Titre: Title:	Accélération du calcul de la dose en radiothérapie sur processeurs graphiques
Auteur: Author:	Sami Hissoiny
Date:	2009
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Hissoiny, S. (2009). Accélération du calcul de la dose en radiothérapie sur processeurs graphiques [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/134/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/134/
Directeurs de recherche: Advisors:	Philippe Després, & Benoît Ozell
Programme: Program:	Génie informatique

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

ACCÉLÉRATION DU CALCUL DE LA DOSE EN RADIOTHÉRAPIE SUR PROCESSEURS GRAPHIQUES

SAMI HISSOINY

DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION

DU DIPLÔME DE MAÎTRISE EN SCIENCES APPLIQUÉES

(GÉNIE INFORMATIQUE)

JUILLET 2009

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé:

ACCÉLÉRATION DU CALCUL DE LA DOSE EN RADIOTHÉRAPIE SUR PROCESSEURS GRAPHIQUES

présenté par: HISSOINY Sami

en vue de l'obtention du diplôme de: Maîtrise ès sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. ROY Robert, Ph.D., président

M. OZELL Benoît, Ph.D., membre et directeur de recherche

M. DESPRÉS Philippe, Ph.D., membre et codirecteur de recherche

M. LACROIX Frédéric, Ph.D., membre

REMERCIEMENTS

J’aimerais présenter ici mes remerciements aux gens qui ont rendu mon travail possible.

Tout d’abord, M. Benoît Ozell qui, ayant remarqué mon intérêt pour la recherche au stade du projet de fin d’études au baccalauréat, a su me guider vers les études supérieures grâce à quelques rencontres. Il a par la suite trouvé un sujet de recherche qui s’insérait parfaitement dans mon champ d’intérêt, sujet qui a su m’intéresser par un seul courriel d’une quinzaine de lignes que j’ai lu à 2000 kilomètres de chez moi, en voyage.

M. Philippe Després, qui m’a accueilli dans son bureau et qui a toléré mes questions de novice en physique avec patience et qui a toujours su y répondre et m’aiguiller dans la bonne direction. Il m’a aussi poussé et guidé à l’écriture de mon premier article et vers ma première conférence. J’ai toujours senti qu’il portait un intérêt réel et qu’il voyait un potentiel concret dans les travaux que j’effectuais.

RÉSUMÉ

Le projet présenté a trait à la planification de traitement en radiothérapie. Plus précisément, il a comme objectif de calculer la dose étant donné une configuration d'irradiation pour vérifier que celle-ci est conforme à la dose prescrite par le médecin traitant, et ce, de façon plus rapide que les techniques informatiques utilisées présentement. Le temps de calcul pour une dose complète est présentement de quelques minutes pour un calcul précis et on veut ramener ce délai à quelques secondes.

Un module de calcul a été développé et inséré dans un logiciel de planification de traitement complet, PlanUNC, développé à l'Université de Caroline du Nord. Le module développé utilise la puissance de calcul d'une carte graphique pour réaliser la majeure partie du travail numérique. La plateforme CUDA de nVidia a été utilisée pour programmer la carte graphique.

Le calcul de la dose consiste à déterminer la quantité de radiation primaire et secondaire à chaque unité de volume (voxel) représentant le modèle du patient tel qu'obtenu par une modalité d'imagerie médicale. Les données nécessaires pour réaliser ce calcul sont la densité de chaque voxel représentant le patient et les caractéristiques de l'accélérateur linéaire utilisé pour l'irradiation (énergie, spectre, collimation, etc.).

Il existe plusieurs méthodes de calcul de dose en radiothérapie. Les plus exactes sont celles faisant intervenir les techniques dites de Monte Carlo, qui s'appuient sur les principes physiques de base pour le calcul du dépôt d'énergie par la radiation. Ces

techniques sont cependant relativement lentes et typiquement incompatibles avec les échelles de temps ciblées plus tôt. Nous avons plutôt utilisé une méthode de convolution/superposition, qui est relativement rapide et qui produit des résultats quasi équivalents à ceux obtenus par la méthode Monte Carlo.

Pour une planification complète, la dose doit être calculée de façon indépendante pour chaque unité de volume. Cette particularité mène naturellement vers une implémentation de l'algorithme sur une machine à traitement parallèle, par exemple, une carte graphique. C'est ce qui a été fait au cours de ce travail.

Nous trouvons des gains en vitesse de 66x pour le calcul de la TERMA et de 550x pour le calcul de la dose, dans nos implémentations les plus rapides et donc les moins précises utilisant une carte GeForce GTX280 et se comparant à une implémentation non parallélisée s'exécutant sur une processeur Intel Q6600.

ABSTRACT

The project presented here is related to treatment planning in radiotherapy. More precisely, it is a mean of finding the configuration of irradiation, in terms of dosage, which will be able to answer the amount prescribed by the radio oncologist and this, in a way faster than the data-processing techniques used at present at the department of radio-oncology of the Centre hospitalier de l'Université de Montréal (CHUM). The computing time is at present of in the order of one minute for a precise calculation and we want to bring this time to the order of a second.

The dose calculation module is part of a complete treatment planning software, PLUNC, developed at the University of North Carolina. Our application uses the computing power which is available with the use of graphics hardware to complete most of the work. The CUDA platform,by nVidia, is used to program the graphics hardware. The calculation of the dose consists in determining the quantity of primary and secondary radiation to each unit of volume (voxel) forming the model of the patient. The data necessary to carry out this calculation are the density in all voxels forming the patient and a precise and detailed characterization of the irradiation beam as well as the ray which is produced.

The methods for calculating this are multiple. The most precise, a Monte Carlo simulation, is not realizable inside the computing time targeted in this work. We have

instead used the convolution/superposition method which aims to be a faster method to arrive at results comparable with those obtained by the Monte Carlo method.

As presented above, the calculation of the dose must be made for each voxel. This characteristic can be naturally translated in the use of parallel hardware to solve the problem, for example, graphics hardware.

We find acceleration factors of 66x for calculation of the TERMA and of 550x for calculation of the dosage, in our fastest implementations using a GeForce GTX280 graphics card and compared with a single threaded implementation being carried out on a Intel Q6600 processor.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGNES ET DES ABRÉVIATIONS

GPU : *Graphics Processing Unit* (processeur graphique)

μ : coefficient d'atténuation [cm^{-1}]

Ψ ou Φ : fluence [MeV cm^{-2} ou photon cm^{-2}, respectivement]

ρ : densité du tissu [g/cm^3]

E : énergie [MeV]

LISTE DES ANNEXES

GLOSSAIRE

FDD : fonction de dispersion, kernel de diffusion (*kernel*)

Voxel : élément de volume, analogue à la notion de pixel mais en trois dimensions

INTRODUCTION

La radiothérapie a pour but d'éliminer les cellules cancéreuses dans le corps à l'aide de rayonnement ionisant. Pour atteindre un objectif thérapeutique précis, il est impératif de connaître le plus exactement possible la dose (énergie impartie moyenne par unité de masse) produite par ce rayonnement. Le projet a donc pour but de faire un calcul le plus exact possible de la dose qui sera déposée dans un patient pour une configuration d'irradiation donnée et ce, dans un temps compatible avec la charge de travail clinique. Une planification avec un algorithme de type convolution/superposition requiert quelques minutes de calcul dans un logiciel de planification de traitement pour une configuration d'irradiation typique. Comme la configuration optimale s'obtient souvent au bout de quelques itérations, ce temps de calcul devient alors un obstacle à l'obtention d'une solution de traitement rapide. Le but de ce projet est donc de ramener ce temps de calcul à l'ordre de la seconde, tout en maintenant l'exactitude dans le calcul de la dose.

L'algorithme retenu pour effectuer le calcul de dose est de type convolution/superposition. Il conserve un haut degré de correspondance avec la méthode la plus exacte de calcul de dose, une simulation Monte-Carlo, tout en réduisant de beaucoup le temps de calcul nécessaire pour l'obtention du résultat. Ces méthodes seront décrites dans le prochain chapitre de ce document.

Le calcul de la dose est une tâche hautement parallèle : la dose doit être calculée pour chaque voxel représentant le volume complet du patient. Ce calcul s'insère donc dans

une classe de problèmes qui se résolvent par une approche « mêmes instructions données multiples » (same instructions multiple data, SIMD). Cette classe de problèmes est particulièrement propice à l'utilisation d'une carte graphique pour la grande majorité des calculs puisque cette dernière comporte plusieurs unités de calcul parallèles qui sont spécifiquement conçues pour exécuter un même ensemble d'opérations sur plusieurs données en parallèle. Une carte graphique de la compagnie nVidia couplée avec l'utilisation de l'environnement de développement CUDA ont été utilisés pour réaliser le calcul de la dose.

Le développement logiciel requis pour ce projet s'appuie sur une plateforme de planification de traitement existante : PlanUNC (*Plan, University of North Carolina*). La majeure partie du logiciel demeure inchangée, hormis l'ajout d'une option pour réaliser le calcul de la dose sur une carte graphique. Le logiciel PlanUNC, dans sa dernière itération, comporte un module de convolution/superposition s'exécutant sur un processeur à usage général (CPU). Ceci servira de base de comparaison, en se rappelant que l'algorithme de PlanUNC fait appel à plus d'approximations de calculs que celui développé dans le cadre de ce travail (plus de détails au chapitre 2).

Le premier chapitre fera état des connaissances actuelles sur les principes physiques ayant trait aux calculs en radiothérapie ainsi que les algorithmes de calcul de dose. Le deuxième chapitre présentera l'architecture de la famille de carte graphique utilisée ainsi que l'environnement de développement CUDA. Ces informations sont nécessaires pour comprendre les choix faits dans les sections suivantes. Nous n'avons pas trouvé d'autres

publications au sujet des application des cartes graphiques dans le calcul en radiothérapie faisant intervenir des éléments de génie informatique, ce qui limitera la section de revue de littérature aux principes physiques et algorithmes à implémenter sans passer par une revue en génie informatique. Les cartes graphiques ont par contre été utilisées dans beaucoup d'autres applications dans le domaine médical, notamment pour la détection de cancer du sein et en reconstruction tomographique. Le développement de l'algorithme qui est au cœur de ce travail sera décrit au troisième chapitre. Le quatrième chapitre présentera les résultats obtenus ainsi que leur analyse. Finalement, le cinquième chapitre s'attardera à la critique et à la discussion des résultats.

CHAPITRE 1 : LE CALCUL DE LA DOSE EN RADIOTHÉRAPIE

Ce chapitre est divisé en cinq sections. Premièrement, une revue des principes physiques utiles au calcul de la dose sera présentée. Deuxièmement, les divers algorithmes existants seront énumérés. Troisièmement, un survol de la plateforme matérielle et logicielle de développement sera effectué. Quatrièmement, la plateforme de planification de traitement PlanUNC sera brièvement détaillée. Finalement, nos objectifs et hypothèses seront énoncés.

1.1 Processus physique de dépôt de dose

La quantité recherchée lors du calcul de la dose déposée dans un patient s'exprime en énergie moyenne impartie par unité de masse et se nomme, dans le système international, la dose absorbée (ICRU, 1980). Cette dose est utilisée pour déterminer les dommages probables aux tissus biologiques.

Les phénomènes physiques menant aux dommages biologiques suite à une irradiation par un champ de photons se divisent en trois phases (Johns & Cunningham, 1983):

- une interaction entre un photon du rayonnement primaire et un électron du corps créant un rayonnement de diffusion secondaire et la mise en marche d'électrons à grande vitesse et la diffusion/absorption du photon;
- le parcours des électrons à grande vitesse dans le corps et l'ionisation d'atomes sur leur passage;

- le bris direct et indirect de liens moléculaires résultant en dommages biologiques.

Nous nous intéresserons ici aux étapes de déposition d'énergie dans le système, représentées par les deux premiers points ci-haut. Dans la littérature ayant trait au calcul de la dose, nous retrouvons ces étapes sous les appellations (Sharpe & Battista, 1993), (Liu, Mackie, & McCullough, 1997), (Papanikolaou, 2004):

- TERMA (*total energy release per mass unit,* énergie totale émise par unité de masse), où l'énergie du rayonnement primaire est transformée en énergie cinétique d'électrons dans le milieu d'interaction;
- DOSE, où ces électrons déposent leur énergie par ionisation le long de leur trajet dans le corps.

1.1.1 TERMA

La première étape d'un calcul de dose effectué selon le formalisme exprimé plus haut, le calcul de la TERMA, est la plus simple et la plus directe. Elle exprime l'interaction du rayonnement primaire avec le milieu et permet d'obtenir l'énergie cinétique en tout point (Papanikolaou, 2004). Le calcul de la TERMA fait intervenir le coefficient d'atténuation linéaire (μ) qui reflète la probabilité d'un photon d'interagir avec le milieu par unité de distance. Ce coefficient dépend de l'énergie E du rayonnement incident, de la densité ρ et du numéro atomique Z des éléments composant le milieu irradié.

Le coefficient d'atténuation est très bien défini en physique. Pour ce mémoire, retenons simplement que l'interaction entre un photon et la matière peut prendre quatre formes

possibles : l'effet photoélectrique, la diffusion cohérente ou de Rayleigh, la diffusion incohérente ou de Compton et la production de paires. Lorsque les photons sont pris individuellement, une seule de ces interactions peut survenir; lorsqu'on étudie un grand nombre d'interactions, comme dans le cas où un faisceau de rayonnement touche un patient, toutes ces interactions sont possibles. La probabilité d'une interaction, quelle qu'elle soit, est la somme de tous les facteurs d'atténuation propres à chaque type d'interactions (Johns & Cunningham, 1983, p. 159).

Le facteur d'atténuation massique total (μ/ρ) comprend donc la participation de chacun des facteurs d'atténuation spécifiques ayant trait aux types d'interactions énumérés précédemment. Il est divisé par la densité de milieu en vue de tabuler des valeurs indépendantes de celle-ci.

Une autre quantité est nécessaire au calcul de la dose: la fluence en énergie Ψ ou la fluence en photons Φ. Elles correspondent, respectivement, à l'énergie ou nombre de particules de rayonnement traversant une sphère en un point arbitraire. Son unité est donc le « MeV cm^{-2} » ou « photon cm^{-2} », respectivement.

Il est finalement possible de définir la TERMA. Elle se calcule comme le produit entre la fluence et le facteur d'atténuation massique total :

$$TERMA = \frac{\mu}{\rho} * \Psi = \frac{\mu}{\rho} * \Phi * E \ [MeV\ g^{-1}] \qquad (1)$$

$\frac{\mu}{\rho}$: *le facteur d'atténuationde masse total*
Ψ: *la fluence d'énergie*
E: *l'énergie*, Φ: la flu□nc□□n photon

Puisque le rayonnement émanant d'un accélérateur linéaire n'est pas constitué d'une seule énergie et que la fluence varie avec la position du point de calcul, nous trouvons un calcul un peu plus complexe en pratique (Jelen, 2007) :

$$TERMA = \int_{Emin}^{Emax} \frac{\mu(E)}{\rho} * \Phi(\mathrm{E},\mathrm{r}) * E \; dE[MeV\; g^{-1}] \qquad (2)$$

E_{max}, E_{min} : les énergies maximale et minimale présentes dans le rayonnement

La fluence à 'r' se calcule à partir d'une fluence $\Phi(\mathrm{E},\mathrm{r}_0)$ obtenue expérimentalement,.

$$\Phi(\mathrm{E},\mathrm{r}) = \Phi(\mathrm{E},\mathrm{r}_0)\left(\frac{\mathrm{r}_0}{\mathrm{r}}\right)^2 e^{\alpha(\mathrm{E},\mathrm{r})} \qquad (3)$$

où

$$\alpha(\mathrm{E},\mathrm{r}) = -\int_{r_0}^{r} u\big(E,\rho(r')\big)dr' \qquad (4)$$

r_0 : *le rayon de référence*
r : *le rayon du point d'interaction étudié*

On trouve donc deux facteurs déterminant la fluence : un premier en l'inverse du carré de la distance et un second exponentiel qui tient compte de l'atténuation par le milieu d'interaction.

1.1.2 DOSE

La deuxième étape de transfert d'énergie consiste à suivre le parcours des électrons qui ont été mis en mouvement lors de la première phase décrite à la section TERMA. Ces électrons ont une énergie limitée et, en conséquence, ils libèrent leur énergie dans un voisinage relativement proche du point d'interaction primaire (point de TERMA). Les

électrons en mouvement sont diffusés par le milieu environnant, laissant à chaque interaction une partie de leur énergie. Il arrive aussi que l'électron passe près d'un noyau atomique. Dans ce cas, l'électron subit l'attraction du noyau positif et peut subir une force importante. Ceci entraîne une décélération brusque et une perte d'énergie correspondante. L'énergie perdue apparaitra sous forme d'un nouveau photon qui pourra à nouveau interagir selon (TERMA). Ce phénomène porte le nom de *bremsstrahlung* (mot allemand signifiant « radiation de freinage »).

Une quantité supplémentaire peut être utile dans certains calculs de la dose : la KERMA (*kinetic energy released in medium,* énergie cinétique transférée au milieu) (Johns & Cunningham, 1983, pp. 218-219). Cette quantité n'est pas essentielle dans le contexte du présent travail mais apporte tout de même un éclairage intéressant sur le processus du dépôt de dose. S'il n'y avait pas de *bremsstrahlung,* la TERMA serait équivalente au KERMA. Notons aussi que, puisque les électrons dispersent leur énergie le long d'un chemin, la dose absorbée et la KERMA n'ont pas lieu au même endroit. C'est donc dire que nous ne pouvons utiliser la formule relativement simple de la KERMA pour calculer la dose absorbée à un endroit en particulier excepté dans quelques conditions particulières. On trouvera dans (Johns & Cunningham, 1983, pp. 220-224), une description de l'équilibre électronique des particules chargées. Ce qu'il est important de retenir est que les conditions de fluence de photons sont impossibles à atteindre dans une irradiation typique pour respecter l'équilibre des particules chargées (CPE). L'équilibre serait atteint seulement si le flux de photon n'était pas atténué en traversant le milieu puisqu'à mesure qu'on avance dans la matière, il y a moins de photons pour mettre en

mouvement des électrons. L'équilibre n'est aussi possible que dans la zone centrale de rayonnement. La notion de pénombre (section 1.2.2.3.2) expliquera cette dernière remarque. Par contre, une autre forme d'équilibre, TCPE (*transient charged particule equilibrium,* équilibre transitoire des particules chargées) est développée dans (Papanikolaou, 2004) qui relie la KERMA et la dose absorbée. Pour ce mémoire, il suffit de mentionner que la plupart des systèmes qui ne font pas un transport complet d'électrons vont assumer que la TCPE tient pour calculer la dose absorbée.

1.2 Algorithmes utilisés lors du calcul de la dose

Les premières méthodes de calcul de dose n'utilisaient qu'une feuille de calcul préformatée où le clinicien entrait la dose désirée et pouvait alors calculer la quantité de rayonnement à utiliser. Cette méthode n'était pas très représentative de la réalité puisqu'elle ne faisait pas intervenir les données spécifiques aux patients. Ces données, notamment la géométrie et la nature du milieu d'interaction (densité, composition chimique), sont pourtant essentielles pour décrire l'interaction entre le rayonnement ionisant et la matière composant le corps.

L'avènement de la tomographie calculée par ordinateur a permis aux cliniciens d'avoir accès aux données de densité électronique correspondant aux diverses structures corporelles. Des avancées informatiques ont aussi permis d'utiliser des méthodes de plus en plus sophistiquées pour calculer la dose déposée dans un patient lors d'une irradiation. Deux grandes familles de méthodes seront abordées ici : les méthodes par simulation de Monte Carlo et les méthodes de convolution/superposition. Il existe un

grand nombre d'autres méthodes, plus ou moins exactes, qui ne seront pas abordées ici. Le lecteur curieux pourra se référer à (Papanikolaou, Battista, Boyer, Kappas, & Mackie, 2004, pp. 27-65)

1.2.1 Méthode par simulation de Monte Carlo

Les simulations dites de type Monte Carlo ne sont pas limitées au domaine de la dosimétrie. Elles sont utilisées dans de nombreuses branches de la physique et des mathématiques. Elles sont basées sur une succession d'événements aléatoires et sur les probabilités décrivant un procédé pour en arriver à un résultat. Elles sont une alternative lorsqu'une solution analytique à un problème donné ne se résoudrait pas dans un temps raisonnable, même à l'aide d'un ordinateur (Metropolis, 1987). Dans certains cas, une solution analytique n'existe tout simplement pas, comme par exemple dans le problème à N corps[1], où N>2.

La méthode de Monte Carlo appliquée au problème de la dosimétrie utilise les procédés fondamentaux de transport et de diffusion d'électrons (Papanikolaou, Battista, Boyer, Kappas, & Mackie, 2004). Il a été mentionné plus tôt que diverses interactions peuvent avoir lieu lorsque les photons entrent en contact avec la matière. Or, les simulations de Monte Carlo utilisent les distributions de probabilité régissant ces phénomènes pour transporter les particules. La génération de millions de particules permet d'approcher une solution. L'utilisation d'un grand nombre de particules permet donc de trouver la dose à divers points puisque le nombre des collisions qui s'y sont produit est connu. La

1 http://en.wikipedia.org/wiki/N-body_problem

simulation de millions de particules requiert cependant un temps de calculs considérable, ce qui rend cette méthode incompatible avec le milieu hospitalier vu le délai requis pour obtenir le résultat d'une simulation. Certaines techniques de réduction de variance permettent de réduire le temps de calcul, voir par exemple (Rogers et al).

Le développement mathématique de la physique utilisé dans une simulation Monte Carlo ne sera pas détaillé ici puisqu'elle dépasse le spectre de ce mémoire. Un tel développement est détaillé dans (Papanikolaou, Battista, Boyer, Kappas, & Mackie, 2004).

La simulation Monte Carlo est en quelque sorte le but à atteindre, en termes d'exactitude, puisque qu'elle repose sur des lois physiques fondamentales régissant le transport des électrons et des photons dans la matière.

Des systèmes complets de transport des particules ont déjà été implémentés et sont disponibles gratuitement. Pour les applications médicales, notons les suites logicielles *Geant4*[2] développé au CERN et *BEAMnrc*[3] développé au Conseil national de recherche du Canada.

1.2.2 Méthodes par convolution/superposition

Les méthodes de calcul de dose utilisant les principes de convolution/superposition sont au cœur de ce travail. La formulation globale du calcul de la dose sera d'abord énoncée et sera ensuite explicitée en détails.

[2] http://lcgapp.cern.ch/project/simu/geant4/
[3] http://www.irs.inms.nrc.ca/BEAM/beamhome.html

La dose en un point donné est exprimée par (Papanikolaou, Battista, Boyer, Kappas, & Mackie, 2004):

$$\boldsymbol{D}(\vec{\boldsymbol{r}}) = \int \boldsymbol{T}\left(\overrightarrow{\boldsymbol{r'}}\right) * \boldsymbol{K}(\vec{\boldsymbol{r}}, \overrightarrow{\boldsymbol{r'}})\boldsymbol{d^3r'} \qquad (5)$$

où le vecteur $\vec{r}$r représente un point dans l'espace, $T(\vec{r})$ la TERMA au point $\vec{r}$ et $\boldsymbol{K}(\vec{\boldsymbol{r}}, \overrightarrow{\boldsymbol{r'}})$ est la fonction de diffusion de dose (FDD ou *kernel*, voir glossaire).

Le terme de TERMA a déjà été explicité à la section TERMA. Le terme de FDD a trait à la deuxième étape du calcul, exprimée comme l'étape DOSE précédemment. L'algorithme de superposition voit chaque point de dépôt primaire de dose (TERMA) comme une source de radiation dans le corps. La FDD représente une fonction de distribution de l'énergie. Comme l'équation 5 le présente, pour obtenir la dose au point $\vec{r}$r, une contribution de tous les points de l'espace doit être sommée, à l'aide de la FDD, pour connaître la contribution de la TERMA d'un point $\vec{r}$r' à la dose au point $\vec{r}$r. On peut donc voir chaque point de l'espace comme une source de radiation avec une intensité égale à la TERMA, et c'est la *superposition* de l'effet de ces sources par *convolution* en chaque point qui constitue le calcul de la dose. Cette opération est très coûteuse en termes de calcul. Sous sa forme la plus simple, en examinant l'équation, on voit qu'elle est de l'ordre de $O(n^6)$ (où n représente le nombre de voxels) lorsque le calcul se fait en trois dimensions. En effet, une triple intégrale, donc sommation, doit être effectuée pour chaque point de l'espace.

La solution habituelle pour diminuer la puissance de calcul requise lors d'une convolution est de résoudre le problème dans le domaine des fréquences et d'utiliser la transformée de Fourrier rapide. Cet artifice mathématique transforme la convolution dans le domaine spatial en une simple multiplication dans le domaine des fréquences. Ceci nous permet de réduire la complexité d'une convolution à une dimension, qui est quadratique ($O(n^2)$) en une complexité en O(n*log(n)) (où n représente le nombre de voxels dans une dimension). Par contre, le théorème de la convolution (Bracewell, 2009) ne s'applique que si le filtre de la convolution est spatialement invariant. Or, ce n'est pas le cas ici pour des raisons qui seront mises en évidence dans les sous-sections suivantes.

Les FDD, le calcul de la TERMA et quelques considérations particulières seront présentés dans les prochaines sections. La référence principale pour la réalisation de ce mémoire, (Liu, Mackie, & McCullough, 1997) (Liu, Mackie, & McCullough, 1997), sera aussi revue plus en profondeur.

1.2.2.1 Les fonctions de diffusion (kernels)

La FDD, comme son nom l'indique, sert à distribuer la dose primaire (TERMA) dans son entourage. La FDD peut donc être vue comme une matrice dont l'élément e[i,j] représentent le ratio de la dose totale qui est distribuée à la position [i,j].

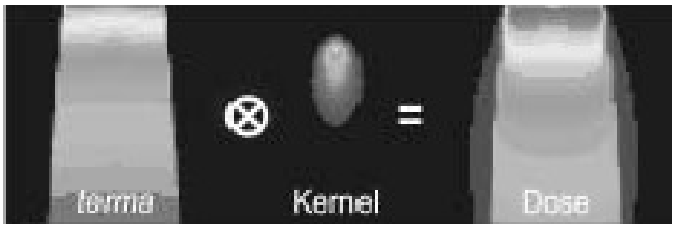

Figure 1-1 Kernel (Papanikolaou, Battista, Boyer, Kappas, & Mackie, 2004)

La Figure 1-1 représente l'effet de la convolution entre la TERMA et la FDD. Nous retrouvons donc une dose finale où l'effet de plusieurs points de TERMA contribue à l'augmentation de la dose environnante. Les FDD peuvent être générées par des simulations Monte Carlo (Mackie, Bielajew, Rogers, & J, 1988) ou de façon analytique (Anhesjö, 1989). On remarque aussi que la FDD est orientée principalement vers le bas de l'image. Ceci s'explique par le mouvement initial des photons qui sont partis de la tête de l'accélérateur linéaire (dans le haut de l'image) et qui mettent en mouvement des électrons vers le bas principalement.

Les FDDs sont généralement obtenues à l'aide de simulations de Monte Carlo. Un faisceau mince de photons est forcé d'interagir en un seul point dans un environnement uniforme (un bloc d'eau par exemple). Le faisceau est typiquement composé de millions de photons mono énergétiques. Le logiciel de transport Monte Carlo se charge alors de déplacer les électrons qui ont été mis en mouvement suite à l'interaction des photons avec la matière. Pour obtenir la FDD, il suffit de comptabiliser les dépôts d'énergie dans les différents voxels environnant le point d'interaction. Ce processus est lancé pour plusieurs valeurs d'énergies de photons dans le but de bien modéliser le spectre polyénergétique de l'accélérateur linéaire.

Tel que mentionné dans la section Méthodes par convolution/superposition, bien que le calcul de convolution puisse être effectué à l'aide de la transformée de Fourier rapide, la réalité en est autrement. En effet, pour pouvoir utiliser le théorème de convolution et par conséquent la transformée de Fourier rapide, la FDD devrait être spatialement invariante. Or, ce n'est pas le cas ici, ce qui sera démontré dans le raisonnement qui suit. La FDD est orientée selon la ligne qui lie le point de dépôt de dose dans le corps au point d'émission du rayonnement, qui en première approximation est ponctuelle. Un vecteur d'orientation différent reliera les positions dans le patient à la source de rayonnement. Comme la FDD sera alignée sur ce vecteur, toutes les FDD seront donc aussi alignées différemment. La Figure 1-2 illustre cette situation.

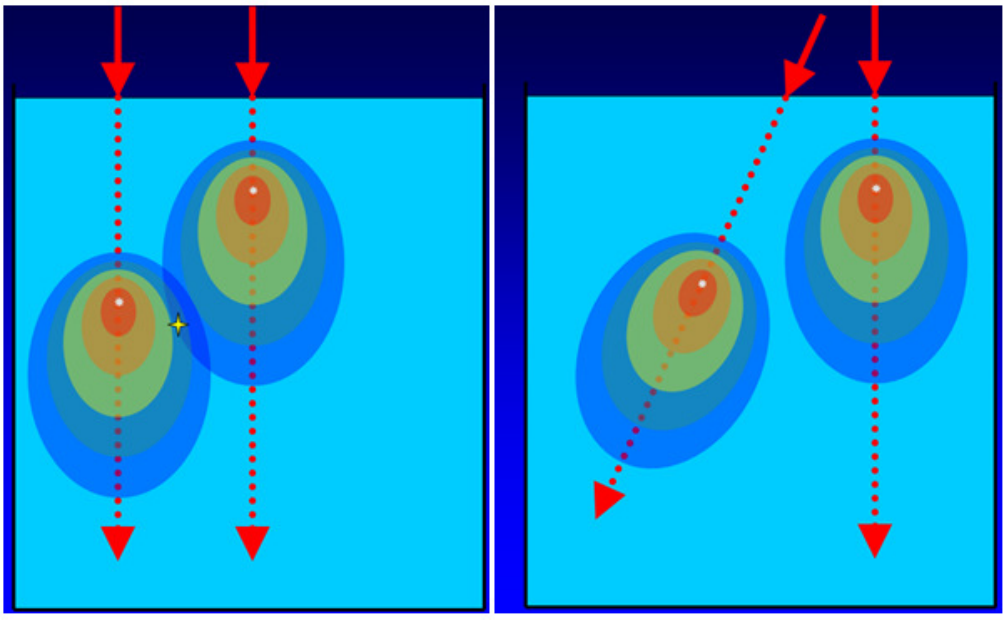

Figure 1-2 Illustration de l'orientation de la FDD (Tiré de « Convolution/Superposition Dose Method in PLanUNC »)

La Figure 1-2 de gauche montre des lignes de rayonnement primaires parallèles qui émaneraient d'une source ponctuelle située à une distance infinie. La Figure 1-2 de droite représente des lignes de rayonnement sortant d'une source de radiation ponctuelle située à une distance finie. Cette situation représente une source de rayonnement dans un accélérateur linéaire, qui peut être considérée ponctuelle en première approximation. Nous voyons donc que les FDD ne seront pas spatialement invariantes à travers tout le corps puisque celles-ci dépendent de la ligne joignant le point de calcul à la source ponctuelle. De plus, comme il sera expliqué à la section 1.2.2.3, le spectre du faisceau change à mesure que l'on pénètre dans le patient. Ceci fait varier la FDD en fonction de la profondeur. Comme la convolution ne dépend plus seulement de la position relative entre le point de calcul et le point de convolution dans le filtre, nous ne pouvons utiliser le théorème de convolution. De plus, comme le corps n'est pas uniforme, la FDD doit être déformée pour que les différents facteurs la formant coïncident toujours avec les distances radiologiques correspondantes dans le bloc d'eau de référence. Ceci élimine du même coup la possibilité d'utiliser une transformée de Fourier rapide.

La méthode pour orienter les FDD est présentée dans la référence principale qui a été utilisée pour réaliser ce mémoire : (Liu, Mackie, & McCullough, 1997).

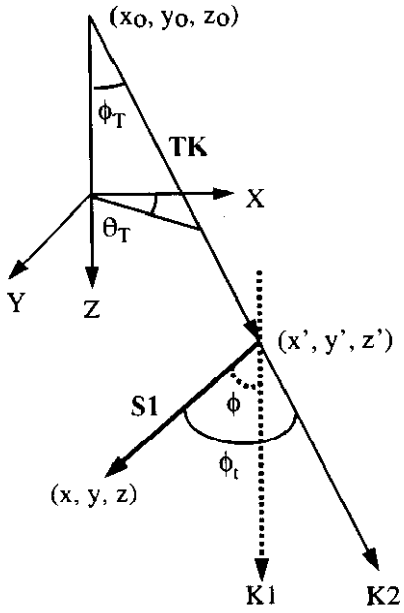

Figure 1-3 Inclinaison de kernel (Liu, Mackie, & McCullough, 1997)

La Figure 1-3 représente l'ensemble des éléments nécessaires pour effectuer l'inclinaison de la FDD. Le point (x_0,y_0,z_0) représente la source de rayonnement, (x',y',z') représente le point d'interaction primaire (TERMA) et le point (x,y,z) représente le point de calcul de la dose. Le vecteur K1 représente la ligne que suivrait la FDD si une source de rayonnement située à une distance infinie était considérée alors que le vecteur K2 représente l'orientation correcte de la FDD qui utiliserait une source ponctuelle de rayonnement. L'essentiel de la solution repose dans le calcul d'un nouvel angle, φ_t, entre le vecteur K2 et le vecteur S1, qui lie le point d'interaction primaire au point de calcul. Cet angle servira donc pour trouver la région de la FDD dans laquelle nous nous trouvons. Puisque celle-ci est rotationellement invariante et symétrique autour de la ligne reliant son origine (x',y',z') et la source de rayonnement, seule la distance *r* entre (x,y,z) et (x',y',z') et l'angle φ_t sont nécessaires pour définir uniquement la valeur

recherchée dans la FDD. Dans(Liu, Mackie, & McCullough, 1997), cet angle est défini comme :

$$\varphi_t = \text{acos}\left(\frac{\overrightarrow{S1}\cdot\overrightarrow{TK}}{|\overrightarrow{S1}|\cdot|\overrightarrow{TK}|}\right) \quad (6)$$

Ces données sont donc suffisantes pour pouvoir utiliser une seule fonction de diffusion mais qui sera inclinée, en pratique, par le choix de l'indice de l'élément souhaité. En d'autres mots, plusieurs FDD orientées différemment ne seront pas générées mais une seule FDD sera échantillonnée de manière à refléter son inclinaison.

1.2.2.2 Le calcul de la TERMA

La TERMA peut être calculée de différentes façons, plus ou moins précises, en commençant par des méthodes qui n'utilisent pas de données spécifiques au patient jusqu'à la méthode plus exacte utilisée en convolution/superposition. Le calcul idéalement doit prendre en compte les inhomogénéités du corps. Ceci peut se faire par le calcul dans un milieu de densité uniforme suivi de l'utilisation de facteurs de correction, ou dans un milieu de densité hétérogène. Seule la méthode utilisée pour ce travail sera décrite. Le lecteur curieux est invité à consulter (Papanikolaou, Battista, Boyer, Kappas, & Mackie, 2004) précisant les méthodes par facteur pour les corrections d'inhomogénéités.

Le calcul de la TERMA présenté ici se fait à l'aide d'un tracé de rayons. Chaque rayon part du point d'émission du rayonnement et traverse le milieu d'interaction pour atteindre le point de calcul. La fluence du faisceau considéré, connue à la surface du

corps, est alors atténuée pour prendre en compte sa distance parcourue dans le corps. Or, comme il a été mentionné, le corps est hétérogène en densité et les calculs doivent idéalement en tenir compte. En effet, le rayonnement sera moins atténué dans un milieu de densité plus faible (section 1.1.1). Il faut donc que le tracé de rayons prenne en compte, non pas la distance géométrique entre la surface et le point d'interaction, mais la distance dite radiologique (Papanikolaou, Battista, Boyer, Kappas, & Mackie, 2004):

$$d' = \sum_i (\Delta d_i)\rho_i \qquad (7)$$

où d' représente la longueur radiologique, d la longueur géométrique et ρ la densité.

Le travail de pionnier de (Siddon, 1985) a permis de rendre le calcul des différentes longueurs radiologiques possibles dans un temps adéquat pour une utilisation clinique. L'algorithme a été amélioré par (Christiaens, Sutter, Bosschere, Campenhout, & Lemahieu, 1998). La méthode se base sur le fait que les voxels sont inscrits dans un repère orthogonal et régulier. Puisque la distance entre chaque indice pour les trois axes est connue et partout la même, il est possible de savoir dans quelle direction (x, y ou z) sera rencontré le prochain voxel. Le nombre de voxels traversés est alors facilement récupérable dès le départ dans un temps constant et les tests de collisions entre rayons et voxels sont éliminés. Il suffit de suivre le rayon du début à la fin, amassant les informations à mesure que l'on croise des voxels.

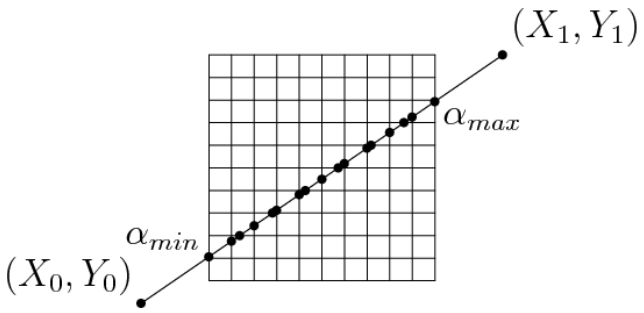

Figure 1-4 Variables utiles pour l'algorithme de Siddon

La Figure 1-4 présente les variables utiles pour l'algorithme de Siddon. Un point de départ (X_0,Y_0) et d'arrivée (X_1,Y_1) doivent être définies. Connaissant les caractéristiques de la grille de pixel (ou de voxel), les valeurs α_{min} et α_{max}, représentant les distances d'entrée et de sortie sur la droite liant les deux points, peuvent être trouvées. Par la suite, les différents points d'intersection de la droite avec une des lignes formant la géométrie peuvent être trouvées itérativement suivant l'implémentation améliorée. L'algorithme dans son ensemble n'est non pas proportionnel à $O(n^3)$ (où n représente le nombre de voxels dans une dimension) comme on pourrait s'y attendre en traçage de rayon traditionnel mais bien en O(3n), soit le nombre de plans formant la géométrie.

1.2.2.3 Éléments influençant le calcul de la TERMA et de la DOSE

Plusieurs éléments doivent être considérés lorsqu'un calcul par convolution/superposition est effectué. Les suivants seront détaillés : le durcissement du faisceau, les FDDs polyénergétiques, les pénombres, la fluence réelle, la contamination d'électrons et l'adoucissement du spectre hors axe.

1.2.2.3.1 Durcissement du faisceau

Nous avons précédemment exposé qu'une seule FDD est utilisée lors des calculs. Celle-ci peut être inclinée, mais ses valeurs restent les mêmes. Or le faisceau, lorsqu'il traverse le corps, ne reste pas inchangé. Le faisceau est poly énergétique, c'est-à-dire qu'à la sortie de la tête de l'accélérateur linéaire, son spectre est continu. Un faisceau de photons de 15MV, par exemple, contient des photons d'énergie comprise entre 0 et 15 MeV. À mesure que le faisceau progresse dans le milieu d'interaction, le corps du patient dans ce cas-ci, il perd progressivement ses composantes à basse énergie qui sont absorbées préférentiellement dans le corps. C'est donc dire que son énergie moyenne augmente. Ce phénomène se nomme le durcissement. Ce phénomène interviendra donc principalement lors de la phase de TERMA, lors de laquelle il faudra tenir compte de la qualité du rayonnement en chaque point dans le patient avant d'établir la valeur de la TERMA à ce point. La qualité du rayonnment détermine grosso modo sa composition spectrale en termes de dureté; un faisceau d'énergie moyenne plus élevé est dit plus dur.

Pour tenir compte du durcissement de rayon lors de la phase DOSE, Liu (Liu, Mackie, & McCullough, 1997) utilise une méthode d'interpolation. Trois FDDs sont utilisées : une à la surface, une autre à une profondeur prédéterminée dans le patient et une autre à mi-chemin entre ces positions. Ces différentes FDD sont toutes générées à partir d'une FDD à une position où les composantes spectrales sont connues, spectre qui sera modulé par l'atténuation exponentielle correspondant à la profondeur considérée pour obtenir les autres FDD. Nous nous trouvons donc avec un certain nombre de FDDs lors de la phase de DOSE et selon la position considérée, nous choisissons les FDD se trouvant de part

et d'autre du point d'interaction, une interpolation ayant lieu entre la FDD supérieure et inférieure.

FDD polyénergétiques

L'utilisation d'un spectre d'énergie, comme présenté au point précédant, amènerait l'utilisation d'une FDD pour chaque plage d'énergie. Il faudrait donc réaliser plusieurs convolutions, chacune avec sa FDD spécifique. Or, plusieurs auteurs, dont (Metcalfe, Hoban, Murray, & Round, 1990) (Metcalfe, Hoban, Murray, & Round, 1990), (Hoban P. W., 1995) (Hoban P. W., 1995), ont trouvé que le gain en précision introduit par ce calcul ne vaut pas le temps de calcul ajouté. Papanikolaou (Papanikolaou, Mackie, Meger-Wells, Gehring, & Reckwerdt, 1993) a développé une méthode pour générer une FDD unique à partir d'un spectre d'énergie. Cette méthode utilise, en entrée, une FDD pour chaque plage d'énergie du spectre, pour ensuite en faire une somme pondérée par la contribution de chaque plage au spectre total, pour fournir en sortie une FDD composite qui sera utilisée lors des calculs de convolution.

1.2.2.3.2 Pénombre

Un autre aspect dont il faut tenir compte est la zone de pénombre (Jaffray, Battista, Fenster, & P., 1993) et (Sharpe, Jaffray, Battista, & Munro, 1995) indique que le phénomène de pénombre se divise en fait en deux phénomènes : la pénombre géométrique et la pénombre radiologique.

La pénombre géométrique (Figure 4) vient du fait que la source de rayonnement n'est pas vraiment ponctuelle mais a une certaine taille. Il y a donc une zone, définie par la

position des collimateurs, de la taille et la position de la source de rayonnement et de la distance au plan de calcul, où la radiation ira en diminuant à mesure que l'on s'éloigne de l'axe principal d'irradiation.

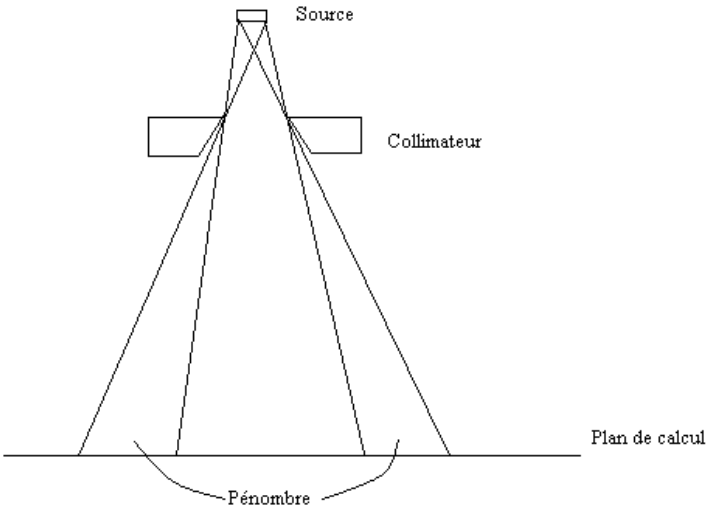

Figure 1-5 Pénombre géométrique

La pénombre radiologique vient de l'interaction entre le rayonnement venant de la source primaire et le médium irradié. Ces interactions produisent du rayonnement diffus qui agit, comme il a été démontré, comme une source de rayonnement secondaire. Ceci fait en sorte que la zone qui reçoit le rayonnement n'est pas définie complètement par les collimateurs puisque le rayonnement qui arrive aux limites de la zone permise par les collimateurs agit lui-même comme une source de rayonnement pour ses voisins immédiats.

1.2.2.3.3 Fluence réelle

La fluence arrivant au corps doit elle aussi être étudiée un peu plus en profondeur. (Sharpe, Jaffray, Battista, & Munro, 1995), basé sur (Jaffray, Battista, Fenster, & P., 1993), montre que la fluence peut être vue comme ayant deux composantes : une fluence focale et une fluence hors focale. La fluence focale est celle attendue dans le cas idéal, où tout le rayonnement provient du point focal de l'accélérateur linéaire. La fluence hors focale est créée par l'interaction du rayonnement primaire avec les autres composantes de la tête de l'accélérateur linéaire (filtre égalisateur, chambres d'ionisation, collimateurs). Ces interactions, à l'image de celles dans le corps, forment des sites d'émission de rayonnement secondaire qui peuvent atteindre le corps, à l'intérieur ou à l'extérieur des zones collimées. Ce phénomène peut compter pour près de 10% de la fluence totale arrivant à la cible. Il a été démontré que cette fluence hors focale est proportionnelle à la fluence focale. Le développement de ce facteur est disponible dans (Sharpe, Jaffray, Battista, & Munro, 1995).

Contamination d'électrons

Un phénomène similaire à celui présent au paragraphe précédant est la contamination d'électrons. Comme pour la fluence réelle, ce phénomène fait intervenir des photons qui interagissent dans la tête de l'accélérateur. Sauf que dans le cas présent, ce sont des électrons qui sont mis en mouvement et qui vont ultimement entrer en contact avec le corps. Or, les électrons étant des particules chargées, ils vont interagir rapidement avec la matière, la peau. C'est donc dire que la contamination d'électrons apporte une dose

supplémentaire en surface par rapport à ce qu'on pourrait s'attendre si la contamination d'électrons n'était pas prise en compte. Ceci est particulièrement important lors de l'utilisation de faisceau à hautes énergies, qui sont utilisées pour traiter en profondeur, puisqu'il faut aussi maintenant tenir compte d'une dose déposée en surface. L'auteur de (Starkschall, 2000) (Starkschall, 2000) énonce une technique pour évaluer la contamination d'électrons. La dose due à la contamination d'électrons, D_e, se calcule comme la multiplication d'un facteur dépendant de la profondeur, F_d, et un dépendant du facteur hors-axe, F_{0A}.

$$D_e(r, d, fs) = F_D(d, fs)F_{0A}(r) \quad (8)$$

$$F_{0A}(r) = e^{-A\theta^2} \quad (11)$$

où d est la profondeur et fs la coordonnée de surface. L'expression de Fd ne sera pas présentée ici puisqu'elle fait appel à un ensemble de facteurs propres à l'appareil utilisé qui ne sont pas nécessaires à la compréhension du phénomène. Disons simplement que le facteur F_d varie en trois phases : linéairement près de la surface, exponentiellement plus loin, et vaut 0 passé une certaine profondeur seuil.

Adoucissement hors axe

À la sortie de la tête de l'accélérateur, le flux de photons passe par un filtre égalisateur. Ce filtre comporte plus de matière au centre et moins sur les côtés pour égaliser le faisceau. Comme il a été montré au paragraphe sur le durcissement de rayon, plus le faisceau doit traverser de matière, plus il perd ses composantes en basse énergie. Ce

phénomène se produit donc aussi au niveau du filtre égalisateur. Ceci résulte en un faisceau qui comporte plus de composantes à basse énergie en périphérie du filtre égalisateur. L'auteur de (Starkschall, 2000) (Starkschall, 2000) énonce une technique pour tenir compte de cette variation qui est facile à implémenter. Pour chaque point, nous pouvons calculer le facteur d'adoucissement de chaque composante du spectre comme :

$$\left[\frac{1}{1+\left(\frac{E_i}{E_{max}}\right)}\right]^{S\theta} \qquad (12)$$

où S représente un facteur d'adoucissement général, θ l'angle par rapport à l'axe central, E_i l'énergie en cours et E_{max} l'énergie maximale du spectre.

Les éléments de la section 1.2.2.3 sont donc particulièrement importants lorsque la TERMA est calculée puisqu'ils font intervenir des caractéristiques de l'accélérateur linéaire et du rayonnement primaire qui y est produit. La méthode de convolution/superposition modélise déjà ces phénomènes de façon inhérente (pénombre radiologique) ou peut être facilement adaptée (durcissement de rayon, pénombre géométrique). La fluence réelle demande des modèles à deux sources de radiation et peut s'intégrer dans cette technique de calcul de dose mais d'une façon moins évidente. La phase de convolution restera la même mais la phase de TERMA devra tenir compte des deux sources de radiation.

1.3 Architecture matérielle et logicielle de développement

Ce chapitre comportera une courte description de l'environnement de développement logiciel et matériel qui a été utilisé pour réaliser le présent travail. La plateforme matérielle est la famille des cartes d'accélération graphique de NVIDIA qui supporte l'environnement logiciel CUDA (NVIDIA, 2008).

1.3.1 Paradigme de programmation

Le développement des processeurs multi cœurs et l'émergence de la programmation des cartes graphiques ont ramené la programmation parallèle à l'avant-scène du calcul haute performance. Le paradigme matériel vise à réduire la vitesse et la complexité des unités d'exécution, mais d'en augmenter substantiellement le nombre. Nous trouvons donc, dans la dernière gamme des cartes NVIDIA (GeForce GT200)[4], 30 multiprocesseurs de huit processeurs chacun, pour un total de 240 processeurs par carte. Ceci donne un avantage remarquable aux cartes graphiques en ce qui concerne la puissance de calcul par rapport aux processeurs à usage général (Figure 1-6).

[4] Automne 2008

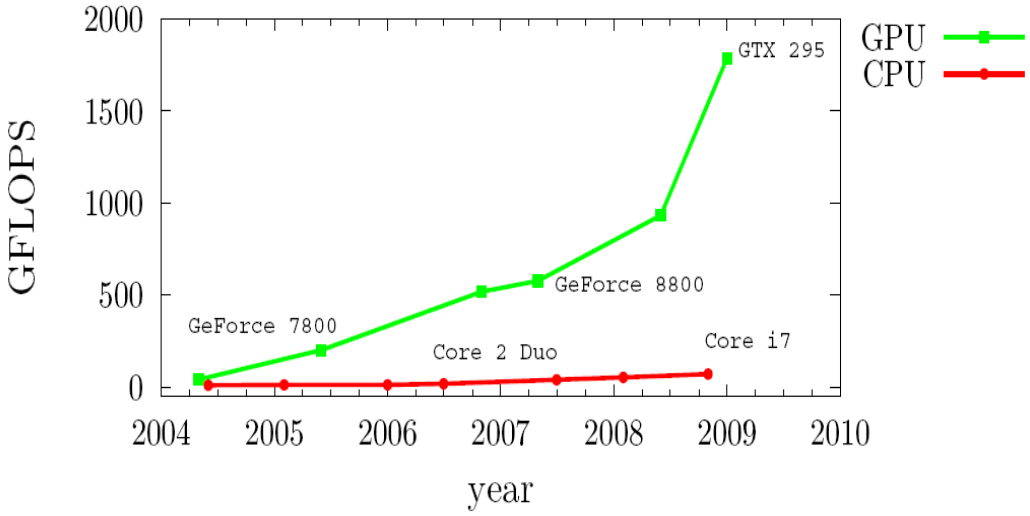

Figure 1-6 Évolution de la puissance de calcul (NVIDIA, 2008)

1.3.2 Plateforme matérielle

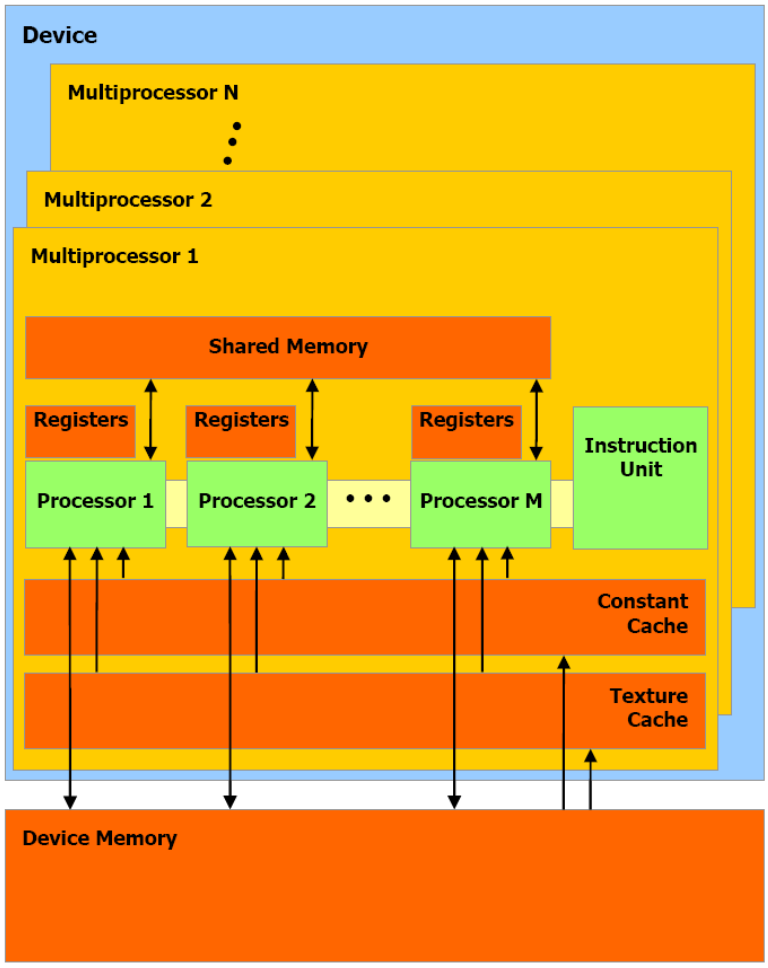

Figure 1-7 Architecture matérielle

La Figure 1-7 présente l'architecture matérielle dans son ensemble. Nous retrouvons donc plusieurs niveaux de granularité ainsi qu'une hiérarchie de mémoire. La mémoire principale est la plus large et la plus lente. Il y a ensuite, pour chaque multiprocesseur de la carte, une mémoire partagée pour un bloc de processeur. Cette mémoire partagée peut avoir un accès aussi rapide que les registres. Pour chaque multiprocesseur, il y a aussi une mémoire constante qui peut être écrite hors des programmes CUDA pour utilisation

en lecture seule à l'intérieur de ceux-ci et une cache de texture pour aider à la localité des lectures de texture. Finalement, chaque processeur a accès à un ensemble de registres.

L'architecture matérielle a été élaborée avec une vision SIMT (*single instruction multiple thread).* Chaque processus léger (*thread*) est assigné à un processeur et un nombre largement supérieur de processus légers, par rapport au nombre de processeurs physiques sur la carte, peut être maintenu par l'ordonnanceur. Ces processus légers sont groupés en *warps* de 32 processus légers qui s'exécuteront physiquement de façon parallèle. Chaque processeur du bloc de processeurs exécutant un *warp* effectue les mêmes opérations que les autres processeurs du bloc mais sur d'autres données en entrée. Même au niveau du contrôle de flot et des conditions, si certains processeurs légers à l'intérieur du *warp* doivent exécuter une instruction différente, le *warp* sera brisé en autant de morceaux que nécessaire pour que la totalité d'un morceau exécute la même instruction et puisque la taille du morceau sera plus petite que la taille d'un *warp*, certains processeurs physiques seront en mode d'attente. Il est donc important de porter attention au contrôle de flot dans une application CUDA puisque si le flot est très divergent, on s'éloigne de l'usage optimal des ressources matérielles.

1.3.3 Plateforme logicielle

La plateforme logicielle de CUDA est une extension au langage C. Le compilateur de NVIDIA, nvcc, prend en charge les extensions au langage C et génère un fichier objet

qui peut ensuite être lié par n'importe quel éditeur de liens prenant en charge le langage C.

Les bouts de programmes s'exécutant sur la carte graphique portent, en suivant le vocabulaire NVIDIA, le nom de *kernel*. Ces *kernels* sont eux aussi écrits en C. Les *kernels* ont accès aux fonctionnalités habituelles du langage C ainsi qu'à une quantité importante de fonctions utilitaires et de fonctions donnant accès à des éléments matériels de la carte. Mentionnons, par exemple, la capacité d'aller chercher un élément dans une texture (en deux ou trois dimensions) avec ou sans interpolation, tout ceci étant réalisé physiquement sur la carte.

Avant de pouvoir exécuter un *kernel* sur la carte, le programme s'exécutant sur la machine hôte doit mettre en place la configuration dans laquelle le *kernel* s'exécutera. Entre autres, il faut définir les dimensions des « grilles » et « blocs » de processus légers.

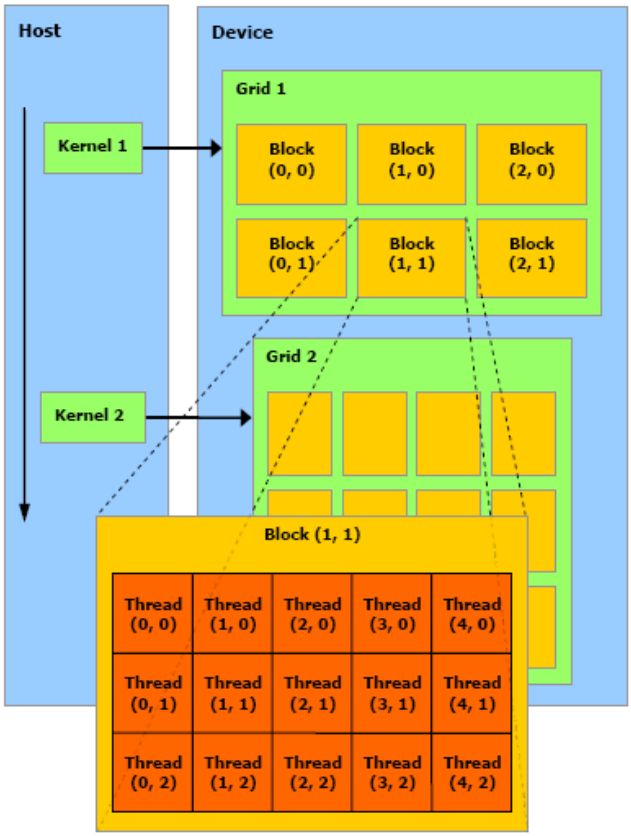

Figure 1-8 Configuration des processus légers

Le programme hôte doit aussi allouer l'espace mémoire à l'intérieur de la mémoire principale sur la carte graphique pour ensuite copier les données de la machine hôte vers la mémoire principale. Le programme hôte est alors prêt à lancer le *kernel*.

À l'intérieur de ce dernier, des variables propres aux *kernels* sont définies : blockIdx, blockDim, et threadIdx. Ces variables servent à identifier, de façon unique, chaque processus léger faisant partie de l'exécution d'un *kernel*. Suite à cette identification, chaque processus léger peut aller charger les données qui lui sont spécifiques en

mémoire globale, faire son traitement et réécrire les données sortantes en mémoire globale.

Le programme hôte peut alors copier les données de la mémoire principale de la carte graphique vers la mémoire de l'hôte, le traitement de la carte graphique étant terminé et le programme hôte continuant son exécution.

Mentionnons finalement un élément du modèle CUDA qui aura une certaine importance à travers ce document : les modes de lecture de mémoire. Lorsque des données sont stockées en mémoire globale, il est possible de les lire dans un ordre qui augmentera grandement la bande passante effective du matériel. En effet, lorsque tous les processus légers d'un même *warp* accèdent à des éléments contigus en mémoire, le contrôleur de mémoire est capable de grouper ces accès mémoire en un seul accès mémoire de grande taille (jusqu'à 128 octets). Ce mode de lecture sera dénommé groupée (*coalesced* en terminologie CUDA). Cette méthode de lecture est donc fortement souhaitable pour augmenter la bande passante effective et réduire le nombre d'accès mémoire individuels qui doit être effectué.

Cette vue à très haut niveau n'est évidemment pas suffisante pour qu'un novice puisse commencer à écrire des programmes avec CUDA. Les lecteurs voulant continuer leur apprentissage peuvent se référer à (NVIDIA, 2008). Les concepts nécessaires à notre application seront aussi définis plus en détails lorsque nécessaire.

Alternatives logicielles

Comme nous nous intéressons à une programmation parallèle, il est important de mentionner les autres possibilités qui pourraient être utilisées. Outre la programmation par processus de base « manuelle » où le programmeur crée ses propres processus ou processus légers en faisant des appels du système d'exploitation, nous retenons deux possibilités : OpenMP et MPI.

OpenMP est une interface de programmation (application programming interface, API) pour calcul parallèle par mémoire partagée. C'est donc dire que le parallélisme est limité à un seul ordinateur et une seule mémoire principale. Le parallélisme s'insert d'une façon très simple et non invasive à l'aide de quelques appels de préprocesseur placées devant les endroits dans le code source devant être parallélisés. L'API est disponible sur la plupart des grandes plateformes, notamment Windows et UNIX.

MPI (*message passing interface)* est une bibliothèque de fonction permettant de paralléliser les calculs sur divers ordinateurs, et donc en mémoire distribuée, à l'aide de passage de messages. L'ajout du parallélisme se fait beaucoup plus difficilement requérant une plus grande participation du programmeur qui doit identifier les ressources de calculs à sa disposition, partitionner manuellement la charge de calcul et gérer les communications et la synchronisation entre les divers nœuds de calcul.

1.4 PlanUNC

PlanUNC[5] est un logiciel complet de planification de traitement en radiothérapie développée par l'unité de radio-oncologie de l'Université de Caroline du Nord. Le code source de l'application est disponible à qui en fait la demande et peut donc être facilement modifié par ses utilisateurs. PlanUNC nous permet donc de nous concentrer sur la partie de calcul de dose puisque le reste du flot de planification est déjà fourni. C'est donc PlanUNC qui gère la lecture des images médicales, le traçage de contours d'organes, la génération des isodoses, la définition des accélérateurs linéaires, etc. PlanUNC a été développé avec une architecture complètement modulaire. Chacun des modules représente une tâche accomplie par le programme, comme ceux que nous venons d'énumérer. Notre travail viendra donc remplacer le module de calcul de dose déjà présent dans PlanUNC. Il faut donc que nous nous assurions de respecter les normes établies et utilisées dans le reste de l'application. Notre solution sera donc une librairie statique, comme tous les autres modules, qui sera utilisée pour construire l'application finale. Notre méthode de calcul de dose est donc un ajout qui ne changera que très peu l'application de base puisque nous ne rajoutons que des items au menu principal de l'application pour permettre la sélection de notre méthode de calcul de dose.

[5] http://planunc.radonc.unc.edu/

1.5 But, objectifs et hypothèses

Le but général de ce projet est d'améliorer les temps d'exécution en calcul de dose. L'objectif principal est donc en terme de performance. Un objectif secondaire est de, suite à la diminution du temps d'exécution, d'intégrer un modèle physique plus complet pour permettre une plus grande exactitude des résultats. Nous tenterons d'atteindre ces objectifs à l'aide de deux implémentations différentes : un port de l'algorithme de PLUNC pour des gains en vitesse et une réécriture de l'algorithme pour des gains en vitesse et en exactitude.

Nous souhaitons ramener le temps du calcul de la dose, avec la méthode de convolution/superposition, à environ une seconde par faisceau. Le temps actuel, pour un faisceau, est d'environ 15 secondes. Ceci améliorerait donc la solution actuelle de PlanUNC d'environ 15x.

Nous souhaitons aussi améliorer l'exactitude des calculs, par rapport à la solution de PlanUNC, en faisant des tracés de rayon à travers le patient pour les étapes de TERMA et de DOSE. L'implémentation actuelle de convolution/superposition dans PlanUNC ne fait des tracés de rayons que dans la phase de TERMA et ceux-ci, même à cette étape, sont approximatifs.

L'hypothèse principale est que l'utilisation d'une carte graphique moderne, couplée à la plateforme logicielle CUDA, nous permettra d'atteindre nos objectifs. Nous postulons de plus que l'effort pour déployer une solution CUDA sera moindre que celle qui devrait être fournie pour utiliser une implémentation faisant appel à plusieurs nœuds de calcul à

l'aide de API (*application programming interface,* interface de programmation), tout en coûtant moins cher à l'achat et à l'utilisation. L'approche CUDA sera aussi plus performante que le meilleur des cas possible à l'aide d'une implémentation utilisant OpenMP, étant donné le nombre limités de cœurs d'exécutions disponibles en mémoire partagée, mais un peu plus demandant pour le programmeur.

CHAPITRE 2 : MÉTHODOLOGIE

Ce chapitre porte sur le développement des solutions que nous avons utilisées pour répondre à la problématique. Le chapitre est divisé en trois sections. Tout d'abord, nous exposerons les grandes lignes des deux solutions que nous avons implémentées pour répondre à nos objectifs. Les méthodes de calcul de doses qui ont été retenues pour nos solutions seront énumérées. Ensuite, nous expliquerons où et comment nous nous insérons dans PLUNC. Finalement, les implémentations sur carte graphique comme telles seront détaillées.

2.1 Implémentations et méthodes de calcul de dose retenues

Dans la première partie de notre solution, l'algorithme de PLUNC est utilisé comme tel au point de vue des opérations à effectuer sur les données mais sera transposé vers une utilisation sur le GPU. Ceci a pour but d'estimer directement le potentiel d'accélération du GPU ainsi que de quantifier l'erreur (si erreur il y a) entrainée par l'utilisation d'une carte graphique pour faire les calculs de dosimétrie.

Nous développons ensuite notre propre solution au calcul de dose. Cette solution a pour but de démontrer le gain en vitesse d'exécution qu'il est possible d'obtenir lorsque l'on développe une implémentation ayant un GPU comme plateforme cible dès le départ. La méthode de convolution/superposition décrite à la section 1.2.2 a été utilisée. Nous tenons aussi compte de l'inclinaison des FDD présentée à la section 1.2.2.1. Notre calcul de TERMA prendra en compte les inhomogénéités du patient comme présenté à la

section 1.2.2.2. Notons que l'étape de DOSE utilisera aussi les données spécifiques du patient (images) pour tenir compte des inhomogénéités.

La méthode Monte Carlo a été rapidement éliminée comme solution possible à implémenter dans le contexte de ce travail. Nous croyons que la carte graphique pourrait s'avérer une candidate intéressante pour l'exécution d'un algorithme basée sur les simulations Monte Carlo mais la taille gigantesque des programmes de transport de photons et d'électrons fait en sorte que cette tâche d'envergure n'est pas compatible avec le temps alloué pour effectuer ce travail. Nous encourageons par contre d'autres équipes qui, à la lecture de nos résultats, pourraient être intéressées à réécrire des programmes faisant des simulations de type Monte Carlo vers le GPU. Nous pensons par exemple qu'un générateur de nombres aléatoires pourrait être implémenté sur GPU pour décharger le CPU de cette tâche.

2.2 Insertion dans PLUNC

PLUNC est construit sur un système de librairies statiques. Chaque librairie a une fonction et l'ensemble de ces librairies est requis pour l'assemblage final de l'exécutable. Notre travail est donc de remplacer la librairie de calcul de dose par notre propre version. PLUNC sera donc construit à l'aide de notre librairie et tous les appels aux fonctions de calcul de dose seront automatiquement réacheminés vers notre nouvelle librairie.

La librairie de calcul de dose est libconv (*MASTER\src\dose\libconv*). À l'intérieur de celle-ci, deux fichiers nous intéressent : terma.cxx et convolution.cxx. Deux fonctions

devront être modifiées : *terma_thread* et *convolve_grid_thread.* Nous insérerons donc nos appels aux *kernels* CUDA à l'intérieur de ces fonctions et ce sont donc à l'intérieur de celles-ci que le branchement vers l'exécution sur la carte graphique sera effectué.

Nous compilons ensuite les fichiers terma.cxx et convolution.cxx pour obtenir leur fichier objet respectif. Ces fichiers objets sont réinsérés dans la structure de fichier originale de PLUNC. Ils seront donc utilisés pour construire libconv, qui a son tour sera utilisée pour construire l'exécutable. De cette façon, nous pouvons garder le script d'installation de PLUNC sans aucune modification, excepté l'inclusion de la librairie CUDA pour l'édition des liens finale. De plus, nous avons parallélisé la portion de l'application qui nous intéressait sans avoir à modifier les centaines d'autres fichiers formant le logiciel.

2.3 Implémentation de la solution sur carte graphique

Comme mentionné dans la section précédente, nous avons développé deux solutions distinctes pour répondre à nos deux objectifs : accélération et exactitude. L'élaboration de ces deux solutions seront donc détaillées individuellement.

2.3.1 Algorithme PLUNC sur GPU

Cette implémentation vise à remplir notre premier objectif qui consiste à vérifier le potentiel d'accélération brut du GPU. Nous nous basons sur une implémentation existante de l'algorithme de calcul de dose, celle présente dans PLUNC, que nous portons pour une exécution sur le GPU. Ceci nous permettra aussi de tester la précision des calculs sur la carte graphique.

Mentionnons tout d'abord que seule l'étape de DOSE sera effectuée sur le GPU. La raison est que l'implémentation de la TERMA dans PLUNC est séquentielle. En effet, les longueurs radiologiques sont calculées de façon itérative en ajoutant, pour chaque point, la longueur radiologique du point au dessus. Il est donc impossible de la porter directement sur GPU puisque nous avons besoin d'une solution où tous les points peuvent être calculés en même temps.

L'implémentation de l'étape DOSE de PLUNC se base sur une approximation de l'algorithme de superposition/convolution : la « convolution de cône affaissé » (*collapsed cone convolution*) (Anhesjö, 1989). L'algorithme ne sera pas étudié au point de vue physique, le lecteur intéressé pourra se tourner vers la référence (Anhesjo, 1989). Nous allons par contre analyser les opérations du point de vue informatique.

2.3.1.1 Description des principales étapes

Premièrement, nous décrirons les principales structures de données utilisées lors du calcul. Quatre tableaux en une dimension ont été créés lors de la phase de TERMA. Ils ont une taille de map_dimx*map_dimy*map_dimz. Ceux-ci contiennent les informations de TERMA (terma_map), de longueur radiologique jusqu'à la source (rpl_map) de densité (density_map) et de contamination d'électrons (ecd_map). La FDD (*kernel*) est caractérisée par cinq tableaux : le tableau principal contenant les valeurs de la FDD (udata), un tableau contenant la valeur des angles zénithales (phi), un tableau contenant la valeur rayons (r), un tableau contenant la valeur des différences de rayons

(dr) et un tableau contenant la valeur de *kernel* à différentes profondeurs pour permettre l'interpolation (kerneln).

L'algorithme effectue une boucle triple sur tous les points formant la grille de calcul (en x, y et z). Pour chaque point sur cette grille de calcul, la fonction calculant la dose en un point est appelée. C'est cette particularité qui nous permet de paralléliser le calcul de la DOSE : chaque calcul de dose en un point est indépendant du calcul de dose pour tous les autres points.

La fonction de calcul de dose débute avec l'initialisation de quelques variables locales. Par la suite, la coordonnée dans la grille de calcul est transformée en coordonnée de la grille du patient. Ceci permet de vérifier si le point de dose qui sera calculé fait bien partie du patient. Si ce n'est pas le cas, l'algorithme cesse immédiatement et une dose de zéro est retournée. Si la dose à calculer est bien à l'intérieur du patient, la première interpolation à l'intérieur des données du patient a lieu. Les coordonnées dans la grille du patient sont arrondies à l'entier inférieur et une interpolation trilinéaire a lieu à l'aide des huit voxels entourant le point de calcul dans le tableau de contamination d'électrons. Par la suite, la convolution s'effectue à l'aide d'une boucle triple sur le nombre d'angles azimutaux, zénithaux et le nombre de rayons. À l'intérieur de cette convolution, trois autres interpolations comme celle décrite ci-haut ont lieu sur les données de densité, de TERMA et de longueur radiologique. Finalement, toujours à l'intérieur de la boucle triple, les données de FDD à différentes profondeurs sont interpolées pour donner la valeur de la FDD à la profondeur courante. Le résultat est retourné.

2.3.1.2 Port de l'algorithme

Le port de l'algorithme vers CUDA s'effectue en quatre étapes : les données sont copiées de l'hôte vers la carte graphique, la configuration d'exécution est déterminée, le calcul est effectué, les données sont copiées de la carte graphique vers l'hôte.

La première étape, dans notre cas, a été effectuée de deux façons différentes. La première façon, la plus simple, consiste à copier directement les données en mémoire globale sur la carte. Nous appellerons cette configuration « mémoire linéaire ». La deuxième copie les données sur la carte sous forme de texture, ce qui permet l'utilisation des engins de texture pour aller chercher les données en mémoire. Sous CUDA, lorsque les données sont stockées en mémoire principale et qu'elles ne sont pas sous la forme de texture, les lectures n'ont aucune cache locale. C'est-à-dire que, contrairement à un CPU qui, lorsque une requête en mémoire vive est faite, va stocker la donnée elle-même mais aussi ses voisines en mémoire cache, la carte graphique ne garde aucune cache des données voisines. La lecture en mémoire linéaire est plutôt orientée vers la lecture séquentielle en mémoire des différents *threads* d'un *warp*. En effet, une transaction en mémoire peut aller chercher 64 ou 128 octets en mémoire lors d'une seule transaction, à condition que les éléments composant les 64 ou 128 octets en mémoire soient contigus. C'est donc dire que lorsque le processus i accède à l'élément i d'un tableau, le processus i+1 à l'élément i+1, et ce jusqu'au processus 16; et que le tableau est composé de données de 32 bits, les 16 requêtes en mémoire peuvent être accomplis avec une seule lecture de 64 octets. Cette méthode de lecture en mémoire groupée (*coalescing* en termes CUDA) permet d'obtenir des gains d'approximativement un ordre de magnitude

(NVIDIA, 2008, p. 53). Or, dans cette application, les données ne sont pas lues en mémoire selon cette méthode mais plutôt d'une façon désordonnée. Lorsque les données sont stockées sous forme de texture et que les engins de texture sont utilisés pour lire en mémoire, une cache de localité est alors possible. Ceci a de l'importance puisque la plupart des lectures en mémoire, dans cette application, sont des interpolations sur les 8 voisins d'un point. Les lectures pourraient donc fortement profiter d'une cache de localité en mémoire. Nous avons implémenté ces deux techniques d'accès en mémoire pour étudier leur effet. La copie des données en mémoire linéaire s'effectue à l'aide d'une seule commande CUDA et ne sera donc pas expliquée plus en détails. La copie des données sous forme de texture s'effectue avec un ensemble de commandes documentées dans (NVIDIA, 2008, p. 37). Nous avons aussi converti le tableau linéaire en texture trois dimensions puisque ceci devrait favoriser la cache des données souhaitées pour une interpolation tridimensionnelle.

La deuxième étape concerne la configuration d'exécution. Comme mentionné à la section 1.3.3, le programmeur a le plein contrôle sur la configuration des tailles de blocs et de grilles, pour autant que celles-ci ne dépassent pas les limites de la carte. Le but de cette configuration est de définir comment les divers processus seront distribués à travers les multiprocesseurs de la carte. La taille d'un bloc décrit le nombre de processus légers à l'intérieur d'un bloc. Un bloc doit pouvoir s'exécuter sur un seul multiprocesseur et le nombre de registres sur un multiprocesseur est fixe, nous avons donc une limite sur la taille d'un bloc (autre que celle de 512 fixée par CUDA). La taille d'un bloc doit alors tenir compte de combien de registres a besoin un processus léger

exécutant le code souhaité. Sur la carte graphique ayant servi pour les tests (une 8800GT, le système de test sera défini dans le chapitre des résultats), nous disposons de 8192 registres par multiprocesseur. Notre implémentation a besoin de 54 registres par processus (donnée qui nous est fournie par le compilateur nvcc), nous trouvons donc que la taille maximale serait de 151 processus légers. Comme les multiprocesseurs contiennent 8 processeurs, nous trouvons le multiple de 8 le plus près, ce qui donne 128 processus par bloc. Par contre, comme nous n'avons pas besoin d'interaction entre les processus d'un bloc, nous n'avons aucune raison d'utiliser la taille de bloc maximale. Nous allons plutôt utiliser la taille de bloc minimale, soit 32 processus par bloc (la taille d'un *warp*). Ceci aura pour effet d'augmenter le nombre de blocs qui s'exécuteront sur chaque multiprocesseur. Avec plus de blocs à sa disposition, l'ordonnanceur pourra mieux cacher la latence d'accès à la mémoire globale. La taille de grille est beaucoup plus simple à définir. Cette taille définit le nombre de blocs qui devra être lancé pour exécuter le code demandé. Comme, dans le cas présent, nous voulons un processus par voxel, et que nous savons combien de processus par bloc nous avons, la taille de grille dépendra du nombre de voxels à traiter. Au moment de cette écriture, le nombre maximal de processus dans une dimension de grille (les grilles peuvent être en deux dimensions) est de 65536. Pour cette raison, nous divisons les blocs en grille en deux dimensions avec ce calcul simple :

$$GridDim_x = GridDim_y = \left\lceil \sqrt{\frac{N_{voxel}}{TailleBloc}} \right\rceil \quad (8)$$

où $GridDim_x$ et $GridDim_y$ représente les dimensions en x et y de la grille, N_{Voxel} le nombre de voxels à traiter et TailleBloc la taille d'un bloc de processus.

La troisième étape est celle où le programme est lancé et les calculs sont effectués. Pour cette première solution, nous avons tenté de minimiser les changements au code original de PLUNC de manière à isoler le potentiel d'accélération du GPU. La plus importante modification a été le passage au calcul en simple précision dans l'ensemble des calculs. Le calcul double précision a été introduit avec les cartes de la famille GT200 de NVIDIA mais les tests ont été effectués avec une carte d'une famille moins récente. Les conséquences de ce passage à la simple précision seront exposées dans le chapitre des résultats. Nous avons aussi déplacé quelques tableaux statiques qui auraient autrement été placés en mémoire locale (donc en mémoire globale) vers des registres. Au moment de cette écriture, cette taille est statique, donc définie à la compilation, et assez petite pour ne pas utiliser un trop grand nombre de registres ce qui ferait que le processus de compilation du *kernel* pourrait manquer de registres. Cette solution n'est par contre pas modifiable à l'exécution puisqu'il est impossible de définir un tableau de registres, les registres doivent donc être instanciés à l'aide de variables individuelles et indépendantes. Des données constantes à tous les voxels, par exemple le nombre de voxels à traiter, les dimensions du patient dans les trois dimensions, la taille des FDDs, etc. ont été placés en mémoire constante. Les lectures en mémoire constante peuvent bénéficier d'un mécanisme de diffusion générale (*broadcast*) lorsque tous les processus d'un *warp* accèdent à la même valeur en mémoire constante, ce qui sera le cas puisque les lectures en mémoire constante ne dépendent pas du processus qui fait la lecture et

que chaque processus devrait avoir besoin des informations en mémoire constante au même moment.

2.3.2 Nouvel algorithme

Notre deuxième objectif, après avoir démontré que le calcul de dose peut être grandement accéléré par l'utilisation d'une carte graphique et ce sans perte de précision due au matériel graphique (voir chapitre des résultats p.77), est d'élaborer une solution avec le matériel graphique en tête qui pourra offrir des gains en vitesse supérieurs lorsque comparée à une implémentation qui ne tient pas compte de l'architecture sous-jacente. Nous avons donc procédé à l'élaboration de notre propre solution en essayant d'éviter les goulots d'étranglement présents dans l'algorithme original de PLUNC tout en implémentant de façon plus rigoureuse certains aspects qui ont été approximés dans la solution de PLUNC par souci d'économie de temps de calcul. L'élaboration de notre solution est divisée en deux phases : TERMA et DOSE. Ces dernières seront étudiées individuellement.

2.3.2.1 TERMA

La phase de TERMA est celle qui a été la moins modifiée par rapport à l'algorithme déjà implémenté dans PLUNC puisque ses opérations sont assez rigides et fixes. La fluence de photon provenant de la tête de l'accélérateur doit être atténuée pour être établie en tout point dans le corps. Les équations 2, 3 et 4 seront donc utilisées pour trouver la fluence réelle en tout point et pour calculer la TERMA. Une courte section sur la conversion de données d'imagerie médicale en densité sera présentée et le calcul de la

TERMA suivra. Finalement, nous présenterons les différences, au point de vue modélisation, entre notre implémentation et celle de PLUNC.

2.3.2.1.1 Implémentation

La première étape à effectuer est de convertir les données qui ont été obtenues par imagerie médicale en données de densité. Nous avons développé un *kernel* qui effectue cette conversion. La table de conversion a été tirée de l'implémentation de PLUNC et est disponible à l'annexe 1. Ce *kernel* est donc très simple. Tout d'abord, les données qui ont été obtenues par imagerie médicale sont chargées sur la carte graphique. Un tableau pour les données en sortie est aussi donné en argument. Il est à noter que les données d'entrée sont des *shorts* alors que les données en sortie sont des *floats*. Ceci peut être un problème puisque les lectures de *shorts* ne seront pas groupées (*coalesced)* lorsque les règles de lectures pour l'architecture 1.0 et 1.1 sont applicables (NVIDIA, 2008, p. 54). En effet, chaque processus devrait accéder à des mots de 32 bits alors que la taille des *shorts* est de 16 bits. Nous estimons que le coût de ceci sera négligeable par rapport au temps d'exécution des autres *kernels*. Les écritures, quant à elles, seront groupées puisque des *floats* sont écrits. Le corps du *kernel* ne fait qu'assigner une valeur de sortie en fonction de la table de conversion et de la donnée en entrée. Ce *kernel* est donc lancé pour chacun des voxels formant la grille de TERMA. La taille de bloc pour ce *kernel* est de 128 processus par bloc. Le *kernel* n'utilise que 5 registres, ce qui veut dire que sur une architecture à 8192 registres par multiprocesseur, comme celle qui sera utilisée pour obtenir les résultats, 12 blocs pourraient être affectés, ce qui dépasse la capacité d'un multiprocesseur. Nous aurons donc un taux d'occupation de 1, ce qui est idéal. Le taux

d'occupation est une métrique développée par NVIDIA qui décrit le nombre de *warps* actifs sur un multiprocesseur par rapport au nombre maximal de *warps* gérables par un multiprocesseur (NVIDIA, 2008, p. 68).

Les données de densité étant maintenant disponibles, nous pouvons passer au calcul de la TERMA. Nous commencerons tout d'abord par une vue d'ensemble des opérations à effectuer pour ensuite les présenter avec plus de détails d'implémentation.

Comme on le remarque dans les équations 2, 3 et 4, nous avons besoin d'effectuer deux atténuations : l'une considérant l'inverse du carré de la distance géométrique parcourue et l'autre en exponentielle négative utilisant le tracé du rayon à travers le milieu et prenant en considération les densités rencontrées.

La première de ces atténuations est la plus simple à évaluer puisque nous connaissons la position de chaque voxel sur la grille de TERMA ainsi que la position de l'appareil d'irradiation par rapport à cette grille. Il suffit donc de prendre la longueur géométrique du vecteur joignant ces deux points.

La deuxième atténuation est par contre beaucoup plus compliquée et exigeante au point de vue calcul. Comme il a été mentionné à la section 1.2.2.2, l'algorithme de Siddon (Siddon, 1985) (Siddon, 1985) et amélioré par Christiaens et al. (Christiaens, Sutter, Bosschere, Campenhout, & Lemahieu, 1998) (Christiaens, Sutter, Bosschere, Campenhout, & Lemahieu, 1998), est utilisé pour faire le tracé radiologique dans notre grille régulière. Le calcul de la longueur radiologique est fait une fois pour chaque voxel. Nous considérons que, en ce qui concerne la longueur radiologique, le milieu

traversé possède une densité électronique équivalente à celle de l'eau. La densité électronique est l'un des facteurs qui influent sur la valeur du facteur d'atténuation d'un milieu. Le calcul de tracé de rayons prend donc en compte la densité réelle traversée mais le calcul de conversion de densité à facteur d'atténuation considère que la somme de densité traversée correspond à une densité équivalente en eau. Au point de vue de l'atténuation et de la densité, nous nous trouvons donc avec de l'eau plus ou moins dense. Si cette simplification n'était pas faite, il faudrait refaire le calcul de longueur radiologique pour tous les niveaux d'énergie du spectre, puisque la valeur du facteur d'atténuation dépend de l'énergie étudiée ainsi que de la densité du milieu. De plus, les tables d'atténuations sont construites pour des énergies précises et elles devraient être interpolées pour nos niveaux énergétiques (Hubbell & Seltzer, 1996) (Hubbell & Seltzer, 1996). Notons aussi que le spectre énergétique a été modifié, de façon indépendante pour chaque voxel, pour tenir compte de l'adoucissement de faisceau hors axe comme présenté à la section 1.2.2.3. Pour faire cet adoucissement hors axe, il est nécessaire, pour chaque point dans la grille, de trouver la fluence réelle de chaque plage d'énergie du spectre. Nous utilisons la technique présentée par Starkschall, (Starkschall, 2000) (Starkschall, 2000), pour effectuer cet adoucissement.

Avec la longueur géométrique et la longueur radiologique, le calcul de la TERMA peut être effectué. Une boucle sur tous les niveaux d'énergie du spectre est nécessaire. L'équation 4 se résume à la longueur radiologique multipliée par le facteur d'atténuation de l'eau pour le niveau d'énergie en cours. L'équation 3 applique le premier facteur d'atténuation mentionné ci-haut. L'équation 2 utilise les résultats de l'équation 3 et 4 et

donne la TERMA. Les opérations ont donc été vues dans leur ensemble, nous pouvons maintenant les présenter du point de vue de leur implémentation.

Nous aurons tout d'abord besoin d'un certain nombre de données spécifiques à la géométrie de l'ensemble appareil-patient ainsi qu'aux caractéristiques physiques du rayon irradiant. Les données caractéristiques de la grille (nombre de points, échelle pour chaque axe, etc.) sont stockées dans une structure. Une deuxième structure contient les informations concernant le spectre du rayon irradiant. Cette structure contient les différents niveaux d'énergie présents dans le spectre ainsi que leur importance respective normalisée et la valeur du facteur d'atténuation de l'eau pour ce niveau énergétique. Nous trouvons aussi la fluence Φ_0 pour chacun de ces niveaux. Notons que ces données sont reprises d'autres structures de données déjà présentes et utilisées dans PLUNC pour d'autres algorithmes de calcul de dose. Ces informations sont stockées en mémoire constante. Nous avons aussi quelques opérations supplémentaires pour établir le spectre hors axe. Les données de spectre non modifiées sont présentes en mémoire constante et nous avons donc besoin d'un endroit pour stocker les données de spectre modifié. Nous utilisons la mémoire partagée pour accomplir cette tâche mais nous l'utilisons comme banque de registres supplémentaires et non comme mécanisme pour que les processus légers puissent communiquer entre eux.

Nous avons aussi besoin des données de densité calculées lors de l'opération précédente. Ces données ont été générées sous forme de tableau en mémoire linéaire. Puisque les accès en mémoire se feront sans un ordre nous permettant de grouper les lectures en

mémoire, nous transférons ces données dans une texture 3D nous permettant de tirer avantage de la cache de localité des unités de texture. Ceci devrait nous aider puisque les lectures de densités se feront à l'intérieur du tracé de rayons, il y a donc une localité dans les lectures de densités.

Nous avons implémenté deux versions du calcul de la TERMA. La première fait intervenir un tracé de rayons complet alors que la deuxième utilise un tracé vertical approximé.

Tracé radiologique complet

Pour la première version faisant intervenir un tracé radiologique complet, nous avons besoin de calculer la longueur radiologique à l'aide de l'algorithme de Siddon et de boucler sur tous les niveaux énergétiques pour sommer la TERMA. L'algorithme de Siddon est tiré assez fidèlement de (Christiaens, Sutter, Bosschere, Campenhout, & Lemahieu, 1998). Notons quand même l'utilisation des divisions rapides `fdividef` qui s'exécutent en 20 cycles au lieu de 36. Nous utilisons aussi les fonctions intrinsèques `min(a,b)` et `max(a,b)` qui se compilent en une seule instruction lorsqu'exécutées sur la carte graphique. Nous utilisons aussi la fonction de saturation (`saturate(x)`) pour limiter les valeurs des facteurs α à [0,1]. Finalement, comme l'algorithme est présenté en deux dimensions, nous en avons fait l'expansion pour tenir compte des trois dimensions de notre grille de patient. Le reste du calcul de la TERMA est fidèle aux équations 2, 3 et 4. Les informations de TERMA sont écrites dans un tableau de *float* d'une façon groupée (l'explication des lectures groupées a été faite à la section 2.3.1.2.

La configuration d'exécution est ensuite définie. Comme il n'y a aucune coopération entre les différents processus, nous n'avons pas besoin d'une grande taille de bloc. L'impact des différentes tailles de bloc sera présenté dans le chapitre des résultats. Comme nous voulons un processus par voxel, nous n'avons aucune liberté quant à la taille de la grille une fois que la taille d'un bloc est définie.

Tracé radiologique vertical

L'implémentation du deuxième *kernel* s'éloigne un peu du modèle traditionnel de calcul de TERMA mais permet d'utiliser plus efficacement la mémoire disponible sur la carte graphique. Une approximation est faite sur le tracé de rayons : nous considérons que les tracés sont parallèles et verticaux. La Figure 2-1 illustre cette approximation.

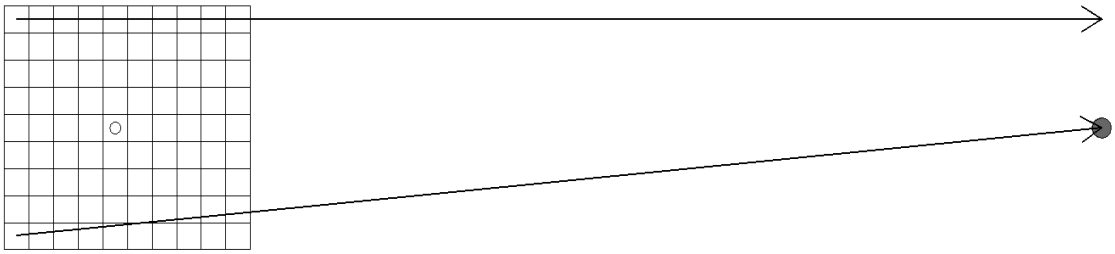

Figure 2-1 Tracé de rayon complet et approximation verticale

La flèche du dessous représente le tracé de rayons utilisé dans le premier *kernel.* Un tracé à partir de chaque voxel pointe sur la source de rayonnement à droite de l'image. La flèche du haut illustre l'approximation que nous ferons pour ce deuxième *kernel*. L'idée de cette approximation est que de cette manière, tous les tracés de rayons pour les voxels d'une même ligne verticale (horizontale dans notre figure) ont besoin des mêmes

données de densité. Nous pouvons donc les charger une seule fois en mémoire partagée et donc s'éviter $(N_{ligne}-1)*N_{ligne}$ lecture en mémoire, où N_{ligne} représente le nombre d'éléments sur une ligne, et ce pour chaque ligne. D'un point de vue global nous avons donc besoin de charger chaque donnée de densité une seule fois de la mémoire globale. Comme on le voit dans la Figure 2-1, qui est à l'échelle du point de vue des distances mais non quant au nombre de voxels, avec une distance source-isocentre de 100 cm, les propriétés géométriques font que relativement peu de voxels sont traversés latéralement même lorsque nous utilisons le voxel le plus loin possible de l'axe central. De plus, comme les densités du corps ne varient en général pas rapidement dans un même voisinage, cette approximation de tracé peut quand même donner de bons résultats puisque des densités semblables devraient être prises en compte. Des problèmes pourraient bien survenir lorsque, par exemple, un os aurait dû être traversé mais que le rayon, en demeurant vertical, reste au côté de l'os. C'est pour cette raison que nous garderons les deux *kernels* dans notre application. Nous devons aussi tenir compte de la plus grande distance géométrique qui est parcourue en suivant le tracé parfait par rapport au tracé en ligne vertical. Une étude trigonométrique nous permet d'obtenir la formulation d'un « facteur d'allongement » :

$$FA = \frac{(x^2+y^2+z^2)}{z} \qquad (9)$$

où x, y et z représente les distances par rapport à la position de la source. Ce facteur d'allongement multipliera la longueur radiologique trouvée.

La configuration d'exécution pour ce *kernel* est ensuite définie. Nous avons besoin de coopération entre tous les *threads* calculant la TERMA pour une ligne verticale. Nous avons donc besoin de choisir la taille d'un bloc en fonction de cette donnée. Nous n'avons par contre besoin d'aucune coopération entre les *threads* calculant la TERMA pour des lignes différentes. Nous choisissons donc un bloc en 3D : (1, 1, N_{ligne}). CUDA nous limite à 64 *threads* pour la coordonnée en z d'un bloc de *threads,* nous utiliserons donc la coordonnée en x, qui elle est limitée à 512. Ceci n'est que pour se plier aux exigences CUDA et n'a pas d'influence sur la logique du *kernel*. Ceci limite donc la dimension en z de notre grille de TERMA à 512 coupes. La taille de grille, comme nous utilisons des blocs couvrant une ligne en profondeur, sera équivalente au couple (longueur, largeur) de notre grille de TERMA. Le *kernel* utilisera aussi N_{ligne} * 4 octets/float octets de mémoire partagée. Comme chaque multiprocesseur possède 16 Ko de mémoire partagée et que N_{ligne} est rarement plus de 300, il ne devrait pas y avoir de problème quant à la taille de mémoire partagée nécessaire.

Optimisation des lectures et écritures

Le grand problème potentiel de ce *kernel* concerne en les lectures et écritures non groupées. En effet, comme nous calculons, à l'intérieur d'un bloc, les résultats pour une ligne en 'z', nous ferons des écritures à travers les 'z', qui ne sont pas contigües en mémoire. En effet, comme nous l'avons déjà mentionné, pour que les lectures soient groupées, les processus successifs doivent écrire des endroits successifs en mémoire. Or, les processus successifs, dans le cas présent, représentent des 'z' successifs alors que les

éléments successifs en mémoires se suivent en ‘x’. Nous avons donc développé un troisième *kernel* utilitaire qui fait une réorganisation d’index (*index resshuffling*) pour passer de matrices XYZ à des matrices ZYX. De cette façon, nous pourrons lire et écrire les éléments en mémoire de façon groupée puisque les ‘z’ sont maintenant contigus en mémoire. Ce *kernel* est donc utilisé deux fois. Une première fois avant le calcul de la TERMA pour réorganiser la matrice de densités de manière à ce que celle-ci puisse être lue de façon continue. Une deuxième fois après le calcul de la TERMA puisque les résultats auront été stockés sous la forme d’une matrice ZYX. Nous devons donc les réorganiser en matrice XYZ avant que PLUNC ne continue son traitement. La logique de ce *kernel* de réorganisation est basée sur un exemple de la suite de développement fournie par NVIDIA. La mémoire partagée est utilisée comme stockage temporaire pour permettre de lire une section carrée de la matrice originale, dans notre cas, un carré de 16x16 dans le plan XZ. Ce carré peut donc être lu de façon totalement groupée, ligne par ligne. Par la suite, ce tableau est lu dans le désordre à partir de la mémoire partagée pour être réécrit de façon groupée dans la matrice de sortie. Nous avons dû adapter l’exemple puisque celui-ci ne fonctionnait que sur des tableaux en deux dimensions. Nous avons donc ajouté une boucle à l’intérieur pour passer toutes les tranches en ‘y’ contenant les sous-matrices XZ à réorganiser. Une courte étude de l’impact de l’utilisation de cette technique plutôt que de faire des lectures et des écritures non groupées sera présentée dans le chapitre de Discussion.

De retour au programme appelant, les données calculées sont copiées dans la structure de données déjà présente dans PLUNC et qui sera accessible lors du calcul de la DOSE à l'étape suivante. Le calcul de la TERMA est alors terminé.

2.3.2.1.2 Différences notables entre notre implémentation et celle de PLUNC

Deux différences principales, au niveau de la modélisation et des calculs, nous différencient de l'implémentation de PLUNC.

Tout d'abord, pour le premier *kernel*, nous utilisons un tracé de rayons complet, de chaque voxel vers la source de radiation. L'implémentation de PLUNC utilise le tracé vertical présenté dans le deuxième *kernel*.

La deuxième différence est la manière de traiter le facteur hors axe pour l'adoucissement de rayon. Dans PLUNC, un angle est calculé à la surface de la grille de TERMA, et cette valeur d'angle hors axe est considéré valide pour tout le tracé radiologique. Or, ceci n'est pas le cas. Puisque le tracé radiologique est vertical, l'angle hors axe n'est pas valide pour toute la durée du tracé radiologique. Nous allons donc, pour chaque voxel, recalculer la valeur de l'angle hors axe et effectuer un adoucissement de spectre basé sur cette valeur.

2.3.2.2 DOSE

Une fois la TERMA calculée, la phase de DOSE peut débuter. Nous nous distinguons ici complètement du modèle défini dans PLUNC pour effectuer le calcul de DOSE. Comme pour la section précédente, nous présenterons les algorithmes d'une façon générale pour

ensuite passer au détail et finalement à une comparaison entre notre méthode et celle de PLUNC

2.3.2.2.1 Implémentation

La première étape consiste à lire le fichier caractéristique pour le spectre ainsi que les divers fichiers caractéristiques de la FDDs. Une fois ces informations lues, la méthode présentée à la section 1.2.2.3 concernant les FDDs polyénergétiques peut être appliquée sur les FDDs spécifiques à chaque plage d'énergie pour former une FDD moyenne qui sera utilisée lors de la convolution. Nous n'avons par contre pas utilisé les données de FDDs présentes dans PLUNC, pour une question de compatibilité entre les données physiques utilisées pour le calcul dans PLUNC et les données dont nous avons besoin pour notre implémentation. Nous avons plutôt opté pour une FDD générée analytiquement, dans MATLAB, prenant la forme d'une gaussienne 2D. Nous n'avons donc pas eu à générer une FDD polyénergétique puisque notre FDD gaussienne n'a pas été issue d'une simulation de Monte Carlo polyénergétique. Ceci est sans importance au point de vue performance puisque cette étape ne fait pas partie des temps d'exécution que nous allons calculer plus tard et que nous lirons la FDD préalablement générée analytiquement de la même façon que si on chargeait de véritables données de FDD.. Il faut par contre remarquer que les données que nous obtiendrons n'auront aucun sens physique absolu. Une FDD « réelle » peut certes s'apparenter à une gaussienne 2D mais nos valeurs de coefficients de FDD ne seront pas reliées à des données physiques en particulier. Il va sans dire que nous nous intéressons donc au temps d'exécution plutôt qu'à l'exactitude absolue des résultats. Nous serons par contre bien sûr en mesure

d'évaluer l'exactitude des résultats par rapport à notre FDD analytique. La Figure 2-2 présente la forme de notre FDD. Il faut porter attention aux axes, la FDD n'est pas aussi large que haute. Le code Matlab pour la générer est le suivant :

```
clear;
pts = -5:.01:5;
N = length(pts);
X = reshape(repmat(pts,1,N),N,N);
Y = reshape(repmat(pts,N,1),N,N);
Z = 0.001*exp((-X.^2)/1-(Y.^2)/0.25);
```

On remarque que σ_x et σ_y n'ont pas la même valeur. Ceci est basé sur le fait que le rayonnement est plus pénétrant mais les valeurs ne sont que qualitatives.

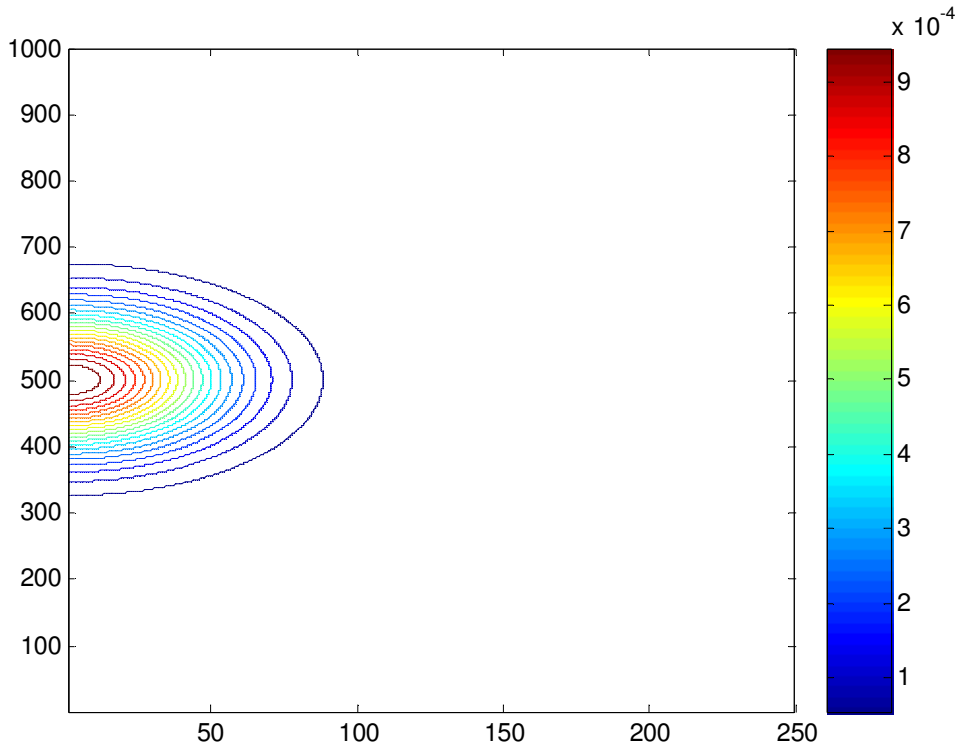

Figure 2-2 FDD analytique

Pour stocker les informations décrites précédemment, une nouvelle structure de données, KERNEL_CART, a été créée et ajoutée à la structure PDOSE qui renferme toutes les informations nécessaires au calcul de la dose. Cette structure KERNEL_CART contient les informations suivantes :

```
int          num_x,num_y;
float        scale_x,scale_y;
int          center_x,center_y;
float        *data_0;
float        *data_mid;
float        *data_1;
float        distance_0;
float        distance_1;
float        distance_mid;
float        lat_dist[NUM_DIST_LAT];
float        prof_dist[NUM_DIST_PROF];
```

Figure 2-3 Informations contenues dans la structure KERNEL_CART

Ceci définit donc une FDD cartésienne. C'est une matrice, en deux dimensions, qui renferme les coefficients de la FDD. Nous n'avons besoin que de deux dimensions puisque la FDD est invariante sous rotation. De plus, les informations latérales (sur l'axe des x) sont symétriques par rapport à l'axe des y, nous n'avons donc besoin que de stocker la FDD pour les valeurs de 'x' supérieures ou égales à zéro.

Les paramètres *num_x* et *num_y* contiennent le nombre d'éléments dans chaque dimension alors que *scale_x* et *scale_y* représentent la dimension de chaque pixel dans la FDD et *center_x* et *center_y* représentent le centre de la FDD selon chaque axe. La valeur de *center_y* sera utile à définir la grandeur étudiée de la diffusion arrière (des électrons qui « remontent » à partir du point d'interaction et à la direction du

rayonnement, illustré à la Figure 1-2) alors que nous nous attendons à ce que *center_x* reste zéro.

Suivant la méthode de Liu (Liu, Mackie, & McCullough, 1997) (Liu, Mackie, & McCullough, 1997), nous utiliserons trois FDDs pour tenir compte du durcissement de rayon. Les tableaux *data_0, data_mid* et *data_1* représente ces différentes FDDs. Ils auront chacun une taille de *num_x*num_x* éléments. Les champs *distance_0*, *distance_mid* et *distance_1* représente la profondeur de chacune des FDDs utilisées pour l'interpolation.

Finalement, les tableaux *lat_dist* et *prof_dist* contiennent les distances non uniformes qui seront utilisées lors de la convolution non uniforme, qui sera expliquée prochainement.

Convolution

Rappelons qu'à ce point dans les calculs, nous avons les informations de TERMA ainsi que les données de densité en tous points dans le patient. Ajoutées aux informations relatives de FDDs, ce sont là les informations nécessaires à la convolution. Comme pour la section de TERMA, nous commencerons par une vue d'ensemble des opérations à effectuer pour ensuite passer au détail de l'implémentation. Nous avons aussi implémenté deux approches complètement différentes pour effectuer les calculs de convolutions, nous les verrons donc chacune séparément. Ces approches sont définies par Hoban (Hoban, Murray, & Round, 1994).

D'un point de vue global, l'opération à effectuer est, pour chaque point dans la grille de dose, d'évaluer la contribution de chaque point de grille de TERMA (le point d'interaction) au point de dose en cours. De plus, comme mentionné dans les sections 1.2.2.1 et 1.2.2.3, nous tenons compte de différents facteurs influençant le calcul de la dose. Nous devons donc faire une réorientation de la FDD, comme présentée à la section 1.2.2.1, en tenant compte des vecteurs entre le point d'interaction et le point de dose et entre le point d'interaction et la source de rayonnement. Les calculs à effectuer ont été présentés à la section 1.2.2.1. Nous utiliserons aussi trois FDDs différentes pour permettre de tenir compte de l'adoucissement de rayon en profondeur. Nous aurons donc besoin de faire une interpolation entre deux de ces trois FDDs, en fonction de la profondeur à laquelle le point d'interaction se trouve dans le patient. Finalement, dans sa forme la plus précise, le calcul de convolution doit tenir compte de la longueur radiologique entre le point d'interaction et le point de dose pour aller chercher le bon coefficient dans la matrice de FDD. Nous aurons donc besoin de notre implémentation de l'algorithme de Siddon développée pour le calcul de la TERMA. À la fin du calcul de dose, une composante de contamination d'électrons est ajoutée suivant la méthode présentée en 1.2.2.3.

La première des deux approches que nous allons utiliser se nomme « point de vue de la dose ». Nous nous plaçons donc à un point donné dans la grille de dose duquel nous accumulons la contribution des points de TERMA environnant, en prenant en compte le facteur de la FDD correspondant à chaque emplacement du point de dose par rapport au point d'interaction, pour définir la dose totale.

La deuxième approche se nomme « point de vue de la TERMA » et est l'inverse de l'approche précédente. Nous nous plaçons à un point donné sur la grille de TERMA et évaluons la contribution de ce point de TERMA à chacun des points constituant la grille de dose.

Nous sommes maintenant prêts à passer au détail des implémentations. Nous avons développé les deux approches mentionnées ainsi que quelques autres variantes des calculs. Nous présentons ici les deux approches de façon séparée et pour chaque approche nous présenterons rapidement les variantes développées.

Quelques points sont valides pour les deux approches. Tout d'abord, de façon analogue au calcul de la TERMA, nous avons besoin de quelques informations quant à la géométrie de notre problème, notamment les points de départ en coordonnées faisceau, la taille de chaque voxel ainsi que le nombre de voxels par dimensions. Ces données (ainsi que d'autres qui ne seront pas mentionnées) sont stockées dans la mémoire constante. Des informations concernant les FDDs sont aussi mises en mémoire constante. Tout ce qui est présenté à la Figure 2-3, excepté les coefficients de la FDDs, sont mis en mémoire constante. Finalement, une structure analogue à celle utilisée dans PLUNC pour le calcul de contamination d'électrons est placée en mémoire constante. Une texture 3D contenant les densités des différents voxels est aussi utilisée dans les deux approches lorsque vient le temps de faire le tracé radiologique entre deux voxels. Finalement, le corps des deux approches est semblable. Une des étapes au calcul consiste à trouver la position du point d'interaction (ou point de TERMA) et le point de

dose. Une fois ces deux points trouvés, les calculs sont identiques : trouver la position dans la FDD et mettre la TERMA à l'échelle à l'aide des bonnes valeurs de FDD. Les deux approches diffèrent donc dans leur manière de trouver les points d'interaction et de dose.

Approche par point de vue de dose

L'approche par point de vue de dose comporte à son tour quelques variations. Des versions avec distance d'intégration uniforme, non uniforme et une intégration dans un espace sphérique ont été développées. Plus de détails seront donnés au cours des prochains paragraphes. Pour l'approche du point de vue de la dose, chaque processus léger représente un point de dose. Les points de dose constituent un des deux tableaux passés en paramètres à la fonction, le deuxième contenant les résultats du calcul. Les points sont lus dans l'ordre, ces lectures seront donc groupées; de même pour l'écriture des résultats. Le reste des données utiles utilisent les divers niveaux mémoire de l'architecture. Les données de TERMA, comme elles seront accédées dans le désordre, sont placées dans une texture trois dimensions, de même pour les données de densités (comme il a déjà été mentionné). Les données de FDDs sont aussi accédées dans le désordre et sont placées dans une texture deux dimensions. Nous avons expérimenté, dans un prototype très peu complet, avec des FDDs chargées en mémoire partagée lors de l'exécution du *kernel* et donc accessible à tous les processus d'un bloc, mais les résultats n'étaient pas intéressants et nous nous trouvions limités quant à la taille maximale des FDDs puisque la mémoire partagée est de petite taille. Elles auraient aussi

pu être mises en mémoire constante mais comme chaque processus accédera à des éléments différents des FDDs lors d'un tour de boucle donné, nous ne pouvons pas prendre avantage du mécanisme de diffusion globale (*broadcast*) de la mémoire constante et nous nous trouvions encore une fois limités quant à la taille maximale des FDDs. Nous avons finalement conservé les FDDs sous forme de trois textures (une par FDD d'interpolation) en deux dimensions.

Opérations à effectuer

Nous commencerons par définir la variante uniforme de notre algorithme et continuerons avec les autres variantes et leurs différences par rapport à la variante uniforme. Nous définissons une triple boucle qui constitue le cœur de la convolution. Cette triple boucle est centrée sur le point de dose en cours et s'étendra latéralement (en 'x' et 'y') de façon symétrique à l'aide d'un paramètre défini à la compilation. La boucle en 'z' ne sera quant à elle pas symétrique puisque les FDDs sont plus étendues vers le bas, suivant le sens du rayonnement incident. L'étendue de la boucle est définie par deux paramètres, l'un définissant le nombre de voxels au dessus du point de dose où nous allons chercher les contributions en TERMA et l'autre définissant le nombre en dessous. Notons ici que puisque nous nous situons au point de dose, la boucle en 'z' doit aller chercher plus de TERMA au dessus du point de dose qu'en dessous. Pour la version uniforme, les limites de boucles indiquent combien de voxels dans chaque direction seront pris en compte et non une longueur en cm. Il faut donc tenir compte de la taille des voxels pour bien fixer le nombre de voxels à prendre en compte pour obtenir des

résultats valides avec des granularités de grille variables tout en conservant une distance d'interpolation égale. Par la suite, deux vecteurs sont créés pour réaliser la réorientation de la FDD. Ces vecteurs permettent de trouver l'angle φ_t présenté à la Figure 1-3 à l'aide de la règle du produit scalaire. La longueur entre le point d'interaction et le point de dose doit aussi être trouvée. Deux options sont possibles : le tracé radiologique complet donnant la véritable longueur radiologique tenant compte des densités du corps, ou une longueur géométrique donnant la longueur radiologique en assumant que le corps est formé d'eau. La première option est extrêmement plus exigeante en temps de calcul, comme il sera présenté au chapitre des résultats. L'angle φ_t et la longueur entre les deux points permettent de définir les coordonnées cartésiennes qui sont nécessaires pour aller chercher le bon élément dans la matrice de FDD. Une étude trigonométrique de la Figure 1-3 nous permet de trouver les relations suivantes,

$$i_{fdd} = \left|\frac{dist*sin(\varphi_t)}{echelle_{fdd_x}}\right| \tag{13}$$

$$j_{fdd} = \frac{dist*cos(\varphi_t)}{echelle_{fdd_y}} \tag{14}$$

où :

i_{fdd} et j_{fdd} représentent les coordonnées cartésiennes dans les FDD
dist représente la longueur (radiologique ou géométrique) entre les deux points
$echelle_{fddx}$ et $echelle_{fddy}$ représentent la taille de chaque pixel de la matrice de FDD.

Si ces coordonnées font partie de la FDD, c'est-à-dire qu'on ne se trouve pas plus loin que le rayon d'action de la FDD, nous trouvons entre quelles FDDs d'interpolation nous

sommes en comparant la valeur en ‘z’ du point d’interaction aux limites de chaque FDD d’interpolation. Une interpolation linéaire entre les deux FDDs d’interpolations trouvées est alors effectuée pour trouver la valeur du coefficient final de dispersion. Ce coefficient est multiplié à la valeur de la TERMA au point d’interaction en cours et normalisé par la taille du voxel pour être accumulé dans la dose partielle. À la fin de cette triple boucle, l’accumulateur contient la dose totale.

La variante non uniforme de cette approche fait intervenir des distances d’intégration non uniformes. C'est-à-dire que, contrairement à l’approche uniforme où tous les voxels voisins du point de dose étaient considérés, nous allons plutôt en considérer de moins en moins à mesure qu’on s’éloigne du point de dose. Puisque la contribution des points éloignés est de plus en plus faible, d’une façon exponentielle, nous ne perdrons pas trop de précision mais gagnerons beaucoup en temps de calcul et en distance totale couverte par la convolution. La Figure 2-4 et la Figure 2-5 présente cette géométrie non uniforme.

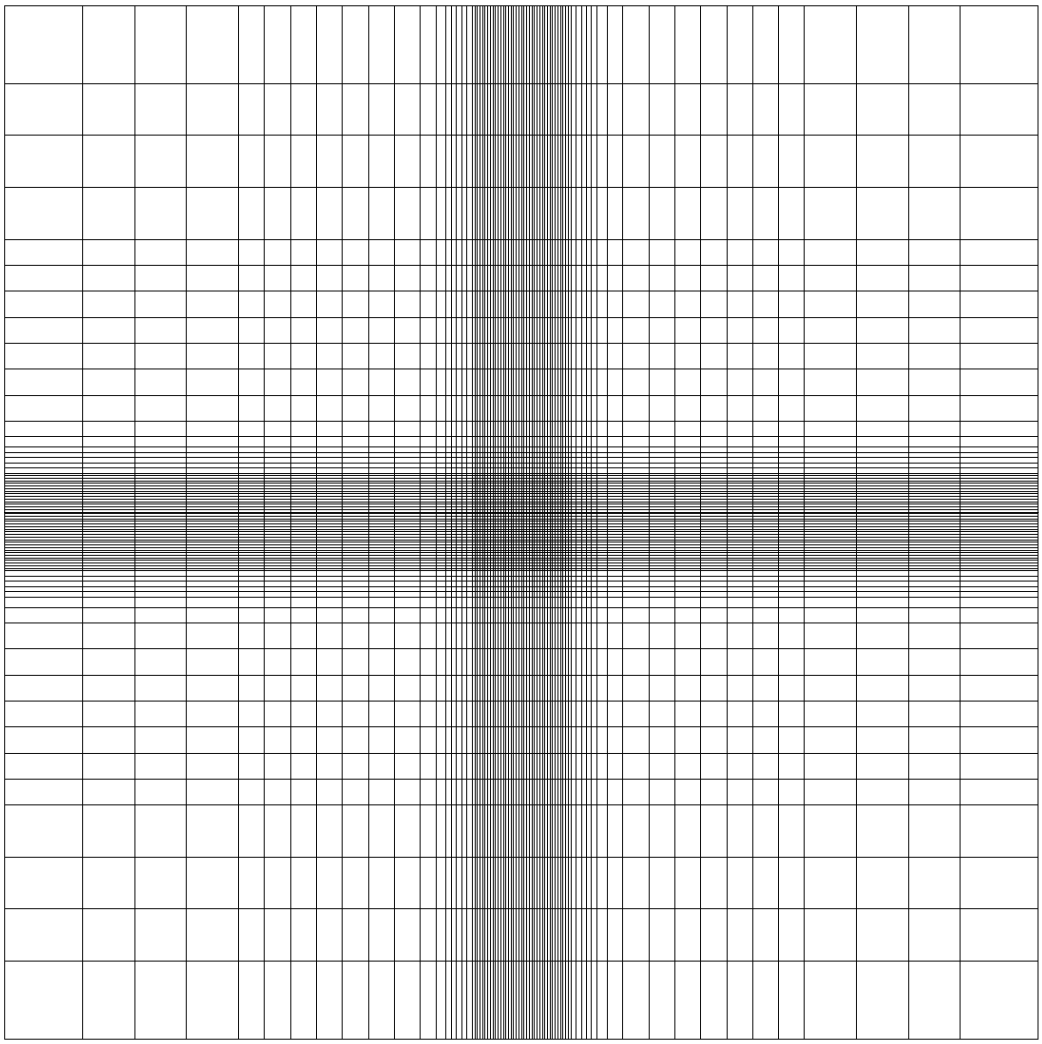

Figure 2-4 Géométrie non uniforme cartésienne de voxels

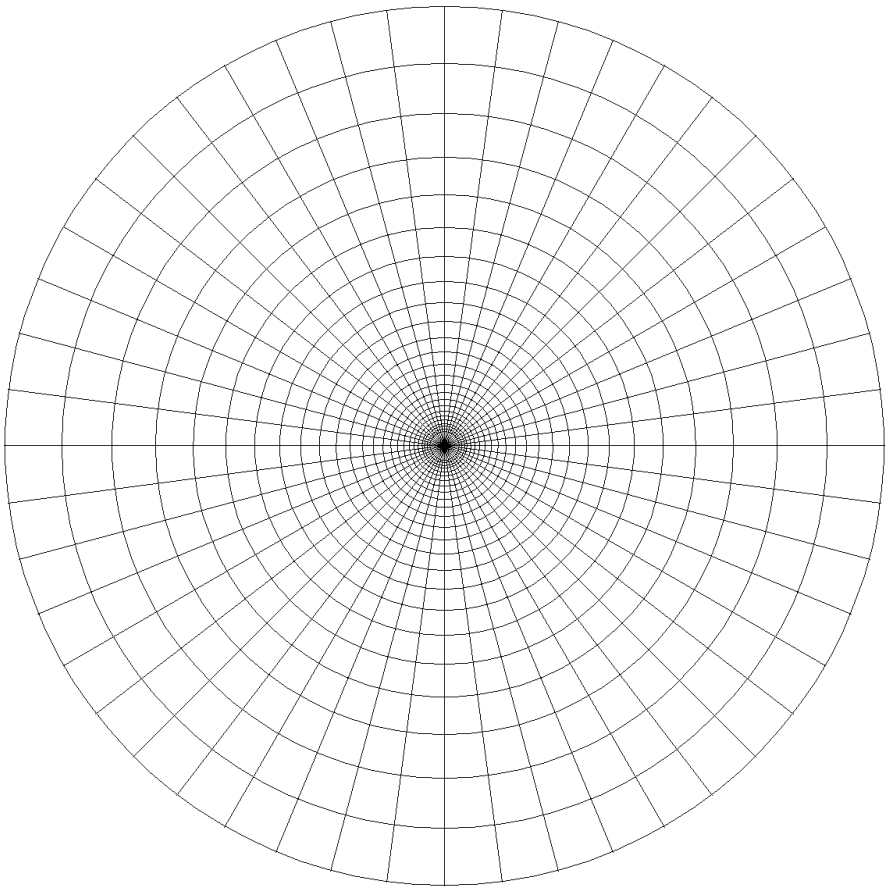

Figure 2-5 Géométrie non uniforme sphérique de voxels

Une autre façon de voir cette solution est que plus le voxel est éloigné, plus il est gros. On voit par contre qu'il y a un peu de travail en trop d'effectué le long des axes où il y a quand même une grande densité de voxels et ce même si on est éloigné du centre d'interaction. La triple boucle s'effectuera donc sur un certain nombre de distances, contenues en mémoire constante sous forme de vecteur. Ces distances seront ajoutées à

la position du point de dose pour trouver la position du point d'interaction à considérer. Les vecteurs de distances latérales et en profondeur ont été placés en annexe.

La variance sphérique, quant à elle, fait intervenir une intégration des différentes contributions de données de TERMA dans un espace sphérique. Notre triple boucle d'intégration se fait donc sur θ ,φ et r. La valeur de θ varie entre [0,2π[, la valeur de φ entre [0,π[et r varie de façon non uniforme entre [0,60[. La variation non uniforme se fait de façon procédurale avec la formule simple suivante :

$$r^k = r^{k-1} * g \qquad (15)$$

où g représente un facteur d'accroissement strictement plus grand que 1. Une fois un triplet (θ,φ, r) trouvé, nous le transformons en coordonnées cartésiennes à l'aide des formules d'usages et le traitement peut alors continuer de la même façon que pour les variantes précédentes. Cette variante nous offre la possibilité d'avoir un tracé radiologique « gratuit » étant donné que les densités rencontrées le long d'un couple (θ,φ) peuvent être sommées à mesure que 'r' augmente, évitant de refaire un tracé complet à chaque itération. Les calculs font par contre intervenir beaucoup d'opérations trigonométriques coûteuses pour passer d'un système de coordonnées à l'autre.

Configuration d'exécution

Pour ce qui est de la configuration d'exécution pour ce *kernel*, comme nous n'utilisons pas de mémoire partagée, ce qui est d'ailleurs la faiblesse de ce *kernel*, peu de choix s'offrent à nous. Nous utilisons un processus léger par point de dose. Ces processus

légers sont regroupés en blocs de 64. La taille de la grille se trouve à l'aide de l'équation 8. Comme mentionné précédemment, nous n'avons pas trouvé de bonne utilisation pour la mémoire partagée. Une dernière avenue qui a été explorée était de stocker les données de TERMA près des points de dose en mémoire partagée puisque ces données seront vraisemblablement utilisées dans tous les calculs de longueurs radiologiques partant de ce point de dose. Or, comme il n'y a que 16 384 octets de mémoire partagée, et que les données de TERMA sont des données en virgule flottante de 4 octets, nous trouvons une capacité de 4 096 éléments. Chaque bloc ayant 64 processus, cela fait donc 64 points de dose pour lesquels il faut garder les éléments à proximité en mémoire, donc une capacité de 64 éléments par point. Ceci constituerait un bloc de 4x4x4 éléments autour d'un point de dose et nous limiterait à un seul bloc par multiprocesseur, tout en introduisant du contrôle de flot supplémentaire. Nous avons donc laissé tomber toutes les alternatives présentées utilisant la mémoire partagée.

L'approche par point de vue de dose offre un avantage majeur : elle permet de facilement limiter l'étendue de la collecte de TERMA a l'aide des limites de boucles. Elle devrait aussi être avantageuse lorsqu'une grille de dose avec peu de points doit être calculée puisque toutes les données de TERMA ne seront pas nécessairement lues, seulement celles qui sont nécessaires au calcul des quelques points. Cette approche sera aussi sensible à la granularité de grille sous deux angles. Premièrement, une granularité plus fine amène un plus grand nombre de voxels traversés pour parcourir une même distance géométrique, ce qui influera sur le temps d'exécution du tracé radiologique. De plus, lorsque les distances uniformes sont utilisées, pour couvrir une même distance

géométrique à l'intérieur de la triple boucle, les limites des boucles devront prendre en compte la taille (réduite) d'un voxel et boucler sur plus d'éléments.

Approche par point de vue de TERMA

L'approche par point de vue de la TERMA est différente. Ici, chaque processus léger représente un point sur la grille de TERMA. Les données de TERMA, comme elles seront lues dans l'ordre, sont passées en paramètre sous forme de tableau, ce qui permettra la lecture groupée. De plus, par un mécanisme qui sera expliqué sous peu, chaque point dans la grille de dose pourra aussi profiter d'une lecture groupée. Par contre, comme les lectures groupées ne sont possibles que sur des éléments ayant une taille de 32 64 ou 128 bits et qu'un point (de type *float3*) est composé de trois éléments à virgule flottante, pour une taille de 96bits, les lectures ne seront pas groupées si nous laissons les données d'entrée sous cette forme. Nous passons donc trois tableaux de *float* : *pointx, pointy* et *pointz*; qui eux peuvent être lus de façon groupée, pour lire les points dans la grille de dose. Les données de densités seront encore lues de façon non ordonnée et résident dans une texture trois dimensions, de même pour les données de FDDs dans des textures à deux dimensions.

D'un point de vue global, les mêmes opérations sont à effectuer sur les données, mais avec un point de vue inversé. Chaque point de TERMA doit aller porter sa contribution à tous les points de dose de la grille. Il faut donc boucler sur tous les points de dose et refaire les mêmes calculs qu'exposés précédemment, c'est-à-dire : calculer la longueur radiologique ou géométrique, trouver les coordonnées cartésiennes dans la FDDs et

interpoler les bonnes FDDs d'interpolation. L'accumulation de dose se fait ici par la contribution de chaque point de TERMA à tous les points de dose, l'accumulation n'est donc pas faite par un seul processus léger mais nous avons plutôt tous les processus légers qui participent à la dose d'un point. C'est une distinction qui a des répercussions importantes et elles seront exposées dans le paragraphe qui suit.

Si les calculs se ressemblent d'un point de vue global, l'implémentation est différente. Nous nous trouvons avec un problème de « toutes-paires », c'est-à-dire que chaque point de TERMA a besoin d'avoir accès à chaque point de dose. Ceci pourrait donc causer problème au niveau du nombre d'accès mémoire qui doivent être effectués. Les gens de NVIDIA (Nyland, Harris, & Prins) nous offrent une technique intéressante pour résoudre ce problème en utilisant la mémoire partagée. L'algorithme est :

```
Nombre_Passe = Nombre_Point_Dose/Taille_Memoire_Partagée
De i = 0 à Nombre_Passe
      MemoirePartagée[idProcessus]                                    =
MemoireGlobale[idProcessus+i*dimBlock]
      Synchronisation de mémoire partagée
      De j=0 à dimBlock
            Calcul(maTerma,MemoirePartagée[j]
```

où dimBlock représente la taille d'un bloc de processus, idProcessus est un identificateur à chaque processus à l'intérieur d'un bloc.

De cette façon, chaque point de dose n'est lu qu'une fois par un bloc de processus, ce qui diminue grandement le nombre d'accès à la mémoire globale à effectuer. Nous utilisons un mécanisme similaire pour l'écriture des résultats : nous réservons une plage de mémoire partagée pour stocker les résultats intermédiaires des contributions des points de TERMA du bloc de processus. C'est donc dire que chaque élément de résultat en mémoire partagée sera incrémenté dimBlock fois avant d'être écrit en mémoire

globale. Cette écriture en mémoire globale est un autre aspect d'implémentation particulier. CUDA n'offre pas d'opération atomique sur les *floats* excepté une fonction d'échange d'une variable locale appartenant à un processus léger avec une variable en mémoire globale. Nous avons besoin d'une opération d'addition atomique puisque deux processus légers pourraient potentiellement vouloir écrire la contribution partielle d'un même point de dose au même moment, ce qui nous mène à une course aux données. Nous n'avons pas accès à une addition atomique pour pouvoir accumuler la valeur des doses partielles générées par chaque bloc de processus. Or, il est absolument nécessaire d'utiliser un mécanisme garantissant que des résultats partiels ne seront pas perdus dans une course aux données (*race condition*). La réponse à ce problème a été trouvée sur le forum de NVIDIA[6]. La solution fait usage de la fonction atomique *exchange* pour effectuer une addition atomique sur des données en virgules flottantes. Le seul problème de cette méthode est qu'aucun temps limite pour l'exécution de l'écriture n'est garanti. Le reste des opérations est sensiblement le même que pour la première approche par point de vue de dose.

Configuration d'exécution

La configuration d'exécution est ici d'une importance capitale. En effet, de par notre approche au problème des « toutes-paires », utilisant la mémoire partagée comme cache de points de dose, la taille des blocs influence directement la quantité de mémoire partagée utilisée. De plus, comme chaque bloc doit lire tous les points sur la grille de

[6] http://forums.nvidia.com/index.php?showtopic=79464&start=0&p=451309&#entry451309

dose une fois, le nombre de blocs à lancer dicte le nombre de transferts répétés d'un même point de dose de la mémoire globale à la mémoire partagée. Le nombre de processus légers total à lancer est équivalent au nombre de points dans la grille de TERMA, ce nombre est donc fixé par des paramètres hors du contrôle direct de notre algorithme. Une étude rapide à l'aide du profileur de NVIDIA nous a donné une taille de bloc de 64 processus légers.

La différence majeure entre cette approche et celle par point de vue de dose est que nous ne pouvons limiter la portée de notre accumulation de façon directe, comme à l'aide des variables de boucles de la première approche. Ce que nous pouvons faire, pour limiter le nombre de distances radiologiques à calculer, est de limiter le corps de l'algorithme à une certaine distance géométrique maximale. C'est-à-dire que le calcul d'accumulation, à partir du calcul de la distance radiologique jusqu'à l'interpolation de FDDs, ne se fait que si la distance géométrique entre point d'interaction et point de dose est plus petite qu'un certain seuil. Cette approche est donc beaucoup moins efficace pour couper des calculs inutiles que l'approche par point de vue de dose puisque les points doivent quand même être chargés en mémoire. Cette approche sera avantageuse lorsqu'une précision « totale » est souhaitée, c'est-à-dire lorsque nous faisons vraiment face à un problème « toutes-paires », lorsque tout les points de TERMA sont considérés pour chaque point de dose.

2.3.2.2.2 Différences notables entre notre implémentation et celle de PLUNC

Notre implémentation est en fait complètement différente de celle de PLUNC. Ce dernier utilise l'algorithme de convolution de cônes (*collapsed cone convolution)* (Anhesjö, 1989) (Anhesjö, 1989). Nous utilisons plutôt l'algorithme classique de convolution sans cette approximation.

Nous faisons aussi, dans certaines variantes, un tracé complet entre les voxels convolués alors que PLUNC somme les densités rencontrées le long d'un des rayons lancés à partir du site d'interaction. C'est cette différence qui est d'ailleurs la faiblesse majeure de notre approche puisque l'utilisation d'une grille cartésienne ne nous permet pas de faire une approche itérative au calcul radiologique puisque la distance parcourue dans chaque voxel variera étant donné que l'angle que fait le vecteur entre le site d'interaction et le site de déposition de dose n'est pas fixe. C'est pourquoi nous avons décidé d'implémenter une version de calcul par point de vue de dose avec variante sphérique. Cette approche nous permet d'obtenir un tracé radiologique « gratuit » puisque les densités rencontrées sont sommées, éliminant le besoin de faire un tracé radiologique du point d'interaction jusqu'au point de dose.

De plus, comme nous l'avons mentionné en début de sous-section, nous utilisons une FDD analytique gaussienne qui n'a rien à voir avec la FDD que PLUNC utilise pour faire ses calculs. Il ne peut donc y avoir aucune comparaison de données entre ce que nous allons obtenir et ce que PLUNC donne.

CHAPITRE 3 : RÉSULTATS ET ANALYSE

Le chapitre présent fera état de nos résultats sous deux angles. Comme nous avons étudié deux solutions dans le chapitre précédent, nous détaillerons ici les résultats spécifiques aux deux solutions. La première solution est le port d'un algorithme déjà établi et faisant partie de notre plateforme de planification de traitement, PLUNC. Nous nous attarderons aux facteurs d'accélération que nous avons obtenus pour cette solution ainsi qu'aux erreurs introduites par l'utilisation d'une carte graphique pour effectuer les calculs. La deuxième solution est celle que nous avons construite à partir des modèles que nous avons décrits au premier chapitre et qui s'insère dans PLUNC, sans faire l'utilisation du code de calcul de dose qui y est déjà présent. Nous ne pouvons pas présenter des résultats en termes de validité puisque la méthode n'a pas été évaluée de façon formelle par des gens ayant les compétences pour le faire, notamment des physiciens médicaux, mais nous exposerons les facteurs d'accélérations obtenus par rapport à la solution originale de PLUNC.

Notons que les résultats qui seront présentés ci-dessous, sauf indication contraire, ont tous été recueillis sur des tous été recueillis sur des ordinateurs ayant la configuration présenté au Tableau 3-1 et

Tableau 3-2 :

Tableau 3-1 Configuration de l'ordinateur PC1

Intel Core2Duo E6550 à 2.33GHz NVIDIA GeForce 8800GT 2Go de mémoire vive

Tableau 3-2 Configuration de l'ordinateur PC2

Xeon Core2 QuadCore à une fréquence de 2.40GHz NVIDIA GTX280	NVIDIA 8800GTX 2Go de mémoire vive

De plus, tous les temps de calcul qui seront rapportés englobent l'ensemble des opérations reliées aux calculs. C'est-à-dire que le compteur est initialisé lorsque le processus léger (processus CPU) est démarré pour une phase du calcul et qu'il est arrêté avant que le processus ne se termine. Pour les opérations GPU, ceci signifie que tous les transferts vers et à partir de la mémoire de la carte graphique sont inclus. Les résultats en temps représentent la moyenne de trois essais. Mentionnons finalement que la version de PLUNC que nous utilisons ne supporte pas une exécution parallèle sur CPU. C'est donc dire qu'un seul des deux cœurs de notre processeur de test sera utilisé.

3.1 Résultats du port de l'implémentation de PLUNC

Cette première section présente notre port de l'algorithme de PLUNC vers une exécution sur matériel graphique. Comme nous l'avons mentionné à la section 2.3.1, les résultats de cette section ne porteront que sur l'étape de DOSE puisque c'est la seule qui a été portée vers CUDA. Nous avons premièrement fait varier la taille de la FDD qui est utilisée lors de la convolution. Ceci a pour effet d'augmenter la taille de la triple boucle de convolution à l'intérieur du calcul de dose pour un point. Le nombre de points de dose à calculer a été fixé à environ 2000 pour être certain de bien saturer tous les

multiprocesseurs de la carte. La carte du PC1 comportant 14 multiprocesseurs, pour que chacun exécute au moins un bloc de processus, et puisque ces blocs comportent 32 processus, nous avons besoin d'au moins 448 points de calcul. Il y a donc amplement de points pour utiliser toutes les ressources de la carte. Nous présentons aussi ici l'impact de l'utilisation des unités de texture sur le temps global de traitement.

Le Tableau 3-3 présente les résultats de temps d'exécution et d'accélération et la Figure 3-1 présente les temps d'exécution en fonction de la taille de la FDD. Les tailles de FDD sont basées sur celles présentes par défaut dans PLUNC et multipliées par deux et quatre. La taille de base a été choisie pour refléter les travaux de Ahnesjö dans (Anhesjö, 1989) (Anhesjö, 1989). Nous avons par la suite multiplié ces valeurs par deux et quatre pour étudier l'effet de la taille de la fonction de dispersement.

Tableau 3-3 Temps d'exécution en fonction de la taille de FDD sur PC1

Taille de FDD (éléments)	T_{CPU} (s)	T_{GPU} (s)	T_{GPUtex} (s)	$A_{GPU/CPU}$	$A_{GPUtex/CPU}$
8x24x48	3.547	0.25	0.25	14.19	14.19
8x44x96	9.579	0.635	0.64	15.09	14.97
8x88x192	18.25	1.266	1.14	14.42	16.01

où T_{GPU} T_{GPUtex} et T_{CPU} réfère au temps d'exécution pour le GPU (sans et avec texture) et le CPU, respectivement. $A_{GPU/CPU}$ représente le facteur d'accélération entre GPU et CPU et est calculé ainsi :

$$A_{GPU/CPU} = \frac{T_{CPU}}{T_{GPU}} \qquad (16)$$

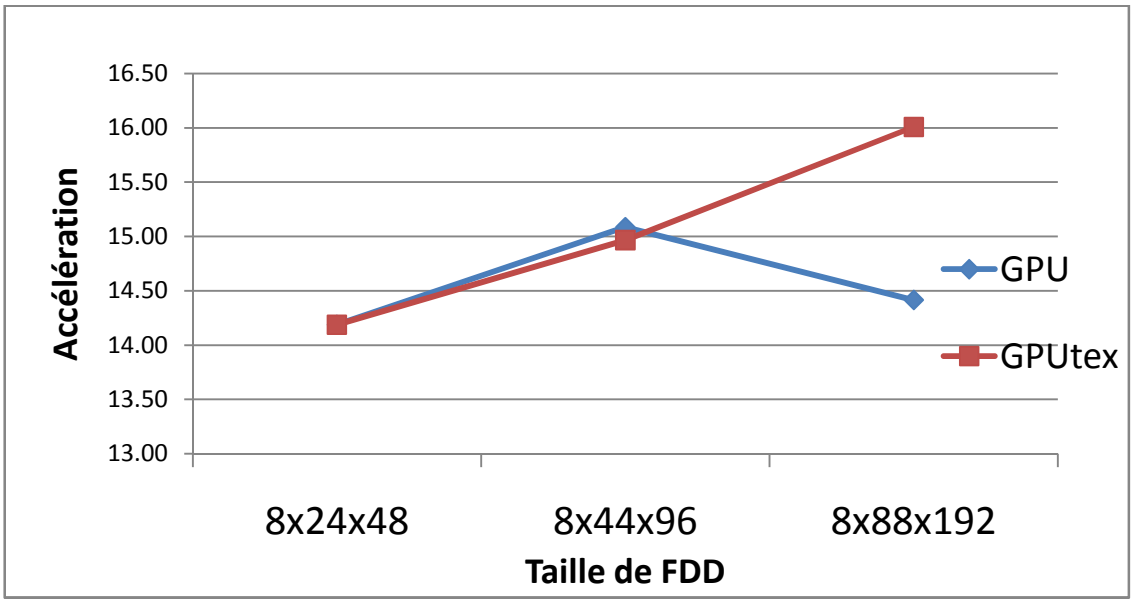

Figure 3-1 Accélération pour taille de FDD variable

Nous avons par la suite fait varier la granularité de la grille. Nous entendons par granularité la distance continue entre deux points sur la grille discrète. Ceci fait indirectement varier le nombre de points de dose à calculer puisque les dimensions de la grille restent les mêmes, seule la granularité change. Le Tableau 3-4 et la Figure 3-2 présentent ces résultats.

Tableau 3-4 Temps d'exécution pour granularité de grille variable (PC1)

Granularité (cm)	T_{CPU} (s)	T_{GPU} (s)	T_{GPUtex} (s)	$A_{GPU/CPU}$	$A_{GPUtex/CPU}$
1.00	9.58	0.74	0.63	13.03	15.33
0.75	15.45	1.03	0.88	14.99	17.66
0.50	34.78	2.14	1.77	16.25	19.70
0.25	138.69	8.64	6.84	16.05	20.27

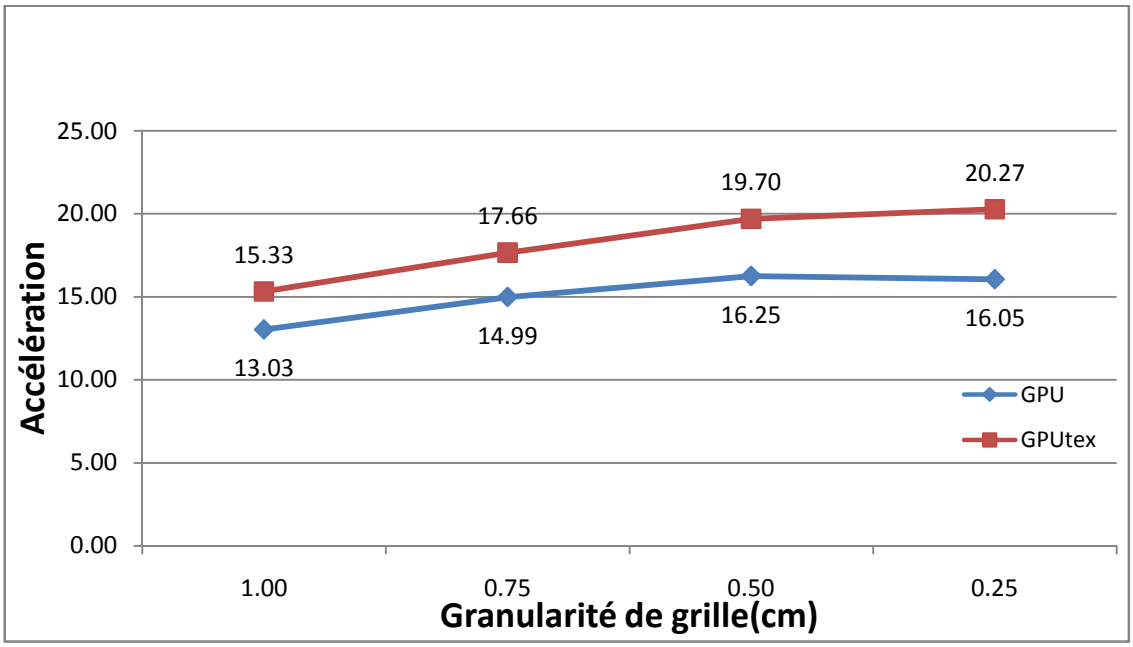

Figure 3-2 Accélération pour granularité variable

Finalement, nous avons calculé l'erreur entre l'implémentation originale pour CPU et l'implémentation sur GPU. Pour ce faire, nous avons utilisé trois versions du calcul de dose. Premièrement, la version originale CPU avec double précision dans les calculs, ensuite, une version en simple précision de l'algorithme sur CPU et finalement notre version en simple précision sur GPU. Nous présentons au Tableau 3-5 les résultats, en termes d'écart type, d'erreur relative moyenne et d'erreur relative maximale. La première ligne de ce tableau représente l'implémentation contre laquelle nous nous comparons, il n'y a donc pas d'erreur associée, la deuxième ligne représente la version roulant sur CPU n'utilisant que la simple précision et la troisième ligne représente la version s'exécutant sur GPU. La Figure 3-3 présente un histogramme de l'erreur entre

l'implémentation CPU à double précision et GPU à simple précision. Nous avons utilisé une granularité de 0.5 cm, pour un total d'environ 8000 points de calcul.

Tableau 3-5 Propriétés de l'erreur relative

	Écart type	Erreur relative moyenne	Erreur relative maximale
		(%)	(%)
CPU double précision	-	-	-
CPU simple précision	0.000010	0.000003	0.000300
GPU simple précision	0.000600	0.000050	0.038600

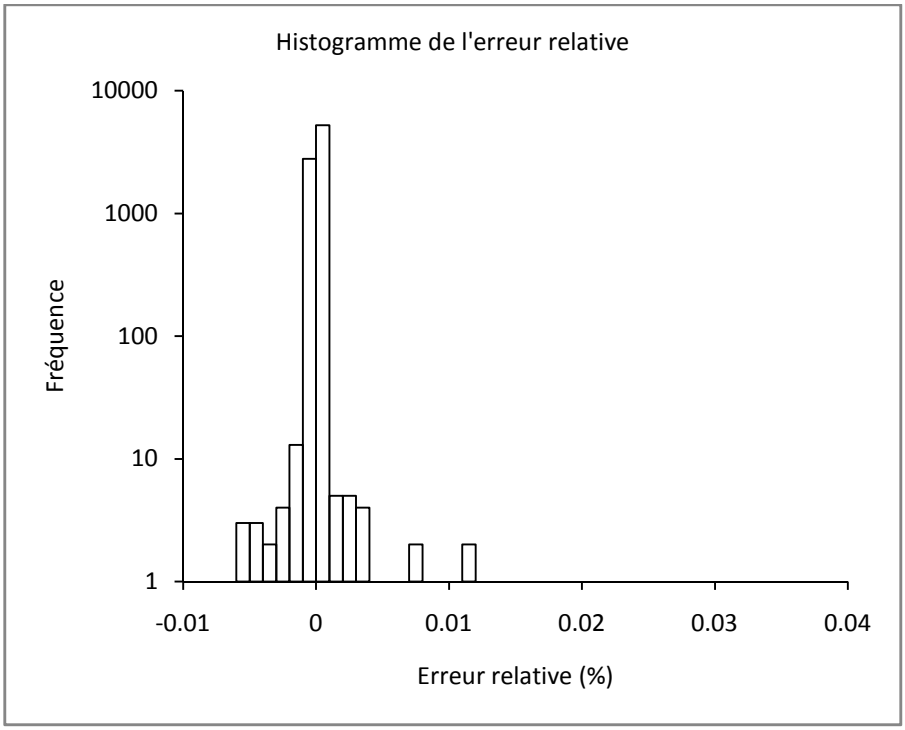

Figure 3-3 Histogramme de l'erreur relative

3.2 Résultats de notre algorithme

Nous présentons ici les résultats concernant l'algorithme que nous avons développé spécifiquement pour le GPU. Nous séparerons ces résultats en deux parties : TERMA et DOSE.

3.2.1 Résultats de TERMA

Comme mentionné à la section 2.3.2.1.1, nous avons conservé deux implémentations de notre algorithme : la première faisant un tracé complet jusqu'à la source, la deuxième faisant une approximation de tracé à la verticale mais utilisant la mémoire partagée de la carte. Ces deux implémentations seront donc étudiées ici. Les tests autres que ceux comparant les diverses cartes graphiques ont été réalisés sur le PC1.

Dans les tableaux qui vont suivre, $A_{GPU/CPU}$ conserve la même signification que celle présentée à l'équation (16).

Nous comparons premièrement l'impact de la taille de la grille de TERMA sur les calculs. Cette taille de grille influence le nombre de points à calculer mais aussi la longueur des tracés à faire pour rejoindre la source d'irradiation.

Tableau 3-6 Temps d'exécution de TERMA pour implémentation avec tracé complet (PC1)

Taille	CPU	GPU	$A_{GPU/CPU}$
	(s)	(s)	
64x64x203	1.05	0.08	13.42
128x128x203	3.70	0.19	19.70
256x256x203	16.75	0.95	17.56

Taille représente la taille de la carte de TERMA, soit le nombre de points de TERMA pour lequel un calcul est effectué.

Tableau 3-7 Temps d'exécution de TERMA pour implémentation avec tracé vertical (PC1)

Taille	CPU (s)	GPU (s)	$A_{GPU/CPU}$
64x64x203	1.05	0.06	17.16
128x128x203	3.70	0.10	38.98
256x256x203	16.75	0.42	39.88

Nous présentons aussi un graphique comportant les accélérations des deux types de tracés implémentés à la Figure 3-4.

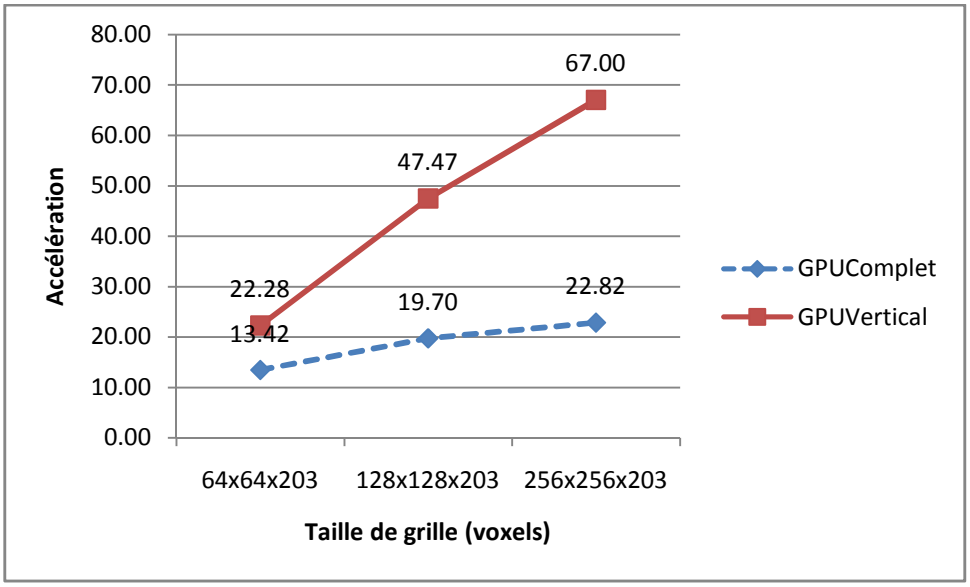

Figure 3-4 Accélération des deux tracés en fonction de la taille de grille

Le prochain ensemble de résultats a été obtenu sur le PC2 décrit au

Tableau 3-2. Nous présentons ici les comparaisons, en termes de temps de calcul, entre l'exécution de notre programme sur une GeForce 8800GTX et une GeForce GTX280.

Tableau 3-8 Temps d'exécution pour les 8800GTX et GTX280 pour le tracé complet (PC2)

Taille	CPU	8800GTX	GTX280	$Acc_{280/8800}$	$Acc_{8800GTX/CPU}$	$Acc_{280/CPU}$
	(s)	(s)	(s)			
64x64x203	0.86	0.05	0.03	1.52	18.28	27.71
128x128x203	3.55	0.22	0.14	1.55	16.20	25.16
256x256x203	16.55	0.97	0.56	1.72	17.10	29.40

Tableau 3-9 Temps d'exécution pour les 8800GTX et GTX280 pour le tracé vertical (PC2)

Taille	CPU	8800GTX	GTX280	$Acc_{280/8800}$	$Acc_{8800GTX/CPU}$	$Acc_{280/CPU}$
	(s)	(s)	(s)			
64x64x203	0.86	0.03	0.02	2.13	26.84	57.27
128x128x203	3.55	0.11	0.06	1.75	32.25	56.30
256x256x203	16.55	0.45	0.25	1.78	37.11	66.20

Nous avons aussi, comme il avait été annoncé au chapitre deux, vérifié l'importance des lectures groupées sur la performance globale du système. Pour ce faire, nous avons utilisé une version du *kernel* qui fait des lectures et des écritures non groupées et qui n'utilise pas le *kernel* de réordonnancement d'index. Nous trouvons les résultats du Tableau 3-10.

Tableau 3-10 Temps d'exécution pour les 8800GTX et GTX280 avec lectures et écritures non groupées (PC2)

Taille	CPU	8800GTX	GTX280	$Acc_{280/8800}$	$Acc_{8800GTX/CPU}$	$Acc_{280/CPU}$
	(s)	(s)	(s)			
64x64x203	0.86	0.03	0.03	1.03	26.84	27.71
128x128x203	3.55	0.14	0.11	1.28	25.16	32.25
256x256x203	16.55	0.55	0.48	1.14	30.26	34.55

Nous avons de plus étudié l'effet des lectures/écritures groupées et non groupées en regardant les temps d'exécution des *kernels* seulement, en utilisant le profileur de NVIDIA. Ce profileur nous donne les temps qu'ont pris les *kernels* pour s'exécuter, sans les pré- et post-traitements effectués du côté de PLUNC. Nous trouvons l'avantage qu'a l'approche par lecture/écriture groupée pour diverses tailles de données en entrée. Ces tests ont été réalisés sur PC2.

Tableau 3-11 Temps d'exécution des kernels pour une matrice de 64x64x64 (PC2)

	Transpose	Terma	Transpose	Total	Accélération groupée
L/E non groupée	-	23.80	-	23.80	0.78
L/E groupée	6.14	18.44	6.10	30.68	

Tableau 3-12 Temps d'exécution des kernels pour une matrice de 128x128x128 (PC2)

	Transpose	Terma	Transpose	Total	Accélération groupée
L/E Non groupée	-	483.00	-	483.00	2.42
L/E groupée	13.80	170.60	14.80	199.20	

Tableau 3-13 Temps d'exécution des kernels pour une matrice de 256x256x256 (PC2)

	Transpose	Terma	Transpose	Total	Accélération groupée
L/E Non groupée	-	1106.00	-	1106.00	2.83
L/E groupée	28.80	342.34	19.90	391.04	

Nous étudions maintenant l'effet de l'utilisation du tracé vertical. Nous ne pouvons comparer nos données directement avec celles que l'algorithme original de PLUNC génère puisque nous n'effectuons pas exactement le même travail. Mentionnons que nous obtenons des résultats en moyenne à 2% près de ceux que PLUNC génère. Notre but ici est d'étudier les répercussions de l'utilisation du tracé vertical. Nous considérons donc nos données générées par l'utilisation d'un tracé complet comme valides et nous comparons ces données à celles générées par le tracé vertical. La Figure 3-5 présente un histogramme de cette erreur relative. Les erreurs sont présentées en pourcentage à la Figure 3-5.

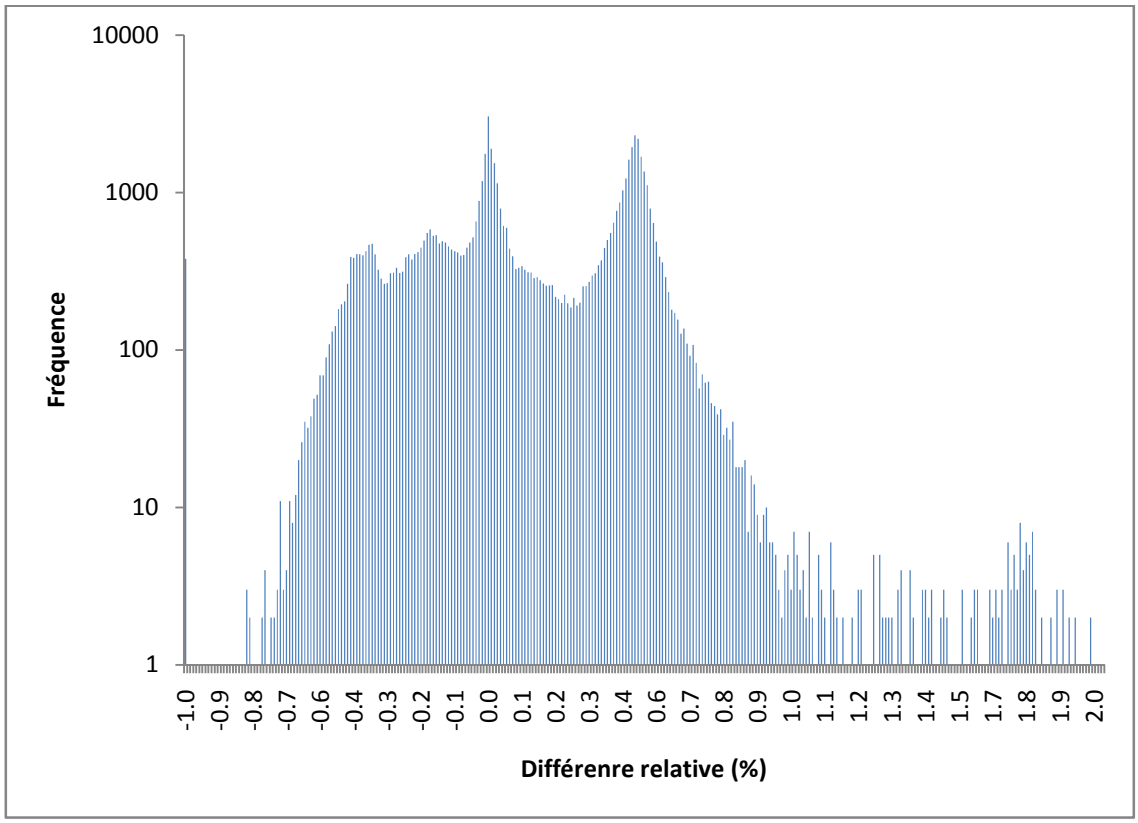

Figure 3-5 Histogramme de la différence relative entre tracé complet et tracé vertical

Finalement, nous montrons l'effet qu'a la taille sur le temps d'exécution du *kernel.* Nous utilisons ici le kernel qui fait un tracé complet et la 8800GT.

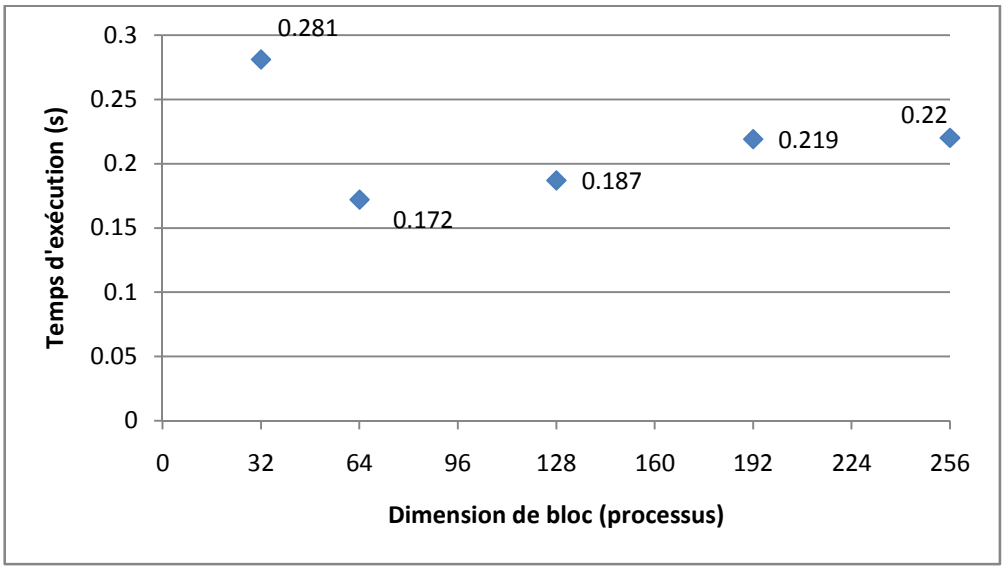

Figure 3-6 Temps d'exécution en fonction de la dimension de bloc

La Figure 3-6 explique pourquoi la taille de bloc a été choisie à 64 pour tous les tests effectués dans cette section de résultats. Ceci n'indique en rien que 64 est le meilleur choix dans tous les cas, mais nous l'avons conservé.

3.2.2 Résultats de DOSE

Cette section présente les résultats des différents algorithmes développés pour effectuer le calcul de dose, donc de convolution. Ces divers algorithmes sont comparés les uns aux autres ainsi que par rapport à l'implémentation originale de PLUNC sur CPU. La comparaison n'est pas parfaite puisque ces deux implémentations ne font pas exactement les mêmes opérations. Il faut donc garder ceci en tête en examinant les résultats. Nous croyons quand même que la comparaison est valide puisqu'ultimement les mêmes quantités sont calculées avec une méthode très semblable. De plus, du point de vue du modèle physique, les mêmes stratégies de calcul sont prises en compte (réalignement de

FDD, interpolation de FDD, tracé radiologique, interpolation trilinéaire des points de TERMA au site d'interaction). Certaines différences dans la quantité de calculs effectués sont à notre désavantage, comme il sera présenté au Chapitre 4. Les différences sont donc inhérentes au fait que nous avons développé ces algorithmes avec une implémentation sur GPU en tête dès le début. Nous utilisons les mêmes ordinateurs de tests, excepté que la carte 8800GTX ne sera plus utilisée puisqu'elle ne supporte pas les opérations atomiques qui sont nécessaires à un des algorithmes utilisés. La notion « d'accélération » est conservée comme à l'équation (16).

Les temps d'exécution qui sont présentés utilisent le même mécanisme de calcul de temps que celui déjà présent dans PLUNC. C'est donc dire que nos temps de calcul ne contiennent pas seulement le temps de convolution mais aussi les temps de transfert de données vers et de la carte graphique ainsi que certaines tâches préalables qui doivent être exécutées dans tous les cas. Notons que des changements de paramètres qui pourraient intuitivement amener un changement linéaire dans le temps d'exécution ne se traduiront pas nécessairement de cette façon. Nous présentons tout d'abord les acronymes qui seront utilisés pour présenter les résultats, ils font références aux algorithmes que nous avons développés :

Tableau 3-14 Acronymes désignant les divers algorithmes de CS

DPOV	Point de vue de dose, distance uniforme
DPOVNU	Point de vue de dose, distance non uniforme
DVPONUPOL	Point de vue de dose, distance non uniforme polaire
TPOV	Point de vue de terma

Les tests qui suivent ont été réalisés à l'aide de la configuration du Tableau 3-15. Si aucune modification à cette configuration de base n'est mentionnée dans le texte explicatif ou dans les variables du tableau, elle est considérée comme en vigueur.

Tableau 3-15 Configuration de base

Taille carte terma		128
$Dist_{DTOV}$	Distance maximale pour l'algorithme TPOV	10
$Dist_{DPOV}$	Distance maximale pour l'algorithme DPOV, couple ($dist_{lat}$,$dist_{prof}$)	(5,10)
NUMPHI	Nombre de discrétisations pour le volume polaire	48
NUMTHETA		8
INCR	Facteur multiplicatif d'incrémentation de rayon pour DPOVNUPOL	1.09
NumPoints	Nombre de points dans la grille de dose	1183

Notons que cette configuration de base fait que l'algorithme DPOVNUPOL intègre le même nombre de points d'interaction que l'algorithme présent dans PLUNC pour évaluer la dose en un point de dose donné, ce qui rend la comparaison plus intéressante.

Tests d'ensemble

Nous commençons par étudier l'effet du nombre de points formant la grille de calcul de dose sur le temps d'exécution. Pour limiter l'espace requis par chaque tableau, seule la valeur de référence CPU sera donnée en absolu, le reste des valeurs représentera une accélération par rapport à cette valeur absolue.

Tableau 3-16 Accélération en fonction du nombre de points de calcul (8800GT)

Nombre de points	CPU	$A_{GPU/CPU}$			
	(s)	DPOV	DPOVNU	DPOVNUPOL	TPOV
676	8.14	5.16	21.71	47.33	6.74
1183	14.34	8.74	35.32	70.64	11.76
2663	32.05	14.45	62.23	102.40	12.98
10655	128.02	23.47	94.13	215.89	14.74

Tableau 3-17 Accélération en fonction du nombre de points de calcul (GTX280)

Nombre de points	CPU	$A_{GPU/CPU}$			
	(s)	DPOV	DPOVNU	DPOVNUPOL	TPOV
676	7.86	5.03	20.96	62.88	10.73
1183	14.64	9.28	42.56	103.84	21.31
2663	31.16	19.00	83.08	220.96	22.91
10655	124.52	59.04	241.78	568.57	25.88

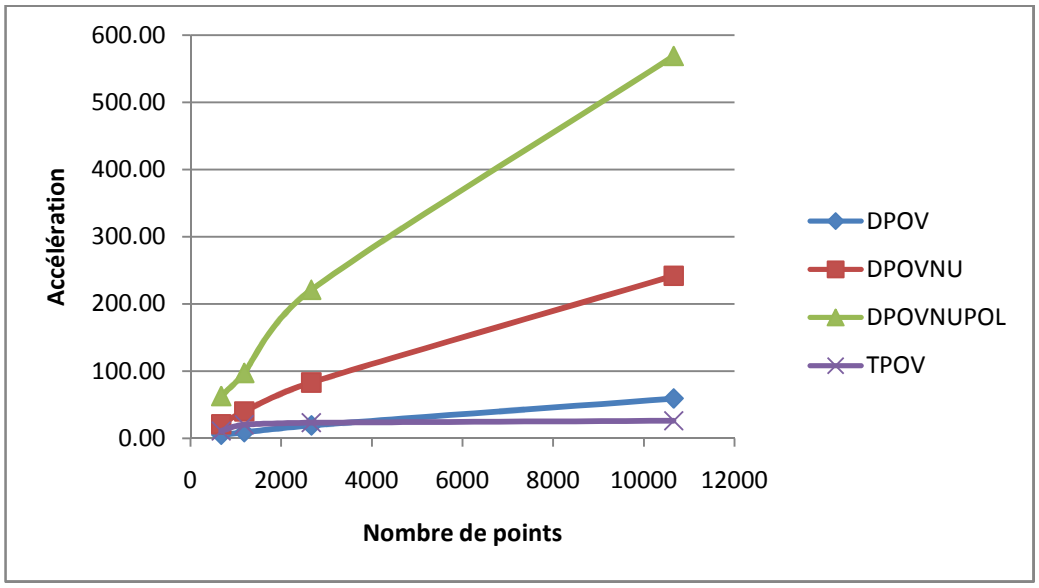

Figure 3-7 Accélération en fonction du nombre de points de calcul (GTX280)

Les valeurs pour le nombre de points à traiter sont le résultat de la discrétisation de la grille de dose en voxels de 1.00cm, 0.75cm, 0.50cm et 0.25cm.

Nous poursuivons en faisant varier la taille (et donc la résolution) de la carte de TERMA utilisée pour faire le calcul de convolution. On remarque que nous ne faisons que varier la taille de la carte pour les dimensions ‘x’ et ‘y’. La raison est que nous avons un contrôle plus facile sur ces deux paramètres alors que la dimension en ‘z’ est plus difficile à changer dans PLUNC.

Tableau 3-18 Accélération en fonction de la taille de la carte de TERMA (8800GT)

Taille carte terma	CPU	$A_{GPU/CPU}$			
	(s)	DPOV	DPOVNU	DPOVNUPOL	TPOV
64x64x203	13.80	31.50	44.08	126.58	44.22
128x128x203	14.34	8.74	35.32	70.64	11.76
256x256x203	14.55	1.71	23.85	38.79	2.96

Tableau 3-19 Accélération en fonction de la taille de la carte de TERMA (GTX280)

Taille carte terma	CPU	Accélération GPU			
		DPOV	DPOVNU	DPOVNUPOL	TPOV
64x64x203	13.53	33.25	41.26	143.96	78.67
128x128x203	14.64	9.28	42.56	103.84	21.31
256x256x203	14.45	2.22	23.69	61.50	5.35

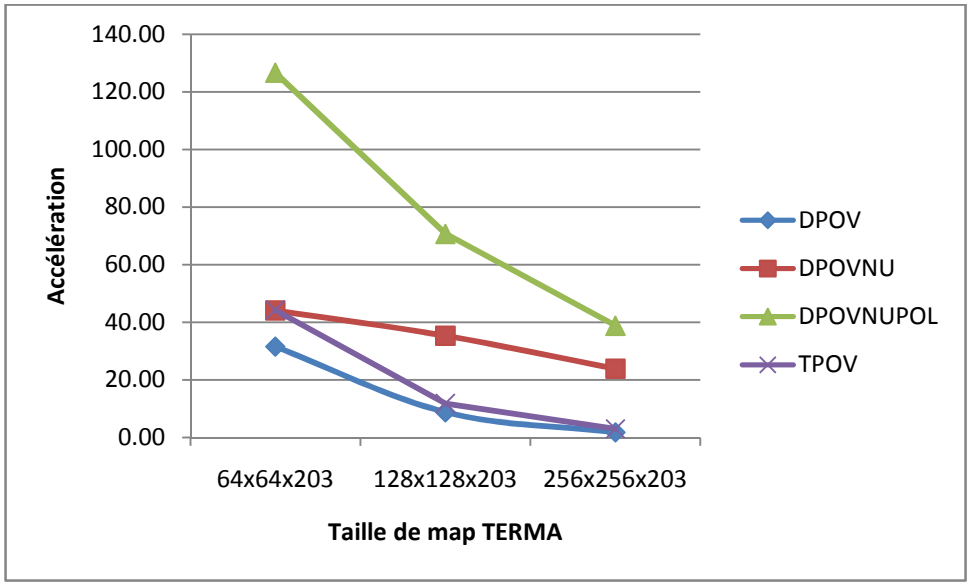

Figure 3-8 Accélération en fonction de la taille de la carte de TERMA (PC1)

Ces deux ensembles de données représentent l'étude la plus large que nous pouvons effectuer sur tous les algorithmes à la fois. Ils représentent aussi les résultats clés de notre travail, qui visait à accélérer le processus de calcul de dose.

Approche non uniforme

Nous présentons maintenant un résultat qualitatif de l'impact d'utiliser une approche non uniforme pour calculer la dose. Le cas utilisé est une tumeur à la prostate qui a servi pour prendre tous les résultats d'accélération présentés. Pour vérifier l'impact sur l'exactitude de l'approche non uniforme, nous limitons les distances à celles utilisées dans l'approche uniforme, soit 5 cm latéralement et 10 cm en profondeur.

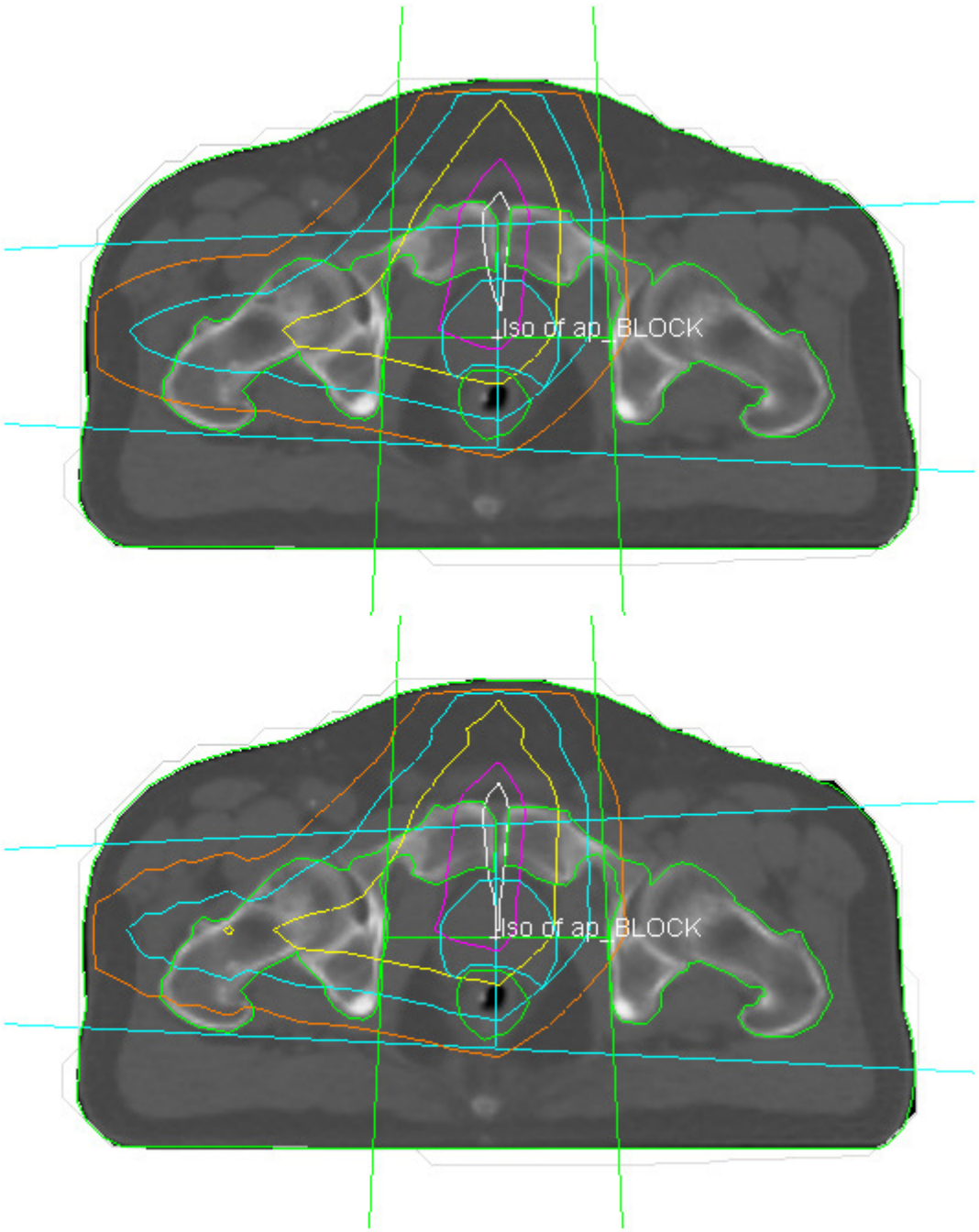

Figure 3-9 Comparaison algorithme uniforme (haut) et non uniforme (bas)

La partie du haut de la Figure 3-9 utilise l'algorithme DPOV alors que la partie du bas utilise DPOVNU.

Étude de sensibilité sur divers paramètres des algorithmes individuels

Les résultats de sensibilité à divers paramètres qui sont présentés portent sur chacun des algorithmes, évaluant l'impact d'un ou plusieurs de ces paramètres. Les tests qui suivent se limiteront à la 8800GT sur PC1.

Nous commençons par la distance maximale à laquelle l'algorithme DPOV intègre les contributions de TERMA. Rappelons que, comme mentionné à la section 2.3.2.2.1, cette distance définit le nombre de voxels à visiter dans chaque direction et par conséquent le nombre de boucles que la convolution effectue. $Distance_{Prof}$ représente la distance en profondeur visitée, elle affecte donc seulement une des trois boucles. $Distance_{Lat}$ affecte les distances dans un plan perpendiculaire au faisceau et affecte donc deux boucles.

Tableau 3-20 Temps d'exécution de DPOV en fonction des distances maximales visitées

3.2.3

Temps d'exécution (s)	Distance Lat	
Distance Prof	5	10
10	0.41	5.98
20	2.05	-

La case sans temps d'exécution indique que le calcul n'a pas eu le temps de terminer avant que le mécanisme de délai d'attente du pilote d'affichage ne termine le processus, soit au moins 8 secondes. Notons rapidement qu'il est possible d'effectuer des calculs qui durent plus de 8 secondes en utilisant CUDA, sous deux conditions : la première est qu'il n'y ait pas de moniteur attaché à la carte graphique effectuant le calcul, la

deuxième est qu'il n'y ait pas de contexte graphique initialisé par le système d'exploitation. Avec le PC1, nous ne pouvions pas répondre à ces conditions.

Nous étudions par la suite l'effet de l'activation du tracé de rayon radiologique dans l'algorithme DPOVNU. Rappelons que si cette option n'est pas activée, le milieu est considéré comme de l'eau et la distance entre deux points n'est effectivement que la norme du vecteur qui les lie. Lorsque le tracé radiologique est activé, un tracé de rayon complet doit être effectué entre le point d'interaction et le point de dose pour cumuler les densités rencontrées.

Tableau 3-21 Coût du tracé de rayon en fonction du nombre de points dans la grille de dose dans DPOVNU

	Nombre de points		
Temps d'exécution (s)	676	1183	2663
Sans tracé rayon	0.375	0.406	0.515
Avec tracé rayon	6.734	7.634	-

La Figure 3-10 montre une comparaison entre un calcul effectué par l'algorithme sans et avec tracé de rayon. Il s'agit d'un cas de tumeur au poumon.

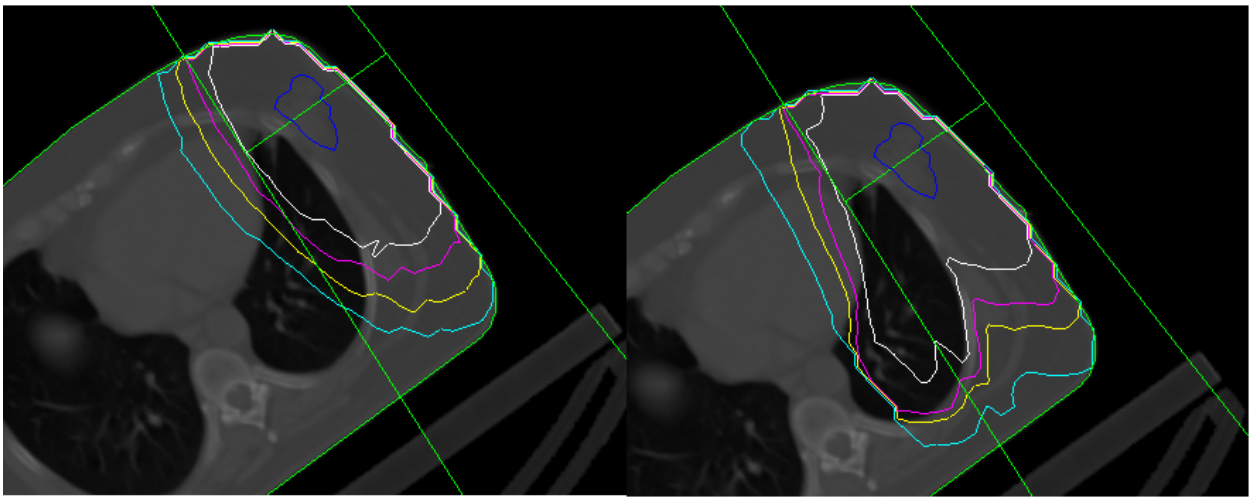

Figure 3-10 Cas de poumons sans (gauche) et avec (droite) tracé radiologique

Comme les poumons sont en grande partie remplis d'air, et de ce fait significativement moins denses que l'eau, ils atténuent moins le rayonnement. Ceci se traduit sur la figure par une pénétration plus grande du rayonnement dans le poumon.

Les algorithmes DPOVNU et DPOVNUPOL font tous deux appel à des distances d'accumulation non uniformes. En d'autres termes, le nombre et la distance des divers échantillons de TERMA qui sont utilisés pour calculer la dose finale sont générés de façon procédurale dans le cas de DPOVNUPOL ou à partir de fichiers de configurations dans le cas de DPOVNU. Par exemple, pour DPOVNUPOL, le produit de NUMPHI, NUMTHETA et le nombre de valeurs de rayon 'r' nécessaires pour atteindre 60cm avec un INCR donné indiquent le nombre d'échantillons de TERMA qui est utilisé. De façon analogue, le produit du carré du nombre d'éléments contenu dans *dist_lat* par le nombre d'éléments dans dist_prof donne le nombre d'éléments visités pour DPOVNU. Une variation de ces paramètres devrait donc influencer le temps de calcul ainsi que

l'exactitude des résultats. Nous nous concentrons ici sur le temps d'exécution puisque nous ne sommes pas en mesure de poser un jugement sur l'aspect quantitatif des résultats. Nous faisons ici varier le nombre d'éléments visités seulement pour DPOVNUPOL puisque l'impact de cette variation doit être le même pour les deux algorithmes. Le triplet entre parenthèses représente le nombre (NUMR, NUMPHI, NUMTHETA).

Tableau 3-22 Temps d'exécution de DPOVNUPOL en fonction du nombre d'éléments visités

Nombre d'éléments visités	16128 (84,24,8)	32256 (84,48,8)	64512 (84,96,8)
Temps d'exécution (s)	0.13	0.20	0.36

Le prochain chapitre discutera de ces résultats ainsi que de nos choix et de l'impact de ce projet de façon plus générale.

CHAPITRE 4 : CRITIQUE ET DISCUSSION

Cette section est aussi divisée en deux sous-sections principales. L'une porte sur le port de l'algorithme de PLUNC et l'autre sur notre nouvel algorithme. Nous présentons aussi une critique et discussion commune aux deux versions.

4.1 Port de PLUNC

Le but de cette implémentation était de montrer le potentiel d'accélération brut et les effets de l'utilisation d'une carte graphique. Ce sont donc ces aspects qui sont discutés ici. Cet exercice est aussi rapporté dans (Hissoiny, 2009) (Hissoiny, 2009).

Section sur erreur numérique

Nous commençons par une discussion sur les erreurs numériques. Rappelons que les données présentées au Tableau 3-5 sont en pourcentage. L'erreur relative moyenne reste basse dans les deux cas (CPU avec simple précision et GPU). Il y a un facteur 10 entre les deux valeurs, mais puisque nous avons des erreurs relatives si petites, il est fort probable que, en soustrayant deux nombres si près l'un de l'autre, nous subissions des erreurs de calcul flottant. Nous avons utilisé des calculs en double précision pour trouver les propriétés de l'erreur et croyons donc que ces résultats sont près de la réalité. Les calculs en double précision sont en mesure de conserver une douzaine de chiffres significatifs et nous trouvons des valeurs d'erreurs relatives autour de 5E-8. Pour trouver cette valeur relative, la valeur absolue de l'erreur a été divisée par la valeur du point de référence, soit environ 1E-2. Nous avons donc fait une soustraction qui a donné

un résultat de l'ordre de grandeur de 1E-10, ce qui devrait être bien supporté par le calcul en double précision. Notons quand même qu'il y a toujours des problèmes d'accumulation d'erreur en soustrayant des nombres si près les uns des autres.

L'erreur relative maximale, quant à elle, fait un saut de 100 entre CPU à simple précision et GPU. L'erreur relative maximale reste quand même petite à, dans le cas GPU, 0.0386%. Nous ne nous inquiétons donc pas outre mesure de cette différence d'erreur relative maximale entre les deux implémentations en simple précision. Le matériel graphique respecte en grande partie les normes IEEE754 gérant le calcul à virgule flottante. Rappelons que notre implémentation en simple précision pour CPU ne faisait que remplacer les variables *float* en *double*. Notons finalement, en ce qui concerne l'erreur, que nous ne pouvons tirer de conclusion autre que dans le cadre spécifique de notre application et de nos données en ce qui concerne l'utilisation de la simple précision. Nous ne sommes nullement en train d'avancer que le calcul en double précision est inutile sur GPU. Pour cette implémentation spécifique, l'usage de la double précision n'était peut-être pas entièrement justifié dans l'algorithme original mais elle pourrait s'avérer critique pour d'autres applications. Il ne faut donc pas prendre nos résultats comme une preuve que le calcul en double précision est inutile sur GPU.

Du point de vue de la vitesse d'exécution, nous trouvons des gains de vitesse entre 15x et 20x. Le facteur d'accélération est d'autant plus grand que la durée des calculs augmente. Nous trouvons donc un facteur de 15x lorsque la granularité de grille est à 1.0 cm jusqu'à 20x lorsque celle-ci passe à 0.25 cm. Nous n'avons pas de raison de penser

que l'avantage du temps d'exécution pourrait passer du côté du CPU. Étant donné la qualité plutôt moyenne de l'implémentation, d'un point de vue exécution sur GPU, ces données d'accélérations plafonnent mais ne diminuent pas.

Nous avons aussi effectué quelques tests qualitatifs sur une autre machine comportant une 8800GTX (14 multiprocesseurs) et une GTX280 (30 multiprocesseurs) sur PC2. Nous n'avons trouvé aucun gain intéressant entre l'exécution sur les deux cartes graphique. Ceci démontre bien que notre programme CUDA est complètement limité par les accès mémoire puisqu'une augmentation de 88% de la puissance de calcul n'amène aucun gain en temps d'exécution.

L'utilisation des unités de texture a un impact non négligeable. Nous voyons en effet un gain de près de 5x additionnel lorsque la granularité de grille augmente. Nous croyons que cette accélération est due à ce que nous avons déjà exposé à la section 2.3.1.2, c'est-à-dire qu'il y a de la localité dans nos lectures en mémoire.

Ces facteurs d'accélération sont intéressants, mais ils pourraient être encore beaucoup mieux si l'algorithme de PLUNC diminuait le volume de lecture et d'écriture en mémoire. De plus, l'utilisation de la mémoire partagée du GPU y est inexistante. Nous n'avons pas jugé bon d'améliorer PLUNC à ce sujet puisque le but de cette section était d'évaluer le potentiel d'accélération brut par l'utilisation d'une carte graphique pour une solution déjà existante. Nous avons donc démontré que cette accélération est à la portée des auteurs qui ont écrit l'application originale. La seule fonction ayant eu à être significativement modifiée est celle qui lance l'exécution sur la carte graphique. Elle

doit allouer les espaces mémoire et préparer les textures. Néanmoins, ces opérations sont bien expliquées dans le manuel CUDA ainsi que présentes dans tous les exemples fournis par NVIDIA.

4.2 Notre algorithme

Comme pour la section des résultats, nous allons diviser la discussion de notre algorithme en deux parties : discussion sur la TERMA et discussion sur la DOSE.

4.2.1 TERMA

Tracé complet

La Figure 3-4 et au Tableau 3-6 présentent les résultats de tracé complet pour le calcul de la TERMA. Nous comparons ces temps de calcul à ceux de l'algorithme original de PLUNC. Nous voyons des accélérations entre un et deux ordres de grandeur. Nous trouvons donc une accélération très intéressante malgré le fait que nous effectuons un travail plus exact que celui réalisé dans PLUNC. En effet, l'implémentation de PLUNC utilise un tracé vertical. C'est donc dire que le calcul de la TERMA, dans PLUNC, n'est pas fidèle aux données du patient en tous points.

La variation de la taille de la grille influe sur deux fronts : tout d'abord, le nombre de points pour lequel la TERMA doit être calculée est plus grand; ensuite, la longueur des tracés, en termes de voxels traversés, augmente. Nous voyons quels effets a ce facteur aux tables et figures susmentionnées. L'accélération est plus ou moins dépendante de la taille de la grille, c'est-à-dire qu'elle reste constante à mesure que la taille de la grille

augmente. Nous ne pouvons donc pas voir de point où l'algorithme s'exécutant sur le GPU pourrait perdre son avantage par rapport à celui sur CPU. Ceci n'est pas nécessairement toujours le cas et sera discuté plus en détails lors de la discussion sur l'implémentation par mémoire partagée et tracé vertical.

Au Tableau 3-8, nous présentons les temps de calcul de la TERMA par rapport à la carte graphique utilisée pour effectuer ce calcul. Comme mentionné au chapitre des résultats, la carte 8800GTX comporte 16 multiprocesseurs alors que la carte GTX280 en comporte 30. Nous avons donc une augmentation de près de 100% dans le nombre de processeurs accessibles, puisque chaque multiprocesseur possède huit processeurs et ce pour les deux cartes. La fréquence d'opération des processeurs est similaire pour les deux cartes à 600MHz. Notons que dans tous les cas, tous les multiprocesseurs de toutes les cartes étaient pleinement utilisés et qu'il n'y a donc pas eu de cas où nous n'avions pas assez de points à calculer pour occuper toute la GTX280. Nous voyons un gain substantiel pour l'utilisation de la GTX280 à mesure que la charge de travail augmente. Quelques facteurs peuvent expliquer ce gain en vitesse.

Premièrement, les cartes de la famille de la 8800GTX ont la possibilité d'effectuer des opérations MADD (MULTIPLY_ADD), c'est-à-dire que les opérations de la forme y=ab+c sont fusionnées en une seule opération, MADD, en plus d'effectuer une multiplication normale par coup d'horloge. Cette fonctionnalité est par contre difficilement accessible sur la famille G80[7] dont la 8800GTX fait partie. L'arrivée de la

[7] 'The case of the missing mul' http://beyond3d.com/content/reviews/1/11

famille des GT200 (dont la GTX280 fait partie) a rendu l'exploitation de la fonctionnalité MADD beaucoup plus fréquente lors de l'exécution du code. Or, notre algorithme fait grand usage d'assignations qui bénéficient de l'opération fusionnée MADD. En effet, nous en trouvons une à chaque itération de l'algorithme de Siddon (donc à chaque fois qu'un voxel est visité) et dans les phases d'accumulation pour le calcul hors axe et de la valeur finale de la TERMA.

Deuxièmement, le nombre de calculs arithmétiques à faire pour un point de TERMA n'est pas négligeable si on le compare au nombre d'accès mémoire à faire. En effet, on doit boucler deux fois sur le nombre de discrétisations du spectre (18 conteneurs pour la modélisation du spectre de l'appareil fourni avec PLUNC), et chacune de ces boucles comporte plusieurs multiplications. Nous pouvons donc penser qu'une partie de la latence des accès mémoire est cachée par les calculs à effectuer. Or, la GTX280 est mieux équipée pour effectuer ces calculs, avec son nombre de processeurs supplémentaires.

Finalement, la bande passante de la GTX280 est supérieure et les écritures à la fin du calcul de la TERMA sont groupées. Ces écritures peuvent donc tirer avantage de cette bande passante plus élevée pour réduire le temps passé à écrire les résultats en mémoire globale.

Notons que nous n'avons utilisé aucun calcul en double précision lors du calcul de la TERMA. Ces calculs auraient été faits en calcul simple précision pour la 8800GTX et donc sans aucune perte de performance mais ils auraient été conservés en double

précision pour la GTX280. Comme il n'y a qu'une unité de calcul en double précision par multiprocesseur (donc un facteur 1:8 par rapport au nombre d'unités en simple précision), le temps de calcul sur la GTX280 en aurait souffert.

Tracé vertical

Les résultats concernant le tracé vertical sont présentés au Tableau 3-7, à la Figure 3-4, à la Figure 3-5 et à la Figure 3-5.

Le Tableau 3-7 et la Figure 3-4 nous donnent les accélérations entre la version de PLUNC sur CPU (qui fait d'ailleurs aussi un tracé vertical) et notre version sur GPU. Nous y trouvons des accélérations de 16x à 39x, ce qui est à l'intérieur de nos objectifs d'un à deux ordres de grandeur. Notons quand même que nous faisons un tracé vertical, comme PLUNC, mais que nous faisons un calcul hors axe pour chaque point alors que PLUNC fait le calcul hors axe pour une ligne verticale. Nous effectuons donc quand même plus de travail que la version PLUNC, pour des taux d'accélération remarquables. Nous semblons par contre atteindre un plafonnement de l'accélération autour de 40x. Nous ne pouvons pas voir de point où la taille de grille ferait pencher l'avantage du côté CPU, tant que par l'espace de mémoire vidéo requis pour stocker les est suffisant. Une grille de 512x512x203 contiendrait environ 53 millions d'éléments, pour 203 Mo de mémoire requise pour les données en entrée, et autant en sortie. Nous nous approchons donc de la limite de l'espace allouable sur une carte graphique en contenant 512MB, comme la 8800GT utilisée pour réaliser une partie des tests. Les calculs pourraient par contre être séparés en plusieurs appels, pour réduire la taille des données nécessaires

pour chaque appel, ce qui serait plus difficile à réaliser pour le tracé complet puisque nous ne savons pas d'avance, d'une façon simple, quelles données seront nécessaires.

Pour ce qui est de la comparaison entre 8800GTX et GTX280 (

Tableau 3-9), nous avons obtenu des résultats correspondant à nos attentes. En effet, nous posions l'hypothèse que le plus haut potentiel de calcul de la carte GTX280 pourrait mieux s'exprimer étant donné que les cartes seraient moins en attente sur des transactions de mémoire globale, puisque nous utilisons principalement la mémoire partagée. Or, nous obtenons pour chaque carte des résultats plus rapides que pour les tests correspondants avec tracé complet et la comparaison est plus favorable pour la GTX280 qui bénéficie d'un gain plus important. En effet, la comparaison passe de 1.55x à 1.74x lorsque la taille de grille est de 128x128x203 en comparant le tracé complet au tracé vertical.

Tel que mentionné au Chapitre 2, nous avons aussi expérimenté avec les lectures non groupées pour ce *kernel*. Comme nous voyons au Tableau 3-10, l'impact sur la performance est très grand pour la GTX280. Elle perd son avantage par rapport à la 8800GTX qui lui est pourtant inférieure en ne regardant que la puissance brute de calcul. Le sous-système de mémoire a par contre été revu sur la carte GTX280 par NVIDIA, pour adoucir les exigences nécessaires aux lectures groupées. Ces changements semblent aussi avoir augmenté la pénalité lorsque des lectures non groupées sont effectuées. Il est donc d'autant plus important de faire des lectures groupées sur les cartes de la famille GT200. Les Tableau 3-12 et Tableau 3-13 montrent que l'avantage de notre méthode est

dépendante de la taille de la matrice à traiter. En effet, plus il y a de données d'entrée, plus leur ordre sera important puisqu'il y aura de plus en plus de lectures à effectuer. Nous avons donc d'autant plus de pénalité pour les lectures désordonnées. Dans les faits, comme nous lisons des données de 4 octets, une lecture groupée est capable de lire la donnée de chacun des 16 processus d'un *half-warp* en une lecture, réduisant effectivement le nombre de lectures à faire par 16. À mesure que le nombre d'éléments dans la matrice augmente, le nombre de lectures, par la technique groupée, augmente 16 fois moins rapidement.

Nous avons aussi fait une étude de l'impact de l'utilisation du tracé de rayons complet et vertical. La Figure 3-5 montre l'histogramme de l'erreur relative entre le tracé complet et le tracé vertical. Nous y voyons une distribution des erreurs relatives qui est, comme on peut s'y attendre, centrée sur 0. On y voit aussi que la très grande majorité des erreurs relatives se situe entre moins deux et plus deux pour cent ([-2%,2%]). Il peut sembler y avoir un bon nombre de points avec une erreur relative importante (>10%) mais il faut porter attention au fait que l'échelle des fréquences est logarithmiques. En effet, seulement 3.7% des points ont une erreur relative de plus de 4% (en valeur absolue) et 1% ont une erreur relative de plus de 10%. Le but ici est d'évaluer l'impact de l'approximation par tracé vertical et nous ne nous attendions bien évidemment pas à obtenir des résultats équivalents en prenant la technique du tracé vertical. Cette technique a été développée pour offrir l'alternative à la technique du tracé complet pour donner des résultats plus rapidement mais en faisant une approximation importante du point de vue de l'utilisation des données spécifiques du patient. L'interprétation de cette

distribution d'erreur est donc laissée aux cliniciens qui sont en mesure de dire si cette stratégie est acceptable.

Une autre remarque importante face à l'approximation du tracé vertical est qu'il devrait être possible d'avoir le meilleur des deux mondes, c'est-à-dire de pouvoir charger une ligne complète de densité en mémoire partagée et de conserver un tracé radiologique réel à travers les « bons » voxels. Il s'agit d'effectuer une phase de prétraitement sur le volume de voxels pour lui faire une mise en forme qui placera effectivement les voxels d'une même ligne de projection les uns à la suite des autres. Cette manipulation est similaire à faire une projection en perspective en rendu graphique. Elle a aussi été développée par Philippe Lacroute pour son algorithme de « shear-warp factorization » (Lacroute, 1995) (Lacroute, 1995) utilisé en rendu volumétrique. Nous invitons d'ailleurs les lecteurs voulant en apprendre davantage sur cette technique à regarder la figure 3.2 de la thèse de P. Lacroute. Nous pourrions ainsi avoir accès aux gains de vitesse de notre algorithme de tracé vertical tout en gardant l'exactitude d'un tracé radiologique complet. Il faut par contre que le temps d'exécution de cette phase de prétraitement reste petit par rapport au temps d'exécution du calcul de la TERMA, ce que nous ne pouvons affirmer.

Finalement, une courte note sur l'impact de la taille de bloc présenté à la Figure 3-6. Comme nous n'avons aucune collaboration entre les processus d'un bloc, nous ne sommes pas bornés quant à la taille des blocs de processus par la taille de la mémoire partagée. Nous voyons donc l'impact que peut avoir la taille de bloc sur l'exécution d'un

programme CUDA. Dans notre cas, nous obtenons un meilleur temps d'exécution pour des tailles plus petites puisque nous sommes en mesure de démarrer plus de blocs de processus par multiprocesseur de façon concourante. En effet, comme chaque bloc de processus utilise un nombre total de registres plus petit, plus de blocs peuvent être assignés à un multiprocesseur avant que celui-ci ne manque de registres.

4.2.2 DOSE

Cette section discute des résultats pour notre implémentation du calcul de convolution pour obtenir la dose.

Les Tableau 3-16 et Tableau 3-17 présentent les facteurs d'accélération pour les différents algorithmes développés pour faire le calcul de la convolution. On peut donc remarquer l'éventail assez important de ces facteurs d'accélération, partant de 5.16x pour aller jusqu'à 568x. Dans tous les cas, plus il y a d'éléments à calculer, plus les facteurs d'accélération sont importants. Ceci s'explique par le fait que l'aspect parallèle de la carte est de mieux en mieux exploité à mesure que le nombre de points à traiter augmente. Ceci s'applique seulement pour les trois premiers algorithmes puisque pour TPOV nous lançons un processus par point de TERMA, il y a donc amplement de processus lancés. Pour le cas à 676 points, comme nous avons 64 processus par bloc, nous ne lançons que 11 blocs, ce qui n'est même pas suffisant pour utiliser tous les multiprocesseurs de la 8800GT. Par contre, avec 10655 points, nous lançons 166 blocs, ce qui est beaucoup plus en accord avec la philosophie CUDA qui indique qu'un grand nombre de blocs lancés permet à l'ordonnanceur de choisir parmi un ensemble de blocs

ceux qui sont prêts à exécuter pendant que d'autres sont en attente d'une transaction mémoire, par exemple. Un des facteurs importants pour ces grandes accélérations, outre l'aspect évident du parallélisme du problème, est le fait que nous obtenons des interpolations trilinéaires « gratuites » en utilisant des textures trois dimensions pour stocker la TERMA et les densités. L'algorithme de PLUNC sur CPU doit effectuer 8 lectures mémoires pour effectuer l'interpolation alors que la carte graphique possède du matériel dédié pour faire ces opérations communes d'interpolation en rendu graphique.

L'algorithme de DPOV est plus lent que les deux autres en point de vue de dose puisqu'il utilise beaucoup plus de points pour faire la convolution. En effet, pour le cas avec une taille de carte de TERMA à 128x128x203, le cas de base, nous nous trouvons avec des facteurs d'échelle pour les axes x et y de 0.15 cm/voxel et 0.12 cm/voxel en z. Comme nous voulons aller à 5 cm latéralement en 10 cm en profondeur, ceci donne un total de 726 000 points de TERMA à considérer. On n'a qu'à comparer cette valeur aux 32 256 points nécessaires pour l'algorithme de DPOVNUPOL (cas de base) pour voir d'où la différence en temps d'exécution provient. L'algorithme DPOV peut donc voir « moins loin » et le fait en plus de temps que les deux variantes non uniformes. Nous pouvons donc déjà à ce point favoriser l'approche non uniforme lorsqu'une valeur de dose rapide sur un volume étendu est souhaitée.

L'approche par point de vue de TERMA est, comme nous l'avions prédit, la plus lente des quatre. Ceci s'explique par le fait que tous les points de TERMA sont lus une fois, puisque chaque point correspond à un processus, et que tous les points de dose sont

considérés pour chaque point de TERMA. Il n'y a donc pas possibilité de faire un test d'exclusion rapide dans la version actuelle des choses. Une classification spatiale des points de dose pourrait permettre de ne charger que les points de doses ayant trait à un ensemble de points de TERMA donné. De plus, dans l'optique où une précision supérieure serait nécessaire, c'est-à-dire où tous les points de TERMA devraient être considérés pour tous les points de dose, nous croyons que cette approche serait la plus rapide grâce à son utilisation poussée de la mémoire partagée.

L'approche DPOVNUPOL semble être en tête à tous points de vue, excepté peut-être celui de la précision supérieure qui vient d'être mentionné. En plus d'être la plus rapide de toutes les solutions, par plus de 100%, elle permet d'obtenir le tracé radiologique gratuit puisque les points de TERMA sont pris le long d'un rayon sur lequel il est possible de sommer les densités rencontrées plutôt que d'effectuer un tracé radiologique complet entre chaque paire de points d'intégration.

En ce qui concerne la comparaison entre la 8800GT et la GTX280 (Tableau 3-17), nous voyons des résultats qui sont attendus. Premièrement, la première ligne du tableau nous montre que la carte graphique n'était pas complètement utilisée pour les deux cartes puisque les facteurs d'accélération sont sensiblement les mêmes, excepté bien sûr pour l'algorithme TPOV qui utilise toutes les capacités de la carte même pour un petit nombre de points de dose. Nous voyons aussi que les gains en accélération subséquents sont environ d'un facteur deux entre la 8800GT et la GTX280. Ceci s'explique par le fait que cette dernière comporte 240 SPs alors que la 8800GT en comporte 114, soit un

facteur d'environ deux. L'époustouflant facteur d'accélération de 565x vaut une mention spéciale d'autant plus que les accélérations augmentent avec le nombre de points de dose à calculer.

Le Tableau 3-18 présente le facteur d'accélération en fonction de la taille de la carte de TERMA. Rappelons que cette carte a été calculée lors de l'étape précédente et qu'elle sert maintenant de donnée d'entrée pour l'étape de convolution.

Pour l'algorithme DPOV, l'augmentation du temps de calcul était prévisible, comme nous l'avions spécifié au Chapitre 2. En effet, comme l'approche utilise des distances uniformes, plus la carte sous-jacente est fine, plus un nombre élevé de voxels doit être visité pour parcourir une même distance géométrique. C'est pourquoi nous voyons un facteur de près de 4 entre chaque facteur d'accélération. La taille en 'x' et 'y' de la carte double à chaque ligne de la table, il y a donc quatre fois plus de voxels pour une tranche en 'z', et donc quatre fois plus de voxels à visiter pour parcourir une distance donnée.

Pour les algorithmes DPOVNU et DPOVNUPOL, la taille de la carte de TERMA ne devrait avoir aucun effet direct sur les temps de calcul de convolution. C'est en quelque sorte ce que nous trouvons, malgré ce que le Tableau 3-18 peut présenter. En effet, en isolant le temps d'exécution du *kernel* qui effectue la convolution (à l'aide du profileur NVIDIA), celui-ci reste approximativement le même à travers les diverses tailles de carte de TERMA. La différence d'accélération provient surtout du transfert de cette carte vers le matériel graphique, sans oublier que la carte de densités sera de la même taille que la carte de TERMA, ce changement de taille a donc un double effet. En effet, une

étude utilisant le profileur de NVIDIA dans une application isolée reproduisant notre calcul de dose avec des données aléatoires montre que le temps de copie passe de 3 ms pour une carte de 128x128x128 à 26 ms pour une carte de 256x256x256. Comme les temps de calcul absolus utilisés pour déduire les accélérations du Tableau 3-18 se situent tous en deçà de 0.6 s, cette augmentation de temps de copie représente une partie significative du temps de calcul complet. De plus, une étude du temps spécifique d'exécution pour le *kernel* effectuant la convolution montre qu'il est stable pour une carte de 64x64x209 et 128x128x209 à 381 ms. Nous affirmons donc que l'influence sur le facteur d'accélération est due aux tâches connexes à la convolution en tant que telle, principalement les copies de mémoire vers la carte graphique. Il faut aussi comprendre que, dans une plateforme pensée pour le GPU, nous n'aurions pas besoin de retransférer les points de TERMA vers la mémoire vidéo pour le calcul de convolution. Nous devons le faire dans le cas présent puisque PLUNC génère des processus (hôtes) pour les diverses tâches à effectuer (TERMA et DOSE) et nous ne pouvons conserver les allocations mémoires vidéo à travers différents processus (hôtes). Dans une meilleure approche, la TERMA et la DOSE se feraient à l'intérieur du même processus (hôte) et nous n'aurions qu'à laisser les données de terma et de densité en mémoire vidéo, où elles pourraient être utilisées subséquemment lors de la phase de DOSE.

L'algorithme TPOV affiche quant à lui une baisse très marquée, et prévisible, des performances lorsque la taille de la carte augmente. Puisqu'un processus est lancé par point dans la carte de TERMA, une augmentation de ce nombre de points amène une augmentation directe et linéaire du travail à effectuer pour la carte graphique.

La Figure 3-9 présente une comparaison entre l'approche uniforme et non uniforme. Les courbes affichées représentent les mêmes isodoses. Nous trouvons donc que les deux approches donnent des résultats très comparables. Les courbes, dans le cas non uniforme, présentent par contre un aspect un peu plus crénelé en comparaison avec les courbes lisses de l'approche uniforme. Ceci a à voir avec les sauts que subissent la taille des pixels de la FDD. En effet, comme la contribution de dose pour un élément de FDD correspond au produit de la valeur de la FDD par la taille du pixel de la FDD par la valeur de terma au centre de ce pixel, il y aura des grandes variations du fait que deux pixels éloignés successifs ont une taille très différente alors que la valeur du pixel ainsi que la valeur de la terma associée varient plus lentement. Finalement, il faut garder en tête que nous avons limité les distances non uniformes aux mêmes que celles utilisées pour l'approche uniforme. L'implémentation non uniforme non modifiée utilise des distances maximales qui vont plus loin que l'approche uniforme.

Étude de sensibilité sur divers paramètres des algorithmes individuels

Le premier cas particulier testé consistait à faire varier la distance maximale pour les boucles de l'algorithme DPOV. Le Tableau 3-20 rapporte ces résultats. On voit l'influence marquée qu'ont ces paramètres sur le temps d'exécution total de la convolution. Comme il était prévisible, le paramètre latéral affecte d'autant plus le temps d'exécution puisqu'il est la limite de deux des trois boucles. Encore une fois, le temps d'exécution manquant indique que le calcul n'a pu s'effectuer dans la limite d'environ 8 secondes permise par le mécanisme de délai d'attente avant que celui-ci ne

termine le processus. Ces résultats montrent aussi qu'il serait impensable d'utiliser cette approche pour des distances maximales de l'ordre de 60 cm, comme utilisées dans les algorithmes non uniformes. Cet algorithme doit donc rester pour des cas où la radiation n'est pas très pénétrante, donc pour des énergies faibles.

Le deuxième paramètre testé est l'activation du tracé radiologique pour les algorithmes cartésiens, tel l'algorithme DPOVNU pris en exemple au Tableau 3-21. L'influence de l'activation du tracé radiologique est énorme. En termes qualitatifs, on passe d'un temps quasi instantané à une attente marquée, ce qui est significatif pour les utilisateurs du logiciel. En termes quantitatifs, c'est une pénalité d'environ 20x qu'amène un tracé radiologique complet. Cette pénalité ne semble par contre pas suivre linéairement le nombre de points à calculer, les caches de texture aidant peut-être à limiter les dégâts. Ceci donne donc une autre indication que l'algorithme DPOVNUPOL est celui de ceux développés qui devrait être utilisé dans un cas général.

La Figure 3-10 montre l'importance du tracé radiologique complet en termes d'exactitude de la solution. Il s'agit d'un poumon, lequel possède un coefficient d'atténuation linéaire plus faible que l'eau. On voit dans la partie de droite de l'image que la radiation est beaucoup moins atténuée à son passage dans le poumon et qu'une plus grande partie de radiation affecte les régions à l'arrière du poumon en prenant le point d'origine du faisceau comme référence. Cette image vise donc à démontrer l'importance de faire un tracé radiologique complet pour les calculs de convolutions.

Mentionnons toutefois qu'il ne s'agit pas là d'une contribution originale. Il est habituel d'effectuer ce genre de calcul qui tient compte des hétérogénéités.

Nous avons par la suite fait varier le nombre de voxels visités pour effectuer le calcul de convolution au Tableau 3-22. C'est sans surprise que la variation dans le temps d'exécution est linéairement influencé par le nombre de points utilisés. Suivant l'équation linéaire décrivant cette relation, 190 600 points pourraient être utilisés pour conserver un temps de calcul d'une seconde, ce qui représente six fois plus de points que la version CPU de l'algorithme équivalent, pour une accélération qui resterait près de 15x. Nous ne pouvons par contre pas commenter l'influence qu'aurait eue une augmentation du nombre de points de TERMA pris en considération pour le calcul de convolution.

4.3 Critique et discussion commune

La présente section discute d'éléments communs aux implémentations effectuées. Nous nous intéressons particulièrement aux impacts que l'approche par GPU amène.

Notre but était d'accélérer le calcul de dose, ce que nous avons réussi avec des accélérations de 15x à 65x, pour la partie TERMA par rapport à une exécution sur processeur général, et ce pour une précision de calcul équivalente et une modélisation parfois supérieure pour l'algorithme sur carte graphique.

Notre objectif premier était une amélioration en termes de temps de calcul. La première partie de ce travail, le port d'un algorithme existant, a atteint cet objectif. Nous avons en

effet démontré qu'à précision et modèle équivalents, une exécution sur carte graphique est accélérée de plus de 20x par rapport à une exécution sur CPU en effectuant un simple port de code existant. De plus, dans la deuxième partie du travail où nous avons fait une implémentation de TERMA et une modélisation supérieure, nous avons obtenu des accélérations jusqu'à près de 30x et jusqu'à plus de 66x pour un modèle de précision équivalente. Notre algorithme de convolution a bénéficié d'accélérations jusqu'à 500x, ce qui représente un changement très significatif pour les utilisateurs des systèmes de planification de traitement. Il est maintenant possible de faire le calcul de dose tridimensionnel avec traçage de rayon complet en un temps compatible avec une utilisation clinique.

Notre deuxième objectif concerne l'amélioration du modèle utilisé pour les calculs. L'utilisation de la carte graphique a permis, à l'intérieur de notre propre algorithme de TERMA, d'implémenter un modèle physique de tracé de rayon plus rigoureux, tout en conservant des temps d'exécution de 13x à 30x plus rapide que l'exécution sur CPU. Nous ne pouvons par contre pas qualifier ce gain en exactitude puisque nous n'avons pas de données maitresses avec lesquelles comparer nos résultats. Ces données auraient été obtenues par simulation de Monte-Carlo.

Nous avons aussi obtenu des accélérations de 500x pour les calculs de convolutions, ce qui nous amène à penser que des modèles physiques plus fidèles à la réalité pourraient être utilisables, pour une utilisation clinique, tout en conservant des temps de calcul intéressants. Nous n'avons donc pas démontré de gains en exactitude pour le calcul de

convolution mais nous suggérons qu'un modèle physique plus fidèle pourrait être utilisé puisque nos temps de calcul sont accélérés substantiellement. Il y a donc possibilité de complexifier le modèle physique, et par le fait même d'augmenter la charge de calcul, tout en conservant un temps de calcul acceptable.

Nous avions aussi émis l'hypothèse qu'une approche par carte graphique demanderait moins d'efforts qu'une approche par grappe de calcul et une architecture par passage de message (MPI). Nous ne pouvons pas quantifier cet effort puisque nous n'avons pas implémenté la solution MPI équivalente. Notre expérience de l'approche MPI et celle acquise au cours de ce projet nous permettent quand même d'affirmer que la solution utilisant une carte graphique est plus accessible qu'une solution équivalente utilisant MPI pour répartir le travail sur une grappe de calcul. Du point de vue logiciel, la modification d'un algorithme pour utiliser MPI est non triviale. L'usager doit gérer les communications ainsi que la séparation explicite du travail à faire par chaque processus. Par contre, l'approche MPI permet d'avoir des fonctionnalités qui ne sont pas seulement SIMD. En effet, un nœud peut travailler à un problème alors qu'un autre nœud peut travailler sur une autre partie du problème, combinant finalement ces deux éléments de solutions. Nous croyons par contre que, pour un algorithme qui se prête bien à un modèle SIMD, l'approche par CUDA est supérieure. Du point de vue matériel, l'approche par carte graphique offre plusieurs avantages. Tout d'abord l'utilisation d'une carte graphique est bien sûr moins coûteuse que l'achat des 10 à 15 postes de travail qui pourraient égaler sa puissance de calcul, du point de vue de l'achat, de la maintenance et du coût d'exploitation. Une étude des coûts du matériel informatique montre une

supériorité d'environ un facteur 10 en ce qui concerne le prix par FLOPS d'un CPU et d'un GPU. En effet, le processeur core i7 d'Intel effectue environ 70 GFLOPS et coûte 350$ alors qu'une GTX280 coûte aussi 350$ mais effectue 933 GFLOPS. De plus, une approche par grappe de calcul requiert l'implémentation d'un réseau de communication à haute bande passante. L'approche par CUDA limite par contre le nombre de nœuds de calcul à notre disposition. En effet, un poste de travail peut contenir un nombre fini de cartes graphique alors qu'il est toujours possible de greffer un nœud de plus à notre grappe de calcul. Nous croyons, malgré tout, que le 200$ que coûte la 8800GT utilisée pour réaliser certains des tests surpasse largement ce que plusieurs fois ce montant permettrait d'acheter en termes de grappe de calcul.

Au cours de ce travail de recherche, nous avons utilisé au moins deux cartes graphiques pour les résultats de temps d'exécution les plus importants. Nous avons pu voir que certains algorithmes bénéficient grandement de la capacité de calcul ajoutée alors que d'autres y sont moins sensibles. Par exemple, notre port de l'algorithme de PLUNC en bénéficie très peu étant donné le nombre élevé de requêtes mémoire faites. Ces requêtes mémoire prennent complètement le dessus sur les calculs algébriques que la carte doit effectuer et la bande passante de la carte devient le facteur limitatif. À l'opposé, notre algorithme de convolution, qui requiert 4 accès mémoire par points de convolution, au lieu de 29 (dans le meilleur des cas) pour le port de l'algorithme de PLUNC, montre des accélérations exceptionnelles.

Finalement, nous aimerions apporter quelques critiques et remarques valides pour l'ensemble du projet.

Comme nous l'avons mentionné au début de ce chapitre, la version de PLUNC utilisée ne supporte pas une exécution parallèle sur les différents cœurs d'exécution disponibles (*multithreading*). Le code source inclut les références aux *pthreads,* qui sont à la base de ce mode de fonctionnement, mais le nombre de processus légers est défini et fixé à l'intérieur du code à 1. Nous ne pouvons que postuler qu'une exécution parallèle sur CPU offrirait aussi des gains très intéressants et, jusqu'à un certain point, quasi linéaire par rapport au nombre d'unités d'exécutions disponibles. Ce point limite serait déterminé par la capacité du bus mémoire à transférer les éléments nécessaires aux calculs sur les différents cœurs d'exécution. Nous n'avons aucune base pour conjecturer sur l'accélération réelle possible ni sur le niveau de parallélisme au-delà duquel le retour sur l'investissement cesserait d'être linéaire. Il est par contre évident que l'avenir se dessine comme une époque multi-cœurs pour les processeurs à usage général avec de plus en plus de cœurs par génération. La même chose est aussi vraie pour les cartes graphiques. La nouvelle carte GTX295 (2008) de NVIDIA comporte 480 SPs, comparé aux 112 utilisés sur PC1 équipé de la 8800GT, et 240 pour PC2 équipé de la GTX280. Cette explosion du parallélisme ne semble pas en voie de s'estomper pour les cartes graphiques.

L'impact de ce projet se fera surtout ressentir auprès des utilisateurs qui ont à établir le plan de traitement des patients. Nos accélérations font passer leur temps d'attente des

résultats de significatifs (dans les dizaines de secondes) à quasi instantané (en dessous de 3 secondes) pour une approche classique à quatre faisceaux. Ceci permet d'augmenter le nombre de plans de traitement qui peuvent être élaborés à l'intérieur d'un temps donné. Ce projet aura aussi des impacts pour les techniques d'optimisation automatisée de plans de traitement, telle l'IMRT où la dose doit être calculée plusieurs fois à l'intérieur d'un processus d'optimisation automatisée. Ces techniques font intervenir une dose cible à la tumeur et une dose maximale aux organes à risque et laisse l'engin d'optimisation positionner les faisceaux pour répondre aux contraintes. Il est donc crucial que l'engin de calcul de dose soit rapide pour pouvoir étudier un plus grand nombre de solutions possibles dans un temps donné et donc converger plus rapidement vers une solution conforme aux contraintes.

Une limite importante de la carte graphique, du moins pour la 8800GT que nous avons utilisée, est la taille de sa mémoire vidéo. Le fait que seulement 512Mo de mémoire soit disponible limite la taille des données en entrée puisqu'il n'y a pas de mécanisme de pagination comme pour la mémoire principale d'un système. Par exemple, un maillage de 256 éléments par dimension, utilise, si les éléments sont à virgule flottante simple précision, 64Mo. Nous avons besoin d'au moins un tableau en entrée (mais probablement plus) et au moins un en sortie ce qui fait que la capacité de la carte est vite atteinte. Sans compter que la totalité de l'espace n'est pas disponible si cette carte est aussi utilisée pour l'affichage à l'écran.

4.4 Travaux futurs

La possibilité a été évoqué rapidement plus haut dans le texte d'utiliser une grappe de calcul munie de cartes graphiques comme nœud de calcul primaire. Cette possibilité offre des perspectives intéressantes. Nous y trouverions donc le meilleur des deux mondes, mais bien sûr aussi le pire des deux mondes. Nous devrions gérer toute l'infrastructure de communication nécessaire aux grappes de calcul mais nous en tirions une puissance de calcul importante avec un plus petit nombre de nœuds de calcul.

Une autre ouverture pour ce projet consiste à le porter vers la spécification OpenCL, qui vise à unifier le calcul parallèle haute performance en se libérant des vendeurs de matériel pour définir une spécification indépendante, similaire à ce qu'est devenue la norme OpenGL. Un port vers cette spécification permettrait d'utiliser une carte graphique d'AMD (anciennement ATI) qui offre aujourd'hui une puissance de calcul plus élevée mais une suite de développement pour calcul scientifique inférieure à celle offerte par NVIDIA.

CONCLUSION

Ce projet portait sur l'accélération de calculs en simulation de plans de traitement de radiothérapie. Le but a été atteint par l'utilisation d'une carte graphique comme matériel principal de calcul. La nature parallèle du problème s'y porte naturellement et les résultats obtenus sont excellents. En effet, nous trouvons des accélérations de 10x à 560x, à travers diverses étapes de calcul et algorithmes pour effectuer ces étapes. Notre projet a utilisé la suite de développement CUDA de NVIDIA et est donc limité à s'exécuter sur les cartes graphiques de cette même compagnie. Depuis le début de ce projet, la spécification OpenCL, qui vise à unifier le calcul scientifique parallèle sur cartes graphiques (et autres périphériques) a été officialisée par le groupe Khronos. Lorsque les pilotes des différentes compagnies de cartes graphiques (principalement AMD et NVIDIA) supporteront cette spécification, il pourrait être intéressant de traduire notre projet pour utiliser OpenCL, ce qui devrait être trivial vu le très haut niveau de similarité entre CUDA et OpenCL.

BIBLIOGRAPHIE

Anhesjö, A. (1989). Collapsed cone convolution of radiant energy for photon dose calculation in heteregenous media. *Medical Physics , 16* (4), 577-592.

Bracewell, N. R. (2009). *Fourier analysis and imaging.* Springer.

Christiaens, M., Sutter, B., Bosschere, K., Campenhout, J., & Lemahieu, I. (1998). A fast algorithm to calculate the exact radiological path through a pixel or voxel space. *Journal of computing and information technology , 6*, 781-790.

Hissoiny, S. (2009). Fast convolution-superposition dose calculation on graphics hardware. *Medical Physics , 32* (6), 1998-2005.

Hoban, P. W. (1995). Accounting for the variation in collision kerma-to-terma ratio in polyenergetic photon beam convolution. *Medical Physics , 22* (12), 2035-2044.

Hoban, P., Murray, D., & Round, W. (1994). Photon beam convolution using polyenergetic energy deposition kernels. *Phys. Med. Biol , 39* (4), 669-685.

Hubbell, J. H., & Seltzer, S. (1996). *Tables of X-Ray Mass Attenuation Coefficients and Mass Energy-Absorption Coefficients.* NIST. Gaithersburg: National Institute of Standards and Technology.

ICRU. (1980). *Radiation quantities and units.* Bethesda.

Jaffray, D. A., Battista, J. J., Fenster, A., & P., M. (1993). X-ray sources of medical linear accelerators: Focal and extra-focal radiation. *Medical Physics , 20* (5), 1417-1427.

Jelen, U. (2007). *Development and verification of a high-precision dose calculation algorithm for IMRT treatment planning.* Cracow: PhD Thesis, AGH University of Science and Technology.

Johns, H. E., & Cunningham, J. R. (1983). *The physics of radiology.* Springfield: Charles C Thomas.

Lacroute, P. G. (1995). *FAST VOLUME RENDERING USING A SHEAR-WARP FACTORIZATION OF THE VIEWING TRANSFORMATION.* Stanford: PhD Thesis, Standford University.

Liu, H. H., Mackie, R. T., & McCullough, E. C. (1997). Correcting kernel tilting and hardening in convolution/superposition dose calculations for clinical divergent and polychromatic photon beams. *Medical Physics , 24* (11), 1729-1741.

Mackie, T. R., Bielajew, A. F., Rogers, D. W., & J, B. J. (1988). Generation of photon energy distribution kernels using the EGS Monte Carlo Code. *Physics in Medicine and Biology , 33* (1), 1-20.

Metcalfe, P. E., Hoban, W. P., Murray, C. D., & Round, ,. W. (1990). Beam hardening of 10 MV radiotherapy X-rays: analysis using a convolution/superposition method. *Physics medical biology , 35* (11), 1533-1549.

Metropolis, N. (1987). THE BEGINNING of the Monte Carlo Method. *Los Alamos Science , 15* (Special Issue), p. 125.

NVIDIA. (2008, 07 06). NVIDIA Cuda : Compute Unified Device Architecture.

Nyland, L., Harris, m., & Prins, J. Fast N-Body simulation with CUDA. Dans *GPU Gems III* (pp. 676-695). NVIDIA.

Papanikolaou, N., Battista, J. J., Boyer, A. L., Kappas, C., & Mackie, T. R. (2004). *Tissue inhomogeneity corrections for megavoltage photon beams.* AAPM.

Papanikolaou, N., Mackie, T. R., Meger-Wells, C., Gehring, M., & Reckwerdt, P. (1993). Investigation of the convolution method for polyenergetic spectra. *Medical Physics , 20* (5), 1327-1336.

Sharpe, M. B., Jaffray, D. A., Battista, J. J., & Munro, P. (1995). Extrafocal radiation: A unified approach to the prediction of beam penumbra and output factors for megavoltage x-ray beams. *Medical Physics , 22* (12), 2065-2074.

Sharpe, M., & Battista, J. (1993). Dose calculation using convolution/superposition principes: the orientation of dose spread kernels in divergent x-ray beams. *Medical Physics , 20* (6), 1685-1694.

Siddon, R. L. (1985). Fast calculation of the exact radiological path for a three-dimensional CT array. *Medical Physics , 12* (2), 252-255.

Simpkin, D. J. (1990). EGS4 Monte Carlo determination of the beta dose kernel in water. *Medical physics , 17* (2), 179-186.

Sjögren, R., & Karlsson, M. (1996). Electron contamination in clinical high energy photon beams. *Medical Physics , 23* (11), 1873-1881.

Starkschall, S. P. (2000). Beam-commissioning methodology for a three dimensional convolution/superposition photon dose algorithm. *Journal of applied clinical medical physics , 1* (1), 8-27.

ANNEXES

Annexe 1 : Table de conversion de CT (unités Hounsfield) vers densité

```
short ct = ct_data[idx];
               ct&=IMAGE_BITS;
               float d;

               ct-=1024;
               if (ct < -1000) d = 0.0f; //Air [0,0) [-1024, -1000)
               else if (ct < 0) d = 1.0f + 0.00100f*(float)ct; //Lung[0, 1) [-1000, 0)
               else if (ct < 1200) d = 1.000f + 0.000500f*(float)(ct - 0); //Tissue[1,
1.6) [0, 1200)
               else if (ct < 1700) d = 1.600f + 0.000400f*(float)(ct - 1200); //Bone[1.6,
1.8) [1200, 1700)
               else if (ct < 2000) d = 1.800f + 0.00300f*(float)(ct - 1700); //Metal[1.8,
2.7) [1700, 2000)
               else d = 7.0; // [2000, 4096)
               density_data[idx] = d;
```

Annexe 2: Vecteur de distances latérales

```
-10.00
-8.00
-6.00
-5.00
-4.00
-3.00
-2.00
-1.50
-1.00
-0.80
-0.60
-0.50
-0.40
-0.30
-0.20
-0.15
-0.10
-0.05
0.05
0.10
0.15
0.20
0.30
0.40
0.50
0.60
0.80
1.00
1.50
2.00
3.00
4.00
5.00
6.00
8.00
10.00
```

Annexe 3 : Vecteur de distances en profondeur

```
-0.30
-0.20
-0.10
-0.05
0.05
0.10
0.15
0.20
0.30
0.40
0.50
0.60
0.80
1.00
1.50
2.00
3.00
4.00
5.00
6.00
8.00
10.00
15.00
20.00
30.00
40.00
50.00
60.00
```