morton, le fils de numéro 0 correspond d'après notre convention au *type de fils* BDL (correspondant à la position *Back-Down-Left*, voir section 5.3.1). Cette décomposition n'impose pas d'affecter un coût aux cellules internes, mais ceci reste théoriquement possible : il suffit d'inclure le coût de chaque cellule interne dans le coût de chaque feuille dont elle est l'ancêtre ;

2. Une autre solution consiste à effectuer des décompositions indépendantes les unes des autres à chaque niveau interne : on se base alors uniquement sur le coûts des cellules au niveau considéré.

La première décomposition a l'avantage d'attribuer à un même *thread* une cellule et ses fils (sauf éventuellement pour les extrémités de chaque intervalle), ce qui permet de minimiser les communications potentielles pour les opérations *M2M* et *L2L* : cette solution pourra donc être retenue en mode multi-processus (voir section 9.2). Mais ici en mode multi-*thread*, nous n'avons pas de communication à effectuer, et cette première solution a l'inconvénient d'aboutir à des niveaux où certains *threads* n'ont pas de cellules, surtout dans le cas de distributions hautement non uniformes. Nous préférons donc adopter la deuxième solution dans le cas de la programmation en mode multi-*thread* afin d'obtenir le meilleur équilibrage de charge possible entre les différents *threads* à chaque niveau.

En ce qui concerne le coût affecté à une cellule de l'octree pour le calcul du champ lointain, on dispose de plusieurs possibilités.

1. On peut tout d'abord se baser uniquement sur le caractère vide ou non de la cellule : à savoir, un coût de 0 pour une cellule vide, et un coût de 1 pour une cellule non vide. On notera ce coût $\mathcal{C}_{NonVide}$;
2. On peut aussi tenter de prendre en compte plus finement le nombre de particules présentes dans la cellule en attribuant un coût proportionnel à ce nombre. On notera ce coût $\mathcal{C}_{NbPart}$ (*Nombre de Particules*) ;
3. Enfin on peut souhaiter prendre en compte la liste d'interaction de chaque cellule : le coût d'une cellule c est alors obtenu en sommant le coût associé à chaque membre de la liste d'interaction de c, ce coût étant 0 pour un membre vide et 1 pour un membre non vide. On notera ce coût $\mathcal{C}_{Liste\_I}$ (*Liste d'Interaction*).

Chacun de ces coûts impliquent une décomposition et donc des intervalles différents au niveau de l'octree considéré. En pratique, on utilise les coûts suivants pour les diverses opérations du calcul du champ lointain :

– le coût $\mathcal{C}_{NonVide}$ est utilisé pour les opérations *M2M*, *P2M* et *L2P* ;
– le coût $\mathcal{C}_{Liste\_I}$ est utilisé pour les opérations *M2L* et *L2L*.

Le coût $\mathcal{C}_{NbPart}$ aurait pu sembler plus approprié pour les opérations *P2M* et *L2P* dont la complexité théorique est linéaire par rapport au nombre de particules (voir tableau 2.1 page 38), mais les tests que nous avons effectués ont montré qu'en pratique le gain ainsi obtenu était très faible : le coût réel est en effet plutôt une fonction affine du nombre de particules, et il vaut mieux garder les mêmes intervalles entre *P2M* et *M2M* pour la remontée. Nous n'avons donc pas retenu cette solution. D'autre part, en utilisant $\mathcal{C}_{Liste\_I}$ et non $\mathcal{C}_{NonVide}$ pour l'opération *M2L*, on améliore clairement la prise en compte du coût de cette opération, y compris dans le cas uniforme car les cellules aux bords de l'octree ont des liste d'interaction

incomplètes (conditions aux limites libres). Enfin, le fait d'affecter les mêmes coûts, et donc les mêmes domaines, aux opérations *M2L* et *L2L* permet de regrouper ensemble les cellules cibles de ces opérations pour un même *thread*, et d'éviter ainsi des conflits en écriture lors de la phase de descente.

9.1.2.2 Phases de remontée et de descente

Nous devons aussi procéder à quelques modifications d'ordre algorithmique pour les phases de remontée et de descente.

Tout d'abord, la phase de calcul direct est désormais intercalée entre la phase de remontée et celle de descente. Ceci permet en effet de relâcher les contraintes de dépendances de données portant sur les premiers niveaux de l'octree entre la fin de la remontée et le début de la descente. De plus, ceci sera de toute façon requis par la version en mémoire distribuée comme nous le verrons en section 9.2.

Si on détaille maintenant les dépendances de données associées à chaque opération, il faut tout d'abord remarquer que les dépendances associées à l'opération *M2L* posent rarement problème : grâce à l'intercalation de la phase de calcul direct entre la remontée et la descente, tous les *threads* ont terminé leur phase de remontée avant que le premier d'entre eux ne débute sa phase de descente, et les développements multipôles sont donc tous déjà calculés au moment de la première opération *M2L* (nous vérifions bien sûr en pratique que tel est bien le cas).

Par contre, les dépendances $P2M \rightarrow M2M$ [1], $M2M \rightarrow M2M$, $L2L \rightarrow L2L$ [2] et $L2L \rightarrow L2P$, qui interviennent entre deux niveaux de l'octree pendant la remontée ou la descente, sont elles plus sensibles. Notre décomposition indépendante des niveaux internes de l'octree fournit certes un équilibre entre les calculs à effectuer par chaque *thread* à chaque niveau, mais cet équilibrage ne peut être parfait. Des retards entre les *threads* peuvent donc intervenir, et certaines dépendances ne sont plus satisfaites. On pourrait résoudre ces dépendances à l'aide de barrières bloquant tous les *threads* entre chaque niveau, mais cette contrainte est trop forte en pratique, et les performances ainsi obtenues ne sont pas satisfaisantes. Nous avons donc mis en place un algorithme « sans barrière » en utilisant des bits dédiés et des files, à la manière de ce qui a été utilisé pour le calcul du champ proche. Plus précisément, deux bits (inutilisés dans la structure implémentant le type cellule) indiquent si le développement multipôle et le développement local ont été calculés ou non. Ces bits sont testés avant chaque opération *M2M*, *L2L* et *L2P*, et si le développement source n'a pas encore été calculé, on stocke la cellule cible dans une file dédiée. Une fois que le *thread* a fini de parcourir les cellules de son intervalle à un niveau donné, il défile cette file (éventuellement en plusieurs fois)

On peut noter que pendant la phase de descente, toutes les opérations *M2L* à effectuer par un *thread* à un niveau donné sont réalisées avant toute opération *L2L*. Ceci permet d'amortir les décalages entre les différents *threads*, et ce d'autant plus que le nombre d'opérations *M2L* est bien supérieur au nombre d'opérations *L2L*. Pour la phase de remontée, ceci n'est pas possible, et de moins bonnes performances sont à prévoir dans ce cas.

Enfin, des conflits sont théoriquement possibles entre les opérations *P2P* de la phase du calcul direct et les opérations *M2P*, *P2P* et *L2P* qui peuvent avoir lieu lors de la phase de descente (voir section 8.1.1). On utilise donc lors des opérations *M2P*, *P2P* et *L2P* les bits et

[1] Effectué dans le sens : fils → cellule courante.
[2] Effectué dans le sens : père → cellule courante.

les verrous du calcul direct (voir section 9.1.1.1) pour obtenir un accès exclusif aux particules. Cependant nous n'utilisons pas ici de système de files et nous préférons l'attente active : ces cas sont en effet rares et l'attente y est généralement brève.

9.1.2.3 Schéma de calcul par BLAS

En ce qui concerne la parallélisation du schéma de calcul par BLAS en mémoire partagée, une première possibilité consiste à utiliser une implémentation multi-*thread* des routines BLAS : un seul appel BLAS est alors traité par plusieurs *threads* de calcul. La bibliothèque ESSL [4] que nous utilisons sur les machines IBM offre une telle possibilité sous l'appellation ESSL SMP (*Symmetric Multi-Processing*). Cependant la taille des matrices que nous utilisons est généralement trop petite pour offrir un grain de calcul suffisant à chaque *thread* : en pratique seule une valeur de $P = 15$ (avec un noyau *M2L* de hauteur double) offre des performances correctes sur 4 processeurs. Comme nous souhaitons obtenir une version multi-*thread* efficace de notre schéma de calcul par BLAS pour toutes les valeurs de P, nous allons procéder différement : chaque appel BLAS sera traité par un unique *thread* de calcul, et plusieurs appels BLAS seront donc effectués concurremment par plusieurs *threads*. La bibliothèque BLAS utilisée doit bien sûr être réentrante.

Suivant que nous sommes dans une zone uniforme (voir section 8.3) ou non, la mise en place d'une version multi-*thread* de notre schéma de calcul par BLAS diffère.

Dans les zones uniformes. Au sein d'une zone uniforme, nous avons distingué aux sections 5.3 et 8.3 les cellules dont la liste d'interaction est entièrement contenue dans la zone uniforme et celles dont certains membres de la liste d'interaction sont situés hors de la zone uniforme : ces dernières sont situées aux bords de la zone uniforme (ce sont les cellules avec liste d'interaction incomplète de la section 5.3).

Pour ces cellules aux bords de la zone uniforme, la double boucle de l'algorithme 5 (section 5.3) est découpée le plus équitablement possible entre les *threads*.

Pour les cellules dont la liste d'interaction est entièrement contenue dans la zone uniforme, nous disposons de notre schéma avec *recopies* et de nos stockages spéciaux des données en mémoire, par *ligne* ou par *tranche* (voir section 5.3.2). En ce qui concerne le schéma par *recopies*, nous conservons l'algorithme 7 choisi à la section 5.3.2 et l'ensemble des développements à traiter est décomposé le plus équitablement possible entre les *threads* en paquets contigus. Enfin pour les stockages de données par *ligne* ou par *tranche*, chaque ligne (ou tranche) est traitée par un unique *thread*. L'ensemble des lignes (ou tranches) à traiter est donc là encore décomposé entre les différents *threads* de la façon la plus équitable possible. Afin de maximiser les effets dûs aux caches de la hiérarchie mémoire, nous attribuons à un même *thread* des lignes (ou tranches) adjacentes dans l'espace, c'est-à-dire contiguës dans la mémoire du processus.

Hors des zones uniformes. Hors des zones uniformes, ou en l'abscence de celles-ci, nous utilisons l'algorithme 5 de la section 5.3 comme justifié en section 8.3.1, et nous distribuons la boucle interne entre les différents *threads* : ceci permet en effet d'éviter tout conflit en écriture.

Calcul *L2L*. La répartition des cellules entre les *threads* pour la version BLAS ne correspondant plus aux intervalles définis en section 9.1.2.1 pour le calcul *M2L* classique : il n'y a plus

de correspondance entre les cellules cibles traitées par un même *thread* pour le calcul *M2L* par BLAS et pour le calcul *L2L*. Nous utilisons donc dans ce cas une barrière entre les opérations *M2L* et les opérations *L2L* à chaque niveau de la phase de descente afin de prévenir les conflits en écriture.

9.1.3 Validation expérimentale

Nous présentons tout d'abord des résultats obtenus sur un nœud SMP (*symmetric multiprocessors*) du cluster IBM p575 du pôle M3PEC[3] localisé à la DRIMM[4] de l'Université Bordeaux 1. Ce nœud est composé de 8 processeurs Power5 dual-core (soit 16 processeurs en tout) cadencés à 1,5 Ghz et partageant 30 Go de mémoire. Les deux *cores* de chaque processeur ont en commun un cache L2 de 1,9 Mo et un cache L3 de 36 Mo.

Pour ces tests, nous utilisons deux valeurs de P : $P = 3$ (et un noyau *M2L* de hauteur simple) pour les basses précisions, et $P = 7$ (et un noyau *M2L* de hauteur double) pour les moyennes précisions. Le grain de calcul est faible dans le premier cas, alors qu'il peut être considéré comme gros dans le second. Nous ne présentons pas ici de résulats pour les hautes précisions ($P = 15$ et un noyau *M2L* de hauteur double) car les tests ont montré que nous obtenions alors les mêmes résultats que pour $P = 7$; le grain de calcul du cas $P = 7$ est en effet suffisamment gros pour être représentatif de l'efficacité parallèle obtenue pour les moyennes et les hautes précisions.

Dans le cas des simulations astrophysiques, nous utiliserons $P = 2$ (avec un noyau *M2L* de hauteur simple) comme justifié en section 8.6.

Dans les tests suivants, nous présentons l'efficacité parallèle pour différentes phases de la FMM. Cette efficacité parallèle est calculée ainsi en mode multi-*thread* :

$$\text{efficacité} = \frac{\text{Temps séquentiel}}{(\text{Temps avec } t \textit{ threads}) \times t}.$$

Elle est présentée ici pour les phases de remontée et de descente, ainsi que pour la phase du calcul direct (champ proche) et pour la phase d'évaluation (opérations *L2P*). Nous indiquons aussi l'efficacité pour le « calcul FMB » qui représente l'ensemble de ces opérations au sein de notre code FMB. Enfin, les différentes décompositions statiques sont considérées comme une phase de précalcul, dont le coût n'est donc pas pris en compte. Ce coût, fonction du nombre de cellules, n'est certes pas négligeable mais il pourra être amorti sur plusieurs pas de temps, et l'établissement de ces décompositions pourra être lui-même partiellement parallélisée.

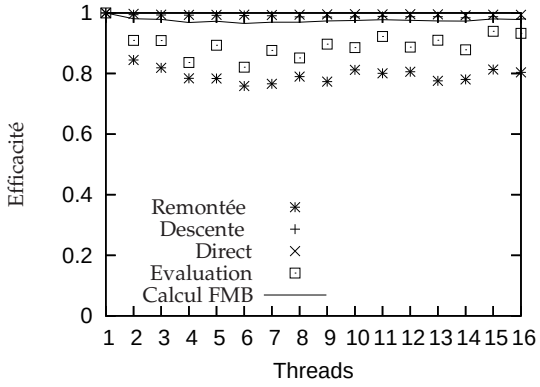
9.1.3.1 Schéma de calcul *M2L* classique

La figure 9.2 présente les efficacités en mode multi-*thread* pour le calcul *M2L* classique avec divers types de distributions présentés en section 8.4. Le tableau 9.1 donne les temps de calcul correspondants. L'ordre utilisé pour la numérotation des cellules est celui de Morton.

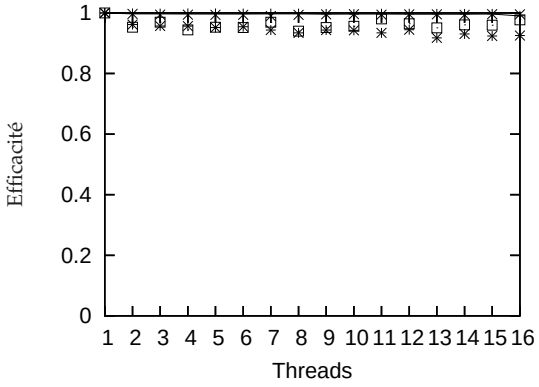
Grâce à nos différentes décompositions dédiées à chaque phase de la FMM, nous obtenons de très bons résultats lorsque le grain de calcul est gros ($P = 7$) mais aussi lorsqu'il

[3]Modélisation Microscopique et Mésoscopique en Physique, dans l'Environnement, en Chimie, Mathématique, Infomatique et Médecine.

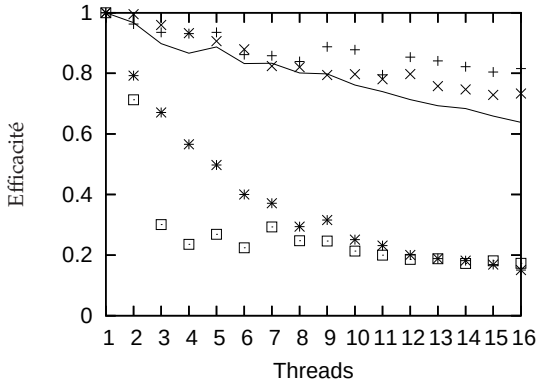
[4]Direction des Ressources Informatiques et Multimédia Mutualisées.



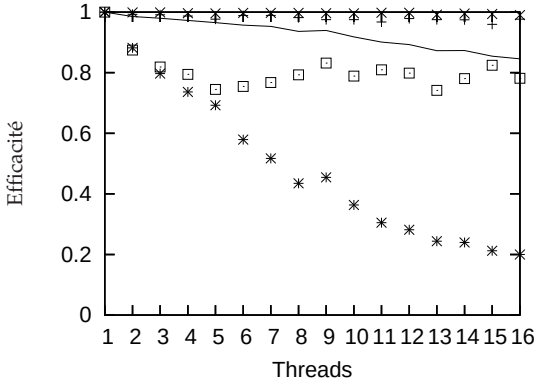
Uniforme (1M) $P = 3$.



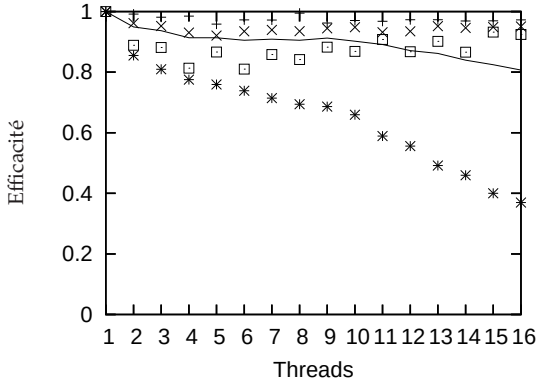
Uniforme (1M) $P = 7$.



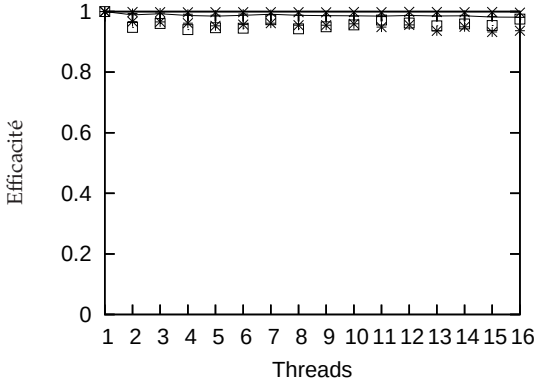
Plummer (1M) $P = 2$.



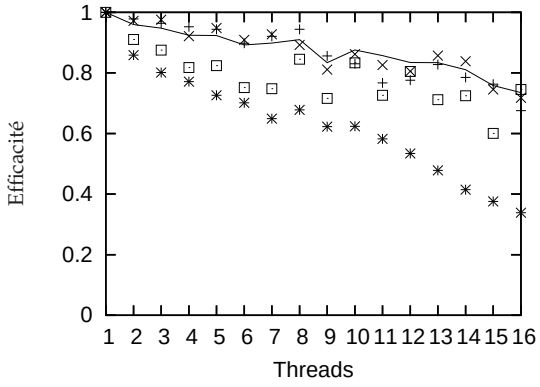
hcg027 $P = 2$.



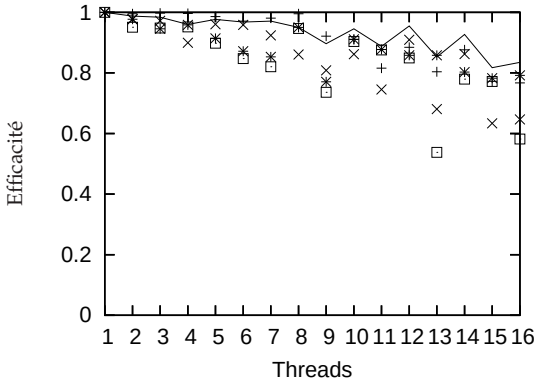
Cylindre (1M) $P = 3$.



Cylindre (1M) $P = 7$.



Sphère (2M) $P = 3$.



Sphère (2M) $P = 7$.

FIG. 9.2 – Efficacités parallèles en multi-*thread* pour le calcul *M2L* classique. La légende est donnée dans le cas uniforme à 1 million de particules et $P = 3$ (en haut à gauche).

threads	Uniforme (1M)		Plummer	*hcg027*	Cylindre (1M)		Sphère (2M)	
	$P = 3$	$P = 7$	$P = 2$	$P = 2$	$P = 3$	$P = 7$	$P = 3$	$P = 7$
1	32,0	240,2	54,3	88,7	15,3	60,2	44,8	176,0
4	8,27	60,3	15,7	22,8	4,2	15,2	12,1	45,8
8	4,1	30,1	8,4	11,8	2,1	7,6	6,2	23,2
16	2,0	15,2	5,3	6,6	1,2	3,8	3,8	13,2

TAB. 9.1 – Temps en secondes pour le « calcul FMB » en mode multi-*thread* : cas du calcul *M2L* classique.

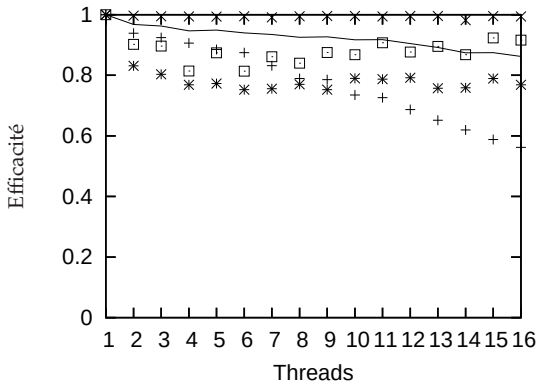
est petit ($P = 3$). Et le fait que les phases de remontée ou d'évaluation soient moins efficacement parallélisées n'a généralement que peu de conséquence sur l'efficacité totale (« calcul FMB »), car ces deux phases ne constituent pas la majeure partie du temps de calcul.

On remarque notamment que la phase du calcul direct (indépendante de la valeur de P) est très efficacement parallélisée : 99 % d'efficacité pour le cas uniforme, et 95 % dans le cas du cylindre. Ceci confirme notre intuition de la section 9.1.1.1, selon laquelle le parcours des intervalles suivant l'ordre de Morton débouche en pratique sur très peu de conflits entre les différentes opérations *P2P* (avec le principe des interactions réciproques). L'intérêt de l'ordre de Hilbert en mode multi-*thread* étant uniquement de réduire ces conflits pour les opérations *P2P* (voir section 9.1.1.2), son emploi n'est donc pas ici nécessaire (ceci a tout de même été confirmé par des tests qui ont montré que son gain n'est ici pas significatif).

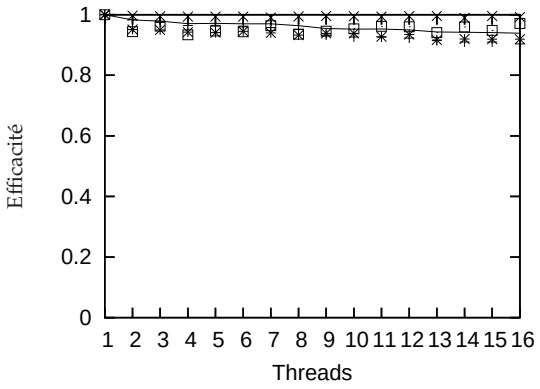
Le cas du modèle de Plummer en astrophysique est hautement non uniforme (voir sa description en section 8.4) ; il est donc naturellement plus difficile à traiter. L'équilibrage de charge est ainsi plus difficile à obtenir de façon statique, et le faible grain de calcul utilisé ($P = 2$ avec un noyau *M2L* de hauteur simple) met à jour un problème dans l'estimation du coût du parcours des cellules vides : leur coût est difficile à évaluer précisément par rapport au coût d'une cellule non vide, et la valeur de 1 choisie en section 9.1.1.2 donne de meilleurs résultats que 0 ou 2, mais ne résout pas complètement ce problème. Nous obtenons néanmoins une efficacité supérieure à 60 % pour 16 *threads* ce qui est convenable pour un équilibrage statique : sans notre distinction entre la décomposition utilisée pour le champ proche et celle utilisée pour le champ lointain, les résultats sont en effet bien moins bons.

Remarque 9.1. *Nous avons obtenu des résultats similaires sur une machine Linux de type CC-NUMA (*Cache Coherent - Non Uniform Memory Access*) formée de 8 processeurs AMD Opteron dual-core (soit 16 processeurs en tout) cadencés à 1,8 Ghz et partageant 64 Go de mémoire. Cette machine est localisé au CREMI*[5] *à l'Université Bordeaux 1. Cependant sur ce type de machine, il est nécessaire de « fixer » chacun des 16* threads *(ou moins) sur un processeur dédié grâce à l'appel de l'appel système adéquat (*`sched_setaffinity` *sous Linux, et* `bindprocessor` *sous AIX) : le déplacement d'un* thread *d'un processeur à l'autre doit en effet être évité sur ces machines CC-NUMA car il y est particulièrement coûteux (invalidation de tous les caches, et transfert de toutes les données entre les mémoires locales). Sur les machines IBM p575, où les processeurs d'un même nœud disposent d'un accès uniforme à la mémoire, ceci n'est pas nécessaire.*

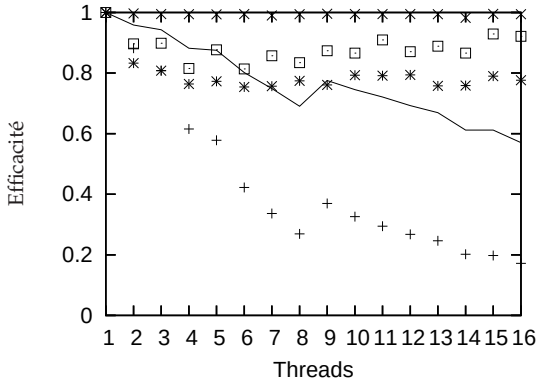
[5]Centre de Ressources pour l'Enseignement des Mathématiques et de l'Informatique.



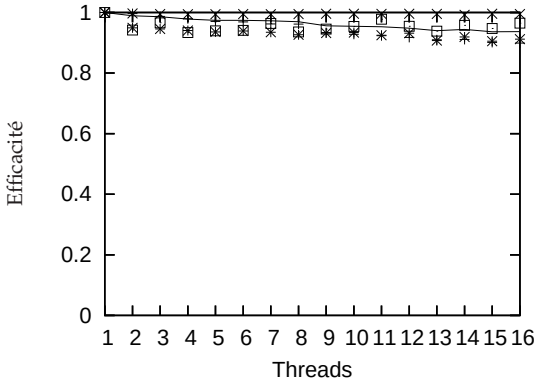
Uniforme (1M) $P = 3$ avec *recopies*.



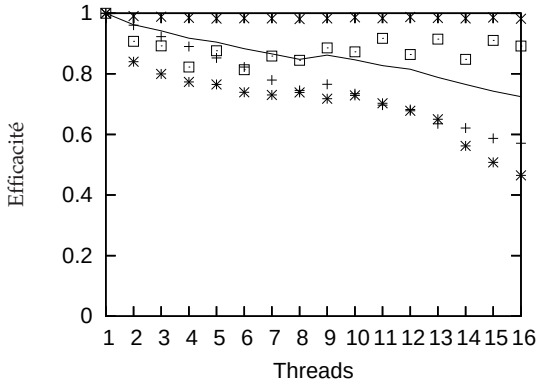
Uniforme (1M) $P = 7$ avec *recopies*.



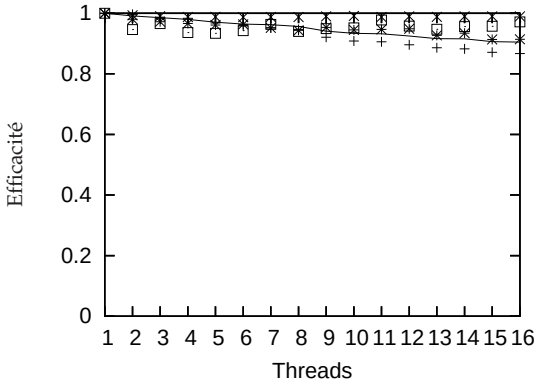
Uniforme (1M) $P = 3$ avec stockage par *ligne*.



Uniforme (1M) $P = 7$ avec stockage par *ligne*.



Balle (2M) $P = 3$ avec stockage par *tranche*.[6]



Barre (3M) $P = 7$ avec stockage par *ligne*.

FIG. 9.3 – Efficacités parallèles en multi-*thread* pour le calcul *M2L* par BLAS sur une zone uniforme. Voir légende en figure 9.2.

9.1.3.2 Schéma de calcul *M2L* par BLAS

Nous présentons ici les résultats obtenus pour la parallélisation de notre schéma de calcul par BLAS présenté en section 9.1.2.3, en distinguant là encore le calcul *M2L* par BLAS au sein d'une zone uniforme et en dehors d'une zone uniforme.

threads	Uniforme (1M) *stockage par ligne*		Balle (2M)	Barre (3M)
	$P = 3$	$P = 7$	$P = 3$	$P = 7$
1	28,6	58,3	60,6	195,7
4	8,1	14,9	16,5	49,9
8	5,2	7,5	8,9	25,6
16	3,1	3,9	5,2	13,5

TAB. 9.2 – Temps en secondes pour le « calcul FMB » en mode multi-*thread* : cas du calcul *M2L* par BLAS sur une zone uniforme. A noter que pour le cas particulier de la distribution uniforme à 1 million de particules avec $P = 3$, le faible gain des BLAS en séquentiel (voir tableau 9.1) est dû à la très forte part du calcul direct dans le temps de calcul total.

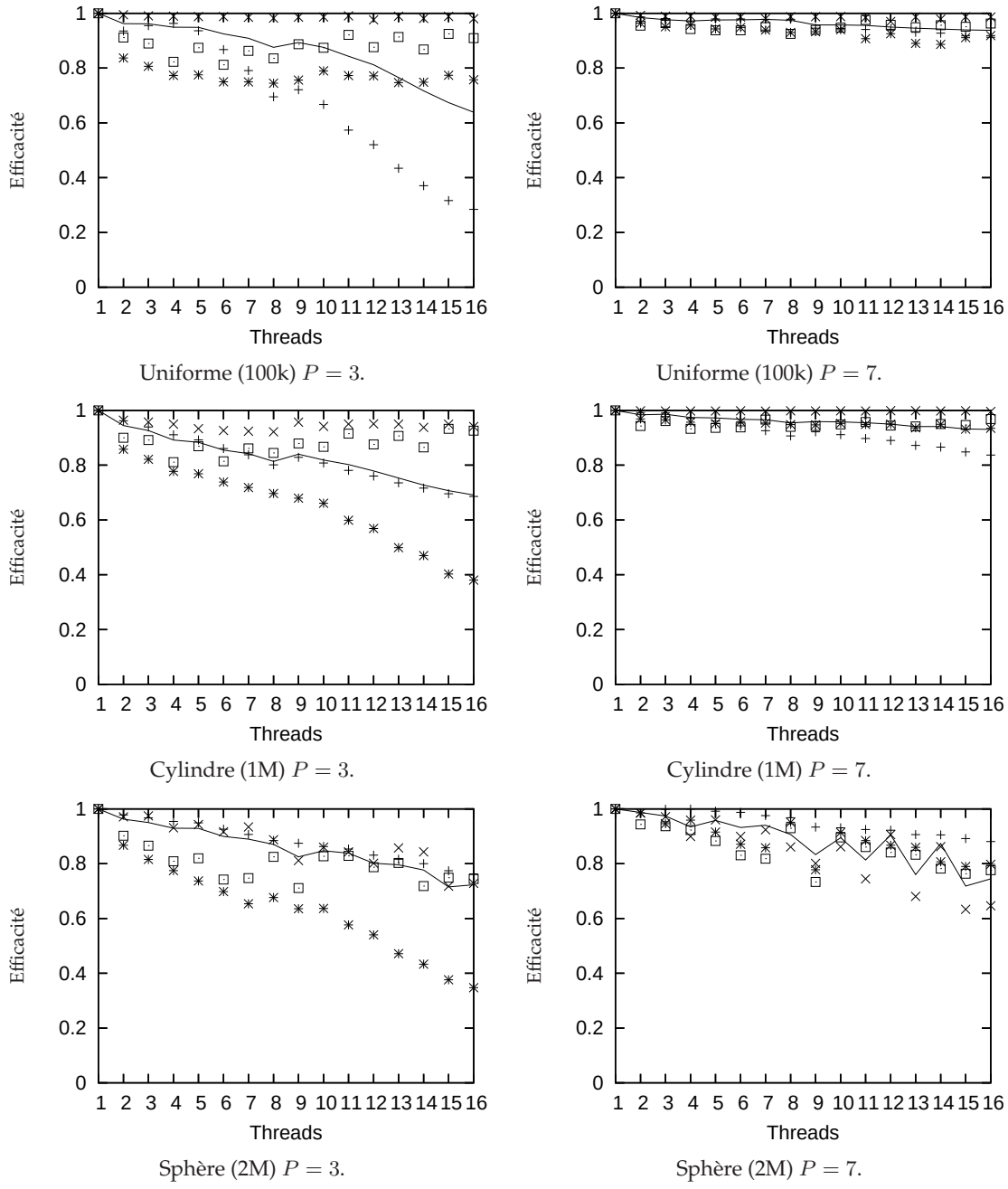
threads	Uniforme (100k)		Cylindre (1M)		Sphère (2M)	
	$P = 3$	$P = 7$	$P = 3$	$P = 7$	$P = 3$	$P = 7$
1	2,46	5,9	22,1	30,1	68,1	102,7
4	0,64	1,5	6,2	7,7	18,3	27,5
8	0,35	0,76	3,4	3,9	9,8	14,2
16	0,24	0,39	2,0	2,0	5,9	8,6

TAB. 9.3 – Temps en secondes pour le « calcul FMB » en mode multi-*thread* : cas du calcul *M2L* par BLAS sans zone uniforme.

Avec zone uniforme. La figure 9.3 présente les efficacités parallèles obtenues avec nos schémas utilisant des BLAS de niveau 3 au sein d'une zone uniforme : le schéma avec *recopies* et le schéma avec un stockage des données par *ligne* ou par *tranche*. Les temps de calcul correspondants sont donnés au tableau 9.2. Lorsque le grain de calcul est faible ($P = 3$), l'efficacité obtenue n'est pas optimale pour le stockage par *ligne* ; le stockage par *tranche* (présenté ici uniquement pour la balle) est alors plus performant grâce à ses matrices plus grandes, et le schéma par *recopies* est lui aussi décomposé plus efficacement entre les différents *threads*. Dès que le grain de calcul est gros ($P = 7$), les efficacités sont alors proches de l'optimal pour tous nos schémas BLAS. Ces conclusions sont valables pour les distributions uniformes comme pour les distributions non uniformes où nous détectons de larges zones uniformes, telles la balle et la barre. Les larges zones uniformes représentent en effet la majeure partie du temps de calcul dans ces distributions.

Sans zone uniforme. Nous présentons en figure 9.4 les résultats obtenus pour des distributions ne présentant pas de zones uniformes assez grandes (distribution uniforme de 100 000 particules), ou ne comportant aucune zone uniforme (cas du cylindre et de la sphère). Les temps de calcul correspondants sont donnés au tableau 9.3. On retrouve là encore les mêmes comportements : à savoir, de très bonnes efficacités lorsque le grain de calcul est gros ($P = 7$), et des efficacités moins bonnes, mais tout à fait satisfaisantes, lorsque ce grain est petit ($P = 3$).

[6] Avec calcul *M2L* classique hors des zones uniformes, voir section 8.4.3.2.



Uniforme (100k) $P = 3$. Uniforme (100k) $P = 7$.

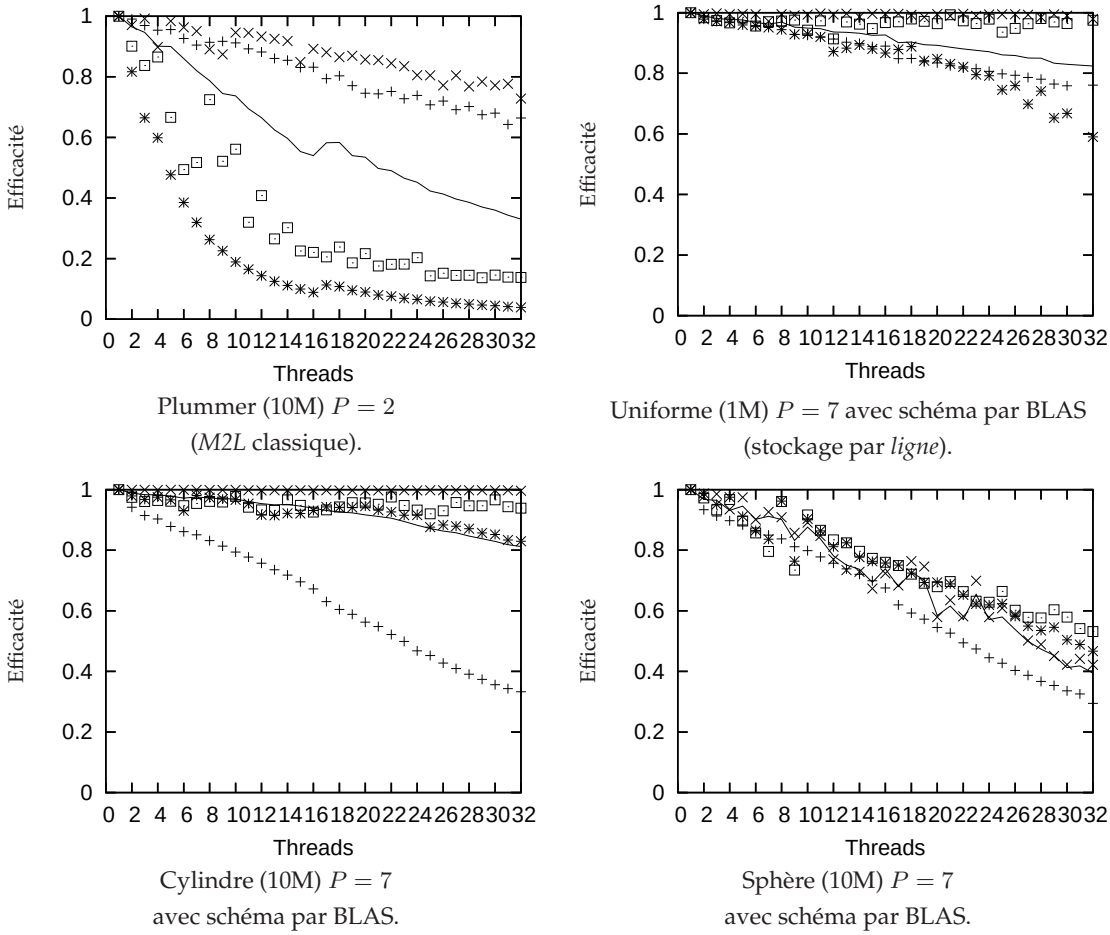
Cylindre (1M) $P = 3$. Cylindre (1M) $P = 7$.

Sphère (2M) $P = 3$. Sphère (2M) $P = 7$.

FIG. 9.4 – Efficacités parallèles en multi-*thread* pour le calcul *M2L* par BLAS sans zone uniforme. Voir légende en figure 9.2.

9.1.3.3 Avec 32 processeurs

Nous présentons finalement à la figure 9.5 quelques tests effectués sur une machine à 32 processeurs avec des simulations plus importantes (sauf dans le cas uniforme). Les temps de calcul correspondants sont donnés dans le tableau 9.4. Nous avons pour cela utilisé un



Plummer (10M) $P = 2$ (*M2L* classique).

Uniforme (1M) $P = 7$ avec schéma par BLAS (stockage par *ligne*).

Cylindre (10M) $P = 7$ avec schéma par BLAS.

Sphère (10M) $P = 7$ avec schéma par BLAS.

FIG. 9.5 – Efficacités parallèles en multi-*thread* avec 32 processeurs. Voir légende en figure 9.2.

threads	Plummer (10M) $P = 2$	Uniforme (1M) $P = 7$	Cylindre (10M) $P = 7$	Sphère (10M) $P = 7$
1	499,2	56,8	385,5	541,5
4	138,5	14,5	98,0	145,3
8	79,3	7,3	49,5	75,1
16	57,8	3,8	25,7	45,8
32	47,3	2,2	14,8	42,9

TAB. 9.4 – Temps en secondes pour le « calcul FMB » en mode multi-*thread* : avec 32 processeurs.

nœud de la machine IBM SP4 localisée au CINES[7] à Montpellier. Ce nœud est composé de 32 processeurs Power4 cadencés à 1,7 Ghz et disposant de 32 ou 64 Go de mémoire.

Certes, les résultats sont généralement un peu moins bons sur l'architecture SP4 que sur l'architecture p575 (ce qui peut s'expliquer par la bande passante mémoire plus faible du Power4), et les performances se dégradent un peu lorsque le nombre de *threads* augmente,

[7]Centre Informatique National de l'Enseignement Supérieur.

mais nos conclusions restent valables, notamment en ce qui concerne l'efficacité parallèle de notre schéma de calcul par BLAS avec un grain suffisant ($P = 7$).

9.1.4 Comparaison en astrophysique avec *falcON*

Nous allons mettre à profit cette version multi-*thread* de notre code pour poursuivre la comparaison avec le code séquentiel *falcON* présentée en section 8.6 dans le cas des simulations astrophysiques. Nous reprenons les conditions dans lesquelles ces comparaisons ont été réalisées dans le cas séquentiel avec néanmoins deux différences.

Premièrement, la machine de test est ici un nœud du cluster IBM p575 (présenté en section 9.1.3). Notre code FMB est compilé avec le compilateur IBM `xlc` (version 7.0) alors que le code *falcON* est compilé avec `g++` (version 3.3.2) car il n'accepte pas `xlc`.

Deuxièmement, nous ne prenons plus ici en compte les temps CPU consacrés à la construction de l'octree (qui était un plus coûteuse pour FMB que pour *falcON*). Comme précédemment, la décomposition statique entre les *threads* n'est pas non plus prise en compte.

Ces deux différences expliquent les rapports légèrement différents entre les temps des deux versions séquentielles par rapport à la section 8.6. On passe ainsi d'un rapport de $6,87$ en section 8.6 dans le cas du modèle de Plummer (à 1 million de particules) à $6,08$ ici, et de $2,88$ à $2,19$ dans le cas uniforme à $500\ 000$ particules. L'écart reste faible et permet donc la comparaison malgré la différence de compilateur.

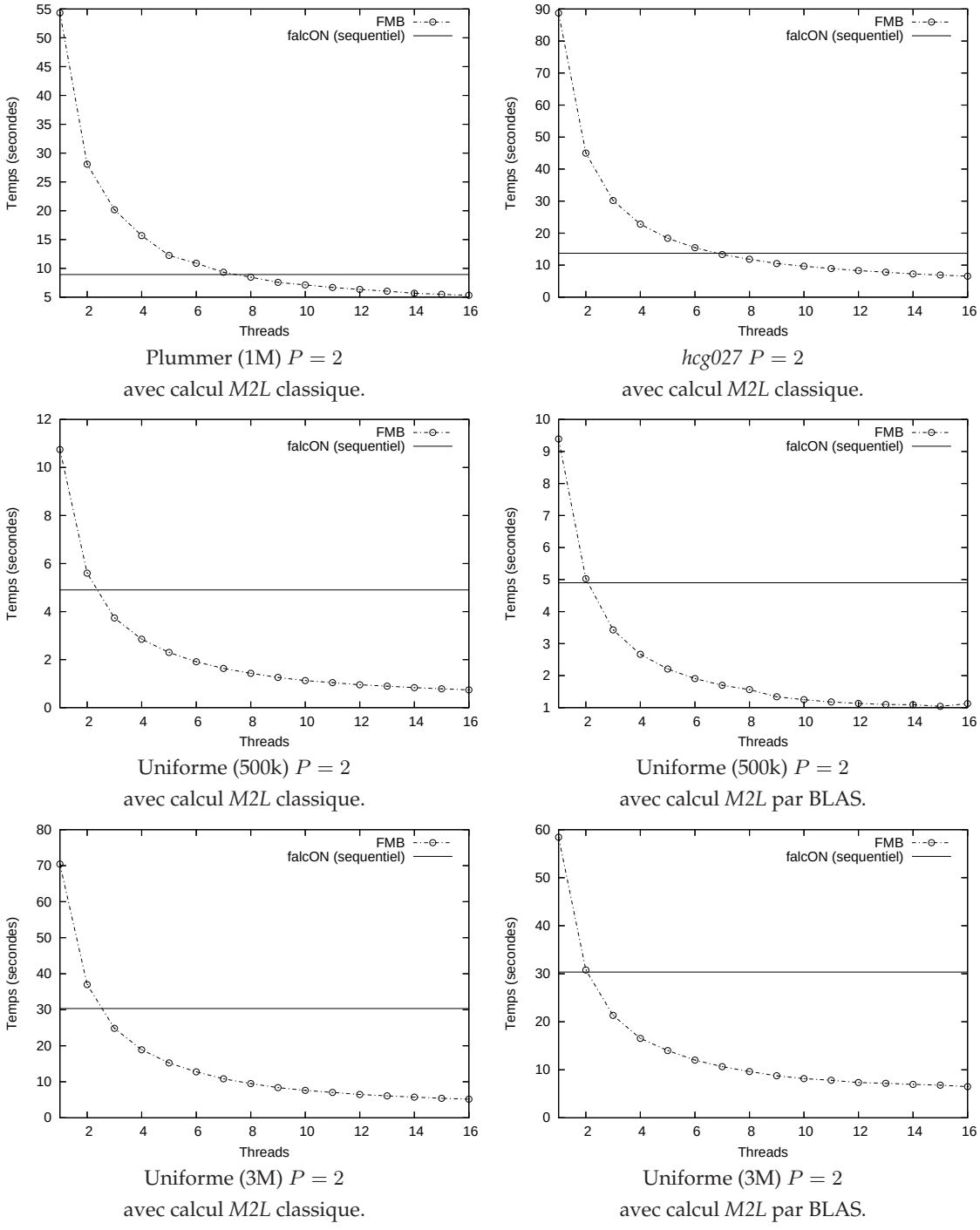
La figure 9.6 présente les résultats de cette comparaison pour deux distributions astrophysiques, ainsi que pour deux cas uniformes qui sont traités avec le calcul *M2L* classique et avec notre schéma *M2L* par BLAS. C'est le stockage par *tranche* qui est présenté ici car il offre les meilleurs performances : le gain est alors de plus de 35 % par rapport au calcul *M2L* classique pour la phase de descente, mais sa répercution sur le coût total varie suivant la hauteur de l'octree utilisée.

Dans le cas des distributions astrophysiques, il nous faut donc 8 processeurs pour obtenir des temps CPU inférieurs ou égaux à ceux de *falcON*. Dans le cas des distributions uniformes, le nombre de processeurs nécessaires tombe à 3, et le recours à notre schéma par BLAS abaisse encore ce nombre à 2. En exploitant entièrement le nœud SMP, notre code permet donc d'améliorer sensiblement les temps de calcul requis pour ces diverses simulations. Nous gagnons ainsi avec 16 *threads* entre 40 et 50 % de temps CPU pour les distributions astrophysiques, et nous divisons par 6 environ les temps de calcul dans les cas uniformes.

9.1.5 Conclusion

En conclusion de cette première partie sur la parallélisation de notre méthode FMM en mode multi-*thread*, les efficacités parallèles obtenues sont très satisfaisantes aussi bien pour la phase du calcul direct (avec prise en compte du principe des interactions réciproques) que pour la phase de descente, ces deux phases constituant la majorité du temps de calcul de la FMM. En ce qui concerne les schémas de calcul par BLAS, les résultats sont aussi très bons dès que le grain de calcul est suffisamment gros (moyennes et hautes précisions), et si les efficacités décroissent avec un grain de calcul plus petit (basses précisions), elles restent là encore tout à fait satisfaisantes (autour de 60 % avec 16 *threads*).

Les cas les plus difficiles sont les simulations astrophysiques, telles le modèle de Plummer, qui présentent des distributions hautement non uniformes et requiert des faibles valeurs pour P ($P = 2$ avec un noyau *M2L* de hauteur simple). Le caractère hautement non



Plummer (1M) $P = 2$ avec calcul *M2L* classique.

hcg027 $P = 2$ avec calcul *M2L* classique.

Uniforme (500k) $P = 2$ avec calcul *M2L* classique.

Uniforme (500k) $P = 2$ avec calcul *M2L* par BLAS.

Uniforme (3M) $P = 2$ avec calcul *M2L* classique.

Uniforme (3M) $P = 2$ avec calcul *M2L* par BLAS.

FIG. 9.6 – Comparaison entre le code *falcON* (séquentiel) et notre code FMB en multi-*thread*.

uniforme dégrade en effet la qualité de notre équilibrage de charge statique, alors que les faibles valeurs de P réduisent le grain de calcul et rendent plus sensible le problème de l'estimation du coût induit par les cellules vides. Nous présenterons au chapitre 10 deux perspectives pouvant améliorer l'efficacité obtenue pour ces simulations astrophysiques :

l'équilibrage de charge dynamique et l'équilibrage de charge corrigé sur plusieurs pas de temps.

Nous allons maintenant paralléliser notre code FMB avec une approche différente : l'approche multi-processus avec une programmation par échanges de messages en se basant sur le standard MPI.

9.2 Mémoire distribuée en mode multi-processus

Avant de présenter les tests de performance de notre version parallèle en mode multi-processus, nous détaillons sa mise en œuvre au niveau de l'algorithmique et de l'implémentation.

9.2.1 Algorithmique et implémentation

Nous nous intéressons ici à une exécution parallèle sur plusieurs processus. Chaque processus disposant de son propre espace d'adressage, des communications sont nécessaires pour échanger des données entre les différents processus. Nous avons donc recours à une programmation par échanges de messages, et nous utilisons pour cela le standard MPI [91] qui assure la portabilité des communications, et pour lequel plusieurs implémentations efficaces sont disponibles.

Nous nous restreignons ici au cas où chaque processus ne comporte qu'un seul *thread* de calcul. Nous attribuons donc un processus à chaque processeur disponible. Nous souhaitons cependant conserver en perspective la possibilité d'étendre notre parallélisation à une version hybride où chaque processus MPI pourrait disposer de plusieurs *threads* de calcul répartis sur plusieurs processeurs (au sein d'un nœud SMP) : comme nous le verrons par la suite, ce couplage MPI-*threads* devrait en effet permettre de lever certaines limites de notre version actuelle.

Comme expliqué au chapitre 3, une parallélisation haute performance en mémoire distribuée nécessite de recouvrir les communications par du calcul, de grouper les envois de petits messages, d'ordonnancer les communications entre les processus, et de préférer des communications dirigées par l'émetteur à des communications dirigées par le récepteur. Nous allons maintenant détailler ces différents points.

9.2.1.1 Décomposition entre processus

Comme en mode multi-*thread*, les cellules de chaque niveau de l'octree sont réparties entre les différents processus, et chaque processus doit traiter, en tant que cellule cible, toutes les cellules appartenant à son *domaine*. Au sein d'un processus les cellules qu'il possède sont dites *locales*, alors que les cellules appartenant à d'autres processus sont dites *distantes*. Une opération mettant en œuvre une cellule cible locale et une cellule source distante impliquera donc une phase de communication pour récuperer les données de la cellule distante. La localité des données est ici cruciale : afin de minimiser les communications pour les opérations *M2L* et *P2P*, qui correspondent à la majeure partie du temps de calcul de la FMM, nous devons faire en sorte que le maximum de membres de la liste d'interaction et de la liste du calcul direct d'une cellule c soient attribués au même processus que la cellule c. Comme nous l'avons justifié au chapitre 3, ceci est obtenu par une décomposition selon l'ordre de Morton ou l'ordre de Hilbert.

Contrairement au mode multi-*thread*, nous ne pouvons plus nous permettre d'utiliser dans le cas du mode multi-processus une décomposition différente pour chaque phase de l'algorithme : nous devons donc établir une unique décomposition de l'octree global entre les différents processus qui assure à la fois l'équilibrage de charge et la localité des données.

La décomposition est effectuée au niveau des feuilles et les niveaux internes de l'octree sont distribués entre les différents processus en se basant sur cette décomposition obtenue au niveau des feuilles ; comme expliqué en section 9.1.2.1, on se base pour cela sur le processus qui possède le fils de numéro 0. Cette méthode a l'avantage de minimiser les communications pour les opérations *M2M* et *L2L* ; nous détaillerons ce point en section 9.2.1.2. Par défaut tous les niveaux $l \geq 2$ sont ainsi décomposés (les niveaux 0 et 1 n'intervenant pas dans la phase de descente du fait des conditions aux limites libres).

Pour cette décomposition entre processus, le coût attribué à chaque feuille doit permettre d'équilibrer la charge de l'ensemble du calcul FMM entre les différents processus. Entre les différentes mesures de coût $\mathcal{C}_{NonVide}$, $\mathcal{C}_{NbPart}$ et $\mathcal{C}_{Liste\_I}$ présentées en section 9.1.2.1, nous choisissons ici $\mathcal{C}_{NbPart}$. Celle-ci donne en effet de meilleurs résultats pour les distributions non uniformes en prenant mieux en compte la variation du nombre de particules d'une feuille à l'autre.

Comme cette décomposition nécessite la construction de l'octree total au sein d'un seul processus, et qu'elle ne dépend que de la hauteur de l'octree utilisée, elle est réalisée au préalable et sauvegardée sur disque. Ainsi au démarrage de la version parallèle multi-processus, chaque processus dispose de son propre fichier de particules à traiter, pour lequel il construira son octree local puis débutera le calcul FMM parallèle.

9.2.1.2 Gestion des communications

La forme des domaines résultant de la décomposition de Morton ou de Hilbert étant comme en multi-*thread* indéfinie, le schéma de communication est totalement irrégulier : le nombre de destinataires et la taille des messages peuvent complètement varier d'un processus à l'autre.

On répartit les communications nécessaires au bon déroulement de la version multi-processus en trois catégories :

- les communications nécessaires au calcul du champ proche (communications du calcul direct) ;
- celles nécessaires aux opérations *M2L* (communications de la liste d'interaction) ;
- et enfin celles nécessaires aux opérations *M2M* et *L2L*.

Les communications du calcul direct font intervenir les particules des cellules sources, alors que les communications de la liste d'interaction et les communications *M2M* et *L2L* font a priori intervenir les développements multipôles ou locaux. Cependant les seuils s^M_{min} et s^L_{min} introduits pour la version adaptative en section 8.1.1 peuvent amener à communiquer des particules à la place des développements dans certains cas. L'implémentation est ici simplifiée par le fait qu'en pratique on a toujours : $s^M_{min} \geq s^L_{min}$.

Pour les communications du calcul direct et de la liste d'interaction, nous allons rassembler les messages de petite taille, correspondant chacun à une cellule donnée et ayant le même destinataire, en un unique message. Nous utilisons pour cela le mécanisme d'empaquetage / dépaquetage (*pack / unpack*) des données offert par MPI. Ce mécanisme implique certes une copie supplémentaire pour chaque donnée envoyée, mais il est bien plus simple qu'un mode « zéro-copie » (où les données sont agencées dès le début correctement en mé-

moire de façon à former les messages à envoyer) : les données à rassembler sont en effet dispersées en mémoire dans notre cas et certaines données doivent être envoyées à plusieurs processus différents. Dans la perspective d'un couplage MPI-*threads*, ce choix éviterait aussi les conflits potentiels entre plusieurs *threads*.

Quant à l'ordonnancement des communications, il est très simple à réaliser : lors de l'envoi des messages, le processus de rang r (parmi NP processus) utilise un ordre circulaire modulo NP en commançant par le processus $r+1$, puis $r+2$, ..., $NP-1$, 0, ..., $r-1$ (et en n'envoyant bien sûr réellement des données que lorsque c'est nécessaire, c'est-à-dire pour les processus qui possèdent des cellules voisines des siennes).

Recouvrement des communications par le calcul. Afin de recouvrir les communications par le calcul, nous entrelaçons les différentes catégories de communication avec les différentes phases de l'algorithme en nous inspirant du code DPMTA (voir [102]). On procède comme suit :

Etape 1 : construction et envoi des messages pour les communications du calcul direct ;

Etape 2 : phase de remontée (incluant les communications *M2M*) ;

Etape 3 : terminaison des communications du calcul direct et traitement des messages reçus ;

Etape 4 : construction et envoi des messages pour les communications de la liste d'interaction ;

Etape 5 : phase du calcul direct ;

Etape 6 : terminaison des communications de la liste d'interaction et traitement des messages reçus ;

Etape 7 : phase de descente (incluant les communications *L2L*) ;

Etape 8 : phase d'évaluation.

Cette approche permet de recouvrir le coût des communications pour le calcul direct et pour la liste d'interaction par des phases de calcul. Plus précisément, nous utilisons en émission des routines de communication MPI non bloquantes et en réception nous choisissons d'utiliser un *thread* supplémentaire, le thread *de réception*, chargé uniquement de la réception des messages (par l'intermédiaire de routines de communication MPI bloquantes). Pour chaque catégorie de communication, nous recevons ainsi dès que possible des messages de n'importe quel destinataire. Ce *thread* de réception a la même priorité que l'unique *thread* de calcul au niveau de l'ordonnanceur du système d'exploitation, et il ne consomme pas de temps CPU entre deux messages. En ce qui concerne l'empaquetage des messages, il est donc réalisé par le *thread* de calcul, qui est aussi chargé de l'envoi des message. Le *thread* de calcul est également chargé du dépaquetage des messages, le *thread* de réception servant uniquement à réceptionner au plus tôt les messsages. En pratique, une file est utilisée pour transmettre les messages du *thread* de réception au *thread* de calcul. Ce type de programmation impose un support total des *threads* au niveau MPI (niveau MPI_THREAD_MULTIPLE), ce qui est de nos jours généralement disponible sur diverses implémentations MPI (MPI IBM, MPICH2 [13]).

Liste d'interaction et liste du champ proche. En ce qui concerne le dilemme du principe des interactions réciproques pour le calcul direct entre deux cellules appartenant à des processus différents (voir section 2.3.3.11), nous choisissons de renoncer à son emploi en mode multi-processus. Ceci nous apparaît en effet plus efficace que la gestion et le surcoût de deux phases de communication pour le calcul direct.

La gestion des communications pour le calcul direct et celle des communications pour la liste d'interaction sont alors similaires : nous devons envoyer les données de chaque cellule source aux processus possédant les membres de sa liste du calcul direct ou de sa liste d'interaction. En effet, comme détaillé au chapitre 3, le choix de notre liste d'interaction basée uniquement sur la géométrie des cellules de l'octree et notre choix de la version adaptative de Nabors *et al.* [93], résulte en une totale symétrie entre la liste d'interaction et la liste d'interaction inverse. Autrement dit, si une opération *M2L* a lieu entre une cellule (source) A et une cellule (cible) B, alors on aura aussi à effectuer une opération *M2L* entre la cellule (source) B et la cellule (cible) A. Et il en va de même pour la liste du calcul direct.

Chaque processus parcourt donc la liste du calcul direct et la liste d'interaction de chaque cellule (non vide) de son domaine afin de déterminer les messages qu'il a à émettre. Pour chaque membre de la liste, si le processus possédant cette cellule est distant, on empaquette les données de cette cellule dans un tampon destiné à ce processus distant. La détermination du processus possédant une cellule donnée est réalisée à partir des intervalles issus de la décomposition entre les processus. Certes, si les cellules distantes sont en fait vides, la communication aura été inutile, mais en procédant ainsi aucune communication préalable n'est nécessaire à l'établissement du schéma de communication, et les communications sont alors entièrement dirigées par l'émetteur.

Comme nous n'envoyons pas de message pour les cellules locales vides (car nous sommes sûrs que de tels messages seront inutiles), certains processus peuvent n'avoir à émettre aucun message pour leur processus voisin : nous devons donc signaler la fin de chaque étape de communication. Chaque processus envoie donc un message (vide) dit de *terminaison* à la fin de la phase de communication du calcul direct et de celle de la liste d'interaction afin d'indiquer aux autres processus qu'ils ne doivent plus attendre un éventuel message de sa part.

Gestion des communications pour les opérations *M2M* et *L2L*. Les communications *M2M* et *L2L* ne représentent qu'une faible part du volume total des communications, mais une contrainte forte pèse sur elles : le délai entre le moment où le développement source est établi dans le processus distant, et le moment où il est nécessaire dans le processus local, est très court. En effet le développement source calculé à un niveau donné pendant la phase de remontée ou de descente est utilisé dès le niveau suivant.

Nous devons donc envoyer au plus tôt ces messages, à savoir dès que le développement source est calculé. Pour cela nous choisissons d'émettre un message pour chaque cellule source et de renoncer au regroupement des petits messages. Nous utilisons ici une particularité des implémentations MPI : la *eager limit*. Les messages dont la taille est inférieure à cette limite sont envoyés *directement* au destinataire sans attendre que celui-ci nous indique qu'il est prêt à recevoir le message (comme c'est le cas dans le mode *rendez-vous* utilisé a priori pour les communications du calcul direct et de la liste d'interaction). Pour les valeurs de P que nous utilisons, la valeur par défaut de cette limite est plus grande que la taille de nos développements multipôles et locaux.

De plus, grâce à notre décomposition basée sur le fils de numéro 0 (voir section 9.2.1.1), le nombre de messages générés pour les opérations *M2M* et *L2L* est faible, ce qui évitera toute saturation des tampons en réception des processus MPI. En effet dans le cas des opérations *M2M*, les seules cellules génératrices de communications sont les cellules situées entre le début du domaine d'un *thread* donné et la première cellule qui est un fils de numéro 0, soit au maximum 7 cellules. Et en réception, la seule cellule susceptible de nécessiter une communication est la dernière cellule du domaine. La situation est inversée pour les communications *L2L*, et dans les deux cas le nombre de messages par processus est faible et leur nombre peut être déterminé localement[8]. A chaque fois, les calculs locaux (qui ne nécessitent pas de communication) sont exécutés avant d'effectuer la réception (bloquante) des messages, et dans le cas de la phase de descente, la présence des nombreuses opérations *M2L* permet de recouvrir plus facilement les communications *L2L*. Enfin la réception des communications *M2M* et *L2L* est exceptionnellement assurée par les *threads* de calcul par souci d'efficacité et de simplicité.

9.2.1.3 Schéma de calcul par BLAS

En ce qui concerne le schéma de calcul par BLAS, chaque processus détecte simplement les zones uniformes disponibles au sein de son domaine en se basant uniquement sur ses cellules locales. La parallélisation de notre schéma par BLAS en mode multi-processus est donc directe. Mais, contrairement au mode multi-*thread* où une même zone uniforme est traitée sans découpage par plusieurs processeurs, les zones uniformes sont ici potentiellement découpées entre les différents processus (chaque processus disposant de son propre espace mémoire) : ainsi lorsque le nombre de processus augmente, la taille des zones uniformes à traiter dans chaque processus diminue, et l'efficacité de nos schémas par BLAS aussi.

A noter que dans le cas des distributions présentant des zones uniformes, nous préférons utiliser le coût $\mathcal{C}_{NonVide}$ pour la décomposition entre processus : ceci permet en effet de mieux découper les zones uniformes, dont la forme est cubique ou rectangulaire ($3D$), entre les différents processus, en particulier lorsque le nombre de processus est une puissance de 2.

9.2.2 Validation expérimentale

Nous allons maintenant présenter les résultats de tests effectués sur la machine IBM p575 (voir section 9.1.3). Le nombre maximal de nœud disponible est 8 (soit $8 \times 16 = 128$ processeurs). Ces nœuds sont reliés par un réseau Federation à 12 Gbit/s, chaque nœud disposant de deux cartes réseau (soit un débit total de 24 Gbit/s). A noter que deux processus MPI situés sur un même nœud communiquent entre eux en utilisant des segments de mémoire partagée, le débit étant ainsi encore plus élevé. En pratique, lorsque le nombre de processus MPI utilisés est inférieur ou égal à 16 nous n'utilisons qu'un seul nœud, et lorsque ce nombre est supérieur à 16 nous utilisons 16 processus par nœud.

Comme pour les tests présentés en mode multi-*thread*, nous utilisons les valeurs de P suivantes : $P = 3$ (noyau *M2L* de hauteur simple) pour les basses précisions, $P = 7$ (noyau *M2L* de hauteur double) pour les moyennes précisions et $P = 15$ (noyau *M2L* de hauteur

[8] A condition d'émettre des messages pour les cellules sources vides pour les communications *M2M* (ceci est inutile pour les communications *L2L* car un fils non vide ne peut pas avoir de père vide). C'est le seul cas où nous devons générer des communications pour des cellules sources vides, et le nombre maximal de ces messages est faible.

Processus	Uniforme (10M) $P = 7$	Plummer (10M) $P = 2$	*hcg027* $P = 2$	Cylindre (10M) $P = 7$	Sphère (10M) $P = 7$
1	2166,0	608,5	90,0	567,1	899,3
8	272,2	80,4	12,1	71,9	122,5
16	142,2	72,1	7,8	38,2	94,9
32	74,3	49,0	5,0	20,4	66,8
64	38,8	29,7	3,4	11,2	51,1
96	26,6	24,4	3,0	8,0	45,2

TAB. 9.5 – Temps en secondes pour le « calcul FMB » en mode multi-processus : cas du calcul *M2L* classique.

double) pour les hautes précisions. Nous omettons parfois le cas $P = 15$, car le grain de calcul est suffisamment gros dans le cas $P = 7$: les performances sont alors équivalentes ou meilleures pour $P = 15$. Dans le cas des simulations astrophysiques, nous utiliserons $P = 2$ (avec un noyau *M2L* de hauteur simple) comme justifié en section 8.6.

En mode multi-processus, l'efficacité parallèle est calculée ainsi :

$$\text{efficacité} = \frac{\text{Temps séquentiel}}{(\text{Temps avec } NP \text{ processus}) \times NP}.$$

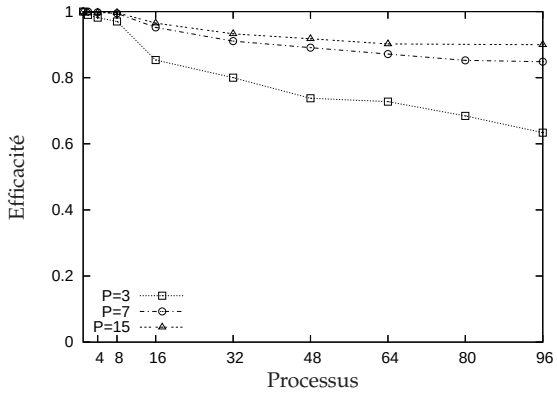
Elle est donnée uniquement pour le « calcul FMB » qui représente l'ensemble des phases de remontée, de descente, de calcul direct et d'évaluation, au sein de notre code FMB. Comme expliqué en section 9.2.1.1, la décomposition des particules entre les processus est réalisée au préalable, et son coût n'est donc pas pris en compte ici.

Nous présentons aussi la part (en pourcentage) des communications dans le temps total. Cette part est calculée en sommant sur tous les processus les temps CPU utilisés pour les communications, puis en la divisant par la somme sur tous les processus du temps CPU utilisé pour le « calcul FMB ». Nous distinguons au sein de ces temps de communication, le temps passé à construire les messages (empaquetage/dépaquetage) et le temps d'envoi, de réception et d'attente des messages.

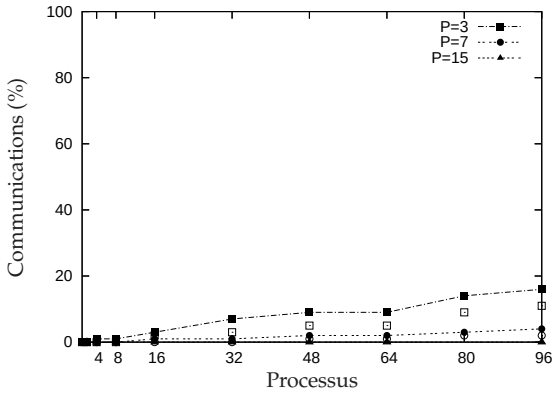
9.2.2.1 Schéma de calcul *M2L* classique

La figure 9.7 présente les efficacités en mode multi-processus pour le calcul *M2L* classique avec divers types de distributions de particules (présentés en section 8.4). Le tableau 9.5 donne une partie des temps correspondants. Comme nous utilisons ici jusqu'à 96 processus, la taille des distributions est plus importante qu'en séquentiel. L'ordre utilisé pour la numérotation des cellules est celui de Morton. Pour la part des communications (colonne de droite), les symboles pleins représentent la part totale des communications (dans le « calcul FMB »), alors que les symboles vides représentent la part due à l'envoi, à la réception et l'attente des messages : la différence entre les deux représente donc la part du temps CPU (pour l'ensemble des processsus) passée à construire les messages (empaquetage/dépaquetage).

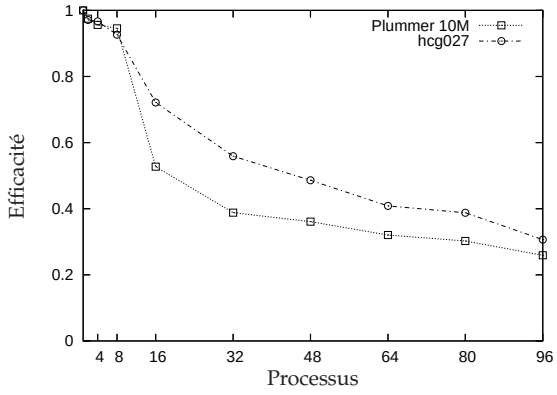
Les efficacités obtenues sont bonnes pour le cas uniforme et le cas du cylindre, voire très bonnes lorsque le grain de calcul est gros (c'est-à-dire que la valeur de P est importante). La part des communications reste alors très faible même pour 96 processus, et n'est sensible que si le grain de calcul est trop petit ($P = 3$). Des tests effectués avec des distributions de tailles



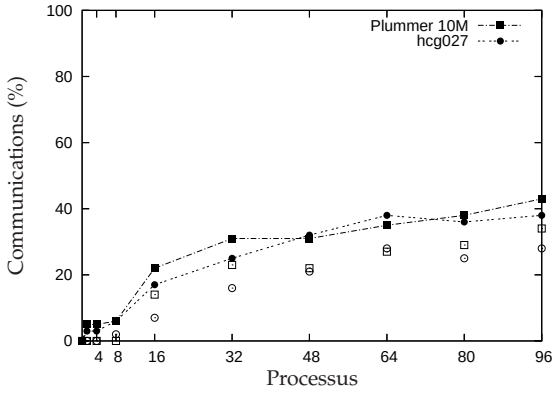
Uniforme (10M).



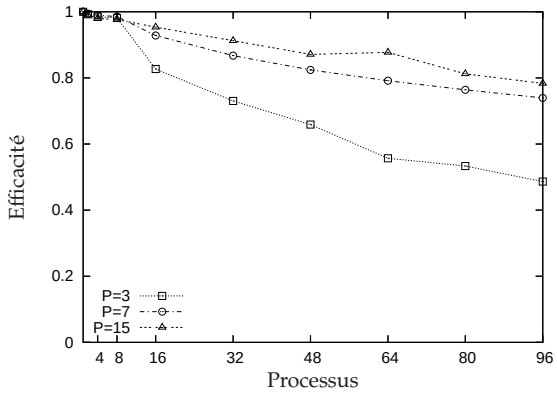
Uniforme (10M).



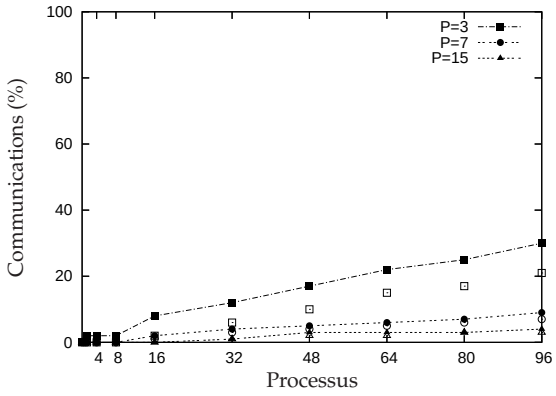
Astrophysique.



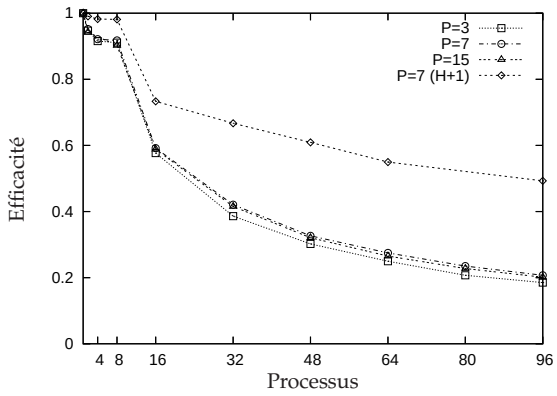
Astrophysique.



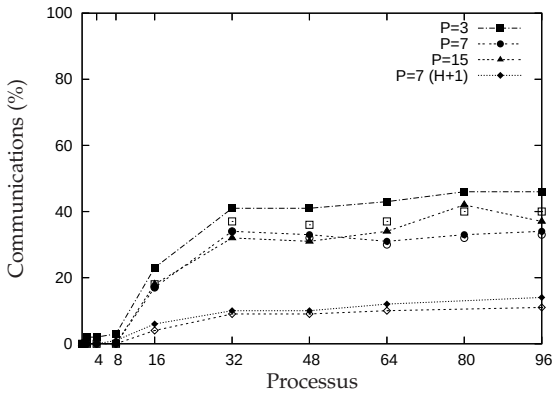
Cylindre (10M).



Cylindre (10M).



Sphère (10M).



Sphère (10M).

FIG. 9.7 – Efficacités parallèles en multi-processus (MPI) pour le calcul *M2L* classique.

différentes montrent par ailleurs que les performaces s'améliorent lorsque le nombre de particules augmente (à nombre de processus constant). En effet la charge de calcul augmente alors plus vite que le volume des communications.

En ce qui concerne les distributions astrophysiques (cas du modèle de Plummer à 10 millions de particules et de *hcg027* en figure 9.7), les efficacités sont moins bonnes qu'en mode multi-*thread* : ceci s'explique par un moins bon équilibrage de charge qui entraîne beaucoup d'attente au niveau des communications (essentiellement pour les communications *M2M* et *L2L*). Ce moins bon équilibrage de charge est dû à notre décomposition statique : comme, contrairement au cas multi-*thread*, nous ne pouvons ici avoir qu'une seule décomposition, il est difficile d'évaluer statiquement le coût d'une cellule de façon à équilibrer chaque phase de la FMM. Le coût $\mathcal{C}_{NbPart}$ utilisé ici est simple et donne de meilleurs résultats que le coût $\mathcal{C}_{NonVide}$, mais ne peut être optimal. Quant au cas test *hcg027*, il comporte trop peu de particules pour être efficacement décomposé sur un grand nombre de processus.

Le cas de la sphère présente lui aussi un déséquilibre de charge (au niveau du calcul direct, à cause de la forte concentration de particules aux pôles) qui induit de longues attentes pour les communications *L2L* et une perte d'efficacité totale. On le vérifie en prenant une hauteur d'octree plus grande (cas « H+1 ») : les feuilles très concentrées en particules sont alors moins nombreuses, et l'équilibrage de charge et l'efficacité s'améliorent sensiblement.

Par ailleurs, comme nous n'avons plus recours au principe des interactions réciproques pour les opérations *P2P* entre deux cellules situées sur deux processus différents, l'efficacité de cette phase ne peut pas rester optimale par rapport au cas séquentiel (où le principe des interactions réciproques est toujours utilisé).

Enfin, on peut remarquer que dans tous les cas la majeure partie du temps consacré aux communications est due aux envois et aux attentes des messages : la part consacrée à l'empaquetage et au dépaquetage des données reste minoritaire, ce qui justifie le choix de ce mode de construction des messages.

9.2.2.2 Décomposition de Morton et décomposition de Hilbert

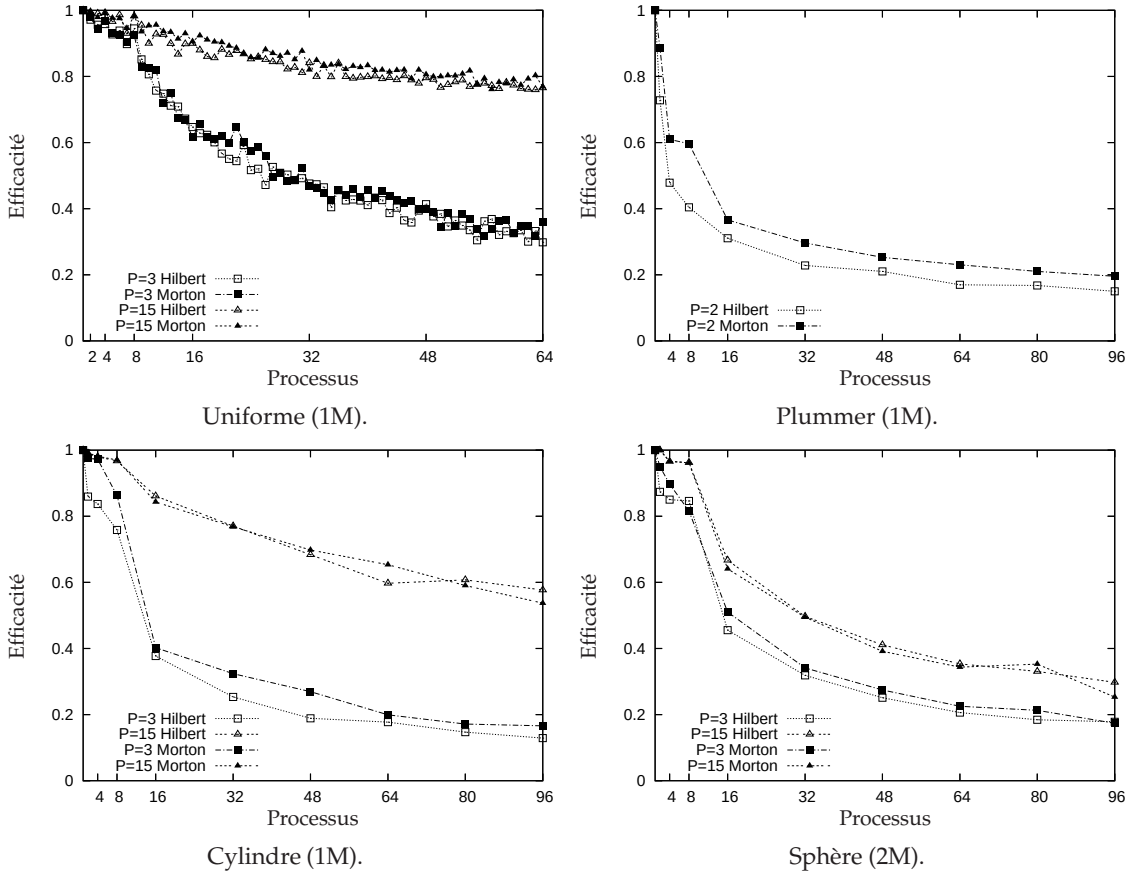
La figure 9.8 présente une comparaison entre la décomposition de Morton et la décomposition de Hilbert pour le schéma de calcul *M2L* classique et pour différents types de distributions. Dans le cas uniforme, nous testons tous les nombres de processus possibles entre 1 et 64 afin d'étudier d'éventuels changements de comportement lorsque le nombre de processus utilisé n'est pas une puissance de 2 ou un multiple de 16.

Aucun gain n'est apporté par la décomposition de Hilbert, et on dénote même une perte d'efficacité due aux surcoûts (en temps CPU ou en espace mémoire) introduits par l'ordre de Hilbert, surtout lorsque la hauteur de l'octree est importante (cas du modèle de Plummer et du cylindre).

9.2.2.3 Schéma de calcul *M2L* par BLAS

Dans cette section nous présentons les performances de notre version parallèle multi-processus pour notre schéma de calcul *M2L* par BLAS, en distinguant comme d'habitude le calcul au sein d'une zone uniforme et le calcul hors d'une zone uniforme.

Avec zone uniforme. La figure 9.9 présente l'efficacité parallèle et la part des communications pour notre schéma par BLAS avec stockage des données par *ligne* sur deux cas uni-



Uniforme (1M). Plummer (1M).

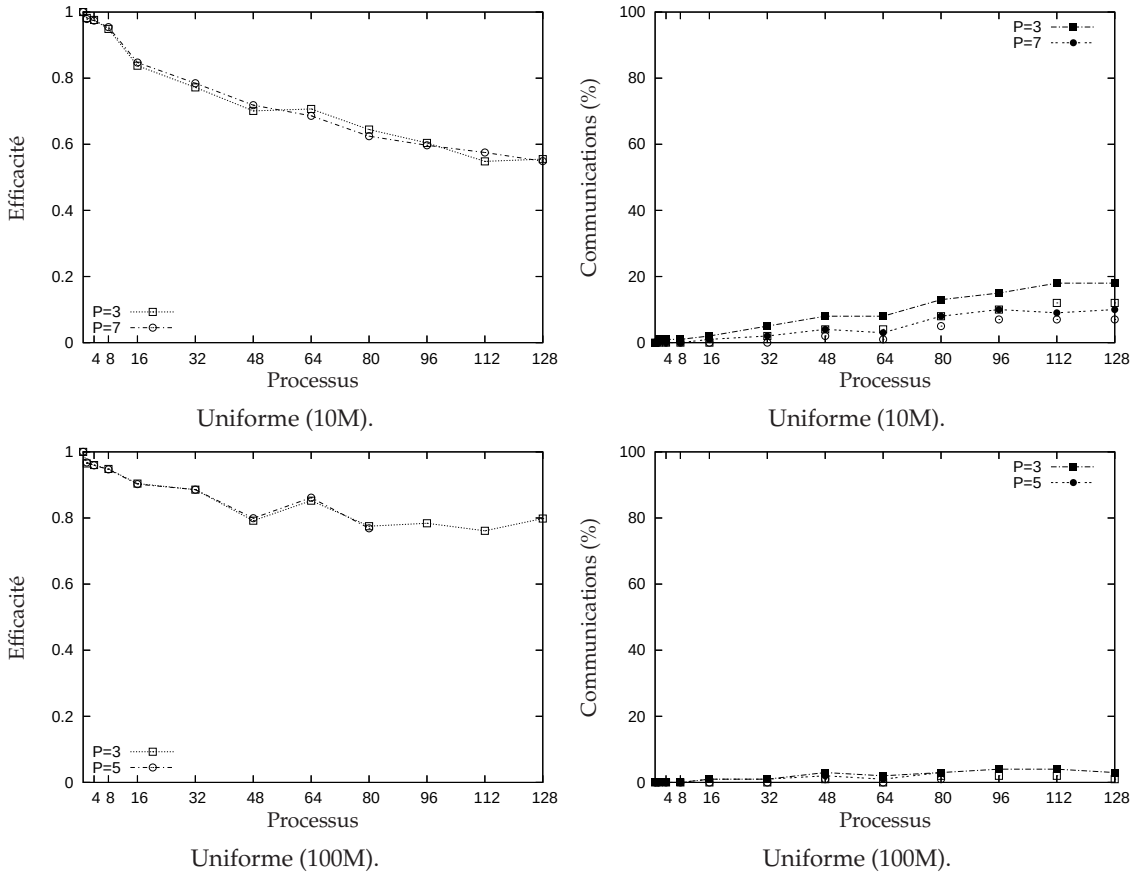
Cylindre (1M). Sphère (2M).

FIG. 9.8 – Comparaison entre la décomposition de Morton et la décomposition de Hilbert (schéma de calcul *M2L* classique).

Processus	Uniforme (10M) $P = 7$	Uniforme (100M) $P = 3$	Barre (30M) $P = 3$
1	595,1	3916,0	859,4
8	77,9	516,3	113,3
16	43,9	270,7	65,4
32	23,7	138,2	36,1
64	13,6	71,8	18,3
128	8,5	38,3	10,3

TAB. 9.6 – Temps en secondes pour le « calcul FMB » en mode multi-processus : cas du calcul *M2L* par BLAS sur une zone uniforme.

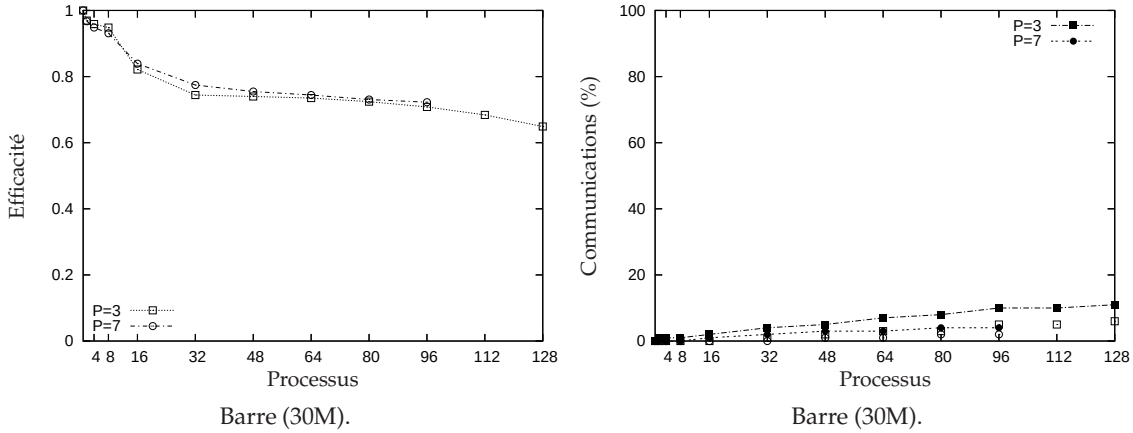
formes, dont un très gros à 100 millions de particules. Les temps de calcul correspondants sont donnés au tableau 9.6. Nous utilisons ici jusqu'à 128 processeurs. Les efficacités parallèles obtenues sont bonnes, voire très bonnes lorsque la taille de la distribution augmente (cas avec 100 millions de particules), et la part des communications reste modérée, voire faible pour le cas à 100 millions. Lorsque le nombre de particules augmente (à nombre de processus constant), nous obtenons donc de meilleurs performances car la charge de calcul augmente alors plus vite que le volume des communications (comme dans le cas du calcul



Uniforme (10M). Uniforme (10M).

Uniforme (100M). Uniforme (100M).

FIG. 9.9 – Efficacités parallèles en multi-processus (MPI) pour le calcul *M2L* par BLAS sur une zone uniforme. Nous utilisons ici le stockage des données par *ligne*. Le cas uniforme à 100 millions de particules est traité avec $P = 3$ (noyau *M2L* de hauteur simple), et $P = 5$ (noyau *M2L* de hauteur double). Voir section 9.2.2.1 pour la légende des communications.



Barre (30M). Barre (30M).

FIG. 9.10 – Efficacités parallèles en multi-processus (MPI) pour le calcul *M2L* par BLAS au sein d'une zone uniforme : cas d'une barre à 30 millions de particules. Nous utilisons ici le stockage des données par *ligne*. Voir section 9.2.2.1 pour la légende des communications.

Processus	Cylindre (10M) $P = 7$	Sphère (10M) $P = 7$
1	431,9	597,2
8	54,8	83,6
16	30,1	68,6
32	16,6	54,0
64	10,1	45,1
96	7,7	41,6

TAB. 9.7 – Temps en secondes pour le « calcul FMB » en mode multi-processus : cas du calcul *M2L* par BLAS sans zone uniforme.

M2L classique). D'autre part, avec de plus grandes distributions de particules, nous disposons pour notre schéma par BLAS de zones uniformes plus grandes et nous gagnons donc en efficacité au niveau des routines BLAS (voir chapitre 6), ce qui améliore donc encore les temps CPU.

Le stockage par *tranche* (non présenté ici) obtient les mêmes performances. Le schéma avec *recopies* (non présenté ici) est lui plus performant : cependant ceci est dû uniquement à son temps séquentiel plus lent (voir section 5.4.2) et les temps CPU avec un grand nombre de processus sont au final équivalents.

Par ailleurs, on note un gain de performance pour 64 processeurs (surtout dans le cas à 100 millions) qui est dû au fait que les processus se partagent alors exactement la zone uniforme cubique en 64 cubes de même taille. Les autres puissances de 2 (16, 32 ou 128 processeurs) aménent à des découpages en rectangles $3D$ réguliers et présentent donc aussi des performances légèrement meilleures que dans les cas restants (48, 80, 96 ou 112 processeurs).

Enfin, les mêmes performances sont obtenues avec notre détection des zones uniformes comme le montre le cas de la barre à 30 millions de particules en figure 9.10 et au tableau 9.6.

Sans zone uniforme. En l'abscence de zone uniforme, l'efficacité de notre schéma par BLAS varie suivant la distribution des particules comme le montre la figure 9.11 et le tableau 9.7. Nous retrouvons ici les conclusions du calcul *M2L* classique : alors que de bonnes performances sont obtenues dans le cas du cylindre, le cas de la sphère est plus problématique à cause du déséquilibre de charge de la phase de calcul direct qui génère beaucoup d'attente dans les communications *L2L*. Comme pour le *M2L* classique, ceci se vérifie en augmentant la hauteur (cas « H+1 ») : l'efficacité parallèle est alors sensiblement améliorée.

On peut aussi remarquer dans le cas du cylindre que les performances sont légèrement moins bonnes que pour le calcul *M2L* classique : ceci provient du plus faible grain de calcul dans le cas des BLAS.

Enfin, on note que les efficacités jusqu'à 8 processus sont parfois très légèrement supérieures à 1 ! Ceci est vraisemblablement dû aux « effets BLAS » : la décomposition effectuée avec quelques processus modifie en effet le nombre de colonnes dans nos produits matriciels (voir sections 5.3.1 et 8.3.1), ce qui influe (ici favorablement) sur l'efficacité des routines BLAS.

En ce qui concerne les performances pour des valeurs de P plus grandes ($P = 15$ avec un noyau *M2L* de hauteur double par exemple) elles sont équivalentes à celles présentées ici (avec ou sans zone uniforme) : nous avons cependant des problèmes de scalabilité mémoire

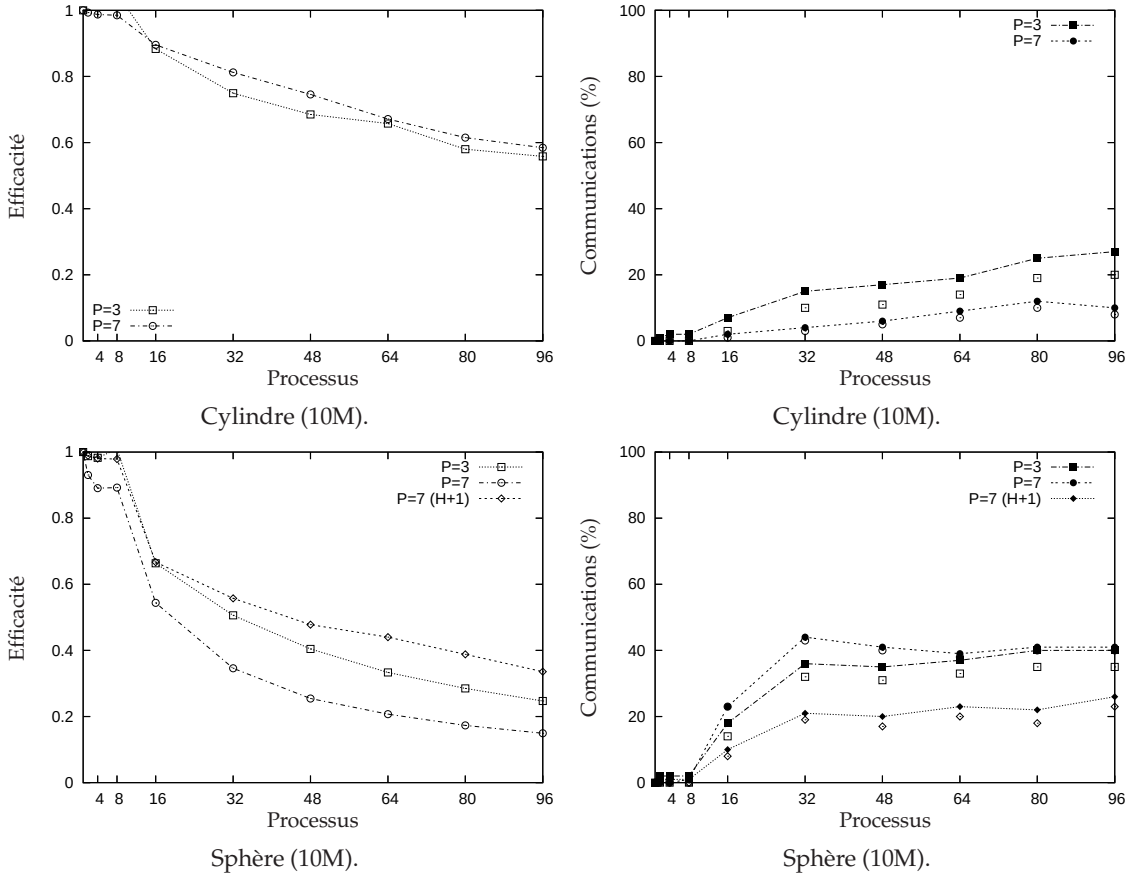


FIG. 9.11 – Efficacités parallèles en multi-processus (MPI) pour le calcul *M2L* par BLAS sans zone uniforme. Voir section 9.2.2.1 pour la légende des communications.

pour les plus grandes valeurs de P (avec un noyeau *M2L* de hauteur double), notamment à cause des matrices de transfert *M2L*. Dans le cas du noyau *M2L* de hauteur double, la taille de ces matrices pleines a en effet tendance à croître fortement avec P (voir sections 6.1 et 5.5) et ceci est aggravé par le fait que nous stockons désormais ces matrices pour tous les niveaux de l'octree à la fois afin d'améliorer les efficacités en mode multi-*thread*.

9.2.3 Conclusion

En conclusion de cette partie sur le mode multi-processus, les efficacités parallèles obtenues dans le cas uniforme comme dans le cas non uniforme (cas du cylindre) sont bonnes, voire très bonnes si le grain de calcul est suffisamment gros. Le fait que la part des communications reste faible dans ces cas là justifie le choix de notre FMM adaptative présenté au chapitre 3 ainsi que les choix effectués dans cette section pour la parallélisation en mode multi-processus : construction (respectivement traitement) des messages par empaquetage (respectivement dépaquetage), modifications algorithmiques pour le recouvrement des communications par le calcul, recours à un *thread* de réception dédié, communications dirigées par l'émetteur, et enfin envoi au plus tôt des communications *M2M* et *L2L*.

Les moins bonnes performances obtenues dans le cas de distributions plus irrégulières

de particules (simulations astrophysiques et cas de la sphère) sont quant à elles dues à un moins bon équilibrage de charge qui génère beaucoup d'attentes au niveau des communications pour les phases de remontée et de descente (communications *M2M* et *L2L*). Nous présenterons au chapitre 10 deux perspectives pouvant lever les limites de notre parallélisation multi-processus : l'équilibrage de charge corrigé sur plusieurs pas de temps et le couplage MPI-*threads*.

Chapitre 10

Bilan et perspectives

En conclusion, nous présentons ici un bilan de nos contributions avant d'ouvrir plusieurs perspectives de recherche.

Bilan des contributions de cette thèse

Nos contributions ont porté sur la « méthode multipôle rapide » (*Fast Multipole Method*, ou FMM) pour des simulations en astrophysique et en dynamique moléculaire dans le cadre de l'équation de Poisson. Nos résultats s'appliquent à toutes les distributions de particules, uniformes ou non, et la gamme de précisions essentiellement visées va de 10^{-2} pour les basses précisions à 10^{-7} pour les moyennes précisions. Dans ce cadre, la FMM a été améliorée sur plusieurs plans : sa borne d'erreur, son algorithmique, son implémentation haute-performance et sa parallélisation.

Borne d'erreur : nous avons distingué deux expressions de l'opérateur *M2L* et nous avons prouvé que le *noyau* M2L *de hauteur simple*, contrairement au noyau de *hauteur double* respecte une borne d'erreur précise. En pratique, le noyau *M2L* de hauteur double est cependant plus efficace. Il est donc toujours utilisé, excepté pour les plus basses précisions où l'on préférera celui de hauteur simple.

Schéma de calcul *M2L* par BLAS : nous avons introduit pour les deux hauteurs de noyaux *M2L* une approche matricielle basée sur les routines BLAS (Basic Linear Algebra Subprograms) inédite pour les développements en harmoniques sphériques de la FMM. L'emploi de routines BLAS de niveau 3, plus performantes, a été rendu possible par un schéma avec *recopies*. Des stockages de données spéciaux par *ligne* ou par *tranche* nous ont ensuite permis d'éviter au sein d'une zone uniforme le surcoût dû à ces recopies (en particulier pour les basses précisions).

Algorithme de la FMM adaptative : nous avons justifié le choix de la version adaptative de Nabors *et al.* [93] par rapport à celle de Cheng *et al.* [34]. Cette version adaptative de la

FMM a été améliorée au niveau des seuils portant sur la construction des développements multipôles et locaux, et nous avons montré que le traitement spécial des cellules non adaptatives était en fait contre-productif.

Octree avec indirection : une nouvelle structure de données a été introduite, *l'octree avec indirection*. Elle permet de traiter des octrees hautement déséquilibrés avec des hauteurs supérieures à 10, tout en conservant une implémentation efficace des distributions uniformes.

Détection des zones uniformes : notre schéma par BLAS a ensuite été étendu aux distributions non uniformes en combinant les routines BLAS de niveau 3, celles de niveau 2 et le calcul *M2L* classique, et une détection des zones uniformes a été mise en place afin d'améliorer l'efficacité de calcul notamment pour les basses précisions.

Comparaison des schémas de calcul *M2L* : l'implémentation efficace du calcul *M2L* classique et des schémas à base de Transformée Rapide de Fourier (FFT) et de rotations a permis d'établir des comparaisons minutieuses entre ces schémas et notre schéma par BLAS. Le surcoût mémoire de la FFT par blocs et le fait que des instabilités numériques subsistent malgré le schéma par blocs ont ainsi été mis en évidence. Pour la gamme de précisions qui nous intéressent, et pour des distributions uniformes et non uniformes de particules, notre schéma par BLAS apparaît comme le plus performant lorsque l'on considère les temps de calcul, les besoins mémoire et la stabilité numérique. A l'aide du code FMMPART3D, nous avons pu comparer notre schéma par BLAS au schéma de calcul à base d'ondes planes : ceci constitue la première comparaison de ce schéma par ondes planes avec une autre amélioration de l'opérateur *M2L*. Toujours dans le cadre des précisions qui nous intéressent, le gain apporté par les routines BLAS est alors supérieur ou au moins égal à celui obtenu avec les ondes planes.

Comparaison de codes en astrophysique : dans le cadre des simulations astrophysiques, nous avons montré qu'une implémentation efficace de la FMM s'avère plus rapide que l'algorithme de Barnes & Hut, contrairement aux résultats disponibles dans la littérature. Notre code séquentiel est par contre plus lent que le code *falcON*, spécialisé exclusivement dans ce type de simulation, le rapport entre les deux codes étant toutefois plus faible que celui présenté dans la littérature.

Parallélisation en mode multi-*thread* : une première parallélisation en mode multi-*thread* a été mise en place grâce à une décomposition statique basée sur l'ordre de Morton ou de Hilbert, et grâce au maintien avec un minimum de conflits du principe des interactions réciproques. Nous avons aussi gérer de manière efficace les accès concurrents et les dépendances temporelles. De très bonnes efficacités parallèles ont ainsi été obtenues pour le calcul *M2L* classique et pour notre schéma par BLAS, les cas les plus difficiles étant les distributions hautement non uniformes traitées en basse précision (c'est-à-dire avec un faible grain de calcul). Notre code ainsi parallélisé offre alors des gains importants en temps CPU par rapport au code séquentiel *falcON*.

Parallélisation en mode multi-processus : enfin, une deuxième parallélisation en mode multi-processus a validé le choix de notre FMM adaptative et notre gestion des communications (construction, respectivement destruction, des messages par empaquetage, respectivement dépaquetage, recouvrement des communications par le calcul, recours à un *thread* de réception dédié, communications dirigées par l'émetteur, et envoi au plus tôt des communications *M2M* et *L2L*). Les efficacités parallèles obtenues

pour des distributions de particules uniformes ou non, sont alors bonnes voire très bonnes si le grain de calcul est suffisamment gros. L'équilibrage de charge est cependant insuffisant pour les distributions les plus irrégulières (simulations astrophysiques et cas de la sphère).

Perspectives et futurs travaux

Plusieurs perspectives de recherche peuvent dès à présent être envisagées. Dans l'immédiat, les travaux les plus intéressants à mettre en œuvre concernent les limites de notre parallélisation.

En ce qui concerne le mode multi-*thread*, les simulations astrophysiques sont moins efficacement parallélisées car elles correspondent à des distributions de particules hautement non uniformes et ne requièrent que des basses précisions, d'où un faible grain de calcul. Deux perspectives s'offrent à nous pour améliorer l'efficacité dans ce cas.

- On peut tout d'abord mettre en place un équilibrage de charge dynamique à l'aide d'un système maître-esclaves. Le maître, exécuté par un *thread* supplémentaire, distribue alors à la volée des paquets de cellules cibles à traiter en tenant compte des dépendances temporelles et des dépendances de données. La taille des paquets devra être suffisamment conséquente pour que le surcoût introduit par le maître soit modéré (notamment en ce qui concerne les cellules vides) et pour qu'il ne constitue pas un goulot d'étranglement. De plus, un ordre dans la distribution des paquets devra être établi afin de minimiser les conflits du calcul direct (recours au principe des interactions réciproques) aussi efficacement qu'avec notre équilibrage de charge statique.
- Mais on peut aussi envisager d'équilibrer la charge de calcul entre les *threads* sur plusieurs pas de temps. En se basant sur des informations collectées au pas de temps précédent, ou moyennées sur plusieurs pas de temps, on peut corriger les décompositions que nous avons établies afin de réduire les déséquilibres de charge et ainsi améliorer l'efficacité parallèle [102] [110].

L'établissement des différentes décompositions statiques entre les *threads* pourra par ailleurs être partiellement parallélisé afin d'en réduire le coût. Seul le parcours de la courbe de Morton ou de Hilbert est en effet intrinsèquement séquentiel, le calcul des coûts de chaque cellule, et du coût total à un niveau donné, pouvant lui aisément être effectué en parallèle.

En ce qui concerne le mode multi-processus, l'équilibrage de charge insuffisant obtenu dans le cas des distributions irrégulières de particules peut lui aussi être corrigé en se basant sur les pas de temps précédents. Cette correction devrait cependant être réalisée moins souvent qu'en mode multi-*thread* car elle nécessite des communications en mode multi-processus : il faut en effet rassembler les coûts de toutes les feuilles au sein d'un même processus (opération de type `Gather`), puis redistribuer certaines particules.

On peut aussi envisager d'exploiter les meilleures performances du mode multi-*thread* pour ces distributions plus irrégulières en couplant le mode multi-*thread* et le mode multi-processus. Ceci peut être réalisé à l'aide d'une programmation hybride couplant le standard MPI et les *threads* POSIX et peut permettre d'exploiter efficacement les grappes de nœuds SMP : on place alors un processus par nœud et ce processus contient autant de *threads* de calcul qu'il y a de processeurs par nœud (en conservant un unique *thread* de réception par processus). Ce couplage MPI-*threads* est une perspective intéressante car il permettra :

- de réduire les communications *M2M* et *L2L* et les attentes qui leur sont liées ;
- d'avoir plus souvent recours au principe des interactions réciproques et donc d'augmenter l'efficacité de la phase du calcul direct ;
- d'utiliser des décompositions différentes pour chaque phase de la FMM entre les différents *threads* d'un même processus, et ainsi de mieux équilibrer la charge de calcul sur les différents processeurs, notamment dans le cas des distributions irrégulières avec un faible grain de calcul (basses précisions) ;
- de n'avoir qu'une seule instance des matrices de transfert *M2L* dans la mémoire d'un nœud SMP ce qui améliorera la scalabilité mémoire pour les hautes précisions ;
- et enfin de limiter le découpage des zones uniformes en réduisant le nombre total de processus : ceci permettra de traiter de plus grandes zones uniformes et augmentera donc les performances de nos schémas par BLAS.

Nos choix effectués pour la parallélisation en mode multi-processus doivent permettre de mettre en place ce couplage MPI-*threads* rapidement et efficacement. Chaque *thread* de calcul est ainsi chargé de l'émission (empaquetage et envoi) des données des cellules qui lui sont attribuées, et la centralisation des réceptions au niveau de l'unique *thread* de réception évite les conflits. On aura alors un schéma de communication de type « *thread* vers processus ». Si de plus plusieurs cartes réseau sont disponibles, les différents *threads* de calcul pourront les utiliser concurremment en émission, et on pourra éventuellement ajouter d'autres *threads* de réception.

Outre l'introduction de section critique (au niveau de la structure de l'octree ou des communications MPI), le problème majeur d'un couplage MPI-*threads* concerne la détérioration de l'équilibrage de charge entre les différents *threads* d'un même processus du fait de la gestion des communications. L'empaquetage et le dépaquetage des messages, ainsi que leur envoi, sont en effet à la charge des *threads* de calcul. Or le volume de données à émettre pourra complètement varier d'un *thread* de calcul à l'autre, d'où un certain déséquilibre difficile à éviter avec une décomposition statique. Comme dans les perspectives du mode multi-*thread*, un équilibrage dynamique, ou une correction de l'équilibrage de charge sur plusieurs pas de temps, devraient permettre d'améliorer l'équilibrage de charge dans ce cas.

A terme, l'exécution de notre code FMB sur plusieurs pas de temps sera aussi requise afin d'intégrer notre travail au sein d'un code complet d'astrophysique ou de dynamique moléculaire. Les mises à jour régulières de notre octree avec indirection, dues au mouvement des particules, ne devraient cependant pas poser de problème. La mise en place de conditions aux limites périodiques pourra alors être utile pour la dynamique moléculaire. En ce qui concerne les simulations astrophyiques, notre code parallèle permet déjà d'obtenir des gains importants en temps CPU sur le code séquentiel *falcON* : seule une version parallèle efficace de *falcON* serait à même de dépasser ces performances, mais elle reste pour l'instant compliquée à mettre en œuvre à cause de la difficiulté de la prévision des communications dans le cadre d'une parallélisation sur des architectures à mémoire distribuée.

Enfin, il peut être intéressant d'appliquer notre approche matricielle de la FMM, et ses routines BLAS, à d'autres types de potentiels, comme le potentiel de Lennard-Jones en dynamique moléculaire, ou à d'autres types de simulations : simulation en électromagnétisme (avec une formulation intégrale des équations de Maxwell ou de Helmholtz), en calcul de capacitance, ou encore en modélisation d'objets $3D$.

Annexes

Annexe A

Harmoniques sphériques

Nous introduisons ici les harmoniques sphériques, qui sont les bases des développements multipôles et locaux utilisés dans la FMM, ainsi que les fonctions associées de Legendre sur lesquelles elles reposent.

A.1 Fonctions associées de Legendre.

Dans un espace à trois dimensions, nous utilisons la convention suivante, couramment utilisée en physique, pour les coordonnées sphériques : θ désigne la coordonnée colatitudinale (avec $\theta \in [0, \pi]$) et ϕ la coordonnée longitudinale (avec $\phi \in [0, 2\pi[$, ϕ correspond à l'angle polaire des coordonnées polaires) comme représenté en figure A.1.

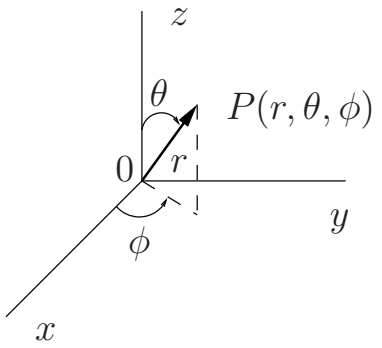


FIG. A.1 – Coordonnées sphériques du point $P(r, \theta, \phi)$.

Nous définissons tout d'abord les polynômes de Legendre grâce à la formule de Rodrigues :

$$P_l(x) = \frac{1}{2^l l!} \frac{d^l}{dx^l} \left(x^2 - 1\right)^l \qquad \forall\, l \geq 0.$$

Ces polynômes vérifient :

$$|P_l(x)| \leq 1 \qquad \forall l \geq 0 \quad \text{et} \quad \forall x \in [-1, 1].$$

Les fonctions associées de Legendre P_l^m sont définies à partir des polynômes de Legendre par :

$$P_l^m(x) = (-1)^m(1-x^2)^{m/2}\frac{d^m}{dx^m}P_l(x) \qquad \forall\,(l,m)\in\mathbb{N}^2 \quad with \quad 0\leq m\leq l.$$

Nous étendons la définition de ces fonctions aux valeurs négatives de m (avec toujours $|m|\leq l$) ainsi :

$$P_l^{-m}(x) = (-1)^m\frac{(l-m)!}{(l+m)!}P_l^m(x). \tag{A.1}$$

Les propriétés suivantes permettent un calcul efficace de ces fonctions :

$$(l-m)P_l^m = x(2l-1)P_{l-1}^m - (l+m-1)P_{l-2}^m,$$

$$P_m^m = (-1)^m(2m-1)!!(1-x^2)^{m/2},$$

où $n!!$ désigne le produit de tous les nombres impairs inférieurs ou égaux à n. De plus nous avons :

$$P_{m+1}^m = x(2m+1)P_m^m.$$

Voici les premières fonctions associées de Legendre :

$P_0^0(x)=1$	$P_1^0(x)=x$	$P_2^0(x)=\frac{1}{2}(3x^2-1)$	$\dots$	$\dots$
	$P_1^1(x)=-(1-x^2)^{\frac{1}{2}}$	$P_2^1(x)=-3x(1-x^2)^{\frac{1}{2}}$	$P_3^3(x)=-15(1-x^2)^{\frac{3}{2}}$	$P_4^3(x)=-105x(1-x^2)^{\frac{3}{2}}$
		$P_2^2(x)=3(1-x^2)$		$P_4^4(x)=105(1-x^2)^2$
				$\dots$

Enfin, nous avons la propriété de parité suivante :

$$P_l^m(-x) = (-1)^{m+l}P_l^m(x).$$

A.2 Harmoniques sphériques.

Avec les fonctions associées de Legendre P_l^m étendues aux valeurs négatives de m, comme défini à l'équation (A.1), une définition courante de l'harmonique sphérique de *degré* l et d'*ordre* m, avec $l\geq 0$ et $-l\leq m\leq l$ est :

$$\Upsilon_l^m(\theta,\phi) = \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-m)!}{(l+m)!}}P_l^m(\cos\theta)e^{im\phi}.$$

Cependant, comme mentionné dans [60], le facteur de normalisation $\sqrt{\frac{(2l+1)}{4\pi}}$ peut être écarté dans le calcul de la FMM. Par conséquent, nous considérons uniquement des harmoniques sphériques définies sans ce facteur de normalisation. De plus, afin de simplifier les formules des opérateurs utilisés dans la FMM (voir section 2.3.3.2), nous introduisons ϵ_m défini par :

$$\epsilon_m = \begin{cases} (-1)^m & \text{si } m\geq 0, \\ 1 & \text{sinon,} \end{cases}$$

de telle sorte que nos harmoniques sphériques (non normalisées) soient définies par :

$$Y_l^m(\theta,\phi) = \epsilon_m\sqrt{\frac{(l-m)!}{(l+m)!}}P_l^m(\cos\theta)e^{im\phi}, \tag{A.2}$$

ce qui peut aussi être écrit (sans les P_l^m d'ordres négatifs) :

$$Y_l^m(\theta,\phi) = (-1)^m \sqrt{\frac{(l-|m|)!}{(l+|m|)!}} P_l^{|m|}(\cos\theta) e^{im\phi}.$$

Remarque A.1. *Cette définition correspond à celle utilisée par Epton & Dembart [52].*

On peut noter que, lorsqu'elles sont normalisées, les harmoniques sphériques forment une base orthonormale pour les fonctions $f(\theta,\phi) \in \mathbb{R}$. Nous soulignons de plus que nos harmoniques sphériques sont considérées comme identiquement nulles pour $l < 0$ ou $|m| > l$.

Par ailleurs, nous avons :

$$Y_l^0(\theta,\phi) = P_l(\cos\theta),$$

et aussi la propriété suivante de symétrie entre deux harmoniques d'ordres opposés :

$$Y_l^{-m} = \overline{Y}_l^m \tag{A.3}$$

où $\overline{z}$ désigne le conjugué complexe de $z \in \mathbb{C}$.

Lorsqu'on considère un vecteur $\mathbf{Z}$ de coordonnées sphériques $\mathbf{Z} = (r,\theta,\phi)$ et son vecteur opposé $-\mathbf{Z}$, dont les coordonnées sphériques s'écrivent $-\mathbf{Z} = (r,\pi-\theta,\pi+\phi)$, nous avons de plus :

$$Y_l^m(\pi-\theta,\pi+\phi) = (-1)^l Y_l^m(\theta,\phi).$$

Pour terminer, nous présentons un moyen de calculer la dérivée partielle d'une harmoniques sphérique par rapport à θ, l'expression de la dérivée partielle par rapport à ϕ étant triviale. La dérivée partielle d'une harmonique sphériques par rapport à θ, à savoir $\frac{\partial Y_j^k(\theta,\phi)}{\partial\theta}$, se déduit directement de la dérivée des fonctions associées de Legendre. Celle-ci peut être calculée grâce à la relation de récurrence suivante :

$$\frac{dP_l^m(\cos\theta)}{d\theta} = \frac{l\cos\theta P_l^m(\cos\theta) - (l+m)P_{l-1}^m(\cos\theta)}{\sin\theta} \qquad \forall\, l \geq 1, \quad \forall\, |m| \leq l-1.$$

Et pour $m = l$:

$$\frac{dP_l^l(\cos\theta)}{d\theta} = \frac{l\cos\theta P_l^l(\cos\theta)}{\sin\theta}.$$

Ces deux équations peuvent être prouvées grâce à la relation suivante :

$$P_l^{m+1}(z) = (z^2-1)^{-\frac{1}{2}} \left((l-m)\, z P_l^m(z) - (l+m)\, P_{l-1}^m(z) \right),$$

tirée du livre [6], chapitre 8, intitulé *Legendre Functions*, page 333, en utilisant la formule de composition des dérivées :

$$\frac{dP_l^m(\mu)}{d\theta} = \frac{dP_l^m(\mu)}{d\mu}\frac{d\mu}{d\theta},$$

avec $\mu = \cos\theta$, et en rappelant que $\theta \in [0,\pi]$.

A.3 Propriété de symétrie entre les termes du développement local d'ordres opposés.

On démontre ici la propriété présentée à l'équation (2.13) de la section 2.3.3.2, à savoir :

$$L_j^{-k} = (-1)^k \overline{L}_j^k.$$

Démonstration. On part de l'équation (2.12) en se plaçant dans le cadre d'un développement fini de degré maximum P. En notant (ρ, α, β), les coordonnées sphériques du vecteur $\mathbf{x} - \mathbf{z_2}$, on a donc :

$$\Phi(\mathbf{x}) = \sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k I_j^k(\rho, \theta, \phi).$$

Le potentiel au point $\mathbf{x}$ étant réel, on a :

$$\overline{\sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k I_j^k(\rho, \theta, \phi)} = \sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k I_j^k(\rho, \theta, \phi).$$

Soit :

$$\sum_{j=0}^{P} \sum_{k=-j}^{j} \overline{L_j^k}.\overline{I_j^k}(\rho, \theta, \phi) = \sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k I_j^k(\rho, \theta, \phi).$$

D'après l'équation (2.9) de la section 2.3.3.2, on peut écrire :

$$\sum_{j=0}^{P} \sum_{k=-j}^{j} \overline{L_j^k}.(-1)^k I_j^{-k}(\rho, \theta, \phi) = \sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k I_j^k(\rho, \theta, \phi).$$

Et, comme $(-1)^k = (-1)^{-k}$:

$$\sum_{j=0}^{P} \sum_{k=-j}^{j} \overline{L_j^{-k}}.(-1)^k I_j^k(\rho, \theta, \phi) = \sum_{j=0}^{P} \sum_{k=-j}^{j} L_j^k I_j^k(\rho, \theta, \phi)$$

$$\sum_{j=0}^{P} \sum_{k=-j}^{j} (L_j^k - (-1)^k \overline{L_j^{-k}}) I_j^k(\rho, \theta, \phi) = 0.$$

Les harmoniques sphériques formant une base (orthonormale lorsqu'elles sont normalisées) pour les fonctions $f(\theta, \phi) \in \mathbb{R}$, les L_j^k sont uniques et on en déduit donc :

$$L_j^k = (-1)^k \overline{L_j^{-k}}, \qquad \forall\, (j,k) \in \mathbb{N} \times \mathbb{Z} \quad \text{avec} \quad 0 \leq |k| \leq j \leq P$$

□

On retrouve cette condition, en suivant le même schéma de preuve, lorsque l'on dérive le potentiel par rapport à ϕ, et que l'on vérifie que la composante longitudinale de la force ainsi obtenue est réelle.

Annexe B

Théorie de la Transformée Discrète de Fourier

Cette annexe présente des définitions et des résultats classiques sur la Transformée de Fourier Discrète (TFD) : voir par exemple [100] ou [29].

Dans la plupart des cas, nous considérons implicitement que la période N est paire, de telle sorte que $\frac{N}{2}$ soit un entier. Les formules correspondantes pour N impair se déduisent directement de celles données ici pour N pair.

B.1 Convolution

Soit $(f_l)_{l\in\mathbb{Z}}$ et $(g_l)_{l\in\mathbb{Z}}$, deux suites *périodiques* de période N, nous définissons la convolution discrète, $h = f * g$, de ces deux suites par :

$$h_k = (f*g)_k = \sum_{l=-\frac{N}{2}+1}^{\frac{N}{2}} f_l g_{k-l} \qquad \forall\, k \in [\![-\frac{N}{2}+1, \frac{N}{2}]\!]. \tag{B.1}$$

Ceci peut aussi s'écrire :

$$h_k = \sum_{\substack{m+l\equiv k[N] \\ (m,l)\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]^2}} f_l g_m \tag{B.2}$$

où $a \equiv b[N]$ signifie que a et b sont « congruents modulo N ».

Une convolution à deux dimensions (convolution $2D$) de deux suites $(f_n^l)_{(n,l)\in\mathbb{Z}^2}$ et $(g_n^l)_{(n,l)\in\mathbb{Z}^2}$, toutes deux périodiques de périodes respectives J et K, est définie par :

$$h_j^k = (f\ *\ g)_j^k = \sum_{n=-\frac{J}{2}+1}^{\frac{J}{2}} \sum_{l=-\frac{K}{2}+1}^{\frac{K}{2}} f_n^l g_{j-n}^{k-l}.$$

B.2 Transformée de Fourier Discrète (TFD)

Soit $(f_l)_{l\in\mathbb{Z}}$ une suite périodique de période N, nous définissons la Transformée de Fourier Discrète de taille N de f par :

$$\widehat{f}_k = \frac{1}{N} \sum_{n=-\frac{N}{2}+1}^{\frac{N}{2}} f_n \omega_N^{-nk} \qquad \forall\, k \in \mathbb{Z}$$

où $\omega_N^j = e^{\frac{i2\pi j}{N}}$ (et $i = \sqrt{-1}$).

La suite $\widehat{f}$ est aussi périodique de période N.

B.3 Transformée de Fourier Discrète Inverse (TFDI)

Soit $(\widehat{f_l})_{l\in\mathbb{Z}}$ une suite périodique de période N, nous définissons la Transformée de Fourier Discrète Inverse (TFDI), de taille N de $\widehat{f}$ par :

$$f_n = \sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} \widehat{f}_k \omega_N^{nk} \qquad \forall\, n \in \mathbb{Z}.$$

B.4 Le théorème de la Convolution Discrète

Théorème 6 (théorème de la Convolution Discrète)**.** *Soit $(f_n)_{n\in\mathbb{Z}}$ et $(g_n)_{n\in\mathbb{Z}}$ deux suites périodiques de période N, nous notons respectivement $(\widehat{f}_k)_{k\in\mathbb{Z}}$ et $(\widehat{g}_k)_{k\in\mathbb{Z}}$ leur TFD de taille N. La TFD $\widehat{h}_k$ de leur convolution h_k est alors égal à :*

$$\widehat{h}_k = N \widehat{f}_k \widehat{g}_k.$$

Une preuve est donnée dans [29].

Les $\widehat{h}_k$ sont périodiques de période N.

B.5 Le théorème de la Convolution Discrète $2D$

Le passage aux convolutions $2D$ est direct.

Théorème 7 (théorème de la Convolution Discrète $2D$)**.** *Soit $(f_j^k)_{(j,k)\in\mathbb{Z}^2}$ et $(g_j^k)_{(j,k)\in\mathbb{Z}^2}$ deux suites définies sur $\mathcal{J}\times\mathcal{K}$ et étendues par périodicité à $\mathbb{Z}^2$. On note J et K les deux périodes. Par exemple : $\mathcal{J} = [\![-\frac{J}{2}+1, \frac{J}{2}]\!]$ et $\mathcal{K} = [\![-\frac{K}{2}+1, \frac{K}{2}]\!]$, avec J et K pairs.*

Leur TFD $2D$ de taille $J\times K$ est définie par :

$$\hat{f}_j^k = \frac{1}{JK} \sum_{n\in\mathcal{J}} \sum_{l\in\mathcal{K}} f_n^l \omega_J^{-nj} \omega_K^{-lk} \qquad \forall\, (j,k) \in \mathcal{J}\times\mathcal{K},$$

et leur TFD inverse $2D$ par :

$$f_j^k = \sum_{n\in\mathcal{J}} \sum_{l\in\mathcal{K}} \hat{f}_n^l \omega_J^{nj} \omega_K^{lk} \qquad \forall\, (j,k) \in \mathcal{J}\times\mathcal{K}.$$

Nous avons alors :

$$\hat{h}_j^k = JK \hat{f}_j^k \hat{g}_j^k \qquad \forall\, (j,k) \in \mathcal{J} \times \mathcal{K}.$$

B.6 Non-périodicité et *zero-padding*

Une convolution de taille N entre deux suites *non-périodiques* $(f_l)_{l\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]}$ et $(g_l)_{l\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]}$, peut être définie par :

$$h_k = \sum_{\substack{l+m=k \\ (l,m)\in[\![-\frac{N}{2}+1,\frac{N}{2}]\!]^2}} f_l g_m \qquad \forall\, k \in [\![-\frac{N}{2}+1, \frac{N}{2}]\!].$$

On souhaite se ramener à la formulation de l'équation (B.2). Pour cela, nous définissons deux suites $(\widetilde{f}_l)_{l\in[\![-\frac{M}{2}+1,\frac{M}{2}]\!]}$ et $(\widetilde{g}_l)_{l\in[\![-\frac{M}{2}+1,\frac{M}{2}]\!]}$ ainsi :

$$\forall\, l \in [\![-\frac{M}{2}+1, \frac{M}{2}]\!], \qquad \widetilde{f}_l = \begin{cases} f_l & \text{si } l \in [\![-\frac{N}{2}+1, \frac{N}{2}]\!], \\ 0 & \text{sinon,} \end{cases}$$

et de même pour g_l.

Nous étendons ensuite le domaine de définition de ces deux suites à $\mathbb{Z}$ par M-périodicité.

Maintenant notons :

$$\widetilde{h}_k = \sum_{\substack{m+l\equiv k[N] \\ (m,l)\in[\![-\frac{M}{2}+1,\frac{M}{2}]\!]^2}} \widetilde{f}_l \widetilde{g}_m \qquad \forall\, k \in [\![-\frac{M}{2}+1, \frac{M}{2}]\!].$$

En choisissant M « assez grand », on obtient :

$$\widetilde{h}_k = h_k \qquad \forall\, k \in [\![-\frac{N}{2}+1, \frac{N}{2}]\!].$$

Remarque B.1. *Les $\widetilde{h}_k$ avec $k \in [\![-\frac{M}{2}+1, \frac{M}{2}]\!] \setminus [\![-\frac{N}{2}+1, \frac{N}{2}]\!]$ ne sont pas nécessairement nuls.*

Quand la convolution est « centrée » (autour de 0, comme dans l'équation (B.2) par exemple), on a généralement besoin de $M \geq \frac{3N}{2}$ pour avoir : $\widetilde{h}_k = h_k, \forall\, k \in [\![-\frac{N}{2}+1, \frac{N}{2}]\!]$. Mais quand la convolution n'est pas « centrée », comme dans la formulation de l'opérateur *M2L* (voir section 4.2.1), nous avons généralement besoin de $M \geq 2N$ (voir [52] et [51]).

Cet ajout de zéros est appelé *zero-padding* en anglais.

On peut désormais utiliser le théorème de la Convolution Discrète (théorème 6) pour calculer les $\widetilde{h}_k$ (avec une TFD inverse) et ainsi obtenir les h_k pour $k \in [\![-\frac{N}{2}+1, \frac{N}{2}]\!]$.

Bibliographie

[1] Atlas homepage. See homepage for details : http ://math-atlas.sourceforge.net/.

[2] Fortran 77 reference implementation source code of the blas from netlib. Electronically available at : `http://www.netlib.org/blas`.

[3] *OpemMP Home Page*. Available from *http://www.openmp.org*.

[4] *Engineering and Scientific Subroutine Library (ESSL) for AIX, Version 3 Release 3, Guide and Reference*, December 2001. Available from *ibm.com/redbooks*.

[5] S.J. Aarseth, M. Henon, and R. Wielen. A comparison of numerical methods for the study of star cluster dynamics. *Astronomy and Astrophysics*, 37 :183–187, 1974.

[6] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions With Formulas, Graphs, and Mathematical Tables*. Tenth Printing, December 1972.

[7] M.P. Allen and D.J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, 1989.

[8] S. Aluru. Greengard's N-body algorithm is not order N. *SIAM Journal on Scientific Computing*, 17(3) :773–776, May 1996.

[9] S. Aluru and F. Sevilgen. Dynamic Compressed Hyperoctrees with Application to the N-body Problem. In *Proceedings of the 19th International Conference on Foundations of Software Technology and Theoretical Computer Science, Springer Verlag Lecture Notes in Computer Science*, volume 1738, pages 21–33, 1999.

[10] C. R. Anderson. An implementation of the fast multipole method without multipoles. *SIAM Journal on Scientific and Statistical Computing*, 13(4) :923–947, July 1992.

[11] R. J. Anderson. Tree data structures for N-body simulation. In *FOCS : IEEE Symposium on Foundations of Computer Science (FOCS)*, 1996.

[12] A. W. Appel. An efficient program for many-body simulation. *SIAM Journal on Scientific and Statistical Computing*, 6 :85–103, 1985.

[13] Argonne National Laboratory. *MPICH2 Home Page*. Available from *http://www.mcs.anl.gov/mpi/mpich2*.

[14] E. Athanassoula. Formation and dynamical evolution of galaxies and of their components. In R. Szczerba, G. Stasińska, and S. K. Górny, editors, *Planetary Nebulae as Astronomical Tools*, volume 804 of *AIP Conference Proceedings*, pages 333–340, Melville, New York, 2005.

[15] E. Athanassoula, A. Bosma, J.C. Lambert, and J. Makino. Performance and accuracy of a grape-3 system for collisionless N-body simulations. *Monthly Notices of the Royal Astronomical Society*, 293(4) :369–380(12), February 1998.

[16] J.A. Board A.Y. Toukmaji. Ewald summation techniques in perspective : a survey. *Computer Physics Communications*, 95 :73–92, 1996.

[17] I. Banicescu and R. Lu. Experiences with fractiling in N-body simulations. In *Proceedings of High Performance Computing Symposium (HPC'98)*, pages 121–126, April 1998.

[18] J. E. Barnes. Treecode guide 1.4. Electronically available at : `http://www.ifa.hawaii.edu/~barnes/treecode/treeguide.html`.

[19] J. E. Barnes. A modified tree code : Don't laugh ; it runs. *Journal of Computational Physics*, 87 :161–170, 1990.

[20] J. E. Barnes and P. Hut. A hierarchical O(N log N) force-calculation algorithm. *Nature*, 324(4) :446–449, December 1986.

[21] S. Behling, R. Bell, P. Farrell, H. Holthoff, F. O'Connell, and W. Weir. *The POWER4 Processor Introduction and Tuning Guide*, November 2001. Available from *ibm.com/redbooks*.

[22] T. C. Bishop, R. D. Skeel, and K. Schulten. Difficulties with multiple time stepping and fast multipole algorithm in molecular dynamics. *Journal of Computational Chemistry*, 18(14) :1785–1791, 1997.

[23] G. Blelloch and G. Narlikar. *A practical comparison of N-body algorithms. In Parallel Algorithms, Series in Discrete Mathematics and Theoretical Computer Science*, volume 30. American Mathematical Society, 1997.

[24] J. A. Board, J. W. Causey, J. F. Leathrum, A. Windemuth, and K. Schulten. Accelerated molecular dynamics with the fast multipole algorithm. *Chemical Physics Letters*, 198 :89–94, 1992.

[25] J.A. Board, C.W. Humphres, C.G. Lambert, W.T. Rankin, and A.Y. Toukmaji. Ewald and multipole methods for periodic N-body problems. Technical report, Duke University, Department of Electrical Engineering, 1997.

[26] J. A. Board Jr., Ziyad S. Hakura, W. D. Elliott, D. C. Gray, W. J. Blanke, and J. F. Leathrum Jr. Scalable implementations of multipole-accelerated algorithms for molecular dynamics. Technical Report 94-002, Duke University, Department of Electrical Engineering, 1994.

[27] J. A. Board Jr., Ziyad S. Hakura, W. D. Elliott, and W. T. Rankin. Scalable variants of multipole-accelerated algorithms for molecular dynamics applications. Technical Report 94-006, Duke University, Department of Electrical Engineering, 1994.

[28] A. H. Boschitsch, M. O. Fenley, and W. K. Olson. A fast adaptive multipole algorithm for calculating screened coulomb (yukawa) interactions. *Journal of Computational Physics*, 151 :212–241, 1999.

[29] W. L. Briggs and E. van Henson. *The DFT, An Owner's Manual for the Discrete Fourier Transform*. SIAM, 1995.

[30] R. Capuzzo-Dolcetta and P. Miocchi. A comparison between the fast multipole algorithm and the tree-code to evaluate gravitational forces in 3-D. *Journal of Computational Physics*, 143(1) :29, 1998.

[31] J. C. Carr, R. K. Beatson, J.B. Cherrie, T. J. Mitchell, W. R. Fright, B. C. McCallum, and T. R. Evans. Reconstruction and representation of 3D objects with radial basis functions. In *ACM SIGGRAPH 2001*, pages 67–76, Los Angeles, CA, August 2001.

[32] J. Carrier, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm for particle simulations. *SIAM Journal on Scientific and Statistical Computing*, 9(4) :669–686, July 1988.

[33] M. Challacombe, C. White, and M. Head-Gordon. Periodic boundary conditions and the fast multipole method. *Journal of Chemical Physics*, 107(23) :10131–10140, December 1997.

[34] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *Journal of Computational Physics*, 155 :468–498, 1999.

[35] H. Cheng, V. Rokhlin, and N. Yarvin. Nonlinear optimization, quadrature, and interpolation. *SIAM Journal on Optimization*, 9(4) :901–923, 1999.

[36] C. H. Choi, J. Ivanic, M. S. Gordon, and K. Ruedenberg. Rapid and stable determination of rotation matrices between spherical harmonics by direct recursion. *Journal of Chemical Physics*, 111 :8825–8831, 1999.

[37] D. Christiansen, J.W. Perram, and H.G. Petersen. On the fast multipole method for computing the energy of periodic assemblies of charged and dipolar particles. *Journal of Computational Physics*, 107(2) :403–405, 1993.

[38] T. Darden, D. York, and L. Pedersen. Particle mesh ewald : An n log(n) method for ewald sums in large systems. *Journal of Chemical Physics*, 98(12) :10089–10092, June 1993.

[39] E. Darve. *Méthodes multipôles rapides : résolution des équations de Maxwell par formulations intégrales*. PhD thesis, Université Paris 6, 1999.

[40] E. Darve and P. Havé. Efficient fast multipole method for low-frequency scattering. *Journal of Computational Physics*, 197(1) :341–363, June 2004.

[41] W. Dehnen. Towards optimal softening in 3D N-body codes : I. minimizing the force error, 2000.

[42] W. Dehnen. A very fast and momentum-conserving tree code. *Astrophysical Journal*, 536(1) :39–42, June 2000.

[43] W. Dehnen. A hierarchical O(N) force calculation algorithm. *Journal of Computational Physics*, 179 :27–42, 2002.

[44] H.-Q. Ding, N. Karasawa, and W. A. Goddard III. Atomic level simulations on a million particles : The cell multipole method for coulomb and london nonbond interactions. *Journal of Chemical Physics*, 97(6) :4309–4315, September 1992.

[45] L. Van Dommelen and E.A. Rundensteiner. Fast, adaptive summation of point sources in the two-dimensional poisson equation. *Journal of Computational Physics*, 83 :126–147, 1989.

[46] J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 16(1) :1–17, March 1990.

[47] J. Dongarra, J. Du Croz, S. Hammarling, and R. J. Hanson. An extended set of FORTRAN Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software*, 14(1) :1–17, March 1988.

[48] J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers (Software, Environments, Tools)*. SIAM, Philadelphia, PA, USA, 1998.

[49] M. Eichinger, H. Grubmüller, H. Heller, and P. Tavan. Famusamm : An algorithm for rapid evaluation of electrostatic interactions in molecular dynamics simulations. *Journal of Computational Chemistry*, 18 :1729–1749, 1997.

[50] W. D. Elliott and J. A. Board, Jr. Fast Fourier Transform accelerated fast multipole algorithm. *SIAM Journal on Scientific Computing*, 17(2) :398–415, 1996.

[51] W.D. Elliott. Multipole algorithms for molecular dynamics simulation on high performance computers. Technical Report 95-003, Duke University, Department of Electrical Engineering, 1995. Doctoral dissertation.

[52] M. A. Epton and B. Dembart. Multipole translation theory for the three-dimensional Laplace and Helmholtz equations. *SIAM Journal on Scientific Computing*, 16(4) :865–897, 1995.

[53] K. Esselink. A comparison of algorithms for long-range interactions. *Computer Physics Communications*, 87 :375–395, 1995.

[54] M. Frigo and S. G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2) :216–231, 2005. special issue on "Program Generation, Optimization, and Platform Adaptation".

[55] Y. Fu, K. J. Klimkowski, G. J. Rodin, E. Berger, J. C. Browne, J. K. Singer, R. A. Van de Geijn, , and K. S. Vemaganti. A fast solution method for three-dimensional many-particle problems of linear elasticity. *International Journal for Numerical Methods in Engineering*, 42 :1215–1229, 1998.

[56] A. Grama, V. Kumar, and A. Sameh. Scalable parallel formulations of the Barnes-Hut algorithm for n-body simulations. In *Proceedings of Supercomputing'94*, Washington DC, November 1994.

[57] A. Grama, V. Kumar, and A. Sameh. On n-body simulations using message passing parallel computers. In *Proceedings of the Seventh SIAM Coference on Parallel Processing for Scientific Computing*, San Francisco, CA, 1995.

[58] A. Graps. Amara's recap of particle simulation methods. Electronically available at : `http://www.amara.com/papers/nbody.html`.

[59] D. C. Gray. Load Balancing the Parallel Fast Multipole Algorithm. Technical Report 94-003, Duke University, Department of Electrical Engineering, 1994.

[60] L. Greengard. *The Rapid Evaluation of Potential Fields in Particle Systems*. MIT Press, Cambridge, MA, 1988.

[61] L. Greengard. Fast algorithms for classical physics. *Science*, 265 :909 – 914, august 1994.

[62] L. Greengard and W. D. Gropp. A parallel version of the fast multipole method. *Computers and Mathematics with Applications*, 20(7) :63–71, 1990.

[63] L. Greengard and J. Huang. A new version of the fast multipole method for screened coulomb interactions in three dimensions. *Journal of Computational Physics*, 180 :642–658, 2002.

[64] L. Greengard, J. Huang, V. Rokhlin, and S. Wandzura. Accelerating fast multipole methods for the Helmholtz equation at low frequencies. *IEEE Computational Science & Engineering*, 5(3) :32–38, 1998.

[65] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73 :325–348, 1987.

[66] L. Greengard and V. Rokhlin. On the efficient implementation of the fast multipole algorithm. Technical Report RR-515, Yale University Department of Computer Science, 1988.

[67] L. Greengard and V. Rokhlin. The rapid evaluation of potential fields in three dimensions, in vortex method. In C. Anderson and G. Greengard, editors, *Vortex Methods*, volume 1360 of *Lecture Notes in Mathematics*, pages 121–141. Springer-Verlag, 1988.

[68] L. Greengard and V. Rokhlin. A new version of the fast multipole method for the Laplace equation in three dimensions. *Acta Numerica*, 6 :229–269, 1997.

[69] L. Greengard and J. Strain. *SIAM Journal on Scientific and Statistical Computing*, 12 :79, 1991.

[70] N. A. Gumerov and R. Duraiswami. Recursions for the computation of multipole translation and rotation coefficients for the 3-D Helmholtz equation. *SIAM Journal on Scientific Computing*, 25(4) :1344–1381, 2003.

[71] B. Hariharan and S. Aluru. Efficient Parallel Algorithms and Software for Compressed Octrees with Application to Hierarchical Methods. In *Proceedings of the 8th IEEE International Conference on High Performance Computing, Spring Verlag Lecture Notes in Computer Science*, pages 125–136, 2001.

[72] B. Hariharan, S. Aluru, and B. Shanker. A Scalable Parallel Fast Multipole Method for Analysis of Scattering from Perfect Electrically Conducting Surfaces. In *Proceedings of the 2002 IEEE/ ACM Supercomputing Conference*, 2002.

[73] R. W. Hockney and J. W. Eastwood. *Computer Simulation Using Particles*. Institute of Physics Publishing, 1988.

[74] Y. Hu and S. L. Johnsson. Implementing O(N) N-body algorithms efficiently in data-parallel languages. *Scientific Programming*, 5(4) :337–364, 1996.

[75] J. F. Leathrum Jr. and J. A. Board Jr. The Parallel Fast Multipole Algorithm in Three Dimensions. Technical Report 92-001, Duke University, Department of Electrical Engineering, 1992.

[76] B. Kågström, P. Ling, and C. Van Loan. GEMM-based level 3 BLAS : high-performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Softw.*, 24(3) :268–302, 1998.

[77] A. Kawai, T. Fukushige, J. Makino, and M. Taiji. Grape-5 : A special-purpose computer for n-body simulations. *Publ. of the Astronomical Society of Japan*, 52 :659–676, August 2000.

[78] S. Krishnan and L.V. Kale. A parallel adaptive fast multipole algorithm for N-body problems. In *International Conference on Parallel Processing*, pages 46–50, 1995.

[79] J. Kurzaka and B. M. Pettitt. Communications overlapping in fast multipole particle dynamics methods. *Journal of Computational Physics*, 203(2) :731–743, March 2005.

[80] C.G. Lambert, T.A. Darden, and J.A. Board. A multipole-based algorithm for efficient calculation of forces and potentials in macroscopic periodic assemblies of particles. *Journal of Computational Physics*, 126 :274–285, 1996.

[81] G. Latu. *Algorithmique parallèle et calcul haute performance dédiés à la simulation d'un système hôte-macroparasite*. PhD thesis, Université Bordeaux 1, December 2002.

[82] J.K. Lawder and P.J.H. King. Using state diagrams for hilbert curve mappings. *International Journal of Computer Mathematics*, 78(3) :327–342, 2001.

[83] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for FORTRAN usage. *ACM Transactions on Mathematical Software*, 5(3) :308–323, September 1979.

[84] P. Liu and S. Bhatt. Experiences with parallel n-body simulation. *IEEE Trans. Parallel Distrib. Syst.*, 11(12) :1306–1323, 2000.

[85] P. Londrillo, C. Nipoti, and L. Ciotti. A parallel implementation of a new fast algorithm for N-body simulations. In R. Capuzzo Dolcetta, editor, *Computational astrophysics in Italy : methods and tools*, volume 1, pages 18–26, Bologna, Italy, July 2002. Memorie della Società Astronomica Italiana Supplementi.

[86] E. Jui-Lin Lu and D. I. Okunbor. A massively parallel fast multipole algorithm in three dimensions. In *HPDC '96 : Proceedings of the High Performance Distributed Computing (HPDC '96)*, pages 40–48, Washington, DC, USA, August 1996. IEEE Computer Society.

[87] E. Jui-Lin Lu and D. I. Okunbor. An efficient load balancing technique for parallel FMA in message passing environment. In *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[88] Y. M. Marzouk and A. F. Ghoniem. K-means clustering for optimal partitioning and dynamic load balancing of parallel hierarchical N-body simulations. *Journal of Computational Physics*, 207(2) :493–528, August 2005.

[89] P. Miocchi. *Globular clusters dynamics and its influence on the galactic nuclei activity*. PhD thesis, University of Rome "La Sapienza", 1995-1998.

[90] P.M. Morse and H. Feshbach. *Methods of Theoretical Physics*. McGraw-Hill, New York, 1953.

[91] MPI Forum. *Message Passing Interface MPI Forum Home Page*. Available from *http:// www.mpi-forum.org/* .

[92] K. Nabors, S. Kim, and J. White. Fact capacitance extraction of general three-dimensional structures. *IEEE Transactions on Microwave Theory and Techniques*, 40(7) :1496–1506, July 1992.

[93] K. Nabors, F. T. Korsmeyer, F. T. Leighton, and J. White. Preconditioned, adaptive, multipole-accelerated iterative methods for three-dimensional first-kind integral equations of potential theory. *SIAM Journal on Scientific Computing*, 15(3) :713–735, May 1994.

[94] K. Nabors and J. White. Fastcap : A multipole accelerated 3-D capacitance extraction program. *IEEE Transactions on Computer-Aided Design*, 10(11) :1447–1459, November 1991.

[95] L. S. Nyland, J. F. Prins, and J. H. Reif. A data-parallel implementation of the adaptive fast multipole algorithm. In *Proceedings of the 1993 DAGS/PC Symposium*, pages 111–123, Dartmouth College, Hanover, NH, June 1993.

[96] S. Ogata, T.J. Campbell, R.K. Kalia, A. Nakano, P. Vashishta, and S. Vemparala. Scalable and portable implementation of the fast multipole method on parallel computers. *Computer Physics Communications*, 153(3) :445–461, July 2003.

[97] J. Perram, H. Petersen, and S. de Leeuw. An algorithm for the simulation of condensed matter which grows as the 3/2 power of the number of particles. *Molecular Physics*, 65(4) :875–893, 1988.

[98] H.G. Petersen, E.R. Smith, and D. Soelvason. Error estimates for the fast multipole method. II. The three-dimensional case. *Proceedings of the Royal Society of London. Series A, Mathematical and physical sciences*, 448 :401–418, 1995.

[99] H.G. Petersen, D. Soelvason, J.W. Perram, and E.R. Smith. The very fast multipole method. *Journal of Chemical Physics*, 101(10) :8870, 1994.

[100] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes in C : The Art of Scientific Computing, Second Edition*. Cambridge University Press, 1992.

[101] W. T. Rankin, J. A. Board Jr., and V. L. Henderson. The impact of data ordering strategies on a distributed hierarchical multipole algorithm. In *Ninth SIAM Conference on Parallel Processing for Scientific Computing*. SIAM Press, 1999.

[102] W.T. Rankin. Efficient parallel implementations of multipole based N-body algorithms. PhD. Dissertation, Duke University, Department of Electrical Engineering, April 1999.

[103] J. K. Salmon and M. S. Warren. Skeletons from the treecode closet. *Journal of Computational Physics*, 111(1) :136–155, 1994.

[104] J.K. Salmon and M.S. Warren. Parallel, out-of-core methods for N-body simulation. In *Eighth SIAM Conf. on Parallel Processing for Scientific Computing*, Philadelphia, 1997. SIAM.

[105] H. Samet. *Design and Analysis of Spatial Data Structures*. Addison-Wesley, Reading, MA, 1990.

[106] K.E. Schmidt and M.A. Lee. Implementing the fast multipole method in three dimensions. *Journal of Statistical Physics*, 63(5/6), 1991.

[107] F. E. Sevilgen, S. Aluru, and N. Futamura. A Provably Optimal, Distribution-Independent Parallel Fast Multipole Method. In *IPDPS*, pages 77–84, 2000.

[108] J. Shimada, H. Kaneko, and T. Takada. Performance of fast multipole methods for calculating electrostatic interactions in biomacromolecular simulations. *Journal of Computational Chemistry*, 15(1) :28–43, 1994.

[109] J. P. Singh, J. L. Hennessy, and A. Gupta. Implications of hierarchical N-body methods for multiprocessor architectures. *ACM Transactions on Computer Systems*, 13(2) :141–202, May 1995.

[110] J. P. Singh, C. Holt, J. L. Hennessy, and A. Gupta. A parallel adaptive fast multipole method. In *Conference on Supercomputing*, Computer Systems Laboratory, Stanford University, Stanford, CA 94305, 1993.

[111] J. P. Singh, C. Holt, T. Totsuka, A. Gupta, and J. Hennessy. Load balancing and data locality in adaptive hierarchical N-body methods : Barnes-Hut, fast multipole, and radiosity. *Journal of Parallel and Distributed Computing*, 27 :118–141, 1995.

[112] D. Soelvason, J. Kolafa, H.G. Petersen, and J.W. Perram. A rigorous comparison of the ewald method and the fast multipole methode in two dimensions. *Computer Physics Communications*, 87 :307–318, 1995.

[113] D. Soelvason and H.G. Petersen. Error estimates for the fast multipole method. *Journal of Statistical Physics*, 86 :391–420, 1997.

[114] V. Springel. "the cosmological simulation code gadget-2". *Monthly Notices of the Royal Astronomical Society*, 2005. submitted.

[115] J. H. Strickland and R. S. Baty. Modification of the Carrier, Greengard, and Rokhlin FMM for independent source and target fields. *Journal of Computational Physics*, 142(1) :123–128, May 1998.

[116] X. Sun and N. P. Pitsianis. A matrix version of the fast multipole method. *SIAM Review*, 43(2) :289–300, 2001.

[117] G. Sylvand. *La Méthode Multipôle Rapide en Electromagnétisme : Performances, Parallélisation, Applications*. PhD thesis, CERMICS/INRIA, 2002.

[118] S.-H. Teng. Provably good partitioning and load balancing algorithms for parallel adaptive N-body simulation. *SIAM Journal on Scientific Computing*, 19(2) :635–656, March 1998.

[119] Z. Wang, J. A. Lupo, A. M. McKenney, and R. Pachter. Large scale molecular dynamics simulations with fast multipole implementations. In *Supercomputing '99*, Portland, Oregon, November 1999.

[120] M. S. Warren and J. K. Salmon. Astrophysical N-body simulations using hierarchical tree data structures. In *Supercomputing '92*, pages 570–576, Los Alamitos, 1992. IEEE Comp. Soc.

[121] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree N-body algorithm. In *Supercomputing '93*, pages 12–21, Los Alamitos, 1993. IEEE Comp. Soc.

[122] M. S. Warren and J. K. Salmon. A portable parallel particle program. *Computer Physics Communications*, 87, 1995.

[123] R. C. Whaley and A. Petitet. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software : Practice and Experience*, 35(2) :101–121, February 2005. `http://www.cs.utsa.edu/~whaley/papers/spercw04.ps`.

[124] C. A. White and M. Head-Gordon. Derivation and efficient implementation of the fast multipole method. *Journal of Chemical Physics*, 101(8) :6593–6605, October 1994.

[125] C. A. White and M. Head-Gordon. Rotating around the quartic angular momentum barrier in fast multipole method calculations. *Journal of Chemical Physics*, 105(12) :5061–5067, 1996.

[126] N. Yarvin and V. Rokhlin. Generalized gaussian quadratures and singular value decompositions of integral operators. *SIAM Journal on Scientific Computing*, 20(2) :699–718, 1998.

[127] L. Ying, G. Biros, and D. Zorin. A kernel-independent adaptive fast multipole algorithm in two and three dimensions. *Journal of Computational Physics*, 196(2) :591–626, 2004.

[128] L. Ying, G. Biros, D. Zorin, and H. Langston. A new parallel kernel-independent fast multipole method. In *Proceedings of the 16th ACM/IEEE Conference on Supercomputing*, November 2003.

[129] F. Zhao. An O(N) algorithm for three-dimensional N-body simulations. Technical Report 995, MIT Artificial Intelligence Laboratory, 1987.

[130] F. Zhao and S. L. Johnsson. The parallel multipole method on the connection machine. *SIAM Journal on Scientific and Statistical Computing*, 12(6) :1420–1437, November 1991.

Liste des publications

[1] P. Fortin, *Multipole-to-local operator in the Fast Multipole Method : comparison of FFT, rotations and BLAS improvements*. INRIA Research Report 5752 (2005), http ://www.inria.fr/rrrt/rr-5752.html.

[2] O. Coulaud, P. Fortin and J. Roman, *High performance BLAS formulation of the multipole-to-local operator in the Fast Multipole Method*. Soumis à Journal of Computational Physics.

[3] O. Coulaud, P. Fortin and J. Roman, *High-performance BLAS formulation of the Adaptive Fast Multipole Method*. Advances in Computational Methods in Sciences and Engineering 2005, Selected Papers from the International Conference of Computational Methods in Sciences and Engineering (ICCMSE 2005), Special Volume of the Lecture Series on Computer and Computational Sciences, VSP/Brill, vol. 4B, pp. 1796-1799.

[4] O. Coulaud, P. Fortin and J. Roman, *High-performance BLAS formulation of the Adaptive Fast Multipole Method*. Accepté sous révision à Journal of Supercomputing (Kluwer Academic).