# Méthodes Multipôles Rapides pour l'électromagnétisme : Parallélisme et Basses Fréquences

Pascal Havé

**Université Pierre et Marie Curie, Paris VI**
**École doctorale des Sciences Mathématiques de Paris Centre**
**UFR 921**

# Méthodes Multipôles Rapides pour l'électromagnétisme : Parallélisme et Basses Fréquences

## THÈSE

présentée et soutenue publiquement le Jeudi 13 Mai 2004

pour l'obtention du

**Doctorat de l'université Pierre et Marie Curie – Paris 6**

**(spécialité Mathématiques Appliquées)**

par

Pascal Havé

**Composition du jury**

| | | |
|---|---|---|
| *Directeur de thèse :* | O. PIRONNEAU | Université Pierre et Marie Curie |
| *Rapporteurs :* | L. GREENGARD | Courant Institute, NYU |
| | P. JOLY | INRIA |
| *Examinateurs :* | B. DESPRÉS | Université Pierre et Marie Curie |
| | P. Frey | Université Pierre et Marie Curie |
| | P. SOUDAIS | Dassault-Aviation |

**Laboratoire Jacques Louis Lions — UMR 7598**

Mis en page avec la classe thloria.

# Remerciements

Tout comme le Tout ne naît pas du vide, tout comme quelques mots peuvent transformer un *encore* rien en un *déjà* quelque chose, il est pour moi temps de remercier ceux (et celles) qui ont jalonné ces quelques années de thèse, à défaut de citer leurs prédécesseurs ayant non moins influencé ma trajectoire jusqu'ici.

Tant le temps passe, tant les choses changent. Ainsi, s'il fallait remonter aux sources de cette thèse, je ne peux que remercier Frédéric Hecht de ne pas m'avoir laissé m'évader des mathématiques et plus particulièrement Olivier Pironneau de m'avoir guider dans une voie que je ne peux regretter. Au délà d'avoir été mon directeur de thèse Olivier Pironneau m'a permis de m'exprimer pleinement tout en me guidant dans de nouveaux espaces de connaissance.

Vient ensuite Stéphane D.P. avec qui, en plus de partager notre directeur de thèse, nous avons partager 2 ans d'une vie dépassant celle de co-bureaux, avec de fructueux échanges dont je lui dois en particulier ma `Debian`isation ainsi que la visite de quelques établissements de fin de journée.

Je me souviendrais encore longtemps de la *présence* des autres fameux *gruik's* Philippe H. et Frédéric L. (notre J adoré), qui pour l'un fut l'essence même du bon *gruik* et pour l'autre un excellent exemplaire de la bonne Nature Humaine.

De l'instabilité peut naître des choses parfois sombres, étranges et parfois orange. C'est ainsi que la relève de la lignée des étudiants de notre 'chef' est arrivée en Cécile D. (MnP). A défaut de rejoindre le *vrai* Monde (≠Mac :), elle a su renouveler l'*air* de notre bureau avec une bonne humeur et un esprit Orange flamboyant.

Comme de la transmission d'un savoir naît l'évolution puis notre envol, je remercie Eric Darve pour ses conseils et une large collaboration dont aujourd'hui vous pourrez en appréhender quelques points.

Mais puisque le monde ne se réduit pas à notre proximité immédiate, mon quotidien de *thésard* remercie en vrac Mlle A., Mourad j'ai gagné !, Martin notre dévoué responsable, Paul les bonnes questions, {Laurent (M. M), Alain, Stéphane & Simona} tir groupé, et tous les autres thésards du passé et du présent que j'ai pu croisé même au-delà des frontières de notre labo, et encore au delà j'ai une pensée à mes amis de *Magic*, de Normandie et d'ailleurs (parfois si proche, et parfois si loin).

Au délà d'une vie de thésard, j'ai eu l'occasion grâce à la confiance de Yvon Maday et Olivier Pironneau, de perfectionner ma passion informatique avec entre autres, le projet *Hydre*, qui au délà d'être une machine, fut pour moi la source d'une consommation peut-être excessive de temps, mais surtout de satisfaction et de progrès. Même si je n'ai jamais rencontré la/ma Question, en pluie et en boulettes sont elles tombées durant ces quelques années, et si j'ai toujours pris chacune comme unique, je vous remercie ; de la question surgit la réponse et de la réponse une nouvelle question, la source de l'Avant.

Je remercie Dassault-Aviation et en particulier Jacques Périaux, Paul Soudais et Quentin Carayol pour notre collaboration ouverte et instructive.

Je remercie Mmes Ruprecht, Boulic et Loriller pour les aides et conseils sur des plans administratifs parfois tortueux. De même, je ne peux oublier M. David qui a permis d'établir ce document alors même que le temps fuyait devant moi.

Auprès de toute l'équipe technique de VisioP7, je présente toute ma gratitude pour l'effort technique et leur accueil de cette visio-présentation.

Pour finir, je remercie largement Leslie Greengard, Patrick Joly et Francis Collino pour l'attention qu'ils ont accordé à la lecture de mon manuscrit et des remarques qui ont suivi, permettant d'améliorer autant que possible la clarté de ce document, sans oublier Paul Soudais et Bruno Desprès pour avoir accepter de faire partie de ce jury de thèse.

Je ne remercie pas le temps et l'espace qui m'ont trop souvent fait défaut, ainsi que la dure *météo* régnant dans le bureau 2D01

*à la mémoire du jour de l'été 1995*

# Table des matières

# Notations

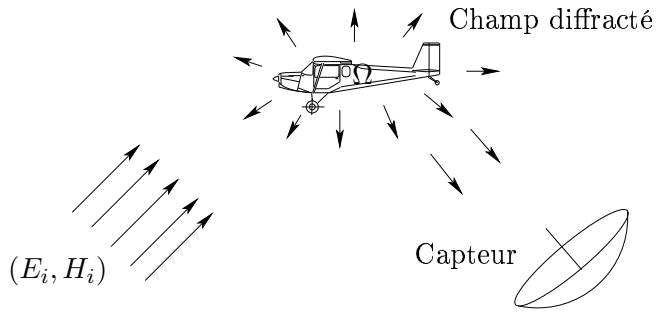
| | |
|---|---|
| $\stackrel{\text{def}}{=}$ | Égalité par définition |
| $\imath$ | Nombre imaginaire pur unité : $\imath \stackrel{\text{def}}{=} \sqrt{-1}$ |
| $\hat{x}$ | Normalisation de $x \neq 0$ : $\hat{x} \stackrel{\text{def}}{=} \frac{x}{\lvert x \rvert}$ |
| $\langle x, y \rangle = x \cdot y$ | Produit scalaire de $x$ et $y$ |
| $S^2 = \{x \in \mathbb{R}^3, \lVert x \rVert_2 = 1\}$ | Sphère unité de $\mathbb{R}^3$ |
| $\text{Span}(\{u_i\}_{i \in I})$ | Espace vectoriel engendré par les éléments $u \in \{u_i\}_{i \in I}$ |
| $\delta_{ij}$ | Symbole de Kronecker ; $\delta_{ii} \stackrel{\text{def}}{=} 1$, $\delta_{ij} = 0 \Leftrightarrow i \neq j$ |
| $A^*$ | Matrice adjointe de $A$ |
| $z^*$, $\bar{z}$ | nombre complexe conjugué de $z$ |
| $\mathbf{1}_A(x)$ | $\mathbf{1}_A(x) = 1$ si $x \in A$, 0 sinon |
| $\log x$ | logarithme népérien (base $e$) de $x$ |
| $\log_{10} x$ | logarithme vulgaire (base 10) de $x$ |

# Introduction

Le problème que l'on se propose d'étudier est l'interaction d'une onde électromagnétique et d'un objet conducteur. Le point de vue que nous adopterons est celui de la simulation numérique. Ces phénomènes intervient dans de nombreux domaines : aéronautique, fours à micro-ondes, voitures, antennes...

Les développements dans le domaine de l'électronique nécessitent de porter une attention particulière aux influences de l'environnement électromagnétique sur les équipements sensibles (électronique embarquée, antennes) ; ceci intervient en terme de compatibilité électromagnétique : le but est d'adapter les équipements électroniques aux effets électromagnétiques tant extérieurs qu'induits par les autres équipements (comme les antennes, de plus en plus nombreuses).

Par ailleurs, ce type d'études apporte de précieuses informations durant la conception d'aéronefs militaires. En effet, l'un des points clefs pour ces d'appareils est leur furtivité. De telles études permettent d'en estimer la discrétion vis à vis des radars par un contrôle de leur signature radar.




*Problème typique de simulation électromagnétique.*

Notons que ce dernier type de problèmes se compose principalement, par nature, de problèmes *extérieurs*, c'est-à-dire que le domaine d'étude est l'extérieur d'un objet, donc souvent non borné.

Ce point induit une première difficulté dès qu'une approche de type différences finies ou éléments finis est envisagée. En effet, ces méthodes ne sont applicables qu'à des domaines bornés (voir cependant les méthodes de type éléments infinis ou inversés [13, 10, 7]). Ainsi, il convient de tout d'abord tronquer le domaine de calcul puis d'y ajouter des conditions limites « transparentes » au bord du domaine (par exemple de type P*erfectly* M*atched* L*ayer*[5]) afin de masquer les effets de la troncature, sans toutefois y parvenir exactement à cause de problèmes de réflexions numériques.

Par ailleurs, la nature oscillante des problèmes électromagnétiques impose une contrainte de discrétisation du domaine en requérant un nombre minimum de points par longueur d'onde (de l'ordre de 10), et produisant ainsi des systèmes de taille $O(1/\lambda^3)$, pour une longueur d'onde $\lambda$ donnée. La génération du maillage volumique peut être une autre difficulté non négligeable en ce qui concerne la méthode des éléments finis, d'autant que les générateurs de maillages automatiques et performants (tant en qualité de maillage qu'en vitesse de génération) sont encore assez peu répandus [31].

Soulignons toutefois que ces approches offrent la possibilité de résoudre des problèmes transitoires et de gérer des matériaux hétérogènes.

Les méthodes intégrales représentent une approche radicalement différente en réduisant le problème extérieur en un problème à la surface de l'objet. À la fois, cette réduction est réversible, en permettant de reconstruire les champs magnétique et électrique extérieurs à l'objet, et vérifie implicitement les conditions de radiation à l'infini. Les inconnues du problème deviennent alors les courants électrique et magnétique à la surface de l'objet décomposés suivant une base de fonctions tangentes à la surface de l'objet. La première difficulté associée à cette approche est que chaque fonction de base, bien que représentant un courant local, rayonne dans tout l'espace et interagit avec toutes les autres fonctions de base ; ce qui génère des systèmes pleins de taille en $O(1/\lambda^2)$.

La résolution d'un système plein d'ordre $N$ est un problème informatique considérable, tant pour le coût du stockage d'informations $O(N^2)$ que pour la résolution (en ($O(N^3)$ pour les méthodes directes, en $O(N_{\text{iter}}N^2)$ pour les méthodes itératives). Il n'est actuellement guère possible de dépasser des tailles de systèmes de 50 000 inconnues sur une station de travail et quelques centaines de millier d'inconnues sur une machine massivement parallèle.

La méthode multipôle rapide fût une grande invention qui a permis de dépasser des contraintes jusqu'alors infranchissables. Elle s'inscrit dans la même veine que les transformées de Fourier rapides (F*ast* F*ourier* T*ransforms*) et les méthodes de tri rapide Quicksort en leurs temps.

Initialement justifiée par L. Greengard et V. Rokhlin pour le problème à N corps [34, 12], puis adaptée à l'électromagnétisme par V. Rokhlin [28, 63] et W.C. Chew [70, 72], la méthode multipôle rapide ou FMM (F*ast* M*ultipole* M*ethod*) suit de continuelles améliorations à la fois sur le plan théorique (étude de l'erreur [21, 11]) et sur l'implémentation ([75, 74]).

C'est dans cette continuité que s'inscrit cette thèse en proposant une méthode alternative à la maintenant très classique FMM de W.C. Chew particulièrement adaptée aux hautes fréquences mais présentant des instabilités lorsque que la précision demandée devient trop importante ou que le nombre de niveaux est grand.

Nous commencerons cette thèse par le chapitre 1 **Formulations intégrales des équations de Maxwell** où nous aborderons les formulations intégrales permettant de poser le problème de Maxwell en terme de formulation variationnelle posée à la surface de l'objet étudié, et ensuite d'en construire le système linéaire résolu par une méthode itérative. Nous y décrirons la méthode itérative GMRES. Nous restreindrons toutefois ce cadre en étudiant la diffraction d'une onde électromagnétique monochromatique plane par un objet parfaitement conducteur.

Ensuite, dans le chapitre 2 **Principes généraux de la Méthode Multipôle Rapide**, nous aborderons formellement le principe de la méthode multipôle rapide, tout en fournissant les propriétés des opérateurs que nous aurons à traiter dans les chapitres suivants.

Le chapitre 3 **Formulation Hautes Fréquences pour l'électromagnétisme** donnera une description de la HF-FMM qui sera par la suite la méthode de référence dans le domaine

des hautes fréquences.

En constatant les instabilités précédemment rappelées de la méthode HF-FMM, L. Greengard et V. Rokhlin ont développé la méthode LF-FMM [33]. C'est en partant de ces mêmes constations et en réutilisant la même décomposition du noyau de Green $e^{\imath\kappa r}/r$ que nous avons défini une nouvelle méthode alternative à la HF-FMM, la S*table* P*lane* W*ave* FMM.

Le chapitre 4 **Formulation Toutes Fréquences - Stable Plane Wave FMM** sera l'occasion de développer cette idée originale, fruit d'une collaboration avec É. Darve et ayant fait l'objet de publications. Le domaine de compétences de la SPW-FMM s'oriente idéalement aux régimes de basses et moyennes fréquences du fait de l'extrême performance de la très répandue méthode HF-FMM en régimes de hautes fréquences. La méthode s'appuie sur la décomposition du noyau de Green en deux termes, propagatif et évanescent, pour lesquels nous proposerons à chaque fois deux méthodes de calcul :

- décomposition en Harmoniques Sphériques et transformées de Fourier rapide pour le terme propagatif,
- décomposition en valeurs singulières avec une optimisation par transformées de Fourier sur une forme modifiée de l'intégrale pour le terme évanescent.

Toutes les discrétisations seront accompagnées d'une estimation de leur complexité en fonction de la précision et de la taille du niveau considéré dans la méthode FMM.

À ce point commence la deuxième partie portant sur les implémentations des méthodes et leurs résultats.

Le chapitre 5 **Approche méthodologique** portera sur des considérations et détails d'implémentation à l'occasion de leur mise en œuvre dans le langage C++[73] de la méthode multipôle rapide adaptée aux formulations précédemment citées. En particulier, ce chapitre portera sur la manière de discrétiser la surface, et plus précisément sur les spécificités techniques propres à la méthode SPW-FMM, dont le traitement directionnel des opérateurs de type agrégation ou transfert. Ce chapitre sera aussi l'occasion d'exposer le principe de FMM adaptative, fortement lié au maillage adaptatif. Ce principe permet, en effet, de ne développer que localement l'arbre FMM pour y introduire des raffinements FMM, tout d'abord, puis de manière plus profonde des raffinements locaux de maillages.

Le chapitre 6 **Parallélisme** définira le type et le niveau de parallélisme considéré, tant du côté logiciel que pour la structuration des données. La structure d'arbre FMM pour le parallélisme sera particulièrement développée puisque source de nombreuses optimisations. Tout d'abord, nous exposerons le développement de méthodes de partitionnement à bases de graphes en vue de répartir les données de manière cohérente avec les opérateurs de la FMM. Le traitement des interactions proches et lointaines ne sera pas délaissés pour autant et nous présentetons différentes techniques de réduction des coûts du parallélisme grâce à des méthodes adaptatives visant à optimiser soit les communications soit la répartition des données relativement à la charge des différents processeurs (concept connu sous le terme de «load balancing»).

Nous finirons par le chapitre 7 **Résultats numériques** portant sur quelques résultats obtenus à la fois au niveau des briques de base de la méthode multipôle (description des discrétisations, du coûts des opérateurs... ) que de simulations numériques de validation. Nous y montrerons les très classiques sphères dont la SER sera comparée aux séries de Mie, ainsi des exemples plus réalistes de type véhicules roulant ou volant.

Les annexes se décomposent en deux groupes : deux illustrations de bibliothèques internes au code développées à l'occasion de cette thèse, et de résultats très calculatoires autour du calcul des fonctions de transfert pour le terme propagatif de la formulation «Stable Plane

Wave FMM».

L'annexe A approfondira l'optimisation des communications dans un environnement éventuellement hétérogène pour un cadre de programmation SPMD (les programmes classiques, ni vectoriels, ni multithreadés). Des résultats sur des gains non négligeables et une adaptation automatique au contexte d'exécution y seront exposés.

L'annexe B reviendra sur un concept très simple qui est la notion d'«itérateur». Le but des itérateurs est de se «déplacer» sur des structures de données. Derrière cette notion simple nous verrons ce qu'il est possible de faire comparativement aux itérateurs STL du C++.

Les annexes suivantes proposeront des constructions des approximations des fonctions de transfert du terme propagatif par projections soit dans une base d'harmoniques sphériques, soit dans une base de Fourier.

Cette thèse aura alors abordé un nouvel algorithme de méthode multipôle rapide adaptée aux équations de Maxwell, «Stable Plane Wave FMM», basée sur une nouvelle expansion du noyau de Green $\exp(\imath\kappa r)/r$ en termes d'ondes planes. Cette nouvelle formulation supprime quelques restrictions par rapport à l'approche traditionnelle. En particulier, elle permet l'utilisation de cellules de taille arbitrairement petite, ce qui est un avantage en basse et moyenne fréquences où l'objet étudié ne mesure plus qu'une fraction de la longueur d'onde. Dans ce cadre, cette méthode se montre efficace comparativement à la traditionnelle formulation, principalement orientée vers les applications en hautes fréquences.

Une conséquence de cette formulation est la possibilité de traiter des maillages adaptés, par un raffinement local de la structure arborescente de la FMM, s'accompagnant d'une potentielle réduction du coût de calcul. Ceci permet de choisir une taille de cellules arbitrairement petite, contrairement à la méthode classique souffrant, à ce niveau, d'instabilités numériques.

De futurs développements porteront sur une amélioration de la méthode SPW-FMM par un traitement simplifié du terme évanescent s'appuyant essentiellement sur la forme particulière de son spectre de Fourier (section 4.3.1) et un usage intensif des transformées de Fourier rapides (FFT).

# Première partie

# Les Méthodes Multipôles en électromagnétisme

# Chapitre 1

# Formulations intégrales des équations de Maxwell

**Sommaire**



Réduire le problème de Maxwell tri-dimensionnel à une formulation intégrale à la surface de l'objet étudié permet d'éviter de nombreux problèmes comme la discrétisation de l'espace et l'établissement de conditions limite transparentes, qui usuellement implique un élargissement du domaine de calcul, et donc du problème traité. À titre d'illustration, une simple sphère d'une surface composée de 200 000 triangles, représente un maillage volumique d'environ 10 000 000 tétraèdres (réalisé par GHS3D). Notons que la détermination du courant électrique à la surface de l'objet permet la reconstruction du champ diffracté dans tout l'espace extérieur à l'objet. Le phénomène étudié est régi par les équations de Maxwell.

## 1.1 Équations de Maxwell Harmonique

Soit un objet $\Omega$ parfaitement conducteur de frontière $\Gamma$ plongé dans le vide. Des extensions à des objets non parfaitement conducteurs (non-PEC) et aux antennes ont déjà été envisagées comme réalisables [11, 74] sans toutefois en proposer d'implémentation.

Soit $\vec{E}^{inc}$ une onde électromagnétique incidente de fréquence $f$ et de pulsation $\omega = 2\pi f$ illuminant $\Omega$.

Notons $\Omega^e$ le domaine extérieur à $\Omega$ et $\vec{n}$ la normale unitaire sortante à $\Gamma$.

Notons les constantes diélectriques du vide $\epsilon_0$ et $\mu_0$, respectivement la permittivité et perméabilité du vide. Ainsi définissons $Z_0 = \sqrt{\frac{\mu_0}{\epsilon_0}}$ l'impédance du vide et $c = 1/\sqrt{\mu_0\epsilon_0}$ la vitesse de propagation de l'onde dans le vide.

Le problème de Maxwell extérieur dans le domaine des fréquences peut s'écrire :

$$\begin{cases} \mathrm{rot}\vec{E} - \imath\omega\mu_0\vec{H} = 0, & \text{dans } \Omega_e \\ \mathrm{rot}\vec{H} + \imath\omega\epsilon_0\vec{E} = 0, & \text{dans } \Omega_e \\ \vec{E}\wedge\vec{n}|_\Gamma = -\vec{E}^{inc}\wedge\vec{n}, & \text{sur } \Gamma \end{cases} \tag{1.1}$$

auquel il convient d'ajouter la condition de radiation à l'infini de Silver-Müller.

$$\lim_{|r|\to+\infty} |r||\sqrt{\epsilon_0}\vec{E} - \sqrt{\mu_0}\vec{H}\wedge r| = 0 \tag{1.2}$$

**Remarque** *Le domaine fréquentiel induit une dépendance implicite en temps de type $e^{-\imath\omega t}$ (l'autre convention possible $e^{\imath\omega t}$ a des répercutions au niveau du noyau de Green ainsi que des formulations en découlant).*

### 1.1.1 Formulations intégrales

L'idée principale de cette section est la construction d'une formulation des équations de Maxwell en tant que problème posé à la surface $\Gamma$ de l'objet. Nous renvoyons le lecteur à [58] pour de plus amples détails concernant les résultats évoqués dans cette section.

Notons $\vec{j} = \vec{n}\wedge H^{tot}$ et $\vec{m} = \vec{n}\wedge E^{tot}$ les traces tangentielles des champs électriques et magnétiques sur $\Gamma$ avec $E^{tot} = E + E^{inc}$ et $H^{tot} = H + H^{inc}$ champs totaux.

Dans notre cas d'un matériau parfaitement conducteur, le champ $\vec{m}$ est nul à la surface de l'objet. Le courant électrique surfacique $\vec{j}$ représente le saut de la composante tangentielle de $H$ au franchissement de la surface $\Gamma$.

L'ensemble du problème peut être ramené à la détermination du courant $\vec{j}$.

**Théorème 1** *Représentation des champs $E$ et $H$ (formules de Stratton-Chu[18]) Connaissant le courant électrique $\vec{j}$ sur la surface $\Gamma$, les champs diffractés $E$ et $H$ peuvent se calculer comme suit :*

$$\forall y\in\Omega_e \begin{cases} E(y) = \imath\omega\mu_0\displaystyle\int_\Gamma G(x-y)j(x)\,\mathrm{d}s(x) + \frac{\imath}{\omega\epsilon_0}\nabla_y\int_\Gamma G(x-y)div_\Gamma j(x)\,\mathrm{d}s(x) \\ H(y) = -\displaystyle\int_\Gamma \nabla_x G(x-y)\wedge j(x)\,\mathrm{d}s(x) = -rot_y\int_\Gamma G(x-y)\,j(x) \end{cases} \tag{1.3}$$

*avec*

$$G(r) = \frac{e^{\imath\kappa|r|}}{4\pi|r|}$$

*fonction de Green, solution élémentaire de l'équation de Helmholtz 3D associée à une condition de radiation sortante*

$$\Delta u + \kappa^2 u = -\delta_0 \quad et \quad \lim_{|r|\to+\infty} r\Big(\frac{\partial u}{\partial r} - \imath\kappa u\Big) = 0$$

*et toujours $\kappa = \omega\sqrt{\epsilon_0\mu_0}$.*

**Formulation EFIE**

À partir de la condition de conducteur parfait

$$E(y) \wedge \vec{n}(y) = 0 \qquad \forall y \in \Gamma$$

nous obtenons l'équation dite EFIE (Electric Field Integral Equation) qui peut se réécrire

$$\forall y \in \Gamma \begin{cases} (n \wedge E^{inc})(y) = \imath\omega\mu_0 \left( \displaystyle\int_\Gamma G(x-y) j(x) \, \mathrm{d}s(x) \right) \wedge n(y) \\ \qquad\qquad + \dfrac{\imath}{\omega\epsilon_0} \left( \displaystyle\int_\Gamma (\nabla_y G(x-y) \mathrm{div}_\Gamma j(x)) \, \mathrm{d}s(x) \right) \wedge n(y) \end{cases} \tag{1.4}$$

qui revient à une égalité des composantes tangentielles

$$\forall t \perp n(y) \quad \begin{cases} -E^{inc}(y) \cdot t = \imath\omega\mu_0 \displaystyle\int_\Gamma t \cdot G(x-y) j(x) \, \mathrm{d}s(x) \\ \qquad\qquad + \dfrac{\imath}{\omega\epsilon_0} \displaystyle\int_\Gamma t \cdot \nabla_y G(x-y) \mathrm{div}_\Gamma j(x) \, \mathrm{d}s(x) \end{cases} \tag{1.5}$$

**Formulation MFIE**

D'une manière similaire une formulation intégrale peut être obtenue à partir du champ magnétique $H$. Rappelons que

$$(n \wedge H^{inc})(y) = C_\Omega(y) \, j(y) + (H \wedge n)(y)$$

où $C_\Omega(y) = 1 - \frac{\Omega}{4\pi}$ pour $\Omega$ angle solide en $y$ à $\Gamma$, et $C_\Omega(y) = \frac{1}{2}$ lorsque $\Gamma$ est régulière en $y$. Nous considérerons $C_\Omega \equiv \frac{1}{2}$ sur tout $\Gamma$, puisque les points anguleux n'interviendront pas dans la discrétisation choisie.

Ceci conduit à l'équation nommée MFIE (Magnetic Field Integral Equation)

$$\forall y \in \Gamma, \quad (n \wedge H^{inc})(y) = \frac{j(y)}{2} + n(y) \wedge \int_\Gamma \nabla_x G(x-y) \wedge j(x)) \, \mathrm{d}s(x)$$

**Formulation CFIE**

Cependant, il convient de remarquer que ces formulations souffrent d'un problème dit de résonance, donnant lieu à un problème d'unicité de la solution pour certaines valeurs du nombre d'onde $\kappa$. Ainsi pour ces valeurs de $\kappa$, le système linéaire associé devient non inversible. On peut montrer qu'une combinaison convexe des équations EFIE et MFIE, nommée CFIE (Combined Field Integral Equation) donne une équation exempte de problème de résonance [55, 56]. Une combinaison usuellement utilisée est

$$CFIE = \alpha EFIE + (1-\alpha)\frac{\imath}{\kappa} MFIE$$

avec $\alpha = 0.2$, valeur empirique donnant de bons résultats à la fois en précision et en conditionnement du système.

Notons que la combinaison de formulations a donné lieu à des développements [69, 68, 67] permettant d'exploiter des propriétés telles que la sensibilité à la résonance, la précision ou même le conditionnement. Ces développements sont plus particulièrement utilisés dans le cas de conducteurs non parfaits.

### 1.1.2 Formulations variationnelles

Une étape préliminaire à la discrétisation des équations intégrales précédemment obtenues pour des objets parfaitement conducteurs est le passage en formulation variationnelle de celles-ci. La plupart des résultats évoqués ici sont notamment disponibles dans [58].

#### Espaces fonctionnels

Pour $s \in \mathbb{R}$, on définit les espaces fonctionnels suivants :

$$H^s_{\mathrm{div}}(\Gamma) = j \in H^s(\Gamma)^3,\ j \cdot n = 0,\ \mathrm{div}_\Gamma j \in H^s(\Gamma)$$

et

$$H^s_{\mathrm{rot}}(\Gamma) = j \in H^s(\Gamma)^3,\ j \cdot n = 0,\ \mathrm{rot}_\Gamma j \in H^s(\Gamma)$$

Il est important de noter que $H^s_{\mathrm{div}}(\Gamma)$ et $H^{-1-s}_{\mathrm{rot}}(\Gamma)$ sont en dualité, et plus particulièrement $H^{-\frac{1}{2}}_{\mathrm{div}}(\Gamma)$ et $H^{-\frac{1}{2}}_{\mathrm{rot}}(\Gamma)$. L'espace $H^{-\frac{1}{2}}_{\mathrm{div}}(\Gamma)$ constitue un espace naturel pour chercher les courants surfaciques. Toutefois, pour une onde incidente suffisamment régulière, on considère que ces courants appartiennent au moins à $H^{\frac{1}{2}}_{\mathrm{div}}(\Gamma)$.

#### Forme variationnelle de l'EFIE

Sachant que l'opérateur induit (suivant $j$) par l'équation de $E$ dans (1.3) transporte $H^{-\frac{1}{2}}_{\mathrm{div}}(\Gamma)$ dans $H^{-\frac{1}{2}}_{\mathrm{rot}}(\Gamma)$, pour tout champ de test $j'$, tangent de l'espace $H^p_{\mathrm{div}}(\Gamma)$ (pour $p \geq -\frac{1}{2}$), une écriture variationnelle de l'EFIE par la méthode de Galerkin donne

$$\begin{aligned}\frac{\imath}{\kappa Z_0}\int_\Gamma E^{inc}(y)\cdot j'(y)\,\mathrm{d}s(y) = &\int_{\Gamma\times\Gamma} G(x-y)\, j(x)\cdot j'(y)\,\mathrm{d}s(x,y)\\ &+\frac{1}{\kappa^2}\int_{\Gamma\times\Gamma}\nabla_y G(x-y)\,\mathrm{div}_\Gamma j(x)\cdot j'(y)\,\mathrm{d}s(x,y)\end{aligned} \tag{1.6}$$

Cette formulation présente le désavantage d'introduire des intégrales au sens de la valeur principale de Cauchy telles que

$$\int_\Gamma \nabla_y G(x-y)\,\mathrm{div}_\Gamma j(x)\,\mathrm{d}s(x)$$

qu'une intégration par parties nous permet de dépasser

$$\frac{\imath}{\kappa Z_0}\int_\Gamma E^{inc}(y)\cdot j'(y)\,\mathrm{d}s(y) = \int_{\Gamma\times\Gamma} G(x-y)\left(j(x)\cdot j'(y) - \frac{1}{\kappa^2}\mathrm{div}_\Gamma j(x)\cdot \mathrm{div}_\Gamma j'(y)\right)\,\mathrm{d}s \tag{1.7}$$

#### Forme variationnelle de la MFIE

La formulation variationnelle de la MFIE nous incite à tester $n\wedge H^{inc}$ contre des fonctions test de $H_{\mathrm{div}}$, ce qui est équivalent à tester $H^{inc}$ contre des fonctions de $H_{\mathrm{rot}}$.

Ainsi en prenant $j' \in H^p_{\mathrm{div}}$ $(p \geq \frac{1}{2})$, nous obtenons

$$\begin{aligned}\int_\Gamma (n\wedge H^{inc})(x)\cdot j'(y)\,\mathrm{d}s(y) = &\ \frac{1}{2}\int_\Gamma j(y)\cdot j'(y)\,\mathrm{d}s(y)\\ &+\int_\Gamma j'(y)\cdot n(y)\wedge\left(\int_\Gamma \nabla_x G(x-y)\wedge j(x))\,\mathrm{d}s(x)\right)\,\mathrm{d}s(y)\end{aligned} \tag{1.8}$$

### 1.1.3 Discrétisation

À partir des précédentes formes variationnelles faibles, il convient de décomposer les fonctions $j$ et $j'$ via une base d'éléments finis qui nous servira de support à la projection des équations. À cette fin, nous considérons l'élément fini de Raviart-Thomas. Cet élément d'arête, conforme pour l'opérateur de divergence surfacique $\mathrm{div}_\Gamma$, est souvent désigné comme fonction de base de Rao-Wilson-Glisson [59].

Considérons une triangulation $\mathcal{T}$ de la surface $\Gamma$. À chaque arête $i$ de $\mathcal{T}$, à l'intersection de deux triangles $T^+$ et $T^-$, on associe une fonction de base $J_i$ définie par

$$J_i(x) = \begin{cases} +\frac{1}{2|T^+|}(x - S^+) & \text{si } x \in T^+ \\ -\frac{1}{2|T^-|}(x - S^-) & \text{si } x \in T^- \end{cases}$$

avec $S^+$ et $S^-$ sommets opposés à l'arête $i$ dans $T^+$ et $T^-$, et $|T|$ l'aire du triangle $T$.

Ainsi, il est possible de chercher une solution approchée $J$ à notre problème dans l'espace engendré par $\{J_i\}$ où $i$ désigne une arête de $\mathcal{T}$

$$J = \sum_i \lambda_i J_i.$$

En appliquant cette décomposition à $j$ et aux fonctions test $j'$ dans la formulation variationnelle de l'EFIE, nous obtenons le système linéaire $M^{EFIE}\lambda^{EFIE} = b^{EFIE}$ suivant à résoudre :

$$M^{EFIE}_{i,j} = \int_{\Gamma\times\Gamma} G(x-y)\Big(J_i(x)\cdot J_j(y) - \frac{1}{\kappa^2}\mathrm{div}_\Gamma J_i(x)\ \mathrm{div}_\Gamma J_j(y)\Big)\ \mathrm{d}s(x)\ \mathrm{d}s(y)$$

$$b^{EFIE}_i = \frac{\imath}{\kappa Z_0}\int_\Gamma J_i(x)\cdot E^{inc}(x)\ \mathrm{d}s(x)$$

La matrice $M^{EFIE}$ issue de l'EFIE est une matrice symétrique (mais non hermitienne). Cette formulation est largement utilisée et permet de traiter la plupart des problèmes dont les problèmes portant sur des objets ouverts.

Par ailleurs, cette méthode appliquée à la MFIE conduit au système linéaire

$$M^{MFIE}\lambda^{MFIE} = b^{MFIE}$$

défini par

$$M^{MFIE}_{i,j} = \frac{1}{2}\int_\Gamma J_i(x)\cdot J_j(x)\ \mathrm{d}s(x) + \int_\Gamma J_i(x)\cdot n(x)\ \mathrm{d}s(x) \wedge \int_\Gamma \nabla_x G(x-y)\wedge J_j(y)\ \mathrm{d}s(y) \tag{1.9}$$

$$b^{MFIE}_i = \int_\Gamma J_i(x)\cdot(n(x)\wedge H^{inc})\ \mathrm{d}s(x) \tag{1.10}$$

Notons que cette formulation MFIE, contrairement à la précédente formulation EFIE, ne permet de traiter que des objets fermés, car nécessitant une orientation $n$ de la surface $\Gamma$.

La forme matricielle de la CFIE est tout simplement obtenue comme une combinaison convexe des formes matricielles de l'EFIE et de la CFIE.

$$M^{CFIE} = \alpha M^{EFIE} + (1-\alpha)\frac{\imath}{\kappa}M^{MFIE}$$

$$b^{CFIE} = \alpha b^{EFIE} + (1-\alpha)\frac{\imath}{\kappa}b^{MFIE}$$

## 1.2 Résolution du système

La résolution d'un système linéaire de type $A\,x = b$ avec $A$ une matrice de dimension $N$ est un problème très courant mais complexe dès que $A$ devient dense et $N$ grand.

S'oppose alors deux grandes familles de méthodes : les méthodes directes, consistant en la détermination de $x$ directement à partir de $A$ et $b$, et les méthodes itératives fournissant une suite de valeurs $x_i$ convergeant vers la valeur $x$.

### 1.2.1 Méthodes directes

Les méthodes de résolution directe s'orientent principalement vers une re-formulation de la matrice $A$ sous une forme factorisée permettant alors une détermination simple de $x$ à partir du second membre $b$.

Une méthode directe communément utilisée est la factorisation LU. La factorisation LU consiste en la construction de deux matrices triangles $L$ et $U$ respectivement inférieure et supérieure telles que $A = LU$. Une fois la factorisation effectuée, la construction de $x$ vient de la résolution successive des systèmes $L\,y = b$ et $U\,x = y$. Du fait que les matrices $L$ et $U$ sont triangulaires, la résolution des systèmes linéaires associés est directe et d'un coût en $O(N^2)$ (avec une petite constante). Ainsi, une fois la décomposition LU établie, la détermination de $x$ en fonction de $b$ est rapide.

Toutefois, la construction de la factorisation $A = LU$ est d'un coût CPU en $O(N^3)$. Par ailleurs, l'espace mémoire nécessaire au traitement de telles matrices est de l'ordre de $N^2$, ce qui pour une matrice de taille $N = 50\,000$, en nombres complexes sur $2 \times 64$ bits, représente déjà 40Go. Avec $N = 200\,000$, nous atteignons alors plus de 600Go ! Les temps de traitements et les limites matériels incitent alors à employer d'autres méthodes de résolution approchée mais beaucoup moins coûteuses à défaut de pouvoir déterminer rapidement la solution exacte.

### 1.2.2 Méthodes itératives

Le principe des méthodes itératives est fort simple. Au lieu de déterminer directement la solution $x$ du problème $A\,x = b$, les méthodes itératives proposent une succession de valeurs $x^{(i)}$ convergeant vers la valeur $x$. Certaines de ces méthodes peuvent toutefois, théoriquement, atteindre la solution $x$ en un certain nombre d'itérations.

Le principal avantage des méthodes itératives est de pouvoir se satisfaire d'une solution approchée suivant une estimation du résidu $r^{(i)} = \|b - A\,x^{(i)}\|$.

Une famille de méthodes itératives sont les méthodes de Krylov. Ce type de méthode cherchent à déterminer $x^{(0)}$ :

$$x_k \in x^{(0)} + \mathrm{Span}(r^{(0)}, A\,r^{(0)}, A^2 r^{(0)}, \ldots, A^k r^{(0)})$$

tel que

$$\|r^{(k)}\| = \|A\,x^{(k)} - b\| = \min_{x \in x^{(0)} + \mathrm{Span}(r^{(0)}, A\,r^{(0)}, \ldots, A^k r^{(0)})} \|A\,x - b\|.$$

Ce type de méthode permettent une résolution en temps de l'ordre de $N^{iter}(c_1 N^2 + C(N,k))$ avec $N^{iter}$ le nombre d'itération, $c_1$ le nombre de produit smatriciels nécessaire par itération et $C(N,k)$ le coût de la résolution d'un problème de minimisation de taille $N$ dans un espace de dimension $k$. Nous voyons dès lors que $N^{iter} \ll N$, le gain en temps de calcul peut être très important.

**GMRES : Generalized Minimal Residual**

Dès que la matrice $A$ n'est plus hermitienne, la détermination de $x^{(k)}$ devient complexe. La méthode GMRES [64] permet de traiter la résolution de systèmes matriciels formés autour de matrices non symétriques définies positives.

GMRES construit une approximation de la solution à partir d'un espace de Krylov dont la dimension croît avec les itérations. C'est pour éviter des problèmes de saturation de la mémoire qu'une variante de GMRES propose un redémarrage de l'algorithme dès qu'une taille critique est atteinte avec comme valeur initiale, la dernière approximation calculée. Cette taille limite est nommée dimension de Krylov ou «restart» que nous noterons $m$.

Notons que GMRES est une méthode efficace tant que le nombre d'itérations ne dépasse pas $m$, ce qui incite à l'utilisation de préconditionneur. Dans notre étude nous nous bornerons à l'utilisation de la formulation CFIE (avec une contrainte de surface fermée) qui donne un bien meilleur conditionnement de la matrice que peut proposer la formulation EFIE.

GMRES(m,$\epsilon$,$x^{(0)}$) préconditionné par $P$

**boucle**

Résoudre $P\,r = b - A\,x^{(0)}$

$v^{(1)} = r/\|r\|_2$ {Initialisation de la base de Krylov}

$z_1 = |r|_2$

**pour** $i = 1, 2, \ldots, m$ **faire**

Résoudre $P\,w = A\,v^{(i)}$

**pour** $k = 1, \ldots, i$ **faire**

$h_{k,i} = \langle w, v^{(k)} \rangle$ {Orthonormalisation de Gram-Schmidt}

$w = w - h_{k,i} v^{(k)}$

**fin pour**

$h_{i+1,i} = \|w\|_2$

$v^{(i+1)} = w/h_{i+1,i}$ {Vecteur de Krylov suivant}

**pour** $k = 1, \ldots, i-1$ **faire**

$$\begin{pmatrix} h_{k,i} \\ h_{k+1,i} \end{pmatrix} = \begin{pmatrix} c_k & s_k \\ -s_k & c_k \end{pmatrix} \times \begin{pmatrix} h_{k,i} \\ h_{k+1,i} \end{pmatrix}$$

**fin pour**

$\alpha = \sqrt{h_{i,i}^2 + h_{i+1,i}^2}$

$s_i = h_{i+1,i}/\alpha$, $c_i = h_{i,i}/\alpha$ {Construction de la prochaine rotation de Givens}

$h_{i,i} = \alpha$, $h_{i+1,i} = 0$

$z_{i+1} = -s_i\, z_i$, $z_i = c_i\, z_i$

**si** $|z_{i+1}| < \epsilon$ **alors**

$$\begin{pmatrix} y_1 \\ \vdots \\ y_i \end{pmatrix} = \begin{pmatrix} h_{1,1} & \ldots & h_{1,i} \\ \vdots & \ddots & \vdots \\ 0 & \ldots & h_{i,i} \end{pmatrix}^{-1} \begin{pmatrix} z_1 \\ \vdots \\ z_i \end{pmatrix}$$

Fin de GMRES : $x = x^{(0)} + \sum_{k=1}^{i} y_k\, z_k$

**fin si**

**fin pour**

$$\begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} = \begin{pmatrix} h_{1,1} & \ldots & h_{1,m} \\ \vdots & \ddots & \vdots \\ 0 & \ldots & h_{m,m} \end{pmatrix}^{-1} \begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix}$$

$x = x^{(0)} + \sum_{k=1}^{m} y_k\, z_k$

**si** $\|b - A\,x\|_2 < \epsilon$ **alors**

Fin de GMRES : $x$

**sinon**

$x^{(0)} = x$ {Redémarrage}

**fin si**

**fin boucle**

# Chapitre 2

# Principes généraux de la Méthode Multipôle Rapide

**Sommaire**



Préalablement à une étude plus approfondie de la méthode multipôle rapide appliquée aux équations de Maxwell, nous allons donner les idées générales de cette méthode. Cette présentation suit une certaine vue des méthodes multipôles, particulièrement sous une forme que nous utiliserons par la suite. Beaucoup de variantes sont possibles pouvant impliquer des étapes complémentaires ou des discrétisations construites sur d'autres bases, mais l'idée de fond reste la même.

## 2.1 Introduction historique

Introduite par V. Rokhlin pour résoudre les équations intégrales de Laplace en 2D dans [61], la Méthode Multipôle Rapide (F*ast* M*ultipole* M*ethod*, FMM) se raffine et s'étend à d'autres champs d'applications (Laplace 3D, problème à N Corps à potentiels colombiens) grâce au travail de L. Greengard [36, 34]. V. Rokhlin utilise la FMM en conjonction avec un solveur itératif afin de réduire la complexité du produit matrice-vecteur de $O(N^2)$ jusqu'à $O(N)$ pour l'équation de Laplace [35] et $O(N^{3/2})$ pour l'équation de Helmholtz [63, 17]. V. Rokhlin utilise alors les moments multipolaires pour représenter des groupes distants de particules et introduit l'expansion locale pour évaluer la contribution d'une particule distante sous la forme d'une série. Le moment multipolaire associé à un groupe distant peut être translaté en coefficients d'une expansion locale associée à un groupe. Parallèlement au travail de V. Rokhlin, L. Greengard introduit la décomposition hiérarchique de l'espace (quad-tree, octree) s'appuyant sur une structure arborescente systématique des groupes.

Durant la dernière décennie, la FMM a été appliquée à de nombreux champs : astrophysique, chimie quantique (équations de Hartree-Fock), méthode des vortex en mécanique des fluides, électromagnétisme...

Son élection dans le top 10 des algorithmes du vingtième siècle montre son influence sur l'analyse numérique [6] (*cf* Annexe F).

## 2.2 Illustration des principes de la FMM : le problème à N corps

Considérons $N$ corps (particules, objets, astres) répartis dans un espace borné. L'objectif est de calculer pour chaque corps l'action induite sur lui par les autres corps, où l'interaction entre corps est définie suivant un certain potentiel (gravitation, électrostatique...).

Un algorithme naïf consiste à prendre les corps 2 à 2 et à en calculer les interactions. Cet algorithme conduit à un calcul de complexité $O(N^2)$.

Toutefois, beaucoup de potentiels usuels vérifient une certaine propriété de régularité, permettant ainsi une forme d'approximation des potentiels à longues distances.

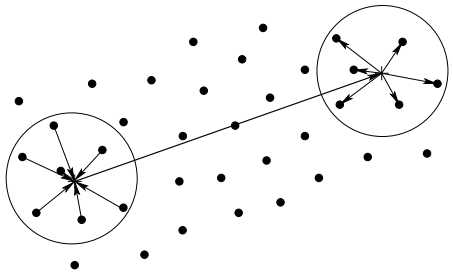


FIG. 2.1 – Problème à $N$ corps

Il est alors imaginable de calculer ces interactions pour une précision $\epsilon$ donnée, suivant l'algorithme :

1. Définir des regroupements de corps suivant un critère de distance.
   Chaque groupe est soit *proche* soit *lointain* de chaque autre groupe ; les concepts *proche* et *lointain* sont liés à la validité de l'approximation longue distance des potentiels.
2. Les interactions entre groupes *proches* sont effectuées par l'algorithme naïf, 2 à 2 parmi tous les corps des 2 groupes.
3. Agrégation : chaque groupe doit pouvoir se résumer en une caractérisation globale des positions et valeurs des potentiels induits des corps le composant. Cette caractérisation doit être transposable, *i.e.* on doit pouvoir retrouver (tout du moins approximativement) les valeurs en fonction des positions des corps formant le groupe.
4. Transfert : les interactions entre groupes lointains sont calculées à partir des caractérisations globales, 2 à 2 parmi les groupes lointains.
5. Synthèse : chaque corps déduit l'action des autres corps sur lui, par cumul des actions des autres groupes sur son groupe puis par inversion de la caractérisation globale.

Cet algorithme nécessiterait beaucoup de précisions quant aux différentes approximations et formulations, mais résume assez bien la méthode multipôle à 1 niveau dont la complexité se situe aux alentours de $N^{3/2}$. Toutefois, pour obtenir le qualificatif *rapide* et une complexité en $N \log N$ (tout comme les transformées de Fourier *rapides*, le *quick*sort...), il convient de

hiérarchiser cette méthode, en traitant les interactions de groupes *proches* par une même approche mais à un niveau plus fin (et récursivement).

## 2.3 Survol de l'algorithme Multipôle Rapide

Cette section reprend les idées de la précédente section mais en y introduisant les équations et les propriétés en jeu, toujours sans un cadre abstrait.

Tout d'abord, considérons un potentiel $G(r)$ ne dépendant que de la distance $r = |x - y|$ entre deux points $x$ et $y$.

L'objectif est de calculer sur chaque particule $x_i$ l'action $u_i$ de l'ensemble des autres particules de position $x_j$ et d'intensité $v_j$,

$$u_i = \sum_{j \neq i} G(x_i - x_j)\, v_j$$

qui peut aussi s'écrire via le système linéaire suivant

$$u = M_G\, v, \quad \text{avec } (M_G)_{ij} = G(x_i - x_j)$$

Un premier prérequis est la possibilité de décomposer $G$ sous une forme :

$$G(r) = \int K(\zeta, s)\, T(\zeta, t)\, \mathrm{d}\zeta, \quad \text{avec } r = s + t \in \mathbb{R}^3 \tag{2.1}$$

ou de façon discrète

$$G(r) = \sum_{k=1}^{L} K_k(s)\, T_k(t), \quad \text{avec } r = s + t \in \mathbb{R}^3 \tag{2.2}$$

Par ailleurs, il sera utile que $K$ possède une propriété d'«additivité»

$$K(\zeta, r_1 + r_2) = K(\zeta, r_1) \cdot K(\zeta, r_2) \ \text{ pour } r_1, r_2 \in \mathbb{R}^3 \tag{2.3}$$

ou sous forme discrète

$$K_k(r_1 + r_2) = K_k(r_1) \cdot K_k(r_2) \ \text{ pour } r_1, r_2 \in \mathbb{R}^3 \tag{2.4}$$

**Remarque** *Cette propriété peut toutefois ne pas être vérifiée directement par certaines formes des méthodes multipôles, l'opérateur $\cdot$ pouvant alors recouvrir des formes plus complexes qu'une simple multiplication. De même l'opérateur de transfert $T$ peut s'avérer complexe. A des fins de performances, nous tenterons de toujours de réduire $T$ à un opérateur de transfert diagonal.*

Voyons comment ces quelques propriétés permettent de construire une méthode multipôle rapide.

### FMM à 1 niveau

Afin de pas trop nous éloigner de notre futur cadre d'application, que sera le calcul du champ électrique induit par une onde électromagnétique incidente à la surface d'un objet, nous ne considérerons ici que des points sur une surface $\Gamma$, et, afin de simplifier les illustrations, nous nous restreindrons à un cadre bidimensionnel. Une extension à d'autres dimensions suit les mêmes principes.

Tout d'abord, nous supposons que tous les points sont inclus dans une boîte englobante (qui sera assimilable à un cube pour plus de commodité).

Il convient de diviser la boîte englobante en cellules $P_r$ (aussi nommées clusters ou paquets) de préférence cubiques et de tailles identiques.

Dès lors il convient de distinguer les cellules *proches* et *lointaines* (aussi nommées *distantes*). Chaque paquet $P_r$ est borné par une sphère $S_r$ de centre $C_r$ et de rayon $R_r = \max_{x_i \in P_r} |x_i - C_r|$. Nous définissions comme proches deux paquets $P_r$, $P_s$ vérifiant :

$$|C_r - C_s| \leq \alpha_0 (R_r + R_s)$$

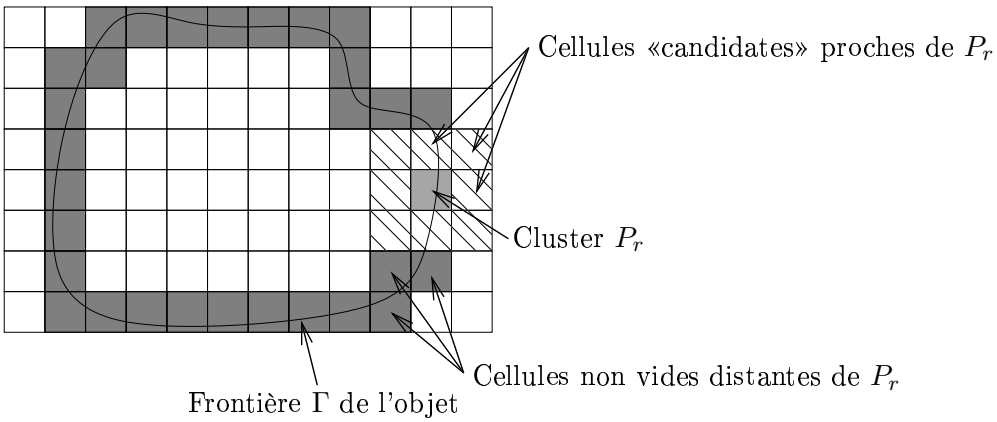
pour une certaine valeur de $\alpha_0$ indépendante des paquets considérés. Cependant, certaines formulations peuvent soit assouplir soit renforcer cette condition suivant la validité de l'approximation multipôle utilisée pour les cellules lointaines.

Via cette décomposition, un produit par la matrice $M_G$ d'un vecteur $v$ peut être écrit comme la somme des interactions lointaines et des interactions proches $u = u^{close} + u^{far}$

$u^{close} = M^{close} v$ avec $M^{close}_{ij} = M_{ij}$ si $x_i$ et $x_j$ sont *proches*, 0 sinon,

$u^{far} = M^{far} v$ avec $M^{far}_{ij} = M_{ij}$ si $x_i$ et $x_j$ ne sont pas *proches* (i.e. *distants*), 0 sinon.

Les méthodes multipôles rapides sont un moyen efficace de *compresser* le produit par $M^{far}$, où la *compression* porte autant sur le stockage mémoire que la complexité algorithmique. Néanmoins, il convient de prendre en compte la matrice $M^{close}$ qui ne peut suivre cet algorithme (contrainte de validité de l'approximation) et requiert alors un calcul exact mais toutefois réduit.




Ainsi nous décomposons le produit $u^{far} = M^{far} v$ en 3 étapes :

– Calcul des fonctions de radiation $F^r_k$ pour chaque cluster $P_r$ et chaque indice $k \in [1 : L]$

$$F^r_k \stackrel{\text{def}}{=} \sum_{x_j \in P_r} v_j \; K_i(C_r - x_j). \tag{2.5}$$

– Transfert entre le cluster $P_r$ et ses clusters distants

$$G_k^r = \sum_{P_t \text{ distant de } P_r} F_k^t \, T_k(C_r - C_t). \tag{2.6}$$

– Calcul de $u_i \in P_r$

$$u_i = \sum_{k=1}^{L} G_k^r \, K_k(x_i - C_r). \tag{2.7}$$

L'algorithme FMM mono-niveau est alors de complexité $O(L\times N)+O(L\times N^{3/2})+O(L\times N)$ soit $O(N^{3/2})$ pour $N$ points répartis uniformément sur $\Gamma$ en $\sqrt{N}$ paquets de taille $\sqrt{N}$ où le paramètre $L$ dépend essentiellement de la qualité de l'approximation, *i.e.* du nombre de termes requis.

### Méthode Multiniveaux

Remarquons que dans la méthode à mono-niveau, le nombre de transferts (fonctions $G_\cdot^r$) croît exponentiellement lorsque la taille des paquets diminue (afin de réduire les interactions proches). La méthode multiniveaux permet, grâce à une décomposition hiérarchique de l'espace, un calcul hiérarchique de l'étape de transfert.

Une telle décomposition hiérarchique de l'espace peut être obtenue par des arbres (arbres binaires (1D), quad-trees (2D), octrees (3D) ...)

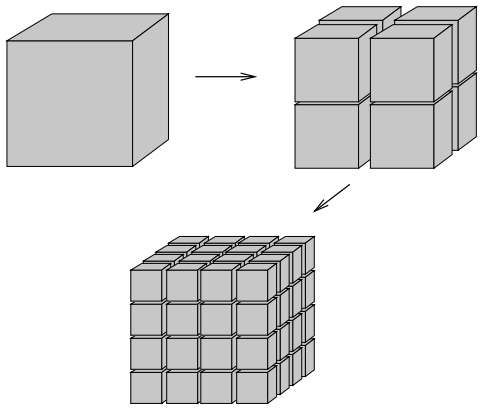


FIG. 2.2 – Décomposition hiérarchique en dimension 3 par l'octree

Les relations hiérarchiques induites par la structure arborescente définissent la notion de parenté. Ainsi, nous nommerons une cellule $A$ «mère» (*resp.* «parent», «père») de la cellule «fille» $B$ (*resp.* «enfant», «fils»), si $B$ est une descendante directe de $A$ au sens de la hiérarchie induite par l'arbre.

Nous allons aborder cette présentation par la forme continue (2.1). Ensuite, nous passerons à la forme discrète qui comporte une part non négligeable dans la complexité de l'algorithme final.

Considérons une frontière courbe $\Gamma$ plongée dans un quad-tree à 3 niveaux, nommés respectivement $L_0$, $L_1$, $L_2$. Nous allons calculer les interactions relatives à trois paquets emboîtés $P_{r_0}$, $P_{r_1}$ and $P_{r_2}$ respectivement des niveaux $L_0$, $L_1$, $L_2$.

Tout comme la FMM mono-niveau, l'algorithme peut se décomposer en 3 étapes.

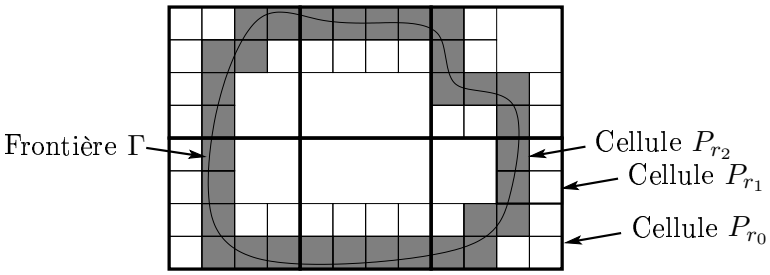



FIG. 2.3 – $P_{r_2} \subset P_{r_1} \subset P_{r_0}$

**«Multipole to Multipole»**

La première étape est le calcul des fonctions de radiation pour chaque paquet $P_t$ et à chaque niveau :

$$F_t(\zeta) \stackrel{\text{def}}{=} \sum_{x_j \in P_t} v_j \; K(\zeta, C_r - x_j). \tag{2.8}$$

**«Multipole to Local»**

Il convient ensuite d'effectuer les transferts nécessaires aux différents niveaux en partant, par exemple, du niveau le plus grossier, $L_0$.

– Le niveau $L_0$ requiert les transferts suivants :

$$G_{r_0}(\zeta) = \sum_{\substack{P_{t_0} \text{ lointain de } P_{r_0} \\ P_{r_0} \text{ et } P_{t_0} \text{ au même niveau}}} F_{t_0}(\zeta) \; T(\zeta, C_{r_0} - C_{t_0}).$$

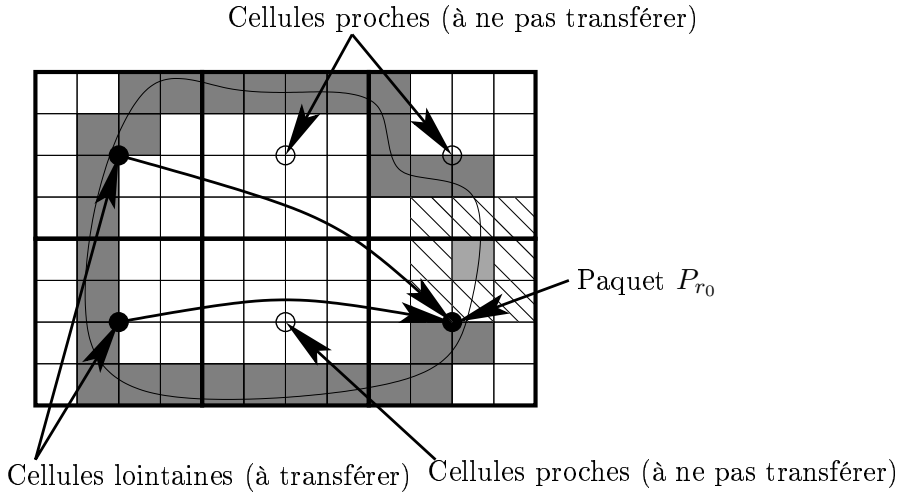



FIG. 2.4 – Transferts au niveau $L_0$

– Nous pouvons alors remarquer qu'un grand nombre d'interactions lointaines avec le paquet $P_{r_1}$ ont indirectement déjà été calculées au niveau $L_0$. Il conviendra donc d'incorporer une étape à l'algorithme pour les prendre en compte.

Nous définissons de manière générale les transferts à effectuer par une cellule $P_r$ par

$$\mathcal{T}(P_r) = \{P_t \text{ lointain de } P_r \text{ et père}(P_t) \text{ proche de père}(P_r)\} \tag{2.9}$$

où père$(X)$ représente la cellule (unique) telle que $X$ en soit un fils.

$$G_{r_1}(\zeta) = \sum_{P_{t_1} \in \mathcal{T}(P_{r_1})} F_{t_1}(\zeta)\, T(\zeta, C_{r_1} - C_{t_1}) \tag{2.10}$$

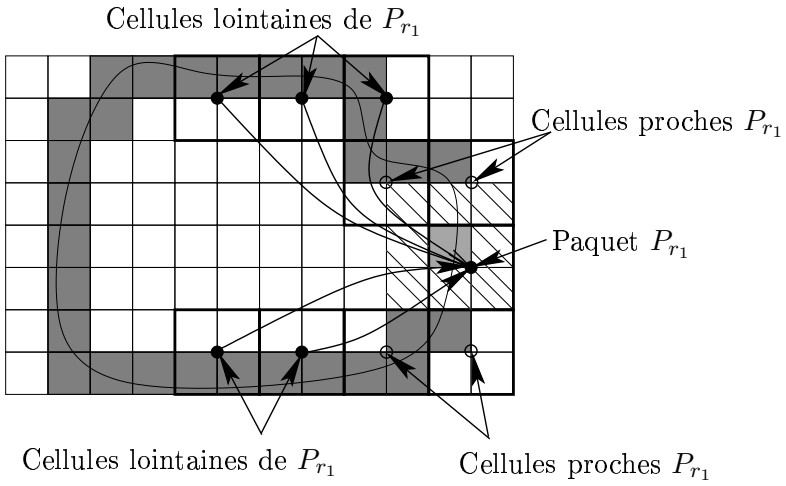



FIG. 2.5 – Transferts au niveau $L_1$

– Au niveau le plus fin, $L_2$, il reste alors beaucoup moins de transferts pour la même raison qu'au niveau 1.

$$G_{r_2}(\zeta) = \sum_{P_{t_2} \in \mathcal{T}(P_{r_2})} F_{t_2}(\zeta)\, T(\zeta, C_{r_2} - C_{t_2}) \tag{2.11}$$

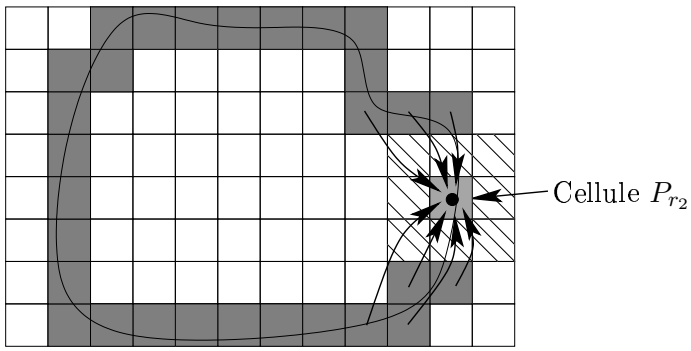



FIG. 2.6 – Transferts au niveau $L_2$

**«Local to Local»**

Il reste alors à propager les transferts indirectement calculés aux niveaux $L_0$ et $L_1$ nécessaires au paquet $P_{r_2}$.

C'est encore grâce à la propriété d'additivité (2.3) qu'il nous est possible d'appliquer une translation entre centres d'expansions multipolaires

$$\begin{aligned} \tilde{G}_{r_1}(s) &= G_{r_1}(s) + K(\zeta, C_{r_1} - C_{r_0})\, G_{r_0}(s), \\ \tilde{G}_{r_2}(s) &= G_{r_2}(s) + K(\zeta, C_{r_2} - C_{r_1})\, \tilde{G}_{r_1}(s). \end{aligned}$$

Il nous est alors possible de reconstruire $u^{far} = M^{far} v$

$$u_i^{far} = \int K(\zeta, x_i - C_{r_2})\, \tilde{G}_{r_2}(\zeta) d\zeta \tag{2.12}$$

pour tout point $x_i \in P_{r_2}$.

Il est déjà possible de faire quelques remarques :

1. L'étape d'initialisation construit l'ensemble des fonctions de radiation $F_t$ pour chaque paquet de chaque niveau. Considérons un paquet $P_{t_l}$ du niveau $l$ ayant les cellules $P^s_{t_{l+1}}$ au niveau $l+1$ comme *fils*. La construction des fonctions de radiation $F_t$ est définie à partir des valeurs et positions aux points $x_i \in P_t$. La fonction de radiation $F_{t_l}$ peut alors être définie à partir des fonctions $F_{t^s_{l+1}}$

$$F_{t_l}(\zeta) = \sum_{P^s_{t_{l+1}} \text{ fils de } P_{t_l}} F_{t^s_{l+1}}(\zeta)\, K(\zeta, C_{t_l} - C_{t^s_{l+1}}). \tag{2.13}$$

   Ceci permet de réduire considérablement le coût de l'initialisation mais nécessite de construire la FMM en partant des feuilles vers la racine.

2. Si l'on considère une forme discrète du précédent algorithme (2.2), il est tout à fait envisageable que la discrétisation utilisée diffère pour chaque niveau, le domaine de validité de ces formes étant lié à la taille des cellules du niveau (penser au rayon de convergence pour les séries entières). Ainsi, tous les précédents calculs mettant en jeu différents niveaux peuvent ne pas être directement compatibles au niveau discret. Il est alors nécessaire de passer par des opérateurs d'interpolation rendant compatibles les différentes formes discrètes. Ce point est très spécifique à la formulation utilisée et sera détaillé au fil de cet exposé.

# Chapitre 3

# Formulation Hautes Fréquences pour l'électromagnétisme

**Sommaire**



Comme nous l'avons vu dans le chapitre 1, une résolution des équations de Maxwell par formulation intégrale, couplée à une méthode itérative requiert le calcul de nombreux produits du type matrice-vecteur.

Par ailleurs dans le chapitre 2, nous avons vu quelques principes des méthodes multipôles rapides, leurs prérequis et leur efficacité algorithmique.

Dans ce chapitre, nous allons voir comment il est possible de construire une méthode multipôle rapide pour l'électromagnétisme en dimension 3. Toutefois, la démarche adoptée comportera quelques limitations et restreindra la plage d'application de cette méthode aux hautes fréquences. Une alternative sera exposée dans le chapitre 4 en tant que nouvelle méthode adaptée aux basses fréquences sans toutefois prendre en possibilités en hautes fréquences. Cette méthode pourra alors être qualifiée de *toutes fréquences*.

À propos de cette formulation hautes fréquences, il conviendra de noter qu'une vaste littérature traite déjà en profondeur de cette méthode [21, 11, 74, 72, 52, 62, 63]. J'invite le lecteur à s'y référer pour de plus amples détails.

## 3.1 Formule fondamentale de la HF-FMM

En partant du noyau de Green

$$G(r) = \frac{e^{\imath\kappa|r|}}{4\pi|r|}$$

la méthode multipôle rapide requiert qu'une séparation des variables puisse être appliquée ; en reprenant les termes du chapitre 2, il convient de trouver des fonctions $K$ et $T$ telles que

$$G(r) = \int K(\zeta, s)\, T(\zeta, t)\, \mathrm{d}\zeta, \qquad \text{pour } r = s + t \tag{3.1}$$

où $K$ vérifie une propriété d'additivité

$$K(\zeta, r_1 + r_2) = K(\zeta, r_1) K(\zeta, r_2). \tag{3.2}$$

Le théorème de Gegenbauer [1] propose le point de départ suivant :

**Théorème 2** *de Gegenbauer (forme simplifiée)*
*Si* $|x| > |y|$ *alors*

$$\frac{e^{\imath\kappa|x+y|}}{|x+y|} = \imath\kappa \sum_{l=0}^{\infty} (2l+1)(-1)^l\, h_l^{(1)}(\kappa|x|)\, j_l(\kappa|y|)\, P_l(\hat{x}\cdot\hat{y}). \tag{3.3}$$

*où* $h_l^{(1)}$ *est la fonction de Hankel sphérique du premier type de rang* $l$, $j_l$ *la fonction de Bessel sphérique de première espèce d'ordre* $l$, $P_l$ *le polynôme de Legendre d'ordre* $l$.

Pour l'ensemble de ces fonctions *spéciales*, nous invitons le lecteur à se reporter aux annexes (en particulier C) ou bien aux ouvrages spécialisés [1, 80, 54].

Cette forme du théorème de Gegenbauer ne sépare pas encore complètement les variables $x$ et $y$, mais déjà pose le problème de la troncature de la série tout en garantissant un contrôle de l'erreur.

On sait déjà que cette série converge dès que $|x| > |y|$. Il a par ailleurs été montré qu'un contrôle uniforme de l'erreur de troncature pouvait être obtenu sous la condition nécessaire suivante :

**Condition 1** *Convergence uniforme de la série de Gegenbauer*
*Pour tout* $\alpha_0 > 1$ *et* $y_0 \in \mathbb{R}^3$, *la série*

$$\imath\kappa \sum_{l=0}^{\infty} (2l+1)(-1)^l\, h_l^{(1)}(\kappa|x|)\, j_l(\kappa|y|)\, P_l(\hat{x}\cdot\hat{y}). \tag{3.4}$$

*converge uniformément sur* $\{(x,y) \in \mathbb{R}^3, |y| \leq |y_0| \text{ et } |x| \geq \alpha_0|y_0|\}$

Un contrôle simple de l'erreur de troncature a été donné dans [21] pour $\alpha_0 \geq \frac{\sqrt{5}}{2}$.

D'efficaces estimations uniformes sont par ailleurs décrites dans [11] dès que $\alpha_0 > 1$.

Ce type de conditions s'inscrit naturellement dans l'algorithme via la décomposition suivante :

**Condition 2** *Réécriture de la condition 1 en terme FMM*
*Soient deux paquets* $P_r$ *et* $P_t$ *de centres et rayons respectifs* $C_r$, $R_r$ *et* $C_t$, $R_t$. *Soient deux points* $x_r \in P_r$ *et* $x_t \in P_t$. *Posons* $r_0 = |C_r - C_t|$ *et* $r = R_r + R_t$, *alors la condition 1 s'écrit*

$$r_0 \geq \alpha_0 r,$$

*critère qui s'applique aisément dans le cadre FMM précédemment décrit.*

Afin d'accomplir la complète séparation des variables, nous utiliserons la relation de Funk-Hecke.

**Théorème 3** *(Représentation en ondes planes)*

$$4\pi \imath^l j_l(\kappa|y|)\, P_l(\hat{x}\cdot\hat{y}) = \int_{S^2} e^{\imath\kappa\langle y,\sigma\rangle} P_l(\hat{x}\cdot\sigma)\, \mathrm{d}\sigma.$$

**Preuve** *À partir de la relation C.12 et d'une décomposition en Harmoniques Sphériques des polynômes des Legendre $P_l$ (définition 1 page 40)*

$$P_l(\hat{x}\cdot\hat{y}) = \frac{4\pi}{2l+1}\sum_{m=-l}^{l} \overline{Y_l^m}(\hat{x})\, Y_l^m(\hat{y})$$

*et de la formule de Jacobi-Anger (théorème 10 page 40)*

$$e^{\imath\kappa\langle\sigma,x\rangle} = \sum_{p=0}^{\infty}(2p+1)\,\imath^p j_p(\kappa|x|)P_p(\sigma\cdot\hat{x})$$

*nous obtenons*

$$\begin{aligned}
\int_{S^2} e^{\imath\kappa\langle y,\sigma\rangle} P_l(\hat{x}\cdot\sigma)\,\mathrm{d}\sigma &= \frac{4\pi}{2l+1}\sum_{p=0}^{\infty}(2p+1)\,\imath^p j_p(\kappa|y|)\int_{S^2} P_p(\sigma\cdot\hat{y})\sum_{m=-l}^{l}\overline{Y_l^m}(\hat{x})\,Y_l^m(\sigma)\,\mathrm{d}\sigma \\
&= \frac{16\pi^2}{2l+1}\sum_{p=0}^{\infty}\imath^p j_p(\kappa|y|)\int_{S^2}\sum_{m'=-p}^{p}\overline{Y_p^{m'}}(\sigma)\,Y_p^{m'}(\hat{y})\sum_{m=-l}^{l}\overline{Y_l^m}(\hat{x})\,Y_l^m(\sigma)\,\mathrm{d}\sigma \\
&= \frac{16\pi^2}{2l+1}\sum_{m=-l}^{l}\sum_{p=0}^{\infty}\imath^p j_p(\kappa|y|)\sum_{m'=-p}^{p}\overline{Y_l^m}(\hat{x})\,Y_p^{m'}(\hat{y})\int_{S^2}\overline{Y_p^{m'}}(\sigma)\,Y_l^m(\sigma)\,\mathrm{d}\sigma.
\end{aligned}$$

*Par l'orthogonalité des harmoniques sphériques*

$$\int_{S^2}\overline{Y_l^m}(\sigma)\,Y_{l'}^{m'}(\sigma)\,\mathrm{d}\sigma = \delta_{ll'}\delta_{mm'}$$

*nous pouvons simplifier l'expression en*

$$\begin{aligned}
\int_{S^2} e^{\imath\kappa\langle y,\sigma\rangle} P_l(\hat{x}\cdot\sigma)\,\mathrm{d}\sigma &= \frac{16\pi^2}{2l+1}\imath^l j_l(\kappa|y|)\sum_{m=-l}^{l}\overline{Y_l^m}(\hat{x})\,Y_l^m(\hat{y}) \\
&= 4\pi\,\imath^l j_l(\kappa|y|)\,P_l(\hat{x}\cdot\hat{y})
\end{aligned}$$

□

Il est possible d'obtenir une pleine séparation des variables indispensables à l'algorithme FMM :

**Théorème 4** *Théorème fondamental de la formulation Hautes Fréquences (HF-FMM)*

$$Si\ |y|<|x|,\quad \frac{e^{\imath\kappa|x+y|}}{|x+y|} = \lim_{p\to+\infty}\int_{S^2} e^{\imath\kappa\langle\sigma,y\rangle}\ T_{p,\sigma}(x)\ \ \mathrm{d}\sigma \tag{3.5}$$

*où la fonction de transfert $T_{p,\sigma}(r)$ est définie par :*

$$T_{p,\sigma}(x) = \frac{\imath\kappa}{4\pi} \sum_{m=0}^{p} (2m+1)\imath^m h_m^{(1)}(\kappa|x|) P_m(\hat{\sigma}\cdot\hat{x}). \tag{3.6}$$

**Preuve** *Cette formulation est une conséquence directe des théorèmes 2 et 3.* □

## 3.2 Instabilités numériques de la formulation HF-FMM

Remarquons qu'à cause de la présence de la fonction $h_m^{(1)}(\kappa|x|)$ dans la fonction de transfert, $T_{p,\sigma}(x)$ diverge lorsque $p \to +\infty$ ou lorsque $\kappa|x| \to 0$. Ces propriétés conduisent à une instabilité de la méthode en cas d'utilisation en grande précision ou en régime basse fréquence. C'est pourquoi la FMM ainsi construite est qualifiée de Haute Fréquence. De plus, il conviendra de trouver la juste troncature permettant de faire converger (3.5) et d'éviter la divergence de la série (3.6).

Afin d'illustrer les problèmes numériques de cette troncature, intimement liée à la convergence de la série (3.3), précisons déjà que $h_n^{(1)} = j_n + \imath y_n$ décrit la fonction sphérique de Hankel comme une combinaison des fonctions de Bessel sphériques de première et seconde espèce.

Bien que l'on puisse montrer qu'asymptotiquement, pour $l$ grand

$$(2l+1)\, y_l(\kappa|x|)\, j_l(\kappa|y|) \sim \frac{1}{\kappa|y|} \left(\frac{|y|}{|x|}\right)^l$$

laissant présager une convergence pour $|x| > |y|$ (théorème 2), il faut noter que cette convergence oppose la fonction $y_l(x)$ qui explose dès que $l > x$ à la fonction $j_l(y)$ à décroissance très rapide dès que $l > y$ et emportant la convergence du produit $j_l \cdot y_l$ (figures 3.1 et 3.2).

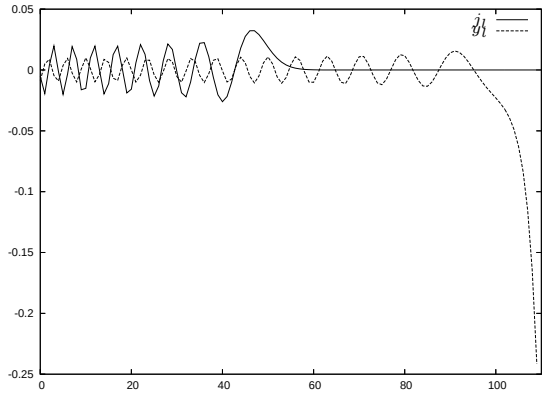



FIG. 3.1 – Variations de $j_l(50)$ et $y_l(100)$

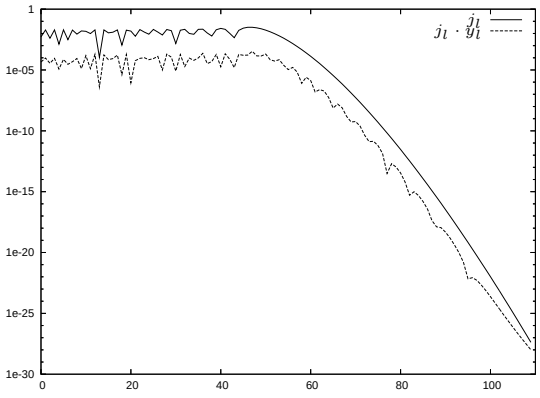



FIG. 3.2 – Comparaison des décroissances de $j_l(50)$ et $j_l(50) \cdot y_l(100)$

Toutefois, dès que l'on sépare les variables (théorème 4), les fonctions $h_l$ apparaissant dans la fonction de transfert (équation 3.6) font apparaître des instabilités numériques du fait de la divergence des fonctions $y_l$.

Ce comportement est problématique en basses fréquences où $\kappa|x|$ est petit ainsi que pour de grands nombres de niveaux où $\kappa|x|$ décroît jusqu'à devenir petit aux niveaux les plus bas.

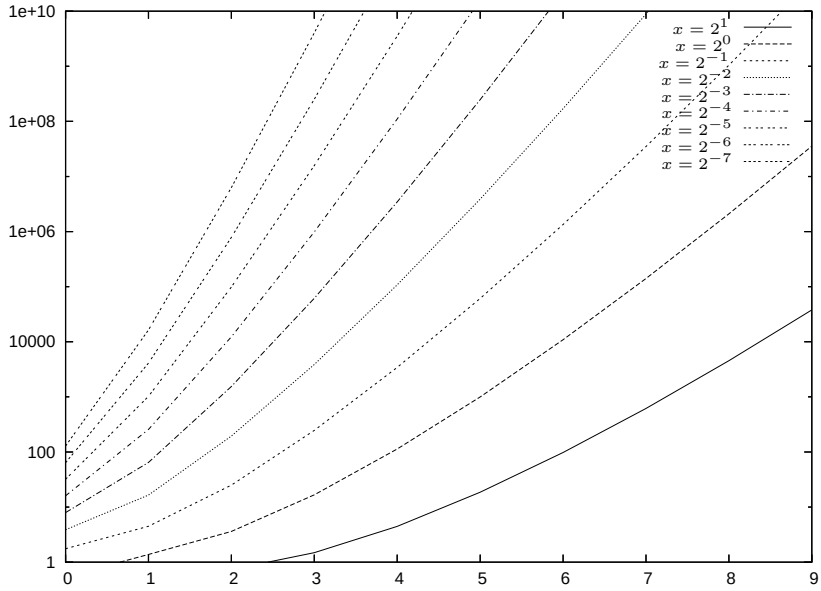



FIG. 3.3 – Explosion des fonctions $y_l(x)$ pour de petits paramètres $x$

En très basses fréquences, la série (3.6) peut même devenir numériquement instable dès $y_1$ (figure 3.3).

C'est dans cette optique que la méthode SPW-FMM *Stable Plane Wave FMM* sera développée à partir du chapitre 4, visant à étendre développer une méthode FMM stable à toutes fréquences.

De même, accroître la précision dans l'évaluation de (3.5) requiert de repousser la troncature ce qui peut encore être numériquement incompatible avec la divergence des fonctions $y_l$.

## 3.3 Troncature de la série

De par la forme de l'approximation du noyau de Green en une limite d'une série (3.5 et 3.6), il convient de tronquer cette série à $L$ termes tel que l'on puisse garantir une erreur $\epsilon$ donnée, sans toutefois provoquer l'explosion du terme de transfert. Ce problème est sujet à de nombreux développements, fournissant des approximations aussi bien empiriques que complexes.

Une fois un choix approprié du paramètre $L$ fait, la relation (3.5) s'utilisera sous la forme

$$\frac{e^{\imath\kappa|x_r-x_t|}}{|x_r-x_t|} \simeq \int_{S^2} e^{\imath\kappa\langle\sigma,x_r-C_r\rangle}\, T_{L,\sigma}(C_r-C_t)\, e^{\imath\kappa\langle\sigma,C_t-x_t\rangle}\quad \mathrm{d}\sigma$$

Tout d'abord, une estimation très répandue du paramètre de troncature $L$ et démontrée dans [51, 21, 22] prend la forme suivante :

**Proposition 5** *Pour tout $\epsilon < 1$, il existe quatre constantes $C_1$, $C_2$, $C_3$, $C_4$ telles qu'une troncature d'ordre $L$ fournisse une erreur inférieure à $\epsilon$ sur l'approximation (3.5) du noyau*

*de Green* $G(|x_i - x_j|) = G(r + r_0)$ *sous la condition 2, avec*

$$L(\epsilon) = C_1 + C_2\kappa r + C_3 \log(\kappa r) + C_4 \log(\epsilon^{-1}).$$

Toutefois, la forme suivante, plus courante mais plus empirique, fournit de bons résultats pour une erreur moyenne de $10^{-3}$ :

$$L \simeq \kappa d + 7.5 \log_{10}(\kappa d + \pi)$$

où $d = \sqrt{3}a$ avec $a$ la longueur de l'arête d'un cube dans la décomposition multiniveaux au niveau où l'on souhaite utiliser l'approximation (3.5).

Il est tout de même possible de faire bien mieux grâce aux résultats suivants :

**Estimation 6** *de Song & Chew [71]*
*L'estimation*

$$L_{SC} = \kappa d + 1.8 d_0^{2/3} (\kappa d)^{1/3}, \quad d_0 = \log_{10}(1/\epsilon) \tag{3.7}$$

*fournit une bonne approximation du paramètre de troncature* $L$.

*Pour des valeurs de* $\kappa d$ *et* $\epsilon$ *vérifiant*

$$\kappa d \in [1, 500], \quad \epsilon \in [10^{-10}, 10^{-1}]$$

*l'approximation* $L_{SC}$ *vérifie* $|L - L_{SC}| \leq 2$.

**Argumentation** *L'idée derrière cette estimation est de considérer que l'erreur commise sur (3.3) est de l'ordre de* $(2L+3)\, j_{L+1}(\kappa d)$ *si l'on tronque la série à l'ordre* $L$. *Suivant l'approximation [1]*

$$j_{L+1}(x) \approx \frac{1}{2\sqrt{f(L,x)x}} e^{f(L,x) - (L+3/2)\log([L+3/2+f(L,x)]/x)} \tag{3.8}$$

*avec* $f(L,x) = [(L+3/2)^2 - x^2]^{1/2}$ *appliquée pour* $x = \kappa d$, *et quelques estimations de décroissance rapide de* $j_l(x)$ *dès que* $l$ *dépasse* $x$ *(théorème 9), Song & Chew obtiennent l'estimation*

$$L \approx \kappa d + \frac{(3\log(1/\epsilon))^{2/3}}{2} (\kappa d)^{1/3}.$$

*soit approximativement*

$$L \approx \kappa d + 1.8 d_0^{2/3} (\kappa d)^{1/3}.$$

♢

Nous nous autoriserons par ailleurs à ré-écrire cette approximation sous la forme

$$L \approx \kappa d + d_e^{2/3} (\kappa d)^{1/3} \text{ avec } d_e = \log(1/\epsilon) \tag{3.9}$$

qui correspond à l'utilisation d'un logarithme népérien au lieu d'un logarithme en base 10, avec une très légère dégradation de la qualité de l'approximation.

Par ailleurs les travaux de [11] proposent une nouvelle formule plus précise lorsque $d$ est grand et asymptotiquement très précise :

**Estimation 7** *de Q. Carayol*

$$L(\epsilon) + \frac{1}{2} \simeq \kappa d + \frac{1}{2}\left(\frac{3}{2}\right)^{\frac{2}{3}} (\kappa d)^{\frac{1}{3}} W\left(\frac{C(\alpha_0)\kappa d}{\epsilon^2}\right), \text{ pour } d \text{ grand} \tag{3.10}$$

*où $C(\alpha_0)$ est une fonction décroissante pour $\alpha_0$ et $W$ la fonction de Lambert définie par*

$$W(t)\, e^{W(t)} = t. \tag{3.11}$$

Par la suite, nous retiendrons l'estimation 6 comme base de nos futures estimations, puisque fournissant une bonne estimation du paramètre $L$ suffisant pour que $(2L+3)\, j_{L+1}(\kappa d)$ soit de l'ordre de $\epsilon$ (pour de meilleurs résultats, il est tout à fait possible d'utiliser l'estimation 7).

## 3.4 Discrétisation de l'intégrale

Au delà de la première approximation due à la troncature de la série portant sur la fonction de transfert 3.6, il convient de discrétiser l'intégrale sur la sphère $S^2$

$$\frac{e^{\imath\kappa|x_r - x_t|}}{|x_r - x_t|} \simeq \sum_k \omega_k\, e^{\imath\kappa\langle\sigma_k, r\rangle}\, T_{L,\sigma_k}(r_0)$$
$$\text{avec } r = (x_r - C_r) + (C_t - x_t) \text{ et } r_0 = C_r - C_t$$

tout en permettant de garantir un contrôle de l'erreur ainsi commise.

### 3.4.1 Discrétisation à base d'harmoniques sphériques

Sachant qu'il est possible de décomposer les fonctions de Legendre sous forme de sommes d'harmoniques sphériques (définition 1 page 40, C.12)

$$P_l(\hat{x}\cdot\hat{y}) = \frac{4\pi}{2l+1} \sum_{m=-l}^{l} \overline{Y_l^m}(\hat{x})\, Y_l^m(\hat{y}), \tag{3.12}$$

il est possible de réécrire la fonction de transfert $T_{L,\sigma}(r_0)$, pour $\sigma \in S^2$, à partir d'harmoniques sphériques

$$T_{L,\sigma}(r_0) = 4\pi \sum_{l=0}^{p} \sum_{m=-l}^{l} \imath^l h_l^{(1)}(\kappa|r_0|)\overline{Y_l^m}(\hat{r_0})\, Y_l^m(\sigma). \tag{3.13}$$

De même pour le facteur $e^{\imath\kappa\langle\sigma,r\rangle}$, encore grâce au théorème de Gegenbauer, sous la forme affaiblie de la formule de Jacobi-Anger (théorème 10 page 40), nous obtenons

$$e^{\imath\kappa\langle\sigma,r\rangle} = 4\pi \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \imath^l j_l(\kappa|r|)\overline{Y_l^m}(\hat{r})\, Y_l^m(\sigma). \tag{3.14}$$

Ces complexes transformations nous permettent d'appréhender le comportement du produit des fonctions $T_{L,\sigma}(r_0)$ et $e^{\imath\kappa\langle\sigma,r\rangle}$. Par l'orthogonalité des harmoniques sphériques, seules les harmoniques sphériques d'ordre inférieur ou égal à $L$ sont à considérer dans l'expression (3.14) de $e^{\imath\kappa\langle\sigma,r\rangle}$. L'approximation ainsi commise est par ailleurs motivée par la décroissance

très rapide des fonctions de Bessel sphériques $j_l$ (théorème 9 page 40). Ces hypothèses sont défendues dans [11] tant numériquement qu'asymptotiquement, ce qui nous permet de tronquer les séries (3.6) et (3.14) suivant le même paramètre de troncature $L$.

Ainsi, les fonctions $T_{L,\sigma}(r_0)$ et $e^{\imath\kappa\langle\sigma,r\rangle}$ apparaissent toutes deux avec une largeur de bande $L$ dont leur produit donne une fonction de largeur de bande $2L$ dans l'espace des harmoniques sphériques, équivalente à

$$\sum_{l=0}^{2L}\sum_{m=-l}^{l}\lambda_l^m Y_l^m(\sigma).$$

La discrétisation de l'intégrale (3.5) peut alors être réduite par une quadrature à $2L+1$ points équidistribués sur $[0:2\pi]$ (longitude $\phi$) et $L+1$ points de Gauss-Legendre sur $[0:\pi]$ (latitude $\theta$) (proposition 11 page 41).

#### Opérateurs d'interpolation

Comme cela avait été évoqué dans le chapitre 2, lors de l'utilisation d'une méthode multiniveau avec une discrétisation différente pour chaque niveau (représentant la variation des largeurs de bande des fonctions utilisées), il convient de fournir des opérateurs d'interpolation permettant les calculs entre niveaux différents, permettant ainsi le changement de représentation (discrétisation).

Notons tout d'abord que l'usage des opérateurs d'interpolation sera d'autant plus important que le nombre de niveaux est grand ; leurs coûts se ressent aux alentours de 8 niveaux (dans notre cadre d'application, chaque cellule ayant en moyenne 4 fils).

Ce point important a donc largement été développé, je vous propose ici un extrait des principaux travaux disponibles.

- Interpolation de Lagrange, fournissant une analyse simple de l'erreur [51].
- Schéma semi-naïf [2], composé d'une FFT directe suivant $\phi$, d'un produit par une matrice pleine suivant $\theta$, et d'une FFT inverse encore suivant $\phi$. C'est un algorithme exact et de complexité $O(S^{1.5})$, où $S$ désigne le nombre de points de discrétisation sur $S^2$.
- Autre schéma de Alpert-Jakob-Chien [2] suivant la même approche que précédemment, mais où la matrice pleine est *compressée* grâce à une 1D-FMM (FMM monodimensionnelle). De plus, N. Yarvin et V. Rokhlin proposent une amélioration de la 1D-FMM applicable à cet opérateur d'interpolation [78, 79, 27]. C'est un algorithme approché, de complexité $O(S\log S)$).
- É. Darve [21] propose un creusement du schéma utilisé dans l'algorithme MLFMA de W. C. Chew, C.-C. Lu, J. Song. Cet algorithme est de complexité $O(S)$.
- Quelques autres algorithmes sont aussi disponibles [8, 9, 49, 50].

### 3.4.2 Autres techniques

La précédente discrétisation est largement diffusée et remplace avantageusement la technique, certes plus simple, consistant à utiliser une discrétisation uniforme à $2L+1$ points suivant les deux directions (longitude et latitude). Les coûts supplémentaires engendrés par une discrétisation approximativement deux fois plus grande sont énormes tant en calcul qu'en mémoire requise, et ne contrebalancent pas la simplicité de l'algorithme qui présentera l'avantage de n'utiliser que de simples transformées de Fourier rapides (F*ast* F*ourier* T*ransforms*) lors des étapes d'interpolation (voir section 3.4.1).

Toutefois l'utilisation d'opérateurs d'interpolation de type FFT présente l'énorme avantage de l'efficacité. C'est pourquoi sur la base d'une idée développée dans le cadre de la formulation stable à toutes fréquences (chapitre 4 page 35), il est possible d'obtenir une quadrature à la fois de taille équivalente à celle vue dans la section 3.4.1, et offrant la possibilité d'utiliser la FFT comme opérateur d'interpolation. Ce point sera détaillé dans la section 4.2.1 page 44.

# Chapitre 4

# Formulation Toutes Fréquences Stable Plane Wave FMM

**Sommaire**



Tbien des égards la méthode HF-FMM (Chapitre 3) propose une méthode très performante permettant d'atteindre des problèmes de très grandes tailles (jusqu'à quelques dizaines de millions d'inconnues) relativement aux méthodes directes techniquement limitées à des problèmes de fréquences moindres.

Toutefois, des études ont déjà été menées afin de permettre à des méthodes multipôles de traiter des problèmes de basses fréquences. Ces formulations proposent des solutions aux défauts pouvant apparaître dans la HF-FMM en particulier, au niveau de la stabilité en basse fréquence, de la précision, ou de la contrainte sur le nombre de niveau. Nous allons, dans ce chapitre, proposer une nouvelle formulation dont l'objectif proposant les mêmes solutions mais dont la complexité algorithmique tente de se rapprocher de celle de la HF-FMM.

## 4.1 Autres formulations basses fréquences

Comme précédemment, la variante de la méthode multipôle rapide que nous allons présenter réalise des produits de type matrice-vecteur de grande taille autour du même noyau de Green donnant ainsi la matrice $M$ définie par :

$$M_{ij} = \frac{\exp \imath\kappa|r_i - r_j|}{|r_i - r_j|}. \tag{4.1}$$

La méthode multipôle de Rokhlin-Greengard [62, 28, 29] est construite sur l'expansion

multipolaire suivante[1] :

$$h_0^{(1)}(r+r') = \sum_{n=0}^{+\infty} (-1)^n (2n+1)\, h_n^{(1)}(\kappa|r|)\, j_n(\kappa|r'|)\, P_n(r \cdot r') \tag{4.2}$$

où $h_n^{(1)}$ est la fonction sphérique de Hankel, $j_n$ la fonction sphérique de Bessel, et $P_n$ le polynôme de Legendre d'ordre $n$.

Nous allons nous attacher à rappeler les formules-clefs, de plus amples détails étant disponibles dans [26, 24, 25, 29].

Cette formulation est construite autour de deux solutions fondamentales de l'équation de Helmholtz l'une régulière, l'autre singulière en $r = 0$ :

$$I_n^m(r) = \imath^n\, j_n(\kappa|r|)\, L_n^m(\theta,\phi),$$
$$O_n^m(r) = \imath^n\, h_n^{(1)}(\kappa|r|)\, N_n^m(\theta,\phi),$$

où $L_n^m$ et $N_n^m$ correspondent à des harmoniques sphériques à une constante de normalisation près :

$$L_n^m(\theta,\phi) = \frac{\sqrt{2n+1}}{(n+|m|)!}\, P_n^m(\cos\theta)\, e^{-\imath m\phi}$$
$$N_n^m(\theta,\phi) = \sqrt{2n+1}\,(n-|m|)!\, P_n^m(\cos\theta)\, e^{\imath m\phi},$$

où $P_n^m$ sont les fonctions de Legendre associées (Annexe C).

Ainsi la formule 4.2 peut être simplifiée :

$$h_0^{(1)}(r+r') = \sum_{n=0}^{+\infty} \sum_{m=-n}^{n} I_n^m(r)\, O_n^m(r'). \tag{4.3}$$

Les transformations classiques Outer-to-Outer, Inner-to-Inner, Inner-to-Outer (aussi nommées Local-to-local, Multipole-to-Multipole, Multipole-to-Local) sont définies par l'intermédiaire de la fonction $E$, symbole 3-j de Wigner [29] :

$$E\begin{pmatrix} m & p & r \\ n & q & s \end{pmatrix} = \frac{1}{4\pi} \int\int N_n^m(\theta,\phi)\, N_q^p(\theta,\phi)\, L_s^r(\theta,\phi)\, \sin\theta\, d\theta d\phi. \tag{4.4}$$

La transformation Inner-to-Outer devient :

$$O_n^m(r+r') = \sum_{m,n,m',n',s} E\begin{pmatrix} m & m' & m+m' \\ n & n' & n+n'-2s \end{pmatrix} I_{n'}^{m'}(r)\, O_{n+n'-2s}^{m+m'}(r'). \tag{4.5}$$

Des équations similaires donnent les transformations Outer-to-Outer et Inner-to-Inner.

Cette approche présente deux inconvénients que nous nous proposons de compenser.

Tout d'abord, cette technique n'est pas applicable en régime haute fréquence. On peut montrer que le nombre de termes nécessaire dans l'expansion multipolaire (4.2) est de l'ordre de $\kappa D$ où $D$ est le diamètre de l'objet. Ainsi, quand $\kappa D$ devient très grand, cette méthode

[1] très proche du théorème de Gegenbauer (théorème 2)

devient coûteuse. Il n'est alors pas rare que plusieurs centaines de termes soient nécessaires pour observer une convergence de (4.2). Ce qui n'est absolument pas raisonnable en pratique.

Par ailleurs, le coût de cette méthode en régime basse fréquence est encore significatif. En effet, le calcul des fonctions $E$, ainsi que ses multiplications par $O^{m+m'}_{n+n'-2s}(r')$, requiert un coût de l'ordre de $O(p^5)$ opérations flottantes (flops), pour une expansion d'ordre $p$.

De plus, en très basses fréquences une re-normalisation des coefficients est requise, cette procédure donne alors en l'algorithme LF-MLFMA [81].

## 4.2 Nouvelle formulation

Dans cette section, nous nous proposons d'étudier une nouvelle formulation *«expansion stabilisée en ondes planes»* (Stable Plane Wave expansion) qui se montrera stable à toutes fréquences et dont le coût des transferts sera réduit à $O(p^2)$ flops comme pour la formulation haute fréquence *«expansion en ondes planes»* (plane wave expansion) précédemment rencontrée.

Afin d'éviter les problèmes de stabilité de la formulation en ondes planes, nous allons tout d'abord suivre [33].

**Théorème 8** *(L. Greengard* et al.*)*

$$\frac{e^{\imath\kappa|r|}}{|r|} = \frac{\imath\kappa}{2\pi}\int_{S^{z+}} e^{\imath\kappa\langle\sigma,r\rangle}\ \mathrm{d}\sigma + \frac{1}{2\pi}\int_{\phi=0}^{2\pi}\int_{\chi=0}^{+\infty} e^{-\chi z}\ e^{\imath\sqrt{\chi^2+\kappa^2}(x\cos\phi+y\sin\phi)}\ \mathrm{d}\chi\ \mathrm{d}\phi \qquad (4.6)$$

*où* $S^{z+} = \{r = (x, y, z) \in \mathbb{R}^3, |r| = 1, z > 0\}$ *(hémisphère supérieur de la sphère unité).*

**Argumentation** *En reprenant la description de cette formule par L. Greengard* et al.*, nous pouvons dire que (4.6) s'obtient en appliquant une transformée de Fourier (3D) sur le noyau* $\frac{e^{\imath\kappa r}}{r}$ *suivie d'une intégration du contour tout en privilégiant la direction* $+z$*. Toutefois cela nécessite alors de prendre* $z > 0$*, pour que cette intégrale de contour soit bien définie. Nous obtenons ainsi :*

$$\frac{e^{\imath\kappa|r|}}{|r|} = \frac{1}{2\pi}\int_0^{\infty} e^{-\sqrt{\lambda^2-\kappa^2}\,z}\int_0^{2\pi} e^{\imath\lambda(x\cos\phi+y\sin\phi)}\frac{\lambda}{\sqrt{\lambda^2-\kappa^2}}\,\mathrm{d}\phi\,\mathrm{d}\lambda.$$

*Dès lors, nous pouvons noter que lorsque* $0 \le \lambda \le \kappa$*, les modes se propagent sans atténuation mais décroissent lorsque* $\kappa \le \lambda \le \infty$*. Il est ainsi possible d'écrire*

$$\frac{e^{\imath\kappa|r|}}{|r|} = \left(\frac{e^{\imath\kappa|r|}}{|r|}\right)_{prop} + \left(\frac{e^{\imath\kappa|r|}}{|r|}\right)_{eva}$$

*où les parties prop et eva correspondent respectivement aux parties* propagative *et* évanescente *du noyau.*

*Le terme propagatif est défini comme suit :*

$$\begin{aligned}\left(\frac{e^{\imath\kappa|r|}}{|r|}\right)_{prop} &= \frac{1}{2\pi}\int_0^{\kappa} e^{-\sqrt{\lambda^2-\kappa^2}\,z}\int_0^{2\pi} e^{\imath\lambda(x\cos\phi+y\sin\phi)}\frac{\lambda}{\sqrt{\lambda^2-\kappa^2}}\,\mathrm{d}\phi\,\mathrm{d}\lambda\\ &= \frac{\kappa}{2\imath\pi}\int_0^{\frac{\pi}{2}} e^{\imath\kappa\cos\theta\,z}\int_0^{2\pi} e^{\imath\kappa\sin\theta(x\,\cos\phi+y\sin\phi)}\,\mathrm{d}\phi\,\sin\theta\,\mathrm{d}\theta\\ &= \frac{\imath\kappa}{2\pi}\int_{S^{z+}} e^{\imath\kappa\langle\sigma,r\rangle}\ \mathrm{d}\sigma\end{aligned}$$

*par changements de variables de la forme* $\lambda = \kappa \sin\theta$ *et* $\sigma(\theta,\phi) = (\sin\theta\cos\phi, \sin\theta\sin\phi, \cos\theta) \in S^2$.

*Le terme évanescent se transforme de la manière suivante :*

$$\begin{aligned}\left(\frac{e^{\imath\kappa|r|}}{|r|}\right)_{eva} &= \frac{1}{2\pi}\int_\kappa^\infty e^{-\sqrt{\lambda^2-\kappa^2}\,z}\int_0^{2\pi} e^{\imath\lambda(x\cos\phi+y\sin\phi)}\frac{\lambda}{\sqrt{\lambda^2-\kappa^2}}\,\mathrm{d}\phi\,\mathrm{d}\lambda \\ &= \frac{1}{2\pi}\int_0^\infty e^{-\sigma z}\int_0^{2\pi} e^{\imath\sqrt{\sigma^2+\kappa^2}(x\cos\phi+y\sin\phi)}\,\mathrm{d}\phi\,\mathrm{d}\sigma \\ &= \frac{1}{2\pi}\int_0^\infty e^{-\sigma z}J_0(\sqrt{\sigma^2+\kappa^2}\sqrt{x^2+y^2})\,\mathrm{d}\sigma\end{aligned}$$

*par le changement de variable* $\sigma^2 = \lambda^2 - \kappa^2$. ♢

En première approximation, nous pouvons déjà constater que cette formulation ne requiert plus l'utilisation de fonctions numériquement instables comme pouvait l'être les fonctions $h_m^{(1)}$ de la formulation HF-FMM classique (théorème 4), ni de sommation infinie imposant dès lors une délicate troncature. Ce point positif est directement contrebalancé par la nécessité de traiter deux termes distincts et de manière directionnelle. Nous verrons que l'approche directionnelle ne sera pénalisante que pour le terme évanescent.

Cependant, bien que notre approche utilise cette même formulation, elle sera, comme nous allons le voir, différente. Tout d'abord, comparée à [33], cette nouvelle approche a le mérite d'être applicable à toutes les fréquences alors que la précédente approche était très coûteuse en régime haute fréquence. De plus, plutôt que d'utiliser une combinaison d'expansion en ondes planes et d'expansion multipolaire classique, nous n'utiliserons qu'une expansion stabilisée en ondes planes.

Hu *et al.* [46, 42, 44, 45, 43, 40, 41] ont aussi proposé une technique (FIPWA), basée sur une intégrale similaire à l'équation (4.6). Cependant, FIPWA et l'équation (4.6) diffèrent sur le chemin de l'intégrale dans le plan complexe ; FIPWA utilise un chemin *steepest descent path*, tandis que 4.6 utilise un chemin parallèle à l'axe réel ou imaginaire [41].

**Remarque** *Comme nous l'avons vu dans la formulation HF-FMM (cf Conditions 1 et 2), la structure hiérarchique de la FMM peut être soumise à des contraintes en particulier au niveau des transferts admissibles. Dans le cas de la HF-FMM, ces conditions imposent que les cellules proches (au sens vu dans le chapitre 2) doivent être traitées sans utiliser l'approximation HF-FMM (théorème 4) mais directement.*

*Nous retrouverons de telles contraintes sur le terme évanescent, mais nous pouvons aussi noter que le terme propagatif ne sera pas soumis à de telles contraintes, ceci permettant (en théorie) de traiter les cellules proches (au sens précédent) par l'opérateur de transfert propagatif. Il parait ainsi possible de réduire le nombre de transferts de 189* $(= 6^3 - 3^3)$ *à 8. Toutefois, aucun développement présenté ici n'exploite encore ce point intéressant mais complexe à intégrer au sein de l'algorithme FMM du fait qu'il conviendrait alors de traiter indépendamment les interactions proches du terme évanescent.*

La construction d'une méthode multipôle autour de cette équation requiert la construction de différentes étapes (*cf* Chapitre 2) s'appuyant sur une forme discrète de (4.6). Notons $\omega_k^\sigma, \sigma_k$ et $\omega_k^{\chi\phi}, \chi_k, \phi_k$ les poids et points de quadrature (sur lesquels nous reviendrons par la suite).

Soit un couple de vecteurs $r$ et $r' = (x', y', z')$, nous utilisons l'approximation suivante :

$$\frac{e^{\imath\kappa|r+r'|}}{|r+r'|} \approx \frac{\imath\kappa}{2\pi} \sum_k \omega_k^{\sigma}(r)\, e^{\imath\kappa\langle\sigma_k, r'\rangle} + \frac{1}{2\pi} \sum_k e^{-\chi_k z'}\, e^{\imath\sqrt{\chi_k^2+\kappa^2}(x'\cos\phi_k + y'\sin\phi_k)}. \tag{4.7}$$

Reprenons l'algorithme FMM et une explication du rôle des paramètres $r$ et $r'$. Soient deux points $r_i$ et $r_j$ localisés dans des paquets $C_a$ et $C_b$ de centres respectifs $O_a$ and $O_b$. Alors, nous avons :

$$\begin{aligned} r &= O_a - O_b, \\ r' &= (r_i - O_a) - (r_j - O_b). \end{aligned}$$

Par ailleurs, dans le cas d'une décomposition en octree, nous avons :

$$|r| > \frac{2}{\sqrt{3}}|r'|.$$

Afin que cette expansion soit utilisable, il convient de satisfaire la condition suivante :

**Condition 3** *Il existe une constante $C$ telle que :*
*Pour tout couple $(r, r')$ de l'équation (4.7), nous avons :*

$$z > 0 \quad et \quad \sqrt{x^2+y^2} < C\, z$$

*avec $r + r' = (x, y, z)$.*

Lorsque cette condition est satisfaite, la seconde intégrale devient bornée et le nombre de points de discrétisation nécessaires en $\phi$, $n_\phi$, reste contrôlé. Notons que $C$ est associé au plus grand $n_\phi$.

Toutefois, dans le cas d'une décomposition en octree, pour deux paquets en interaction (dans le cas usuel 2.3 et dans le cas spécifique SPW-FMM 5.3.1), il est toujours possible de choisir $x$, $y$ et $z$ tels que :

$$z > 0 \quad \text{et} \quad \sqrt{x^2+y^2} < 3\sqrt{2}\, z$$

Cette condition est ainsi toujours naturellement satisfaite.

La construction d'un algorithme FMM repose sur la construction d'efficaces opérateurs d'agrégation des paquets enfants vers leurs parents et de dispersion des paquets parents vers leurs enfants. Ce type d'opérateurs a déjà été rencontré dans le cas de la formulation haute fréquence (HF-FMM) [22, 23] [15, 16, 70, 14].

En première approche, il est admissible de penser que l'étape d'agrégation revient à un accroissement du nombre de points de discrétisation, afin de représenter une plus grande largeur de bande dans l'espace de Fourier. Et inversement, l'étape de dispersion conduit à une réduction du nombre de points de discrétisation, due à une réduction de la largeur de bande.

Nous allons à présent entrer plus avant dans les détails par une analyse des différents opérateurs entrant en jeu dans notre approche. Tout d'abord l'équation 4.2 est composée de deux termes que nous nommerons respectivement *terme propagatif* et *terme évanescent*. Notons par ailleurs que ces deux termes ne sont pas si indépendants, car le choix d'une orientation de l'axe $z > 0$ influe sur le domaine d'intégration du terme propagatif et sur la convergence du terme évanescent. Nous reviendrons sur ce point.

### 4.2.1 Terme propagatif

Le calcul de cette première intégrale ressemble fort à celui de la méthode HF-FMM classique. Cependant, la discontinuité en $z = 0$ dans le domaine d'intégration induit une difficulté supplémentaire : le spectre dans le domaine de Fourier de l'intégrant décroît lentement.

Différentes stratégies sont alors applicables. L'idée en arrière plan sera toujours de supprimer les hautes fréquences (dans un espace à définir) de la fonction de transfert

$$T_r(\sigma) = \mathbf{1}_{S^{z+}}(\sigma)\, e^{\imath\kappa\langle\sigma,r\rangle} \quad \text{pour } \sigma \in S^2,\, r \in \mathbb{R}^3 \tag{4.8}$$

de telle manière que celles-ci ne contribuent pas à l'intégrale :

$$\int_{S^{z+}} e^{\imath\kappa\langle\sigma,r+r'\rangle}\, \mathrm{d}\sigma \overset{\text{def}}{=} \int_0^{2\pi}\int_0^{\frac{\pi}{2}} e^{\imath\kappa\langle\sigma(\theta,\psi),r+r'\rangle}\, \sin\theta\ \mathrm{d}\theta\ \mathrm{d}\psi.$$

Il est en effet important d'effectuer une troncature afin de réduire au minimum la taille de la quadrature nécessaire.

Au préalable, nous allons rappeler deux résultats classiques.

**Théorème 9** *Estimation des fonctions de Bessel sphériques de première espèce $j_l$*

$$|j_l(v)| \leq \sqrt{\frac{e}{2}}\frac{(e|v|)^l}{(2l+1)^{l+1}} \tag{4.9}$$

*pour tout paramètre $l \geq 0$ et tout argument $v > 0$.*

**Théorème 10** *Séries de Jacobi-Anger ; Théorème de Gegenbauer*

$$e^{\imath\kappa\langle\sigma,x\rangle} = \sum_{p=0}^{\infty}(2p+1)\,\imath^p j_p(\kappa|x|)P_p(\sigma\cdot\hat{x}) \tag{4.10}$$

*pour tout $\sigma = (\theta,\psi) \in S^2$, $x \in \mathbb{R}^3$, avec $P_p$, polynômes de Legendre (Définition C.1).*

#### Transformation en Harmoniques Sphériques

La transformation en Harmoniques Sphériques est notre première proposition aux difficultés rencontrées sur l'intégration du terme propagatif. Celle-ci consistera en un lissage de l'intégrant $T_r$ dans l'espace des fonctions harmoniques sphériques.

**Définition 1** *L'espace des fonctions harmoniques sphériques est l'espace engendré par les fonctions $\{Y_{lm}\}_{l\in\mathbb{N},|m|\leq l}$ définies par*

$$\forall l \geq 0, \forall m \in \{0,\ldots,l\} \quad Y_{lm}(\sigma) = \sqrt{\frac{2l+1}{4\pi}}\, C_l^m P_l^m(\cos\theta)\, e^{\imath m\psi} \tag{4.11}$$

*et*

$$\forall l \geq 0, \forall m \in \{0,\ldots,l\} \quad Y_{l,-m} = (-1)^m Y_{l,m}^* \tag{4.12}$$

*où $C_l^m = \sqrt{\frac{(l-m)!}{(l+m)!}}$ et $P_l^m$ sont les fonctions de Legendre associées (Définition C.13).*

*$\{Y_{lm}\}_{l\in\mathbb{N},|m|\leq l}$ constitue une base orthonormée de $L^2(S^2)$. Toute fonction $f$ de $L^2(S^2)$ peut ainsi être décomposée sous la forme :*

$$f = \sum_{l\in\mathbb{N}} \sum_{|m|\leq l} \mathcal{H}(f)_{lm} Y_{lm} \tag{4.13}$$

*avec*

$$\mathcal{H}(f)_{lm} = \int_{S^2} f(\sigma) Y^*_{lm}(\sigma)\, \mathrm{d}\sigma. \tag{4.14}$$

Une quadrature des fonctions $Y_{lm}$ est donnée par le résultat classique suivant :

**Proposition 11** *Toute fonction de* $\mathrm{Span}(\{Y_{lm}\}_{0\leq l\leq 2L,|m|\leq l})$ *est exactement intégrée avec une quadrature de Gauss-Legendre à $\frac{2L+1}{2}$ points suivant $\theta$ et une quadrature uniforme à $2L+1$ points suivant $\psi$.*

**Preuve** *Intégrer exactement toutes fonctions $Y_{lm}(\sigma) = \sqrt{\frac{2l+1}{4\pi}}\, C_l^m P_l^m(\cos\theta)\, e^{\imath m\psi}$, pour $L \geq 0$ et $0 \leq |m| < 2L$, $Y_{lm}$ revient à trouver une quadrature permettant de calculer exactement*

$$\int_{S^2} P_l^m(\cos\theta) e^{\imath m\psi}\, \sin\theta\, \mathrm{d}\theta\, \mathrm{d}\psi = \int_0^{2\pi} e^{\imath m\psi}\, \mathrm{d}\psi \int_0^{\pi} P_l^m(\cos\theta)\sin\theta\, \mathrm{d}\theta$$

*par le théorème de Fubini.*

*Les fonctions $e^{\imath m\psi}$ s'intègrent exactement sur $[0, 2\pi]$ par une quadrature uniforme $\psi_i$ à $2L+1$ points et de poids $\frac{2\pi}{2L+1}$. Et, dès que $m \neq 0$, nous avons $\sum_i e^{\imath m\psi_i} = 0$. Ainsi, quel que soit le choix de la quadrature $\theta_j$, l'intégration est exacte pour $m \neq 0$. Il suffit alors de trouver une quadrature pour*

$$\int_0^{\pi} P_l(\cos\theta)\sin\theta\, \mathrm{d}\theta = \int_{-1}^{1} P_l(x)\, \mathrm{d}x$$

*qui peut être obtenue par une quadrature de Gauss-Legendre à $\frac{2L+1}{2}$ points, du fait que les $P_l$ sont des polynômes d'ordre $l$ et que $n$ points de Gauss-Legendre permettent d'intégrer tout polynôme d'ordre inférieur ou égal à $2n-1$.* □

Tout d'abord, présentons un théorème largement utilisé dans la HF-FMM sur l'approximation nécessaire du terme $e^{\imath\kappa\langle\sigma,r\rangle}$, auquel ressemble fort l'intégrant du terme propagatif.

**Estimation 12** *Soit la fonction $f_r$ définie par $f_r(\sigma) \overset{def}{=} e^{\imath\kappa\langle\sigma,r\rangle}$. Pour tout $\epsilon > 0$, nous avons :*

$$\left| f_r(\sigma) - \sum_{0\leq|m|\leq l\leq L} \mathcal{H}(f)_{lm} Y_{lm}(\sigma) \right| \leq \epsilon$$

*pour $L \gtrsim \kappa|r| + 1.8\, d_0^{2/3} (\kappa|r|)^{1/3}$ et $d_0 \overset{def}{=} \log_{10}(1/\epsilon)$.*

*La qualité de cette estimation dépend directement de celle de l'estimation 6 de Song & Chew.*

**Argumentation** *L'argument principal de cette estimation tient tout d'abord en une réécriture de $f_r$ en terme de série de Jacobi-Anger (théorème 10) et de sa décomposition harmonique. Il est dès lors possible de retrouver un contexte d'application de l'estimation 6 de Song & Chew basée autour d'une estimation du paramètre de troncature $L$ tel que $(2L+3) j_{L+1}(\kappa|r|) \approx \epsilon$.* ♢

Maintenant prolongeons ce théorème afin d'obtenir une discrétisation optimale du terme propagatif.

**Estimation 13** *Soit $T_r$ la fonction de transfert définie par*

$$T_r(\sigma) \stackrel{def}{=} \mathbf{1}_{S^{z+}}(\sigma)\, e^{\imath\kappa\langle\sigma,r\rangle} \qquad \textit{pour } \sigma \in S^2,\ r \in \mathbb{R}^3,$$

*Soient $\epsilon > 0$ l'erreur considérée sur l'approximation de l'intégrant du terme propagatif et $d$ le diamètre des cellules au niveau où $T_r$ opère.*

*Soit $\widetilde{T}_r^L$ la projection de la fonction de transfert $T_r$ dans une base d'harmoniques sphériques $\{Y_{lm}\}_{0\leq|m|\leq l\leq L}$*

$$\widetilde{T}_r^L(\sigma) = \sum_{0\leq|m|\leq l\leq L} \mathcal{H}(T_r)_{lm}\, Y_{lm}(\sigma)$$

*avec $L \approx \kappa d + 1.8\, d_0^{2/3}(\kappa d)^{1/3}$, $d_0 = \log_{10}(1/\epsilon)$.*

*L'utilisation de $\widetilde{T}_r^L$ en tant que fonction de transfert permet une évaluation du terme propagatif grâce à une quadrature à $L+1$ points de Gauss-Legendre suivant $\theta \in [0,\pi]$ et $L+1$ points uniformément répartis suivant $\phi \in [0,\pi]$.*

**Argumentation** *D'après l'estimation 12, $e^{\imath\kappa\langle\sigma,r'\rangle}$ peut être approché à $\epsilon$ près dans une base d'harmoniques sphériques d'ordre $L$ pour $|r'| \leq d$.*

*En réutilisant un argumentaire déjà employé dans la section 3.4.1, nous pouvons dire que toute projection de $T_r$ dans une base d'harmoniques sphériques d'ordre supérieur à $L$ est inutile (par orthogonalité des fonctions de la base). Dès lors le produit $e^{\imath\kappa\langle\sigma,r'\rangle}\,\widetilde{T}_r^L$ est d'ordre $2L$ dans l'espace des harmoniques sphériques.*

*Par la proposition 11, une quadrature à $(2L+1)\times(L+1)$ points suffit pour intégrer exactement $e^{\imath\kappa\langle\sigma,r'\rangle}\,\widetilde{T}_r^L$.*

*Par ailleurs, une réduction du nombre de points à $(L+1)\times(L+1)$ s'obtient par la propriété de symétrie suivante :*

$$e^{\imath\kappa\langle\sigma(\theta,\psi,r\rangle} = e^{\imath\kappa\langle\sigma(2\pi-\theta,\pi+\psi),r\rangle}$$

♢

**Remarque** *Une construction effective de $\widetilde{T}_r^L$ est donnée en Annexe D.*

Il faut tout d'abord noter que cette première approche permet une efficace réutilisation des algorithmes classiques proposés dans le cadre HF-FMM (chapitre 3) au niveau des opérateurs d'inter/anter-polation (étapes d'agrégation et de dispersion) Ainsi, nous utiliserons, comme précédemment, un schéma semi-naïf [2] comprenant un passage dans l'espace de Fourier via une FFT sur la composante $\psi$ (sur/sous-échantillonnages) et un produit par une matrice pleine en $\theta$.

Il convient à présent d'illustrer cette méthode pour un calcul rapide et précis du terme propagatif.

Considérons $N$ points $x_i$ de potentiels respectifs $q_i$ dans une cellule de taille $\rho$. Notre illustration portera sur une comparaison de la vitesse de décroissance des erreurs d'intégration $E_L$ et $\widetilde{E}_L$ définies par

$$E_L(\{(x_i,q_i)\}) = \left| \sum_i q_i \int_{S^2} T_r(\sigma)\, e^{\imath\kappa\langle\sigma,x_i\rangle}\, \mathrm{d}\sigma - \sum_i q_i \sum_{\substack{0\leq l\leq L\\ |m|\leq L}} \omega_{lm} T_r(\sigma_{lm})\, e^{\imath\kappa\langle\sigma_{lm},x_i\rangle} \right|$$

$$\widetilde{E}_L(\{(x_i,q_i)\},\kappa,\rho,\epsilon) = \left| \sum_i q_i \int_{S^2} T_r(\sigma)\, e^{\imath\kappa\langle\sigma,x_i\rangle}\, \mathrm{d}\sigma - \sum_i q_i \sum_{\substack{0\le l\le L\\ |m|\le L}} \omega_{lm} \widetilde{T}_r^{L(\kappa,\rho,\epsilon)}(\sigma_{lm})\, e^{\imath\kappa\langle\sigma_{lm},x_i\rangle} \right|.$$

où $(\omega_{lm},\sigma_{lm})_{0\le l\le L,|m|\le L}$ désigne la quadrature introduite dans la proposition 11 et $L(\kappa,\rho,\epsilon)$, le paramètre de troncature proposé dans l'estimation 13. De nouveau, nous considérons $\sigma \in S^2$ comme la paramétrisation $\sigma(\theta,\psi) = (\cos\psi\sin\theta, \sin\psi\sin\theta, \cos\theta)$.

Nous opposons ainsi l'erreur d'intégration de la fonction $T_\sigma$ à celle de son approximation $\widetilde{T}_r^{L(\kappa,\rho,\epsilon)}$, obtenue par lissage dans l'espace des Harmoniques Sphériques.

La valeur de référence est obtenue par une quadrature très fine (et donc très lente) garantissant une erreur «machine» de $10^{-14}$.

Les figures 4.1 et 4.2 montrent l'erreur d'intégration pour $N = 100$, $\rho = 1$, $r = (6,6,6)$ (long transfert), pour une précision demandée de $\epsilon = 10^{-4}$. $\kappa$ est le seul paramètre variant entre ces deux figures, y valant respectivement $\kappa = 1$ et $\kappa = 10$.

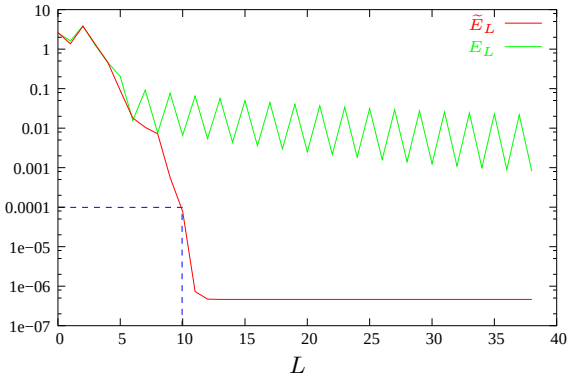



FIG. 4.1 – Convergence : $\kappa = 1$, $\epsilon = 10^{-4}$

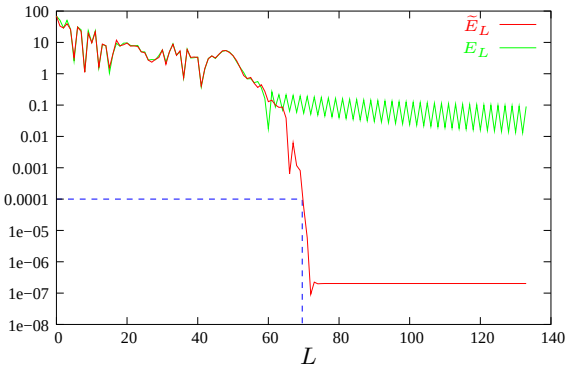



FIG. 4.2 – Convergence : $\kappa = 10$, $\epsilon = 10^{-4}$

Les observations que l'on peut en tirer sont que dans tous les cas l'erreur tend à décroître avec l'augmentation du nombre de points de quadrature ; ce qui est tout à fait normal. Mais le point intéressant est la brutale décroissance de l'erreur à l'approche de la troncature ($L(\kappa,\rho,\epsilon)$ vaut 20 lorsque $\kappa = 1$ et 125 pour $\kappa = 10$). Celle-ci décroît de manière excessive au-delà de l'erreur $\epsilon$ demandée. Cela montre un manque de précision dans notre estimation de la troncature, car comme cela a été précédemment indiqué, le théorème 9 ne nous permet que d'obtenir une borne supérieure de la troncature, donc sous-optimale, par rapport à l'erreur réelle. De meilleurs estimateurs permettraient une réduction de la quadrature, de la discrétisation et donc une économie en terme de calculs.

Pour finir, la figure 4.3 reprend la même présentation avec $\kappa = 10$ et $r = (1,2,3)$ (c'est ce vecteur plus court qui conduit à une réduction du nombre de points). Elle illustre la possibilité d'atteindre toute précision jusqu'à l'erreur machine $\epsilon = 10^{-14}$.

En conclusion, le lissage par Harmoniques Sphériques nous a ici permis la construction d'une intégration efficace, tout en permettant une réutilisation des techniques mises en œuvre pour HF-FMM.

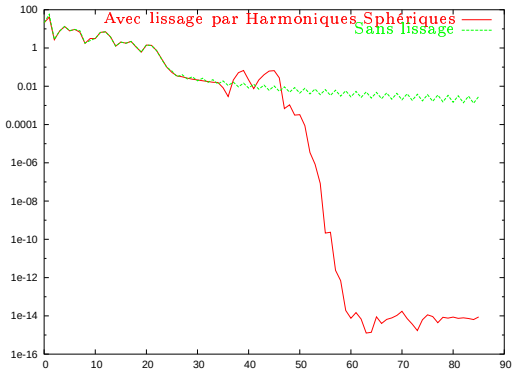



FIG. 4.3 – $\kappa = 10$, sans restriction sur $\epsilon$

**Forme alternative de $T_r$**

Cela part d'une remarque simple : les Transformées de Fourier Rapides (FFT[2]) sont un outil remarquablement performant relativement au schéma semi-naïf déjà rencontré. En effet, cela réduirait la complexité algorithmique de $O(N^2)$ (produit matrice-vecteur) à $O(N \log N)$ (FFT). Par ailleurs, les algorithmes en seraient d'autant simplifiés (utilisation de quadrature simplifiée, pas de fonctions spéciales...). Cette idée rejoint celle évoquée dans [65].

La discontinuité de $T_r$ en $z = 0$ est le principal obstacle à une utilisation massive des FFT. L'objectif de cette section est donc de fournir une forme équivalente de $T_r$ permettant cette évolution. Notons, par ailleurs, que bien qu'elle soit ici construite pour $T_r$, la démarche est tout à fait applicable à la méthode HF-FMM, c'est-à-dire pour l'intégration de $e^{\imath\kappa\langle\sigma,r\rangle}$.

Mise à part l'utilisation de FFT, ce calcul du terme propagatif sera équivalent à celui déjà réalisé grâce au lissage par Harmoniques Sphériques. En fait, le nombre de points de quadrature sera équivalent, mais aura l'avantage de comporter moins de calculs complexes de type fonctions spéciales, fort coûteux, en particulier durant les étapes de anter/inter-polation.

Nous avons :

$$\int_{S^2} T_r(\sigma)\, e^{\imath\kappa\langle\sigma,r'\rangle}\, \mathrm{d}\sigma = \int_{S^{z^+}} e^{\imath\kappa\langle\sigma,r+r'\rangle}\, \mathrm{d}\sigma = \int_0^{\pi} \mathrm{d}\theta \int_0^{2\pi} \mathrm{d}\psi\ \mathbf{1}_{[0,\pi/2]}(\theta)\ \sin\theta\ e^{\imath\kappa\langle\sigma,r\rangle}\ e^{\imath\kappa\langle\sigma,r'\rangle}. \tag{4.15}$$

La transformation suivante nous permet d'aborder une symétrie dans $T_r$ en portant l'intégration suivant $\theta$ de 0 à $2\pi$ au lieu de $\pi$ :

$$S(\theta,\psi) = (2\pi - \theta, \pi + \psi) \tag{4.16}$$

et est telle que :

$$\sigma(S(\theta,\psi)) = \sigma(\theta,\psi). \tag{4.17}$$

Nous pouvons alors écrire :

$$\int_{S^2} T_r(\sigma)\, e^{\imath\kappa\langle\sigma,r'\rangle}\, \mathrm{d}\sigma = \int_0^{2\pi} \mathrm{d}\theta \int_0^{2\pi} \mathrm{d}\psi\ F_r(\theta,\psi)\, e^{\imath\kappa\langle\sigma,r'\rangle}, \tag{4.18}$$

avec

$$F_r(\theta,\psi) \stackrel{\text{def}}{=} \frac{1}{2}\Big(\mathbf{1}_{[0,\pi/2]}(\theta)\ \sin\theta + \mathbf{1}_{[0,\pi/2]}(2\pi-\theta)\ \sin(2\pi-\theta)\Big)\ e^{\imath\kappa\langle\sigma(\theta,\psi),r\rangle}. \tag{4.19}$$

[2]Fast Fourier Transform

**Estimation 14** *Soit $F_r$ la fonction de transfert définie par*

$$F_r(\sigma(\theta,\psi)) \stackrel{def}{=} \frac{1}{2}\left(\mathbf{1}_{[0,\pi/2]}(\theta)\ \sin\theta + \mathbf{1}_{[0,\pi/2]}(2\pi-\theta)\ \sin(2\pi-\theta)\right) e^{\imath\kappa\langle\sigma(\theta,\psi),r\rangle}$$

*pour $r \in \mathbb{R}^3$ et $\sigma(\theta,\psi) = (\cos\psi\sin\theta, \sin\psi\sin\theta, \cos\theta) \in S^2$.*

*Soient $\epsilon > 0$ l'erreur considérée sur l'approximation de l'intégrant du terme propagatif et $d$ le diamètre des cellules au niveau où $F_r$ opère.*

*Soit $\widetilde{F}_r^L$ la projection de la fonction de transfert $F_r$ dans une base de Fourier 2D, aussi bien suivant $\theta$ que suivant $\psi$, composée de fréquences $l \in [-L:L]$ avec $L \approx \kappa d + 1.8\, d_0^{2/3}(\kappa d)^{1/3}$, $d_0 = \log_{10}(1/\epsilon)$.*

*L'utilisation de $\widetilde{F}_r^L$ au lieu de $F_r$ permet une approximation du terme propagatif grâce à une quadrature à $1 + L \times (L+1)$ points.*

**Argumentation** *En suivant une démarche analogue à l'estimation 13, nous obtenons une approximation de $e^{\imath\kappa\langle\sigma,r'\rangle}$ à $\epsilon$ près par une projection dans une base de Fourier 2D composée de fréquences $l \in [-L:L]$ suivant $\theta$ et $\psi$ avec $L \approx \kappa d + 1.8\, d_0^{2/3}(\kappa d)^{1/3}$, $d_0 = \log_{10}(1/\epsilon)$.*

*Dès lors une approximation de $F_r$ suffit. Ce qui conduit à l'utilisation d'une quadrature à $(2L+1)$ dans chaque direction.*

*Ce nombres peut être réduit à $L+1$ par la symétrie en $\psi = 0$ et en $\theta = \pi$. Toutefois, aux pôles de $S^2$ la discrétisation de $\psi$ peut se réduite à 1 point. Ainsi nous obtenons $1 + L \times (L+1)$ points de discrétisation.* ♢

**Remarque** *Une construction effective de $\widetilde{T}_r^L$ est donnée en Annexe E.*

Les théorèmes 10 et 9 permettent d'établir une borne supérieure de la largeur de bande de $e^{\imath\kappa\langle\sigma,r'\rangle}$ suivant $\psi$ et $\theta$.

Numériquement, une estimation du nombre de points de discrétisation suivant $\psi$ peut aussi être obtenue au regard de la largeur de bande dans l'espace de Fourier de la fonction $e^{\imath\kappa|r'|\cos\psi}$. De même suivant $\theta$, la largeur de bande de $e^{\imath\kappa|r'|\cos\theta}$ définit une estimation du nombre de points de discrétisation suivant $\theta$.

Notons que les discrétisations $(\psi_k)_k$ et $(\theta_k)_k$ sont identiques par construction. Par ailleurs, il est aussi possible de réduire le coût, à la fois en terme de stockage et de calculs d'un facteur 2, en remarquant que :

$$\begin{aligned} e^{\imath\kappa\langle\sigma(\theta,\psi),r\rangle} &= e^{\imath\kappa\langle\sigma(S(\theta,\psi)),r\rangle} \\ F_r(\theta,\psi) &= F_r(S(\theta,\psi)). \end{aligned}$$

Ainsi, seulement les angles $\theta$, $0 \le \theta \le \pi$, nécessitent d'être stockés et calculés.

### 4.2.2 Terme évanescent

Autant le terme propagatif pouvait par certains aspects ressembler à la formulation HF-FMM, autant le terme évanescent va nécessiter un traitement moins classique, qui en première approche pourra sembler complexe.

**Troncature de l'intégrale**

Considérons notre noyau $K$ ainsi que son intégrale $V$

$$K(\chi, z, \rho) \stackrel{\text{def}}{=} e^{-\chi z} e^{i\rho\sqrt{\chi^2+\kappa^2}},$$
$$V(x, y, z) \stackrel{\text{def}}{=} \int_0^{+\infty} \mathrm{d}\chi \int_0^{2\pi} \mathrm{d}\phi \; K(\chi, z, x\cos\phi + y\sin\phi),$$
$$V_0(x, y, z) \stackrel{\text{def}}{=} \frac{1}{2\pi} V(x, y, z).$$

Notons la présence d'une intégrale généralisée suivant $\chi$ dans $V$. Nous ne proposerons pas ici de méthodes permettant de la traiter sans troncature. Dès lors, nous nous limiterons à

$$\tilde{V}_{0\epsilon}(x, y, z) \stackrel{\text{def}}{=} \frac{1}{2\pi} \int_0^{\chi'_{max}(\epsilon)} \mathrm{d}\chi \int_0^{2\pi} \mathrm{d}\phi \; K(\chi, z, x\cos\phi + y\sin\phi)$$

où $\chi'_{max}(\epsilon)$ vise à satisfaire

$$\left| V_0(x, y, z) - \tilde{V}_{0\epsilon}(x, y, z) \right| \leq \epsilon$$

avec $x$, $y$, $z$ dans leur espace de définition respectif.

Il est très simple de majorer $|V_0(x, y, z) - \tilde{V}_{0\epsilon}(x, y, z)|$ :

$$\begin{aligned}
\left| V_0(x, y, z) - \tilde{V}_{0\epsilon}(x, y, z) \right| &= \left| \frac{1}{2\pi} \int_{\chi'_{max}(\epsilon)}^{\infty} \mathrm{d}\chi \int_0^{2\pi} \mathrm{d}\phi \; K(\chi, z, x\cos\phi + y\sin\phi) \right| \\
&\leq \int_{\chi'_{max}(\epsilon)}^{\infty} \mathrm{d}\chi \; \exp(-\chi z) \\
&= \frac{e^{-\chi'_{max}(\epsilon) z}}{z}
\end{aligned}$$

Dès lors, il convient de prendre

$$\chi'_{max}(\epsilon) \geq -\frac{\log(z_{min}\epsilon)}{z_{min}}$$

avec $z_{min}$ désignant la borne inférieur suivant $z$.

Toutefois en de nombreux points des sections qui suivent, nous préférerons considérer l'erreur commise sur l'approximation de $K$ qui consiste définir le plus petit $\chi_{max}(\epsilon')$ vérifiant

$$|K(\chi, z, \rho)| \leq \epsilon \text{ pour tout } \chi \geq \chi_{max}(\epsilon).$$

Dès lors, il convient de prendre

$$\chi_{max}(\epsilon) \geq -\frac{\log \epsilon}{z_{min}}.$$

Toutefois, pour $\epsilon'$ et $z_{min}$ donnés, il est possible d'adapter la précision $\epsilon$ afin que $\chi_{max}(\epsilon) = \chi'_{max}(\epsilon')$.

Dans tous les cas ces erreurs seront des erreurs relatives du fait de la présence de coefficients multiplicatifs issus de potentiels.

**Discrétisation suivant $\phi$**

Les Transformées de Fourier Rapides (FFT) permettent des interpolations et anterpolations extrêmement performantes. Cette méthode peut encore être utilisée suivant $\phi$ pour le terme évanescent.

**Estimation 15** *Soit $\epsilon > 0$ l'erreur $L^\infty$ tolérée sur le noyau $K$. Soit $R$ le côté d'une cellule au niveau auquel on souhaite calculer $V$.*

*La discrétisation suivant $\phi$ peut être obtenue par une quadrature uniforme de taille $n_\phi \approx L+1$ avec*

$$L = L_0 + d_e^{2/3} L_0^{1/3},$$

$$d_e \stackrel{def}{=} \log(1/\epsilon) \quad et \quad L_0 \stackrel{def}{=} \sqrt{2}R\sqrt{\left(\frac{2d_e}{R}\right)^2 + \kappa^2}.$$

**Argumentation** *Il est de nouveau possible de considérer ce résultat comme une conséquence de Song & Chew (estimation 6) et dans le même esprit que les estimations 13 et 14.*

*Tout d'abord, la plus grande valeur de $\chi$ est $-\log\epsilon/z_{min}$ soit $2d_e/R$ (proposition 22). Ainsi, nous obtenons la relation*

$$\sqrt{\chi^2+\kappa^2}\cdot\sqrt{x^2+y^2} \leq \sqrt{\left(\frac{2d_e}{R}\right)^2+\kappa^2}\cdot\sqrt{2}R \stackrel{def}{=} L_0$$

*qui par conséquent donne une estimation de l'ordre de troncature de la série de Jacobi-Anger sous-jacente.*

*Dès lors une quadrature uniforme de $[0, 2\pi]$ à $2L+1$ points $\phi_k$ peut être choisie.*

*Par ailleurs, il est possible de réduire cette discrétisation par 2, suivant la symétrie :*

$$e^{\imath\sqrt{\chi^2+\kappa^2}(x\cos(\pi+\phi)+y\sin(\pi+\phi))} = \left(e^{\imath\sqrt{\chi^2+\kappa^2}(x\cos\phi+y\sin\phi)}\right)^*$$

*Ce qui conduit à $n_\phi \approx L+1$.* ♢

**Corolaire 16** *Deux approximations de $n_\phi$ sont possibles pour les régimes basses et hautes fréquences :*

- *Lorsque que $R \ll 2d_e/\kappa$ (*i.e. *$z \ll 1/\kappa$ pour $\epsilon$ donné), le nombre de points de discrétisation suivant $\phi$, $n_\phi$, est indépendant de la taille de la cellule, et ne dépend que de la tolérance $\epsilon$. Dans les termes du théorème 15, cela signifie que $L_0 \approx d_e\sqrt{2}$.*
- *Lorsque que $R \gg 2d_e/\kappa$ (*i.e. *$z \gg 1/\kappa$ pour $\epsilon$ donné), le nombre de points de discrétisation suivant $\phi$, $n_\phi$ est dominé par les termes $R$ et $\kappa$. Dans les termes du théorème 15, cela signifie que $L_0 \approx \sqrt{2}\kappa R$.*

**Remarque** *De nouveau, il est possible d'estimer numériquement $n_\phi$ comme $n_\phi^{max}$, largeur de bande suivant $\phi$ de la fonction $\exp\left(\imath\sqrt{\chi_k^2+\kappa^2}\sqrt{(x')^2+(y')^2}\,\cos\phi\right)$ avec $r' = (x', y', z')$.*

**Discussion sur une discrétisation suivant $\chi$**

Les difficultés autour du terme évanescent nécessitent de porter un soin tout particulier à la variable $\chi$.

Si l'on souhaite regarder le comportement de

$$G_0^{z\rho}(\chi) = e^{-\chi z}\, e^{\imath\rho\sqrt{\chi^2+\kappa^2}},$$

dans l'espace de Fourier, il convient tout d'abord de la rendre plus régulière sur l'intervalle d'étude $[0, \chi_{max}]$, tout d'abord en la rendant périodique. Un prolongement par périodicité de la forme $G_0^{z\rho}(|\chi|)$ est possible mais singulier en 0. Ces types de singularités réduisent la vitesse de décroissance des coefficients de Fourier associés à la fonction. Nous préférerons donc utiliser le changement de variable $\chi \to x^2$ donnant

$$G^{z\rho}(\chi) = e^{-\chi^2 z}\, e^{\imath\rho\sqrt{\chi^4+\kappa^2}}.$$

Bien entendu, cela revient alors à calculer

$$\int_{-\infty}^{+\infty} e^{-\chi^2 z}\, e^{\imath\sqrt{\chi^4+\kappa^2}(x\cos\phi+y\sin\phi)}\, \chi \mathbf{1}_{\mathbb{R}_+}(\chi)\ \mathrm{d}\chi$$

ou bien

$$\frac{1}{2}\int_{-\infty}^{+\infty} e^{-\chi^2 z}\, e^{\imath\sqrt{\chi^4+\kappa^2}(x\cos\phi+y\sin\phi)}|\chi|\ \mathrm{d}\chi$$

au lieu de

$$\int_{0}^{+\infty} e^{-\chi z}\, e^{\imath\sqrt{\chi^2+\kappa^2}(x\cos\phi+y\sin\phi)}\ \mathrm{d}\chi.$$

Dans les deux cas, il conviendra de traiter le facteur introduit par le changement de variable. La méthode la plus simple sera d'éliminer les hautes fréquences hors du spectre du facteur principal, induites par ce facteur.

$G^{z\rho}$ a un spectre (dans l'espace de Fourier) similaire à celui de

$$e^{-\chi^2 z}\, e^{\imath\kappa\rho}$$

pour de grandes valeurs de $z$ et $\rho$. Nous ne pouvons fournir une preuve élégante de ceci. L'argument principal est que pour de grandes valeurs de $z$, $\chi$ se doit d'être petit sinon $e^{-\chi^2 z}$ deviendrait négligeable. Par ailleurs :

$$z \gg \frac{1}{\kappa} \ \Rightarrow\ \begin{cases} \left| e^{-\chi^2 z}\, e^{\imath\sqrt{\chi^4+\kappa^2}\rho} \right| \leq \epsilon \quad \text{ou} \\ \chi^2 \ll \kappa \end{cases}$$

À l'opposé, quand $z \ll \frac{1}{\kappa}$, $G^{z\rho}(\chi)$ a un spectre de Fourier similaire à

$$e^{-\chi^2 z}\, e^{\imath\chi\rho}.$$

Dans ces deux cas, le comportement attendu du spectre est similaire à une gaussienne. Cependant, le comportement dans la région intermédiaire, où $z \approx \frac{1}{\kappa}$, est moins clair. En l'absence d'une preuve rigoureuse, nous nous référerons à des tests numériques.

La figure 4.4 montre le spectre de Fourier de $G^{z\rho}(\chi)$ pour un panel de valeurs de $z$ avec $\rho = z$ et $\kappa = 1$. Bien évidemment, le choix de $\kappa = 1$ n'est pas restrictif vu que ceci correspond à une mise à l'échelle des axes telle que $\kappa = 1$.

Le spectre est comparé à une gaussienne. Ceci nous permet de voir clairement qu'en régime intermédiaire $z = 10^{-2}$, le spectre de Fourier décroît très lentement. Cependant en régime très basse fréquence $z = 10^{-4}$, nous observons une décroissance très rapide suivi d'un plateau aux alentours de $10^{-7}$. Ainsi relativement à $z$, le comportement est une décroissance rapide suivie d'un plateau qui, empiriquement, peut être trouvé aux alentours de $10^{-3}\ z$.

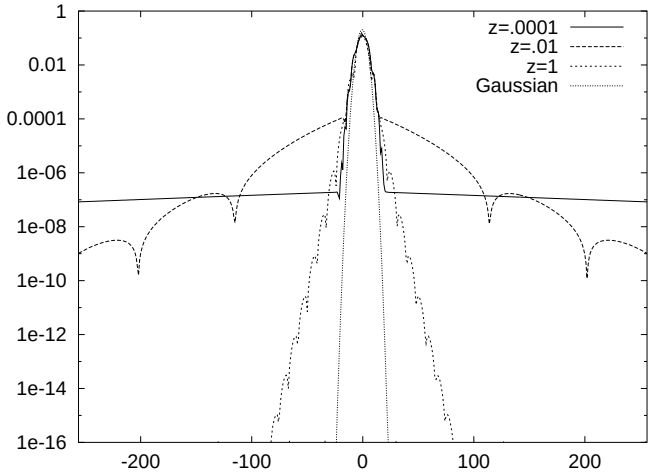



FIG. 4.4 – Spectre de Fourier de $G^{z\rho}(\chi)$ en régime basse fréquence

La figure 4.5 présente un comportement similaire mais en régime haute fréquence : le spectre est très similaire à une gaussienne et la décroissance s'amplifie avec $z$.

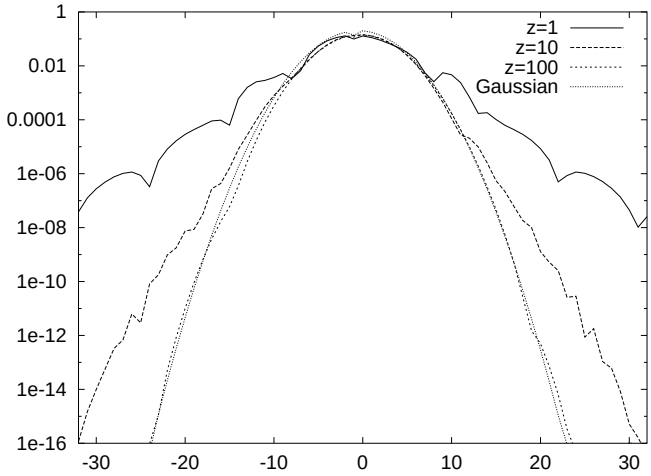



FIG. 4.5 – Spectre de Fourier de $G^{z\rho}(\chi)$ en régime haute fréquence

Ces tests confirment notre prévision, par conséquent nous en tirons la conclusion suivante :
– En régime très basse fréquence ($z \lesssim 10^{-4}/\kappa$) ainsi qu'en régime haute fréquence ($z \gtrsim$

$1/\kappa$), le spectre de Fourier décroît rapidement comme une gaussienne.
- En régime basse fréquence (région intermédiaire, $z \approx 10^{-2}/\kappa$), la décroissance est très lente.

À la vue de ces réponses, nous nous posons la question suivante : pour une erreur $\epsilon$ donnée, combien de points de quadrature uniformément distribués sont-ils nécessaires pour représenter $G^{z\rho}(\chi)$ ? Pour nous faire une idée de la réponse, nous prenons $\epsilon = 10^{-6}$ et traçons le nombre de points de quadrature nécessaires en fonction de $z$. En régime très basse fréquence ($z < \frac{10^{-4}}{\kappa}$) ainsi qu'en régime haute fréquence ($z > \frac{1}{\kappa}$), le spectre est similaire à une gaussienne et avec une décroissance très rapide. Ce résultat conduit à un nombre de points de quadrature nécessaires très réduit. En région intermédiaire ($z \approx \frac{10^{-2}}{\kappa}$), le nombre de points nécessaire pour représenter $G^{z\rho}(\chi)$ devient très grand.

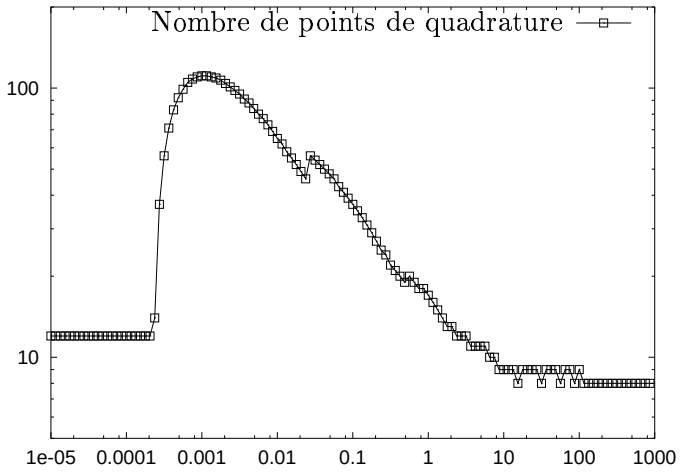



FIG. 4.6 – Nombre de points quadrature uniformément distribués en $\chi$ permettant de représenter $G^{z\rho}(\chi)$ en fonction de $z$ pour $\epsilon = 10^{-6}$.

Une implication pratique de cette observation est qu'un lissage standard et une stratégie d'interpolation basée sur les transformées de Fourier rapides (section 3, [22, 23]) ne seraient applicables sur $G^{z\rho}(\chi)$ qu'en régimes très basse fréquence et haute fréquence. Le régime intermédiaire ne permet pas cette approche, et par conséquent l'interdit en tant qu'approche unifiée construite sur $G^{z\rho}(\chi)$ allant des très basses fréquences aux hautes fréquences.

Notre première alternative sera d'envisager une quadrature non-uniforme, pour laquelle nous suivrons une idée de V. Rokhlin [77] permettant de construire des quadratures «optimales» grâce à l'utilisation de décompositions en valeurs singulières [32].

### Décomposition en valeurs singulières & quadratures généralisées

Notre but est d'obtenir une quadrature optimale (au sens du nombre de points) suivant $\chi$ de $K$. Par ailleurs, il est primordial que toute quadrature ainsi obtenue permette de construire les autres étapes de la FMM, et principalement les étapes d'agrégation et de dispersion qui correspondent à un changement d'échelle local du problème et donc à une changement de quadrature.

Cette section reprend les idées de l'article de Yarvin & Rokhlin [77] de construction de quadratures optimales par utilisation de la décomposition en valeurs singulières (S*ingular*

V*alue* D*ecomposition*). Nous appliquerons ce principe à notre noyau $K$, et nous le prolongerons dans l'optique d'une adaptation à la FMM.

**Proposition 17** *Pour tout $\varepsilon > 0$, il existe un ensemble de fonctions réelles orthonormales $\{u_p(\chi)\}_{0\leq p<N}$, un ensemble de fonctions complexes orthonormales $\{v_p(z,\rho)\}_{0\leq p<N}$ et des coefficients réels positifs $\{s_p\}_{0\leq p<N}$ tels que*

$$\begin{aligned} &\forall(\chi,z,\rho)\in[0,\chi_{max}]\times[z_{min},z_{max}]\times[-\rho_{max},+\rho_{max}],\\ &\left|K(\chi,z,\rho)-\sum_{p=0}^{N-1}s_p\,u_p(\chi)\,v_p(z,\rho)\right|\leq\varepsilon \end{aligned} \tag{4.20}$$

*Par ailleurs : $v_p(-z,\rho)=(v_p(z,\rho))^*$*

**Preuve** *Ce premier lemme reprend le principe de décomposition en valeurs singulières (SVD) dans son cadre d'origine.*

***Lemme 18*** *Singular Value Decomposition [32].*
*Soit $A$ une matrice de taille $n\times m$, il existe deux matrices $U$ et $V$ à colonnes orthonormales de tailles respectives $n\times p$ et $m\times p$, ainsi qu'une matrice diagonale $S=[s_{ij}]$ à coefficients réels positifs telles que $A=USV^*$ et $s_{ii}\geq s_{i+1,i+1}$ pour tout $i=0,\ldots,p-1$.*

*Comme nous pouvons le voir, le principe de décomposition SVD s'applique en dimension finie. Cependant, le noyau $K$ est défini suivant des intervalles continus.*

*Résumons la procédure de Yarvin & Rokhlin conduisant au résultat voulu.*

*Tout d'abord, il convient de discrétiser $K$ suivant une certaine base qui conduira à sa représentation en dimension finie.*

*Considérons tout d'abord le cas simplifié d'une fonction $f:[-1,1]\to\mathbb{R}$ que l'on souhaite interpoler aux $n$ points $x_i$ avec $f_i=f(x_i)$.*

*Il est connu qu'une répartition non appropriée d'un grand nombre de points d'interpolation $\{x_i\}$ conduit à un polynôme interpolateur $p$ de Lagrange mal conditionné*

$$p(x)=\sum_{i=1}^{n}f_i\prod_{\substack{j=1\\j\neq i}}^{n}\frac{x-x_j}{x_i-x_j},$$

*tant relativement aux valeurs $f_i$ que tant numériquement lors son évaluation.*

*Ce problème peut être résolu par l'utilisation de points de Chebyshev et une évaluation de $p$ en terme de polynômes de Chebyshev ou de même, par l'utilisation de points d'interpolation, racines du polynôme de Legendre d'ordre $n$ et une évaluation en terme de polynômes de Legendre.*

***Lemme 19*** *Soient $x_1,\ldots,x_n\in[-1,1]$, les racines du polynôme de Legendre d'ordre $n$ et $\omega_1,\ldots,\omega_n\in\mathbb{R}$ les poids de Gauss associés. Soit $p:[-1,1]\to\mathbb{R}$ le polynôme interpolateur de Lagrange (de degré $n-1$) défini par $p(x_i)=f_i$ pour $i=1,\ldots,n$. Alors il existe $c_0,\ldots,c_{n-1}\in\mathbb{R}$ tels que*

$$p(x)=\sum_{i=0}^{n-1}c_iP_i(x) \tag{4.21}$$

*avec $P_i(x)$ le polynôme de Legendre d'ordre $i$ et nous avons*

$$\sum_{i=1}^{n} \omega_i f_i^2 = \int_{-1}^{1} p(x)^2 \, \mathrm{d}x = \sum_{i=0}^{n-1} c_i^2 \frac{2}{2i+1} \tag{4.22}$$

***Preuve*** *Tout d'abord remarquons que la famille $\{P_i\}_{i\geq 0}$ forme une base orthogonale (au sens de la norme $L^2$ et la relation C.5) de $\mathbb{R}[X]$ où chaque $P_i$ est de degré $i$. Ensuite la quadrature de Gauss d'ordre $n$ de points et poids $\{(x_i, \omega_i)\}_{i=1,\dots,n}$ intègre exactement tout polynôme de degré inférieur ou égal à $2n-1$ et en particulier $\{p \cdot P_i\}_{i=0,\dots,n-1}$ et $p^2$.*

*Ainsi, $p$ se projette exactement sur $\{P_i\}_{i=0,\dots,n-1}$ et l'on obtient (4.21) avec*

$$c_i = \int_{-1}^{1} p(x) P_i(x) \, \mathrm{d}x = \frac{2i+1}{2} \sum_{j=1}^{n} \omega_j f_j P_i(x_j)$$

*ainsi que la relation (4.22).* □

*Choisissons une base de polynômes de Legendre d'ordre élevé $\{P_l\}_{l=0,\dots,L-1}$ et $\{(x_i, \omega_i)\}_{i=1,\dots,L}$ les zéros de $P_L$ et les poids de Gauss associés. L'ordre $L$ est déterminé tel que l'erreur d'interpolation associée au polynôme d'interpolation de Lagrange $p$ soit inférieure à $\epsilon$*

$$\left| K(\chi, z, \rho) - \sum_{i,j,k} K(\chi_i, z_j, \rho_k) \, p(\chi, z, \rho) \right| \leq \epsilon \tag{4.23}$$

*avec $\{\chi_i\}$, $\{z_j\}$, $\{\rho_k\}$ projections sur leur intervalle respectif des points d'interpolation $\{x_i\} \in [-1, 1]$ et $\{w_i^\chi\}$, $\{w_j^z\}$, $\{w_k^\rho\}$ leurs poids de Gauss respectifs.*

*Nous définissons alors la matrice $A_{i,(j,k)}$, où le couple $(j,k)$ représente un indice de colonne, comme suit*

$$A_{i,(j,k)} = K(\chi_i, z_j, \rho_k) \sqrt{w_i^\chi \, w_j^z \, w_k^\rho}.$$

*Ceci revient à considérer $(\cdot,\cdot)$ comme une bijection de $[1 : j_{max}] \times [1 : k_{max}]$ dans $[1 : j_{max} k_{max}]$.*

*Il est alors possible d'appliquer la décomposition SVD à $A$, et nous obtenons la factorisation*

$$A = USV^*$$

*avec $U$ et $V$ matrices à colonnes orthonormales de tailles respectives $n \times p$ et $m \times p$ et $S$ matrice diagonale de taille $p \times p$.*

*De cette manière, nous définissons les fonctions orthonormales suivantes*

$$u_p(\chi) = \sum_i \frac{u_{i,p}}{\sqrt{w_p^\chi}} \, P_i(\chi), \tag{4.24}$$

$$v_p(z, \rho) = \sum_{j,k} \frac{v_{(j,k),p}}{\sqrt{w_j^z \, w_k^\rho}} \, P_j(z) \, P_k(\rho), \tag{4.25}$$

*où $u_{i,p}$ et $v_{(j,k),p}$ sont les éléments des matrices $U$ et $V$ après projection dans la base de polynômes de Legendre d'ordre $L$ définie pour (4.23) et au sens du lemme 19.*

*Ceci permet de retrouver une approximation du noyau $K$, similaire à (4.23)*

$$\left| K(\chi, z, \rho) - \sum_p s_p \, u_p(\chi) \, v_p(z, \rho) \right| \leq \varepsilon$$

*en restreignant $p$ aux valeurs vérifiant $\forall p \geq N,\ s_p < \epsilon$.*

*Par ailleurs notre noyau $K$ nous fournit en propriété complémentaire que $(u_p)_p$ est une base de fonctions réelles. En effet, à partir de l'intégrale suivante qui est de valeur réelle pour tout $\chi$, $\chi'$ :*

$$\int_{z_{min}}^{z_{max}} \int_{-\rho_{max}}^{\rho_{max}} K(\chi, z, \rho)(K(\chi', z, \rho))^* \ \mathrm{d}z\, \mathrm{d}\rho$$
$$= \int_{z_{min}}^{z_{max}} e^{-(\chi+\chi')z}\, \mathrm{d}z \times \sin\left(\rho_{max}(\sqrt{\chi^2+\kappa^2} - \sqrt{\chi'^2+\kappa^2})\right)$$

*nous obtenons que $\sum_p s_p^2 u_p(\chi)(u_p(\chi'))^*$ est réel. L'argument de la valeur complexe $u_p(\chi)$ est donc indépendante de $\chi$. Il est alors toujours possible de modifier l'argument complexe de $v_p(z,\rho)$ tel que $u_p(\chi)$ soit réel. Par ailleurs, nous avons $v_p(z,-\rho) = (v_p(z,\rho))^*$.* □

Le point clef que nous avons ici obtenu est la séparation de la variable $\chi$ du couple $(z, \rho)$.

**Remarque** *Il n'est pas nécessaire de connaître le polynôme interpolateur $p$ pour la suite du calcul, seul $L$ suffit.*

*Afin de réduire l'ordre des polynômes tout en garantissant une bonne approximation, il est possible de considérer une décomposition dans une base de polynômes de Legendre par morceaux où chaque morceau est construit de manière adaptative.*

*L'ordre de l'interpolation dans chaque direction pourrait tout à fait être différent.*

En suivant l'approche de quadrature gaussienne généralisée de Yarvin & Rokhlin [77], il est possible de construire une quadrature en $\chi$ à $N/2$ points $\chi_i$ et $N/2$ poids $w_i$ intégrant exactement les $N$ premières fonctions $u_p(\chi)$. Ainsi, grâce à la décomposition 4.20, nous obtenons une quadrature de

$$\int_0^{\chi_{max}} K(\chi, z, x\cos\phi + y\sin\phi)\ \mathrm{d}\chi$$

avec une précision $\epsilon$, pour tout $(x, y, z)$ tel que $\sqrt{x^2+y^2} \leq \rho_{max}$ et $z \in [z_{min}, z_{max}]$.

Par commodité, notons $n \in \mathbb{N}$ le nombre vérifiant $2n = N$. Si toutefois $N$ devait être impair, nous convenons d'arrondir $N$ au nombre pair supérieur.

Cette quadrature $x = (\chi_1 \dots \chi_n, w_1 \dots w_n)$ est solution du système non-linéaire de dimension $2n$ suivant :

$$\sum_{j=1}^{n} w_j u_i(\chi_j) = \int_0^{\chi_{max}} u_i(\chi)\, \omega(\chi)\, \mathrm{d}\chi \quad \forall i \in \{1 \dots 2n\} \tag{4.26}$$

avec $\omega(\chi) \equiv 1$ le poids d'intégration du système non modifié. Notons que les intégrales issues du second membre peuvent être calculées exactement grâce à la décomposition (4.24).

**Lemme 20** *Méthode de Newton [53]*

*Soit un système d'équations non-linéaires de la forme $F(x) = 0$, avec $F : \mathbb{R}^N \to \mathbb{R}^N$ et $F \in C^1$. Soit $J$ la matrice jacobienne de $F$ définie par*

$$J_{ij}(x) = \frac{\partial F_i}{\partial x_j}$$

*avec $F_i$ et $x_i$ respectivement $i^{ème}$ composantes de $F$ et de $x$.*

*Supposons que $x$ existe tel que $F(x) = 0$ et $detJ(x) \neq 0$.*
*Soit $x^0 \in \mathbb{R}^N$ un point initial et la suite $x^1, x^2 \ldots$ définie par*

$$x^{k+1} = x^k - J^{-1}(x^k)F(x^k), \tag{4.27}$$

*alors, il existe $\epsilon > 0$ tel que pour tout $x^0$ vérifiant $\|x^0 - x\| < \epsilon$, la suite (4.27) converge quadratiquement.*

Bien que la méthode de Newton offre une méthode convergente vers la solution du système (4.26), le choix du point initial est déterminant et d'autant plus difficile que $N$ est grand.

La méthode de continuation permet d'aborder le problème du choix du point initial.

**Lemme 21** *Méthode de continuation [53]*
*Supposons que l'on souhaite résoudre $F(x) = 0$ comme dans le lemme 20.*
*Soit $G : [0,1] \times \mathbb{R}^N \to \mathbb{R}^N$ vérifiant les propriétés suivantes :*
- *Pour tout $x \in \mathbb{R}^N$, $G(1,x) = F(x)$.*
- *La solution $x_0$ de $G(0,x) = 0$ est connue.*
- *Pour tout $\alpha \in [0,1]$, l'équation $G(\alpha, x_\alpha) = 0$ a une unique solution $x_\alpha$ vérifiant les hypothèses du lemme 20.*
- *La solution $x_\alpha$ est une fonction continue de $\alpha$.*

*Alors, il existe une suite croissante $\{\alpha_i\}_{i\in\{0\ldots m\}}$ avec $\alpha_0 = 0$, $\alpha_m = 1$ telle qu'il est possible de résoudre successivement*

$$G(\alpha_i, x_{\alpha_i}) = 0$$

*par la méthode de Newton à partir d'un point initial choisi comme la solution $x_{\alpha_{i-1}}$ du système associé à $\alpha_{i-1}$.*

*Un exemple de telle suite est $\alpha_i = i/m$ avec $m$ suffisamment grand afin que la méthode de Newton converge à chaque étape.*

L'application des lemmes 20 et 21 amène à la résolution du système (4.26) avec en particulier :

$$\frac{\partial F_i}{\partial w_j} = u_i(\chi_j), \quad \frac{\partial F_i}{\partial \chi_j} = w_i u_i'(\chi_j).$$

et

$$\omega(\alpha, x) = \alpha + (1-\alpha)\sum_{j=1}^{n} \delta(x - c_j)$$

où $\omega$ est une modification du poids d'intégration (second membre de 4.26) supportant la forme de continuation autour de points $c_j \in [0, \chi_{max}]$, a priori, arbitraires. Ce poids d'intégration vérifie les propriétés de la méthode continuation :
- Avec $\alpha = 1$, nous obtenons le système $F(x) = 0$.
- Avec $\alpha = 0$, la solution est connue ($w_j = 1$, $\chi_j = c_j$).
- Les poids et points de quadrature $(\chi_i, w_i)$ dépendent continûment de $\alpha$.

La construction de la suite $\{\alpha_i\}_{i\in\{0\ldots m\}}$ est adaptative de telle manière que le pas $\delta_i \stackrel{\text{def}}{=} \alpha_{i+1} - \alpha_i$ est réduit lorsque la méthode de Newton ne converge pas en un nombre limité d'itérations, et augmenté en cas de succès répétés.

Parallèlement à la méthode de continuation, il est possible d'éviter d'affronter le système de taille $2n$ directement. Une résolution successive des sous-systèmes de taille 2, 4, 6,...,2n

(toujours résolus par continuation) permet d'aborder progressivement chaque problème de dimension supérieure avec une construction naturelle de l'initialisation du système de taille $2m$ à partir de la solution $(x_1, \ldots, x_{m-1})$ du système de taille $2(m-1)$ :

$$\begin{aligned} c_1 &= x_1 \\ c_i &= (x_{i-1} - x_i)/2, \quad i = 2, \ldots, m-1 \\ c_n &= x_{m-1} \end{aligned}$$

Cette approche de la résolution du système non-linéaire permet une construction fiable des points et poids de quadrature $(\chi_i, w_i)$.

Illustrons numériquement qu'un nombre réduit de points $n_\chi$ permet d'approcher efficacement $K(\chi, z, \rho)$. Nous prendrons comme cas de référence le précédent test (figure 4.6) présentant le nombre de points de quadrature uniforme nécessaire en fonction de $z$ avec $\kappa = 1$.

Ce test est reproduit par quadrature optimale sur la figure 4.7. Une première remarque est qu'une allure générale similaire est observable : 2 régions, basse et haute fréquences, où peu de points sont requis, et 1 région intermédiaire nécessitant plus de points. Cependant, il convient de noter que le maximum de points nécessaires passe de 100 à 17 et que le nombre de points de quadrature optimal reste toujours en dessous de celui obtenu par quadrature uniforme.

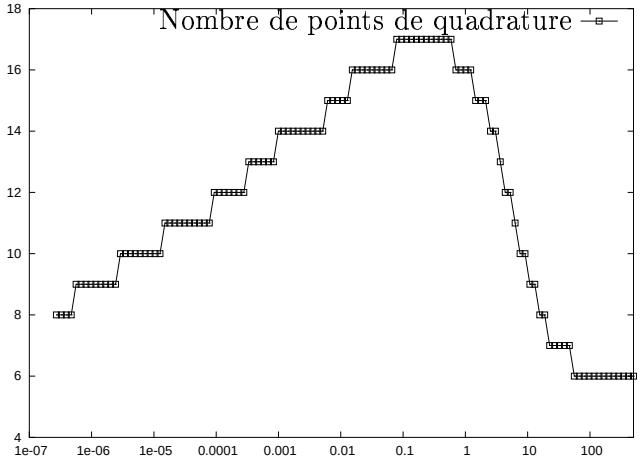



FIG. 4.7 – Nombre de points de quadrature suivant $\chi$ en fonction de $z$, par méthode de quadrature optimale

La figure 4.8 nous permet par ailleurs d'apprécier le comportement de $n_\chi$ en fonction de l'erreur $\epsilon$ aux environs de l'extremum de la figure 4.6, c'est-à-dire pour $z = 10^{-3}$, $\kappa = 1$. Ainsi, nous observons que $n_\chi$ y est de l'ordre de $O(\log \epsilon^{-1})$.

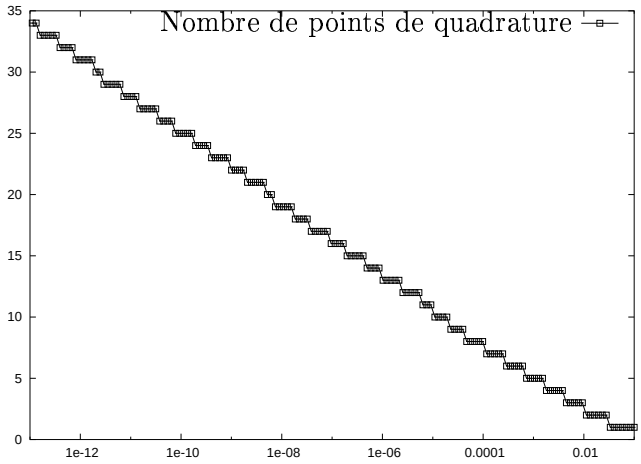



FIG. 4.8 – Nombre de points de quadrature suivant $\chi$ en fonction de l'erreur donnée $\epsilon$, par méthode de quadrature optimale

### Application à la construction d'un algorithme FMM

Dans le soucis d'alléger les équations, nous allons introduire quelques notations :

$$
\begin{aligned}
l &: \text{niveau dans l'arbre ; les feuilles à } l = 0, \text{ la racine à } l = l^{max}, \\
u_p^l(\chi)\&v_p^l(z,\rho) &: \text{décomposition SVD de } K \text{ au niveau } l, \\
\vec{z} &= (x, y, z) \in \mathbb{R}^3, \\
\vec{\phi} &= (\cos\phi, \sin\phi, 1) \in \mathbb{R}^3 \text{ pour tout angle } \phi \in [0, 2\pi], \\
\vec{u} : \vec{v} &= (u_3 v_3, u_1 v_1 + u_2 v_2) \in \mathbb{R}^2 \quad \text{pour } u = (u_1, u_2, u_3), v = (v_1, v_2, v_3), \\
v_p^l(\vec{z} : \vec{\phi}) &\stackrel{\text{def}}{=} v_p^l(z, x\,\cos\phi + y\,\sin\phi), \\
K(\chi, \vec{z} : \vec{\phi}) &\stackrel{\text{def}}{=} K(\chi, z, x\,\cos\phi + y\,\sin\phi).
\end{aligned}
$$

L'algorithme FMM peut se résumer en quelques étapes (*cf* Chapitre 2) :

1. *Initialisation* : passage de la position des particules à une représentation discrète.
2. *Agrégation* : construction de la représentation discrète d'un niveau supérieur en partant de la représentation discrète des cellules enfants.
3. *Transfert* : ajout de la contribution d'une cellule lointaine.
4. *Dispersion* : reconstruction des discrétisations des cellules enfants en partant d'une cellule du niveau supérieur (Correspond globalement à l'opérateur transposé de l'*Agrégation*).
5. *Finalisation* ou *Intégration* : calcul du potentiel final résultat de l'action des autres particules (Cette étape revient à calculer une intégrale à partir de la forme discrète).

Nous allons maintenant reprendre chacune de ces étapes afin d'en expliquer donner quelques détails.

***Initialisation* : d'une position de particule à une représentation** $(p, \phi)$

A un niveau $l$, définissons l'expansion multipolaire $\nu^p_{C^l_a}(\phi)$ comme somme sur toutes les sources $\vec{z}_i$ d'une cellule $C^l_a$ d'intensités respectives $\mu_i$

$$\nu^p_{C^l_a}(\phi) \stackrel{\text{def}}{=} \int_0^{\chi^l_{max}} u^l_p(\chi)^* \sum_{\vec{z}_i \in C^l_a} \mu_i \, K\left(\chi, (\vec{z}_i - \vec{z}^{\,a,l}_g) : \vec{\phi}\right) \, \mathrm{d}\chi \tag{4.28}$$

Cette expansion est centrée autour du point $\vec{z}^{\,a,l}_g$. Ce centre d'expansion peut être choisi de manière optimale comme suit :

**Proposition 22** *Notons $O^l_a$ le centre de la cellule $C^l_a$ et $r^l$ la longueur d'un de ses côtés.*
*Prendre*

$$\vec{z}^{\,a,l}_g : \begin{cases} x^{a,l}_g = x_{O^l_a} \\ y^{a,l}_g = y_{O^l_a} \\ z^{a,l}_g = z_{O^l_a} + \frac{3}{2}\, r^l \end{cases}$$

*est optimal au sens où, pour une erreur donnée $\epsilon$, le nombre de termes dans la décomposition SVD est minimal.*

**Argumentation** *Introduisons un paramètre $\alpha$ et choisissons le centre d'expansion comme :*

$$\begin{cases} x^{a,l}_g = x_{O^l_a} \\ y^{a,l}_g = y_{O^l_a} \\ z^{a,l}_g = z_{O^l_a} + (\alpha + \frac{1}{2})\, r^l \end{cases}$$

*Afin de simplifier cette preuve, considérons le niveau 0 avec $r^0 = 1$. La décomposition SVD doit être considérée pour :*

$$\chi \in [0, \chi^l_{max}] \text{ tel que } \exp(-\chi^l_{max}\, \alpha) \leq \epsilon,$$
$$z \in [\alpha, 1 + \alpha],$$
$$\rho \in [-2^l/\sqrt{2}, 2^l/\sqrt{2}].$$

*Lorsque $\alpha$ est petit, $\chi^l_{max}$ devient grand et $K$ devient très oscillant suivant $\rho$, ce qui signifie que la décomposition SVD convergera lentement.*

*Par ailleurs, quand $\alpha > 1$, la fonction*

$$\sum_i \mu_i \, K\left(\chi, (\vec{z}_i - \vec{z}^{\,a,l}_g) : \vec{\phi}\right)$$

*se verra multipliée (à voir comme une étape de transfert) par*

$$K\left(\chi, (\vec{z}^{\,a,l}_g - \vec{z}_j) : \vec{\phi}\right).$$

*Or, en général, $z^{a,l}_g - z_j$ ne peut être négatif à moins que $\alpha \leq 1$. Si jamais $z^{a,l}_g - z_j$ était négatif, la fonction $K\left(\chi, (\vec{z}^{\,a,l}_g - \vec{z}_j) : \vec{\phi}\right)$ deviendrait arbitrairement grande, ce qui rendrait la méthode instable.*

*Ainsi, la valeur optimale ne peut être que $\alpha = 1$.* ♢

### *Agrégation* et *Dispersion* par un décalage des origines

La représentation (4.28) doit être obtenue pour chaque paquet à chaque niveau. Il serait bien coûteux de réutiliser la méthode utilisée au niveau des feuilles aux niveaux supérieurs. En effet, on peut voir que les niveaux supérieurs contiennent les mêmes informations (positions et potentiels des particules), mais dont les centres d'expansion (liés aux cellules) différent. Ainsi, une meilleure efficacité peut être obtenue en réutilisant les $\nu^p_{C_a^l}(\phi)$ du niveau $l$ pour calculer les $\nu^p_{C_b^{l+1}}(\phi)$ du niveau $l+1$.

Ces calculs consistent donc en une agrégation des calculs liés aux cellules $C_a^l$, cellules enfants de $C_b^{l+1}$ (au sens de l'octree), pour en déduire $\nu^p_{C_b^{l+1}}(\phi)$. Cette agrégation implique un ré-échantillonnage des discrétisations suivant $\chi$ et $\phi$ ainsi qu'un décalage des origines de $\vec{z}_g^{\,a,l}$ vers $\vec{z}_g^{\,b,l+1}$.

Nous avons déjà vu que ce ré-échantillonnage suivant $\phi$ (discrétisation uniforme de $[0, 2\pi]$) peut être obtenu simplement via une Transformée de Fourier Rapide (FFT). Ainsi nous sommes capables de passer d'un niveau fin à un niveau plus grossier, et vice versa, suivant la variable $\phi$.

L'opération suivant $\chi$ est plus originale et nous utiliserons pour cela le résultat suivant :

**Proposition 23** *Considérons un paquet $C_a^l$, son parent direct $C_b^{l+1}$ et leurs centres d'expansion respectifs $\vec{z}_g^{\,a,l}$ et $\vec{z}_g^{\,b,l+1}$.*

*Il existe une matrice pleine $M^l(\vec{z} : \vec{\phi}) = [M^l_{pq}(\vec{z} : \vec{\phi})]$ vérifiant*

$$\left\| \nu_{C_b^{l+1}, C_a^l}(\phi) - M^l\big((\vec{z}_g^{\,a,l} - \vec{z}_g^{\,b,l+1}) : \vec{\phi}\big)\ \nu_{C_a^l}(\phi) \right\|_\infty \leq \epsilon,$$

i.e. *qui permet un décalage des origines d'expansion, une interpolation adaptée suivant $\chi$ et donnant la contribution $\nu^p_{C_b^{l+1}, C_a^l}(\phi)$ de $C_a^l$ à son parent $C_b^{l+1}$.*

*$\nu^p_{C_b^{l+1}}(\phi)$ est alors obtenu par sommation des différentes contributions $\nu^p_{C_b^{l+1}, C_a^l}(\phi)$ de ses enfants $\{C_a^l\}_a$*

$$\nu^p_{C_b^{l+1}}(\phi) = \sum_{C_a^l \text{ fils de } C_b^{l+1}} \nu^p_{C_b^{l+1}, C_a^l}(\phi).$$

**Preuve** *Introduisons la matrice $M^l(\vec{z} : \vec{\phi})$ suivante*

$$M^l_{pq}(\vec{z} : \vec{\phi}) = \int (u_p^{l+1}(\chi))^*\ K\Big(\chi, \vec{z} : \vec{\phi}\Big)\ u_q^l(\chi)\ \mathrm{d}\chi.$$

*Par définition de $\nu^p_{C_b^{l+1}}(\phi)$, la contribution de $C_a^l$ à $C_b^{l+1}$ est*

$$\nu^p_{C_b^{l+1}, C_a^l}(\phi) = \int u_p^{l+1}(\chi)^* \sum_{\vec{z}_i \in C_a^l} \mu_i\ K\left(\chi, (\vec{z}_i - \vec{z}_g^{\,b,l+1}) : \vec{\phi}\right)\ \mathrm{d}\chi.$$

*Puisque $K$ vérifie*

$$K\Big(\chi, (\vec{z}_1 + \vec{z}_2) : \vec{\phi}\Big) = K(\chi, \vec{z}_1 : \vec{\phi})\ K(\chi, \vec{z}_2 : \vec{\phi}),$$

*que la décomposition SVD de $K$*

$$K(\chi, z, \rho) \approx \sum_{p=0}^{N_l - 1} s_p^l\, u_p^l(\chi)\, v_p^l(z, \rho)$$

*et que* $\{u_p^{l+1}\}_p$ *est une base orthonormale,* $M^l$ *vérifie la proposition.* □

L'étape de dispersion correspond conceptuellement à la transposée de l'agrégation. Il nous reste à montrer cela.

Tout d'abord, il convient de définir le centre d'expansion de l'étape de dispersion.

$$\vec{z}_s^{a,l} : \begin{cases} x_s^{a,l} = x_{O_a^l} \\ y_s^{a,l} = y_{O_a^l} \\ z_s^{a,l} = z_{O_a^l} - \frac{3}{2}\, r^l. \end{cases}$$

La proposition 22 peut être adaptée à l'étape de dispersion. Ce centre est optimal pour l'étape de dispersion.

Au niveau $l$, nous définissons l'expansion locale $\lambda^p_{C_a^l}(\phi)$ comme la somme sur toutes les sources $\vec{z}_i$ localisées dans la cellule $C_a^l$ d'intensités respectives $\mu_i$

$$\lambda^p_{C_a^l}(\phi) \stackrel{\text{def}}{=} \int_0^{\chi^l_{max}} u_p^l(\chi) \sum_{\vec{z}_i \in C_a^l} \mu_i \; K\left(\chi, (\vec{z}_i - \vec{z}_s^{a,l}) : \vec{\phi}\right) \; \mathrm{d}\chi$$

Nous avons vu précédemment comment la matrice $M^l(\vec{z} : \vec{\phi})$ permettait un décalage du centre d'expansion multipolaire ainsi qu'une interpolation suivant $\chi$. Voyons maintenant comment cette même matrice permet encore un décalage du centre d'expansion local et une anterpolation appropriée suivant $\chi$.

**Proposition 24** *Considérons une cellule* $C_a^l$*, son parent* $C_b^{l+1}$ *et leurs centres d'expansions locaux respectifs* $\vec{z}_s^{a,l}$ *et* $\vec{z}_s^{b,l+1}$*.*

*La contribution à* $C_a^l$ *en partant de* $C_b^{l+1}$ *peut être obtenue comme vérifiant*

$$\left| \lambda^p_{C_a^l}(\phi) - \sum_q M^l_{qp}((\vec{z}_s^{b,l+1} - \vec{z}_s^{a,l}) : \vec{\phi}) \; \lambda^q_{C_b^{l+1}}(\phi) \right| \leq \epsilon \tag{4.29}$$

**Preuve** *Pour tout* $\vec{z}_i$*, nous avons*

$$\begin{aligned} &\sum_q M^l_{qp}\Big((\vec{z}_s^{b,l+1} - \vec{z}_s^{a,l}) : \vec{\phi}\Big) \int u_q^{l+1}(\chi) \; K\left(\chi, (\vec{z}_i - \vec{z}_s^{b,l+1}) : \vec{\phi}\right) \; \mathrm{d}\chi \\ &\quad = \int u_p^l(\chi) \; K\Big(\chi, (\vec{z}_s^{b,l+1} - \vec{z}_s^{a,l}) : \vec{\phi}\Big) \\ &\qquad \times \sum_q (u_q^{l+1}(\chi))^* \int_0^{\chi^{l+1}_{max}} u_p^{l+1}(\tilde{\chi}) \; K\left(\tilde{\chi}, (\vec{z}_i - \vec{z}_s^{b,l+1}) : \vec{\phi}\right) \; \mathrm{d}\tilde{\chi} \; \mathrm{d}\chi \end{aligned}$$

*La preuve serait immédiate si nous avions*

$$K\Big(\chi, (\vec{z}_i - \vec{z}_s^{b,l+1}) : \vec{\phi}\Big) \approx \sum_q (u_q^{l+1}(\chi))^* \int_0^{\chi^{l+1}_{max}} u_q^{l+1}(\tilde{\chi}) K\Big(\tilde{\chi}, (\vec{z}_i - \vec{z}_s^{b,l+1}) : \vec{\phi}\Big) \; \mathrm{d}\tilde{\chi}.$$

*Cependant cela n'est pas vrai dans un cas général.*

*Nous pouvons toutefois prouver l'identité suivante :*

$$M_{pq}^{l}((\vec{z}_s^{\,b,l+1} - \vec{z}_s^{\,a,l}) : \vec{\phi}) = \frac{s_p^{l+1}}{s_q^l} \int (v_q^l(z,\rho))^* \; v_p^{l+1}((z,\rho) + (\vec{z}_s^{\,b,l+1} - \vec{z}_s^{\,a,l}) : \vec{\phi}) \; \mathrm{d}z \; \mathrm{d}\rho. \quad (4.30)$$

*Cette identité nous permet alors d'écrire :*

$$\begin{aligned}
\sum_q M_{qp}^{l}((\vec{z}_s^{\,b,l+1} - \vec{z}_s^{\,a,l}) : \vec{\phi}) \int u_q^{l+1}(\chi) K\Big(\chi, (\vec{z}_i - \vec{z}_s^{\,b,l+1}) : \vec{\phi}\Big) \; \mathrm{d}\chi \\
&\approx \frac{1}{s_p^l} \int\int (v_q^l(z,\rho))^* \; K(\chi, z, \rho) \; \mathrm{d}z \; \mathrm{d}\rho \\
&\quad \times \; K\Big(\chi, (\vec{z}_i - \vec{z}_s^{\,b,l+1}) : \vec{\phi}\Big) \; K\Big(\chi, (\vec{z}_s^{\,b,l+1} - \vec{z}_s^{\,a,l}) : \vec{\phi}\Big) \; \mathrm{d}\chi \\
&= \int u_q^l(\chi) \; K\Big(\chi, (\vec{z}_i - \vec{z}_s^{\,a,l}) : \vec{\phi}\Big) \; \mathrm{d}\chi
\end{aligned}$$

*Ainsi, par sommation sur $\vec{z}_i \in C_a^l$, de potentiels respectifs $\mu_i$, nous obtenons le résultat. L'erreur commise est bornée par la différence*

$$\Big| K\Big(\chi, (z,\rho) + (\vec{z}_s^{\,b,l+1} - \vec{z}_s^{\,a,l}) : \vec{\phi}\Big) - \sum_q s_q^{l+1} \; u_q^{l+1}(\chi) \; v_q^{l+1}((z,\rho) + (\vec{z}_s^{\,b,l+1} - \vec{z}_s^{\,a,l}) : \vec{\phi}) \Big| \leq \epsilon.$$

□

Une borne $\epsilon / s_p^l$ signifie que pour de grandes valeurs de $p$ les coefficients sont calculés moins précisément que pour de petites valeurs de $p$. Cependant, ceci n'affecte en rien la précision du résultat final du produit matrice-vecteur, puisque le potentiel est obtenu comme :

$$\sum_p s_p^l v_p^l(z,\rho) \; \lambda_{C_a^l}^p(\phi).$$

L'erreur sur le produit matrice-vecteur est ensuite inférieure à $\epsilon$.

En appliquant l'équation (4.29), nous utilisons une approximation de $\lambda_{C_a^l}^p(\phi)$, $\tilde{\lambda}_{C_a^l}^p(\phi)$, pour laquelle :

$$\big|\lambda_{C_a^l}^p(\phi) - \tilde{\lambda}_{C_a^l}^p(\phi)\big| \leq \frac{\epsilon}{s_p^l}.$$

Nous prouvons que la majoration de l'erreur (4.29) reste encore valide lorsque nous utilisons $\tilde{\lambda}_{C_a^l}^p(\phi)$ au lieu de $\lambda_{C_a^l}^p(\phi)$ :

**Proposition 25** *Supposons que*

$$\big|\lambda_{C_a^l}^q(\phi) - \tilde{\lambda}_{C_a^l}^q(\phi)\big| \leq \frac{\epsilon}{s_q^{l+1}}$$

*pour tout $q$, alors :*

$$\Big|\lambda_{C_a^l}^p(\phi) - \sum_q M_{qp}^l((\vec{z}_s^{\,b,l+1} - \vec{z}_s^{\,a,l}) : \vec{\phi}) \; \tilde{\lambda}_{C_b^{l+1}}^q(\phi)\Big| \leq \frac{\epsilon}{s_p^l}.$$

Ce qui est essentiel afin d'assurer la précision de la méthode multiniveaux.

**Preuve** *Utiliser l'équation (4.30).* □

***Transfert : Inner-to-Outer transform***

Cette étape correspond au transfert de chaque paquet vers ses paquets lointains. Il faut noter que cette étape représente la plus grande part du coût calculatoire de la FMM. Il est donc primordial d'en obtenir un algorithme efficace.

La meilleure efficacité peut être obtenue à l'aide la propriété d'additivité du noyau $K$

$$K(\chi, z+z', \rho) = K(\chi, z, \rho)\ K(\chi, z', \rho).$$

Une étape supplémentaire est alors nécessaire en recalculant une représentation $(\chi, \phi)$

$$\sum_p \nu^p_{C^l_a}(\phi) u_p(\chi)$$

qui n'est qu'une approximation de

$$\sum_i \mu_i K\Big(\chi, (\vec{z}_i - \vec{z}^{\,a,l}_g) : \vec{\phi}\Big).$$

Ce calcul est un calcul préalable à l'étape de transfert à proprement dite.

Il est alors possible d'effectuer tous les transferts via une multiplication terme à terme par $T^l_{ba}(\chi, \phi)$ :

$$T^l_{ba}(\chi, \phi) \stackrel{\text{def}}{=} K\Big(\chi, (\vec{z}^{\,a,l}_g - \vec{z}^{\,b,l}_s) : \vec{\phi}\Big).$$

L'opérateur de transfert est alors diagonal, ce qui constitue un bien meilleur algorithme qu'un opérateur de transfert basé sur le développement multipolaire $(p, \phi)$. La complexité algorithmique de cette étape rejoint ainsi celle de l'expansion standard en onde plane : $O(p^2)$, qui était un de nos objectifs initiaux.

Une fois tous les transferts effectués, nous devons reprendre un représentation de type $(p, \phi)$ afin poursuivre l'algorithme par le calcul des fonctions $\lambda^q_{C^l_a}(\phi)$. Cette étape nécessite une quadrature adaptée en $\chi$ comme l'énonce la proposition suivante :

**Proposition 26** *Il existe une quadrature en $\chi$ permettant de calculer*

$$\lambda^q_{C^l_a}(\phi) = \int u^l_q(\chi) \Big\{ \sum_b T^l_{ba}(\chi, \phi) \Big( \sum_p \nu^p_{C^l_a}(\phi)\, u^l_p(\chi) \Big) \Big\}\ \mathrm{d}\chi$$

*avec une précision de $\epsilon / s^l_q$.*

**Preuve** *V. Rokhlin, dans [77], a introduit les quadratures gaussiennes généralisées ("Generalized Gaussian Quadratures") afin d'intégrer des fonctions comme $K(x,t)$ suivant $x$ pour $t$ variant dans un certain intervalle. Notre cas nous demande de considérer $t$ comme $(z, \rho) \in \mathbb{R}^2$ (ou plutôt des intervalles bornés).*

*La construction de cette quadrature suit la décomposition SVD de notre noyau $K$.*

*Supposons que nous travaillons au niveau 0 et que la taille des cellules est alors de 1. La proposition 17 nous donne*

$$\Big| K(\chi, z, \rho) - \sum_{p=1}^{N} s_p u_p(\chi) v_p(z, \rho) \Big| \leq \epsilon$$

*pour tout* $\chi \in [0, \chi^0_{max}]$, $\chi^0_{max} = \log 1/\epsilon$, $z \in [1,4]$ *et* $\rho \in [-4\sqrt{2}, 4\sqrt{2}]$. *Ces intervalles représentent l'intervalle d'intégration optimal (relativement à* $\epsilon$*) suivant* $\chi$ *et les portées maximales des transferts à ce niveau suivant* $z$ *et* $\rho$.

*En suivant la démarche de V. Rokhlin, il nous est possible de construire une quadrature en* $\chi$ *avec seulement* $N/2$ *points et poids :* $\{(\chi_k, \omega_k)\}_k$ *(cf 4.2.2).*

*Prenons deux points* $\vec{z}_a \in C^0_a$ *et* $\vec{z}_b \in C^0_b$, *tels que* $C^0_b$ *soit dans les interactions (transferts) de* $C^0_a$.

*Nous avons*

$$\Big| \int K\Big(\chi, (\vec{z}_a - \vec{z}_b) : \vec{\phi}\Big)\ \mathrm{d}\chi - \sum_{k=1}^{m/2} \omega_k\ K\Big(\chi_k, (\vec{z}_a - \vec{z}_b) : \vec{\phi}\Big)\Big| \leq \epsilon$$

*qu'il nous est possible de décomposer, grâce à la propriété d'additivité de* $K$ :

$$K\Big(\chi, (\vec{z}_a - \vec{z}_b) : \vec{\phi}\Big) = K\Big(\chi, (\vec{z}_a - \vec{z}^{\,a,0}_g) : \vec{\phi}\Big)\ T^0_{ba}(\chi, \phi)\ K\Big(\chi, (\vec{z}^{\,b,0}_s - \vec{z}_b) : \vec{\phi}\Big).$$

*En appliquant la décomposition SVD à* $K\Big(\chi, (\vec{z}^{\,b,0}_s - \vec{z}_b) : \vec{\phi}\Big)$ *et en utilisant le fait que* $v^0_p(z, \rho)$ *est une base orthonormale, nous prouvons que* $\{(\chi_k, \omega_k)\}$ *permet d'intégrer*

$$\int s^0_q u^0_q(\chi)\ T^0_{ba}(\chi, \phi)\ K\Big(\chi, (\vec{z}_a - \vec{z}^{\,a,0}_g) : \vec{\phi}\Big)\ \mathrm{d}\chi$$

*avec une précision de* $\epsilon$. *Par ailleurs, l'erreur sur* $\lambda^q_{C^0_b}(\phi)$ *est de l'ordre de* $\epsilon / s^0_q$.

*Cette preuve est applicable à tout autre niveau* $l$, *comportant des cellules de taille* $r$, *en choisissant* $z \in [r, 4r]$, $\rho \in [-4\sqrt{2}r, +4\sqrt{2}r]$, $\chi \in [0, \chi^l_{max}]$, $\exp(-\chi^l_{max} r) \leq \epsilon$. □

## 4.3 Extensions

### 4.3.1 Optimisation des calculs aux feuilles

La section 4.2.2 donne une formule permettant l'initialisation des coefficients $\nu^p_{C^l_a}$. Toutefois, dans un but d'implémentation numérique, cette équation n'est pas constructive. Une idée fut d'utiliser l'orthonormalité de la famille $\{u_p\}_p$

$$\int_0^{\chi_{max}} u^*_p(\chi)\, u_q(\chi)\, \mathrm{d}\chi = \delta_{pq}$$

qui permet d'écrire

$$\nu^p_{C^l_a}(\phi) = \sum_{\vec{z}_i \in C^l_a} \mu_i\ v^l_p\ \Big((\vec{z}_i - \vec{z}^{\,a,l}_g) : \vec{\phi}\Big)$$

Cependant les fonctions $v^l_p$ sont coûteuses à évaluer. C'est pourquoi, nous allons proposer une autre solution plus performance, basée sur un calcul efficace de l'intégral définissant $\nu^p_{C^l_a}$.

**Proposition 27** *Pour toute précision* $\epsilon$ *donné, une distribution uniforme de points suivant* $\chi$ *de taille* $n^{max}_\chi$ *permet un calcul de* $\nu^p_{C^l_a}(\phi)$ *à* $\epsilon$ *près, avec* $n^{max}_\chi$ *la plus grande largeur de bande de*

$$\mathcal{R}eK(\chi^2)\ \textit{et}\ \mathcal{I}mK(\chi^2)/\sqrt{\chi^4 + \kappa^2}$$

**Preuve** *Ce calcul utilisera un argument maintenant usuel : «les fréquences en dehors du spectre de Fourier ne contribuent pas à l'intégrale».*

*Une première chose est une réécriture de l'intégrale*

$$\int_0^{\chi_{max}} u_p^*(\chi) K(\chi, \vec{z} : \vec{\phi}) \ \mathrm{d}\chi = \int_{-\sqrt{\chi_{max}}}^{\sqrt{\chi_{max}}} K(\chi^2, \vec{z} : \vec{\phi}) \ 2\chi \, u_p^*(\chi^2) \mathbf{1}_{\mathbb{R}^+}(\chi) \ \mathrm{d}\chi$$

*Un obstacle à une utilisation généralisée de ce principe est que le spectre de Fourier de $K(\chi^2)$ est à décroissance lente. Cependant, nous avons remarqué que les spectres de Fourier de $\mathcal{R}eK(\chi^2)$ et de $\mathcal{I}mK(\chi^2)/\sqrt{\chi^4+\kappa^2}$ ont une décroissance rapide (figure 4.9), et ceci contrairement à celui de $\mathcal{I}mK(\chi^2)$ (figure 4.10).*

*Voyons comment nous pouvons tirer avantage de ces comportements afin d'accélérer le calcul.*

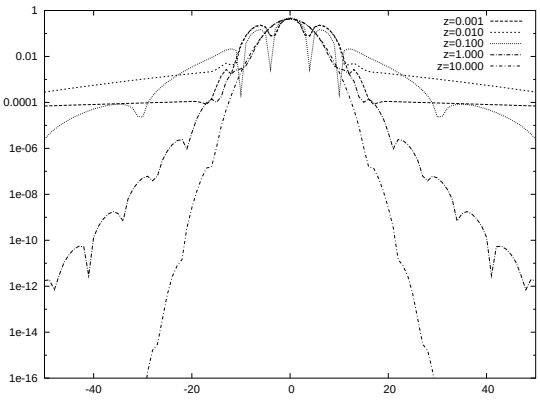



FIG. 4.9 – Spectre de Fourier de $\mathcal{I}mK(\chi^2)$ suivant différentes valeurs de $z$

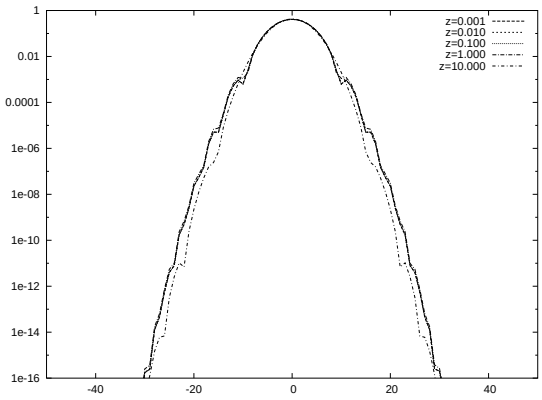



FIG. 4.10 – Spectre de Fourier de $\mathcal{I}mK(\chi^2)/\sqrt{\chi^4+\kappa^2}$ suivant différentes valeurs de $z$

*Notons $n_\chi^{max}$ la plus grande largeur de bande de $\mathcal{R}eK(\chi^2)$ et $\mathcal{I}mK(\chi^2)/\sqrt{\chi^4+\kappa^2}$ (relativement à une erreur imposée de $\epsilon$).*

*Il est maintenant possible d'écrire*

$$\int_0^{\chi_{max}} u_p^*(\chi) \, K(\chi, \vec{z} : \vec{\phi}) \ \mathrm{d}\chi$$
$$= \int_{-\sqrt{\chi_{max}}}^{\sqrt{\chi_{max}}} \mathcal{R}eK(\chi^2, \vec{z} : \vec{\phi}) \times \alpha(\chi) + \mathcal{I}mK(\chi^2, \vec{z} : \vec{\phi})/\sqrt{\chi^4+\kappa^2} \times \beta(\chi) \ \mathrm{d}\chi$$

*avec $\alpha(\chi) = 2\chi \, u_p^*(\chi^2)\mathbf{1}_{\mathbb{R}^+}(\chi)$ et $\beta(\chi) = 2\chi\sqrt{\chi^4+\kappa^2}\, u_p^*(\chi^2)\mathbf{1}_{\mathbb{R}^+}(\chi)$.*

*$\alpha(\chi)$ et $\beta(\chi)$ peuvent ainsi être lissées en supprimant les fréquences plus grandes que $n_\chi^{max}/2$, puisqu'elles ne contribueront pas aux intégrales. Ce lissage permet ensuite d'utiliser une simple quadrature uniforme à $n_\chi^{max}$ points de l'intervalle $[-\sqrt{\chi_{max}}, +\sqrt{\chi_{max}}]$ pour un calcul précis (à $\epsilon$ près) de l'intégrale, et donc de $\nu^p_{C_a^l}(\phi)$.*

*Par ailleurs, le coût de ce calcul peut être réduit en utilisant la symétrie en $\chi$ (bien entendu, avant d'appliquer $\alpha(\chi)$ et $\beta(\chi)$) ainsi que suivant $\phi$ en écrivant $e^{\imath\sqrt{\chi^4+\kappa^2}\,(x\cos\phi+y\sin\phi)}$ comme $e^{\imath\sqrt{\chi^4+\kappa^2}\,x\cos\phi}\, e^{\imath\sqrt{\chi^4+\kappa^2}\,y\sin\phi}$.* □

### 4.3.2 Optimisation par FFT du terme évanescent

Il est possible d'étendre l'accélération d'initialisation des feuilles données en proposition 27 à l'ensemble du terme évanescent.

Ce point n'est pas encore complètement développé, et je ne résumerai qu'en quelques mots les idées sous-jacentes.

L'initialisation, l'intégration et les transferts peuvent être traités par une discrétisation et un lissage définis de manière similaire à la section 4.3.1.

Le point manquant à l'établissement d'un nouvel algorithme FMM est la définition d'opérateurs d'inter/anter-polation.

L'interpolation (respectivement l'anterpolation) peut de nouveau utilisée la décroissance rapide des coefficients de Fourier de $\mathcal{R}eK(\chi^2)$ et $\mathcal{I}mK(\chi^2)/\sqrt{\chi^4+\kappa^2}$. C'est pourquoi, pendant cet étape, seront traités séparément la partie réelle $\mathcal{R}eK(\chi^2)$ et la partie imaginaire modifiée $\mathcal{I}mK(\chi^2)/\sqrt{\chi^4+\kappa^2}$. Notons que le décalage des centres (multiplication par $K(\chi^2,(\vec{z}_g^{\,a,l}-\vec{z}_g^{\,b,l+1})\,:\,\vec{\phi})$) peut encore être effectué sous cette forme modifiée et évite ainsi de multiple divisions par $\sqrt{\chi^4+\kappa^2}$.

Cependant, le traitement par FFT de l'interpolation requiert de travailler sur le même intervalle. Or pour 2 cellules $C^l$ et $C^{l+1}$ telles que $C^l \subset C^{l+1}$, nous avons $\chi^l_{max} > \chi^{l+1}_{max}$. Ce qui rend impossible un traitement direct par FFT. Toutefois, il suffit de rendre compatible ces deux intervalles par une extension du plus court jusqu'au plus grand. Dans le domaine discret, cela impose une discrétisation vérifiant

$$2\chi^l_{max} = k\frac{2\tilde{\chi}^{l+1}_{max}}{n^{max}_{\chi}},\ \text{pour } k \in \mathbb{N}$$

avec $\chi^l_{max}$ fixé, $n^{max}_{\chi}$ la taille de la discrétisation en $\chi$ du niveau $l+1$ et $\tilde{\chi}^{l+1}_{max}$ une valeur adaptée à la contrainte de $\chi^{l+1}_{max}$ ($2\chi_{max}$ correspond à le mesure de l'intervalle $[-\chi_{max},+\chi_{max}]$).

# Deuxième partie

# Implémentation, optimisation logicielle et outils pour la parallélisation

# Chapitre 5

# Approche méthodologique

**Sommaire**



Une fois les fondements algorithmiques généraux exposés (Chapitre *Principes généraux de la Méthode Multipôle Rapide*) et mathématiques propres à notre problème (Chapitres *Formulation Hautes Fréquences pour l'électromagnétisme* et *Formulation adaptatée Toutes Fréquences*), il convient de mettre en œuvre conjointement l'ensemble de ces principes.

Toutefois, plusieurs choix techniques se présentent à l'interface entre ces deux points de vue.

## 5.1 Discrétisation de la surface

Dans le chapitre *Formulations intégrales des équations de Maxwell*, nous avons vu que la discrétisation du problème était réalisée par des éléments finis de type Raviart-Thomas eux-mêmes appuyés sur une triangulation de la surface $\Gamma$.

Toutefois dans le cadre des méthodes multipôles nous avons principalement évoqués des méthodes liées à une représentation en points de l'espace.

Il convient donc de définir une interface entre ces deux exigences. La solution la plus évidente (qui sera effectivement retenue) est d'interfacer les deux approches grâce à une interpolation, permettant de passer de la discrétisation en mode «triangle» à une discrétisation en mode «point», et vice-versa. Cette approche est celle majoritairement développée dans la plupart des codes traitant des équations de Maxwell par formulation intégrale et méthodes multipôles rapides. Par ailleurs, l'ordre d'interpolation et son type peuvent aussi être soumis à un choix, qui dans notre cas, se rangera derrière la solution la plus commune : 1 ou 3 points de Gauss par triangle [21, 11, 74].

Notre problème initial d'interfaçage des deux discrétisations n'est pas pour autant complètement résolu. En effet, l'algorithme multipôle impose de traiter les points (ici, points de Gauss) par paquets suivant la cellule dans laquelle ils se trouvent, c'est-à-dire que l'octree définissant les cellules réalise un partitionnement des points par cellule. Or, il n'est, en général, pas possible d'obtenir que ce partitionnement corresponde exactement avec un partitionnement des triangles : les points de Gauss d'un même triangle peuvent appartenir à différentes cellules (figure 5.1).

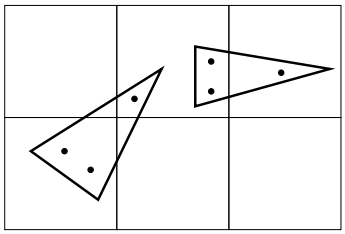


FIG. 5.1 – Partitionnement incompatible entre les points et les triangles

Se présentent alors trois méthodes de partitionnement.

### Partitionnement par triangles

Ce partitionnement consiste tout simplement à localiser un triangle en fonction de son centre de gravité, mais présente l'inconvénient de nuire à la précision du calcul FMM. En effet, cette partition peut contraindre à l'utilisation de fonctions de base $J_i$ débordant de leur cellule de partitionnement, et plus concrètement à considérer comme éloignés (*resp.* proches) des points de Gauss associés qui seraient en réalité proches (*resp.* éloignés), *i.e.* en dehors des plages de validité de l'approximation associée. Il faut toutefois relativiser l'imprécision inhérente à ce partitionnement par le fait que la plupart des triangles sont dans la pratique de taille plus petites que la cellule à laquelle ils sont attribués.

### Partitionnement par arêtes

Tout comme pour le partitionnement par triangles, le partitionnement par arêtes (où l'on dit qu'une arête appartient à une cellule si son centre lui appartient) induit une erreur complémentaire due au débordement potentiel.

Toutefois cette partition semble numériquement plus simple et permet de considérer plus facilement les variantes de formulations intégrales à 3 ou 2 composantes [30].

### Partitionnement par points de Gauss

Ce type de partitionnement permet de considérer à leur juste mesure les interactions proches/lointaines sans accumulation d'erreurs complémentaires, mais présente l'inconvénient de rendre plus complexes les calculs autour des matrices d'interactions proches.

En définitive, nos choix se sont tournés vers le partitionnement en triangles pour sa simplicité, ce qui provoque le partage entre cellules des degrés de liberté (les arêtes), et vers un partitionnement par points de Gauss dans la formulation stable à toutes fréquences. L'utilisation d'une quadrature optimale et de la décomposition SVD requièrent en effet un partitionne-

ment parfait tel que le partitionnement en points de Gauss, puisque les fonctions associées ne peuvent être définies en dehors de leurs cellules respectives. Ce point s'oppose à la HF-FMM qui tolère assez bien ces débordements, notamment grâce à l'utilisation de fonctions dont la validité s'étend jusqu'au cercle circonscrit à la cellule et aux FFT souvent d'ordre légèrement supérieur au strict nécessaire (pour des raisons d'arrondi à un multiple supérieur).

## 5.2 Méthodes implémentées

La présente implémentation recouvre la formulation intégrale de type CFIE (*cf* chapitre 1) à 4 composantes à la fois par une méthode de type HF-FMM (*cf* chapitre 3) et SPW-FMM (chapitre 4). Le principal but de cette implémentation est de démontrer la faisabilité d'une méthode «toutes fréquences» et d'en mesurer les implications tant au niveau algorithmique qu'au niveau des coûts, relativement à la très classique HF-FMM maintes fois déjà implémentée [21, 74, 11].

Par ailleurs, l'une des possibilités offertes par la SPW-FMM est la possibilité de permettre une FMM à un nombre quelconque de niveaux, sans contrainte liée à la fréquence. Toutefois, une limitation pratique est de considérer des tailles de cellules toujours au moins de l'ordre du triangle. Cette restriction n'a réellement d'implication que locale (FMM Adaptative, *cf* section 5.6). La suppression de la contrainte sur le nombre de niveaux de la HF-FMM autorise en particulier l'adaptation locale de maillage qui sera exposée seulement dans un cadre d'une démonstration de faisabilité (une étude des critères d'adaptation serait à développer en tant qu'extension naturelle).

## 5.3 Spécificités SPW-FMM

Une première spécificité de cette méthode est que l'approximation du noyau est constituée de deux intégrales discrétisées de manières différentes.

### 5.3.1 Sous-itérations FMM et traitement des directions privilégiées

La première intégrale s'inscrit assez facilement dans le même cadre que la HF-FMM, au traitement de la fonction de transfert près (Annexes D & E).

La seconde intégrale est déjà nettement moins habituelle, à cause d'une restriction quant à sa validité aux transferts de directions $z > 0$. Bien entendu, il convient de traiter l'ensemble des transferts comprenant les cas $z \leq 0$ ; ceci peut être pris en compte par décomposition de l'ensemble des directions possibles en 6 catégories, de directions principales $+x$, $-x$, $+y$, $-y$, $+z$ et $-z$ et 6 changements de repères associés proposant à chaque fois d'associer la direction principale à l'axe $z > 0$. Ceci permet ainsi de réduire le traitement effectif au cas $+z$, à la composition d'un changement de repère près.

Considérons une cellule de centre $C = (x_c, y_c, z_c)$, et $\mathcal{T}(C)$ l'ensemble des transferts issus de la cellule $C$. Nous avons déjà que pour tout $t \in \mathcal{T}(C)$, $C + t$ est une cellule distante de $C$ mais de parent proche du parent de $C + t$.

Définissons alors les ensembles $\mathcal{T}_X(C)$ pour $X \in \{+x, -x, +y, -y + z, -z\}$ comme suit

$$\mathcal{T}_{+x}(C) = \{(x,y,z) \in \mathcal{T}(C) : (|y| \leq +x) \text{ et } (|z| < +x)\}$$
$$\mathcal{T}_{-x}(C) = \{(x,y,z) \in \mathcal{T}(C) : (|y| \leq -x) \text{ et } (|z| < -x)\}$$
$$\mathcal{T}_{+y}(C) = \{(x,y,z) \in \mathcal{T}(C) : (|x| < +y) \text{ et } (|z| \leq +y)\}$$
$$\mathcal{T}_{-y}(C) = \{(x,y,z) \in \mathcal{T}(C) : (|x| < -y) \text{ et } (|z| \leq -y)\}$$
$$\mathcal{T}_{+z}(C) = \{(x,y,z) \in \mathcal{T}(C) : (|x| \leq +z \text{ et } |y| < +z) \text{ ou } (|x| = |y| = +z)\}$$
$$\mathcal{T}_{-z}(C) = \{(x,y,z) \in \mathcal{T}(C) : (|x| \leq -z \text{ et } |y| < -z) \text{ ou } (|x| = |y| = -z)\}$$

Nous avons alors bien

$$\mathcal{T}(C) = \bigcup_{X \in \{+x,-x,+y,-y+z,-z\}} \mathcal{T}_X(C).$$

Notons que le choix de cette partition est arbitraire et qu'il est légèrement déséquilibré en faveur des ensembles $\mathcal{T}_{\pm z}$ puisque qu'ils possèdent les diagonales de la forme $|x| = |y| = \pm z$.

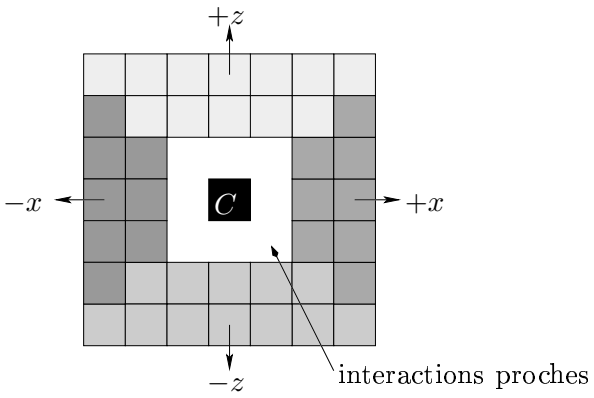



FIG. 5.2 – Répartition des transferts suivant leur direction principale.

En résumé, l'ensemble des changements de repères nécessite 7 sous-itérations de l'algorithme FMM afin d'effectuer un produit FMM complet d'un vecteur scalaire par la matrice FMM associée. Notons déjà que la taille réduite des discrétisations du terme évanescent permet toutefois des performances raisonnables (*cf* Chapitre 7) qui réduisent l'impact des sous-itérations sur l'algorithme global.

La décomposition par direction propose tout de même une optimisation, mineure, des transferts.

Cette optimisation vient de la remarque suivante : vu que les transferts sont effectués en plusieurs étapes pour chacune des directions privilégiées, pour une direction de transfert donnée $X$, il existe des cellules $C$ dont aucun des transferts possibles de cette direction ne va atteindre une autre cellule non vide même à un niveau supérieur (figure 5.3). Ainsi de telles cellules n'ont ni besoin d'être initialisées, ni d'être agrégées au niveau supérieur. Cette même remarque est applicable à l'étape d'intégration des cellules ne recevant aucun transfert et s'applique à partir des feuilles mais peut se propager plus haut tant qu'il existe des cellules vérifiant cette propriété. En supposant que toutes les cellules sont de pères proches, seules les cellules au bord pour une direction donnée ne seront pas utiles à l'étape d'initialisation (le bord opposé pour l'étape d'intégration).

Ce traitement par direction du terme évanescent impose quelques aménagements des étapes d'agrégation et de dispersion. En effet, il convient d'accorder les décalages des origines de façon à ce que le bon vecteur de translation soit associé à une cellule fille, relativement au repère

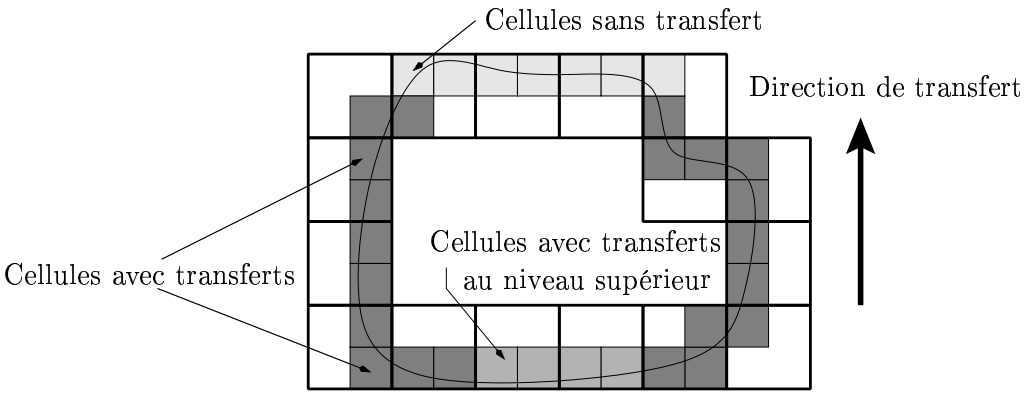



FIG. 5.3 – Optimisation des initialisations suivant une direction

associé à la direction de transfert. En d'autres termes, chaque cellule possède jusqu'à 8 fils numérotés de 0 à 7 suivant la direction principale $+z$. Les étapes d'agrégation et de dispersion requièrent un décalage de l'origine entre le centre d'expansion de cellule fille et celui de la cellule mère (*cf* proposition 22). Ce décalage fait partie des opérateurs d'inter/anter-polation qui sont, tout du moins partiellement, pré-calculés dans un repère donné de type $+z$. En l'état, la numérotation réelle d'un fils relativement à sa mère ne peut représenter le type précis d'opérateur à utiliser du fait de ce changement de repère.

$$\begin{array}{ll}
\text{direction } +x : & abc \rightarrow cab \\
\text{direction } -x : & abc \rightarrow c\bar{b}a \\
\text{direction } +y : & abc \rightarrow bca \\
\text{direction } -y : & abc \rightarrow b\bar{a}c \\
\text{direction } +z : & abc \rightarrow abc \\
\text{direction } -z : & abc \rightarrow a\bar{c}b
\end{array}$$

TAB. 5.1 – Table des fonctions $\mathcal{D}map$

Si l'on considère une interpolation dans le repère associé à une direction $\mathcal{D}$, pour tout fils $n$ écrit $abc_2$ (en base 2), il convient d'utiliser l'opérateur associé au fils $\mathcal{D}map(abc)$ pré-calculé dans le repère $+z$ (*cf* table 5.1 [3]), cette transformation provient directement du changement de repère sous-jascent.

Il en va de même, dans une moindre mesure, pour les étapes d'initialisation et d'intégration.

## 5.4 Boîtes strictes

Comme nous l'avons vu dans la section 5.1, la discrétisation de la surface au niveau de la méthode FMM passe souvent par des cas de triangles débordant légèrement des cellules auxquelles ils sont attachés (partitionnement par triangles et partitionnement par arêtes). Cependant, ce débordement ne devient plus négligeable lorsque l'on considère le terme évanescent de la formulation SPW-FMM avec la méthode à base de SVD (*cf* section 4.2.2). En effet, la décomposition SVD requiert un espace de paramètres précis tenant compte des transferts minimaux et maximaux. Alors tout débordement de l'espace des paramètres rend

[3] $\bar{x}$ représente le complémentaire du bit $x \in \{0,1\}$ dans $\{0,1\}$

invalide la méthode et demande une adaptation en conséquence. Le principal effet résultant est une explosion du nombre de points de discrétisation essentiellement due au fait que

$$\forall \epsilon < 1, \quad (z \to 0 \text{ et } e^{-\chi^2 z} < \epsilon) \Rightarrow \chi^2 \to +\infty.$$

Le paramétrage est choisi indentique pour toutes les cellules et est à considérer sur le pire des cas.

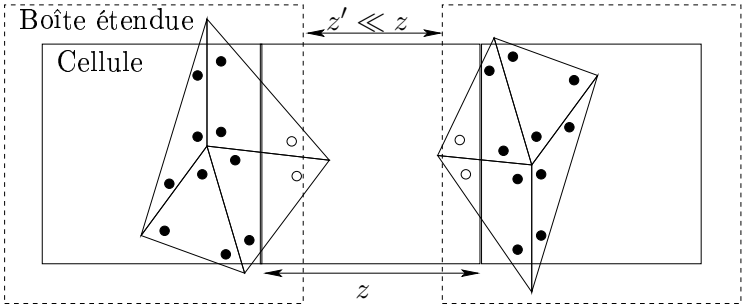



FIG. 5.4 – Débordement induisant un recalibrage de l'espace des paramètres

Au vu de ces restrictions, il est devenu indispensable de passer à un partitionnement en points de Gauss, certes plus complexe mais ne provoquant pas de telles pertes d'efficacité et de précision. La complexité liée à ce partitionnement apparaît tout d'abord dans les différentes étapes liées aux points de Gauss (initialisation, intégration), mais aussi dans la construction de la matrice d'interactions proches devant gérer les conflits possibles à cause des interactions partielles entre triangles. La méthode alors adoptée consiste en un calcul de la matrice d'interactions proches à partir de triangles complets et corrigée en éliminant les interactions redondantes avec les futurs interactions calculées par méthode FMM.

## 5.5 Pré/post-transferts

La discrétisation du terme evanescent dans la méthode à base de décomposition SVD est constituée de deux discrétisations différentes notées $(p, \phi)$ et $(\chi, \phi)$. Ces deux discrétisations sont utilisées à des étapes distinctes : $(p, \phi)$ lors des étapes d'initialisation, d'inter/anterpolation et d'intégration et $(\chi, \phi)$ durant la phase des transferts permettant ainsi un opérateur de transfert rapide et simple consistant en un produit terme à terme avec la fonction de transfert. La cohabitation de ces deux discrétisations implique des conversions pour passer de l'une à l'autre. Elles se produisent autour de l'étape de transferts et seront nommées pré-transfert et post-transfert. Les détails de cette étape sont décrits dans la section 4.2.2.

## 5.6 Maillage Adaptatif et FMM Adaptative

Un des buts de la formulation toutes fréquences est la possibilité de traiter des objets dont la taille est inférieure à une longueur d'onde. Ce cas se produit localement dans la méthode multipôle au niveau des petites cellules issues de la décomposition en octree avec éventuellement de nombreux niveaux.

La plupart des estimations du coût des méthodes FMM est donnée dans un cas optimal d'équi-distribution des points sur la surface. Toutefois, une surpopulation de points dans une

région de la surface engendre des coûts prohibitifs au niveau des étapes de calculs direct d'interactions (interactions proches). Ce déséquilibre se produit en particulier dans le cadre du raffinement local de maillages, en vue de mieux capter la solution du problème dans des régions critiques.

Usuellement, un palliatif est une augmentation du nombre de niveaux dans la décomposition de type octree. Cependant, une augmentation non mesurée du nombre de niveaux accroît considérablement le coût des autres étapes, inter/anter-polation et principalement des transferts, de plus en plus nombreux.

La FMM Adaptative est en fait une solution à ce même problème, avec encore une augmentation du nombre de niveaux, mais seulement localement autour des régions raffinées du maillage, créant ainsi des arbres multipôles non-uniformes. On ne peut, a priori, pas augmenter localement le nombre de niveaux, du fait des nombreuses relations complexes entre les cellules par les interactions proches et surtout distantes, nécessitant la disponibilité de l'information distante d'autres cellules non raffinées de niveaux identiques.

Cette proposition est basée sur la création de cellules *virtuelles* exclusivement dédiées à satisfaire les contraintes de type transfert (figure 5.5).

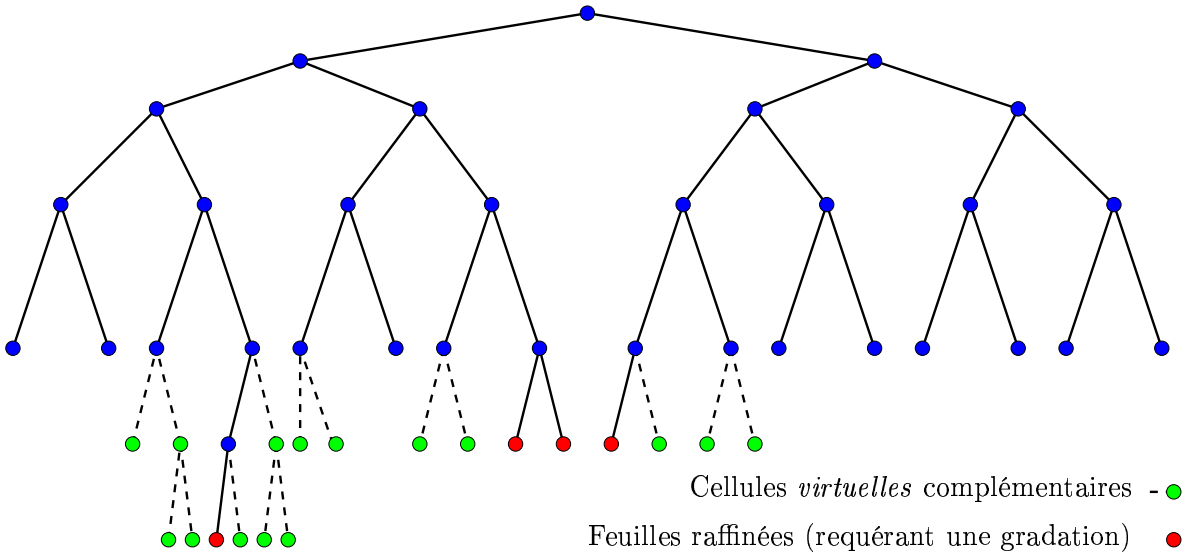



FIG. 5.5 – Construction de l'adaptation locale de l'arbre FMM (1D)

L'approche FMM s'applique alors avec des modifications mineures :
- Initialisation à partir des feuilles virtuelles au lieu des feuilles naturelles.
- Interactions proches au niveau le plus fin. Afin de limiter une explosion des coûts liés aux interactions proches avec les cellules raffinées, il convient de les traiter au niveau le plus bas possible (figures 5.6 et 5.7).

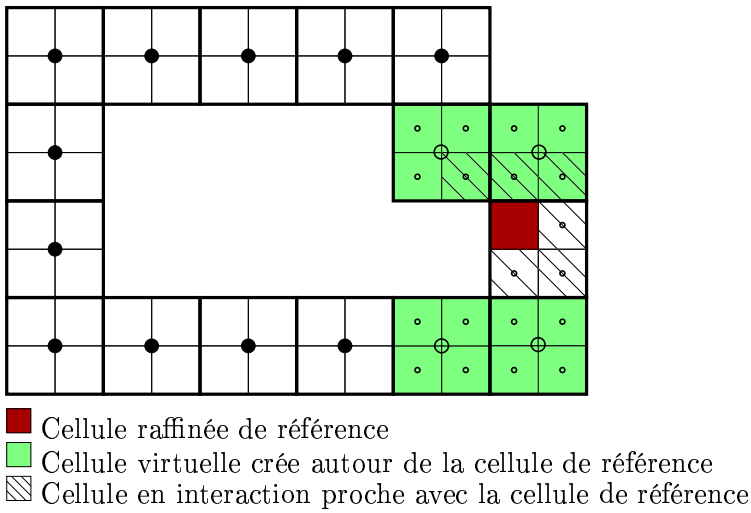



FIG. 5.6 – Interactions autour d'une cellule raffinée

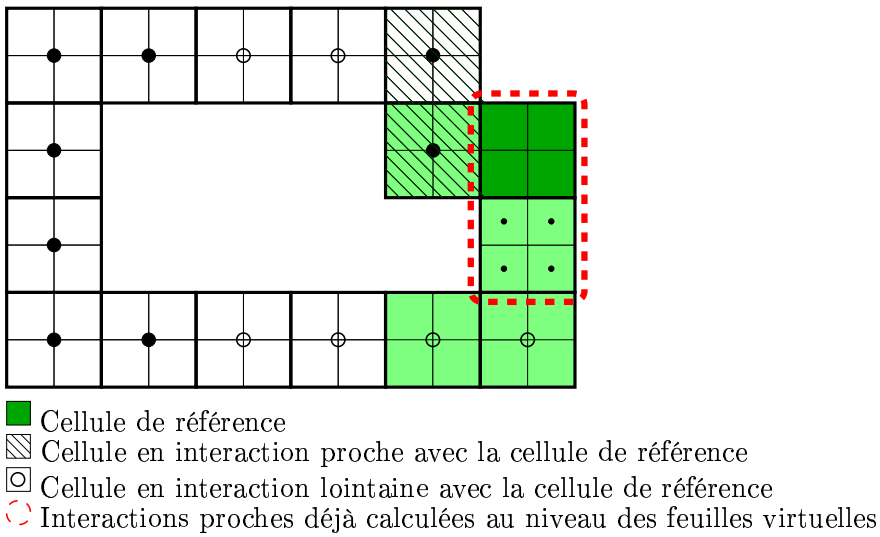



FIG. 5.7 – Interactions autour d'une cellule à descendance virtuelle

# Chapitre 6

# Parallélisme

**Sommaire**



Dans le cadre exposé dans le précédent chapitre, le parallélisme apparaît principalement en vue de traiter des problèmes de tailles raisonnables (plusieurs centaines de milliers d'inconnues). Notre objectif est de réaliser un code parallèle performant, sans toutefois y axer majoritairement nos efforts.

## 6.1 Paradigmes & Choix techniques

De nombreuses solutions techniques s'offrent à l'écriture de codes, et parallèles de surcroît.

Tout d'abord vient l'aspect conceptuel du langage à utiliser, qui va des langages fonctionnels aux langages impératifs en passant par les langages dits «logiques». A cela peut se greffer l'aspect objet de manière plus ou moins orthogonale.

Il est reconnu que les langages impératifs proposent de meilleures performances dans les codes de calcul lourds grâce à une syntaxe proche du fonctionnement des processeurs. Toutefois trop de proximité nuit à la portabilité et l'aisance accordée dans l'écriture (ex. les langages type assembleur). D'ailleurs, les algorithmes des codes de calculs manipulent à la fois des entités simples (nombres entiers, flottants...) et de plus complexes (vecteurs, matrices, arbres, formulations mathématiques, parallélisme...) : il convient donc de trouver un juste milieu. Ainsi, nos choix ont rapidement exclu le langage FORTRAN 77 (impossible de manipuler des concepts dépassant la notion de matrice, n'inclut pas d'allocation dynamique de la mémoire),

puis le langage C au profit du langage C++ (alliance entre le langage C et le concept objet). Le langage JAVA (impératif, très orienté objets) offre aussi de grandes possibilités de conceptualisation, mais souffre encore de quelques manques quant à l'optimisation du code généré (bien que les compilateurs natifs et «Just In Time» progressent à grands pas). Certes, le C++ n'est pas parfait (comme tout langage) mais il apparaît à mes yeux comme une alternative raisonnable et vivante aux langages bien plus restreints à des niches (Ada, Fortran $\geq$90...).

De même le champ des possibilités pour les codes parallèles s'étend de la parallélisation automatique (langages parallèles, OpenMP...), de la programmation distribuée (Active Objects (JAVA), C++//...) à la programmation par échanges de messages. La parallélisation automatique s'applique soit à des problèmes de nature particulière soit à des architectures particulières (OpenMP : machines à mémoire distribuée). La programmation distribuée offre de grandes possibilités grâce à de petites concessions (subdivision des tâches en tâches fondamentales le plus indépendantes possible...) et des performances raisonnables mais sujettes à des pénalités issues de contraintes complexes (Haute Disponibilité, équilibrage de charge, (trop de) transparence d'accès aux données...). Bien que cette dernière semble prometteuse, il apparaît encore que la proximité à l'architecture des machines offrent de bien meilleurs résultats. Ainsi, la communication par passage de messages - les communications sont gérées explicitement par envoi ou réception de segments de données éventuellement collectivement - offre des performances dans une section critique des codes parallèles, la communication. Les évolutions technologiques rapides sont encore en défaveur des réseaux derrière la mémoire et les processeurs. La minimisation des coûts spécifiques comme la latence ($\sim$ coût d'établissement d'une connexion) reste encore assez chère en matériel mais est déjà possible bien que souvent propriétaires (Myrinet, InfiniBand, Dolphin, Colony/Federation (IBM)). Cette diversité doit s'uniformiser afin de garantir une portabilité des codes développés et un maximum d'indépendance vis à vis du matériel : un standard encore reconnu aujourd'hui est MPI (et son évolution MPI-2). MPI propose de nombreux modèles de communication - sur lesquels nous reviendrons - non/-bloquant, a/-synchrone, point à point/collectif et est disponible à partir de la machine SMP (multiprocesseurs symétriques) aux grands calculateurs en passant par les grappes de calcul à mémoire distribuée.

Il convient de pondérer ces propos relativement à l'inertie existante au sein d'un projet et à la disponibilité et conformité aux standards des implémentations.

Les grappes de calcul sont une réponse économique à l'explosion du calcul numérique. C'est dans ce cadre que j'ai pu effectuer mes travaux sur la grappe de calcul du Laboratoire Jacques-Louis Lions. Cette grappe fonctionne sous Debian/GNU-Linux ayant évolué du Intel Pentium III-800Mhz (réseau Fast Ethernet) à AMD Opteron (réseau Giga Ethernet). C'est dans ce cadre qu'ont été effectués les développements décrits dans cette thèse.

**Remarque** *Par la suite, nous ferons la distinction entre* processeur*, unité de calcul physique, et* processus*, programme en exécution allouant des ressources de type processeur. Un* processeur *peut gérer de manière concurrente plusieurs processus par une méthode de partage du temps; un* processus *ne peut être partagé entre plusieurs processeurs à moins qu'il ne soit* multi-threadé *ce qui implique qu'il partage son contexte d'exécution avec un ou plusieurs processus légers, chacun pouvant posséder son propre flux d'instructions indépendant du processus parent.*

## 6.2 Optimisations Logicielles

Il est de nombreux domaines où les développements ont permis d'obtenir des implémentations d'algorithmes usuels très performants, essayant de tirer le maximum d'une architecture (réseau, mémoire, processeur...). Dans ces domaines, il n'est pas nécessaire de vouloir réinventer la roue à moins de proposer mieux.

En ce qui concerne l'algèbre linéaire, il existe les bibliothèques nommées BLAS et LAPACK offrant de nombreuses opérations matricielles de base, résolution de système linéaire et complexes décompositions (de type SVD par exemple) le plus souvent optimisée pour l'architecture de la machine (versions constructeurs, ATLAS (Automatically Tuned Linear Algebra Software)).

Traditionnellement basée sur une décomposition d'ordre $2^n$, les transformées de Fourier Rapides (Fast Fourier Transforms) sont un élément clef pour les méthodes multipôles présentées dans cette thèse. À la fois leur utilisation intensive nécessite des algorithmes rapides, mais la possibilité de réduire les contraintes sur la taille des échantillons transformés permet d'accéder à de forts gain en mémoire. En effet, des extensions aux échantillons de taille $n = \prod_i p_i^{\alpha^i}$ ($p_i \in \mathcal{P}$ nombres premiers) permet de trouver un bon compromis entre vitesse et coût mémoire. L'économie de mémoire réalisée vient du fait que toute discrétisation requiert un nombre de points minimal pour permettre une bonne approximation. Dans le cas des discrétisations uniformes sur lesquelles il est nécessaire d'effectuer des FFT (sur/sous-échantillonnage par exemple), ce nombre doit être majoré par $n$ répondant à la contrainte $n \in \{\prod_i p_i^{\alpha_i}$ avec $p_i \in \mathcal{P}' \subset \mathcal{P}\}$. Une FFT classique ($\mathcal{P}' = \{2\}$) peut alors provoquer jusqu'à un quasi doublement du nombre de points nécessaires pour répondre à cette contrainte. Toutefois, prendre l'ensemble $\mathcal{P}'$ trop grand induit une pénalité en terme de vitesse (figure 6.1). Nous avons défini notre équilibre à $\mathcal{P}' = \{2, 3, 5, 7\}$ grâce à l'utilisation de la bibliothèque FFTW.

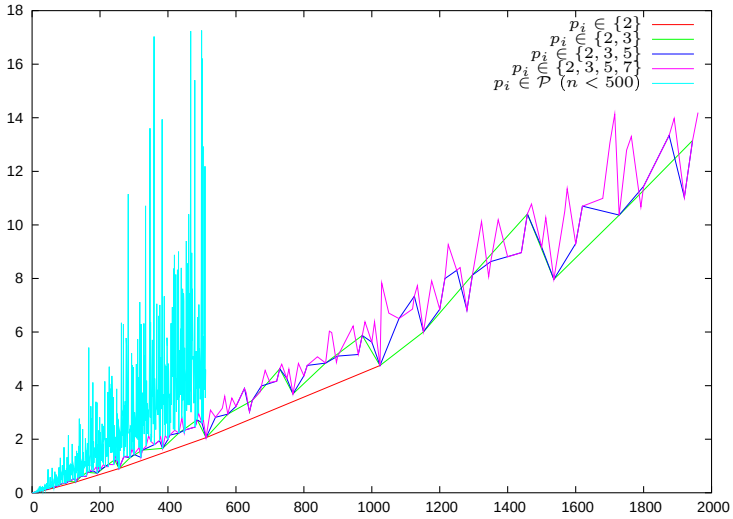



FIG. 6.1 – Coût des FFT suivant la taille des échantillons $n < 2000$

La partie communiquante étant critique dans les codes parallèles et peut-être encore plus dans les méthodes FMM (étape de transferts), un ensemble d'optimisations a été développé dans le cadre de la bibliothèque EasyMSG (section 6.6, annexe A).

L'utilisation de telles bibliothèques sera sous-entendue partout où l'efficacité est un point critique.

## 6.3 Octree Parallèle pour les Méthodes Multipôles

La structure arborescente des méthodes multipôles multiniveaux est l'un des points clefs de la méthode. Autour de cet arbre se déroulent les étapes du calcul allant de l'initialisation des feuilles, à leur intégration en passant par les interpolations, les transferts et les anterpolations. L'ordre et les relations entre toutes ces étapes ressortent directement de la structure de l'arbre.

Il convient donc de ne pas négliger cette structure, très classique en informatique et dans les méthodes numériques multi-échelles et multiniveaux.

### 6.3.1 Vocabulaire

Un arbre est un graphe orienté où chaque nœud peut avoir de multiples descendants ainsi que un ou zéro ascendant.

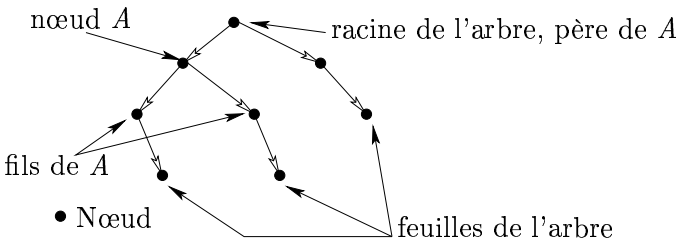



FIG. 6.2 – Arbre

- **Racine** : la *racine* est l'unique nœud sans ascendant (pourra aussi être nommée sommet de l'arbre) ;
- **Père** : le *père* du nœud $A$ est son ascendant (la racine n'a pas de père) ;
- **Fils** : le *fils* d'un nœud $A$ est l'un de ses descendants ;
- **Feuille** : une *feuille* est un nœud sans fils ;
- **Cluster** : un *cluster* est une agrégation de triangles ;
- **Cellule** : la *cellule* est la boîte associée à un nœud, contenant les triangles du cluster associés à ce même nœud.

Dans le cas d'un octree chaque nœud peut avoir jusqu'à $2^3 = 8$ fils associés aux 8 cellules (cubes) obtenues par la décomposition d'une cellule (cube) en découpant en 2 chaque arête.

### 6.3.2 Quelles sont nos contraintes ?

Comme cela a déjà été précisé, le code de calcul FMM sous-jacent est destiné à des machines parallèles à mémoire distribuée (faiblement SMP). Afin de maximiser le parallélisme, il convient tout d'abord de minimiser les communications. L'arbre ne peut y échapper. Pour cela nous voulons

- un arbre dynamique à la fois à hauteur non constante (les feuilles ne sont pas toutes au même niveau) sans nuire aux manipulations élémentaires comme les parcours d'un niveau ou de feuilles.

- une partition de l'arbre en relation optimisant au mieux les différentes étapes du calcul FMM ; ces différentes contraintes pouvant être conflictuelles, il conviendra de fixer des priorités,
- que chaque feuille (cellule terminale) ne soit pas partagée entre plusieurs processus,
- une construction/destruction/modification de cellule requérant un faible coût de communication en vue du support de l'adaptativité,
- une migration simple de cellules en vue d'un équilibrage de charge,
- réduire les dépendances inter-processus de cellules fréquemment en relation.

### 6.3.3 Quelques pistes...

#### Identification universelle

Afin de pouvoir localiser une cellule de manière unique et rapide dans un arbre distribué, nous proposons d'associer à chaque cellule un identificateur global unique. Cet identificateur permettra simplement de qualifier une cellule (lors des communications, des transferts...).

Cet identificateur global est formé de la succession de nombres à $N$ bits, où $N$ est la dimension de l'espace (chaque nombre à $N$ bits est associé à un des $2^N$ fils possibles d'une cellule).

Nous définissons l'identificateur de racine par *1*, cellule associée à la boîte englobante du maillage $B_0$. L'identificateur est construit comme l'association entre l'espace physique et les nombres binaires.

**Algorithme 1** Localisation de la cellule contenant un point $X$

1: Soit $B_0$ la boîte englobante de l'octree
2: Soit $X$ un point de $B_0$
3: Soient $l = 0$, $loc_0 = 1$ l'identificateur de la racine
4: **tant que** $B_l$ n'est pas associé à une feuille **faire**
5: Calculer la sous-boîte $B_{l+1}$ relative à $B_l$ telle que $X \in B_{l+1}$
6: Notons $s$ l'indice relatif de $B_{l+1}$ par rapport $B_l$
7: Chaque bit de $s$ correspond au type de décalage du centre de $B_{l+1}$ relativement au centre de $B_l$ pour une direction donnée
8: Nouvel identificateur partiel : $loc_{l+1} = loc_l.s$ (concaténation)
9: Niveau suivant : $l \leftarrow l + 1$
10: **fin tant que**
11: L'identificateur de la cellule $X$ est $loc_l$ au niveau $l$

**Remarque** *Cet algorithme est aussi employé pour l'insertion des triangles dans l'octree (ou points de Gauss suivant le type de partitionnement).*

Informatiquement parlant l'identificateur étant stocké sur un entier, cela impose une contrainte sur le nombre de niveaux. En effet, un entier 32 bits ne permet de représenter que 10 niveaux (+ la racine), ce qui est toutefois suffisant pour des cas raisonnables[4]. Cette limite n'est qu'une contrainte de représentation aisément dépassée grâce à la structure objet sous-jacente.

[4]Si nécessaire, l'extension aux nombres de 64 bits permet d'atteindre 21 niveaux (+ la racine)

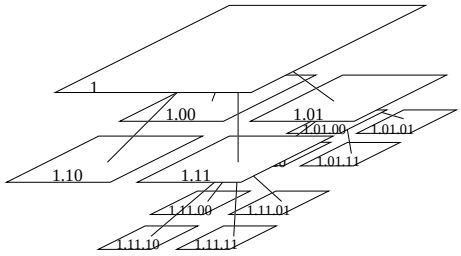



FIG. 6.3 – Indexation des feuilles dans un quad-tree ($N = 2$)

Toute cellule est alors identifiée de manière unique et globale et aucune ne sera partagée en écriture entre plusieurs processus ; toutefois une réplication en lecture (recopie) est admissible dans un but d'optimisation des communications.

### Partitionnements classiques

Le partitionnement de la surface et par conséquent de l'arbre est lié à de multiples contraintes, allant de la nécessité de grouper les cellules proches (étapes d'agrégation, de dispersion et de calcul de interactions proches) à la nécessité de minimiser les interactions lointaines inter-processus.

La contrainte de proximité peut sembler facilement satisfaite par l'utilisation d'un partitionnement de type «Space Filling Curves» [57] dans lequel entre les courbes de Morton (aussi qualifiées de hiérarchiques, figure 6.4) et de Hilbert (aussi nommée de Peano, figure 6.5) recouvrant récursivement l'espace. Leur intérêt est de fournir une indexation des cellules telles que la proximité de deux cellules le long de ces courbes favorise la proximité spatiale.

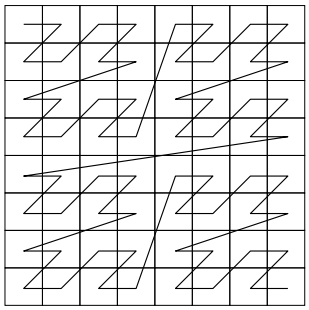


FIG. 6.4 – Courbe de Morton 2D à 3 niveaux

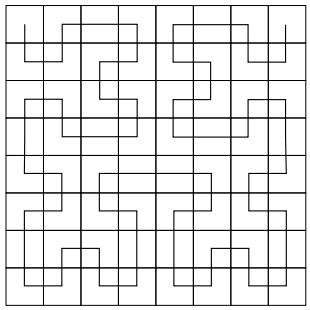


FIG. 6.5 – Courbe de Hilbert 2D à 3 niveaux

Notons à ce propos que l'agrégation ainsi issue de la courbe de Hilbert est bien meilleure lorsqu'il s'agit de remplir l'espace (la courbe en Z de Morton présente des sauts). Toutefois lorsqu'il s'agit d'utiliser ce procédé pour le partitionnement d'une surface dans un espace tridimensionnel, ces procédés ne peuvent plus garantir cette compacité des partitions (il est plus commode d'imaginer ce phénomène en 2D, avec le partitionnement d'une courbe dans le plan). Au cas par cas, certaines méthodes peuvent donner de bons résultats (ex : maillages allongés partitionnés suivant leurs directions privilégiées), mais la robustesse de l'algorithme de partitionnement requiert une adaptation automatique aux différentes situations, notamment lorsqu'il sera question d'adaptativité avec des arbres à hauteurs variables.

Il convient par ailleurs de ne pas oublier de partitionner l'intérieur de l'arbre. Les cellules internes peuvent avoir des descendants répartis sur différents processus. Une politique simple de répartition des cellules internes repose sur une élection, pour chaque cellule en partant des feuilles, du processus possédant déjà le nombre de fils locaux le plus grand (les conflits en cas d'égalité sont résolus par le « hasard »).

### Partitionnements de graphes

Par ailleurs, il convient de prendre en compte la seconde contrainte liée aux échanges distants. Cette contrainte est celle entrant en jeu lors de l'étape de transfert, qui est l'étape la moins scalable comme nous le verrons par la suite. Ce problème dépasse le simple partitionnement de maillages et nous préférons le placer au niveau du partitionnement de graphes. Dans ce cadre nous avons utilisé la bibliothèque MeTiS[47, 48] qui permet entre autres de partitionner des graphes à poids (aussi bien sur les sommets du graphe - assimilable à des coûts CPU - que sur les arêtes du graphe - assimilable à des coûts de communication -) sous des contraintes de minimalité d'arêtes coupées ou de poids total d'échanges. MeTiS peut être considéré comme un représentant de l'état de l'art (comme Zoltan, Scotch, Jostle, PARTY quoique certains ne présentent plus de signes de vitalité).

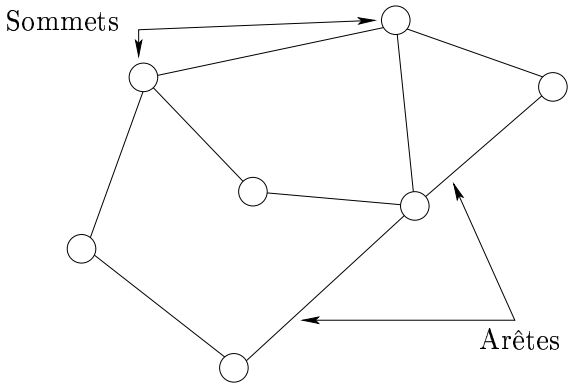



FIG. 6.6 – Exemple de graphe

L'idée est de représenter l'ensemble des contraintes dans un graphe. Les sommets de ce graphe seront les nœuds de l'arbre, des feuilles jusqu'au niveau le plus haut comportant des interactions. Les arêtes du graphe seront les relations de proximité. Au niveau de la définition des poids des sommets et des arêtes, plusieurs variantes sont possibles. La version ici présentée donne un poids égal au nombre de triangles ($\approx$ nombre de degrés de liberté) aux feuilles et zéro aux autres nœuds, permettant ainsi une équi-répartition des feuilles, et donc un partitionnement équitable pour les étapes d'initialisation et d'intégration. Le poids attribué à une arête dépend de la hauteur des cellules en relation dans l'arbre. Nous fixons un poids de 1 au niveau le plus bas, puis le poids associé à chaque autre niveau est donné comme $fc$ fois celui du niveau inférieur. Il faut ensuite lier les niveaux entre eux. Ces liaisons seront des arêtes liant une cellule à son père de poids $fp$ fois le poids associé au niveau du père. Une conséquence de ce poids est une contrainte de minimisation des inter/anter-polations non locales. Bien que tous les contraintes ne portent que sur les interactions proches, les interactions lointaines sont prises en compte en se rappelant qu'elles sont effectuées entre cellules de parents proches (ce moyen détourné permet par ailleurs de réduire considérablement la taille

| Objet | CPUs | Décomposition par Octree | | | | Partitionnement de graphes | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | Volume | Feuilles | Transf | Up | Volume | Feuilles | Transf | Up |
| Sphere | 8 | 38196 | 95.6s | 406.0s | 60.3s | 38223 | 95.6s | 397.8s | 60.3s |
| | 10 | 50949 | 95.3s | 525.2s | 66.6s | 46500 | 95.6s | 482.9s | 68.1s |
| Falcon | 8 | 35075 | 69.4s | 418.0s | 10.8s | 23107 | 69.5s | 193.5s | 11.0s |
| | 10 | 30290 | 69.5s | 318.6s | 11.3s | 21180 | 69.4s | 226.1s | 13.6s |
| RFalcon | 8 | 31608 | 51.1s | 268.3s | 17.0s | 24770 | 50.6s | 180.6s | 18.2s |
| | 10 | 29351 | 50.6s | 316.1s | 18.4s | 21798 | 50.9s | 216.8s | 19.6s |

- *Volume* : Volume en Ko par processeur de données à transférer pour une étape de transfert
- *Feuilles* : Temps de calcul aux feuilles (initialisation + intégration) pour 4 itérations FMM
- *Transf* : Temps d'une étape de tansfert
- *Up* : Temps d'une étape d'aggrégation (tous niveaux)

TAB. 6.1 – Coûts FMM liés au partitionnement

des graphes manipulés).

Ainsi, un partitionnement optimisant la proximité à tous les niveaux repose sur un partitionnement de graphes sous contraintes d'équilibrage des poids (feuilles) et de minimisation des arêtes coupées (communication de transfert et d'inter/anter-polation). Ce point le distingue des simples partitionnements de surfaces ne travaillant qu'au niveau des triangles, assimilable aux feuilles dans notre cas. La minimisation du nombre d'arêtes coupées ne correspond pas tout à fait à la factorisation des transferts distants, mais permet déjà d'obtenir de bons résultats même pour des maillages complexes où les gains sont assez considérables par rapport à une décomposition de l'espace par des courbes type Morton ou Hilbert.

La qualité du partitionnement en fonction de $fc$ et $fp$ n'a été que sommairement étudiée ; nous retiendrons dans nos tests $fc = 2$ et $fp = 2$.

La table 6.1 représente les coûts de différentes étapes d'un calcul FMM à 9 niveaux sur 3 différents objets :

- *Sphere* : Sphere unité de 400 000 triangles ($27\lambda$)
- *Falcon* : Falcon[5] de 450 000 triangles ($37\lambda$)
- *RFalcon* : Falcon de 450 000 triangles ayant subit une rotation de 45° suivant les axes $x$ et $y$ ($51\lambda$)

Les tailles des graphes associés sont de 300 000 sommets et 3 000 000 arêtes pour la sphère et 50 000 sommets et 600 000 arêtes pour les Falcon.

Le cas de la sphère est significatif de la grande spécificité de la décomposition par octree pour des maillages présentant des symétries pour des partitionnements de taille relative à la symétrie (puissance de 2 dans le cas de la sphère). Mais dès que les géométries deviennent plus complexes ou que le nombre de partitions ne suit plus des règles strictes (ou même partition non équilibrée pour l'équilibrage de charge), le partitionnement par graphes devient très performant (géométrie quelconque, partition de tailles quelconques).

Les illustrations (figures 6.7, 6.8, 6.9, 6.10) permettent d'apprécier visuellement la dispari-

[5]Maillage fourni par Dassault Aviation

| Objet | CPUs | Volume | Transf | Up | Global |
|---|---|---|---|---|---|
| Sphere | 8 | +0.0% | -0.2% | +0.0% | |
| | 10 | -8.8% | -8.1% | +2.2% | |
| Falcon | 8 | -34.1% | -53.7% | +1.9% | -32.0% |
| | 10 | -30.1% | -29.0% | +20.0% | -29.9% |
| RFalcon | 8 | -21.6% | -32.7% | +7.1% | -23.0% |
| | 10 | -25.7% | -31.4% | +6.5% | -23.5% |

TAB. 6.2 – Performance obtenue par le partionnement de graphes relativement à la décomposition par octree

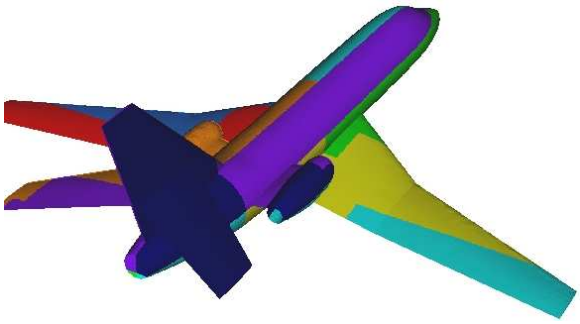


FIG. 6.7 – Décomposition en octree

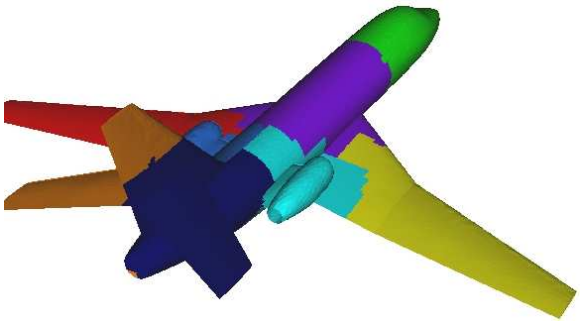


FIG. 6.8 – Partitionnement de graphes

Partitionnement en 8 du maillage *Falcon*

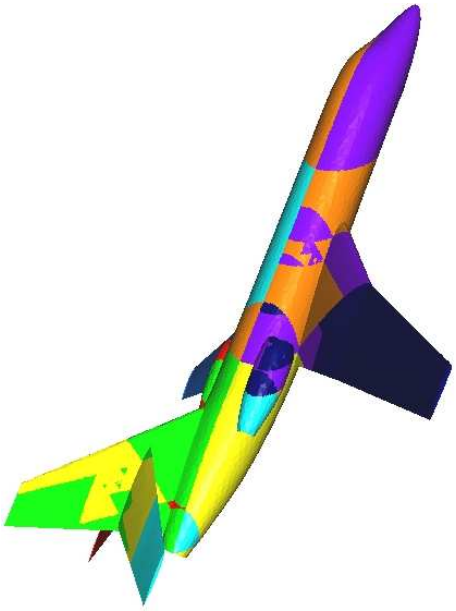


FIG. 6.9 – Décomposition en octree

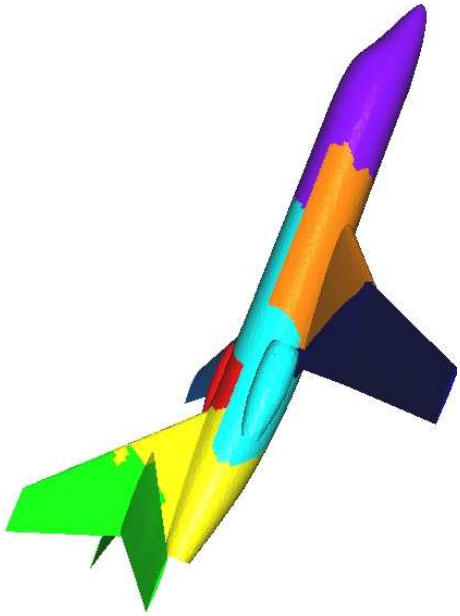


FIG. 6.10 – Partitionnement de graphes

Partitionnement en 8 du maillage *RFalcon*

tion des artefacts de partitionnement lié à la décomposition par octree. Ceux-ci auraient déjà pu être réduits en autorisant un partitionnement moins équitable des cellules (notamment des feuilles) ou bien en limitant la profondeur de décomposition dans l'arbre, ce qui dans les cas présentés n'engendre pas d'amélioration notables. Toutefois notons que ce ne sont pas ces artefacts qui rendent compte de la qualité d'un partitionnement, puisque malgré tous les artefacts visibles le cas *RFalcon* est bien meilleur que *Falcon* (cf table 6.1).

## 6.4 Interactions proches

Le calcul des interactions proches est un point critique non pas pour le temps de traitement nécessaire ni même pour la complexité algorithmique, mais pour le volume de données pouvant y être associé. Sont traitées par interactions proches toutes les cellules non traitées par la FMM, c'est à dire d'autant plus que le nombre de niveaux est faible ou que le nombre d'inconnues est grand.

À ce problème de volume de données, trois alternatives sont possibles :

1. tout stocker en mémoire central (RAM), dans la mesure où cela est encore possible ;
2. recalculer au vol les données ;
3. tout stocker dans des espaces mémoire plus grands mais plus lents tels que les disques durs.

Chacune de ces trois possibilités offrent avantages et inconvénients, respectivement :

1. vitesse contre limitation matérielle ;
2. pas de limitation de taille contre surcoût du recalcul (à chaque fois) ;

3. grande capacité contre relativement lenteur des supports.

Ces trois solutions ont donc été développée autour d'une même structure : les matrices distribuées creuses par blocs avec délégation des méthodes d'accès aux blocs (Distributed Block Sparse Row Matrices). Ce type d'objet propose des algorithmes parallèles d'opérations de type produit matrice-vecteur.

L'un des points forts de ces algorithmes est la performance parallèle par une optimisation des mémoires au différents niveaux :

- au niveau du processeur, le traitement par blocs des produits matrice-vecteur augmente les performances par une bonne gestion de la mémoire cache,
- au niveau des disques, un accès linéaire autorisant des techniques de prefectch (prélecture des données et utilisation des caches disque),
- au niveau du réseau, par un recouvrement des communications et des calculs par blocs.

Une illustration des performances de ce type d'algorithme est donné dans la table 6.3 page 87, où l'on peut constater des performances au delà de 100% !

## 6.5 Interactions lointaines

Le traitement des interactions lointaines est composé de plusieurs parties. Il y a certes le traitement à chaque itération, appliquant les fonctions de transferts (produit terme à terme) aux différents niveaux, mais une étape préalable est la construction des interactions (liste des cellules en interaction) et des fonctions de transfert (*cf* chapitre 2 pour le principe).

Un choix de notre implémentation est le pré-calcul et le stockage des fonctions de transfert. Le pré-calcul suit un algorithme spécifique à la méthode (*cf* chapitres 3 et 4 pour les détails propres aux formulations) et est réalisé en début de calcul (une mise à jour des fonctions de transferts est parfois nécessaire lors de méthodes adaptatives). Leur stockage peut se réaliser soit en mémoire (fort coûteux, dupliqué sur chaque processus et utilisé seulement dans l'étape de transfert), soit sur disque (en attendant un chargement au niveau de l'étape de transfert). Ce stockage peut être accompagné d'astuces telles que l'utilisation des symétries et redondances de l'opérateur de transfert (détaillées et implémentées dans [21, 11, 74]).

Nous avons privilégié le stockage. Toutefois il convient de noter les symétries évidentes de l'opérateur de transfert du terme évanescent de la SPW-FMM (section 5.3.1) réduit déjà d'un facteur moyen de 6 le coût de leur stockage.

Lors d'une implémentation parallèle de l'étape de transfert, l'algorithme peut être réécrit afin de réduire considérablement le coût en communication. En effet, l'algorithme s'écrit (notation du chapitre 2)

$$G_k^r = \sum_{P_t \in \mathcal{T}(P_r)} F_k^t \, T_k(C_r - C_t).$$

ce qui d'un point de vue algorithmique revient à

**Algorithme 2** Transfert : mode *Pull*

1: $G_k^r$ transferts partiels associés à $P_r$ initialisés à 0
2: **pour tout** $P_r$ cellule **faire**
3: **pour tout** $P_t \in \mathcal{T}(P_r)$ **faire**
**Prérequis:** Connaître $F_k^t$
4: $G_k^r \leftarrow G_k^r + F_k^t\, T_k(C_r - C_t)$
5: **fin pour**
6: **fin pour**

Or le prérequis *Connaître* $F_k^t$ suppose dans le cas parallèle de le transmettre s'il n'est pas connu localement, ce qui implique autant de transferts que de cellules distantes.

L'algorithme 2 peut être réécrit

**Algorithme 3** Transfert : mode *Push*

1: $G_k^r$ transferts partiels associé à $P_r$ initialisés à 0
2: **pour tout** $P_t$ cellule **faire**
3: **pour tout** processus $cpu_p$ possédant une cellule $P_r$ tel que $P_t \in \mathcal{T}(P_r)$ **faire**
**Prérequis:** Connaître $F_k^t$
4: **pour tout** $P_r$ tel que $P_t \in \mathcal{T}(P_r)$ **faire**
5: $G_k^r \leftarrow G_k^r + F_k^t\, T_k(C_r - C_t)$
6: **fin pour**
7: **fin pour**
8: **fin pour**

Étant donné que le prérequis *Connaître* $F_k^t$ devient factorisé à chaque processus, la quantité de données échangées est réduite au strict nécessaire, c'est-à-dire une seule transmission par $F_k^t$.

## 6.6 Optimisation dynamique des communications

Pour tout programme parallèle, les communications par le réseau sont pénalisantes car bien plus lente, à la fois en latence et en débit, que des accès à la mémoire locale de type RAM. C'est pourquoi, il convient de toujours leur accorder une attention particulière.

Deux points de vue sont envisageables. On peut considérer que les communications sont effectuées dans un ordre invariant prédéterminé, ou bien que les communications suivent un ordre déterminé par l'évolution de l'état du calcul (dont les moyens de communication comme le réseau). Dans un environnement de type grappe de calcul hétérogène et partagée entre de multiples utilisateurs, nous avons privilégié le point du vue des communications adaptatives qui tentent de s'extraire des perturbations probables de l'environnement (dont le réseau dans le cadre des communications).

Parallèlement, il existe deux types de communication

- bloquante : les requêtes de communication se terminent lorsque la communication est effectivement accomplie ;
- non-bloquante : les requêtes de communication se terminent immédiatement et les communications effectives s'accomplissent en tâche de fond (cela requiert des tests de complétude).

| PEs | local interactions | non-local interactions | timing | efficiency |
|---|---|---|---|---|
| CSR | - | - | 6.60s | - |
| 1 | $1150 \times 10^3$ | 0 | 4.08s | 100% |
| 2 | $577 \times 10^3$ | 2300 | 2.02s | 101% |
| 4 | $287 \times 10^3$ | 2300 | 1.00s | 102% |
| 8 | $143 \times 10^3$ | 1700 | 0.49s | 104% |
| 16 | $71 \times 10^3$ | 2000 | 0.31s | 82%* |

– Les *interactions* sont exprimées en termes de blocs.
– * : Pénalité due aux accès concurrents aux périphériques.
– Calculs effectués sur PentiumIII/800Mhz en réseau Ethernet 100Mb

TAB. 6.3 – Interactions proches (stockage «Block CSR Matrix» de $16 \cdot 10^6$ valeurs $\mathbb{C}$)

En partant du principe que les communications de type réseau consomment peu de ressources de type processeur mais relativement beaucoup de temps à cause des protocoles et des latences et débits du réseau, il est important d'éviter de les attendre en ne faisant rien. Ainsi, nous utiliserons principalement des communications non bloquantes et nous tenterons de minimiser le nombre de messages par une agrégation des messages et de maximiser le recouvrement des communications par des calculs, c'est-à-dire que des calculs seront réalisés en attendant la complétion des communications.

Dans cette optique la bibliothèque EasyMSG a été développée (annexe A).

La partie comportant l'agrégation de messages consiste tout simplement en une concaténation de plusieurs petits messages jusqu'à obtenir une taille raisonnable (dépendante du type de réseau) puis à envoyer le tout en une seule fois. Ce principe fort simple et très classique suppose juste une étape préparatoire (concaténation) ainsi qu'une étape de reconstruction des paquets indépendants d'information après réception. L'usage de la bibliothèque EasyMSG apporte principalement un confort d'utilisation (Ses étapes sont totalement transparentes grâce à l'encapsulation que permet le langage C++).

En ce qui concerne un recouvrement de type calcul/communication, il existe assez peu de possibilités de solutions dans le cadre de programmes SPMD[6] qui, par nature, comprend l'ensemble des programmes parallèles composés d'un flux d'instructions unique appliqué à un ensemble de données réparties sur l'ensemble des processus de calculs (l'avancement de le flux d'instructions peut cependant être indépendant dans chaque processus composant une exécution parallèle). En général, cela se réduit à un flot d'instructions de calcul, entrecoupé de requêtes non-bloquantes de communication et de points de contrôle validant la fin des requêtes précédemment émises. Le recours à la programmation *multithread* est aussi envisageable mais peut poser des problèmes techniques du fait du partage du contexte d'exécution entre les processus légers (cela dépend principalement de la bibliothèque de communications, où peu d'implémentations thread-safe sont disponibles, en particulier concernant MPI).

[6]SPMD : Single Program Multiple data, opposable à Single Instruction Multiple Data, décrivant les machines vectorielles.

## 6.7 Équilibrage de charge dynamique

Toujours dans notre cadre d'étude d'une grappe de calcul interconnectée par un réseau de type Ethernet, il apparaît important de pouvoir s'accommoder de l'hétérogénéité du matériel et des variations possibles des charges de calcul associées aux processeurs, durant un long calcul. En effet, nous avons constaté que la variation de charge de traitement pouvait induire une variation dans les performances du matériel, en particulier sur certaines architecture matériel à mémoire partagée. Cette notion d'équilibrage de charge dynamique trouve aussi des applications dans le traitement de l'adaptativité avec raffinement de maillage, ayant tendance à déséquilibrer la répartition des charges de calcul parmi les processeurs.

Nos motivations sont résumées en

- de la difficulté d'une estimation statique de la charge induite par un calcul sur chacun des processeurs en jeu :
  Algorithmes, communications...
- de la gestion adaptée d'un environnement de calcul hétérogène :
  Processeurs, réseau, compétition d'accès aux périphériques...
- de la prise en compte en cours de calcul de la variation locale de charge liée à l'adaptation par raffinement de maillages.

Nous pouvons illustrer les performances de l'équilibrage de charge dynamique par un calcul en environnement hétérogène. Le calcul illustré par les figures 6.12 et 6.11 se déroule sur deux types de processeurs : des processeurs A qualifiés de rapides et des processeurs B approximativement de vitesse moitié de ceux de type A. Les processeurs de type A sont montés par deux pour former des machines SMP bi-processeurs.

Une première catégorie de processus de calcul (02,03,08) s'exécutera sur les machines SMP à raison d'un seul processus par machine. Une seconde catégorie (04,05,06,07) est composée de processus de calcul s'exécutant sur les machines SMP à raison de 2 processus sur 2 processeurs (ce qui entraîne une compétition d'accès aux périphériques dont le réseau et la mémoire). Enfin les processus (00,01) s'exécuteront sur les processeurs B.

Le calcul à proprement parlé porte sur l'éclairage d'une sphère unité par une onde électromagnétique 3GHz. Le maillage utilisé comporte 300 000 degrés de liberté (arêtes) et la FMM utilise 8 niveaux.

La mise en oeuvre du procédé repose principalement sur le choix d'estimateurs :

- un estimateur *a priori*, permettant de définir un coût dans une unité commune de la charge attribuée. Cet estimateur doit être construit sur des coûts les plus fondamentaux, idéalement le coût exact en flops (floatting point operation). Dans notre cas FMM, nous pouvons l'estimer comme le nombre de degrés de liberté ou bien le nombre de feuilles ou cellules attribuées ;
- un estimateur *a posteriori*. Cet estimateur permettra de définir une vitesse locale à chaque processeur de traitement de la charge estimée par l'estimateur *a priori*. Après normalisation suivant les vitesses de traitement, la répartition des données se fera uniformément suivant cette mesure. L'estimateur *a posteriori* est construit comme le temps passé au cours d'une fraction du calcul. Afin de pouvoir mieux prendre en compte les différentes étapes du calcul FMM, nous avons défini cette fraction du calcul en nombre d'itérations. L'adaptation peut alors être effectuée périodiquement ou suivant un seuil de d'équilibre.

Une fois ces concepts définis, l'équilibrage de charge consiste simplement en un re-partitionnement des données conformément aux données receuillies par l'estimateur *a posteriori*. Ce

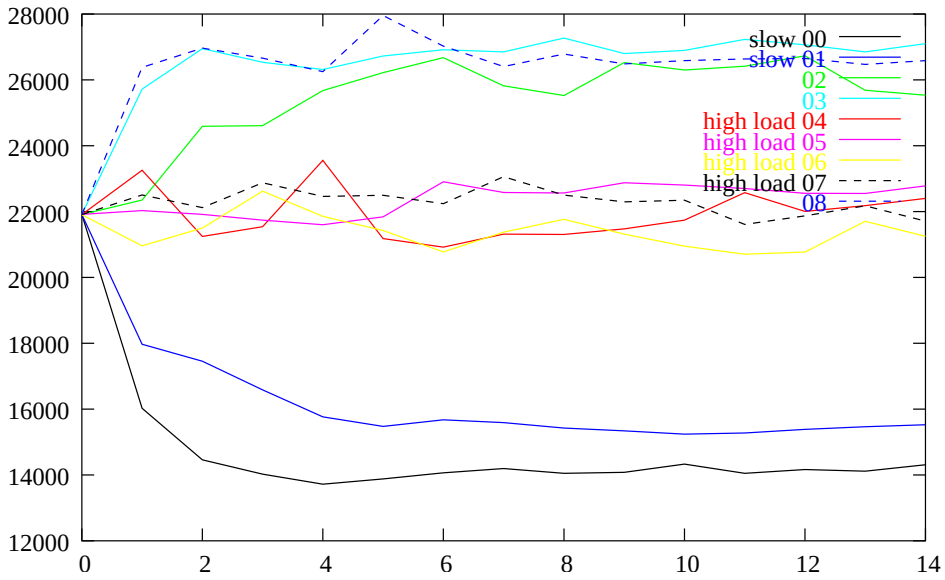



FIG. 6.11 – Répartition au cours du temps de la charge des processeurs.

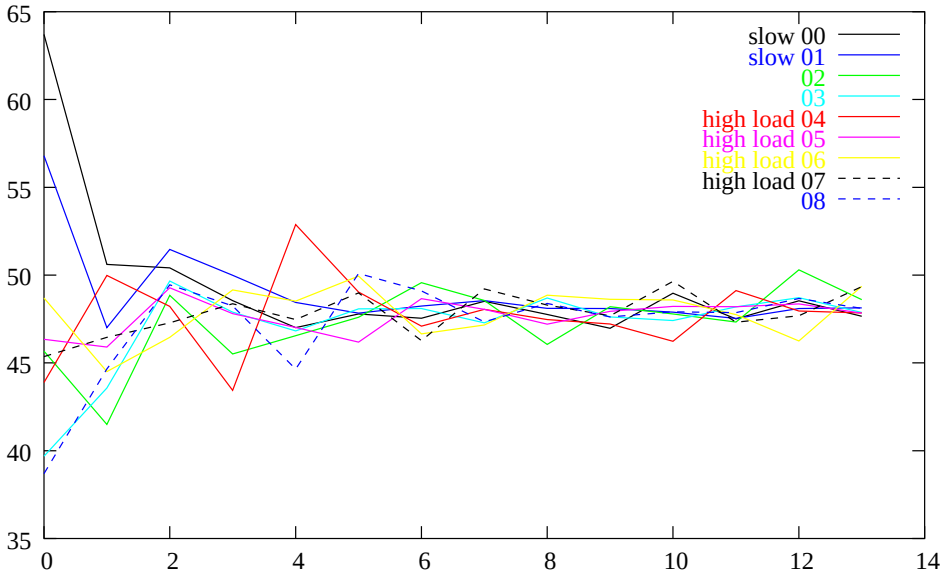



FIG. 6.12 – Temps par processeur d'une itération au fil de l'équilibrage (inclut le coût du repartitionnement).

re-partitionnement se doit de réduire les migrations de données. En effet, à la suite de chaque migration il convient de reconstruire certaines informations, qui dans le cadre de la FMM consistent en la matrice d'interactions proches, les fonctions de transferts, ainsi qu'une mise à jour des structures parallèles comme les vecteurs, les données de synchronisation et de distribution. Ce re-partitionnement s'appuie sur les algorithmes de partitionnement décrits dans la section 6.3.3. La minimisation des migrations de données n'est pas assurée par l'algorithme de type octree, mais induit de faibles migrations pour de faibles variations de vitesse relative (grâce au partitionnement par linéarisation des cellules à répartir). Les autres algorithmes de type graphe s'appuient de nouveau sur la bibliothèque METIS en terme de re-partionnement de graphes.

## 6.8 Calcul itératif sans FMM

Afin de fournir des solutions comparatives pour des cas non abordables par la méthode HF-FMM, le code de calcul propose une résolution par un solveur itératif mais dont la matrice est totalement assemblée et donc pleine. Bien entendu, ce type de matrice requiert un important espace mémoire, et c'est pour cela que l'enregistrement sur disque de la matrice est nécessaire. Toutefois, cette fonctionnalité s'appuie sur le code parallèle dans son ensemble en prenant comme matrice d'interactions proches (*cf* section 6.4) toute la matrice et la partie FMM étant réduite au vide. Il n'est plus raisonnable d'utiliser cette possibilité dès que le nombre d'inconnues devient de l'ordre de 30 000.

## 6.9 Performances générales

L'étude des performances d'un code parallèle traitant de la FMM dépend beaucoup de l'environnement de calcul utilisé : tant du matériel que des logiciels employés (compilateurs, bibliothèques...). Par leur structure algorithmique, les méthodes FMM sont à la fois de bons candidats à la parallélisation, et très exigeants quant au matériel.

En effet, les méthodes FMM se composent de trois parties ayant des caractéristiques spécifiques :

- opération aux feuilles, étapes d'initialisation & d'intégration
  : totalement scalable
- opération inter-niveaux, agrégation & dispersion
  : relativement peu de communications
- étape de transfert
  : énormément de communications, faible scalabilité

Regardons les performances obtenues autour d'une grappe de calcul de machine Bi-processeurs en réseau Ethernet (dont le type sera mentionné). Le coût des interactions proches n'est pas inclus dans ces résultats puisque leur stockage en mémoire principale rend impossible l'usage d'un faible nombre de processeurs, et que le stockage sur disque dégrade fortement les performances dès que plus d'un processus utilise intensivement le disque.

**Remarque** *Nous définissons l'*efficacité *comme le complément de la perte relative de puissance de calcul induit par l'accroissement du nombre de processeurs.*

| PEs | 1 Itération | Transfert | Hors transfert | Efficacité globale |
|---|---|---|---|---|
| 1 | 460s | - | - | - |
| 2 | 270s | 63% | 98% | 85.3% |
| 4 | 143s | 54% | 98% | 80.6% |
| 8 | 80s | 41% | 97% | 71.6% |
| 12* | 80s | 21% | 86% | 47.4% |
| 16* | 63s | 20% | 82% | 45.6% |

TAB. 6.4 – Efficacité - Sphère unité 300K inconnues à 2GHz - Pentium III 800MHz

* : Pénalités induites par des accès concurrents aux périphériques (utilisation de plusieurs processeurs d'une machine SMP)

| PEs | 1 Itération | Transfert | Hors transfert | Efficacité globale |
|---|---|---|---|---|
| 2 | 141s | - | - | - |
| 4 | 73s | 83% | 99% | 96.4% |
| 8 | 40s | 63% | 97% | 89.1% |
| 12* | 33s | 40% | 85% | 71.9% |
| 16* | 29s | 25% | 80% | 59.0% |

TAB. 6.5 – Efficacité - Sphère unité 300K inconnues à 1GHz - Pentium III 800MHz

| PEs | 1 Itération | Transfert | Hors transfert | Efficacité globale |
|---|---|---|---|---|
| 2 | 564s | - | - | - |
| 4 | 302s | 85% | 99% | 93.5% |
| 8 | 164s | 71% | 98% | 96.2% |
| 12* | 145s | 45% | 88% | 65.0% |
| 16* | 115s | 41% | 85% | 61,2% |

TAB. 6.6 – Efficacité - Sphère unité 600K inconnues à 3GHz - Pentium III 800MHz

Au regard des ces résultats, nous pouvons noter que la taille locale par processus du problème à une forte incidence sur les performances (*cf* figure 6.14), notamment au niveau de l'étape de transfert. Ainsi quand le nombre de processeurs croît, nous pouvons noter une forte dégradation des performances.

En l'état, je sais que quelques optimisations devraient être effectuées, notamment au niveau

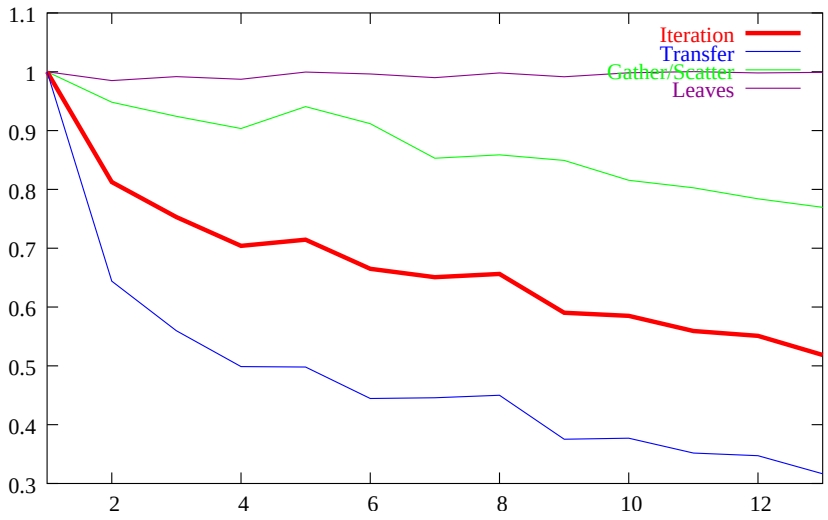



FIG. 6.13 – Efficacité des composants FMM - Sphère 300K inconnues à 3GHz - Pentium IV 2.4GHz

du traitement des transferts proche de la racine, souvent peu nombreux mais très volumineux. Par ailleurs, l'architecture matériel induit quelques pénalités en partie au niveau des surcoûts issus de la compétition aux périphériques (principalement la mémoire partagée), et au niveau du réseau de type Ethernet, pas forcément le mieux adaptée au traitement de gros volume de données issus de l'étape de transfert.

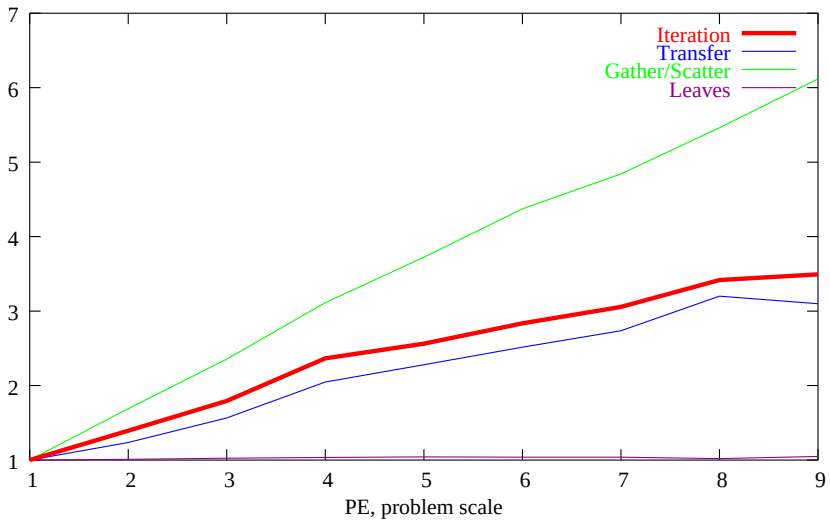



FIG. 6.14 – Efficacité à charge constante par processeur (50K triangles) - Sphère à 3GHz - Pentium IV 2.4GHz

# Chapitre 7

# Résultats numériques

[caca]

**Sommaire**



## 7.1 Tests numériques

### 7.1.1 Comparaison des discrétisations

Dans cette section, nous allons comparer les discrétisations nécessaires aux méthodes HF-FMM et S*table* P*lane* W*ave*-FMM (tables 7.1, 7.2 and 7.3).

Le terme propagatif de la SPW-FMM sera discrétisé suivant la méthode donnée dans la section 4.2.1 qui permet de réutiliser le même type d'opérateurs et de discrétisations tout comme l'expansion en ondes planes (HF-FMM). En régime basse fréquence, le nombre de points de quadrature tend vers 0 lorsque $\kappa z$ tend vers 0, ce qui réduira d'autant le coût calculatoire de ce terme.

Le terme évanescent de la SPW-FMM sera décrit comme $(n_a : n_b) \times n_c$, signifiant que $n_a$ points de quadrature ont été utilisé dans l'étape Outer-to-Inner, $n_b$ points de discrétisation dans la décomposition SVD, *i.e.* , $2\,n_b$ fonctions $s_p$, $u_p$, $v_p$ sont utilisées dans les étapes Outer-to-Outer et Inner-to-Inner, et $n_c$ points de discrétisation suivant la variable $\phi$. Le stockage alors requis pour la discrétisation d'une telle cellule est de $\max(n_a, 2\,n_b) \times n_c$.

Le symbole «-» signifie que la valeur n'est pas disponible à cause d'un problème de stabilité de l'expansion de type HF-FMM ou bien que la précision $\epsilon$ désirée ne peut être atteinte sur

l'approximation du noyau. Le paramètre $\epsilon > 0$ est donné comme l'erreur absolue maximale attendue sur l'approximation des intégrants des différents termes FMM pour un potentiel de 1, c'est à dire que $\epsilon$ correspond à l'erreur relative sur les intégrants dans le cas de potentiels non nuls (aussi bien les termes propagatif, évanescent et HF-FMM). Il est communément estimé que la méthode HF-FMM devient instable lorsque la taille de la cellule, $\rho$, est plus petit que $\lambda/2$, avec $\lambda = 2\pi/\kappa$ la longueur d'onde.

Une première remarque sera que la méthode SPW-FMM permet d'atteindre n'importe quel nombre de niveaux à toutes fréquences.

Conformément au théorème 15 et au corollaire 16, le nombre de points de discrétisation suivant $\phi$ dans le terme évanescent atteint une constante lorsque $z \ll 1/\kappa$.

| | Stable Plane Wave FMM | | HF-FMM |
|---|---|---|---|
| | Terme propagatif | Terme évanescent | |
| $\rho = 1$ | $3 \times 5$ | $(12:4) \times 25$ | - |
| $\rho = \frac{1}{2}$ | $3 \times 5$ | $(12:4) \times 25$ | - |
| $\rho = \frac{1}{4}$ | $3 \times 5$ | $(11:4) \times 25$ | - |
| $\rho = \frac{1}{8}$ | $2 \times 3$ | $(11:4) \times 25$ | - |
| $\rho = \frac{1}{16}$ | $2 \times 3$ | $(10:3) \times 25$ | - |
| $\rho = \frac{1}{32}$ | $2 \times 3$ | $(10:3) \times 25$ | - |

TAB. 7.1 – Discrétisations pour $\kappa = 0.1$ $\epsilon = 10^{-4}$

| | Stable Plane Wave FMM | | HF-FMM |
|---|---|---|---|
| | Terme propagatif | Terme évanescent | |
| $\rho = 1$ | $10 \times 20$ | $(5:2) \times 25$ | - |
| $\rho = \frac{1}{2}$ | $8 \times 15$ | $(5:2) \times 25$ | - |
| $\rho = \frac{1}{4}$ | $6 \times 12$ | $(5:2) \times 25$ | - |
| $\rho = \frac{1}{8}$ | $5 \times 9$ | $(5:2) \times 25$ | - |
| $\rho = \frac{1}{16}$ | $4 \times 8$ | $(5:1) \times 25$ | - |
| $\rho = \frac{1}{32}$ | $3 \times 5$ | $(4:1) \times 25$ | - |

TAB. 7.2 – Discrétisations pour $\kappa = 1$ $\epsilon = 10^{-4}$

### 7.1.2 Coût calculatoire de chaque étape

La table 7.4 montre le coût des différents principaux opérateurs des algorithmes HF-FMM et SPW-FMM (étapes d'initialisation, d'interpolation (Outer-to-Outer) et de transfert (Outer-to-Inner)) en régime haute fréquence avec $\kappa = 10$ et $\epsilon = 10^{-4}$. Ces coûts sont donnés en secondes et correspondent à 10000 évaluations de ces opérateurs. Nous nous limiterons une fois de plus aux niveaux vérifiant $\rho \geq \frac{1}{2}$ en ce qui concerne la méthode HF-FMM. Le développement

| | Stable Plane Wave FMM | | HF-FMM |
|---|---|---|---|
| | Terme propagatif | Terme évanescent | |
| $\rho = 1$ | $52 \times 108$ | $(5:3) \times 83$ | $52 \times 108$ |
| $\rho = \frac{1}{2}$ | $31 \times 64$ | $(6:3) \times 49$ | $31 \times 64$ |
| $\rho = \frac{1}{4}$ | $20 \times 40$ | $(8:3) \times 33$ | - |
| $\rho = \frac{1}{8}$ | $13 \times 25$ | $(9:3) \times 27$ | - |
| $\rho = \frac{1}{16}$ | $9 \times 18$ | $(9:3) \times 25$ | - |
| $\rho = \frac{1}{32}$ | $7 \times 15$ | $(10:3) \times 25$ | - |

TAB. 7.3 – Discrétisations pour $\kappa = 10$ $\epsilon = 10^{-4}$

en ondes planes stabilisé (SPW-FMM) ne suit pas cette restriction et nous pourrons voir les valeurs correspondant aux niveaux de $\rho = 1$ à $\rho = \frac{1}{16}$.

Nous pouvons y voir que les coûts des étapes d'initialisation (colonnes *Init*), d'interpolation (colonnes *O-to-O*) et de transfert (colonnes *O-to-I*) de l'expansion SPW-FMM à la fois en régime intermédiaire et basse fréquences, de $\rho = 1/4$ à $\rho = 1/32$, sont comparables aux coûts de l'expansion HF-FMM à $\rho = 1/2$. Cependant aux niveaux $\rho = 1/2$ et $\rho = 1$, la formulation SPW-FMM est plus coûteuse en terme que la HF-FMM.

| | HF Init | SPW Init | HF O-to-O | SPW O-to-O | HF O-to-I | SPW O-to-I |
|---|---|---|---|---|---|---|
| $\rho = 1$ | 6.80 | 12.38 | 82.40 | 90.8 | 0.691 | 1.056 |
| $\rho = \frac{1}{2}$ | 2.40 | 4.56 | 21.05 | 27.0 | 0.244 | 0.503 |
| $\rho = \frac{1}{4}$ | - | 2.71 | - | 10.6 | - | 0.331 |
| $\rho = \frac{1}{8}$ | - | 2.12 | - | 5.48 | - | 0.254 |
| $\rho = \frac{1}{16}$ | - | 1.78 | - | 2.65 | - | 0.218 |
| $\rho = \frac{1}{32}$ | - | 1.70 | - | 2.00 | - | 0.233 |

TAB. 7.4 – Coûts des principaux opérateurs. $\kappa = 10$, $\epsilon = 10^{-4}$

Le test numérique suivant considère le coût total de la méthode pour un problème modèle d'un cube de taille 4. Nous y voyons que la stabilité de la SPW-FMM permet d'accroître le nombre de niveau utilisable dans l'algorithme FMM. En régime basse fréquence cela peut permettre de réduire fortement la consommation mémoire et le coût calculatoire. Les tables 7.5 et 7.6 montrent le coût total estimé. L'unité de mesure est la seconde. Pour le cas de la SPW-FMM, nous avons pris $\kappa = 10$, $\epsilon = 10^{-4}$ et un oct-tree à 7 niveaux. Au niveau 1, $\rho$ vaut 4, et au niveau 7, $\rho$ est égal à $\frac{1}{16}$. Chaque cellule du niveau 7 comprend 100 points de Gauss.

À cause de restrictions sur la HF-FMM, il n'est pas possible d'atteindre 7 niveaux dans cet exemple où nous sommes limités à 4 niveaux. Cela permet d'assurer que $\rho$ est toujours plus grand que $\lambda/2$. C'est bien cette contrainte qui provoque une forte décroissance de la performances globale de telle manière que pour un objet rayonnant dans un régime de fréquence intermédiaire ou pour un grand nombre de niveaux, la SPW-FMM est plus efficace que la HF-FMM.

Quelques commentaires au regard des tables 7.5 et 7.6 :

- *Prop. Leaf* et *Eva. Leaf* montrent les coûts de l'initialisation des cellules aux feuilles ainsi que le coût de l'intégration finale (équations 4.28, 4.2.2), respectivement pour les termes propagatif et évanescent. Au niveau le plus fin, chaque boite comprend 100 points.
- *Prop. Interpolation* et *Eva. Interpolation* au niveau de taille $l$ montrent le coût des étapes d'interpolation (Outer-to-Outer) du niveau de taille $l$ au niveau de taille $2l$ et d'anterpolation (Inner-to-Inner) du niveau de taille $2l$ a niveau de taille $l$, respectivement pour les termes propagatif et évanescent.
- *Prop. Transfer* et *Eva. Transfer* montrent les coûts de transferts (Outer-to-Inner) respectivement pour les termes propagatif et évanescent à chaque niveau.
- La table 7.6 donne le même type de résultats pour la formulation HF-FMM, qui sont équivalent aux valeurs obtenues pour le terme propagatif de la SPW-FMM, vu que nous avons utilisé le même type de discrétisation et d'opérateurs. Toutefois, étant donné que nous utilisons le même nombre de particules entre les cas SPW-FMM and HF-FMM, et que le nombre de niveaux est réduit sur la formulation HF-FMM, le nombre de particules par cellule du niveau 4 (les feuilles) dans la HF-FMM est de l'ordre de 3000. La principale conséquence apparaît à l'initialisation. Par ailleurs, le coût résultant des interactions proches (traitées au niveau 4) fait exploser le coût total.
- la colonne *Total* donne alors le coût cumulé des différentes étapes de l'algorithme ainsi que d'une itération de l'algorithme FMM dans cette configuration.

Il est possible de voir qu'une méthode stable en fréquences basses et intermédiaires permet de réduire le coût significativement.

| | $\rho = 1$ | $\rho = \frac{1}{2}$ | $\rho = \frac{1}{4}$ | $\rho = \frac{1}{8}$ | $\rho = \frac{1}{16}$ | Total |
|---|---|---|---|---|---|---|
| Prop. Leaf | | | | | 1.05 | 1.05 |
| Eva. Leaf | | | | | 7.43 | 7.43 |
| Prop. Interpolation | | 1.25 | 1.79 | 3.30 | 5.63 | 12.0 |
| Eva. Interpolation | | 0.35 | 1.09 | 3.03 | 6.97 | 11.4 |
| Prop. Transfer | 0.15 | 0.26 | 0.40 | 0.60 | 1.14 | 2.55 |
| Eva. Transfer | 0.01 | 0.04 | 0.13 | 0.45 | 1.58 | 2.21 |
| SPW-FMM | | | | | | 36.6 |
| Coût total comprenant les interactions proches | | | | | | 170 |

TAB. 7.5 – Temps d'exécution de la SPW-FMM pour le cube test. $\kappa = 10$, $\epsilon = 10^{-4}$

Le nombre optimal de niveau peut être déterminé par le test suivant (table 7.7). Nous utilisons la même configuration du problème, le même nombre total de particules, mais un nombre variable de niveaux allant de 3 à 7. Le nombre optimal de niveau sera donné par celui réduisant au minimum le coût global *i.e.* , le coût de la SPW-FMM et des interactions proches. Ainsi, le nombre optimal de niveaux est de 7, ce qui correspond à 100 particules par cluster au niveau des feuilles.

| | $\rho = 1$ | $\rho = \frac{1}{2}$ | Total |
|---|---|---|---|
| Leaf | | 11.4 | 11.4 |
| Interpolation | | 1.25 | 1.25 |
| Transfer | 0.15 | 0.26 | 0.41 |
| HF-FMM | | | 13.1 |
| Coût total comprenant les interactions proches | | | 1150 |

TAB. 7.6 – Temps d'exécution de la HF-FMM pour le cube test. $\kappa = 10$, $\epsilon = 10^{-4}$

| Nombre de niveaux | 3 niveaux | 4 niveaux | 5 niveaux | 6 niveaux | 7 niveaux | 8 niveaux |
|---|---|---|---|---|---|---|
| Prop. Leaf | 32.4 | 11.4 | 4.6 | 2.4 | 1.1 | 0.67 |
| Eva. Leaf | 26.6 | 10.3 | 8.3 | 7.7 | 7.4 | 7.4 |
| Prop. Interpolation | | 0.62 | 1.52 | 3.16 | 6.0 | 23.4 |
| Eva. Interpolation | | 0.17 | 0.72 | 2.23 | 5.7 | 38.8 |
| Prop. Transfer | 0.15 | 0.41 | 0.81 | 1.41 | 2.55 | 5.43 |
| Eva. Transfer | 0.01 | 0.05 | 0.18 | 0.63 | 2.21 | 9.07 |
| SPW-FMM | 59.1 | 23.8 | 18.4 | 22.9 | 36.6 | 84.7 |
| Interactions proches | 3500 | 1100 | 300 | 80 | 20 | 5 |
| Coût total | 3740 | 1200 | 375 | 170 | 165 | 345 |

TAB. 7.7 – Coût de la SPW-FMM en fonction du nombre de niveaux. $\kappa = 10$, $\epsilon = 10^{-4}$

## 7.2 Applications numériques

Dans cette section, nous allons présenter quelques résultats numériques montrant à la fois la validité des solutions obtenues par une comparaison avec des méthodes de référence, ainsi que la faisabilité de cas réalistes.

Tous les résultats ont été obtenues par une formulation CFIE (*cf* section 1.1.1) sans préconditionnement. L'absence de préconditionneur peut induire une convergence assez lente et donc un nombre d'itérations important. Toutefois, nous nous intéresserons ici principalement à la précision et la stabilité de la méthode.

Dans l'ensemble de ces tests, nous avons pris $\mu_0 = 1.25751 \times 10^{-6}$, $\epsilon_0 = 8.84806 \times 10^{-12}$ soit une vitesse de la lumière $c = 299792548.2 m/s$ ; nous rappelons que les objets considérés sont parfaitement conducteurs. L'ensemble des objets sera éclairé par une onde électromagnétique dans la direction $-\vec{z}$.

L'approximation du noyau est construite telle que l'erreur commise soit de $\epsilon = 10^{-4}$. Sauf mention explicite, le solveur itératif sera de type GMRES (Generalized Minimal Residual) avec un redémarrage tous les 100 itérations et un critère d'arrêt sur le résidu normalisé fixé à $10^{-4}$.

La plupart des maillages seront présentés accompagnés d'un indicateur de qualité des triangles. La qualité d'un triangle $T$ de sommets $A$, $B$, $C$ est définie par

$$\mathcal{Q}_T = \frac{4\sqrt{3}\,|T|}{p \cdot h_{max}}$$

avec

- $|T|$, l'aire du triangle $T$,
- $p$, le périmètre de $T$ ; $p = \|\overrightarrow{AB}\| + \|\overrightarrow{BC}\| + \|\overrightarrow{CA}\|$,
- $h_{max}$, la longueur du plus grand côté de $T$ ; $h_{max} = \max(\|\overrightarrow{AB}\|, \|\overrightarrow{BC}\|, \|\overrightarrow{CA}\|)$,
- $\|.\|$, la norme euclidienne.

Les triangles optimaux sont alors les triangles équilatéraux dans la métrique donnée, et alors de qualité 1.

### 7.2.1 Régime basse fréquence : Sphère unité de taille $\frac{\lambda}{15}$

Ce premier exemple (figure 7.2) présente la SER (Surface Équivalent Radar) d'une sphère unité maillé par 3000 arêtes (degrés de liberté) illuminé par une onde électromagnétique de longueur d'onde $\lambda = 30$. La solution de référence est la solution exacte donnée par les séries de Mie.

Rappelons que les séries de Mie permettent de déterminer directement le champ lointain diffracté d'une sphère. Considérons une sphère parfaitement conductrice de rayon $a$, illuminée par une onde électromagnétique plane arrivant de la direction $(\theta = 0, \phi = 0)$, polarisée verticalement, alors il est possible de déterminer $\sigma(\theta, \phi)$, la valeur de la surface équivalente

radar normalisée émise dans la direction $(\theta, \phi)$.

$$\begin{cases} \psi_n(x) = x j_n(x) \\ \zeta_n(x) = x h_n^{(1)}(x) \\ a_n = \frac{\psi_n(\kappa a)}{\zeta_n(\kappa a)} \\ b_n = \frac{\psi_n'(\kappa a)}{\zeta_n'(\kappa a)} \\ S_1(\theta) = -\imath \sum_{n=1}^{\infty} (-1)^n \frac{2n+1}{n(n+1)} \left[ b_n \frac{\partial P_n^1(\cos\theta)}{\partial \theta} - a_n \frac{P_n^1(\cos\theta)}{\sin\theta} \right] \\ S_2(\theta) = \imath \sum_{n=1}^{\infty} (-1)^n \frac{2n+1}{n(n+1)} \left[ -a_n \frac{\partial P_n^1(\cos\theta)}{\partial \theta} + b_n \frac{P_n^1(\cos\theta)}{\sin\theta} \right] \\ \sigma(\theta, \phi) = \frac{4\pi}{\kappa^2} \left( |S_1(\theta)|^2 \cos^2\phi + |S_2(\theta)|^2 \sin^2\phi \right) \end{cases} \tag{7.1}$$

avec $h_n^{(1)}$ les fonctions de Hankel sphériques du premier type, $j_n$ la fonction de Bessel sphérique, $P_n^l$ les fonctions de Legendre associées.

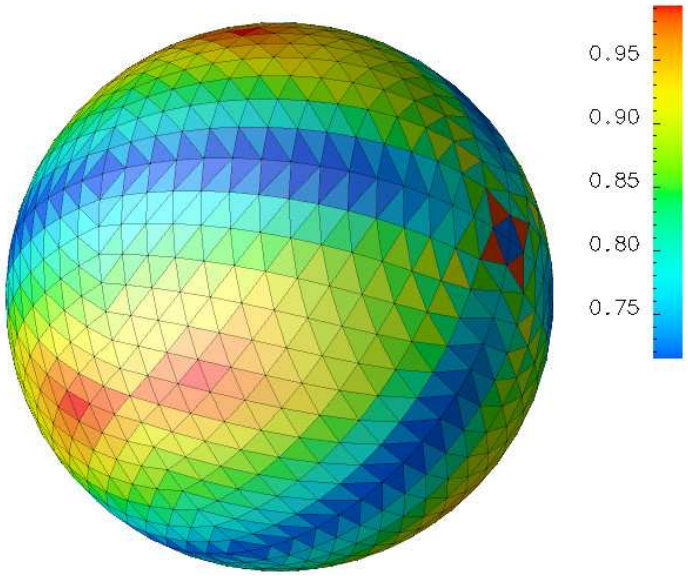



FIG. 7.1 – Maillage de la sphère et représentation de la qualité des triangles

La stabilité de la méthode SPW-FMM comparativement à la méthode HF-FMM peut être montrée par le fait que la HF-FMM ne permet pas de traiter ce cas même avec un nombre de niveau minimum (c'est-à-dire 3 niveaux, en deçà duquel la méthode n'utilise plus d'approximation FMM, puisque que toutes les cellules sont proches les unes des autres).

Pour ce petit exemple, nous nous sommes réduit à 4 niveaux (2 niveaux FMM effectifs), dont les discrétisations au niveaux 3 et 4 sont de tailles respectifs $4 \times 6$, $3 \times 4$ pour le terme propagatif (de la forme $n_\theta \times n_\psi$) et $(11:4) \times 28$ points à chaque niveau pour le terme évanescent (même notation que dans la section 7.1.1).

L'erreur de type $L^2$ relative vaut $1,1.10^{-3}$, et l'erreur de type $L^\infty$ relative vaut $1,4.0^{-3}$. Ces valeurs correspondent à une comparaison des courants surfaciques obtenus par la méthode FMM et un solveur itératif avec assemblage de la matrice.

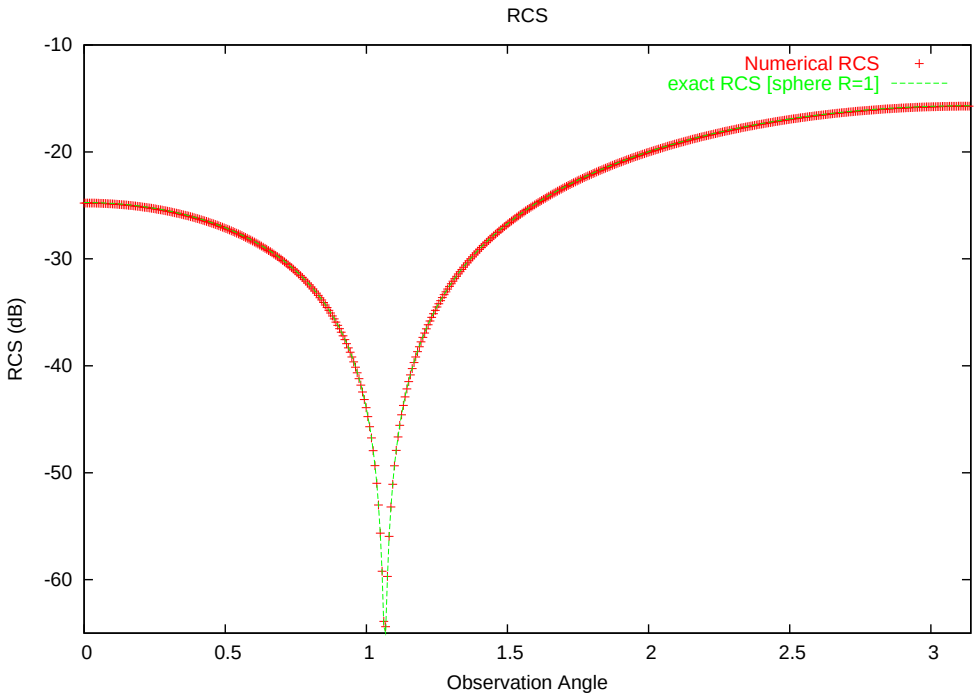



FIG. 7.2 – SER d'une sphère unité de taille $\frac{\lambda}{15}$

### 7.2.2 Régime de fréquences intermédiaires : Sphère de taille $2\lambda$

Ce second exemple (figure 7.3) reprend le même type de présentation qu'en régime basse fréquence et présente la SER d'une sphère unité à 3000 degrés de liberté éclairée par une onde de longueur $\lambda = 1$. La solution exacte est encore donnée par les séries de Mie.

Dans cette configuration, la méthode HF-FMM échoue (en terme de stabilité) dès que le nombre de niveaux devient supérieur ou égal à 5.

Ce résultat a été obtenu avec une décomposition SPW-FMM à 6 niveaux, où les discrétisations des niveaux de 3 à 6 sont de tailles $15 \times 28$, $10 \times 18$, $8 \times 14$, $6 \times 10$ points pour le terme propagatif et $(9:3) \times 28$, $(9:3) \times 28$, $(10:3) \times 28$, $(10:3) \times 28$ points pour le terme évanescent.

L'erreur de type $L^2$ relative vaut $1,2.10^{-2}$, et l'erreur de type $L^\infty$ vaut $3.10^{-2}$. Ces valeurs correspondent à une comparaison des courants surfaciques obtenues par la méthode FMM et un solveur itératif avec assemblage de la matrice.

### 7.2.3 Régime haute fréquence : Sphère $5.5\lambda$

À présent, voici un modeste exemple (figure 7.5) en régime haute fréquence (encore relativement bas par rapport à de prochains cas réalistes). Encore une fois, nous y représentons la SER calculée comparée à une solution donnée par les séries de Mie ; la longueur d'onde est de $\lambda = 0.36$. Le maillage utilisé comporte 18000 arêtes.

Nous sommes à présent dans le champ d'application de la méthode HF-FMM, qui ne présente donc pas de problème de stabilité.

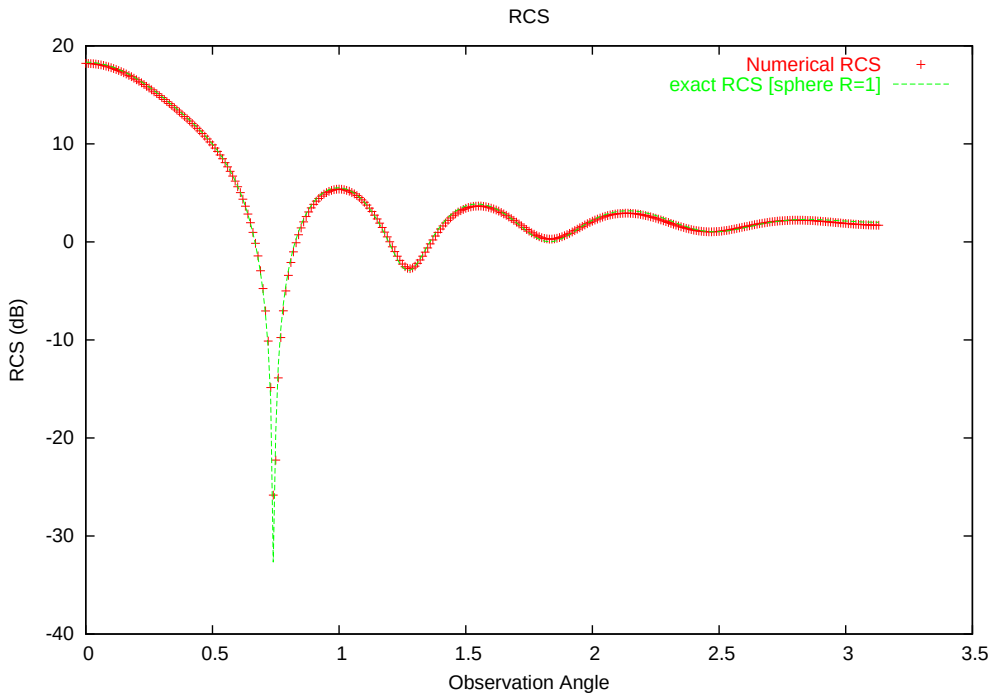



FIG. 7.3 – SER d'une sphère unité de taille $2\lambda$

Le calcul utilise 7 niveaux FMM, où les discrétisations des niveaux de 3 à 7 sont de $28 \times 54$, $17 \times 32$, $13 \times 24$, $9 \times 16$, $7 \times 12$ points pour le terme propagatif et de $(7:3) \times 48$, $(8:3) \times 32$, $(9:3) \times 28$, $(9:3) \times 28$, $(9:3) \times 28$ points pour le terme évanescent.

**Remarque** *La courbe SER comporte une défaut de convergence au niveau de la partie droite, aux environs d'un angle d'observation de $\pi$ (ombre relative à l'onde incidente). Ce défaut peut être corrigé en réduisant la tolérance dans le solveur itératif.*

### 7.2.4 Alphajet de taille $\lambda/20$

Ce premier exemple débute une série d'exemples plus réalistes que les précédentes sphères. Cependant, de tels exemples ne disposent pas de solution analytique connue. Toutefois, quand cela sera possible ces résultats seront comparés à des solutions obtenues par d'autres moyens tels qu'un solveur itératif (avec assemblage de la matrice) ou la HF-FMM.

Le maillage de l'Alphajet (figure 7.6) a été fourni par la société Dassault Aviation. Ce maillage comporte 7956 sommets et 15908 triangles. Bien entendu, ce maillage est largement sur-dimensionné, mais ne disposions pas d'outils permettant de simplifier la surface de l'objet de manière convenable.

La résolution à été effectué par la SPW-FMM à 5 niveaux dont les niveaux de 3 à 5 sont discrétisés en $3 \times 4$, $3 \times 4$, $3 \times 4$ points pour le terme propagatif et $(12:4) \times 28$, $(12:4) \times 28$, $(11:3) \times 28$ pour le terme évanescent. Par ces discrétisations nous pouvons constater les effets du basse fréquence sur les discrétisations : les discrétisations ont atteint une taille minimale et l'ajout de niveau ne permet de bénéficier des classiques réductions de coûts. Toutefois,

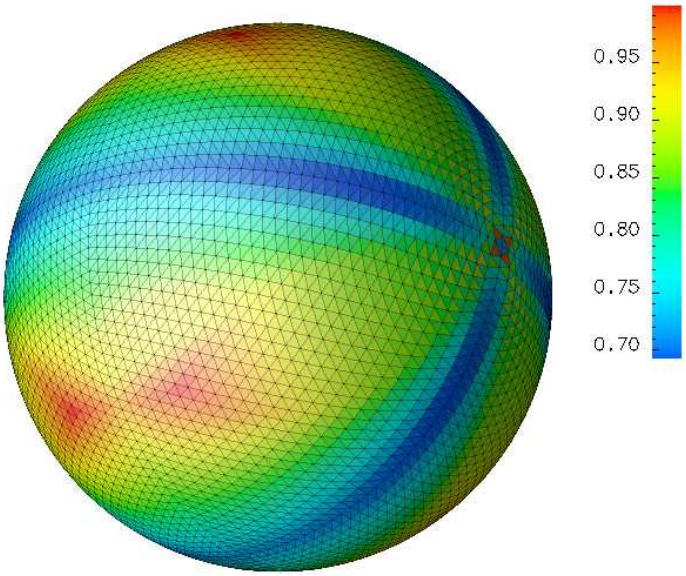



FIG. 7.4 – Maillage de la sphère et représentation de la qualité des triangles

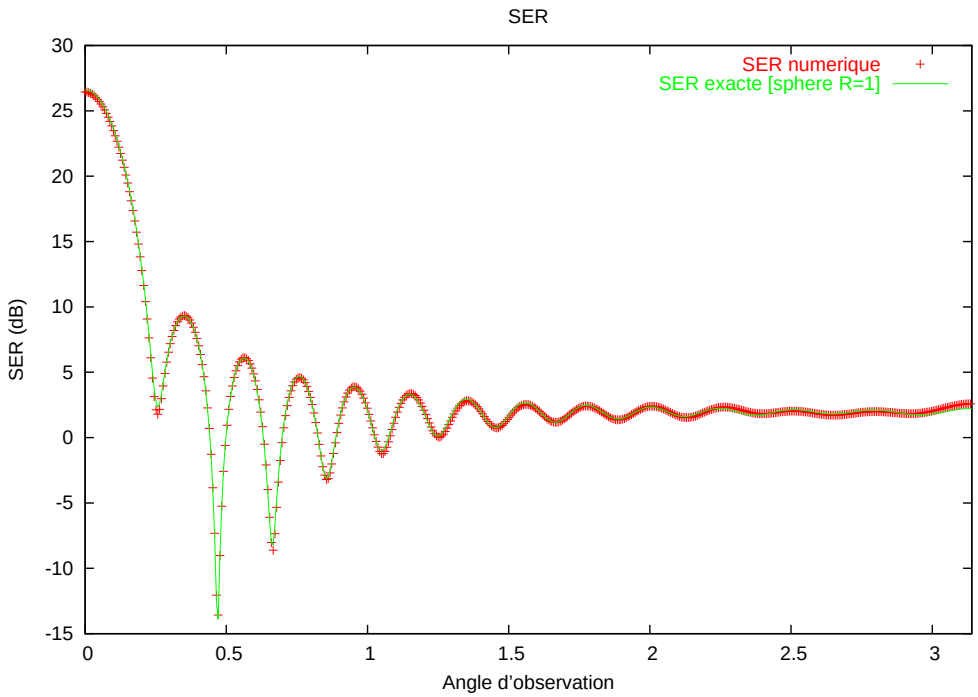



FIG. 7.5 – SER d'une sphère unité de taille $5.5\lambda$

le gain en espace mémoire est toujours présent, permettant de réduire le coût de la matrice d'interactions proches (la consommation mémoire totale est ici de 200Mo).

Le calcul (figure 7.7) a été réalisé en parallèle sur 8 processeurs en 5 heures et 1200 itérations pour atteindre un résidu normalisé (GMRES) de $1,2.10^{-3}$. L'objet mesure $0,0512\lambda$. Les figures 7.6 et 7.7 permettent une direction d'observation $-\vec{z}$ qui correspond par ailleurs à la direction d'illumination de l'objet.

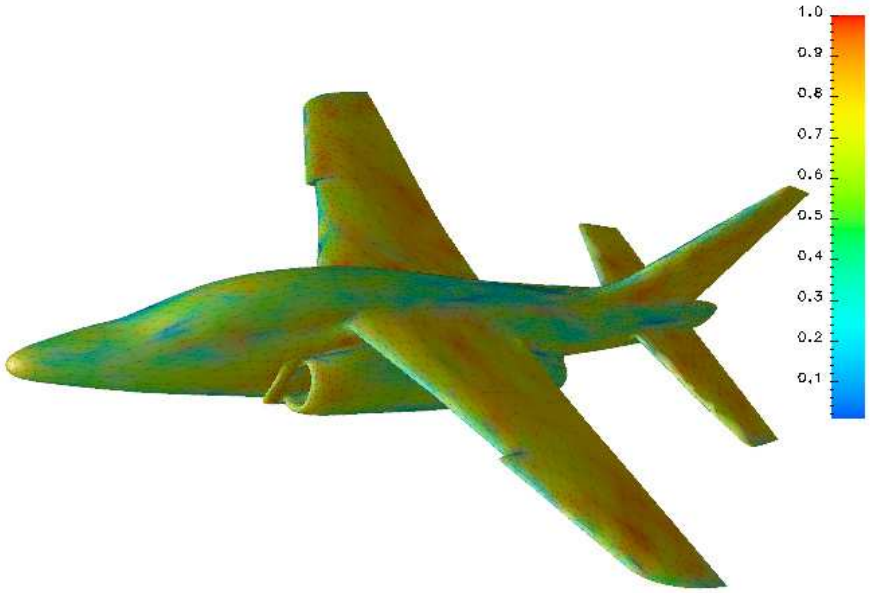



FIG. 7.6 – Maillage de l'Alphajet et représentation de la qualité des triangles

### 7.2.5 Voiture de taille $\frac{\lambda}{2}$

Ce second exemple «réaliste» consiste en l'éclairage par une onde électromagnétique provenant d'au dessus de la voiture. La taille de l'objet est ici la voiture de la longueur d'onde soit déjà 10 fois plus grande que dans le précédent exemple.

Les figures 7.9 et 7.11) montrent le maillage et la répartition de l'intensité du courant électrique à la surface de voiture composée de 7560 triangles.

La figure 7.10 compare les SER obtenues par la méthode SPW-FMM ainsi qu'un solveur itératif par assemblage de la matrice. Les deux méthodes convergent en 352 itérations avec un résidu inférieur à $10^{-4}$. Les temps de calcul (8 processeurs) et coûts mémoire pour la SPW-FMM et le solveur avec assemblage de la matrice sont respectivement sont de 42min/320Mo et 6min/3,2Go.

À ce régime, la méthode HF-FMM échoue (stabilité) dès que l'on utilise plus de 3 niveaux. Le calcul a été effectué avec 5 niveaux dont les discrétisations des niveaux de 3 à 5 sont $6 \times 10$, $5 \times 8$, $4 \times 6$ pour le terme propagatif et $(12 : 4) \times 28$ points pour chacun des niveaux du terme évanescent.

L'erreur de type $L^2$ relative vaut $1,08.10^{-2}$, et l'erreur de type $L^\infty$ relative vaut $1,6.10^{-2}$.

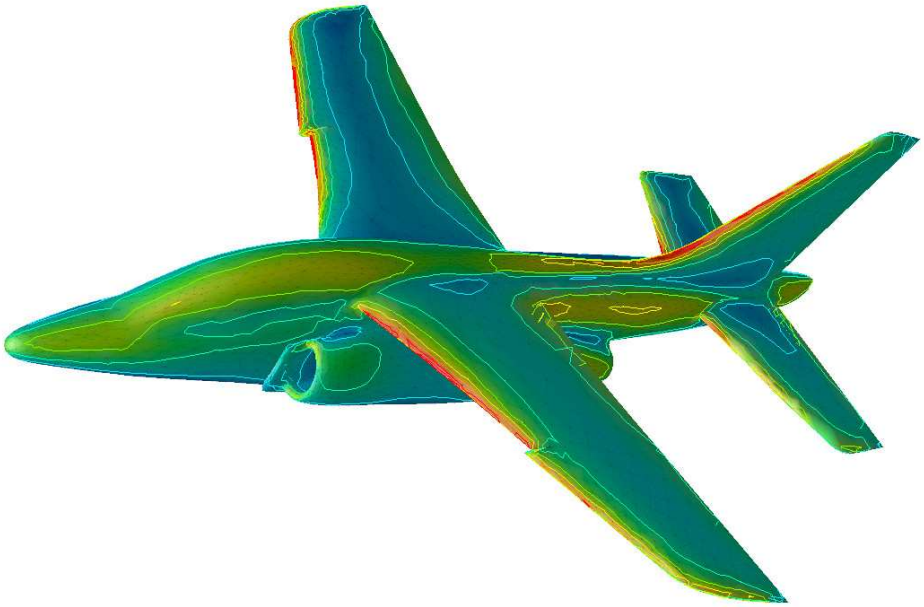


FIG. 7.7 – Intensité du courant électrique à la surface d'un Alphajet

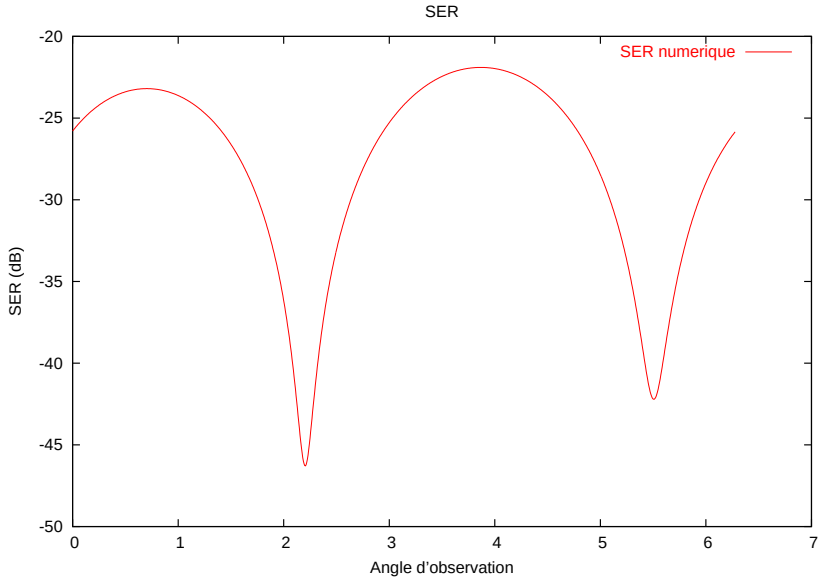



FIG. 7.8 – SER de l'Alphajet suivant le plan vertical central

Ces valeurs correspondent à une comparaison des courants surfaciques obtenues par la méthode FMM et un solveur itératif avec assemblage de la matrice.

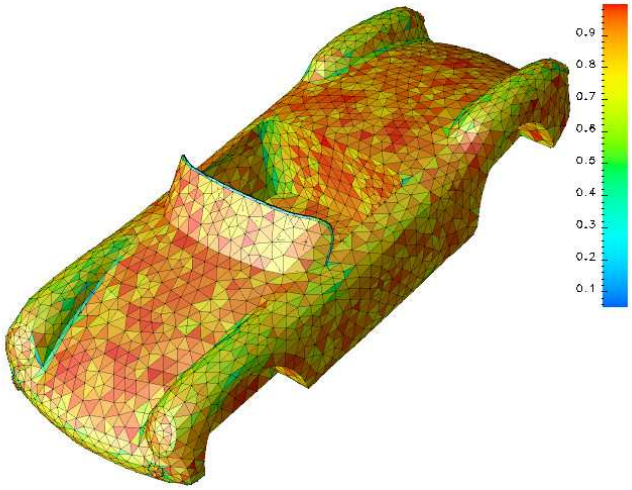



FIG. 7.9 – Maillage de la voiture et représentation de la qualité des triangles

### 7.2.6 Avion en régime haute fréquence

Abordons à présent un cas haute fréquence. L'objet considéré est un avion dont le maillage a été trouvé sur Internet. Ce maillage (figure 7.12) a été traité par subdivision de triangles afin d'atteindre la taille suffisante de 329592 triangles (soit 500 000 degrés de liberté). L'objet, dans sa plus grande dimension, mesure $41,4\lambda$. Le nombre moyen de points par longueur d'onde est de 8.9 avec un minimum à 5.7 et un maximum à 18.4. Les méthodes multipôles utilisées comportent 10 niveaux avec une discrétisation de $6\times12$ à $137\times280$ pour la méthode HF-FMM et de $6\times10$, $(11:4)\times28$ à $129\times256$, $(5:3)\times224$ pour la méthode SPW-FMM (resp. terme propagatif et évanescent).

Un problème de cet ordre approche ce que l'on peut attendre des méthodes multipôles en haute fréquence. Le temps de résolution par une méthode de type HF-FMM est de l'ordre de 3h avec une convergence en 76 itérations. Le temps de résolution avec la méthode SPW-FMM est ici de l'ordre de 13h pour 79 itérations. Cette différence est considérable pour un problème que l'on peut résoudre par une autre méthode.

Voyons la répartition des coûts (table 7.8).

Le principal argument à cette forte différence de performance est le traitement par directions qui multiplie par 7 le nombre de sous itérations contenues dans une itération du solveur itératif. Toutefois, ces sous itérations étant légèrement moins coûteuses qu'une itération HF-FMM, le rapport des temps peut être réduit à 4 dans le cas étudié.

Bien que les différentes directions ($\pm x$, $\pm y$, $\pm z$) soient traitées distinctement, le nombre de transferts (toutes directions confondues) reste identique dans les deux méthodes (au facteur 2 près induit par le traitement des deux intégrales) ; c'est pourquoi, la proportion du coût des transferts est réduite dans la méthode SPW-FMM.

Toujours en raison du traitement par direction, le coût des étapes d'inter/anter-polation (agrégation/dispersion) augmente mais reste dans les mêmes proportions que dans le cadre de

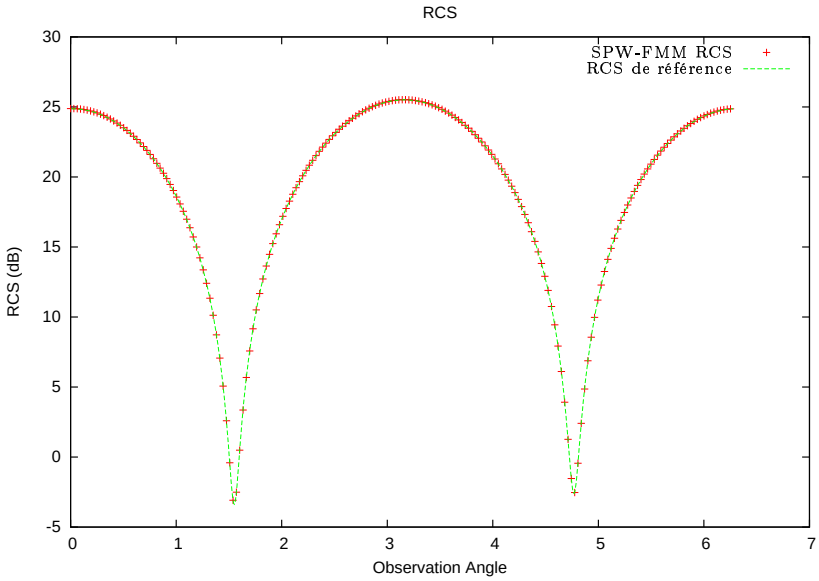



FIG. 7.10 – SER d'une voiture de taille $\lambda/2$

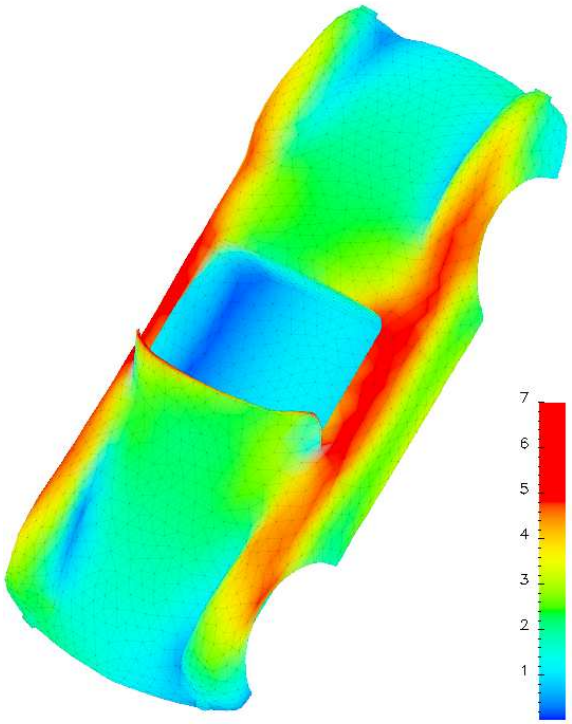



FIG. 7.11 – Courant électrique à la surface de la voiture

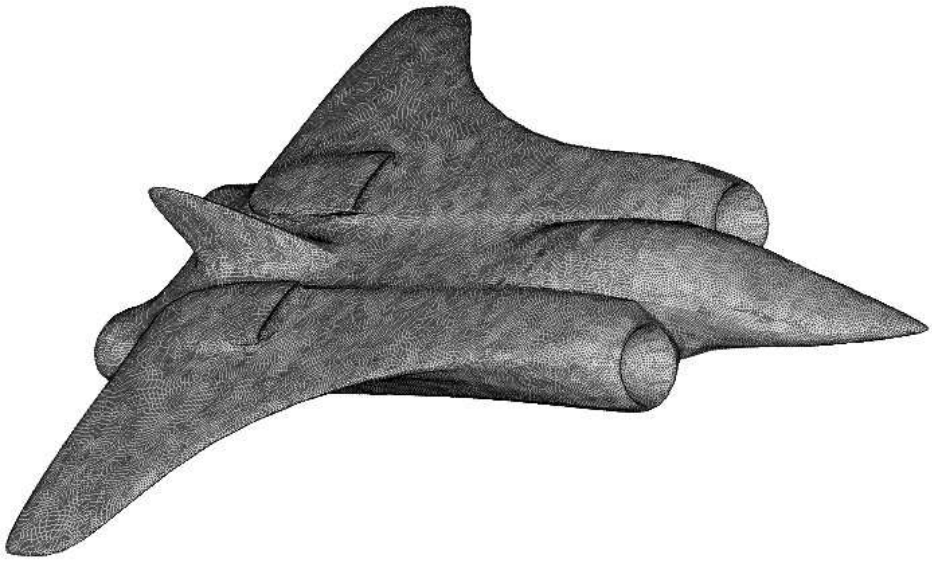


FIG. 7.12 – Maillage de l'avion

| | HF-FMM | | SPW-FMM | |
|---|---|---|---|---|
| Initialisation+Intégration | 9s | (6%) | 92s | (16%) |
| Agrégation | 18s | (13%) | 96s | (17%) |
| Transfert | 103s | (73%) | 318s | (57%) |
| Dispersion | 11s | (8%) | 52s | (9%) |
| Total | 141s | | 557s | |

TAB. 7.8 – Comparaison des coûts FMM pour une itération du solveur itératif

la méthode HF-FMM.

Si l'on regarde la répartition relative des coûts, un des principaux points handicapant de la méthode SPW-FMM est ici les feuilles. L'initialisation des feuilles dans la méthode SPW-FMM à base de décomposition SVD (sections 4.2.2 et 4.3.1) est en effet une construction bien plus complexes que celles en œuvre dans la méthode HF-FMM (simple évaluation de la fonction noyau).

Regardons de plus près le coût de l'initialisation en fonction du nombre de points dans une cellule. Dans les sections 4.2.2 et 4.3.1, nous avons pu voir des constructions de coûts affines décomposables en un coût linéaire consistant en l'évaluation de fonctions similaires au noyau, suivi du coût constant d'un produit de type matrice-vecteur (assimilable aux produits par $u_p^l(\chi)^*$ de la section 4.2.2). Or, ce produit matrice-vecteur provoque une forte chute des performances dès que le nombre de points par cellule devient trop petit relativement au coût d'un produit matrice-vecteur. Ce minimum est estimé (et dépend fortement des machines) à quelques dizaines (entre 20 et 50).En ce qui concerne l'exemple en cours, notons que le nombre moyen de points par cellule est ici de 1 avec un maximum à 9. Même avec ce minimum atteint, il est évident qu'il n'est pas possible de rivaliser avec l'étape d'initialisation de la méthode HF-FMM, mais le contraste en serait moins marqué, et refléterait les gains possibles avec les discrétisations optimisées par directions.

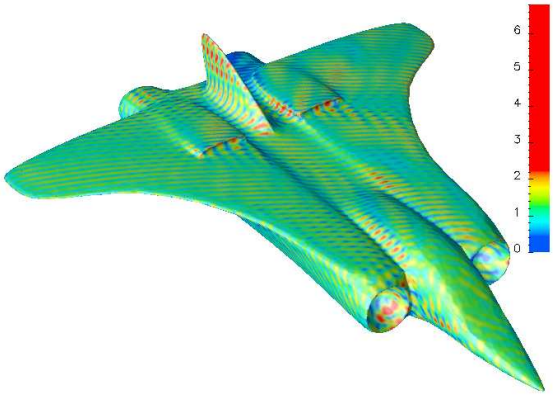



FIG. 7.13 – Méthode HF-FMM

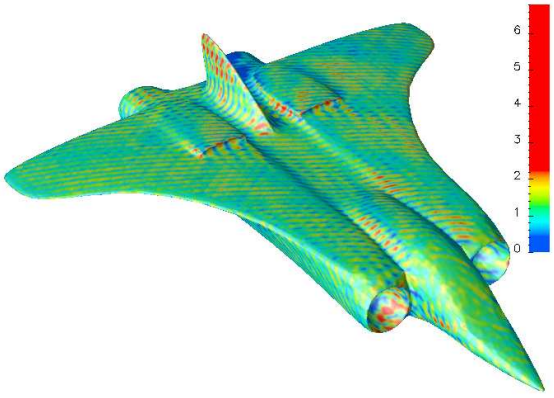



FIG. 7.14 – Méthode SPW-FMM

Comparaison des courants électriques calculés par les méthodes HF-FMM et SPW-FMM sur le cas d'un avion en haute fréquence

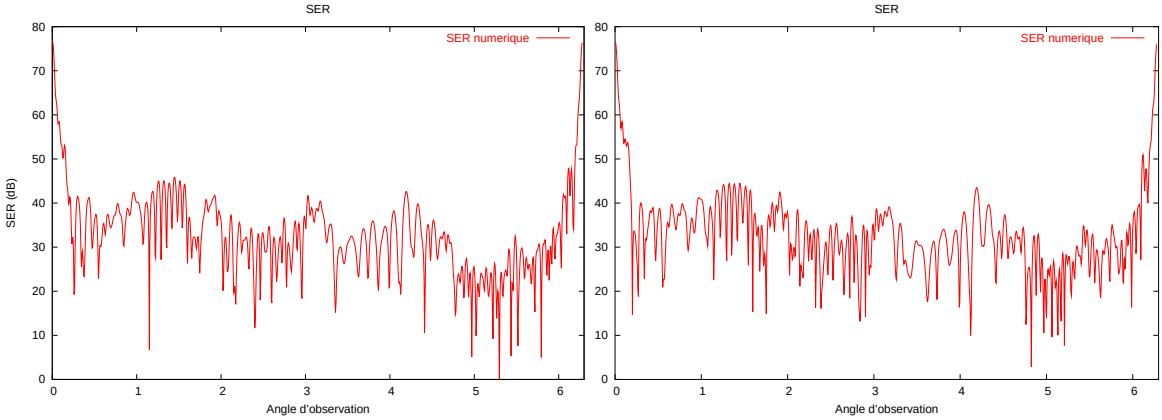



FIG. 7.15 – Méthode HF-FMM

FIG. 7.16 – Méthode SPW-FMM

Comparaison des SER calculés par les méthodes HF-FMM et SPW-FMM sur le cas d'un avion en haute fréquence

# Annexe A

# EasyMSG

Cette annexe présente le développement de la bibliothèque EasyMSG (prononcer «Easy Message») dont le but est la simplification de la gestion des communications et leur optimisation par un traitement simultané de communications asynchrones et de calculs sans communication.

Cette bibliothèque est un point important des développements dans le cadre des méthodes multipôles dont l'étape de «transfert» est particulièrement critique en terme de communications.

Cette section a fait l'objet d'une publication [39].

Une version orientée multi-threading de l'overlapping (opposée à la notion de programmation SPMD[7]) a par ailleurs été développée mais ne sera pas abordée dans cette annexe.

The development of a parallel solver for Maxwell Equations by integral formulations and Fast Multipole Method (FMM) was a good place for code optimization of critical parts with a lot of communications and computations. Generally, many parallel programs need to communicate, but choosing explicitly the way and the instant may decrease the efficiency of the overall program. So, the overlapping of computations and communications may be a way to reduce this drawback. We will see a implementation of this techniques using dynamic and adaptive overlapping based on the *EasyMSG* high level C++ library over MPI, a case of SPMD programming.

## A.1 Introduction

We study here an implementation with overlapping in a Parallel Fast Multipole Algorithm applied to Maxwell Equations[62] solved by integral formulations [38]. This implementation contents a critical part : *the transfers*.

This article will present adaptive overlapping techniques used for our problem after a brief overview of the mathematical background which leads us to consider these techniques as a true alternative to the classical implementations. These techniques are wrapped into a library *EasyMSG*, which provides a abstraction of lower communication layer like *MPI*[37]. The performance given by a such implementation are really significant as we will see in section A.5.

[7] Single Program Multiple Data

### A.1.1 Problem statement

A critical part of this algorithm is the "*transfers*" or computations involving data of "*far*" cells.

A *transfer* is a computation between two values located on two *far* cells (see section 2 for more explanation).

However, these *transfers* (computations) may use cells (data) located on different processes. If the cells are located on the same process we have *local transfers*, else we have *non-local transfers* between different processes with migrations by communications.

This step of the Fast Multipole Algorithm may be summarized as matrix-vector products $y^l = M^l\ x^l$, where we have mapped the cells of level $l$ on a vector $x^l$. Each line of the matrix $M^l$ represents *transfers* to do :

- $M^l_{i,j} \neq 0$ if the cell mapped on $x^l_i$ is *far* from $x^l_j$, and by symmetry $M^l_{j,i} \neq 0$ ;
- $M^l_{i,j} = 0$ otherwise.

Moreover, each value $x^l_j$ of the vector $x^l$ is also a small vector, so each scalar operation must be replaced with a vectorial operation. For clarity, we simplify the computation of the *transfer* by choosing $M^l_{j,i} = M^l_{i,j} = 1$ or 0.

The matrices are very sparse and we can see that this part of the Fast Multipole Algorithm leads to an important exchange of data (non-local transfers) but we can overlap the communications with many *local-transfers*. For increasing the overlapping, we will overlap all the matrix-product $M^l\ x^l$ together.

Every parallel program needs to compute and communicate. In a MIMD[8] approach, we can decompose the program into many parts where each process can compute or communicate when it wants. Moreover, when reaching the low level programming (OS, Hardware), the decomposition and overlapping appear clearly and may be optimized by a better knowledge of the behavior of communications (hardware interruptions, ...). However, for increasing the portability with a higher level of abstraction, we need to understand how the communication works regardless of the low level architecture, even more so in SPMD[9] context.

In SPMD, a parallel program is written as a sequential program running on $N$ processes. Using a message passing library (like MPI-1) in a SPMD context implies choosing when sending and receiving messages along the sequential codes.

---

A 'bad' algorithm

**while** Communications needed **do**
  Start communications (non blocking sending)
  Do $n$ computations (if possible)
  Finish communications (waiting and receiving)
**end while**
Do remaining computations

---

Algorithm A.1.1 uses a predefined parameter $n$ and the communications are uniformly distributed between computations. If $n = 0$ the algorithm does all the communications before the computations ; if $n = \infty$, the algorithm does all computations between sending and receiving the data. This simple algorithm allows to embed the data transfers between computations.

[8]MIMD stands for "Multiple Instructions, Multiple Data".
[9]SPMD stands for "Single Program, Multiple Data".

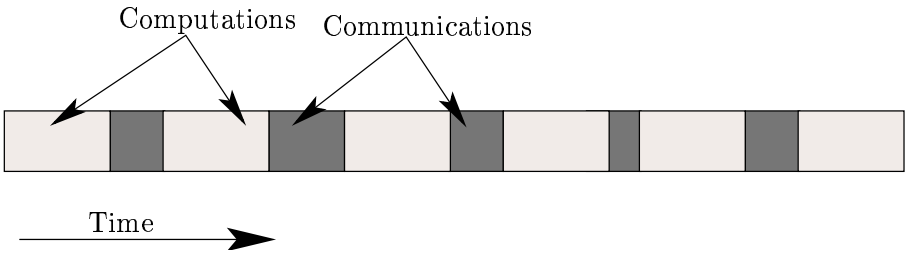



FIG. A.1 – Static Synchronous Communications/Computations mixing

The main drawback is the 'waiting' for the communications, so each communication time may be variable.

As is well known [66, 4], a better choice is to use asynchronous communications : we can send or receive a message by testing if there is such an incoming or outgoing message, else we do a few computations.

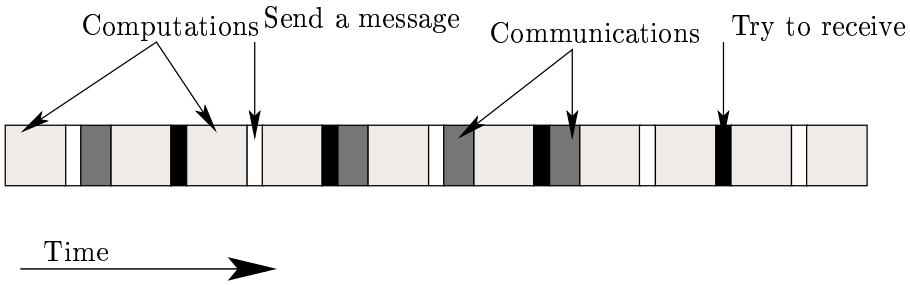



FIG. A.2 – Static Asynchronous Communications/Computations mixing

A problem may occur when there are many processes : we need to define a schedule of the incoming and outgoing communications. Furthermore, each communication implies a incompressible time for negotiation ; we call it *latency* time. Then, we must try to aggregate communications for reducing several latency times to only one. Aggregating communications needs to know **when** to send the buffer and **how** to receive aggregated data. Here comes the *EasyMSG* library.

Many other libraries exist for distributed scientific computing, with many different objectives as providing efficient objects for scientific computing over MPI (POOMA[60], PetsC[3]) or with more distributed objects (C++//[20], ProActive[19]). Our first approach was to simplify the usage of the message passing library by providing tools for an efficient overlapping of communications and computations without a special care for complex structures and their potential dead-locks.

*EasyMSG* provides a few orthogonal features implemented over a Message Passing Library as communication layer (currently MPI) :

- synchronous or asynchronous communications ;
- unbuffered or buffered communications :
  *a buffered communication is sent only when a buffer (the size is fixed by the user) is full (or when another datum cannot be added)* ;

- block communications
  *may be used for sending vectors of data (need a buffer with enough space)* ;
- special communication tagging
  *for sending a tag datum with data ; used for identify received data* ;
- several algorithms for multiple receivers or senders
  *as several buffers to a target process or from a process* ;
- Wrappers for the identification of the type to transfer are also implemented. However, a limitation is to send only serialized data, *i.e.* vectors of data. An implementation of the serialization of any object may be done inside *EasyMSG*, if we can determinate the overall size of such an object.

**Remark** *An important characteristic of such communications is the unpredictable order of communication reception. If one sends two data* A, B, *we cannot know how they will arrive :* A *then* B, *or* B *then* A *! This is a very good reason for using tagged data.*

## A.2 Operations decomposition

Even using asynchronous communications may be not enough for optimizing a program. Optimizing the overlapping of communications and computations may be necessary.

We call *local computations*, computations without any communication. Furthermore, we define communication tasks for each target process. Our goal is to know what kind of communication and computation we will do for each target at any time. Such decomposition of the work allows to use the following algorithm :

A simple way to overlap communications and computations

```
Let InBuffer, OutBuffers be collections of buffers for receiving and sending
while Something to do do
  if there is a ready datum in InBuffer then
    Treat the incoming message
  else
    for all cpu_i do
      if OutBuffers[i] is not full then
        Prepare and push a datum into OutBuffers[i]
      end if
    end for
  else
    Do some local computations
  end if
end while
```

**Remark** *Each test on* OutBuffers *and* InBuffer *is a test which uses specific algorithms, like a rotating buffer (uses a circular multiple buffer), or a multiple buffer (uses free buffers even if the sending order is different).*

Then, the important aspect for a good efficiency is the definition of "*do some local computations*". Its granularity must be small enough for avoiding to spend lot of time (and not

be ready to treat communications) and not too small to avoid useless tests (and to *stress* the network).

An optimal solution is obtained if we can do all communications and computations with a minimal time where the parameter is the number of computations by loop, called *aggregation parameter*.

## A.3 C++ Implementation of *EasyMSG*

This library must be designed for heavy usage, with many calls to basis functions such as arrival test functions and many more. Another requirement is to allow simple interface for implementing new algorithms or container types. A way provided by the C++ language is inheritance and polymorphism via virtual functions. However, this way may be very inefficient, if we abuse of virtual functions. Virtual functions allow to access to functions of sub-classes from top level classes. Calling a virtual function from an abstract object (object with virtual function referring to sub-classes functions) needs to scan virtual function tables for defining where the corpus of the *real* function (in a sub-class). This is the cost of virtual functions. But, if we use them just for defining interfaces, this drawback disappears. We associate virtual functions to a well-defined interface and *templates* for a generic fast static access (defined at compile time).

**Remark** *Another way for associating a well-defined interface and genericity is to use static polymorphism.*

The design of this library is organized around ;
- **Locks** : objects designed for managing (single or multiple) communications : to know when they are done, available or buzzy ;
- **Containers** : there are two kinds of Containers : for receiving (based on *InBufferModel*) and sending (based on *OutBufferModel*) ; they represent fixed buffers for storing linear data to transmit by applying (un-)serialization ;
- **Communication algorithms** : to provide a common support for the *Containers* for managing *when* the data are received/sent ; we can use multiple targets, multiple buffers, FIFO scheduling, ...
- **External Parameters** : for the previous classes in order to provide the end user with a way to communicate with the communication level functions. In the MPI case, they allow to change/read MPI specific tags, targets, status.

Its main advantage is to allow to write easily efficient high-level parallel concurrent programs without any deadlocks.

The current implementation uses MPI as communication layer, but any other message passing library may be used by adapting the containers to the new library. The algorithms may be kept without any changes.

We will show three interfaces of common objects : the fundamental *InBufferModel*, a top level *MPI_InBlockComplexTagBuffer*, and an *MultiSender* algorithm.

### A.3.1 InBufferModel

The InBufferModel class is the top level class of abstraction of buffers for incoming messages. Any higher Container receiving data must inherit from it.

```
1   template<typename ObjT, typename InOutT, typename StorT>
2   class InBufferModel {
```

*The specific types are propagated by some* typedef*'s*

```
3   public:
4     typedef ObjT ObjectType;
5     typedef InOutT InOutType;
6     typedef StorT StorableType;
7   protected:
8     typedef InBufferModel<ObjectType,InOutType,StorableType> Model;
9   public:
```

*Constructors and destructors*

```
10    InBufferModel();
11    InBufferModel(const unsigned _bufferSize);
12    virtual ~InBufferModel();
```

*The main function : extracting a value of the buffer*

```
13    virtual InOutType pop() = 0;
```

*Functions about managing the buffer size and its capacity*

```
14    virtual void resize(const unsigned _bufferSize);
15    virtual bool empty() const;
16    virtual unsigned sizeLeft(void) const;
17    virtual unsigned sizeOf(const InOutType &value) const = 0;
```

*Functions about managing a new value (if it can)*

```
18    virtual bool canDoIt() const;
19    virtual bool mustReset() const;
20    virtual void reset();
```

*Functions about starting/stopping communications*

```
21    template<typename Lock> void start(Lock &lock) = 0;
22    template<typename Lock> void stop(Lock &lock) = 0;
23  };
```

### A.3.2 MPI_InBlockComplexTagBuffer

The *MPI_InBlockComplexTagBuffer* object allows to receive variable-length vector of data with an additional value : a *tag*. This *tag* must be of any fixed-length type. Moreover, this buffer carries an additional information for un-serializing the vector : the size.

In order to avoid large copied data as return values of the *pop* function, we use a couple of pointers to define the vector of data : one on the first element, another on the end of the vector *i.e.* equal to the first one plus the number of values. However, we want a single argument for the *pop* function, so the *tag* is also carried with the couple of pointers into a `triple<const T*,const T*,Tag>`.

```
24  template<typename T,typename Tag,typename Parameter>
25  class MPI_InBlockComplexTagBuffer
26  : public InBufferModel<T,triple<const T*,const T*,Tag>,unsigned char> {
27  protected:
```

*An external Parameter for driving low level communication function*

```
28    Parameter *parameter;
29  public:
```

*A constructor referring to the top level class*

```
30    MPI_InBlockComplexTagBuffer(Parameter & _parameter,const unsigned _bufferSize)
31     : Model(_bufferSize), parameter(& _parameter) { resize(bufferSize); }
```

*This main function manages the un-serialization.*
*For performance reason, we avoid to use an additional buffer for un-serializing. However, we assume a non-growing size during the unpacking by MPI (with* MPI_Unpack*) (else we need an additional buffer and many copies).*

```
32    InOutType pop() {
33      assert(canDoIt());
34      unsigned data_count;
35      MPICHECK(MPI_Unpack(toPointer(buffer.begin()),buffer.size(),&position,
36                          &data_count,1,MPI_UNSIGNED,MPI_COMM_WORLD));
37      Tag tag;
38      MPICHECK(MPI_Unpack(toPointer(buffer.begin()),buffer.size(),&position,&tag,1,
39                          MPI_Object<Tag>::Type_Id,MPI_COMM_WORLD));
40      ObjectType *ptr = reinterpret_cast<ObjectType *>(toPointer(buffer.begin()));
41      MPICHECK(MPI_Unpack(toPointer(buffer.begin()),buffer.size(),&position,ptr,
42                          data_count,MPI_Object<ObjectType>::Type_Id,MPI_COMM_WORLD));
43      return InOutType(ptr,ptr+data_count,tag);
44    }
```

*The* sizeOf *function computes the size of a vector of data including tag and size description.*

```
45    unsigned sizeOf(const InOutType &value) const;
```

*The communication functions* start/stop *mix the two control parts* ***Parameter***, ***Lock*** *for communicating. This allows the end user to drive the communication functions (here via MPI) and the algorithm to know the state of the communication.*

```
46    template<typename Lock> void start(Lock &lock);
47    template<typename Lock> void stop(Lock &lock);
48  };
```

### A.3.3 MultiSender

The *MultiSender* algorithm provides multiple buffers of the same type for sending data. The scheduling is 'as soon as possible' for each buffer. The order may be not preserved but we have a larger capacity without waiting to fill every buffer before sending.

```
49   template <typename Container,typename MultiLock>
50   class MultiSender {
51   protected:
52     Container *currentBuffer;
```

*Internal function for cleaning the current (pointed by* currentBuffer*) buffer referring to* reset *function of the specific container*

```
53     void reset();
```

*A constructor including generic parameters for building vectors of Containers (*buffers*) and Locks (*multiLocks*)*

```
54   public:
55     template<typename Parameter1, typename Parameter2>
56     MultiSender(const unsigned _bufferSize,Parameter1 &param1, Parameter2 &param2);
```

*Accumulate a new value : if the current buffer is full, search another ready buffer, else wait.*
*This function may be written as two separated non-blocking parts for improving the performance :* test *for testing before and* fpush *for sending without any test. Moreover an* again *function can try to start the buffer with a* 'full enough ?' *a priori test ; this is optional for outgoing communications but essential for incoming communications : starting a reception* ***before*** *needing it.*

```
57     void push(const typename Container::InOutType &value) {
58       if (!test(value)) {
59         flush();
60         wait();
61       }
62       assert(currentBuffer->canDoIt(value));
63       currentBuffer->push(value);
64     }
```

*Functions for waiting/testing data of the current buffer.*
*The management of communications over a buffer is delegated to its lock.*

```
65     void wait();
66     bool test(const typename Container::InOutType &value) {
67       if (!currentBuffer)
68         if ((currentId = multiLocks.test_any()) != -1) {
69           currentBuffer = &buffers[currentId];
70           reset();
71         } else return false;
72       return currentBuffer->canDoIt(value);
73     }
```

*Send* ***now*** *the current (not empty) buffer, and invalidate it*

```
74      void flush() {
75        if (!currentBuffer || currentBuffer->empty()) return;
76        currentBuffer->start(multiLocks[currentId]);
77        currentBuffer = NULL;
78      }
```

*Stop all buffers by waiting the end of the communications.*
*(An additional* abort *function exists for killing communications before exiting)*

```
79      void end() {
80        flush();
81        multiLocks.wait_all();
82      }
83    };
```

### A.3.4 An example

This example shows how we use our library for writing overlapping programs. The goal is simply to exchange data (send/receive) between `ncpus` processes and to do local computations. This is the *transfer* step ; which has been simplified (no explicit computation are given).

The performance of this code will be described in last part of this article.

*Current EasyMSG I/O Type with a tag typed* `uaddr_t` *(universal address of a cluster)*

```
84      typedef triple<const BaseType *,const BaseType *,uaddr_t> InOutBufferType;
```

*The Receiver parameters (start the receiving communication soon as possible)*

```
85      MPI_Parameters pop_params;
86      pop_params.setTarget(MPI_ANY_SOURCE);
87      pop_params.setTag(TRANSFER_TAG);
88
89      MultiReceiver<
90        MPI_InBlockComplexTagBuffer<BaseType,uaddr_t,MPI_Parameters>,
91        MPI_MultiLocks<MPI_Parameters> > popper(TRANSFER_BUFFER_NUMBER,
92                                                TRANSFER_BUFFER_SIZE,
93                                                pop_params,pop_params);
94      popper.start();
```

*We use an array of senders (one for each target).*
*We want to send arrays of BaseType with a tag*(`uaddr_t`).

```
95      std::vector<MPI_Parameters> pushers_params(ncpus);
96      typedef MPI_OutBlockComplexTagBuffer<BaseType,uaddr_t,
97                                           MPI_Parameters> SenderContainer;
98       SenderArray<
99         SimpleSender<
100          SenderContainer,
101          MPI_SimpleLock<MPI_Parameters> > > pushers(ncpus,TRANSFER_BUFFER_SIZE,
102                                                    pushers_params);
103     for(int cpu=0;cpu < ncpus; ++cpu) {
```

```
104       pushers_params[cpu].setTarget(cpu);
105       pushers_params[cpu].setTag(TRANSFER_TAG);
106     }
```

*We assume the knowledge the number of expected requests (*expectedRequests*), the list of communications to send (*transferGroups*), and a list of local computations (*localProgresser*).*

```
107     unsigned expectedRequests = totalTransferRequests.sum();
108     for(int cpu=0;cpu<ncpus;++cpu) transferGroups[cpu].again();
109     TransferIterator localProgresser;
110
111     while(something_to_do(expectedRequests,transferGroups)) {
```

*Treat the incoming message from* `pop_params.getSource()`. *We have to call the* again *function in order to inform* popper *that we have successfully treated the incoming request and it can release* ***now*** *the buffer area allocated for this datum.*

```
112       if (popper.test()) {
113         do {
114           InOutBufferType val = popper.pop();
115           /* Treat the incoming request */
116           popper.again();
117         } while(popper.test());
118       }
```

*Send tagged vector* v *tagged by* tag *(its address) to the other processes while it can. We use a combination of* test *and* fpush *(faster than* push *after* test*), which avoid deadlock.*

```
119       for(int cpu=0;cpu<ncpus;++cpu)
120         while(!transferGroups[cpu].empty()) {
121           InOutBufferType val(v.begin(),v.end(),tag);
122           if (pushers[cpu].test(val))
123             pushers[cpu].fpush(val);
124         }
```

*Do a few local computations according* localProgresser, *where* /* how many ? */ *is a iteration limiter*

```
125       while(/* how many? */ && !localProgresser.end())
126         /* Do one local computation */
127     }
```

*Do remaining local computations*

```
128     while(!localProgresser.end())
129       /* Do one local computation */
```

*Wait for the end of the previously sent communications*

```
130     popper.abort();
131     pushers.end();
```

## A.4 Adaptive dynamic overlapping

### A.4.1 Context definition

Usually, the network traffic is non-linear, even unpredictable during high traffic load. However, many scientific programs do repetitive tasks such as in an iterative solver, and by iterating we accumulate informations about the context of execution for improving the average timing. We will illustrate a way to manage the network traffic in order to increase the overall speed of parts mixing communications and computations used into our parallel solver of Maxwell equations by Fast Multipole Method A.3.4.

Applying generic Algorithm A.2 needs to define "*do some local computations*". The main goal of our approach is to reduce the global execution time.

Complex communications (with many sources, targets, buffers) may be expensive to test for nothing, when no communication is ready. That is why increasing the *local computations* for each loop may reduce the overall execution time. However, if *local computations* are too long (or the granularity of *local computations* is too coarse), incoming/outgoing communications may wait to be treated, then the execution time may decrease.

Furthermore, the architecture and the load of the computers and the network may be variable between different machines and between different times. Then, defining this aggregation statically will never be optimal for every computer.

Our first step is to optimize the global time of a part of code by computing a more *optimal* aggregation parameter $p$.

### A.4.2 Explorative exponential search

The optimization may be done, by minimizing the *'execution time'* as a nearly convex function with a dichotomic method with a single starting point $p = 0$ (no aggregation). Usually, the dichotomy with a single left (resp. right) starting point, find a right (resp. left) bound by an exponential search. Then, a standard dichotomy can start with two starting points.

However, the *'execution time'* is not really a well-defined function : evaluated at several times with a same aggregation parameter $p$ does not give a stable value, especially if the evaluations are not consecutive. This 'function' is linked to the global state of the computer load (CPUs, network ...).

Thus, we need a very stable and responsive algorithm, if the 'optimum' changes during the optimization (the environment may change during execution).

Our main difference with a standard dichotomy is our *explorative exponential search*(Fig. A.3) is not based on placing one new *better* point between two other points, but placing one better point from the knowledge of one point and one vector of exploration : the length of this vector is increased when the next try is *better* ; the direction of the vector is changed and the length is reduced when the next try is *worse*.

**Remark** *A* better *point means a point that produces a more "near to optimum" point than previous points, all previous points for the dichotomic scheme, the previous one for our explorative scheme.*

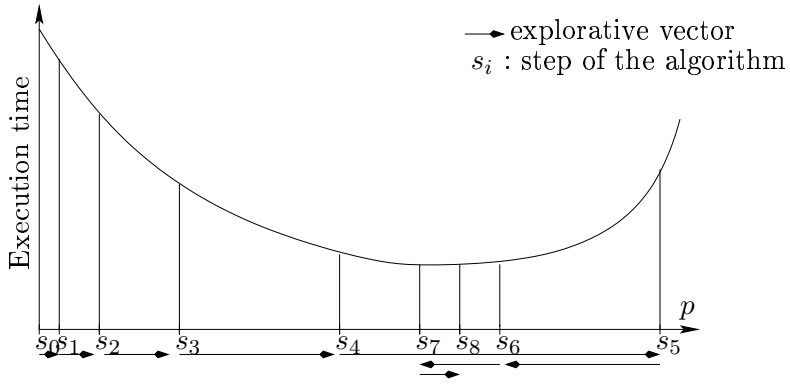



FIG. A.3 – Explorative exponential search scheme on an ideal smooth function

### A.4.3 Optimizing in a real world

The previous *explorative exponential search* algorithm is used at each execution of Algorithm A.2 for adapting the aggregation parameter $p$.

Some remarks for algorithmic improvements.

- The *execution time* is a very fuzzy function : even several consecutive measures with a same parameter $p$ may give different results. We use a mean of $n$ evaluations as reference values, *i.e.* we change the aggregation parameter $p$ every $n$ executions (currently, $n$ is 3 or 4).
- Our search domain for $p$ is bounded on the left by 0.
  The first explorative vector is always rightwards.
  If the algorithm tries to go beyond 0, we slow down the search by decreasing the length of the explorative vector so that $p > 0$.
- We interpret non integer value of $p$ as an aggregation parameter mean.
  $\forall p \in \mathbb{R}_+$, we aggregate $p_i = \lfloor p \cdot i - p_{i-1} \rfloor$ local computations at loop index $i \geq 1$ ($p_0 = 0$, $\lfloor \cdot \rfloor$ represents the integer part operator).
- If the length of the explorative vector is very low, the algorithm becomes less responsive to a significant modification of the global state. Moreover, the *execution time* is not well-defined and widely varying between several evaluations, so two too close parameters may be not significant : we bound the minimal length of the explorative vector.

## A.5 Computational results

We will study the efficiency of our optimization for adaptive overlapping on three problems of different sizes, all arising from the multipole method.

- Small : $N = 75,000$ degrees of freedom, 4MB of data to send divided into 700 vectors for each process ;
- Medium : $N = 300,000$ degrees of freedom, 14MB of data to send divided into 1300 vectors for each process ;
- Large : $N = 600,000$ degrees of freedom, 31MB of data to send divided into 2000 vectors for each process.

For each case the global number of transfers to compute is $O(N)$. We will use 8 CPUs Intel PIII/1266MHz-1GB for each computation and a switched Ethernet 100Mb network.

### A.5.1 Without our adaptive overlapping

Our first tests are without our adaptive overlapping but a simple static overlapping (1 local computation for each "*do some local computations*".
The following graphs (Figures A.4, A.5, A.6) show the execution times of a part of code by the 8 processors during 140 iterations for the three problem sizes : *Small*, *Medium*, *Large*.

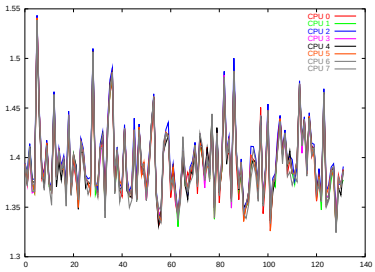



FIG. A.4 – *Small*

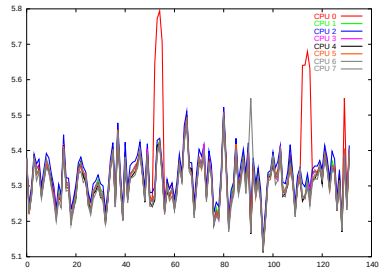



FIG. A.5 – *Medium*

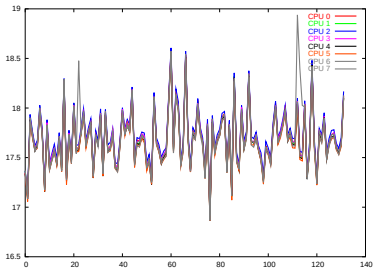



FIG. A.6 – *Large*

Execution timings without adaptive overlapping.

These measures are our reference execution times.

### A.5.2 With our adaptive overlapping

We will study the same problems with our algorithm by optimizing the aggregation parameter $p$ using our explorative exponential algorithm (part A.4.2). This method leads to a uniform decomposition of the local computations as fixed-size packets of local computations during the communication step. When there is no more communication, the main loop stops and does the remaining local computations.

We show the execution times and the 'convergence' of the aggregation parameters for each problem.

According to these graphs, we can see independent timings and convergences for each process, even if our homogeneous architecture provides very close values. The algorithm starts with the worst value (when $p = 0$) and try to reduce the average timing; at the second iteration $p = 1$ and we obtain the same execution timing as the references cases (see section A.5.1). This parameter $p$ grows exponentially up to obtain a execution timing worst than the previous result.

We obtain a more "optimal" timing (about 18% faster) after about 20 evaluations (*i.e.* 5 updates of $p$), and the convergence of $p$ is near to be stabilized after 15 updates (*i.e.* 60 evaluations) (Figs. A.7-A.12)

The number of updates before giving a good result, even with a nearly chaotic function, is really the main goal for our algorithm without falling in a local optimum; many other classical algorithms, using very smooth functions, are really inefficient here.

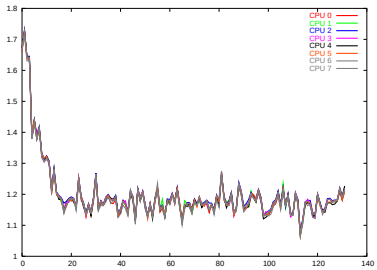



FIG. A.7 – *Small*

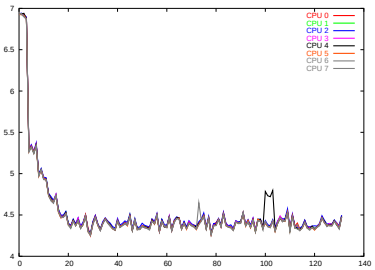



FIG. A.8 – *Medium*

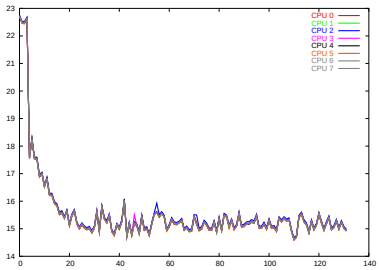



FIG. A.9 – *Large*

Execution timings with adaptive overlapping

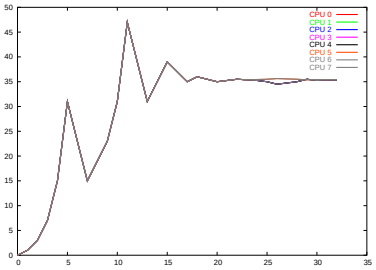



FIG. A.10 – *Small*

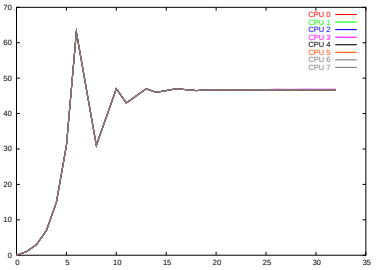



FIG. A.11 – *Medium*

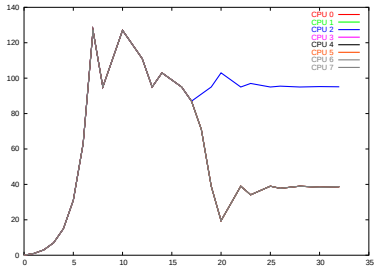



FIG. A.12 – *Large*

Aggregation parameter $p$ convergence ($p$ is updated after an average of 4 evaluations)

### A.5.3 With a more responsive adaptive overlapping

The previous case uses a constant aggregation parameter for the aggregation of local computations in the main loop, and updates it between each full execution. However, the network traffic is not homogeneous during one whole execution : at the beginning, many processes start their communications and there are many communications to treat. Nevertheless, our previous implementation was designed for treating as a priority the communications, so performing as a special case the inhomogeneous communications may not improve the global execution time in the same context.

However, our previous test was done in an ideal network without any other traffic. In many cases, a shared cluster may produce a very inhomogeneous traffic without any foreseeable behavior. That is why we will compare our previous algorithm with another one with a better responsiveness to the unstable environments.

This new test uses the same algorithm for optimizing the execution time but the aggregation parameter is replaced by a variable aggregation based on an increasing evolution when there is no communication (according to the last communication tests), and a fast decreasing evolution when there are communications : we will call it *fast adaptive algorithm*.

In fact, we use the following heuristic : if the previous test says "there are some incoming/outgoing data", the associated buffers were full, so there may remain some incoming/outgoing data for the same buffers, then we want to test them as early as possible

and do less local computations; else we can do more local computations because if the last test fails and this may be a indicator of a lower network activity.

The following graphs (Figures A.13, A.14) show the same computations with a stressed network (an additional program providing an irregular high traffic on the cluster network)

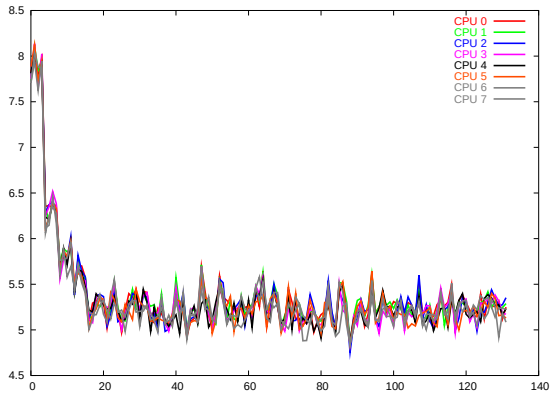



FIG. A.13 – Original $p$ aggregation

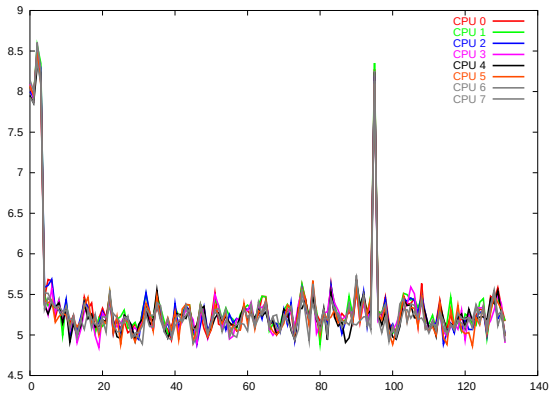



FIG. A.14 – New Adaptive aggregation

Execution timings **with** manual communication aggregation

Even if the mean of timings is better for our fast adaptive algorithm, these results are not so significant. In fact, the previous algorithm was already written for aggregating communications (a loop try to fill/flush the current buffer). But, if we remove this optimization, we can see the effect of the last algorithm against a manual optimization by the programmer.

**Remark** *The* new *algorithm converges faster to the solution, because it is more adaptive to the environment (the aggregation parameter grows when there are few communications) and its evolution parameter of the adaptive communication aggregation is smaller, so it needs fewer steps of exponential search for finding good candidates.*

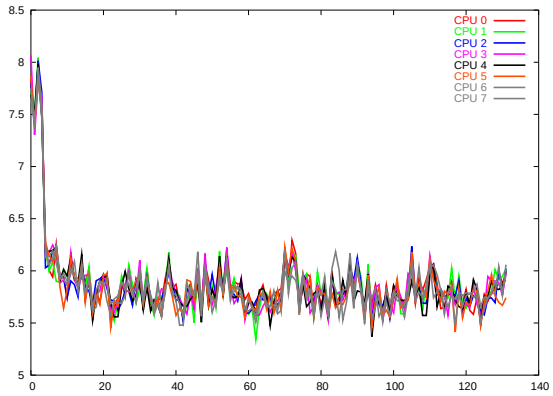



FIG. A.15 – Original $p$ aggregation

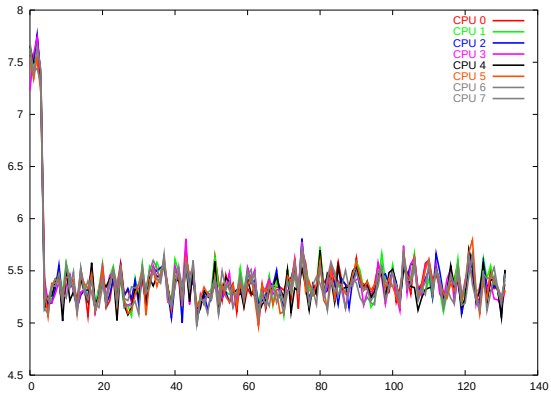



FIG. A.16 – New Adaptive aggregation

Execution timings **without** manual communication aggregation

The last results show a real better efficiency (about 10%) for the fast adaptive algorithm : it tries itself to aggregate the communications or increases the local communications according to the network traffic. However, these results are less efficient than a manually optimized version because the manual tuning was done in order to avoid local computations between consecutive communication tests, which is our final goal : the algorithm understands how reducing the execution timings.

### A.5.4 Low speed CPU environment

Finally, we will test the adaptability on a different CPU environment. Here, we use the same network but with 8 PIII-800MHz processors.

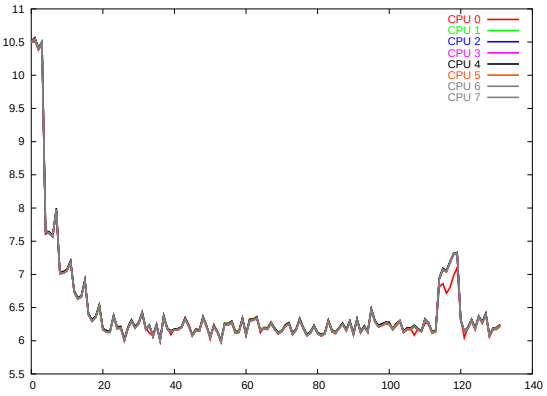



FIG. A.17 – Execution timing

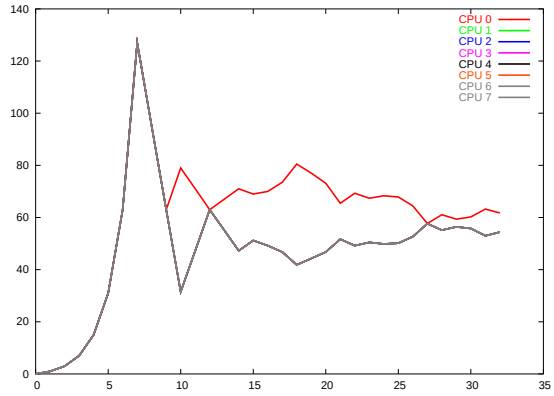



FIG. A.18 – Aggregation parameter

As planned, the algorithm converges towards another parameters without any help (Figures A.15-A.18). This is the first motivation for using these algorithms : the tuning is automatically done.

## A.6 Conclusion

This adaptive overlapping method with our *EasyMSG* communication library over MPI provides an improved efficiency even for codes already optimized by statically defined overlapping. Our tests show a performance up to 20% better into a massively communicant program as the transfer step of the Fast Multipole Algorithm applied to High Frequency Maxwell Equations. Moreover, we have seen another type of optimization with the fast adaptive algorithm, where the inner loop is also optimized on the fly according to the network traffic. This last algorithm may be more efficient than the first algorithm when the programmer did not aggregate explicitly the communications as long as there are, even with more unstable computing environment : the algorithm tries to aggregate dynamically communications (*i.e.* with less local computations between communications).

However, these algorithms use a too competitive optimization method between processes (each process tries to optimize according its own criterions), where a more cooperative method (using global criterions) may be more efficient (for the global execution timing, not the convergence rate).

There are many other small parameters to tune for improving the results (as the buffer size), but they are very difficult to optimize with few evaluations (about 20) and *measuring* the

environment is also a part of the solution (measuring the bandwidth may provide a optimal buffer size for a given transfer time).

Furthermore, the convergence to an other "optimal" parameter may be improved when the environment changes during the computations. Our current implementation needs many steps to adapt again the new *optimal* parameter by increasing the explorative vector, even with a low bound on this parameter.

Future improvements trend towards optimizing communicant codes with few efforts.

# Annexe B

# i*Warp*


térer sur des structures est sûrement un concept très simple, souvent utilisé, mais dont la pratique présente des situations complexes. Tout d'abord, j'appelle «itérer sur une structure» le fait de parcourir les éléments de cette structure suivant un ordre et éventuellement des contraintes du type *sélection* ou *transformation*.

Le code informatique produit dans le cadre de cette thèse présente une grande variété d'algorithme ; beaucoup d'entre eux s'appuient sur des structures récursives, arborescente ou multiniveaux (méthodes de tris, FMM, FFT). L'efficacité de ces méthodes comportent souvent un contre-coût en complexité des structures sous-jacentes.

Regardons le cas de l'arbre. Nous pouvons vouloir, le parcours en profondeur (traitement du noeud après ses enfants), superficiel (traitement du noeud avant ses enfants), des feuilles, par niveaux ...

A tous ces type de parcours, on peut vouloir y adapter une sélection, *i.e.* parcourir l'ensemble des cellules vérifiant une propriété, ou bien une transformation, *i.e.* voir les cellules parcourues sous un certain aspect, par exemple une propriété de la cellule.

Orthogonalement, on peut aussi vouloir parcourir les structures tout en permettant de les modifier ou non.

La bibliothèque C++ i*Warp* est une réponse à tous ses besoins. Le concept avait déjà été développé dans la Standart Template Library (STL) du C++, bien plus assimilables à une généralisation du concept de pointeur. Les itérateurs STL consistent en des objets itérant sur les containers STL (`vector, list, map, set ...`) formé par les containers eux-mêmes via des méthodes nommées `begin` et `end`, décrivant respectivement le début et la fin d'un container.

```
1     for(Container::iterator i = container.begin();
2         i != container.end(); ++i) {
3       // computations on *i, current pointed value
4     }
```

Cette notion de pointeur permet une modifications des extrémités comme

```
5     for(Container::iterator i = container.begin()+1;
6         i != container.end()-2; ++i) {
7       // computations on *i, current pointed value
8     }
```

mais la sélection et la transformation fond partie du corps d'instructions

```
 9    for(Container::iterator i = container.begin();
10        i != container.end(); ++i) {
11       if (select(*i)) {
12         // computations on transform(*i), current pointed value
13      }
14    }
```

Cette notion permet déjà de se détacher du type de container au niveau de l'algorithme en manipulant de la même manière un vecteur ou une liste chaînée.

Cependant, il n'est pas possible d'itérer sur des structures emboîtées avec un seul itérateur, et nous sommes contraint d'écrire

```
15    for(Container<SubContainer>::iterator i = container.begin();
16        i != container.end(); ++i) {
17      for(SubContainer::iterator j = i->begin();
18          j != i->end(); ++j) {
19            // computations on *j, current pointed value
20      }
21    }
```

ce qui, mathématiquement parlant, n'est rien d'autre qu'une composition de structures et donc a priori d'itérateurs.

C'est cette notion de composition que met en avant la bibliothèque i*Warp* . La composition d'opérateurs d'itération suppose que ses objets sont simples et non décomposés en plusieurs morceaux, comme les itérateurs de la STL formés par un début (`begin`) et constamment comparés à une fin (`end`). Un itérateur i*Warp* est une entité construite sur un objet à itérer, contenant lui-même la connaissance du début et de la fin, ce qui permet d'interrompre le parcours à tout instant et de la reprendre par la suite, même pour des structures complexes, imbriquées ou arborescentes.

De manière générique, cela conduit à écrire

```
22    for(iWarpIterator<NotConstType,Container> i(container); !i.end(); ++i) {
23      // computations on *i, current pointed value
24    }
```

Les itérateurs étant construits au dessus de la structure, il convient d'en avoir au moins autant que de structures. Toutefois, grâce à la généricité permise par l'utilisation des modèles (`template`'s) du C++.

Ainsi il existe déjà, des modèles permettant l'itération

- de structures STL ou assimilées (`STLWarpper`) disposant de méthode `begin` et `end`,
- de structures indexées (`IndexedWarpper`) disposant d'une méthode `size`,
- de structures dynamiquement indexées (`DynamicIndexedWarpper`) où l'on suppose que la plage d'indices est donnée à la construction de l'itérateurs
- de structures statiquement indexées (`StaticIndexedWarpper`) où l'on suppose que la plage d'indices est connue à la compilation de l'itérateur

mais aussi de structures plus complexes comme les arbres (`LeafTreeWarpper`, `DeepTreeWarpper`, `ShallowTreeWarpper`) qui ressemblent plus à des itérateurs auto-composés à profondeur non

définie sur des noeuds d'un arbre, où chaque noeud «connaît» ses enfants dans un sous-container.

À cela s'ajoute les `Adapter`'s fournissant une autre vue d'un élément `pointée` par un itérateur, ainsi que les `Selector`'s permettant d'éviter d'arrêter l'itérateur sur des éléments indésirables (suivant un certain critère).

Tout cela est construit autour de trois classes principales `iWarpModel` pour toutes les formes d'itérateurs, `iWarpAdapt` pour les `Selector`'s et les `Adapter`'s construits au dessus des itérateurs, et `iWarpCompose` permettant de composer des deux itérateurs entre eux. Toutes ces classes sont construites via polymorphisme statique[76] assurant l'efficacité du code générer pour le traitement des itérateurs.

*Classe modèle de tous itérateurs*

```
template<typename _InType,typename _OutType,typename _isConst,typename _SelfType>
class iWarpModel {
protected :
#ifndef NDEBUG
  bool initialized;
  bool canDeference;
#endif

public :
  //! is it a constant iterator ?
  typedef _isConst isConst;

  //! Structure type which will be iterated
  typedef typename ConditionalConst<_InType,_isConst>::Type  InType;

  //! Element type, may be converted by an Adapter
  typedef typename ConditionalConst<_OutType,_isConst>::Type OutType;

  //! Self type
  typedef _SelfType SelfType;

protected :
  //! Static polymorphism tools
  SelfType &self() { return static_cast<SelfType&>(*this); }
  const SelfType &self() const { return static_cast<const SelfType&>(*this); }
  //! Type of mother class
  typedef iWarpModel<InType,OutType,isConst,SelfType> Mother;

  iWarpModel() { IFDEBUG(initialized = canDeference = false); }
public :
  SelfType &operator=(InType &in) {
    IFDEBUG(initialized = true);
    self().set(in);
    IFDEBUG(canDeference =!end());
    return self();
```

```
  }

  SelfType &operator=(const SelfType &it) {
    IFDEBUG(initialized = it.initialized);
    IFDEBUG(canDeference = it.canDeference);
    self().set(it);
    return self();
  }

  OutType &operator*() const { assert(canDeference); return self().get(); }

  OutType *operator->() const { return &(operator*()); }

  SelfType &operator++() {
    assert(initialized && canDeference);
    self().step();
    IFDEBUG(canDeference =!end());
    return self();
  }

  bool end() const { assert(initialized); return self().isEnd(); }

  bool operator==(const SelfType &it) const {
    assert(initialized);
    assert(it.initialized);
    assert(object == it.object);
    return self().isEqual(it);
  }

  bool operator!=(const SelfType &it) const { return !(self() == it); }
};
```

Comme on le voit, cette structure délègue beaucoup à sa classe mère (`set`, `get`, `step`, `isEqual`) et n'est là que pour assurer la confirmé de l'interface sans passer par de coûteuses fonctions virtuelles. La généricité réduit le code nécessaire au développement des itérateurs, mais il convient de toujours s'adapter au pire des cas qui en matière de code généré est la méconnaissance de certains paramètres à la compilation, ce qui ne permet pas d'utiliser à fond des techniques proches de *expression templates* ; en effet certains itérateurs peuvent avoir un comportement définie à l'exécution, ce qui implique d'appeler la classe mère via des appels de type fonctions (incluant un contexte d'exécution).

*Exemple de codes utilisant iWarp*

```
// Les structures de base
typedef std::vector<int> Vector;
struct Test { Vector champs; };
typedef std::valarray<Test> Structure;

// Adapteur de vue permettant de voir le champs 'champs' d'un élément itéré
```

```
template<typename _Iterator, typename _isConst = typename _Iterator::isConst>
class TestAdapt
 : public iWarpAdapt<_Iterator,Vector,TestAdapt<_Iterator,_isConst>,_isConst> {
public:
  typedef iWarpAdapt<_Iterator,Vector,TestAdapt<_Iterator,_isConst>,_isConst> Mother;
  typedef typename Mother::Iterator Iterator;
  typedef typename Mother::InType    InType;
  typedef typename Mother::OutType   OutType;
  typedef typename Mother::SelfType SelfType;
  typedef typename Mother::isConst   isConst;
  friend class Mother;
private:
  OutType & adapt(const Iterator & it) const { return (*it).champs; }
  void select(Iterator & it) { }
public:
  SelfType & operator=(InType   & in)            { return Mother::operator=(in); }
  SelfType & operator=(const SelfType & it)  { return Mother::operator=(it); }
  SelfType & operator=(const Iterator & it)  { return Mother::operator=(it); }
};

// Fonction d'accumulation des valeurs fournies par un itérateur
template<typename Iterator>
int accumul(const Iterator &_K) {
  int val = 0;
  for(Iterator K = _K;!K.end();++K) val += *K;
  return val;
}

int main(void) {
  Structure object(20);
  // ... remplissage de object

  // Les itérateurs utilisés
  typedef StaticIndexedWarpper<NotConstType,Structure,Test,StaticRange<3,12> > OuterIterator;
  typedef TestAdapt<OuterIterator> AdaptedOuterIterator;
  typedef STLWarpper<NotConstType,Vector> InnerIterator;
  typedef iWarpCompose<AdaptedOuterIterator,InnerIterator> CompositeIterator;
  CompositeIterator I;

  // Calculs de la somme sur la structure composée limitée à object[3..12]
  say() << "Summation : "
        << accumul((AdaptedOuterIterator() * InnerIterator()) = object) << "\n";
  say() << "Summation : "
        << accumul(CompositeIterator() = object) << "\n";
  say() << "Summation : "
        << accumul(I = object) << "\n";
```

```
  // Parcours de chaîne de caractères
  char *str = "abcdefghij";
  DynamicIndexedWarpper<ConstType,char *,char> i = DynamicRange(2,10);
  for(i = str; !i.end();++i)
    std::cout << *i << "\n";
}
```

Pour conclure, je vous présente un extrait de code proposant la définition d'un nouvel itérateur dont l'objectif est de permettre un parcours des noeuds de l'arbre FMM, suivant un ordre «en profondeur d'abord» et conditionné par un prédicat de sélection `Predicat` (qui répond à la question «cette cellule est-elle intéressante») et un prédicat d'accès `View` (qui répond à la question «cette cellule de l'arborescence est-elle accessible»).

La conjonction de ces possibilités permet de traiter l'étape d'agrégation qui requiert un parcours des noeuds en partant des feuilles, mais conditionné à la nécessité d'effectuer un agrégation dans le cas SPW-FMM (*cf* section 5.3.1).

```
//! Itérateur générique
/*! Permet entre autre de fournir une base commune qui maximise le code commun et
 *  et autorise des affectations entre itérateurs de type différents */
template<typename isConst, bool initStep>
class CommonIterator {
private :
  //! Contrôle d'initialisation ['cause iWarp copy]
  IFDEBUG_INIT(bool _initialized);
protected :
  //! Définition conditionel du const
  typedef typename ConditionalConst<Node,isConst>::Type NodeType;

  //! Pile des Noeuds parcourus
  NodeType * _nodeStack[MaxLevel+1];

  //! Pile des prochains fils à parcourir associés aux noeuds
  unsigned _sonStack[MaxLevel+1];

  //! Taille sourante des piles
  unsigned topLevel;

protected :
  //! Constructeur par défaut
  CommonIterator() {
    topLevel = -1; push(NULL); assert(topLevel == 0);
    IFDEBUG(_initialized = true);
  }

  //! Opérateur d'initialisation de l'itérateur
  void set(NodeType & node) {
    assert(_initialized);
```

```
    push(&node);
    if (initStep) step();
  }

  //! Opérateur de copie sur itérateur
  template<typename _isConst_, bool _initStep_>
  void set(const CommonIterator<_isConst_,_initStep_> & li) {
    assert(_initialized); assert(li._initialized);
    topLevel = li.topLevel;
    for(unsigned i=0;i<=topLevel;++i) {
      _nodeStack[i] = li._nodeStack[i];
      _sonStack[i]  = li._sonStack[i];
    }
    // NE PAS Faire step() sur un itérateur déjà initialisé, sinon on perd un élément
  }

  //! Déréférencement de l'itérateur
  NodeType & get() const { return *_nodeStack[topLevel]; }

  //! Teste de fin de parcours
  bool isEnd() const { return (topLevel==0); }

  //! Teste d'équivalence entre itérateurs
  template<typename _isConst, bool _initStep>
  bool isEqual(const CommonIterator<_isConst,_initStep> & it) const {
    if (topLevel != it.topLevel) return false;
    for(unsigned i=0;i<=topLevel;++i)
      if (_nodeStack[i] != it._nodeStack[i]
          || _sonStack[i] != _sonStack[i]) return false;
    return true;
  }

  //! Fonction virtuelle d'avance de l'itérateur
  /*! Elle n'est utilisée en virtuelle QUE pour la construction à partir d'un nouveau noeud */
  virtual void step() = 0;
protected :
  //! Insertion d'un nouvel élément en pile [internal]
  inline void push(NodeType * const node) {
    ++topLevel;
    assert(topLevel <= MaxLevel);
    _nodeStack[topLevel] = node;
    _sonStack[topLevel]  = 0;
  }

public :
  //! Donne la profondeur du noeud courant
  /*! 0 => arbre vide */
```

```
  unsigned depth() const { return topLevel; }
};


//! Itérateur générique en profondeur d'abord
/*! S'arrête sur tous les noeuds jusqu'aux feuilles,
 *  la 'profondeur d'abord' donne l'impression que cela part des feuilles */
template<typename _isConst, typename Predicate, typename View = LocalView>
class PredicateDeepIterator :
  public CommonIterator<_isConst,true>,
  public iWarpModel<Node, Node, _isConst,
                    PredicateDeepIterator<_isConst, Predicate, View> > {
public :
  typedef iWarpModel<Node, Node, _isConst,
                     PredicateDeepIterator<_isConst, Predicate, View> > Mother;
  typedef typename Mother::InType    InType;
  typedef typename Mother::OutType   OutType;
  typedef typename Mother::SelfType  SelfType;
  typedef typename Mother::isConst   isConst;
  friend class Mother;
private :
  typedef CommonIterator<isConst,true> Common;
  Predicate predicate;
public :
  //! Constructeur par défaut
  PredicateDeepIterator(const Predicate & p = Predicate())
: Common(), predicate(p) { }

  //! Constructeur à partir d'un arbre
  PredicateDeepIterator(const Tree & tree, const Predicate & p = Predicate())
: Common(), predicate(p) {
    Mother::operator=(*tree._root);
  }

  //! Constructeur à partir d'un noeud
  PredicateDeepIterator(InType & root, const Predicate & p = Predicate())
: Common(), predicate(p) {
    Mother::operator=(root);
  }

  SelfType & operator=(InType & root) {
    return Mother::operator=(root);
  }

  //! Constructeur par copie
  template<typename _isConst_, bool _initStep_>
  SelfType & operator=(const CommonIterator<_isConst_,_initStep_> & it) {
```

```
    return Mother::operator=(it);
  }

  //! Accesseur au prédicat
  const Predicate & getPredicate() const { return predicate; }
  Predicate & getPredicate() { return predicate; }
private:
  //! Avancer effective de l'itérateur [internal]
  void step();
};


template<typename isConst, typename Predicate, typename View>
void PredicateDeepIterator<isConst,Predicate,View>::step() {

  { // Préparation très locale
    assert(topLevel > 0);
    const unsigned & refSon = _sonStack[topLevel];
    // Sauf dans le cas initial, c'est une pause
    assert((topLevel == 1 && refSon == 0) || refSon == MaxSons);
    if (refSon == MaxSons)
      --topLevel; // Dépilement
  }

  while(topLevel > 0) {
    unsigned & refSon = _sonStack[topLevel];
    const OutType * refNode = _nodeStack[topLevel];

    if (refSon == MaxSons || predicate.isLeaf(Const(*refNode))) { // Une feuille
      refSon = MaxSons; // will be pop next time [un peu inutile qq refSon==MaxSons]
      return;
    }

    while(refSon < MaxSons) {
      const Link & son = refNode->getSon(refSon);
      ++refSon; // next Son, when we'll go back
      if (View::isViewable(son)) { // On ne parcours que l'arbre visible
        push(&*son);
        break;
      }
    }
  }
}
```

# Annexe C

# Formules autour des polynômes de Legendre

**Legendre Formulae**

These formulae are available in [80] or can be easily established. $P_n$, $P_l^m$, $P_l^{-m}$ are defined for $n \geq 0$, $l \geq 0$, $m \geq 0$ and $x \in [-1, 1]$.

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2 - 1)^n \tag{C.1}$$

$$P_n(1) = 1 \tag{C.2}$$

$$P_n(-1) = (-1)^n \tag{C.3}$$

$$P_n(0) = \begin{cases} 0 & n : \text{odd} \\ (-1)^{\frac{n}{2}} \frac{1 \cdot 3 \cdot 5 \cdot \ldots \cdot (n-1)}{2 \cdot 4 \cdot 6 \cdot \ldots \cdot n} & n : \text{even} \end{cases} \tag{C.4}$$

$$\int_{-1}^{1} P_l(x) P_m(x) \, \mathrm{d}x = \frac{2}{2n+1} \delta_{lm} \tag{C.5}$$

$$(n+1) P_{n+1}(x) = (2n+1) x P_n(x) - n P_{n-1}(x) \tag{C.6}$$

$$(2n+1) P_n(x) = P'_{n+1}(x) - P'_{n-1}(x) \tag{C.7}$$

$$(x^2 - 1) P'_n(x) = n x P_n(x) - n P_{n-1}(x) \tag{C.8}$$

$$P'_{n-1}(x) = x P'_n(x) - n P_n(x) \tag{C.9}$$

$$P'_{n+1}(x) = x P'_n(x) + (n+1) P_n(x) \tag{C.10}$$

$$(1 - x^2) P''_n(x) - 2x P_n(x) + n(n+1) P_n(x) = 0 \tag{C.11}$$

$$P_l(\hat{x}\cdot\hat{y}) = \frac{4\pi}{2l+1}\sum_{m=-l}^{l}\overline{Y_l^m}(\hat{x})\,Y_l^m(\hat{y}) \tag{C.12}$$

$$P_l^m(x) = (-1)^m\,(1-x^2)^{\frac{m}{2}}\frac{d^m}{dx^m}P_l(x) \tag{C.13}$$

$$P_n^{-m}(x) = (-1)^m\frac{(n-m)!}{(n+m)!}P_n^m(x) \tag{C.14}$$

$$(2l+1)xP_l^m(x) = (l+m)P_{l-1}^m(x) + (l-m+1)P_{l+1}^m(x) \tag{C.15}$$

$$P_m^m(x) = (-1)^m\frac{(2m)!}{2^m m!}(1-x^2)^{\frac{m}{2}} \tag{C.16}$$

# Annexe D

# Fonctions de transfert propagatives par projections harmoniques

For any radiation function $f$, our goal is to compute accurately

$$\int_{S^{z^+}} f(\sigma) e^{\imath\kappa<\sigma,x>} \, \mathrm{d}\sigma \tag{D.1}$$

The problem is that the function $\mathbf{1}_{S^{z^+}} e^{\imath\kappa<\sigma,x>}$ is very hard to integrate, due to the discontinuity at $z = 0$. In order to reduce the number of quadrature points, we will smooth this function by removing high frequencies.

We use the following axes decomposition :

$$\begin{cases} x(\theta,\phi) = \cos\phi \sin\theta \\ y(\theta,\phi) = \sin\phi \sin\theta \\ z(\theta,\phi) = \cos\theta \end{cases} \tag{D.2}$$

Then

$$\begin{aligned} \cos(\sigma, X) &= \frac{\sin\theta \ (x\cos\phi + y\sin\phi) + z\cos\theta}{|X|} \\ &= r_1 \sin\theta \cos(\phi + \phi_0) + r_2 \cos\theta \end{aligned} \tag{D.3}$$

where

$$r_1 = \frac{\sqrt{x^2+y^2}}{|X|}, \ \cos\phi_0 = \frac{x}{\sqrt{x^2+y^2}}, \ \sin\phi_0 = \frac{-y}{\sqrt{x^2+y^2}}, \ r_2 = \frac{z}{|X|} \tag{D.4}$$

By applying theorem 10, we can see that the Analytic Harmonic Transform (AHT) of $\mathbf{1}_{S^{z^+}} e^{\imath\kappa<\sigma,x>}$ can be obtained by computing the AHT $\mathcal{H}(f)^p_{lm}$ of $P_p(\cos(\sigma, X)) \ \mathbf{1}_{S^{z^+}}$ ; we will compute for $m \geq 0$

$$\mathcal{H}(f)^p_{lm} = \int_{S^{z^+}} P_p(\cos(\sigma, X)) \ P_l^m(\cos\theta) \ e^{-\imath m\phi} \, \mathrm{d}\sigma \tag{D.5}$$

where we assume $\sigma = (\theta, \phi)$ and $d\sigma = \sin\theta \ \mathrm{d}\theta \, \mathrm{d}\phi$.

**Remark** *We have removed some constants in order to simplify the final result. Using the coefficients $C_l^m$ produces more expensive computation, many square roots. The case $m \leq 0$ will be treated later.*

Finally, we will obtain

$$\mathcal{H}(f)_{lm} = \sqrt{\frac{2l+1}{4\pi}} C_l^m \sum_{p=0}^{\infty} (2p+1) \imath^p j_p(\kappa|x|) \mathcal{H}(f)_{lm}^p \tag{D.6}$$

## Number of terms in summation

The decrease of first kind Bessel functions $j_l$ follows theorem E.5 :

$$|j_l(v)| \leq \sqrt{\frac{e}{2}} \frac{(e|v|)^l}{(2l+1)^{l+1}}$$

for any $l \geq 0$ and any argument $v$.

Joined to a Schwarzian decomposition and the orthonormality of $Y_{lm}$

$$\begin{aligned}
\sqrt{\frac{2l+1}{4\pi}}\, C_l^m |\mathcal{H}(f)_{lm}^p| &= \left| \int_{S^2} P_p(\sigma \cdot \hat{X})) \cdot \mathbf{1}_{S^{z+}}\, Y_{lm}^*(\sigma)\, \mathrm{d}\sigma \right| \\
&\leq \int_{S^{z+}} |Y_{lm}(\sigma)|\, \mathrm{d}\sigma \\
&\leq \sqrt{2\pi}
\end{aligned}$$

We have $|\mathcal{H}(f)_{lm}| \leq \sqrt{2\pi}$

With equation D.6, we can deduce that the bandwidth of $\mathbf{1}_{S^{z+}} e^{\imath\kappa<\sigma,x>}$ in the Spherical Harmonics space is $O(|\kappa x|)$, because of the fast decrease of $j_l$ functions from $p \sim \frac{e|\kappa x|-1}{2}$. Any accuracy $\epsilon$ can be easily satisfied.

## Recurrence Formula

We will search a formula in order to obtain $\mathcal{H}(f)_{lm}^p$. Many ways are available, but we have chosen to search it as a "linear" recurrence formula.

Let $m > 0$ and

$$\mathcal{H}(f)_{lm}^{p+1} = \int_{S^{z+}} P_{p+1}(\cos(\sigma, X))\, P_l^m(\cos\theta)\, e^{-\imath m\phi}\, \mathrm{d}\sigma$$

by recurrence formula (C.6) and the decomposition (D.3) we have

$$P_{p+1}(\cos(\sigma, X)) = \frac{2p+1}{p+1}\big(r_1 \sin\theta \cos(\phi+\phi_0) + r_2 \cos\theta\big) P_p(\cos(\sigma, X)) - \frac{p}{p+1} P_{p-1}(\cos(\sigma, X))$$

Then, we have

$$\mathcal{H}(f)_{lm}^{p+1} = \frac{2p+1}{p+1}\big(r_1 \mathcal{H}(a)_{lm}^p + r_2 \mathcal{H}(b)_{lm}^p\big) - \frac{p}{p+1} \mathcal{H}(f)_{lm}^{p-1}$$

where

$$\mathcal{H}(a)^p_{lm} = \int_{S^{z^+}} a_{lm}(\sigma)\ P_p(\cos(\sigma, X))\ \mathrm{d}\sigma$$
$$\mathcal{H}(b)^p_{lm} = \int_{S^{z^+}} b_{lm}(\sigma)\ P_p(\cos(\sigma, X))\ \mathrm{d}\sigma$$

with

$$a_{lm}(\sigma) = \sin\theta \cos(\phi + \phi_0)\ P_l^m(\cos\theta)\ e^{-\imath m\phi}$$
$$b_{lm}(\sigma) = \cos\theta\ P_l^m(\cos\theta)\ e^{-\imath m\phi}$$

## Computation of $\mathcal{H}(b)^p_{lm}$

By recurrence on $P_l^m$ on $l$ (C.15) we have

$$(2l+1)\cos\theta\ P_l^m(\cos\theta) = (l+m)P_{l-1}^m(\cos\theta) + (l-m+1)P_{l+1}^m(\cos\theta)$$

so that

$$\mathcal{H}(b)^p_{lm} = \frac{l+m}{2l+1}\,\mathcal{H}(f)^p_{l-1,m} + \frac{l-m+1}{2l+1}\,\mathcal{H}(f)^p_{l+1,m}$$

## Computation of $\mathcal{H}(a)^p_{lm}$

First, we have

$$\cos(\phi+\phi_0) = \frac{e^{\imath\phi_0}}{2}e^{\imath\phi} + \frac{e^{-\imath\phi_0}}{2}e^{-\imath\phi}$$

Then,

$$a_{lm}(\sigma) = \frac{e^{\imath\phi_0}}{2}\sin\theta\ P_l^m(\cos\theta)e^{-\imath(m-1)\phi} + \frac{e^{-\imath\phi_0}}{2}\sin\theta\ P_l^m(\cos\theta)e^{-\imath(m+1)\phi}$$

If we want to write $\mathcal{H}(a)^p_{lm}$ as a "linear" combination of $\mathcal{H}(f)^p_{l_i m_i}$, we have to write $\sin\theta\ P_l^m(\cos\theta)$ as two "linear" combinations of $P_{l_i}^{m+1}$ and $P_{l_i}^{m-1}$.

– by (C.7) and the definition of $P_l^m$, we have

$$\begin{aligned}(2l+1)\sin\theta P_l^m(\cos\theta) &= (1-\cos^2\theta)^{\frac{1}{2}}(-1)^m(1-\cos^2\theta)^{\frac{m}{2}}\frac{d^m}{dx^m}\big(P'_{l+1} - P'_{l-1}\big)(\cos\theta)\\ &= (-1)^{m+1}(1-\cos^2\theta)^{\frac{m+1}{2}}\frac{d^{m+1}}{dx^{m+1}}\big(P_{l-1} - P_{l+1}\big)(\cos\theta)\\ &= P_{l-1}^{m+1}(\cos\theta) - P_{l+1}^{m+1}(\cos\theta)\end{aligned}$$

– **Lemme 28** *$\forall m \geq 0$ and $\forall n > 0$, we have*

$$(2n+1)xP_n^{(m)} = (n+1-m)P_{n+1}^{(m)} + (n+m)P_{n-1}^{(m)} \tag{D.7}$$

**Preuve** *By (C.6), we have the result for $m = 0$.*
*Then by derivating (D.7)*

$$\Big((2n+1)xP_n^{(m)}\Big)' = (n+1-m)P_{n+1}^{(m+1)} + (n+m)P_{n-1}^{(m+1)}$$
$$(2n+1)xP_n^{(m+1)} + (2n+1)P_n^{(m)} = (n+1-m)P_{n+1}^{(m+1)} + (n+m)P_{n-1}^{(m+1)}$$

*With the relation (C.7) valid even by derivating (linearity of the relation)*

$$\begin{aligned}(2n+1)xP_n^{(m+1)} &= -(2n+1)P_n^{(m)} + (n+1-m)P_{n+1}^{(m+1)} + (n+m)P_{n-1}^{(m+1)} \\ &= -\big(P_{n+1}^{(m+1)} - P_{n-1}^{(m+1)}\big) + (n+1-m)P_{n+1}^{(m+1)} + (n+m)P_{n-1}^{(m+1)} \\ &= (n+1-(m+1))P_{n+1}^{(m+1)} + (n+(m+1))P_{n-1}^{(m+1)}\end{aligned}$$

□

**Lemme 29** *$\forall m \geq 0$ and $\forall n > 0$, we have*

$$(2n+1)(1-x^2)P_n^{(m+1)} = \beta_n^m P_{n-1}^{(m)} - \alpha_n^m P_{n+1}^{(m)} \tag{D.8}$$

*where $\alpha_n^m = (n-m+1)(n-m)$ and $\beta_n^m = (n+m+1)(n+m)$*
**Preuve** *By (C.8) and (C.6), we have the result for $m = 0$ :*

$$\begin{aligned}(2n+1)(1-x^2)P_n'(x) &= (2n+1)\big(nP_{n-1}(x) - nxP_n(x)\big) \\ &= (2n+1)nP_{n-1}(x) - n\big((n+1)P_{n+1}(x) + nP_{n-1}(x)\big) \\ &= (n+1)nP_{n-1}(x) - n(n+1)P_{n+1}(x)\end{aligned}$$

*Then by derivating (D.8) and using (D.7)*

$$\begin{aligned}\Big((2n+1)(1-x^2)P_n^{(m)}\Big)' &= \beta_n^{m-1}P_{n-1}^{(m)} - \alpha_n^{m-1}P_{n+1}^{(m)} \\ (2n+1)(1-x^2)P_n^{(m+1)} - 2(2n+1)xP_n^{(m)} &= \beta_n^{m-1}P_{n-1}^{(m)} - \alpha_n^{m-1}P_{n+1}^{(m)} \\ (2n+1)(1-x^2)P_n^{(m+1)} &= 2(2n+1)xP_n^{(m)} + \beta_n^{m-1}P_{n-1}^{(m)} - \alpha_n^{m-1}P_{n+1}^{(m)} \\ &= (\beta_n^{m-1} + 2(n+m))P_{n-1}^{(m)} - (\alpha_n^{m-1} - 2(n-m+1))P_{n+1}^{(m)} \\ &= \beta_n^m P_{n-1}^{(m)} - \alpha_n^m P_{n+1}^{(m)}\end{aligned}$$

□

By applying the previous lemma using $x = cos(\theta)$, we obtain

$$\begin{aligned}(2l+1)\ \sin\theta\ P_l^m(\cos\theta) &= (2l+1)(1-\cos^2\theta)^{\frac{1}{2}}(-1)^m(1-\cos^2\theta)^{\frac{m}{2}}\frac{d^m P_l}{dx^m}(\cos\theta) \\ &= -(-1)^{m-1}(1-\cos^2\theta)^{\frac{m-1}{2}}\ (2l+1)(1-\cos^2\theta)\frac{d^m P_l}{dx^m}(\cos\theta) \\ &= (-1)^{m-1}(1-\cos^2\theta)^{\frac{m-1}{2}}\Big(\alpha_l^{m-1}P_{l+1}^{(m-1)} - \beta_l^{m-1}P_{l-1}^{(m-1)}\Big)(\cos\theta) \\ &= \alpha_l^{m-1}P_{l+1}^{m-1}(\cos\theta) - \beta_l^{m-1}P_{l-1}^{m-1}(\cos\theta)\end{aligned}$$

where $\alpha_n^m = (n-m+1)(n-m)$ and $\beta_n^m = (n+m+1)(n+m)$
Then we have

$$\begin{aligned}\mathcal{H}(a)_{lm}^p = \frac{1}{2(2l+1)}\Big(&e^{-\imath\phi_0}\Big(\mathcal{H}(f)_{l-1,m+1}^p - \mathcal{H}(f)_{l+1,m+1}^p\Big) \\ &+ e^{\imath\phi_0}\Big((l-m+2)(l-m+1)\mathcal{H}(f)_{l+1,m-1}^p - (l+m)(l+m-1)\mathcal{H}(f)_{l-1,m-1}^p\Big)\Big)\end{aligned} \tag{D.9}$$

## Decomposition for negative orders : $\mathcal{H}(f)^p_{l,-m}$

Let $m > 0$ and

$$\mathcal{H}(f)_{l,-m} = \int_{S^2} f(\sigma) Y^*_{l,-m}(\sigma)\ \mathrm{d}\sigma = (-1)^m \int_{S^2} f(\sigma) Y_{l,m}(\sigma)\ \mathrm{d}\sigma$$

we can write, by the same method

$$\mathcal{H}(f)^{p+1}_{l,-m} = (-1)^m \int_{S^{z+}} P_{p+1}(\cos(\sigma, X))\ P^m_l(\cos\theta)\ e^{\imath m\phi}\ \mathrm{d}\sigma$$
$$\mathcal{H}(f)^{p+1}_{l,-m} = \frac{2p+1}{p+1}\left(r_1 \mathcal{H}(a)^p_{l,-m} + r_2 \mathcal{H}(b)^p_{l,-m}\right) - \frac{p}{p+1}\mathcal{H}(f)^{p-1}_{l,-m}$$

where

$$\mathcal{H}(a)^p_{l,-m} = (-1)^m \int_{S^{z+}} \sin\theta\cos(\phi+\phi_0)\ P^m_l(\cos\theta)\ e^{\imath m\phi}\ P_p(\cos(\sigma, X))\ \mathrm{d}\sigma$$
$$\mathcal{H}(b)^p_{l,-m} = (-1)^m \int_{S^{z+}} \cos\theta\ P^m_l(\cos\theta)\ e^{\imath m\phi}\ P_p(\cos(\sigma, X))\ \mathrm{d}\sigma$$

Then

$$\mathcal{H}(b)^p_{l,-m} = \frac{l+m}{2l+1}\ \mathcal{H}(f)^p_{l-1,-m} + \frac{l-m+1}{2l+1}\ \mathcal{H}(f)^p_{l+1,-m}$$

and

$$\mathcal{H}(a)^p_{l,-m} = \frac{-1}{2(2l+1)}\Big(e^{\imath\phi_0}\left(\mathcal{H}(f)^p_{l-1,-(m+1)} - \mathcal{H}(f)^p_{l+1,-(m+1)}\right) + e^{-\imath\phi_0}\left((l-m+2)(l-m+1)\mathcal{H}(f)^p_{l+1,-(m-1)} - (l+m)(l+m-1)\mathcal{H}(f)^p_{l-1,-(m-1)}\right)\Big) \tag{D.10}$$

**Remark** *the sign* $-1$ *comes from* $(-1)^m$ *which not has the same sign as* $(-1)^{m\pm1}$ *in* $\mathcal{H}(a)^p_{l,-m}$.

## Initial cases

– $p = 0$

$$\mathcal{H}(f)^0_{lm} = \int_0^{\frac{\pi}{2}} 1 \times P^m_l(\cos\theta)\sin\theta\ \mathrm{d}\theta \underbrace{\int_0^{2\pi} e^{-\imath m\phi}\ \mathrm{d}\phi}_{=0 \text{ if } m\neq 0}$$

$$\mathcal{H}(f)^0_{l0} = \int_0^{\frac{\pi}{2}} 1 \times P^0_l(\cos\theta)\sin\theta\ \mathrm{d}\theta \times 2\pi$$
$$= 2\pi\int_0^1 P_l(t)\ \mathrm{d}t = \begin{cases} 2\pi & \text{if } l = 0 \\ 0 & \text{if } l \text{ even and } l \neq 0 \\ \frac{2\pi}{l+1}P_{l-1}(0) & \text{if } l \text{ odd} \end{cases}$$

The value is obtained by (C.7), (C.2) and (C.4) :

$$\begin{aligned} P_{n+1}(0) - P_{n-1}(0) &= (-1)^{\frac{n+1}{2}} \frac{1\cdot 3\cdot 5\cdot \ldots \cdot n}{2\cdot 4\cdot 6\cdot \ldots \cdot (n+1)} - (-1)^{\frac{n-1}{2}} \frac{1\cdot 3\cdot 5\cdot \ldots \cdot (n-2)}{2\cdot 4\cdot 6\cdot \ldots \cdot (n-1)} \qquad n : \text{odd} \\ &= -(-1)^{\frac{n-1}{2}} \frac{1\cdot 3\cdot 5\cdot \ldots \cdot n + 1\cdot 3\cdot 5\cdot \ldots \cdot (n-2)(n+1)}{2\cdot 4\cdot \ldots \cdot (n+1)} \\ &= -(-1)^{\frac{n-1}{2}} (2n+1) \frac{1\cdot 3\cdot 5\cdot \ldots \cdot (n-2)}{2\cdot 4\cdot \ldots \cdot (n+1)} \\ &= -\frac{2n+1}{n+1} P_{n-1}(0) \end{aligned}$$

– $p = 1$

$$\mathcal{H}(f)^1_{lm} = \int_{S^{z^+}} \cos(\sigma, X) P^m_l(\cos\theta) e^{-\imath m\phi} \, \mathrm{d}\sigma$$

$$\begin{aligned} \mathcal{H}(f)^1_{lm} &= \int_{S^{z^+}} \big(r_1 \sin\theta \cos(\phi+\phi_0) + r_2 \cos\theta\big) P^m_l(\cos\theta) e^{-\imath m\phi} \, \mathrm{d}\sigma \\ &= r_1 \mathcal{H}(a)^0_{lm} + r_2 \mathcal{H}(b)^0_{lm} \end{aligned}$$

which is 0 when $|m| > 1$. (because the support of $f^1_{lm}$ is in $f^0_{lm} = 0$)
We have the same result for $\mathcal{H}(f)^p_{l,-m}$.

– $m = -1$

$$\mathcal{H}(f)^1_{l,-1} = r_1 \frac{l(l+1)}{2(2l+1)} e^{-\imath\phi_0} \big(\mathcal{H}(f)^0_{l-1,0} - \mathcal{H}(f)^0_{l+1,0}\big)$$

– $m = 0$

$$\mathcal{H}(f)^1_{l0} = r_2 \Big(\frac{l}{2l+1} \, \mathcal{H}(f)^0_{l-1,0} + \frac{l+1}{2l+1} \, \mathcal{H}(f)^0_{l+1,0}\Big)$$

– $m = 1$

$$\mathcal{H}(f)^1_{l,1} = r_1 \frac{l(l+1)}{2(2l+1)} e^{\imath\phi_0} \big(\mathcal{H}(f)^0_{l+1,0} - \mathcal{H}(f)^0_{l-1,0}\big)$$

– $m \notin \{-1, 0, 1\}$

$$\mathcal{H}(f)^1_{l,m} = 0$$

**Remark** *For $p = 1$ we can optimize the computation with $\mathcal{H}(f)^0_{l,0} \neq 0$ if $l > 0$ and $l$ even number. Moreover,*

$$\begin{aligned} \forall m : |m| > p \quad \mathcal{H}(f)^p_{lm} = 0 \\ \forall m : |m| > l \quad \mathcal{H}(f)^p_{lm} = 0 \\ \forall l : l > 2p_{max} - p \quad \mathcal{H}(f)^p_{lm} = 0 \end{aligned}$$

*which define the space of parameter $(p, l, m)$*

– $m = 0$

$m = 0$ is a singular value for our formulae : a common point of the two formulations for positive and negative order of $m$. We use the same method as previously
$\forall p \geq 2, m = 0$

$$\mathcal{H}(f)^{p+1}_{l0} = \frac{2p+1}{p+1} \big(r_1 \mathcal{H}(a)^p_{l0} + r_2 \mathcal{H}(b)^p_{l0}\big) - \frac{p}{p+1} \mathcal{H}(f)^{p-1}_{l0}$$

with

$$\mathcal{H}(b)^p_{lm} = \frac{l+m}{2l+1}\,\mathcal{H}(f)^p_{l-1,m} + \frac{l-m+1}{2l+1}\,\mathcal{H}(f)^p_{l+1,m}$$

and

$$\mathcal{H}(a)^p_{l0} = \int_{S^{z+}} a_{l0}(\sigma)\; P_p(\cos(\sigma, X))\;\mathrm{d}\sigma$$

$$a_{l0}(\sigma) = \sin\theta\cos(\phi+\phi_0)\; P_l(\cos\theta) = \frac{e^{\imath\phi_0}}{2}\sin\theta\; P_l(\cos\theta)e^{\imath\phi} + \frac{e^{-\imath\phi_0}}{2}\sin\theta\; P_l(\cos\theta)e^{-\imath\phi}$$

By (C.7),

$$\begin{aligned}(2l+1)\sin\theta P_l(\cos\theta) &= \sin\theta\left(P^{(1)}_{l+1} - P^{(1)}_{l-1}\right)(\cos\theta)\\ &= (-1)^1(1-\cos^2\theta)^{\frac{1}{2}}\frac{d^1}{dx^1}\left(P_{l-1} - P_{l+1}\right)(\cos\theta)\\ &= P^1_{l-1}(\cos\theta) - P^1_{l+1}(\cos\theta)\end{aligned}$$

Then

$$\mathcal{H}(a)^p_{l0} = \frac{1}{2(2l+1)}\left(e^{\imath\phi_0}\left(\mathcal{H}(f)^p_{l+1,-1} - \mathcal{H}(f)^p_{l-1,-1}\right) + e^{-\imath\phi_0}\left(\mathcal{H}(f)^p_{l-1,1} - \mathcal{H}(f)^p_{l+1,1}\right)\right)$$

- $l = 0$

When $l = 0$, then we have $m = 0$, else $\mathcal{H}(f)^p_{0m} = 0$.

$$\mathcal{H}(f)^{p+1}_{0,0} = \int_{S^{z+}} P_{p+1}(\cos(\sigma, X))\;\mathrm{d}\sigma$$

by recurrence formula (C.6) and the decomposition (D.3) we have

$$P_{p+1}(\cos(\sigma, X)) = \frac{2p+1}{p+1}\left(r_1\sin\theta\cos(\phi+\phi_0) + r_2\cos\theta\right)P_p(\cos(\sigma, X)) - \frac{p}{p+1}P_{p-1}(\cos(\sigma, X))$$

but,

$$\begin{aligned}2\sin\theta\cos(\phi+\phi_0) &= \sin\theta\left(e^{\imath\phi_0}e^{\imath\phi} + e^{-\imath\phi_0}e^{-\imath\phi}\right)\\ &= -P^1_1(\cos\theta)\left(e^{\imath\phi_0}e^{\imath\phi} + e^{-\imath\phi_0}e^{-\imath\phi}\right)\\ &= e^{\imath\phi_0}\;(-1)^1P^1_1(\cos\theta)\;e^{\imath\phi} - e^{-\imath\phi_0}\;P^1_1(\cos\theta)\;e^{-\imath\phi}\end{aligned}$$

Then

$$\begin{aligned}\mathcal{H}(a)^p_{0,0} &= \frac{1}{2}\left(e^{\imath\phi_0}\mathcal{H}(f)^p_{1,-1} - e^{-\imath\phi_0}\mathcal{H}(f)^p_{1,1}\right)\\ \mathcal{H}(b)^p_{00} &= \mathcal{H}(f)^p_{1,0}\end{aligned}$$

which are a generalization of $\mathcal{H}(a)^p_{lm}$ and $\mathcal{H}(b)^p_{lm}$ where we consider $\mathcal{H}(f)^p_{-1,m} = 0$ : our equations are coherent.

$$\mathcal{H}(f)^{p+1}_{0,0} = \frac{2p+1}{p+1}\big(r_1\mathcal{H}(a)^p_{0,0} + r_2\mathcal{H}(b)^p_{0,0}\big) - \frac{p}{p+1}\mathcal{H}(f)^{p-1}_{0,0}$$

# Annexe E

# Fonctions de transfert propagatives par décomposition de Fourier

Nous utilisons la décomposition suivante de $\sigma$

$$\begin{cases} x(\theta,\phi) = \cos\phi\sin\theta \\ y(\theta,\phi) = \sin\phi\sin\theta \\ z(\theta,\phi) = \cos\theta \end{cases} \tag{E.1}$$

Ainsi

$$\begin{aligned} \cos(\sigma, X) &= \frac{\sin\theta\ (x\cos\phi + y\sin\phi) + z\cos\theta}{|X|} \\ &= r_1 \sin\theta\cos(\phi+\phi_0) + r_2\cos\theta \end{aligned} \tag{E.2}$$

avec

$$r_1 = \frac{\sqrt{x^2+y^2}}{|X|},\ \cos\phi_0 = \frac{x}{\sqrt{x^2+y^2}},\ \sin\phi_0 = \frac{-y}{\sqrt{x^2+y^2}},\ r_2 = \frac{z}{|X|} \tag{E.3}$$

Nous sommes donc amenés à calculer

$$\hat{f}_{pq} = \sum_{l=0}^{\infty} (2l+1)\imath^l j_l(\kappa|x|)\hat{f}_l^{pq} \tag{E.4}$$

avec

$$\hat{f}_l^{p,q} = \int_{S^+} P_l(\cos(\sigma, X)) e^{-\imath p\theta} e^{-\imath q\phi}\ \mathrm{d}\sigma$$

où $S^+$ est une demi sphère dont la direction sera précisée ultérieurement.

De plus, par le fait que

$$|j_l(v)| \leq \sqrt{\frac{e}{2}} \frac{(e|v|)^l}{(2l+1)^{l+1}} \tag{E.5}$$

et

$$\begin{aligned} |\hat{f}_l^{pq}| &= \left| \int_{S^2} P_p(\cos(\sigma, X)) \cdot \mathbf{1}_{S^+}\ e^{-\imath p\theta} e^{-\imath q\phi}\ \mathrm{d}\sigma \right| \\ &\leq \int_{S^+} 1\ \ \mathrm{d}\sigma = 2\pi \end{aligned}$$

Nous pouvons alors en déduire un majorant du nombre de terme utiles $l_{max}$ dans la série E.4 avec $l_{max} \sim \frac{e|\kappa x|-1}{2}$.

## Cas général

L' objectif est donc simple : réduire l'ordre $l$ afin d'atteindre $\hat{f}_0^{p,q}$ et $\hat{f}_1^{p,q}$, facilement calculables car ne faisant plus apparaître de polynômes. Pour cela, nous allons procéder par récurrence linéaire.

Ainsi nous avons par la formule de récurrence (C.6) et la décomposition (E.2) :

$$P_{l+1}(\cos(\sigma, X)) = \frac{2l+1}{l+1}\big(r_1 \sin\theta \cos(\phi+\phi_0) + r_2 \cos\theta\big) P_l(\cos(\sigma, X)) - \frac{l}{l+1} P_{l-1}(\cos(\sigma, X))$$

donne

$$\hat{f}_{l+1}^{pq} = \frac{2l+1}{l+1}\big(r_1 \hat{a}_l^{pq} + r_2 \hat{b}_l^{pq}\big) - \frac{l}{l+1}\hat{f}_{l-1}^{pq}$$

avec

$$\hat{a}_l^{pq} = \int_{S^+} a^{pq}(\sigma)\ P_l(\cos(\sigma, X))\ \mathrm{d}\sigma$$
$$\hat{b}_l^{pq} = \int_{S^+} b^{pq}(\sigma)\ P_l(\cos(\sigma, X))\ \mathrm{d}\sigma$$

et

$$a^{pq}(\sigma) = \sin\theta \cos(\phi+\phi_0)\ e^{-\imath p\theta}\ e^{-\imath q\phi}$$
$$b^{pq}(\sigma) = \cos\theta\ e^{-\imath p\theta}\ e^{-\imath q\phi}$$

par le fait que

$$\cos(\phi+\phi_0) = \frac{e^{\imath\phi_0}}{2}e^{\imath\phi} + \frac{e^{-\imath\phi_0}}{2}e^{-\imath\phi}$$
$$\cos\theta = \frac{e^{\imath\theta}}{2} + \frac{e^{-\imath\theta}}{2}$$
$$\sin\theta = \frac{e^{\imath\theta}}{2\imath} - \frac{e^{-\imath\theta}}{2\imath}$$

nous obtenons

$$a^{pq}(\sigma) = \frac{1}{4\imath}\Big(e^{\imath\phi_0}e^{-\imath(p-1)\theta}\,e^{-\imath(q-1)\phi} - e^{\imath\phi_0}e^{-\imath(p+1)\theta}\,e^{-\imath(q-1)\phi} + e^{-\imath\phi_0}e^{-\imath(p-1)\theta}\,e^{-\imath(q+1)\phi} - e^{-\imath\phi_0}e^{-\imath(p+1)\theta}\,e^{-\imath(q+1)\phi}\Big)$$
$$b^{pq}(\sigma) = \frac{1}{2}\Big(e^{-\imath(p-1)\theta}\,e^{-\imath q\theta} + e^{-\imath(p+1)\theta}\,e^{-\imath q\theta}\Big)$$

Ceci nous conduit à

$$\hat{a}_l^{pq} = \frac{e^{\imath\phi_0}}{4\imath}\Big(\hat{f}_l^{p-1,q-1} - \hat{f}_l^{p+1,q-1}\Big) + \frac{e^{-\imath\phi_0}}{4\imath}\Big(\hat{f}_l^{p-1,q+1} - \hat{f}_l^{p+1,q+1}\Big)$$
$$\hat{b}_l^{pq} = \frac{1}{2}\Big(\hat{f}_l^{p-1,q} + \hat{f}_l^{p+1,q}\Big)$$

Il ne reste alors plus qu'à considérer les cas initiaux $l=0$ et $l=1$.

## Cas initial $l = 0$

Cela revient à calculer

$$\hat{f}_0^{p,q} = \int_{S^+} e^{-\imath p\theta} e^{-\imath q\phi} \, \mathrm{d}\sigma$$

En considérant que $S^+$ peut être décrit comme l'union de pavés disjoints , nous pouvons appliquer une technique de tensorisation :

$$\hat{f}_0^{p,q} = \int_{\bigcup_i S_i} e^{-\imath p\theta} e^{-\imath q\phi} \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi = \sum_i \int_{S_i} e^{-\imath p\theta} e^{-\imath q\phi} \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi$$

Avec $S_i = [\theta_a, \theta_b] \times [\phi_a, \phi_b]$, on obtient

$$\int_{S_i} e^{-\imath p\theta} e^{-\imath q\phi} \sin\theta \, \mathrm{d}\theta \, \mathrm{d}\phi = \int_{\theta_a}^{\theta_b} e^{-\imath p\theta} \sin\theta \, \mathrm{d}\theta \times \int_{\phi_a}^{\phi_b} e^{-\imath q\phi} \, \mathrm{d}\phi$$

Ainsi suivant $\theta$

$$A_\theta^p = \int_{\theta_a}^{\theta_b} e^{-\imath p\theta} \sin\theta \, \mathrm{d}\theta = \frac{1}{2\imath} \int_{\theta_a}^{\theta_b} e^{-\imath(p-1)\theta} - e^{-\imath(p+1)\theta} \, \mathrm{d}\theta$$

pour $p \notin \{-1, 1\}$ on a

$$\begin{aligned}\int_{\theta_a}^{\theta_b} e^{-\imath p\theta} \sin\theta \, \mathrm{d}\theta = &\frac{\imath}{p-1} \, e^{-\imath(p-1)\frac{\theta_a+\theta_b}{2}} \sin((p-1)\tfrac{\theta_a-\theta_b}{2}) \\ &- \frac{\imath}{p+1} \, e^{-\imath(p+1)\frac{\theta_a+\theta_b}{2}} \sin((p+1)\tfrac{\theta_a-\theta_b}{2})\end{aligned}$$

pour $p = -1$ on a

$$\begin{aligned}\int_{\theta_a}^{\theta_b} e^{\imath\theta} \sin\theta \, \mathrm{d}\theta = &\frac{\imath}{2} \, e^{\imath(\theta_a+\theta_b)} \sin(\theta_a - \theta_b) \\ &- \frac{\imath}{2}(\theta_a - \theta_b)\end{aligned}$$

pour $p = 1$ on a

$$\begin{aligned}\int_{\theta_a}^{\theta_b} e^{-\imath\theta} \sin\theta \, \mathrm{d}\theta = &\frac{\imath}{2}(\theta_a - \theta_b) \\ &- \frac{\imath}{2} \, e^{-\imath(\theta_a+\theta_b)} \sin(\theta_a - \theta_b)\end{aligned}$$

De même suivant $\phi$ pour $q \neq 0$

$$B_\phi^q = \int_{\phi_a}^{\phi_b} e^{-\imath q\phi} \, \mathrm{d}\phi = -\frac{2}{q} e^{-\imath q\frac{\theta_a+\theta_b}{2}} \sin(q\tfrac{\theta_a-\theta_b}{2})$$

et pour $q = 0$

$$\int_{\phi_a}^{\phi_b} e^{-\imath 0\phi} \, \mathrm{d}\phi = \phi_b - \phi_a$$

## Cas initial $l = 1$

$$\hat{f}_1^{p,q} = \int_{S^+} \cos(\sigma, X) e^{-\imath p\theta} e^{-\imath q\phi} \, \mathrm{d}\sigma$$

Par le fait que

$$\cos(\sigma, X) = r_1 \sin\theta \cos(\phi + \phi_0) + r_2 \cos\theta$$

et en utilisant le même principe de tensorisation nous avons à calculer

$$\int_{\theta_a}^{\theta_b} e^{-\imath p\theta} \sin^2\theta \, \mathrm{d}\theta = \int_{\theta_a}^{\theta_b} \frac{1-\cos 2\theta}{2} e^{-\imath p\theta} \, \mathrm{d}\theta = \tfrac{1}{2}\int_{\theta_a}^{\theta_b} e^{-\imath p\theta} \, \mathrm{d}\theta - \tfrac{1}{2}\int_{\theta_a}^{\theta_b} \cos 2\theta \; e^{-\imath p\theta} \, \mathrm{d}\theta = \tfrac{1}{2}\big(B_\theta^p - C_\theta^p\big)$$

$$\int_{\theta_a}^{\theta_b} e^{-\imath p\theta} \cos\theta \sin\theta \, \mathrm{d}\theta = \frac{1}{2}\int_{\theta_a}^{\theta_b} \sin 2\theta \; e^{-\imath p\theta} \, \mathrm{d}\theta = \tfrac{1}{2} D_\theta^p$$

$$E_\phi^q = \int_{\phi_a}^{\phi_b} e^{-\imath q\phi} \cos(\phi + \phi_0) \, \mathrm{d}\phi$$

Alors

$$\hat{f}_1^{p,q} = \int_{[\theta_a,\theta_b]\times[\phi_a,\phi_b]} \cos(\sigma, X) e^{-\imath p\theta} e^{-\imath q\phi} \, \mathrm{d}\sigma = \frac{r_1}{2}(B_\theta^p - C_\theta^p)E_\phi^q + \frac{r_2}{2} D_\theta^p B_\phi^q$$

En effet pour $|p| \neq 2$, on a

$$\begin{aligned} C_\theta^p = {} & \tfrac{1}{2-p} e^{-\imath(p-2)\frac{\theta_a+\theta_b}{2}} \sin((p-2)\tfrac{\theta_a-\theta_b}{2}) \\ & - \tfrac{1}{2+p} e^{-\imath(p+2)\frac{\theta_a+\theta_b}{2}} \sin((p+2)\tfrac{\theta_a-\theta_b}{2}) \end{aligned}$$

si $p = -2$ on a

$$C_\theta^{-2} = \tfrac{1}{4} e^{2\imath(\theta_a+\theta_b)} \sin(2(\theta_a - \theta_b)) + \tfrac{1}{2}(\theta_b - \theta_a)$$

et si $p = 2$ on a

$$C_\theta^{2} = \tfrac{1}{2}(\theta_b - \theta_a) + \tfrac{1}{4} e^{-2\imath(\theta_a+\theta_b)} \sin(2(\theta_a - \theta_b))$$

On remarque que au signe près et à un facteur $-\imath$ près on obtient

$$\begin{aligned} D_\theta^p = {} & \tfrac{-\imath}{2-p} e^{-\imath(p-2)\frac{\theta_a+\theta_b}{2}} \sin((p-2)\tfrac{\theta_a-\theta_b}{2}) \\ & - \tfrac{\imath}{2+p} e^{-\imath(p+2)\frac{\theta_a+\theta_b}{2}} \sin((p+2)\tfrac{\theta_a-\theta_b}{2}) \end{aligned}$$

si $p = -2$ on a

$$D_\theta^{-2} = \tfrac{-\imath}{4} e^{2\imath(\theta_a+\theta_b)} \sin(2(\theta_a - \theta_b)) + \tfrac{\imath}{2}(\theta_b - \theta_a)$$

et si $p = 2$ on a

$$D_\theta^{2} = \tfrac{\imath}{2}(\theta_a - \theta_b) + \tfrac{\imath}{4} e^{-2\imath(\theta_a+\theta_b)} \sin(2(\theta_a - \theta_b))$$

En usant du même argument que pour $A_\theta^p$, au signe près, au facteur $\imath$ près, on obtient pour $q \notin \{-1, 1\}$,

$$\begin{aligned} E_\phi^q = {} & \frac{-e^{\imath\phi_0}}{q-1} \, e^{-\imath(q-1)\frac{\phi_a+\phi_b}{2}} \sin((q-1)\tfrac{\phi_a-\phi_b}{2}) \\ & - \frac{e^{-\imath\phi_0}}{q+1} \, e^{-\imath(q+1)\frac{\phi_a+\phi_b}{2}} \sin((q+1)\tfrac{\phi_a-\phi_b}{2}) \end{aligned}$$

pour $q = -1$ on a

$$E_\phi^{-1} = \frac{-e^{\imath\phi_0}}{2}\, e^{\imath(\phi_a+\phi_b)} \sin(\phi_a - \phi_b) - \frac{e^{-\imath\phi_0}}{2}(\phi_a - \phi_b)$$

pour $q = 1$ on a

$$E_\phi^{1} = \frac{-e^{\imath\phi_0}}{2}(\phi_a - \phi_b) - \frac{e^{-\imath\phi_0}}{2}\, e^{-\imath(\phi_a+\phi_b)} \sin(\phi_a - \phi_b)$$

**Remarque** *L'ensemble des coefficients utiles afin de calculer les $\hat{f}_l^{pq}$ pour $l$ maximal est en forme de pyramide tronquée.*

# Annexe F

# Top 10 des algorithmes du siècle

Top 10 des algorithmes du siècle issu de "Computing in Science & Engineering" [6]

**1946** : The Metropolis Algorithm for Monte-Carlo. Through the use of random processes, this algorithm offers an efficient way to stumble toward answers to problems that are too complicated to solve exactly.
**1947** : Simplex Method for Linear Programming. An elegant solution to a common problem in planning and decision-making.
**1950** : Krylov Subspace Iteration Method. A technique for rapidly solving the linear equations that abound in scientific computation.
**1951** : The Decompositional Approach to Matrix Computations. A suite of techniques for numerical linear algebra.
**1957** : The Fortran Optimizing Compiler. Turns high-level code into efficient computer-readable code.
**1959** : QR Algorithm for Computing Eigenvalues. Another crucial matrix operation made swift and practical.
**1962** : Quicksort Algorithms for Sorting. For the efficient handling of large databases.
**1965** : Fast Fourier Transform. Perhaps the most ubiquitous algorithm in use today, it breaks down waveforms (like sound) into periodic components.
**1977** : Integer Relation Detection. A fast method for spotting simple equations satisfied by collections of seemingly unrelated numbers.
**1987** : Fast Multipole Method. A breakthrough in dealing with the complexity of n-body calculations, applied in problems ranging from celestial mechanics to protein folding.

# Bibliographie


[1] M. Abramovitz and I. Stegun, *Handbook of Mathematical Functions*, Applied Mathematical Series, National Bureau of Standards, Washington DC, 1964.

[2] B. Alpert and R. Jakob-Chien, *A fast spherical filter with uniform resolution*, J. Comput. Phys., 136 (1997), p. 580.

[3] S. Balay, W. Gropp, L. McInnes, and B. Smith., *Petsc 2.0 users manual. technical report*, 1996. Argonne National Laboratory.

[4] S. D. Baude F., Caromel D. Furmento N., *Overlapping communication with computation in distributed object systems*, High-Performance Computing and Networking, (1999), pp. 744–753.

[5] J.-P. Berenger, *A perfectly matched layer for the absorption of electromagnetic waves*, J. Comput. Phys., 114 (1994), pp. 185–200.

[6] J. Board and K. Schulten, *The fast multipole algorithm*, Computing in Science & Engineering, 2 (2000), pp. 76–79. `http://www.computer.org/cise/articles/Top_Algorithms.htm`.

[7] T. Boulmezaout, *Inverted elements method for solving elliptic equations in unbounded domains*, (2003). Publications du Laboratoire Jacques-Louis Lions.

[8] O. Bucci, C. Gennarelli, R. Riccio, and C. Savarese, *Electromagnetic fields interpolation from nonuniform samples over spherical and cylindrical surfaces*, IEEE Proc. Microw. Antennas Propag., 141 (1994), pp. 77–84.

[9] O. Bucci, C. Gennarelli, and C. Savarese, *Optimal interpolation of radiated fields over a sphere*, IEEE Trans. Antennas Propagat., AP-39 (1991), pp. 1633–1643.

[10] D. S. Burnett and R. L. Holford, *Multipole-based 3-D infinite elements : an ellipsoidal acoustic element and a spherical electromagnetic element*, in IUTAM Symposium on Computational Methods for Unbounded Domains (Boulder, CO, 1997), vol. 49 of Fluid Mech. Appl., Kluwer Acad. Publ., Dordrecht, 1998, pp. 53–62.

[11] Q. Carayol, *Développement et Analyse d'une méthode multipôle multiniveau pour l'électromagnétisme*, PhD thesis, Université Pierre et Marie Curie, 2002.

[12] J. Carrier, L. Greengard, and V. Rokhlin, *A fast adaptive multipole algorithm for particle simulations*, SIAM J. Sci. Statisc. Comput., 9 (1988).

[13] W. Cecot, W. Rachowicz, and L. Demkowicz, *An hp-adaptive finite element method for electromagnetics. III. A three-dimensional infinite element for Maxwell's equations*, Internat. J. Numer. Methods Engrg., 57 (2003), pp. 899–921.

[14] W. Chew, J.-M. Jin, C.-C. Lu, E. Michielssen, and J. Song, *Fast solution methods in electromagnetics*, IEEE trans. on antennas and propa., 45 (1997), pp. 533–543.

[15] W. C. Chew, *Fast algorithms for wave scattering developed at the university of illinois's electromagnetics laboratory*, IEEE antennas propag. mag., 35 (1993), pp. 22–32.

[16] W. C. Chew, C.-C. Lu, and Y. M. Wang, *Review of efficient computation of three-dimensional scattering of vector electromagnetic waves*, J. Opt. Soc. Amer. A, 11 (1994), pp. 1528–1537.

[17] R. Coifman, V. Rokhlin, and S. Wandzura, *The fast multipole method for the wave equation : A pedestrian prescription*, IEEE Antennas and Propagation Magazine, 35 (1993), pp. 7–12.

[18] D. Colton and R. Kress, *Inverse Acoustic and Electromagnetic Scattering Theory*, vol. 2, Springer-Verlag, 1992.

[19] J. V. D. Caromel, W. Klauser, *Towards seamless computing and metacomputing in java*, in Concurrency Practice and Experience, G. C. Fox, ed., Wiley & Sons, Ltd, September-November 1998, pp. 1043–1061.

[20] Y. R. D. Caromel, F. Belloncle, *The C++// System*, in Parallel Programming Using C++, G. Wilson and P. Lu, eds., MIT Press, 1996, pp. 257–296.

[21] E. Darve, *Méthodes multipôles rapides : Résolution des équations de Maxwell par formulations intégrales*, PhD thesis, Université Pierre et Marie Curie, 1999.

[22] E. Darve, *The fast multipole method (i) : Error analysis and asymptotic complexity*, SIAM Num. Anal., 38 (2000), pp. pp. 98–128.

[23] ——, *The fast multipole method : Numerical implementation*, Journal of Computational Physics, 160 (2000), pp. pp. 195–240.

[24] B. Dembart and G. Shubin, *A 3d fast multipole method for electromagnetics with multiple levels*, Technical document ISSTECH-97-004, Boeing, December 1994.

[25] B. Dembart and E. Yip, *A 3-d moment method code based on fast multipole*, in URSI Radio Sci. Meet. Dig., Seattle, WA, June 1994, p. 23.

[26] ——, *A 3d fast multipole method for electromagnetics with multiple levels*, in Proceedings of the 11th annual review of progress in applied computational electromagnetics, vol. 1, Monterey, California, March 1995, pp. 621–628.

[27] A. Dutt, M. Gu, and V. Rokhlin, *Fast algorithms for polynomial interpolation, integration and differentiation*, SIAM J. Num. Anal., 33 (1996), pp. 1689–1711.

[28] N. Engheta, W. Murphy, V. Rokhlin, and M. S. Vassiliou, *The fast multipole method (fmm) for electromagnetic scattering problems*, IEEE trans. on antennas and propa., 40 (1992), pp. 634–641.

[29] M. A. Epton and B. Dembart, *Multipole translation theory for three-dimensional laplace and helmholtz equations*, SIAM J. Sci. Comput., 16 (1995), pp. 865–897.

[30] F. M. F. Collino, *La méthode multipôle à deux composantes pour l'électromagnétisme*, Tech. Rep. TR/EMC/01/22, CERFACS, Toulouse, France, 2001.

[31] P.-L. George, *Premiéres expériences de maillage automatique par une méthode de delaunay anisotrope en trois dimensions*, Tech. Rep. RT-0272, INRIA Rocquencourt, 2002.

[32] G. H. Golub and C. F. Van Loan, *Matrix computations*, vol. 3 of Johns Hopkins Series in the Mathematical Sciences, Johns Hopkins University Press, Baltimore, MD, 1983.

[33] L. Greengard, J. Huang, V. Rokhlin, and S. Wandzura, *Accelerating fast multipole methods for the helmholtz equation at low frequencies*, IEEE Computational Science & Engineering, (1998), pp. 32–38.

[34] L. Greengard and V. Rokhlin, *A fast algorithm for particle simulations*, J. Comp. Physics, 73 (1987), pp. 325–348.

[35] ——, *Rapid evaluation of potential fields in three dimensions*, in Vortex Methods, vol. 1360, Springer-Verlag, 121, 1988.

[36] ——, *A new version of the fast multipole method for the laplace equation in three dimensions*, Acta numerica, 6 (1997), pp. 226–269.

[37] W. Gropp, E. Lusk, and A. Skjellum, *Using MPI : portable parallel programming with the message passing interface*, no. ISBN 0-262-57104-8, The MIT Press, 1994.

[38] P. Havé, *A parallel implementation of the Fast Multipole Method for Maxwell equations*, no. Eccomas2001-7, Laboratoire d'Analyse Numérique de l'université Pierre et Marie Curie, John Wiley & Sons, 2001.

[39] ——, *EASYMSG : Tools and techniques for an adaptive overlapping in SPMD programming*, Mathematical Modelling and Numerical Analysis, 36 (2002), pp. 863–882.

[40] B. Hu, C. Chew, W., E. Michielssen, and J. Zhao, *Fast inhomogeneous plane wave algorithm for the fast analysis of two-dimensional scattering problems*, Radio Science, 34 (1999), pp. 759–72.

[41] B. Hu, C. Chew, W., and S. Velamparambil, *Fast inhomogeneous plane analysis of electromagnetic wave algorithm for the scattering*, Radio Science, 36 (2001), pp. 1327–1340.

[42] B. Hu and W. Chew, *Fast inhomogeneous plane wave algorithm for electromagnetic solutions in layered medium structures*, in 36th Annual Technical Meeting Society of Engineering Science, Austin, Texas, Oct. 25-27 1999.

[43] B. Hu and W. C. Chew, *Fast inhomogeneous plane wave algorithm for electromagnetic solutions in layered medium structures : two-dimensional case*, Radio Science, 35 (2000), pp. 31–43.

[44] ——, *Fast inhomogeneous plane wave algorithm for multi-layered medium problems*, in IEEE Antennas and Propagation Society International Symposium. Transmitting Waves of Progress to the Next Millennium, Salt Lake City, UT, USA, 16-21 July 2000, pp. 606–609.

[45] ——, *Fast inhomogeneous plane wave algorithm for scattering from objects above the multilayered medium*, IEEE Transactions on Geoscience and Remote Sensing, 39 (2001), pp. 1028–1038.

[46] B. Hu, W. C. Chew, E. Michielssen, and J. Zhao, *Fast inhomogeneous plane wave algorithm (FIPWA) for the fast analysis of two-dimensional scattering problems.* University of Illinois, Urbana.

[47] G. Karypis and V. Kumar, *A fast and high quality multilevel scheme for partitioning irregular graphs*, SIAM J. Sci. Comput., 20 (1998), pp. 359–392 (electronic).

[48] ——, *Parallel multilevel k-way partitioning scheme for irregular graphs*, SIAM Rev., 41 (1999), pp. 278–300 (electronic).

[49] J. J. Knab, *Interpolation of band-limited functions using the approximate prolate series*, IEEE Trans. Information Theory, 6 (1979), pp. 717–720.

[50] ——, *The sampling window*, IEEE Trans. Information Theory, 1 (1983), pp. 157–159.

[51] S. Koc, J. Song, and W. Chew, *Error analysis for the numerical evaluation of the diagonal forms of the scalar spherical addition theorem*, Research report CCEM-32-97, Univeristy of Illinois, Urbana, IL 61801-2991, October 8 1997.

[52] C.-C. Lu and W. C. Chew, *Fast far field approximation for calculating the rcs of large objects*, Micro. Opt. Tech. Lett., 8 (1995), pp. 238–241.

[53] J. Ma, V. Rokhlin, and S. Wandzura, *Generalized Gaussian quadrature rules for systems of arbitrary functions*, SIAM J. Numer. Anal., 33 (1996), pp. 971–996.

[54] W. Magnus and F. Oberhettinger, *Special functions of Mathematical Physics*, New-York, Chelsea, 1949.

[55] J. Mautz and R. Harrington, *H-field, e-field, and combined-field solutions for conducting bodies of revolution*, Archiv für Elektronik und Übertragungstechnik (A.E.U), 32 (1978), pp. 157–163.

[56] ——, *A combined-source formulation for radiation and scaterring from a perfectly conducting body*, IEEE Trans. Antennas Propagat., AP-27 (1979), pp. 445–454.

[57] B. Moon, H. V. Jagadish, C. Faloutsos, and J. H. Saltz, *Analysis of the clustering properties of the hilbert space-filling curve*, Knowledge and Data Engineering, 13 (2001), pp. 124–141.

[58] J. C. Nédélec, *Cours de dea de l'école polytechnique et de l'université paris 6*, 1999.

[59] S. M. Rao, D. R. Wilton, and A. W. Glisson, *Electromagnetic scattering by surfaces of arbitrary shape*, IEEE Transactions on Antennas and Propagations, AP-30 (1982), pp. 409–418.

[60] J. V. W. Reynders, *The POOMA FrameWork—a templated class library for parallel scientific computing*, in Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing (Minneapolis, MN, 1997), Philadelphia, PA, 1997, SIAM, p. 6 pp. (electronic).

[61] V. Rokhlin, *Rapid solution of integral equations of classical potential theory*, J. Comput. Phys., 60 (1985), p. 187.

[62] ——, *Rapid solution of integral equations of scattering theory in two dimensions*, Journal of Computational Physics, 86 (1990), pp. 414–439.

[63] ——, *Diagonal forms of translation operators for the helmholtz equation in three dimensions*, Research Report YALEU/DCS/RR-894, Yale University Department of Computer Science, March 1992.

[64] Y. Saad and M. H. Schultz, *GMRES : a generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Sci. Statist. Comput., 7 (1986), pp. 856–869.

[65] J. Sarvas, *Performing interpolation and anterpolation entirely by fast fourier transform in the 3-d multilevel fast multipole algorithm*, Research Report No. CCEM-13-00, Laboratory, University of Illinois at Urbana-Champaign, Center for Computational Electromagnetics and Electromagnetics.

[66] D. C. Schmidt and T. Suda, *Measuring the performance of parallel message-based process architectures*, in INFOCOM (2), 1995, pp. 624–633.

[67] X. Sheng, J. Jin, J. Song, and W. Chew, *On the formulation of hibrid finite-element and boudary-integral methods for 3-d scattering*, IEEE Antennas and Propagation, (1998).

[68] ——, *Solution of combined-field integral equation using multilevel fast multipole algorithm for scattering by homogeneous bodies*, IEEE Antennas Propagation, 46(11) (1998), pp. 1718–1726.

[69] X.-Q. Sheng, J.-M. Jin, J. Song, C.-C. Lu, and W. Chew, *Multilevel fast mulitpole algorithm for electromagnetic scattering from large complex objects*, IEEE trans. on antennas and propag., 45(10) (1997), pp. 1488–1493.

[70] J. Song and W. Chew, *Multilevel fast-multipole algorithm for solving combined field integral equations of electromagnetic scattering*, Microwave and Optical Technology Letters, 10 (1995), pp. 14–19.

[71] J. Song and W. C. Chew, *Error analysis for the truncation of multipole expansion of vector green's functions*, IEEE microwave and wireless components letters, 11 (2001), pp. 311–313.

[72] J. Song, C.-C. Lu, and W. Chew, *Multilevel fast multipole algorithm for electromagnetic scattering by large complex objects*, IEEE trans. on antennas and propag., 45 (1997), pp. 1488–1493.

[73] B. Stroustrup, *The C++ Programming Language*, Addison-Wesley, 1998. third edition.

[74] G. Sylvand, *La méthode multipôle rapide en électromagnétisme : Performances, Parallèlisation, Applications*, PhD thesis, Ecole Nationale des Ponts et Chaussées, 2002.

[75] S. Velamparambil, W. C. Chew, and J. Song, *10 million unknowns : Is it* that *big ?*, Tech. Rep. 15-01, Center for Computational Electromagnetics, 2001. Submitted to IEEE Antennas and Propagation Magazine.

[76] T. Veldhuizen, *Techniques for scientific c++*, Tech. Rep. 542, Indiana University Computer Science, August 2000.

[77] N. Yarvin and V. Rokhlin, *Generalized gaussian quadratures and singular value decompositions of integral operators*, Tech. Rep. 1109, Department of computer science research report, Yale University, 1996.

[78] ——, *An improved fast multipole algorithm for potential fields on the line*, SIAM Journal on Numerical analysis, 36 (1999), pp. 629–666.

[79] N. Yarvin and V.Rokhlin, *A generalized one-dimensional fast multipole method with application to filtering of spherical harmonics*, Jour. Comp. Phys., 147 (1998), pp. 594–609.

[80] S. Zhang and J. Jin, *Computation of Special Functions*, Wiley Interscience, 605 Third Avenue, New York, NY 10158-0012, 1996.

[81] J. Zhao and W. Chew, *Applying lf-mlfma to solve complex pec structures*, Microwave and Optical Technology Letters, 28 (2000), pp. 155–160.