Adéquation Algorithme Architecture pour la reconstruction 3D en imagerie médicale TEP

Nicolas Gac

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'Institut polytechnique de Grenoble

Spécialité : *« Signal, Image, Parole, Télécom »*

préparée au laboratoire Gipsa-lab de Grenoble

dans le cadre de **l'Ecole Doctorale** *« Electronique, Electrotechnique, Automatique, Traitement du Signal »*

présentée et soutenue publiquement

par

Nicolas GAC

le 17 Juillet 2008

Adéquation Algorithme Architecture pour la reconstruction 3D en imagerie médicale TEP

DIRECTEUR DE THESE : Michel DESVIGNES
CO-ENCADRANT DE THESE : Stéphane MANCINI

JURY

M.	Laurent DESBAT	, Président
M.	Didier DEMIGNY	, Rapporteur
M.	Yves MATHIEU	, Rapporteur
M.	Michel DESVIGNES	, Directeur de thèse
M.	Stéphane MANCINI	, Co-encadrant
M.	Bertrand GRANADO	, Examinateur

Résumé

L'amélioration constante de la résolution dynamique et temporelle des scanners et des méthodes de reconstruction en imagerie médicale, s'accompagne d'un besoin croissant en puissance de calcul. Les accélérations logicielles, algorithmiques et matérielles sont ainsi appelées à réduire le fossé technologique existant entre les systèmes d'acquisition et ceux de reconstruction.

Dans ce contexte, une architecture matérielle de rétroprojection 3D en Tomographie à Emission de Positons (TEP) est proposée. Afin de lever le verrou technologique constitué par la forte latence des mémoires externes de type SDRAM, la meilleure Adéquation Algorithme Architecture a été recherchée. Cette architecture a été implémentée sur un SoPC (System on Programmable Chip) et ses performances comparées à celles d'un PC, d'un serveur de calcul et d'une carte graphique. Associée à un module matériel de projection 3D, cette architecture permet de définir une paire matérielle de projection/rétroprojection et de constituer ainsi un système de reconstruction complet.

Mots Clefs : imagerie médicale, Tomographie à Emission de Positons, TEP, problème inverse, reconstruction itérative, rétroprojection , Adéquation Algorithme Architecture, accélération matérielle, parallélisme, FPGA, SoPC, GPU, prefetching, cache mémoire

Abstract

The medical imaging improvements in spatial and temporal resolutions involve increasing needs in computational power. The software, algorithmic and hardware accelerations are called to reduce the current technologic gap existing between acquisition and reconstruction systems.

In this context, a hardware architecture of 3D backprojection for Positron Emission Tomography (PET) is proposed. Thanks to an Algorithm Architecture Adequacy, our architecture overcomes successfully the limitations due to the latency of the external memory SDRAM like. It has been implemented on a SoPC (System on Programmable Chip) and its performances have been compared to the ones achieved by a desktop PC, a workstation and a graphic board. With an additional forward projector IP, a hardware pair of projection/backprojection can be build which allows the design of a complete reconstruction system.

Keywords : medical imaging, Positron Emission Tomography, PET, inverse problem, iterative reconstruction, backprojection, architecture design, computer architecture, hardware acceleration, parallelism, FPGA, SoPC, GPU, prefetching, cache memory

Remerciements

Je souhaite adresser mes premiers remerciements à Michel Desvignes et Stéphane Mancini pour m'avoir apporter leur confiance en me proposant ce sujet de thèse et pour la qualité de leur encadrement tout au long de ces quatre années. Je remercie Stéphane pour sa disponibilité quasi quotidienne lorsque pour des armées de bugs envahissaient la carte. Je remercie Michel pour ses conseils et sa disponibilité lors des réunions régulières d'avancement de thèse. J'adresse mes sincères remerciements à Didier Demigny et Yves Mathieu, rapporteurs de cette thèse. Merci à Laurent Desbat d'avoir accepter de présider le jury de thèse, ainsi qu'à Bertrand Granado pour avoir participer à ce jury.

Je remercie Dominique Houzet qui peu de temps après son arrivée au laboratoire a participé avec enthousiasme à ces travaux de thèse. L'étude comparative avec les GPUs n'aurait pas été possible sans lui. Merci à Anthonin Reilhac du laboratoire CERMEP pour son aide précieuse sur les données TEP. Je tiens également à remercier Florian Deboissieu et Emmanuel Gouy pour leur implication durant leur stage au laboratoire Gipsa-lab.

Je remercie Jean-Marc Chassery pour m'avoir accueilli au sein du Laboratoire Gipsa-lab ainsi que Christian Jutten pour son accueil au département Image et Signal (DIS). Je remercie également Inbar Fijalkov de m'avoir accueilli au laboratoire ETIS de l'université de Cergy Pontoise en tant qu'ATER durant ma quatrième année de thèse. Merci à François Verdier et Amine Benkhelifa de l'équipe Architecture du laboratoire ETIS pour leur soutien.

Je tiens à remercier également tous les cocos, colocs ou cobureaux, confidents de thèse : Ben (parrain quoi !), Ana (un petit coup de pub pour la boutique : les muses, 3 rue Pierre Duclot à Grenoble), Adrian (le roumain le plus grand que je connaisse ... bon j'en connais qu'un), Brice (bio dans ses baskets), Yannick (un autre converti au bio), Tomasz (tombé dans la potion informatique quand il était petit). J'adresse également une pensée aux jolies blondes de passage au labo (Andrea et Hafrun), un coucou à mes amis de promo de master (Jasmina, Nourredine et Grégoire) et un chaleureux merci à tous ceux qui ont peuplé et animé Viallet et en ont fait un lieu où je me suis bien plu pendant ces quatre années : Marino, Nathalie, Rosie, Barth, Khaled, Vincent, Cédric[2], Alex, Manu, Duc, Zakia, Corentin, Guillermo et tous ceux que j'oublie.

Enfin, je remercie ma famille pour son soutien et ses encouragements durant ces ~~quelques années d'études~~ (11 ans, tu veux dire petit c..!). Merci soeurette pour les relectures !

Table des matières

Table des figures

Liste des tableaux

Glossaire

3D-RP 3D ReProjection
A^3 Adéquation Algorithme Architecture
ART Algebraic Reconstruction Technique
BPF Back Projection Filtered
CT Computed Tomography
CUDA Compute Unified Device Architecture
EM Expectation Maximization
FOV Field Of View
FBP Filtered Back Projection
FDG FluoroDésoxyGlucose marqué au fluor-18
FDK Feldkamp, Davis and Kress
FORE FOurier REbinning
HRRT High Resolution Research Tomograph
IP Intellectual Property (désigne de manière plus général un bloc matériel)
LOR Line Of Response
mrd maximum ring difference
MAP Maximum A Posteriori
ML Maximum Likelihood
MPI Message Passing Interface
LMEM List-Mode Expectation Maximization
LSV (modèle) Linéaire et Spatialement Variant
OS-EM Ordered Subsets Expectation Maximization
OSMAPOSL Algorithme OSEM de STIR
OSL One-Step Late
P/R paire de Projection/Rétroprojection
PET Positron Emission Tomography
$\mathbf{P_{Lr}}$ Projection à Lancer de rayon

PRSB Pic de Rapport Signal sur Bruit

REQM Racine de l'Erreur Quadratique Moyenne

RSB Rapport Signal sur Bruit

$\mathbf{R_{VIB}}$ Rétroprojection Voxel driven avec Interpolation Bi-linéaire

SPECT Single Positron Emission Computed Tomography

SPMD Single Program Multiple Data

TEP Tomographie à Emission de Positons

TF Transformée de Fourier

UT Unité de Traitement

VIB (Rétroprojecteur) Voxel-driven avec Interpolation Bilinéaire

Introduction

Les techniques d'imagerie médicale comme la tomographie X (CT), l'imagerie par Résonance Magnétique (IRM) ou la Tomographie par Emission de Positrons (TEP) offrent un accès immédiat et fiable à des informations anatomiques et métaboliques jusqu'alors invisibles au diagnostic clinique. Grâce aux efforts conjointement menés pour perfectionner les systèmes d'acquisition et les méthodes de reconstruction, quatre progrès essentiels ont été apportés au domaine de l'imagerie médicale. Premièrement, l'amélioration de la résolution spatiale n'a pas cessé depuis l'apparition de l'imagerie médicale. Elle est la double conséquence de l'augmentation constante du nombre de détecteurs des scanners et de l'élaboration de méthodes de reconstruction entièrement 3D (*fully 3D*) qui utilisent de la manière la plus pertinente possible l'ensemble des données acquises. Deuxièmement, des progrès en résolution temporelle sont constamment obtenus. Deux raisons concourent à cela : l'augmentation de la vitesse d'acquisition (*frames* par seconde) et l'élaboration de méthodes 4D (trois dimensions spatiales + une dimension temporelle). Troisièmement, la qualité de reconstruction a été améliorée grâce à l'intégration de modèles physiques d'acquisition dans les méthodes itératives de reconstruction. Enfin, les scanners à double modalité (PET/CT) ont permis via la fusion d'image de faire apparaître à la fois les informations anatomiques et celles métaboliques.

Outre les efforts de perfectionnement des systèmes d'acquisition et des méthodes de reconstruction, un troisième effort doit être mené afin de pouvoir bénéficier en pratique des progrès de l'imagerie médicale. Cet effort vise à concevoir un système de reconstruction rapide et précis afin de satisfaire les contraintes temporelles, plus moins importantes selon les applications. Pour un simple examen en routine clinique, la reconstruction ne doit pas excéder quelques minutes afin de s'assurer que l'acquisition se soit bien déroulée. Pour l'imagerie dédiée à l'intervention chirurgicale, une reconstruction en temps réel est requise afin de fournir au chirurgien une séquence d'images lui permettant de visualiser quasi immédiatement ses gestes chirurgicaux. Or bien que la puissance de calcul des processeurs classiques augmente constamment et malgré les importants efforts pour optimiser les algorithmes, les solutions logicielles standards sont actuellement insuffisantes pour satisfaire ces contraintes temporelles lorsque les méthodes itératives de reconstruction les plus performantes sont utilisées. Ainsi, plusieurs heures de reconstruction sur un PC sont actuellement nécessaires afin de bénéficier pleinement des résolutions spatiales et temporelles et de la qualité de reconstruction offertes par les scanners à haute définition. Les contraintes temporelles empêchent donc la diffusion à plus large échelle des progrès de l'imagerie médicale dans les cliniques et les laboratoires de recherche. L'accélération matérielle est ainsi appelée à réduire le fossé technologique existant entre les systèmes d'acquisition et ceux de reconstruction.

Problématique de l'accélération matérielle

L'accélération matérielle s'appuie sur deux principes : la conception sur puce d'unités de calcul performantes et la constitution de réseaux d'interconnexion de ces puces. Nous focaliserons notre propos sur le premier point.

D'une part, les architectures des unités de calcul classiques de type CPUs (*Central Processing Unit*), reposent sur les mêmes principes utilisées pour les premières machines

Von Neuman. Elles sont conçues de manière à exécuter de la manière la plus efficace possible des programmes ou logiciels (*software*). S'appuyant sur une croissance forte et régulière de la complexité et des fréquences de fonctionnement des circuits intégrés, leur performance n'a cessé de croître. Cependant des limites physiques sont apparues depuis quelques années, et il devient de plus en plus difficile d'augmenter la fréquence de fonctionnement. A présent le gain en performances des CPUs, est obtenu grâce au parallélisme des calculs avec l'intégration de plusieurs coeurs sur une même puce. Des processeurs moins généralistes que les CPUs, dédiés à des familles d'applications tels que les jeux vidéos ou le calcul scientifique, offrent une puissance de calcul plus importante. Leur architecture intègre un plus grand nombre d'unités de calculs et utilisent des unités de traitement vectoriel des données. C'est sur ces principes qu'a été conçu le processeur Cell et ce vers quoi tendent les processeurs graphiques ou GPUs (*Graphic Procesing Unit*) depuis peu avec le développement des GPGPUs (*General Purpose GPU*).

D'autre part, les circuits intégrés ASICs sont une alternative purement matérielle (*hardware*) qui a pour but la conception d'une architecture spécifique à une application précise. Cette solution offre des performances supérieures aux processeurs classiques. Toutefois son coût de fabrication ne la rend viable seulement à partir de gros volumes de production ($>$ 100 000 puces). Par ailleurs, entre les approches exclusivement *software* (processeur) et *hardware* (ASIC), le marché a vu émerger des dispositifs mixtes, les systèmes sur puce programmable (SoPC : System on Programmable Chip). Ils permettent un partitionnement adéquat entre les aspects matériels et logiciels, tout en offrant une efficacité, une flexibilité et une reconfigurabilité totale. Sur ces puces, sont intégrés de la logique programmable, de la mémoire et des processeurs classiques.

Une problématique commune aux approches *software* (CPU), *hardware* (ASIC) ou mixte (GPU et SoPC) est le goulot d'étranglement que constitue l'accès à la mémoire externe à la puce. Sans stratégie mémoire adaptée, les unités de calcul sont inactives durant tout le temps de réponse de la mémoire externe. Afin de s'affranchir de ce "mur mémoire", des caches mémoires ou des techniques "ad hoc" de préchargement de données sont ainsi utilisés.

Contexte et objectifs de la thèse

Dans cette étude, nous nous focaliserons sur l'accélération de la reconstruction en Tomographie à Emission de Positons (TEP) et plus spécifiquement sur l'accélération de la rétroprojection 3D. Une architecture matérielle de rétroprojection 3D implémentée sur un système SoPC est proposée. Cette architecture a pour objectif de tirer profit au maximum des ressources matérielles offertes par un tel système. L'Adéquation entre l'Algorithme et l'Architecture (A^3) de rétroprojection 3D sera activement recherchée. Dans cette optique, une stratégie mémoire visant à masquer le temps d'accès à la mémoire externe de type SDRAM, a été mise en place. La forte latence mémoire de ces mémoires constitue un verrou technologique à l'accélération de la rétroprojection sur les cartes SoPC. La matrice FPGA, étant vierge de caches ou de modules de préchargement mémoire, l'utilisation d'un cache 3D Prédictif et Adaptatif (cache 3D-AP) est proposée.

Ces travaux de thèse s'inscrivent dans le cadre plus général du projet ArchiTEP. Ce dernier a pour but la conception d'une architecture générique paramétrable et reconfigurable pour la reconstruction TEP 3D via une démarche d'Adéquation Algorithme Architecture. L'objectif final de ce projet est la validation de la méthodologie A^3 mise en place avec la conception d'un prototype sur plateforme SoPC. Ce projet est mené par le laboratoire Gipsa-lab en collaboration avec trois laboratoires de la région Rhône-Alpes : le LPSC, le TIMC et le CERMEP. Ce projet est financé par l'Agence National de Recherche (ANR STIC-Santé), la région Rhône-Alpes (Fonds d'Incitation au Transfert de Technologie) et l'INPG (Bonus Qualité Recherche).

Organisation du manuscrit

Le manuscrit comporte trois parties, chacune constituée de deux chapitres :

I Etat de l'art de la reconstruction tomographique. Le chapitre 1 porte sur les algorithmes de reconstruction en TEP. Après une brève présentation de l'imagerie TEP, ce chapitre exposera sous sa forme mathématique le problème inverse, les méthodes analytiques et itératives de reconstruction et l'échantillonnage des données couramment utilisée. Dans le chapitre 2, nous nous intéresserons à l'accélération de la reconstruction. Ce chapitre s'attachera à passer en revue les différents types d'accélération : logicielle, algorithmique et matérielle. L'accent portera principalement sur la présentation des travaux d'accélération matérielle sur machines parallèles (machines multiprocesseurs, processeurs spécifiques et architectures matérielles de type ASIC/FPGA).

II Architecture matérielle pour la rétroprojection 3D en TEP. Le chapitre 3 présentera la démarche d'Adéquation Algorithme Architecture mise en place lors de la conception de l'architecture de rétroprojection 3D. Le verrou technologique constitué par l'accès à la mémoire externe sera mis en avant. Les choix algorithmiques, architecturaux et de structure de données qui ont guidé la conception de l'architecture 3P (architecture en Pipeline, Préchargement mémoire via un cache 3D et Parallélisation des unités de calcul) seront argumentés. Le chapitre 4 présentera les performances de cette architecture, obtenue après implémentation sur une carte SoPC. La qualité et l'efficacité de reconstruction, ainsi que la pertinence de notre stratégie mémoire seront évaluées.

III Etude comparative et système complet de reconstruction. Le chapitre 5 aura pour objectif la comparaison des performances offertes par notre architecture face à d'autres solutions technologiques. Deux implémentations logicielles optimisées seront ainsi présentés, l'une sur CPU et l'autre sur GPU. Nous confronterons les performances des stratégies mémoire des trois implémentations (CPU, GPU et FPGA). Enfin, le chapitre 6 présentera la paire matérielle de projection et rétroprojection, clef de voûte du système complet de reconstruction TEP du projet ArchiTEP. Les déclinaisons de l'architecture globale selon les différentes méthodes de reconstruction (analytique ou itérative) seront exposées et évaluées en terme de qualité et d'efficacité de reconstruction. Le facteur d'accélération obtenu en comparaison d'une solution logicielle standard sera également évalué.

Première partie

Etat de l'art de la reconstruction tomographique

Notations pour la partie I

Paramètre	**Expression continue**	**Expression discrète**	**Pas d'échantillonnage**
Capteur	$\vec{LOR}$	i	
Paramétrisation TEP par couples d'anneaux : $LOR(A_p, A_q, P_i, Q_j)$			
Premier anneau p	A_p	`p`	
Deuxième anneau q	A_q	`q`	
Capteur i de l'anneau p	P_i	`i`	
Capteurj de l'anneau q	Q_j	`j`	
Paramétrisation TEP par couples d'anneaux : $LOR(\phi, \rho, \lambda, \Delta)$			
Angle azimutal	ϕ	`phi`	δ_ϕ
Coordonnée radiale	ρ	`rho`	δ_ρ
Hauteur moyenne des anneaux	λ	`lambda`	$\delta_{z\lambda}$
Différence entre anneaux	Δ	`Delta`	δ_Δ
Faisceau de LOR d'angle ϕ	$F_{LOR}(\Delta, \lambda, \phi)$		
Paramétrisation TEP par plans de projection : $LOR(\psi, \theta, u_\parallel, v_\parallel)$			
Angle azimutal	ψ		
Angle polaire	θ		
Abscisse sur le plan	$u_\parallel$		
Ordonnée sur le plan	$v_\parallel$		
Paramétrisation CT par plans de projection : $X(\alpha, u_\vee, v_\vee)$			
Position de la source	α		
Abscisse sur le plan	$u_\vee$		
Ordonnée sur le plan	$v_\vee$		
Paramétrisation TEP par point et vecteur directeur : $LOR(A, \vec{V})$			
Point sur la LOR	A		
Vecteur directeur d'une LOR	$\vec{V}$		

Paramètres	**Continue**	**Discret**	**Pas d'échantillonnage**
Elément de volume	$\vec{r}$	j	
Coordonnée x	x	`xn`	δ_{xy}
Coordonnée y	y	`yn`	δ_{xy}
Coordonnée z	z	`zn`	$\delta_{z\lambda}$
Fonctions			
Mesure		y_i	
Densité d'émission	$f(\vec{r})$	f_j	
Densité instantanée d'émission	$\eta(\vec{r}, t)$		
Fonction de base du volume	$b_j\vec{r}$		
Projection	$p(\Delta, \lambda, \phi, \rho)$	P	
Rétroprojection	$rp(x, y, z)$	RP	
Constantes			
Matrice système		H	
Largeur du champ de vue		L_{FOV}	
Hauteur du champ de vue		H_{FOV}	
Nombre d'anneaux du scanner		N_a	
Ecart entre anneaux		d_a	
Nombre de détecteurs par anneau		N_d	
Rayon des anneaux du scanner		R_a	
Hauteur du scanner		H_{scan}	
Angle axial maximal du scanner		Θ_{scan}	
Angle axial maximal des données complètes		θ_C	

Chapitre 1

La reconstruction en Tomographie à Emission de Positons

La tomographie, des racines grec tomos (coupe) et graphia (écrire) est une technique permettant une cartographie d'un paramètre interne à un objet selon un ou plusieurs plans de coupes à partir de mesures externes et de calculs. D'un point de vue mathématique, la tomographie définit d'une part un modèle direct, celui des phénomènes physiques mesurés et d'autre part un modèle inverse ou reconstruction, celui des calculs nécessaires à la restitution de l'objet à partir des mesures. Cette technique s'applique à de nombreux domaines avec la reconstruction de divers paramètres internes d'objets tels que :

- la structure du corps humain en imagerie médicale à l'aide de scanners,
- la cartographie des champs magnétiques de surface des étoiles en astrophysique à l'aide de l'imagerie Zeeman-Dopler,
- la structure géologique des couches terrestres en géologie à l'aide de l'enregistrement des ondes sismiques ou des radars SAR (Synthetic Aperture Radar),
- la structure des cellules et des micro-organismes en biologie à l'aide de microscopes électroniques,
- la structure de pièces industrielles (réacteurs, pipelines, blocs moteurs...) pour le contrôle non destructif en tomographie industrielle.

Dans ce chapitre, nous nous focaliserons sur l'étude de la Tomographie à Emission de Positons (TEP). Cette technique d'imagerie médicale nucléaire permet de visualiser certains aspects du métabolisme des organismes vivants. Elle est notamment utilisée pour la détection de cancers.

Nous présenterons tout d'abord le principe, les applications et les scanners de cette technique d'imagerie médicale. Puis après avoir exposé sous sa forme mathématique le problème inverse de la tomographie TEP, nous présenterons les deux familles d'algorithmes de reconstruction, les algorithmes analytiques et itératifs. Ensuite, nous donnerons l'expression discrète de ces algorithmes, utilisée en pratique, en présentant l'échantillonnage des données couramment employé en TEP. Enfin nous comparerons la rétroprojection utilisée en tomographie CT avec celle utilisée en tomographie TEP.

1 Imagerie médicale TEP

Dans cette section, nous décrivons le principe de cette technique d'imagerie nucléaire fonctionnelle, avant de présenter les applications de la TEP en routine clinique et en recherche clinique mais également de la μTEP sur petit animal en recherche pré-clinique. Nous présentons enfin plus en détail le système d'acquisition des scanners TEP.

1.1 Imagerie anatomique et imagerie fonctionnelle

En imagerie médicale, on distingue l'imagerie structurelle dont les examens permettent d'obtenir des informations sur l'anatomie des organes, de l'imagerie fonctionnelle qui s'intéresse au métabolisme c'est-à-dire à l'ensemble des transformations moléculaires et des transferts d'énergie qui se déroulent dans l'organisme. Parmi les techniques d'imagerie structurelle, nous pouvons citer les imageries utilisant les rayons X comme la radiographie et la tomodensitométrie (scanner X), l'Imagerie à Résonance Magnétique nucléaire anatomique (IRM_a) et l'échographie. Parmi les techniques d'imageries fonctionnelles, les plus répandues sont l'IRM fonctionnelle (IRM_f) qui évalue le niveau d'activité des zones du cerveau en mesurant les concentrations de désoxy-hémoglobine (hémoglobine sans oxygène) directement liées à l'activité cérébrale par effet BOLD, la magnéto-encéphalographie (MEG) et les techniques d'imageries nucléaires. Des scanners combinent imagerie anatomique et imagerie fonctionnelle comme les scanners TEP/X. Ils sont apparus sur le marché de l'imagerie médicale vers la fin des années 90 et sont à présent adoptés par la plupart des services cliniques. Ces scanners permettent de superposer l'information métabolique de faible résolution obtenue à l'aide d'un scanner TEP, avec l'anatomie du patient de très bonne résolution obtenue à l'aide d'un scanner X.

1.2 Techniques d'imageries nucléaires

Les techniques d'imageries nucléaires sont basées sur l'injection par intraveineuse d'un traceur constitué d'un vecteur moléculaire et d'un isotope radioactif. Ce traceur va se concentrer dans les régions d'intenses activités métaboliques et émettre des rayons γ. La détection de ces rayons γ permettra après reconstruction tomographique de quantifier in vivo et de manière non invasive des processus biochimiques variés à une échelle nano-moléculaire. Si pour la Tomographie à Emission MonoPhotonique (TEMP) l'émission des rayons γ provient directement de la désintégration des traceurs radioactifs, ce n'est pas le cas pour la Tomographie à Emission de Positons (TEP). En TEP, les traceurs radioactifs émettent lors leur désintégration un rayonnement $\beta+$. Le positon émis, après un parcours de quelques millimètres va perdre toute son énergie cinétique. Puis lors de sa rencontre avec un électron va s'annihiler avec ce dernier et alors émettre deux rayons γ colinéaires d'une énergie de 511 keV chacun (voir figure 1.1). La synthèse des traceurs radioactifs se fait à l'aide de cyclotrons médicaux. Afin de distribuer les traceurs radioactifs dont la durée de demi-vie permette un transport vers les centres hospitaliers proche du cyclotron, cette infrastructure lourde est répartie géographiquement sur l'ensemble du territoire d'un pays (il en existe une dizaine en France).

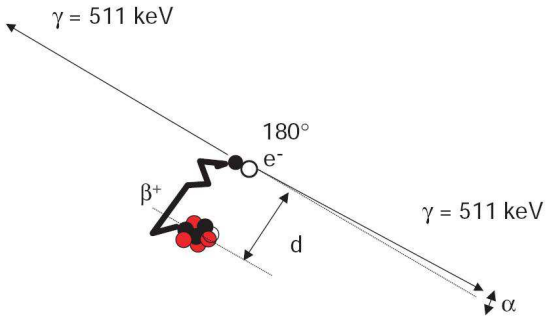


FIG. 1.1 – Désintégration $\beta+$ suivi de la réaction d'annihilation entre le positon émis et un électron voisin qui provoquera l'émission de deux photons colinéaires d'une énergie de 511 keV chacun.

1.3 Applications de l'imagerie TEP

1.3.1 Applications en routine clinique

Depuis les années 70, l'imagerie TEP s'est fortement développée notamment en *oncologie* avec la découverte (ou plus exactement la redécouverte) d'un traceur radioactif, le ^{18}flurodésoxyglucose ($^{18}F - FDG$). Cette molécule, analogue du glucose, permet le diagnostic de cancers. Après injection, ce traceur est consommé et piégé par les cellules ayant un fort métabolisme comme les cellules cancéreuses. Les tissus cancéreux sont alors repérés par une concentration excessive de $^{18}F-FDG$ (voir figure 1.2). Un des atouts du FDG est sa durée de demi-vie relativement longue (110 mn). Elle autorise l'utilisation de FDG dans les cliniques ne possédant pas de cyclotron mais situées suffisamment proches d'un centre de synthèse du FDG (1 à 2 heures de temps de transport). Le coût élevé d'un examen au FDG (1200 euros par patient) s'explique par le coût du consommable (entre 400 et 500 euros par dose), des scanners TEP (environ 3 millions d'euros) et des frais de fonctionnement d'un service clinique nucléaire (environ 1 million d'euros par an). Dans la cadre du plan cancer de 2003-2007 et afin de rattraper son retard dans ce domaine, la France s'est fixée comme objectif d'avoir un scanner TEP pour 800 000 habitants. En 2007, l'objectif est proche d'être atteint avec un peu plus de 60 machines TEP sur le territoire.

La *cardiologie* est un autre domaine d'application de l'imagerie TEP. La recherche de viabilité du muscle cardiaque après infarctus permettant un meilleur diagnostic avant toute opération chirurgicale, est un examen clinique en voie de développement. Deux examens sont effectués : un au FDG qui permet d'avoir une image de son métabolisme et un autre au $^{13}NH_3$ ou au ^{82}Rb qui permet de visualiser la perfusion du myocarde. Puisque les tissus proches de la nécrose par ischémie (absence de perfusion) présentent durant quelques heures un hypermétabolisme, ils restent visibles à l'examen FDG et ont disparu de l'image de perfusion comme illustré sur la figure 1.2. Le Rubidium-82 pouvant être produit à partir du strontium-82 d'une durée de demi-vie de 25 jours à l'aide d'un générateur au coût beaucoup plus faible qu'un cyclotron, est promis à un bel avenir.

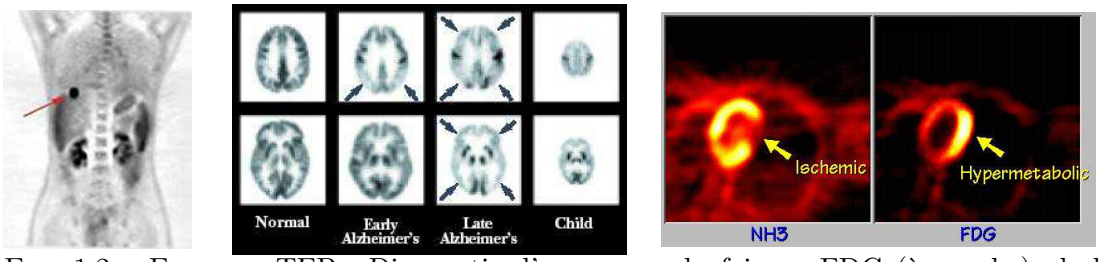


FIG. 1.2 – Examens TEP : Diagnostic d'un cancer du foie au FDG (à gauche), de la maladie d'Alzeihmer au FDG (au centre) et d'anomalie cardiaque au FDG et NH3 (à droite)

1.3.2 Applications en recherche clinique

Les laboratoires de recherche clinique possédant un cyclotron, peuvent synthétiser des traceurs composés d'isotopes à faible durée de vie comme le ^{15}O, le ^{13}N ou le ^{11}C. Ils ont respectivement une demi-vie de 2, 10 et 20 minutes. Ces isotopes constitutifs de l'ensemble des composés biologiques, permettent de réaliser des marquages efficaces pour un plus grand nombre de molécules. La recherche en neurologie utilise l'imagerie TEP comme outil de diagnostic pour les maladies neurologiques. Ainsi, le $^{18}F-DOPA$ permet de diagnostiquer la maladie de Parkinson en mettant en évidence les cellules ne produisant plus de dopamine. La synthèse d'un autre marqueur isotopique, le PIB, par des chercheurs de l'université de Pittsburgh aux Etats-Unis et d'Uppsala en Suède pourraient bien également aboutir à la mise au point d'un diagnostic précoce de la maladie d'Alzeihmer ([**?**]). GE Healthcare, fabricant d'appareils TEP, a obtenu les droits de distribution et de commercialisation des produits dérivés de cette molécule associée à un atome radioactif de ^{18}F ([**?**]).

1.3.3 Applications en recherche pré-clinique

Des scanners TEP aux dimensions adaptées au petit animal (souris ou rat) sont utilisés comme outil d'imagerie in vivo pour la recherche pré-clinique. Ce domaine appelé μTEP permet l'études de pathologies et la validation de modèles physiologiques ou de procédures thérapeutiques. Depuis le début des années 90, l'imagerie μTEP a connu un essor important et a pour défi de réduire sa résolution d'un facteur 10 (actuellement la résolution des images μTEP est de l'ordre de 1 mm). Ces scanners μTEP sont utilisés par exemple pour "tracer" l'expression génétique (via les ARN messagers) et pourraient apporter un développement capital pour la thérapie génique.

1.4 Scanners TEP

Le but des scanners TEP est de localiser et d'enregistrer toutes les désintégrations $\beta+$ caractérisées par une ligne de réponse ou LOR (Line Of Response). Elle correspond à la droite tracée par la paire de photons γ émise lors la réaction d'annihilation. C'est cette

information sur le lieu d'annihilation qui est utilisée lors de la phase de reconstruction.

1.4.1 Détection des paires de photons γ

Pour détecter les émissions de paires de photons, les scanners sont constitués de couronnes de détecteurs de photon γ placées tout autour du patient (voir figure 1.3). Les détecteurs élémentaires d'une taille de quelques mm^2 sont regroupés dans des panneaux, chacun formé de plusieurs blocs de cristaux de scintillation et de photomultiplieurs. Chaque détecteur a pour but d'arrêter les photons incidents et de déterminer leur énergie et leur date d'émission. Ensuite une électronique de détection, ou circuit de coïncidence, va apparier les deux photons issus d'une même réaction d'annihilation. Une fenêtre temporelle de quelques ns et une fenêtre en énergie autour des 511 keV de l'émission, sont alors utilisées lors de l'appariement.

Les coïncidences sont stockées soit en mode liste soit en mode histogramme. En mode liste, les coïncidences sont simplement enregistrées de manière chronologique alors qu'en mode histogramme ou sinogramme les événements correspondants à une même paire de détecteurs sont sommés puis stockés dans une seule et même donnée appelée *bin*. Dans ce dernier cas, le scanner comptabilise alors le nombre d'événements enregistrés sur chaque LOR.

1.4.2 Rapport signal sur bruit

La résolution en temps et en énergie de ce circuit de coïncidence n'est pas parfaite En effet, aux coïncidences vraies viennent s'ajouter des erreurs de mesures (voir figure 1.4). Le ratio NEC (Noise Equivalent Counting rate) est la mesure standard pour évaluer le rapport signal sur bruit des scanners TEP. Le signal est constitué des coïncidences vraies. Le bruit des erreurs de mesure est constitué des coïncidences diffusées C_{diff} et aléatoires C_{aleat} :

$$NEC = \frac{C_{vrai}^2}{C_{aleat} + C_{diff} + C_{vrai}} \tag{1.1}$$

La puissance du signal dépend du taux d'atténuation des photons dans le corps du patient, du temps de réponse du scanner qui peut amener à la saturation du système de détection (*dead time*) et de l'ajout ou non de septa en plomb entre chaque couronne de détection (mode d'acquisition 2D ou 3D). Les septa en plomb ont pour but d'éliminer les photons ayant un angle d'incidence trop important (voir figure 1.5). Avec ces septa, seules les paires de photons détectés dans la même couronne sont pris en compte ce qui correspond à une acquisition uniquement transversale (2D). Si nous nous appuyons sur le ratio NEC, le mode 3D (sans septa) offre une acquisition plus intéressante jusqu'à un certain niveau d'activité (voir figure 1.6). Car même si le mode 3D a le désavantage d'être plus sensible aux coïncidences aléatoires et diffusées que le mode 2D, il offre une sensibilité jusqu'à trois fois plus importante aux coïncidences vraies.

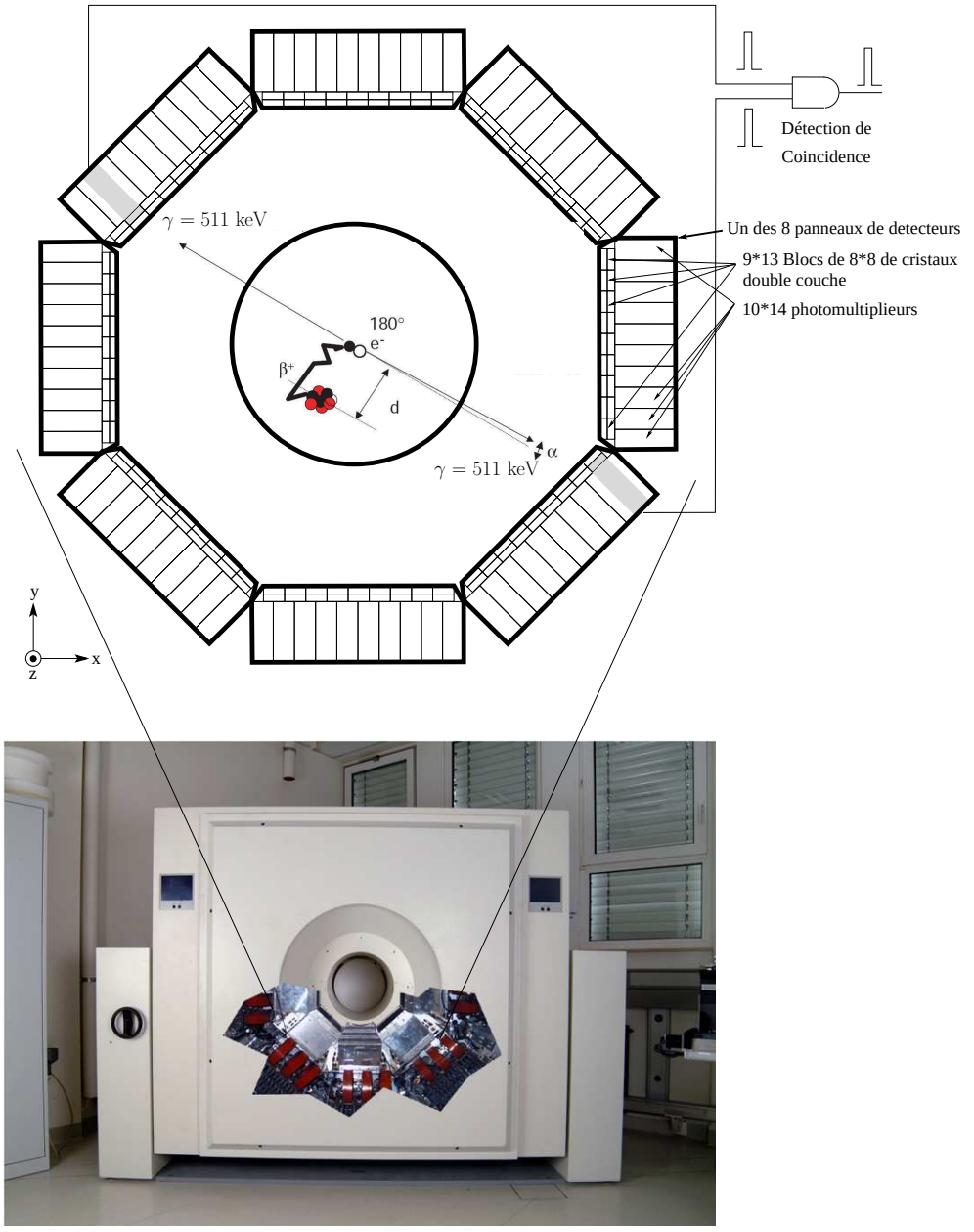


FIG. 1.3 – Acquisition TEP sur le scanner HRRT. Le scanner HRRT est formé de 8 panneaux de détecteurs placés en octogone autour du champs de vue. Un panneau est formé de $13 * 9$ blocs (13 en profondeur et 9 en largeur) de $8 * 8 * 2$ détecteurs (le HRRT a une double couche de cristaux). Le scanner peut donc être considéré comme constitué de $13 * 8 = 104$ anneaux de détecteurs comme celui représenté schématiquement au dessus de la photo du HRRT de l'institut Max-Planck.

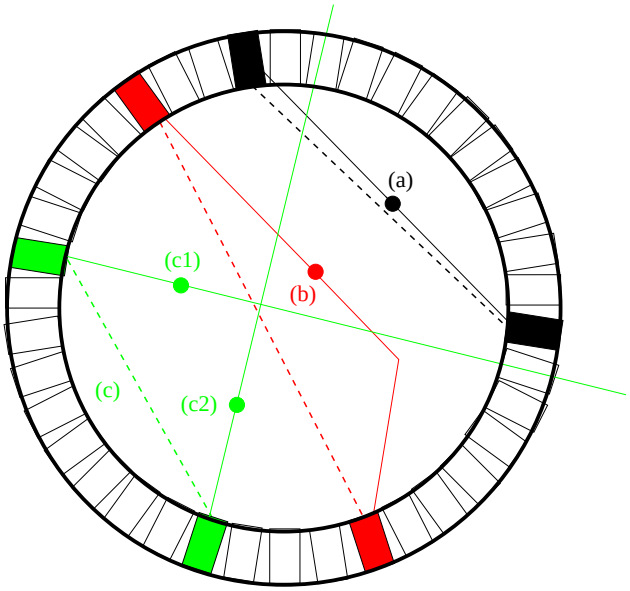


FIG. 1.4 – La détection des deux photons issus d'une même annihilation correspond à une coïncidence vraie (a), l'événement est associé à une LOR (en pointillé sur la figure). Une coïncidence diffusée (b) est enregistrée dès qu'un des deux photons est dévié de sa trajectoire suite à une diffusion de Compton. La LOR enregistré (en traits pointillés) ne correspond plus alors à la véritable LOR (en trait plein). Une coïncidence aléatoire (c) correspond à la détection de deux photons provenant de deux annihilations distinctes (c1 et c2). Elle comptabilise alors une annihilation sur une LOR où aucun événement ne s'est produit.

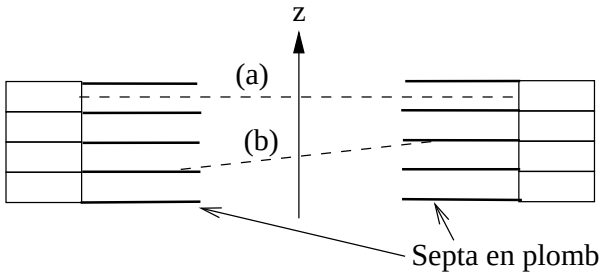


FIG. 1.5 – Lorsque des septa en plomb sont positionnés entre chaque couronne de détecteur, seuls les événements dont la LOR est horizontale (a) sont enregistrés. Les événements dont la LOR est oblique ((b) voient leurs photons émis stoppés par les septa en plomb.

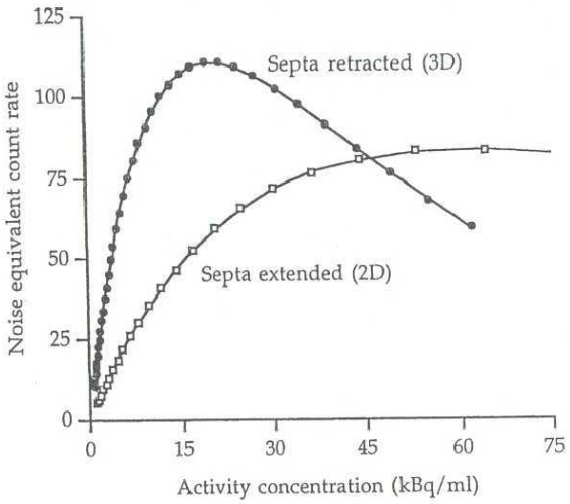


FIG. 1.6 – Courbes NEC pour l'acquisition 2D et 3D du scanner HR+ [**?**]

2 Formulation du problème de reconstruction

La reconstruction tomographique est qualifiée de problème inverse car elle consiste à déterminer la cause (l'organisme étudié) à partir de l'observation des effets (les mesures faites à l'aide du scanner). Nous précisons dans cette section à quoi correspondent en tomographie TEP, l'espace des causes (espace image) et l'espace des effets (espace de projection).

2.1 Définition du volume reconstruit

L'objectif de l'imagerie moléculaire est de suivre le chemin parcouru par tous les radiotraceurs à travers le corps humain. L'imagerie TEP cherche à retrouver plus modestement la densité de traceurs à une position donnée et à un instant donné plutôt que le parcours individuel de chaque radiotraceur. On fait pour cela l'hypothèse que les positions individuelles d'un radiotraceur à un instant t sont des variables aléatoires indépendantes et identiquement distribuées (*iid*) suivant la même densité de probabilité $p_t(\vec{r})$.

La densité de traceurs $p(\vec{r})$ n'est pas observée directement et instantanément mais indirectement via la réaction d'annihilation résultante de la désintégration radioactives des traceurs pendant la durée de l'acquisition. C'est pourquoi le volume reconstruit correspond à la densité d'émission radioactive $f(\vec{r})$, c'est à dire à l'intégrale de la densité du taux d'émission radioactive $\eta(\vec{r},t)$ sur la durée d'acquisition (voir équation 1.2). Puisque la densité du taux d'émission radioactive $\eta(\vec{r},t)$ est proportionnelle à la densité instantanée de traceurs $p_t(\vec{r})$, nous pouvons considérer que cette densité d'émission radioactive $f(\vec{r})$ constitue un bon reflet de la densité de traceurs.

$$f(\vec{r}) = \int_0^T \eta(\vec{r},t)dt$$

$$\begin{aligned} &avec \quad \eta(\vec{r},t) = \mu_N \frac{e^{-\frac{t}{\mu_T}}}{\mu_T} p_t(\vec{r}) \\ &\quad \mu_N \text{ nombre de radiotraceurs administrés au patient} \\ &\quad \mu_T \text{ constante de temps de désintégration radioactive du traceur} \end{aligned} \tag{1.2}$$

2.2 Problème inverse

Le problème inverse consiste pour chaque vecteur position $\vec{r}$ à retrouver la densité d'émission $f(\vec{r})$ à partir des données des capteurs. Mais puisque les capteurs ne font que comptabiliser le nombre y_i de paires d'électrons frappant en coïncidence leurs surfaces, une simple mesure ou *bin* ne nous permet pas à elle seule de retrouver le vecteur position $\vec{r}$. En effet la zone de couverture d'une paire de détecteur, même si elle restreint l'espace des solutions, regroupe plusieurs lieux possibles d'annihilation (en première approximation, le parallélépipède délimité par les surfaces des deux détecteurs appariés). Ainsi la mesure pour un couple de détecteurs d'une densité de traceurs situés à la position $\vec{r}$,

se retrouve mélangée avec celle des densités de traceurs situées à des positions $\vec{r}$ différentes mais contenues également dans le champs de vue du couple de détecteur. De plus, puisque l'activité nucléaire associée à un vecteur position $\vec{r}$ atteint potentiellement plusieurs capteurs (l'émission radioactive est isotropique), la mesure de l'activité radioactive d'une densité de traceurs à la position $\vec{r}$ se retrouve donc dispersée dans plusieurs capteurs. Le défi de la résolution du problème inverse consiste précisément à regrouper et "démêler" ces mesures pour quantifier l'activité nucléaire de chaque zone de l'espace image. La résolution de ce problème inverse est abordée par deux principales familles d'algorithmes de reconstruction : les méthodes analytiques et les méthodes itératives.

2.3 Dimension de l'espace de projection $\mathbb{P}$

En acquisition 2D, l'espace de projection $\mathbb{P}$ est de même dimension que l'espace image $\mathbb{I}$. Le problème inverse est suffisamment déterminé. En mode d'acquisition 3D par contre, l'espace des mesures $\mathbb{M}$ est de dimension 4 avec l'ajout de l'angle θ d'inclinaison des LORs, alors que l'espace image $\mathbb{I}$ est de dimension 3. Le problème inverse est donc sur-déterminé et l'espace de projection est implicitement redondant. L'information additionnelle correspondante permet d'améliorer la robustesse des algorithmes de reconstruction aux bruits statistiques venant entacher les mesures.

2.4 Modélisation de l'espace de projection $\mathbb{P}$ et de l'espace image $\mathbb{I}$

Tel que nous avons modélisé le problème de reconstruction jusqu'à présent, l'espace image $\mathbb{I}$ est continu avec l'image représentée par la fonction à variables continues $f(\vec{r})$ et l'espace de projection $\mathbb{P}$ est discontinu avec les données correspondant au nombre d'événements y_i de chaque paire de capteur. Les méthodes de reconstruction modifient cette représentation Discrète-Continu (D-C). Les méthodes analytiques utilisent une modélisation Continu-Continu (C-C) : elles interprètent les données discrètes y_i comme des échantillons d'une fonction à variables continues dans l'espace de projection $\mathbb{P}$. Les méthodes discrètes (ou itératives) utilisent une modélisation Discrète-Discrète (D-D) : elles interprètent la fonction inconnue $f(\vec{r})$ comme une combinaison linéaire d'un nombre fini de volumes élémentaires.

3 Méthodes analytiques

La solution analytique est appelée inverse de la transformée de Radon et date de 1917 [Toft 96]. Elle associe les projections (espace de Radon) au volume 3D (espace image). Les méthodes analytiques basées sur l'inverse de la transformée de Radon, sont rapides et largement utilisées en routine clinique. Toutefois basées sur un modèle physique très simplifié du processus de mesure, elles offrent une qualité de reconstruction inférieure aux méthodes itératives.

Nous verrons dans cette section que la modélisation purement analytique de l'acquisition permet de créer une correspondance dans le domaine de Fourier entre l'espace image $\mathbb{I}$ et l'espace de projection $\mathbb{P}$ et que cette correspondance est une preuve de l'existence

d'une solution analytique au problème inverse. Nous présenterons ensuite deux méthodes analytiques basées sur ce résultat : la reconstruction directe dans l'espace de Fourier et la reconstruction par rétroprojection filtrée.

3.1 Modèle physique du processus d'acquisition

Le domaine d'intégration des événements enregistrés par un récepteur est réduit à une seule dimension : elle se fait, de manière uniforme, le long d'une ligne infinie qui remplace le tube de réponse (parallélépipède ayant pour bases les surfaces des deux détecteurs appariés).

Pour localiser les LORs en 2D, on utilisera les coordonnées polaires (ρ, ϕ) dans le plan image (voir figure 1.7). En 3D, on utilisera les coordonnées $(u_\parallel, v_\parallel, w_\parallel)$ définies par les rotations axiale (angle θ) et transaxiale (angle ψ) des coordonnées originales (x, y, z) comme illustrées en figure 1.8. Dans ce système de coordonnées, la LOR $(u_\parallel, v_\parallel, \psi, \theta)$ a pour direction $\vec{w_\parallel}$ et passe par le point $(u_\parallel, v_\parallel)$ du plan $\Pi_{\psi,\theta}$. Pour caractériser le processus d'acquisition, on définit l'opérateur de projection P comme l'intégration d'une densité $f(\vec{r})$ le long d'une LOR :

$$Pf(u_\parallel, v_\parallel, \psi, \theta) = \int_{LOR(u_\parallel, v_\parallel, \psi, \theta)} f(x, y, z) dw_\parallel \tag{1.3}$$

On interprète alors de façon géométrique la projection $p_{\psi,\theta}(u_\parallel, v_\parallel) = Pf(u_\parallel, v_\parallel, \psi, \theta)$ comme étant la valeur de la projection du volume sur le plan $\Pi_{\psi,\theta}$ prise au point $(u_\parallel, v_\parallel)$. A chaque mesure correspond ainsi une LOR comme illustrée sur les figures 1.7 et 1.8.

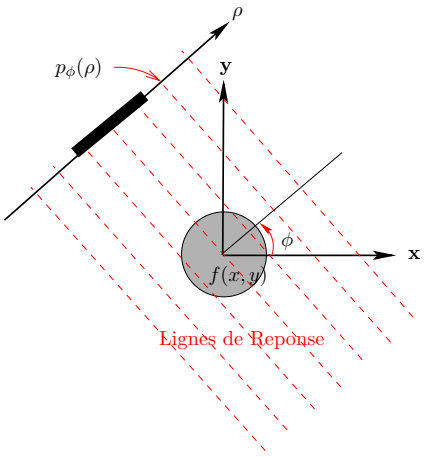


FIG. 1.7 – Repère de projection 2D

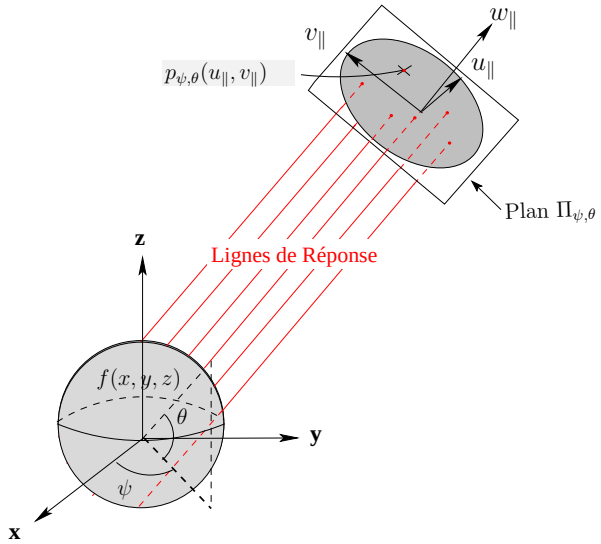


FIG. 1.8 – Repère de projection 3D

3.2 Correspondance des espaces $\mathbb{I}$ et $\mathbb{P}$ dans le domaine de Fourier

Dans cette section, nous énonçons le théorème de la coupe centrale qui expose la correspondance existant entre l'espace image $\mathbb{I}$ et l'espace de projection $\mathbb{P}$ dans le domaine de Fourier, puis nous définissons la sphère d'Orlov, système de représentation des plans de projection. Nous répondrons ensuite à la question suivante : à partir de quels plans de projection, peut on calculer $F(\vec{v})$, la valeur à la fréquence v de la transformée de Fourier de l'image f ? La réponse nous permettra de définir "la" solution analytique au problème inverse.

3.2.1 Théorème de la coupe centrale (Fourier Slice Theorem)

Enoncé du théorème *La Transformée de Fourier 2D $P_{\psi,\theta}(\upsilon_{u_\parallel}, \upsilon_{v_\parallel})$ de la projection $p_{\phi,\theta}(u_\parallel, v_\parallel)$ du volume $f(x,y,z)$ sur le plan $\Pi_{\psi,\theta}$ est équivalente aux valeurs de la Transformée de Fourier $F(\upsilon_x, \upsilon_y, \upsilon_z)$ du volume $f(x,y,z)$ situées sur le plan passant par l'origine et définie par les angles ψ et θ :*

$$P_{\psi,\theta}(\upsilon_{u_\parallel}, \upsilon_{v_\parallel}) = F(\upsilon_x, \upsilon_y, \upsilon_z)\big|_{\upsilon_{w_\parallel}=0} \tag{1.4}$$

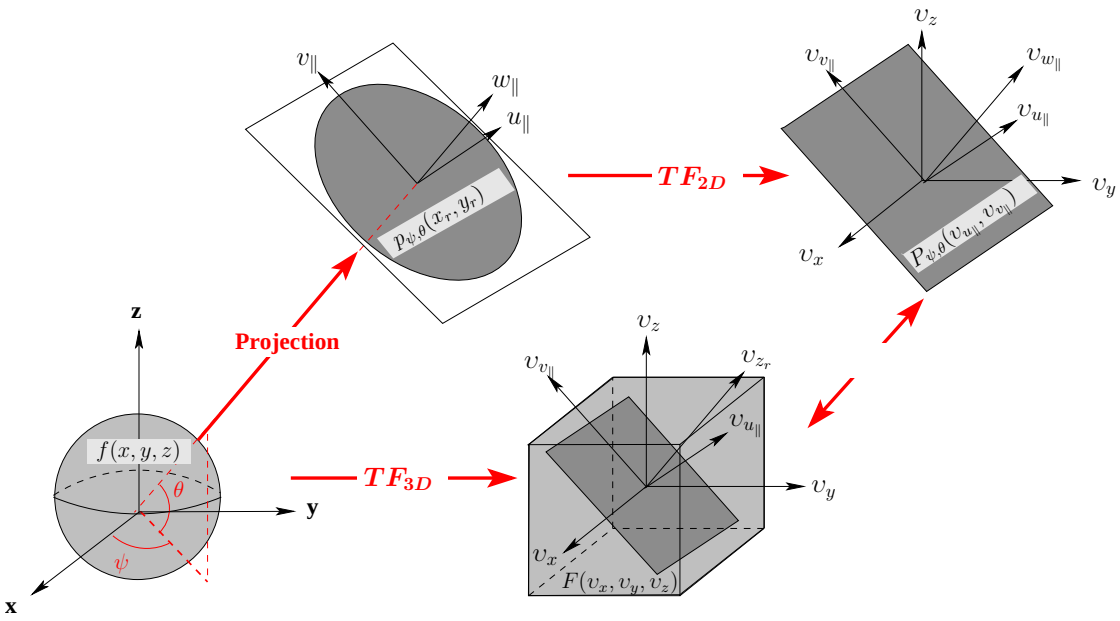


FIG. 1.9 – Théorème de la coupe centrale

Ce théorème clef de la tomographie nous montre le lien qui existe dans l'espace de Fourier entre le volume d'origine et ses projections comme illustré sur la figure 1.9. Ce principe fondamental sous-tend tout algorithme analytique et nous aidera à les comprendre de manière intuitive. Certaines méthodes utilisent même directement le résultat 1.4 de ce théorème.

3.2.2 Plans de projection solutions dans l'espace d'Orlov

L'espace d'Orlov représente sur une sphère unité l'ensemble des positions prises par le vecteur $\vec{w_\parallel}$ normal aux plans de projection . Dans cet espace, ce vecteur pointe un point $\Omega_{\psi,\theta}$ sur la sphère unité (voir figure 1.10). En pratique, l'ensemble des plans de projection acquis par un scanner est réduit à l'ensemble $\Omega_{scan} = \left\{\vec{w_\parallel}(\psi,\theta)|0 \leq \psi < \pi, |\theta| \leq \Theta_{scan}\right\}$, avec Θ_{scan} ouverture angulaire maximale du scanner.

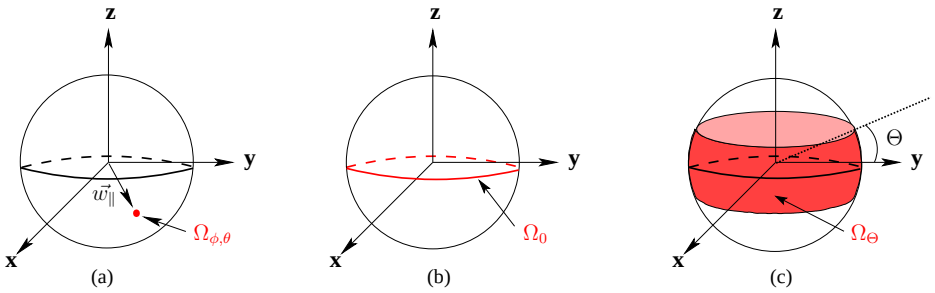


FIG. 1.10 – Espace d'Orlov : (a) point $\Omega_{\psi,\theta} = \left\{\vec{w_\parallel}(\psi,\theta)\right\}$; (b) cercle $\Omega_0 = \left\{\vec{w_\parallel}(\psi,\theta)|0 \leq \psi < \pi, \theta = 0\right\}$; (c) ensemble $\Omega_{scan} = \left\{\vec{w_\parallel}(\psi,\theta)|0 \leq \psi < \pi, |\theta| \leq \Theta_{scan}\right\}$

D'après le théorème de la coupe centrale, la valeur $F(\vec{v})$ peut être obtenue à partir de la TF des plans de projection de normale $\vec{w_\parallel}$ orthogonal au vecteur fréquence $\vec{v}$:

$$F(v_{u_\parallel}, v_{v_\parallel}, \psi_v, \theta_v) = P_{\psi,\theta}(v_{u_\parallel}, v_{v_\parallel}) \quad \Leftrightarrow \quad \vec{v_{w_\parallel}}(\psi_v, \theta_v).\vec{w_\parallel}(\psi,\theta) = 0 \tag{1.5}$$

Or il existe un angle ψ solution de $\vec{v_{w_\parallel}}(\psi_v, \theta_v).\vec{w_\parallel}(\psi,\theta) = 0$ si et seulement si $|\sin\theta| \leq |\cos\theta_v|$. Ainsi l'ensemble des solutions de l'équation 1.5, forment un cercle tout autour de la sphère d'Orlov. Mais l'intersection de ce cercle avec l'ensemble des plans acquis par le scanner Ω_{scan} restreint l'espace des solutions aux plans $\Pi_{\psi,\theta}$ normaux au vecteur $\vec{v}$ et tels que $-\Theta_{lim} \leq \theta \leq \Theta_{lim}$. L'angle limite Θ_{lim} est défini par l'équation 1.6.

$$\Theta_{lim} = \begin{cases} \Theta_{scan} & \text{si } cos(\theta_v) > sin(\Theta_{scan}) \\ \pi/2 - \theta_v & \text{sinon} \end{cases} \tag{1.6}$$

Comme illustré sur la figure 1.11, cet espace de solution trace sur la sphère d'Orlov un cercle (a) ou un arc de de cercle (b) selon l'angle d'inclinaison θ_v du vecteur fréquence $\vec{v}$.

Le calcul de $F(\vec{v})$ se fait alors en calculant la moyenne des projections acquises par le scanner (Ω_{scan}) et solutions de l'équation 1.5. Ces projections sont pondérées selon les angles d'inclinaison du vecteur fréquence θ_v et du plan de projection θ de la manière suivante ([Kinahan 04]) :

$$F(\vec{v}) = H_n(\vec{v}) \int_{-\Theta_{lim}}^{+\Theta_{lim}} \frac{2\cos\theta}{\sqrt{\cos^2\theta_v - \sin^2\theta}} P_{\psi,\theta}(v_{u_\parallel}, v_{v_\parallel}) d\theta \tag{1.7}$$

$H_n(\vec{v})$ représente le facteur de normalisation. Sa valeur varie selon le nombre de plans de projections permettant de calculer $F(\vec{v})$:

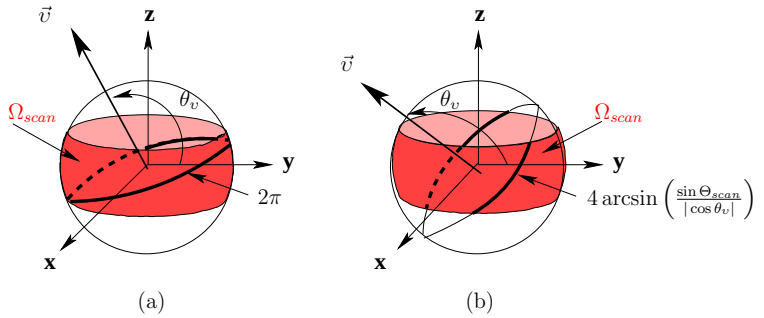


FIG. 1.11 – L'ensemble des plans de projections permettant de calculer la TF_{3D} de $f(\vec{r})$ pour la fréquence $\vec{v}$, trace sur Ω_{scan} soit un cercle (a) soit un arc de cercle (b)

$$\begin{aligned} \frac{1}{H_n(\vec{v})} &= \int_{-\Theta_{lim}}^{+\Theta_{lim}} \frac{2\cos\theta}{\sqrt{\cos^2\theta_v - \sin^2\theta}} d\theta \\ &= \begin{cases} 2\pi & \text{si } cos(\theta_v) \leq sin(\Theta_{scan}) \\ 4\arcsin\left(\dfrac{\sin\Theta_{scan}}{\cos\theta_v}\right) & \text{sinon} \end{cases} \end{aligned} \tag{1.8}$$

3.2.3 "La" solution au problème inverse

Le théorème d'Orlov indique qu'une condition suffisante pour la reconstruction d'un volume est la possession des projections complètes sur un cercle unité interceptant la sphère. Ainsi la solution unique au problème inverse est définie en utilisant par exemple l'ensemble des sinogrammes directes complets Ω_0 :

$$\Omega_0 = \left\{ \vec{w_\parallel}(\psi, \theta) | 0 \leq \psi < \pi, \theta = 0 \right\}$$

Toutefois, nous pouvons remarquer qu'un cercle dans l'espace d'Orlov représente une infinité de projections et que nous négligeons tout bruit de mesure. Or le problème inverse est dit "mal posé " ("*ill-posed problem*"). Ainsi, en pratique non seulement l'unicité de la solution n'est plus vérifiée à cause du nombre fini de projections (voir exemple 1.12) mais l'existence même de solutions disparaît avec le bruit entâchant les mesures. Toute solution n'est alors plus qu'une approximation.

3.3 Reconstruction directe dans l'espace de Fourier

Grâce au calcul de $F(\vec{v})$ en utilisant le théorème de la coupe centrale, il est possible de reconstruire directement le volume d'origine en prenant la transformée de Fourier inverse de $F(\vec{v})$. Dans la pratique, cette méthode est accélérée par l'utilisation de la Fast Fourier Transform (FFT). La méthode consiste tout d'abord à calculer par FFT la TF_{2D} des plans de projection, de les mapper dans l'espace 3D de Fourier et de calculer ensuite l'inverse de Fourier par iFFT. A ces étapes, il faut rajouter une étape d'interpolation complexe (*Gridding*) qui place les échantillons des TF_{2D} des plans de projection sur

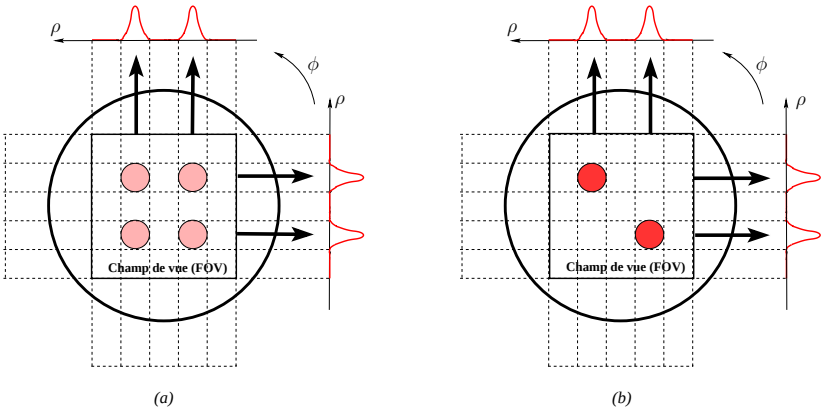


FIG. 1.12 – Dans le cas 2D où l'espace des projections est réduit aux deux projections $\phi = 0$ et $\phi = \pi/2$, les deux images (a) et (b) sont solutions du problème inverse

une grille cartésienne dans l'espace de Fourier afin que l'inverse de Fourier puisse être calculée. Par exemple en 2D, les échantillons ne sont pas placés sur une grille cartésienne mais sur une grille polaire (voir figure 1.13). En effet, chaque ligne de projection est échantillonnée uniformément mais ces lignes sont placées dans le plan de Fourier selon leur angle polaire ϕ avec l'axe x.

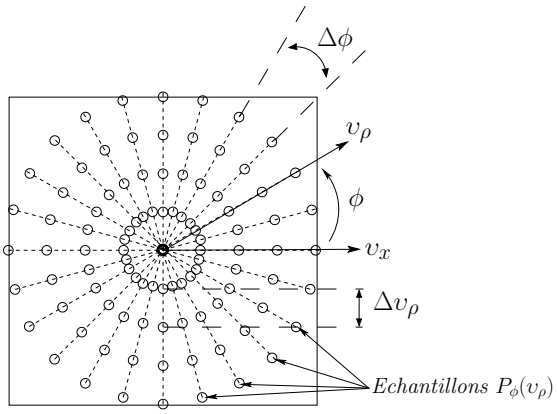


FIG. 1.13 – Répartition des échantillons de la transformée de Fourier de la projection dans l'espace de Fourier de l'image (cas 2D)

3.4 Reconstruction par rétroprojection filtrée

Nous présentons d'abord la rétroprojection filtrée dans le cas 2D, qui utilise un espace de projection réduit (l'angle d'inclinaison θ est nul) avant d'étendre sa définition au cas 3D.

3.4.1 La Rétroprojection 2D

La technique classique de reconstruction consiste à rétroprojeter dans l'espace du volume les projections (de coordonnée radiale ρ) selon les différents angles de mesure ϕ :

$$RPf(\vec{r}) = Rp(\rho, \phi) = \int_0^{\pi} p(\rho, \phi) d\phi \tag{1.9}$$

L'opération de rétroprojection R est définie mathématiquement comme le dual de l'opérateur de projection P. Elle n'est pas l'exacte inverse de la transformée de Radon ($RPf \neq f$) car comme nous pouvons l'observer sur la figure 1.14, des artefacts en étoile apparaissent autour de l'objet après rétroprojection. En effet, lors d'une simple rétroprojection en un point $\vec{r}$ toutes les valeurs de l'image contribuent à la valeur finale $RPf(\vec{r})$. Plus précisément la fonction $f(\vec{r})$ est convoluée par une fonction noyau $h(\vec{r})$. En 2D, $RPf(x, y)$ peut alors se mettre sous la forme suivante :

$$RPf(x,y) = f(x,y) ** h(x,y) \quad \text{avec} \quad \begin{cases} h(x,y) = \dfrac{1}{\sqrt{x^2+y^2}} \\ ** : \text{opérateur de convolution 2D} \end{cases}$$

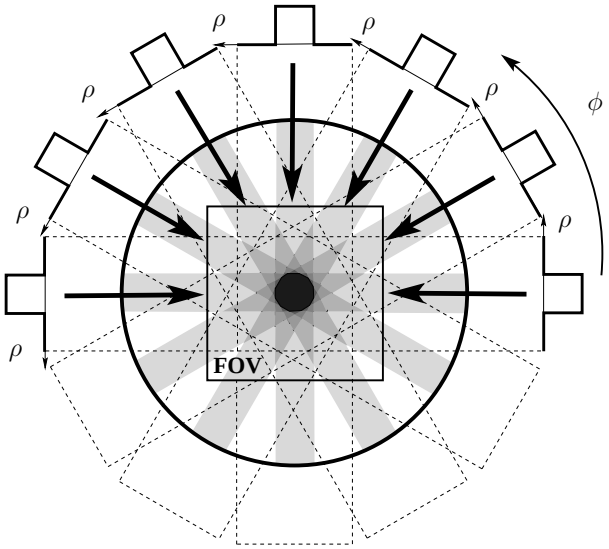


FIG. 1.14 – Rétroprojection en TEP 2D

Pour obtenir l'image originale $f(\vec{r})$, il faut donc déconvoluer l'image rétroprojetée. La déconvolution se fait par filtrage soit avant (FBP : Filtered BackProjection), soit après (BFP : BackProjection Filtered) la rétroprojection.

3.4.2 La rétroprojection filtrée 2D (Algorithme FBP2D)

L'algorithme FBP est l'algorithme le plus couramment utilisé. Il évite une interpolation dans l'espace fréquentiel en filtrant les projections avant rétroprojection plutôt que le volume après rétroprojection. La première étape consiste donc pour chaque projection $p_\phi(\rho)$ à calculer sa TF_{1D}, la filtrer par $H(\vec{v})$ et en prendre l'inverse de Fourier :

$$p^F(\rho,\phi) = TF_{1D}^{-1}\left\{H(\vec{v})TF_{1D}\left\{p_\phi(\rho)\right\}\right\} \tag{1.10}$$

Le filtre H est composé d'un filtre rampe $|v|$, correspondant au noyau h(x,y) décrit précédemment et d'une fenêtre d'apodisation W :

$$H(\vec{v}) = W(|\vec{v}|)\cdot|\vec{v}| \tag{1.11}$$

La fenêtre d'apodisation $\boldsymbol{W(|\vec{v}|)}$ est utilisée pour réduire l'amplification du bruit due à l'opération de filtrage. Différentes fenêtres sont utilisées comme la fenêtre de Hamming (fig. 1.15).

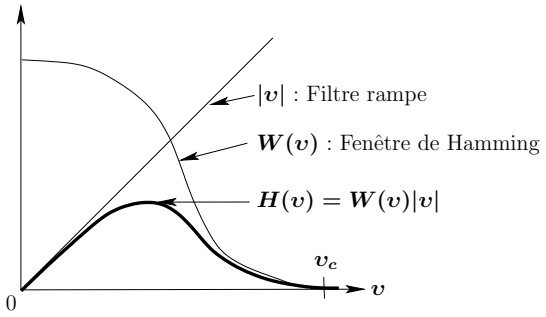


FIG. 1.15 – Apodisation du filtre rampe par la fenêtre de Hamming

Une fois filtrées, les projections sont utilisées pour reconstruire le volume original par rétroprojection :

$$\boldsymbol{f(\vec{r}) = Rp^F(\rho,\phi)} \tag{1.12}$$

3.4.3 Réarrangement des données (*rebinning*)

La reconstruction par rétroprojection filtrée 2D utilise les projections complètes décrivant un cercle à l'équateur de la sphère d'Orlov (cercle $\boldsymbol{\Omega_0}$ sur la figure 1.10). Si cette technique de reconstruction respecte ainsi le théorème d'Orlov, elle sera toujours moins robuste au bruit de mesure que les algorithmes de reconstruction utilisant l'ensemble des plans de projections $\boldsymbol{\Omega_\Theta = \left\{\vec{w}_\parallel(\psi,\theta)|0 \le \psi < \pi, |\theta| \le \Theta\right\}}$.

Pour permettre aux algorithmes de reconstruction 2D de prendre en compte l'ensemble des données acquises par le scanner, les algorithmes avec réarrangement de données (*rebinning*) visent à réduire la taille de l'espace de projection. Après *rebinning*, les données sont composées d'autant de sinogrammes qu'il y a de coupes transverses à

reconstruire. Parmi tous les algorithmes de *rebinning* existant, nous pouvons citer les algorithmes de *Single-Slice ReBinning*(SSRB), de *Multi-Slice ReBinning* (MSRB) et de *FOurier REbinning* (FORE). Ces techniques de compression des données sont couramment utilisées en clinique car elles permettent de simplifier l'étape de reconstruction. Toutefois, elles constituent des techniques approximatives de reconstruction. La qualité de reconstruction obtenue sera toujours moins bonne que celle obtenue avec les algorithmes utilisant des données de projection non compressées.

3.4.4 Rétroprojection filtrée 3D (FBP3D)

La rétroprojection 3D utilise l'ensemble des données de projection sans *rebinning* et effectue une intégration dans l'espace 4D de projection selon $\boldsymbol{\psi}$ et $\boldsymbol{\theta}$:

$$\boldsymbol{RPf(\vec{r}) = Rp(u_{\parallel}, v_{\parallel}, \psi, \theta) = \int_{-\Theta}^{+\Theta} \int_{0}^{\pi} p(u_{\parallel}, v_{\parallel}, \psi, \theta) \cos\theta d\theta d\psi} \tag{1.13}$$

L'algorithme FBP3D effectue comme en 2D, une étape de filtrage des données de projection préalable à la rétroprojection 3D. La première étape consiste donc pour chaque projection $\boldsymbol{p_{\psi,\theta}(u_{\parallel}, v_{\parallel})}$ à calculer sa $\boldsymbol{TF_{2D}}$, la filtrer par $\boldsymbol{H(\upsilon_{u_{\parallel}}, \upsilon_{v_{\parallel}}, \theta)}$ et en prendre l'inverse de Fourier :

$$\boldsymbol{p^F(u_{\parallel}, v_{\parallel}, \psi, \theta) = TF_{2D}^{-1}\left\{ H(\upsilon_{u_{\parallel}}, \upsilon_{v_{\parallel}}, \theta) TF_{2D}\left\{ p_{\psi,\theta}(u_{\parallel}, v_{\parallel}) \right\} \right\}} \tag{1.14}$$

Le filtre $\boldsymbol{H(\vec{\upsilon})}$ de normalisation classiquement utilisé en TEP 3D est celui de Colsher $\boldsymbol{H_c(\vec{\upsilon})}$. Il permet de compenser le nombre de projections contribuant à une composante fréquencielle $\vec{\upsilon}$ donnée. A une composante radiale $|\vec{\upsilon}|$ correspondant au filtrage rampe équivalent à celui utilisé en FBP2D, s'ajoute le filtre de normalisation angulaire $\boldsymbol{H_n(\vec{\upsilon})}$ défini en 3.2.2 :

$$\boldsymbol{H_c(\vec{\upsilon}) = |\vec{\upsilon}| \cdot H_n(\vec{\upsilon}) = \begin{cases} \dfrac{|\vec{\upsilon}|}{2\pi} & \text{si } cos(\theta_{\upsilon}) \leq sin(\Theta_{scan}) \\ \dfrac{|\vec{\upsilon}|}{4 \arcsin\left(\frac{\sin\Theta_{scan}}{\cos\theta_{\upsilon}}\right)} & \text{sinon} \end{cases}} \tag{1.15}$$

3.4.5 Algorithme 3D de ReProjection (3D-RP)

L'algorithme de rétroprojection filtrée 3D ne peut s'appliquer comme l'algorithme FBP2D sur les seules données acquises par le scanner. En effet, contrairement au mode 2D avec septa, les projections ne sont plus complètes pour certaines valeurs de $\boldsymbol{\theta}$. La figure 1.16 illustre le caractère incomplet de certains plans de projection : l'angle solide sous lequel est vu une source depuis les récepteurs du scanner diminue plus on s'éloigne du centre du champ de vue (FOV).

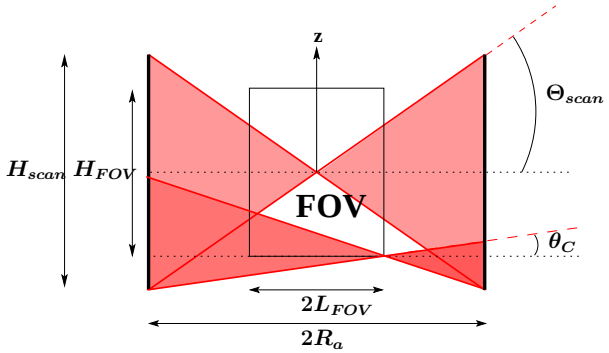


FIG. 1.16 – Variation spatiale de l'acquisition selon la position axiale

L'algorithme 3D de ReProjection (3D-RP) de [Kinahan 89] consiste donc à compléter les données tronquées ($\boldsymbol{\theta_C < |\theta| \leq \Theta_{scan}}$) par une estimation des données manquantes obtenue à partir d'une première reconstruction de l'image. La première reconstruction est effectuée à l'aide des données complètes ($\boldsymbol{0 \leq |\theta| < \theta_C}$). La dernière rétroprojection se fait à partir de tous les plans de projection $\boldsymbol{P_{\psi,\theta}}$ avec $\boldsymbol{0 \leq |\theta| \leq \Theta_{scan}}$ et $\boldsymbol{0 \leq \psi < \pi}$. Les étapes de l'algorithme 3D-RP sont illustrées sur la figure 1.17.

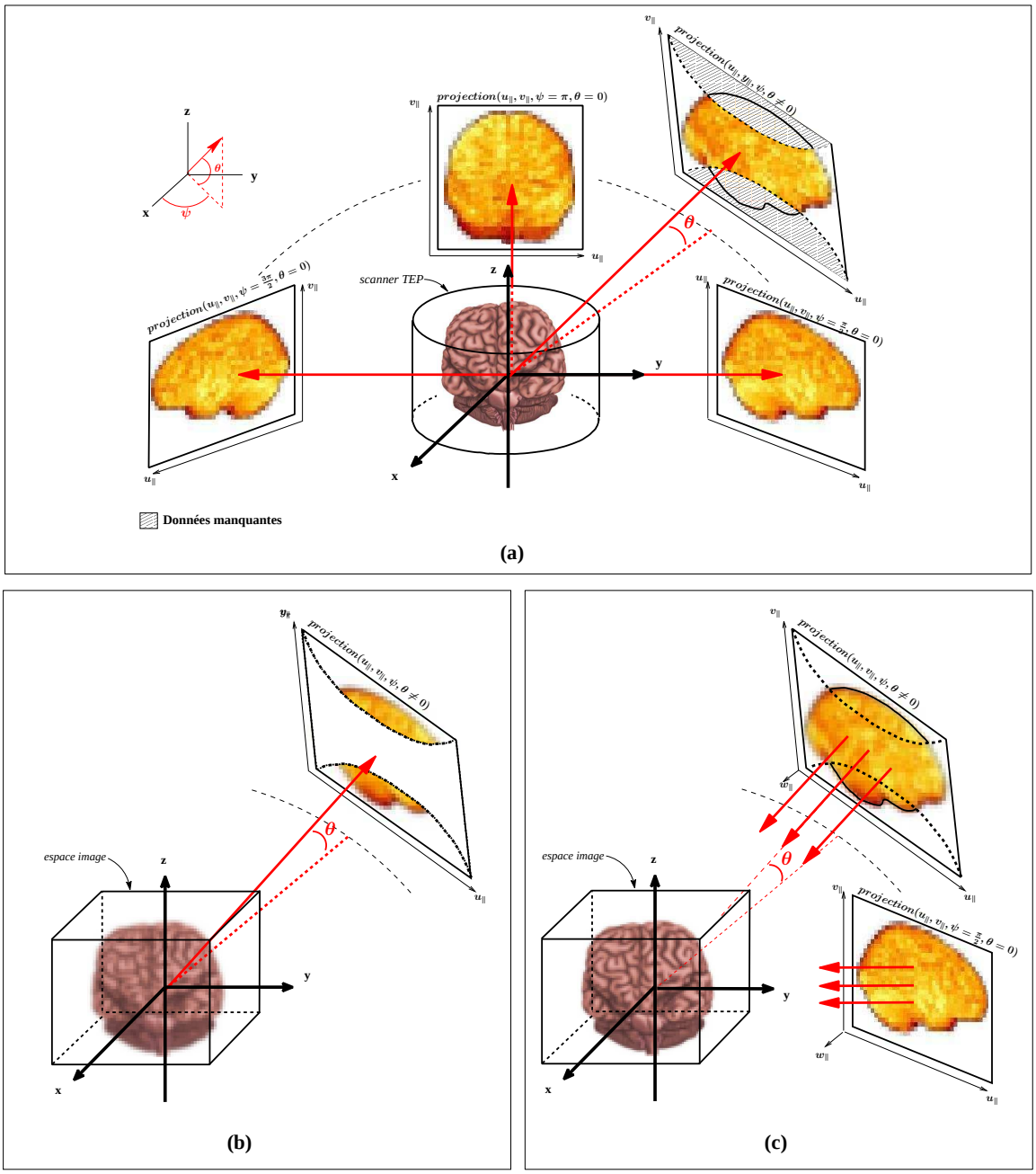


FIG. 1.17 – Algorithme 3DRP : (a) le scanner effectue une acquisition incomplète des plans de projection trop inclinés ($|\boldsymbol{\theta}| > \boldsymbol{\theta_C}$) ; (b) les données de projection tronquées sont complétées en projetant selon les LORs correspondantes la première estimée du volume (obtenue par FBP2D) ; (c) le volume est reconstruit par une rétroprojection filtrée 3D (FBP3D) finale à partir des données acquises et estimées.

4 Méthodes itératives

Plutôt que d'effectuer une modélisation purement analytique du problème de reconstruction en confondant les données acquises et la transformée de Radon du volume, les méthodes itératives vont opter pour une modélisation algébrique du problème. Les données composées de $\boldsymbol{I}$ *bins* (vecteur $\boldsymbol{y}$) sont vues comme une combinaison linéaire de $\boldsymbol{J}$ éléments définissant le volume (vecteur $\boldsymbol{f}$). Les vecteurs $\boldsymbol{y}$ et $\boldsymbol{f}$ sont liés l'un à l'autre via une matrice $\boldsymbol{H}$ définissant le système d'acquisition :

$$\boldsymbol{y} = \boldsymbol{H}\boldsymbol{f} \tag{1.16}$$

La reconstruction consiste alors à trouver l'inverse de la matrice H. Cette matrice étant d'une taille trop importante pour calculer directement son inverse, les méthodes itératives effectuent des approximations successives afin d'affiner de proche en proche la solution courante. Ces méthodes peuvent être purement algébriques ou bien s'appuyer sur une modélisation statistique du problème.

Nous présentons dans cette section comment sont définies de manière générale les méthodes itératives en nous intéressant plus particulièrement à la méthode ML-EM (*Maximum Likelihood - Expectation Maximization*). Les cinq choix définissant les méthodes itératives seront abordées : paramétrisation du volume, modélisation du système d'acquisition, modèle statistique d'émission des positons, fonction de coût $\boldsymbol{\Psi}$ et l'algorithme de maximisation associé.

4.1 Paramétrisation de l'objet

La densité $\boldsymbol{f(\vec{r})}$ du volume est approchée par une combinaison linéaire d'un nombre fini de fonctions de base $\boldsymbol{b_j(\vec{r})}$:

$$\boldsymbol{f(\vec{r})} \simeq \sum_{\boldsymbol{j=1}}^{\boldsymbol{J}} \boldsymbol{f_j b_j(\vec{r})} \tag{1.17}$$

La fonction de base communément utilisée est le *voxel* qui n'est autre qu'un cube élémentaire. D'autres volumes élémentaires existent comme les *blobs*, fonctions à symétrie sphérique qui sont une généralisation de la fenêtre de pondération de Kaiser-Bessel.

4.2 Modèle Physique du Processus de mesure des événements vrais

4.2.1 Un modèle linéaire et spatialement variant (LSV)

Chaque mesure $\boldsymbol{y_i^{vrai}}$ représente une réalisation du processus aléatoire d'émission de positons de moyenne $\boldsymbol{\bar{y}_i^{vrai}}$. Cette espérance est associée avec la densité d'émission $\boldsymbol{f(\vec{r})}$ par un modèle Linéaire et Spatialement Variant (LSV) du processus de mesure [Lewitt 03] :

$$\boldsymbol{\bar{y}_i^{vrai}} = \iiint_{\boldsymbol{FOV}} \boldsymbol{h_i(x,y,z)\ f(x,y,z)\ dxdydz} \tag{1.18}$$

La fonction $\boldsymbol{h_i(\vec{r})}$ désigne la probabilité qu'un événement à la position $\vec{\boldsymbol{r}}$ soit enregistrée par le récepteur $\boldsymbol{i}$. Cette fonction appelée noyau représente la sensibilité des récepteurs. Etendue dans l'espace, elle possède des valeurs non nulles à l'intérieur d'une région en forme de tube et des valeurs nulles en dehors. La discrétisation de la fonction de densité $\boldsymbol{f(\vec{r})}$ amène à définir une version discrète de cette fonction noyau : la matrice système $\boldsymbol{H}$. L'élément $\boldsymbol{h_{ij}}$ de cette matrice représente la probabilité qu'un événement provenant du voxel j soit comptabilisé par le capteur i :

$$\boldsymbol{h_{ij}} \triangleq \iiint_{\boldsymbol{FOV}} \boldsymbol{h_i(x,y,z)\ b_j(x,y,z)\ dxdydz} \tag{1.19}$$

Ainsi, les mesures discrètes $\boldsymbol{\bar{y}_i^{vrai}}$ et les sources discrètes $\boldsymbol{f_j}$ sont liées par l'équation :

$$\boldsymbol{\bar{y}_i^{vrai}} = \sum_{\boldsymbol{j=1}}^{\boldsymbol{J}} \boldsymbol{h_{ij} f_j} = \boldsymbol{[Hf]_i} \tag{1.20}$$

4.2.2 Factorisation de la matrice système

Nous présentons ici la factorisation de [Qi 98] de la matrice système :

$$\boldsymbol{H = H_{sens.det.} H_{err.det.} H_{attn} H_{geom} H_{positon}} \tag{1.21}$$

($\boldsymbol{H_{positon}}$) Incertitude de localisation de la source $\boldsymbol{\beta^+}$ Matrice $\boldsymbol{J * J}$ de lissage locale appliquée sur la source pour corriger l'incertitude due à la distance de vol des positons lors de la thermalisation $\boldsymbol{\beta^+}$ (phénomène intrinsèque au TEP).

($\boldsymbol{H_{geom}}$) Facteur Géométrique Matrice $\boldsymbol{I * J}$ d'association géométrique entre les sources et les récepteurs. L'élément (i,j) correspond à la probabilité qu'une paire de photons produite au voxel j parvienne au détecteur i.

($\boldsymbol{H_{attn}}$) Facteur d'Atténuation Matrice $\boldsymbol{I * I}$ diagonale contenant les probabilités de survie au phénomène d'atténuation le long de chaque LOR.

($\boldsymbol{H_{sens.det.}}$) Facteur de sensibilité des détecteurs Matrice $\boldsymbol{I * I}$ diagonale contenant l'efficacité de détection des paires de récepteurs évaluée lors de la calibration du scanner. On peut introduire dans cette matrice un facteur de correction des temps morts des récepteurs qui déforment de façon non-linéaire la réponse des capteurs.

($\boldsymbol{H_{err.det}}$) Erreurs de détection Matrice $\boldsymbol{I * I}$ de lissage locale appliquée sur les mesures pour corriger trois effets : (i) la non-colinéarité des paires de photons (phénomène intrinsèque au TEP) ; (ii) la diffusion dans les cristaux des détecteurs ; (iii) l'erreur de parallaxe.

Cette factorisation permet de faire ressortir les facteurs intervenant dans le processus de mesure des événements vrais. Dans la suite de notre étude, nous nous focaliserons uniquement sur la matrice géométrique $\boldsymbol{H_{geom}}$ en laissant de côté tous les facteurs liés aux erreurs de mesure.

4.3 Distribution de probabilité des mesures autour de leur valeur espérée

Après la modélisation du volume et du système d'acquisition, la dernière modélisation concerne le processus d'émission des positons. Elle fait le lien entre la mesure $\boldsymbol{y_i^{vrai}}$ et l'espérance $\boldsymbol{\bar{y}_i^{vrai}}$ du processus d'émission.

4.3.1 Modèle non statistique

Les méthodes algébriques utilisent aucun modèle statistique. Elles ne font pas la différence entre la mesure $\boldsymbol{y^{vrai}}$ et l'espérance $\boldsymbol{\bar{y}^{vrai}}$:

$$\boldsymbol{y^{vrai} = \bar{y}^{vrai}} \tag{1.22}$$

4.3.2 Modèle poissonien

La probabilité de désintégration radioactive des traceurs suit une loi exponentielle. En faisant l'hypothèse que les événements sont indépendants, alors une loi de poisson est une statistique toute indiquée pour caractériser le processus aléatoire $\boldsymbol{y^{vrai}}$:

$$\boldsymbol{y^{vrai} \sim Poisson\left\{\bar{y}^{vrai}\right\}} \tag{1.23}$$

Le problème de reconstruction est alors défini par la recherche de la densité $\boldsymbol{f}$ telle que :

$$\begin{aligned} \boldsymbol{\bar{y}^{vrai} &= Pf} \\ \text{avec} \quad \boldsymbol{p(y_i^{vrai}|f)} &= \boldsymbol{\prod_{i=1}^{I} \frac{\bar{y}_i^{y_i} e^{-\bar{y}_i}}{y_i!}} \end{aligned} \tag{1.24}$$

4.3.3 Modèle gaussien

Lorsque le nombre moyen d'événements est important, la loi de poisson peut être approchée par une loi gaussienne autour de l'espérance moyenne $\boldsymbol{\bar{y}^{vrai}}$:

$$\boldsymbol{y^{vrai} \sim \mathcal{N}(\bar{y}^{vrai}, \sigma^2)} \tag{1.25}$$

4.3.4 Amélioration des modèles statistiques

La donnée mesurée $\boldsymbol{y^{scan}}$ ne correspond pas uniquement aux événements vrais $\boldsymbol{y^{vrai}}$. Le scanner comptabilise également des événements aléatoires $\boldsymbol{y^{aleat}}$ et des événements déviés par la diffusion des photons dans l'objet $\boldsymbol{y^{diff}}$:

$$\boldsymbol{y^{scan} = y^{vrai} + y^{aleat} + y^{diff}} \tag{1.26}$$

Contrairement aux méthodes analytiques qui effectuent des pré-corrections, les méthodes statistiques offrent la possibilité grâce à des modèles améliorés des mesures $\boldsymbol{y^{scan}}$

d'intégrer ces événements aléatoires et diffusés dans la résolution globale du problème inverse.

4.4 Choix de la fonction de coût $\psi(f)$

Après la modélisation du problème, reste à définir la méthode de reconstruction. Et en premier lieu, c'est la fonction de coût de la méthode qu'il faut définir. Cette dernière pose les contraintes que doivent satisfaire la solution $\hat{f}$ au problème. Les fonctions de coûts sont généralement composée d'un terme $C(y^{vrai}, Hf)$ de correspondance entre les mesures et le modèle, mais également d'un terme de régularisation $R(f)$ dont l'importance est contrôlée par un paramètre de régularisation β. Des contraintes comme la non négativité de la densité f, peuvent aussi être prises en compte dans la fonction de coût. On exprime alors la solution $\hat{f}$ sous la forme suivante :

$$\hat{f} = \min_{f \geq 0} \left[C(y, Hf) + \beta \cdot R(f) \right] \tag{1.27}$$

Le terme de régularisation est là pour palier à la nature "malade" ou "mal posée" du problème inverse (voir 2.2) : une solution qui colle trop aux données devient trop sensible aux bruits qui entachent les mesures. Cette régularisation peut se faire soit en redéfinissant le problème pour le "guérir" totalement en ajoutant par exemple de l'information à priori dans la fonction de coût, soit de manière plus pragmatique en prescrivant des remèdes apaisant la "maladie" comme l'arrêt des algorithmes avant leur convergence ou bien encore l'utilisation de filtres passe-bas sur le volume pour éliminer les hautes fréquences correspondant au bruit.

4.4.1 L'équation algébrique comme seule contrainte

Les méthodes algébriques cherchent à résoudre directement l'équation $y = Hf$. Le problème de reconstruction étant mal-posé, ces méthodes comme par exemple l'ART (*Algebraic Reconstruction Technique*) ne convergent pas. Elles oscillent dans un ensemble de solutions proches de la solution théorique. Des dérivés de la méthode ART (SART, SIRT) introduisent des paramètres de relaxation afin de contrôler la convergence.

4.4.2 Critères des moindres carrés

Les méthodes LS (*Least Square*) visent à minimiser la distance (euclidienne) entre les mesures et les valeurs issues du modèle d'acquisition :

$$\hat{f} = \min_{f \geq 0} ||y - Hf||^2 \tag{1.28}$$

Les dérivés de la méthode ART (SART, SIRT) peuvent être considérées comme une approximation des méthodes des moindres carrés. Parce que les données en TEP sont bruitées, ces méthodes qui visent à coller au plus près des données ont tendance à former des volumes excessivement bruités. Elles sont pour cette raison peu utilisées pour la

reconstruction TEP depuis quelques années. Les méthodes SART et SIRT sont toutefois encore utilisées en tomographie CT où les données sont moins bruitées.

4.4.3 Maximum de vraisemblance

Les méthodes ML (*Maximum Likelihood*) ont pour but de trouver la densité $\boldsymbol{f}$, parmi toutes les densités possibles, qui maximisent la vraisemblance aux données $\boldsymbol{p(y/f)}$:

$$\hat{\boldsymbol{f}} = \max_{\boldsymbol{f \geq 0}} \boldsymbol{p(y|f)} \tag{1.29}$$

Que ce soit en TEP ou en CT, les méthodes ML sont les plus utilisées.

4.4.4 Approche bayésienne

A la différence des méthodes statistiques classiques, l'approche bayésienne définit une densité de probabilité $\boldsymbol{p(f)}$, appelée à priori, pour la densité de traceurs. Ainsi une densité peut être qualifiée de plus plausible qu'une autre selon sa probabilité $\boldsymbol{p(f)}$. Une probabilité à posteriori $\boldsymbol{p(f/y)}$ est également définie et utilisée comme critère de résolution du problème inverse par les méthodes MAP (*Maximum A Posteriori*) :

$$\hat{\boldsymbol{f}} = \max_{\boldsymbol{f \geq 0}} \boldsymbol{p(f|y)} \tag{1.30}$$

En utilisant les lois de Bayes et en prenant le logarithme des quantités à maximiser, on peut faire apparaître une vraisemblance des données au volume $\boldsymbol{p(y/f)}$ et un à priori sur le volume $\boldsymbol{p(f)}$:

$$\hat{\boldsymbol{f}} = \max_{\boldsymbol{f \geq 0}} \big[\ln \boldsymbol{p(y/f)} + \ln \boldsymbol{p(f)} \big] \tag{1.31}$$

4.5 Choix de l'algorithme de maximisation/minimisation de $\psi(\boldsymbol{f})$

Dernière étape, le choix de l'algorithme de maximisation de la fonction de coût $\boldsymbol{\psi(f)}$ permettra d'atteindre plus ou moins rapidement le critère de reconstruction. Dans ce paragraphe, seul l'algorithme EM (*Expectation Maximization*) de la méthode ML-EM est décrit.

4.5.1 Principe de l'algorithme EM

L'algorithme EM (*Expectation-Maximization*) de [Dempster 77], utilise des données dites cachées qui complètent les mesures et ainsi facilitent la recherche de la solution ML de nombreux problèmes statistiques. L'algorithme EM effectue en alternance jusqu'à la convergence, une étape E (*Expectation*) de calcul de l'espérance de la vraisemblance exprimée à l'aide des données complètes, puis une étape M (*Maximization*) de maximisation de cette espérance. En l'absence de bruits de mesure, l'algorithme EM correspond à une descente de gradient et garantit une convergence vers les maxima locaux de vraisemblance. La méthode ML-EM appliquée à la reconstruction TEP, introduite par Lange

et Carson [Shepp 82, Lange 84], intègre la modélisation poissonienne des mesures $\boldsymbol{y_i}$ présentée en 4.3.2.

4.5.2 Structure itérative de la méthode ML-EM

L'algorithme ML-EM présente une structure itérative semblable au schéma général des méthodes itératives (voir figure 1.18). L'algorithme part d'une première estimée du volume $\boldsymbol{f_0}$ qui peut être un volume quelconque ou un volume reconstruit avec une méthode analytique. Ensuite à chaque itération lors de l'étape de projection, une estimation des mesures $\hat{\boldsymbol{y}}^{(\boldsymbol{n})}$ est effectuée à partir de l'estimée courante du volume $\boldsymbol{f}^{(\boldsymbol{n})}$. Ces mesures estimées sont ensuite comparées aux vraies mesures pour calculer une erreur contenue dans l'espace de projection. Cette erreur $\boldsymbol{y_i}/\hat{\boldsymbol{y}}_{\boldsymbol{i}}$ est ensuite renvoyée dans l'espace image lors de l'étape de rétroprojection pour obtenir un correctif $\boldsymbol{C}^{(\boldsymbol{n})}$ appliqué ensuite à l'estimée courante du volume $\boldsymbol{f}^{(\boldsymbol{n})}$. Après normalisation par la sensibilité $\boldsymbol{N}$ du scanner, la n+1 ième estimée du volume $\boldsymbol{f}^{(\boldsymbol{n+1})}$ est alors obtenue :

$$\boldsymbol{f_j^{(n+1)}} = \frac{\boldsymbol{f_j^{(n)}}}{\boldsymbol{N_j}} \boldsymbol{C_j^{(n)}}$$

$$\begin{cases} \boldsymbol{C_j^{(n)}} = \displaystyle\sum_{\boldsymbol{i=1}}^{\boldsymbol{I}} \boldsymbol{h_{ij}} \left(\frac{\boldsymbol{y_i}}{\hat{\boldsymbol{y}}_{\boldsymbol{i}}} \right) \quad \textbf{avec} \quad \hat{\boldsymbol{y}}_{\boldsymbol{i}} = \sum_{\boldsymbol{j'=1}}^{\boldsymbol{J}} \boldsymbol{h_{ij'}} \boldsymbol{f_{j'}^{(n)}} \\ \boldsymbol{N_j} = \displaystyle\sum_{\boldsymbol{i=1}}^{\boldsymbol{I}} \boldsymbol{h_{ij}} \end{cases} \tag{1.32}$$

4.5.3 Interprétation de l'algorithme ML-EM appliqué en TEP

L'expression classique (1.32) de l'algorithme ML-EM ne fait pas apparaître les variables cachées $\boldsymbol{VC_{ij}}$ qui constituent le ressort de l'algorithme EM. En TEP, elles correspondent à la proportion $\boldsymbol{\alpha_{ij}}$ d'événements captés par la paire de détecteur $\boldsymbol{i}$, provenant du voxel $\boldsymbol{j}$. Le calcul de $\boldsymbol{\alpha_{ij}^{(n)}}$ utilise l'expression du nombre d'événements enregistrés par le détecteur $\boldsymbol{i}$ provenant d'un seul voxel $\boldsymbol{j}$ de la $\boldsymbol{n}$ ième estimée du volume, autrement dit $\boldsymbol{h_{ij}} \boldsymbol{f_j^{(n)}}$:

$$\boldsymbol{\alpha_{ij}^{(n)}} = \frac{\boldsymbol{h_{ij}} \boldsymbol{f_j^{(n)}}}{\sum_{\boldsymbol{j'=1}}^{\boldsymbol{J}} \boldsymbol{h_{ij'}} \boldsymbol{f_{j'}^{(n)}}} \tag{1.33}$$

En multipliant chaque mesure $\boldsymbol{y_i}$ par $\boldsymbol{\alpha_{ij}}$, il est alors possible de démêler les données réelles $\boldsymbol{y}$, en redistribuant les événements selon leur provenance dans le volume. Le calcul de $\boldsymbol{\alpha_{ij}}$ permet bien alors d'atteindre une information jusque là cachée car les données brutes comptabilisent les événements sans faire la distinction entre les différentes sources du volume. Enfin en normalisant le nombre total d'événements émis par le voxel $\boldsymbol{j}$ par la sensibilité $\sum_{\boldsymbol{I}}^{\boldsymbol{i=1}} \boldsymbol{h_{ij}}$ du scanner, une nouvelle estimée $\boldsymbol{f_j^{(n+1)}}$ du volume est alors obtenue :

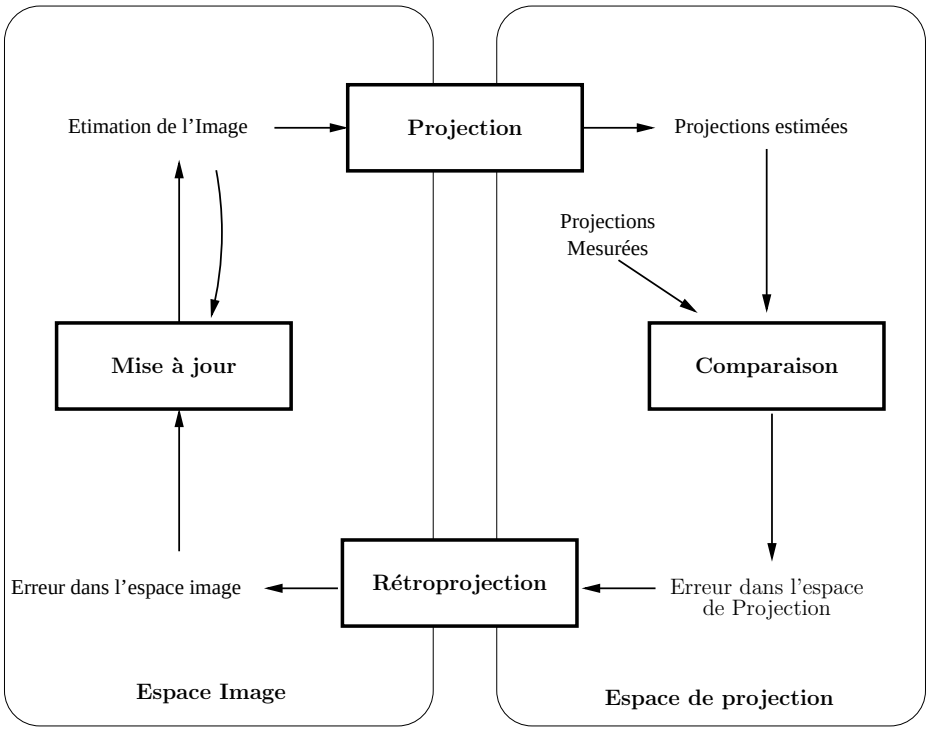


FIG. 1.18 – Schéma général des algorithmes itératifs

$$f_j^{(n+1)} = \frac{\sum_I^{i=1} y_i \cdot \alpha_{ij}^{(n)}}{\sum_I^{i=1} h_{ij}} \tag{1.34}$$

4.5.4 Propriétés de la méthode ML-EM

Avec 30 à 50 itérations nécessaires pour aboutir à une reconstruction correcte, la méthode ML-EM offre une faible vitesse de convergence. Ainsi même si elle permet d'obtenir une meilleure qualité de reconstruction que les méthodes analytiques, son temps d'exécution d'un ou deux ordres de magnitude supérieure à celui des méthodes analytiques, ne permet pas de généraliser son utilisation en clinique. L'utilisation du maximum de vraisemblance comme seul critère de convergence est un autre point faible de cette méthode. Car si l'algorithme EM assure une augmentation de la vraisemblance au fur et à mesure des itérations, cela ne signifie pas que la qualité de reconstruction du volume s'améliore. Au contraire, à partir d'un certain nombre d'itérations, le volume devient de plus en plus bruité par contagion avec le bruit qui entache les mesures. Pour ces deux raisons, il faut dans la pratique arrêter l'algorithme de manière prématurée.

4.5.5 Reconstruction en mode liste

L'algorithme EM peut s'appliquer directement avec des données au format liste comme nous l'avons vu en 1.4.1. En mode liste, un événement k est stocké dans une seule case mémoire avec la date $\boldsymbol{t_k}$ et la LOR d'indice $\boldsymbol{i_k}$ qui lui correspond. L'expression de l'algorithme LMEM (*List Mode EM*) change peu par rapport à celle de l'algorithme EM classique :

$$\boldsymbol{f_j^{(n+1)} = \frac{f_j^{(n)}}{N_j} C_j^{(n)}}$$

$$\begin{cases} \boldsymbol{C_j^{(n)} = \sum_{k=1}^{N_{evenement}} h_{i_k j} \left(\frac{1}{\hat{y}_{i_k}} \right)} \quad \textbf{avec} \quad \boldsymbol{\hat{y}_{i_k} = \sum_{j'=1}^{J} h_{i_k j'} f_{j'}^{(n)}} \\ \boldsymbol{N_j = \sum_{i=1}^{I} h_{ij}} \end{cases} \tag{1.35}$$

Une différence majeure toutefois entre les méthodes EM et LMEM, est que l'espace de projection n'est plus alors parcouru LOR après LOR mais événement après événement. La reconstruction en mode liste est particulièrement intéressante lorsque le nombre d'événements est inférieur au nombre de LORs, puisque alors dans ce cas, la taille des données et la complexité de l'algorithme deviennent moins importantes. La reconstruction en mode liste est donc selon les cas préférée à la méthode classique. Par exemple en TEP dynamique, les multiples acquisitions 3D dites à "faibles statistiques", sont en principe reconstruites de manière plus efficace en mode liste qu'en mode sinogramme.

5 Echantillonnage des données

Nous avons vu que les méthodes itératives utilisent une modélisation discret-discret (espace image $\mathbb{I}$ discret et espace de projection $\mathbb{P}$ discret) et que les méthodes analytiques utilisent une modélisation continu-continu. Toutefois, en pratique quelle que soit la méthode de reconstruction utilisée, une modélisation discret-discret devra être adoptée. En effet, la physique du scanner et la nécessité de stockage des données acquises par le scanner et du volume reconstruit imposent une discrétisation de ces deux espaces de données. A partir des systèmes de coordonnées communément utilisés, nous présenterons la manière dont sont organisées les données acquises par le scanner en mode sinogramme et nous exposerons l'expression discrète de l'algorithme de rétroprojection associée à ce format de données.

5.1 Coordonnées continues

5.1.1 Espace image

Le volume à imager est échantillonné en volume élémentaire appelé voxel. Le voxel est généralement défini comme un parallélépipède dont le centre est localisé par les

coordonnées cartésiennes (x, y, z).

5.1.2 Espace de projection

Dans un scanner TEP, un couple de détecteurs est défini par le quadruplé (A_p, A_q, P_i, Q_j) qui désigne la LOR reliant le capteur P_i de l'anneau A_p au capteur Q_j de l'anneau A_q. Un couple de détecteurs est caractérisé par un couple de numéros d'anneaux (p,q) et par un couple de numéros de détecteurs (i,j). La droite associée à une LOR est alors caractérisée géométriquement par le point et le vecteur directeur suivant :

$$\begin{cases} A : P_i \\ \vec{V} : \overrightarrow{P_i Q_j} \end{cases} \tag{1.36}$$

Puisque l'ensemble des droites dans un espace de dimension 3 (le champ de vue du scanner) définit un espace de dimension 4, quatre coordonnées sont nécessaires pour localiser la LOR associée à un couple de détecteurs. Un système de coordonnées de l'espace de projection équivalent à celui en (A_p, A_q, P_i, Q_j), utilise ainsi les quatre coordonnées $(\phi, \rho, \lambda, \Delta)$ (voir les figures 1.19 et 1.20) telles que :

- les coordonnées axiales (λ, Δ) désignent un couple d'anneaux par l'écart entre anneaux $\Delta = (\lambda_q - \lambda_p)$ et la coordonnée axiale moyenne $\lambda = (\lambda_p + \lambda_q)/2$ avec λ_p et λ_q les coordonnées axiales des anneaux A_p et A_q.
- les coordonnées transaxiales (ϕ,ρ) désignent une LOR particulière pour chaque couple d'anneaux. L'angle ϕ définit un faisceau de LORs parallèles en positionnant l'axe tangentiel des détecteurs selon $\vec{i_\phi}$. La coordonnée ρ définit la position des deux détecteurs de la LOR sur cet axe.

Dans le repère de rotation $\mathcal{R}_{\phi,\vec{k}} = (\vec{i_\phi}, \vec{j_\phi}, \vec{k})$, la droite caractéristique d'une LOR est alors définie par :

$$\begin{cases} A : \rho \cdot \vec{i}_\phi - \sqrt{R_a^2 - \rho^2} \cdot \vec{j}_\phi + (\lambda - \dfrac{\Delta}{2}) \cdot \vec{k} \\ \vec{V} : 2\sqrt{R_a^2 - \rho^2} \cdot \vec{j_\phi} + \Delta \cdot \vec{k} \end{cases} \tag{1.37}$$

Et dans le repère $\mathcal{R} = (\vec{i}, \vec{j}, \vec{k})$, une LOR est définie par :

$$A : \begin{cases} x_a = \rho \cos\phi + \sqrt{R_a^2 - \rho^2} \sin\phi \\ y_a = \rho \sin\phi - \sqrt{R_a^2 - \rho^2} \cos\phi \\ z_a = \lambda - \frac{\Delta}{2} \end{cases} \quad \vec{V} : \begin{cases} x_v = -2\sqrt{R_a^2 - \rho^2} \sin\phi \\ y_v = 2\sqrt{R_a^2 - \rho^2} \cos\phi \\ z_v = \Delta \end{cases} \tag{1.38}$$

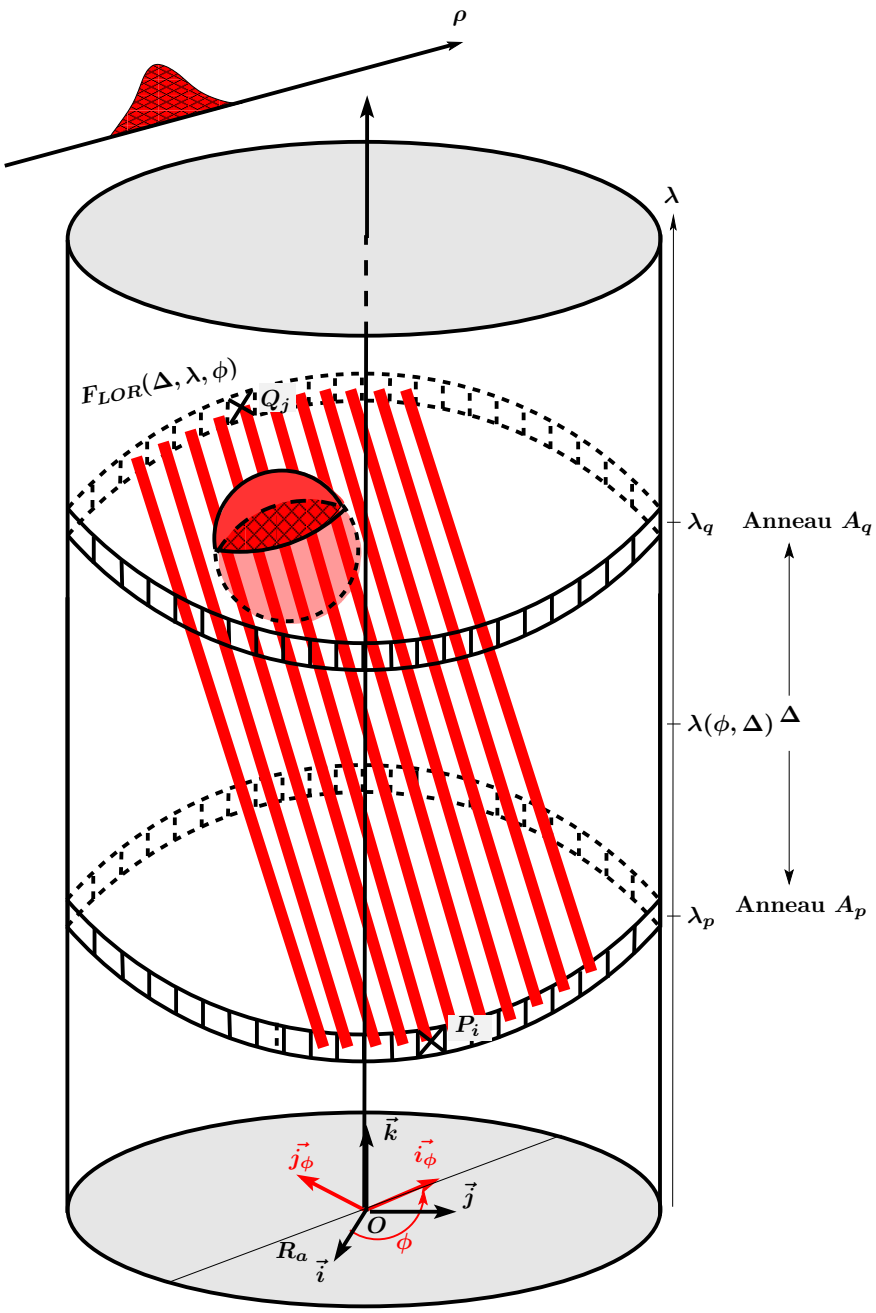


FIG. 1.19 – Paramétrisation des LORs (vue perspective). Le faisceau de LORs $\boldsymbol{F_{LOR}(\Delta, \lambda, \phi)}$ correspond aux LORs parallèles d'angle de direction transverse $\boldsymbol{\phi}$ reliant les détecteurs des anneaux $\boldsymbol{A_p}$ et $\boldsymbol{A_q}$. Ce couple d'anneaux est définit par l'écart axial $\boldsymbol{\Delta}$ et la coordonnée axiale moyenne $\boldsymbol{\lambda}$. Une LOR particulière (ici $\boldsymbol{P_i\vec{Q}_j}$) du faisceau $\boldsymbol{F_{LOR}(\Delta, \lambda, \phi)}$ est définie par sa coordonnée tangentielle $\boldsymbol{\rho}$.

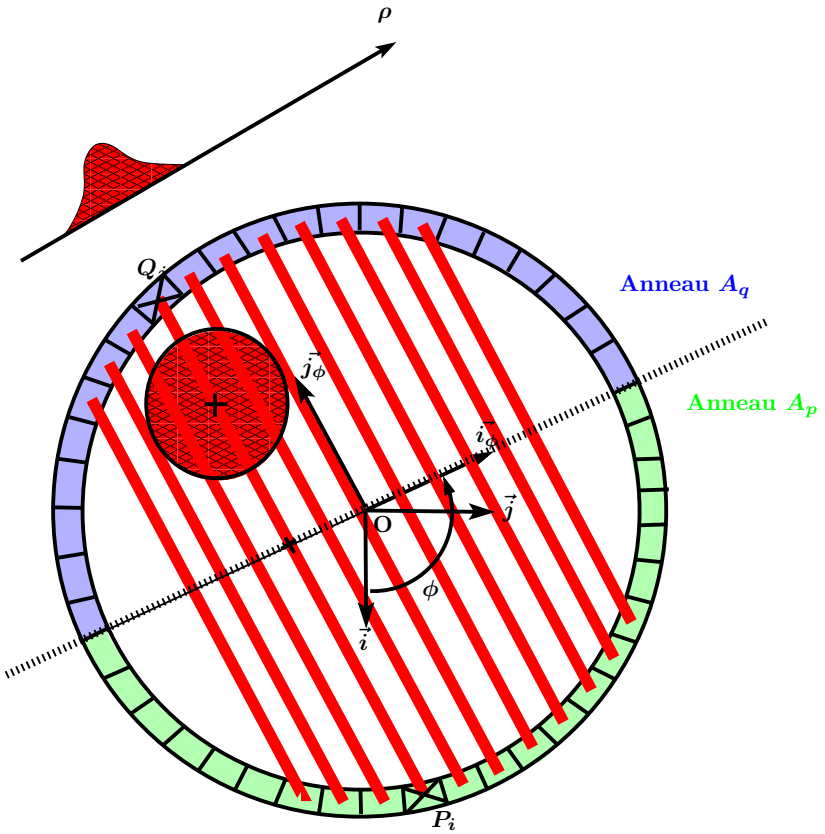


FIG. 1.20 – Paramétrisation des LORs (vue de dessus). Le faisceau de LORs $\boldsymbol{F_{LOR}(\Delta, \lambda, \phi)}$ de la figure 1.19 reliant les détecteurs de l'anneau $\boldsymbol{A_p}$ (en bleu) à ceux de l'anneau $\boldsymbol{A_q}$ (en vert) est représenté en vue de dessus.

5.2 Coordonnées discrètes

5.2.1 Espace image

Les dimensions du voxel sont liées à la résolution des scanners TEP, déterminée notamment par la taille des détecteurs élémentaires (mais pas uniquement). Les scanners HR+ et HRRT de Siemens ont des résolutions respectives de l'ordre de 5 mm et 3 mm. Les scanners $\boldsymbol{\mu TEP}$ peuvent atteindre une résolution de l'ordre de 1 mm, proche de la limite intrinsèque de l'imagerie TEP. Pour une taille des voxels fixée, les dimensions d'un volume (nombre de voxels) sont déterminées par la dimension axiale $\boldsymbol{H_{FOV}}$ et la dimension transverse $\boldsymbol{L_{FOV}}$ du champs de vue ou FOV (Field of View) du scanner. Ainsi les volumes des scanners HR+ et HHRT ont pour dimensions respectives $\mathbf{128^2 \cdot 63}$ voxels et $\mathbf{256^2 \cdot 207}$ voxels.

Dans la suite, nous utiliserons la correspondance suivante entre les coordonnées discrètes $(\mathtt{xn}, \mathtt{yn}, \mathtt{zn})$ et continues $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$:

$$\begin{cases} \boldsymbol{x} = (\mathtt{xn} - \mathtt{xn}_0) \cdot \boldsymbol{\delta_x} & \text{avec} \quad \mathbf{0} \leq \mathtt{xn} < \mathtt{xn}_{\max} \quad \boldsymbol{\delta_y} = \dfrac{\boldsymbol{L_{FOV}}}{\mathtt{xn}_{\max}} \\ \boldsymbol{y} = (\mathtt{yn} - \mathtt{yn}_0) \cdot \boldsymbol{\delta_y} & \text{avec} \quad \mathbf{0} \leq \mathtt{yn} < \mathtt{yn}_{\max} \quad \boldsymbol{\delta_y} = \dfrac{\boldsymbol{L_{FOV}}}{\mathtt{yn}_{\max}} \\ \boldsymbol{z} = (\mathtt{zn} - \mathtt{zn}_0) \cdot \boldsymbol{\delta_z} & \text{avec} \quad \mathbf{0} \leq \mathtt{zn} < \mathtt{zn}_{\max} \quad \boldsymbol{\delta_z} = \dfrac{\boldsymbol{H_{scan}}}{\mathtt{zn}_{\max}} \end{cases} \tag{1.39}$$

Deux remarques sur la discrétisation en pratique des coordonnées $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$:
- Le triplet $(\mathtt{xn}_0, \mathtt{yn}_0, \mathtt{zn}_0)$ décale l'échelle des coordonnées discrètes afin de les convertir en adresses mémoires (nombre positif).
- Deux pas d'échantillonnage sont utilisés : un transverse $\boldsymbol{\delta_{xy}} = \boldsymbol{\delta_x} = \boldsymbol{\delta_y}$ et un axial $\boldsymbol{\delta_z}$.

5.2.2 Espace de projection

Pour un scanner ayant $\boldsymbol{N_a}$ anneaux de rayon $\boldsymbol{R_a}$, une hauteur $\boldsymbol{H_{scan}}$, une distance entre anneaux de $\boldsymbol{d_a}$ et $\boldsymbol{N_d}$ capteurs par anneaux, les coordonnées de projection $(\boldsymbol{\rho}, \boldsymbol{\phi}, \boldsymbol{\lambda}, \boldsymbol{\Delta})$ peuvent être exprimées sous la forme discrète suivante :

$$\begin{cases} \boldsymbol{\Delta} = (\mathtt{delta} - \mathtt{delta}_0) \cdot \boldsymbol{\delta_\Delta} & \text{avec} \quad \mathbf{0} \leq \mathtt{delta} < \mathtt{delta}_{\max}, & \boldsymbol{\delta_\Delta} = \dfrac{\boldsymbol{H_{scan}}}{\boldsymbol{N_a}} = \boldsymbol{d_a} \\ \boldsymbol{\lambda} = (\mathtt{lambda} - \mathtt{lambda}_0) \cdot \boldsymbol{\delta_\lambda} & \text{avec} \quad \mathbf{0} \leq \mathtt{lambda} < \mathtt{lambda}_{\max}, & \boldsymbol{\delta_\lambda} = \dfrac{\boldsymbol{\delta_\Delta}}{\mathbf{2}} \\ \boldsymbol{\phi} = \mathtt{phi} \cdot \boldsymbol{\delta_\phi} & \text{avec} \quad \mathbf{0} \leq \mathtt{phi} < \mathtt{phi}_{\max}, & \boldsymbol{\delta_\phi} = \dfrac{\boldsymbol{\pi}}{\boldsymbol{N_d}} \\ \boldsymbol{\rho} = (\mathtt{rho} - \mathtt{rho}_0) \cdot \boldsymbol{\delta_\rho} & \text{avec} \quad \mathbf{0} \leq \mathtt{rho} < \mathtt{rho}_{\max}, & \boldsymbol{\delta_\rho} = \dfrac{\mathbf{2} \cdot \boldsymbol{R_a}}{\boldsymbol{N_d}} \end{cases} \tag{1.40}$$

Quelques remarques sur la discrétisation des coordonnées $(\boldsymbol{\rho}, \boldsymbol{\phi}, \boldsymbol{\lambda}, \boldsymbol{\Delta})$ adoptée en pratique :

- Les grandeurs $\texttt{delta}_0$, $\texttt{lambda}_0$ et $\texttt{rho}_0$ permettent de décaler les échelles de coordonnées de manière à ce que les coordonnées discrètes puissent être vues également comme des adresses mémoires (c'est-à-dire positif).
- Le pas d'échantillonnage de la coordonnée axiale $\boldsymbol{\lambda}$ est égal à la moitié de l'autre grandeur axiale $\boldsymbol{\Delta}$. En effet, $\boldsymbol{N_a}$ anneaux définissent $\boldsymbol{2 * N_a + 1}$ coordonnées axiales moyennes de couple d'anneaux.
- Le pas d'échantillonnage $\boldsymbol{\delta_z}$ est pris égal à celui de $\boldsymbol{\delta_\lambda}$. Ainsi les volumes ont pour dimension axiale $\boldsymbol{2 * N_a + 1}$ (par exemple 63 pour le scanner HR+ qui a 32 anneaux).
- Les LORs trop excentrées n'interceptant pas le champs de vue ($\boldsymbol{|\rho| > R_a/2}$), ne sont pas enregistrées. Ainsi seulement la moitié des LORs sont prises en compte ($\boldsymbol{-R_a/2 \leq \rho < R_a/2}$). De même les LORs ayant un angle d'inclinaison trop important ne sont pas enregistrées. Cela correspond à limiter la différence entre anneaux à une valeur maximum appelée mrd (*maximum ring difference*).
- Si les grandeurs $\boldsymbol{\Delta}$, $\boldsymbol{\lambda}$ et $\boldsymbol{\phi}$ sont dès l'acquisition échantillonnés uniformément, ce n'est pas le cas pour la grandeur $\boldsymbol{\rho}$. En effet, les détecteurs sont placés sur un arc de cercle et non sur l'axe $\boldsymbol{\vec{i_\phi}}$. La correspondance entre l'indice du détecteur et la grandeur physique $\boldsymbol{\rho}$ est plus complexe et fait appel aux fonction sinus et arcsinus. Différentes techniques de correction en arc sont utilisées pour corriger les données avant la reconstruction (voir annexe A).

5.3 Organisation des données en mode sinogramme

La matrice 4D des données du scanner est organisée en plusieurs matrices 3D appelées segments. Ces segments peuvent être stockés en mémoire de deux manières différentes selon le choix des indices des matrices 3D. De manière générale, l'adresse mémoire d'une donnée pour une matrice 3D de taille $\boldsymbol{I \cdot J \cdot K}$, correspond à $\boldsymbol{i + j * I + k * I * J}$.

En mode *viewgrams*, un segment 3D aura pour premier indice la coordonnée `rho` (indice $\boldsymbol{i}$), pour deuxième indice la coordonnée `lambda` (indice $\boldsymbol{j}$) et pour troisième indice `phi` (indice $\boldsymbol{k}$). Le segment 3D est alors stocké sous la forme d'une collection de matrices 2D d'indices $\boldsymbol{\rho}$ et $\boldsymbol{\lambda}$, chacune correspondant à une vue (*view*) du volume selon un angle de projection définit par $\boldsymbol{\phi}$ et $\boldsymbol{\Delta}$.

En mode sinogramme, un segment 3D a pour premier indice la coordonnée `rho` (indice $\boldsymbol{i}$), pour deuxième indice la coordonnée `phi` (indice $\boldsymbol{j}$) et pour troisième indice `lambda` (indice $\boldsymbol{k}$). Le segment 3D est alors stocké sous la forme d'une collection de matrices 2D d'indices $\boldsymbol{\rho}$ et $\boldsymbol{\phi}$, chacune correspondant au sinogramme d'un couple d'anneaux défini par $\boldsymbol{\lambda}$ et $\boldsymbol{\Delta}$.

Nous nous intéressons ici au mode sinogramme, utilisé en pratique pour le scanner HR+. Dans la pratique, les sinogrammes 2D subissent un entrelacement, une compression axiale (*span factor*) et transaxiale (*mashing factor*) afin de simplifier leur utilisation lors de la phase de reconstruction ([Fahey 02]).

5.3.1 Sinogrammes entrelacés

Les données du scanner sont organisées en sinogrammes $\mathcal{S}_{\texttt{delta,lambda}}$, matrices 2D qui regroupent l'ensemble des *bins* associés au couple d'anneaux de coordonnée axiale moyenne $\boldsymbol{\lambda}$ et d'écart entre anneau $\boldsymbol{\Delta}$. Un sinogramme range les *bins* de ce couple d'anneaux en lignes de projection regroupant tous les *bins* associés à un angle $\boldsymbol{\phi}$ donné (voir figure 1.21). Pour une scanner avec $\boldsymbol{N_d}$ détecteurs et un champ de vue égal à $\boldsymbol{R_a/2}$, un sinogramme sera formé de $\boldsymbol{N_d}$ lignes de projections (nombre d'échantillons de l'angle $\boldsymbol{\phi}$), constituées chacune de $\boldsymbol{N_d/4}$ *bins* (nombre d'échantillons de la coordonnée $\boldsymbol{\rho}$). Mais si les lignes de projection possèdent toutes le même pas d'échantillonnage $\boldsymbol{\delta_\rho}$, il existe un décalage de $\boldsymbol{\delta_\rho/2}$ entre les lignes de projection adjacentes comme cela est illustré sur la figure 1.21. Afin de s'affranchir de cet arrangement des données difficilement utilisable en pratique, on entrelace les lignes de projection adjacentes. Cela revient à multiplier par 2 le pas d'échantillonnage de $\boldsymbol{\phi}$ et à diviser par 2 le pas d'échantillonnage de $\boldsymbol{\rho}$. On a alors $\boldsymbol{\delta_\phi = \pi/N_d}$ et $\boldsymbol{\delta_\rho = 2 \cdot R_a/N_d}$. On peut toutefois remarquer que cela entraîne une approximation d'un angle $\boldsymbol{\delta_\phi/2 = \pi/(2 \cdot N_d)}$ sur certaines LORs (représentées en rouge sur la figure 1.21).

La compression transaxiale des données d'un sinogramme consiste à sommer $\boldsymbol{2^n}$ lignes de projection puis à stocker cette somme dans une seule ligne. Le facteur n désigne le taux de compression des données ou *mashing factor*. La résolution angulaire $\boldsymbol{\delta_\phi}$ est ainsi divisée par $\boldsymbol{2^n}$. Contrairement à l'entrelacement, il s'agit d'une compression avec perte.

5.3.2 Segments

Un segment regroupe tous les sinogrammes ayant le même écart entre anneaux $\boldsymbol{\Delta}$. De manière similaire à ce que nous avons observé pour les lignes adjacentes d'un sinogramme, tous les sinogrammes d'un segment ont le même pas d'échantillonnage $\boldsymbol{\delta_\lambda}$ sur l'axe axial ; mais il existe un décalage de $\boldsymbol{\delta_\lambda/2}$ entre chaque sinogramme adjacent ($\mathcal{S}_{\texttt{delta,lambda}_i}$ et $\mathcal{S}_{\texttt{delta,lambda}_{i+1}}$). Dans ce cas pour simplifier le stockage des données, les sinogrammes ne seront pas entrelacés mais sommer selon leur parité : les sinogrammes "impairs" seront sommés deux par deux pour former des sinogrammes dit "indirects" et les sinogrammes "pairs" seront laissés tel quel et formeront les sinogrammes "directs" (voir figures 1.22 et 1.23). Cela revient à multiplier par deux le pas d'échantillonnage en $\boldsymbol{\lambda}$ et à diviser par deux celui en $\boldsymbol{\Delta}$. De plus il s'agit bien d'une compression axiale des données du scanner car des sinogrammes d'écart entre anneaux $\boldsymbol{\Delta}$ proche (les sinogrammes "impairs") sont sommés et l'on attribue au sinogramme résultant un $\boldsymbol{\Delta}$ moyen (voir figures 1.22 et 1.23). Le *span factor* indique le taux de compression des sinogrammes directs et indirects. La compression la plus simple des données est équivalent à un *span factor* de $\boldsymbol{3 = 1 + 2}$, car les sinogrammes directs sont compressés d'un facteur 1 (pas de compression) et les sinogrammes indirects d'un facteur 2. Une compression correspondant à un span de 5 est présentée sur la figure 1.23.

Par exemple pour le scanner HR+, un span de $\boldsymbol{9 = 4 + 5}$ est utilisé en pratique ce qui correspond à compresser d'un facteur 4 les sinogrammes directs et d'un facteur 5 les sinogrammes indirects. La différence maximum autorisée entre anneaux

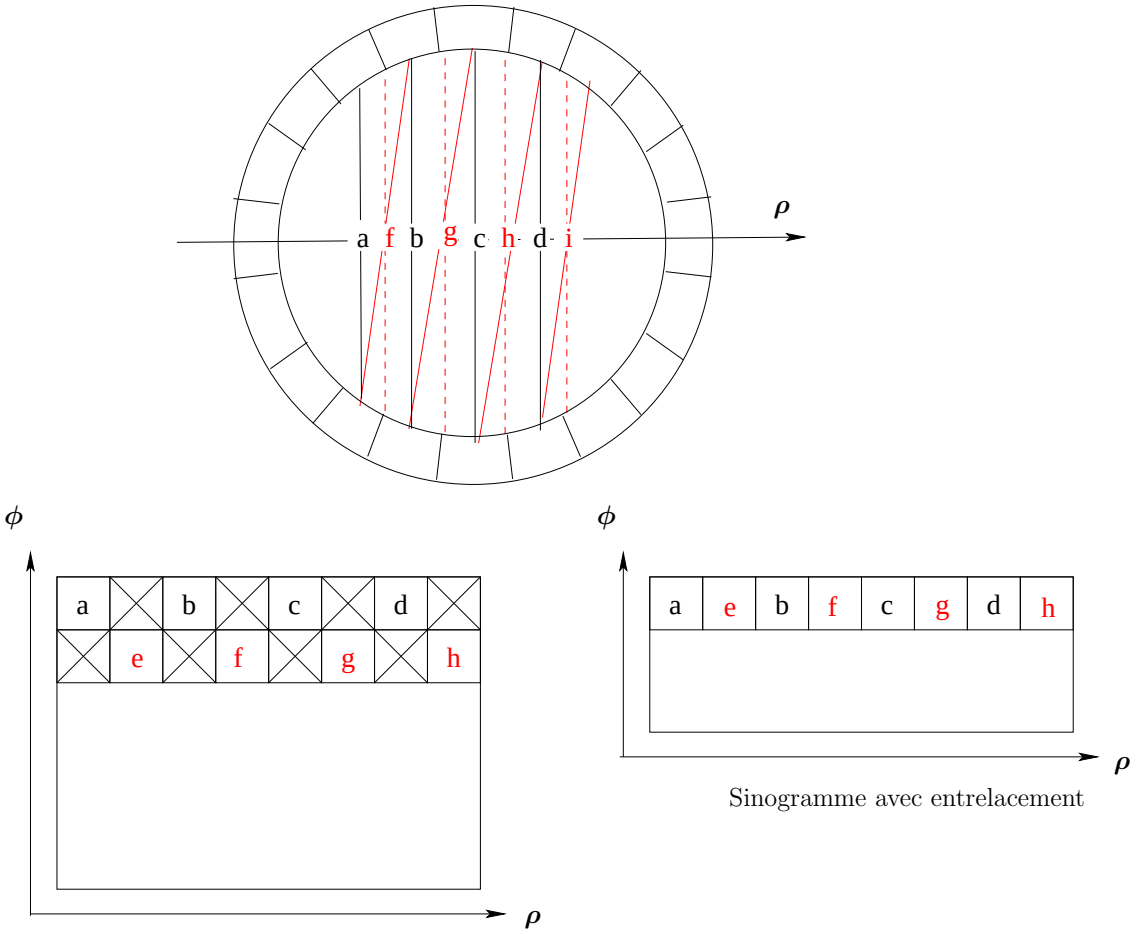


FIG. 1.21 – L'entrelacement des données double virtuellement la résolution sur ρ et diminue de moitié celle sur ϕ. Les LORs réelles sont en traits pleins et les LORs équivalentes après entrelacement en pointillés

(mrd) étant de 22, il existe alors 5 segments correspondant à un écart entre anneaux de $-18d_a, 9d_a, 0, +9d_a, +18d_a$ appelés respectivement segments $-2, -1, 0, +1, +2$ (voir annexe B).

5.4 Expression discrète de la rétroprojection

En utilisant le système de coordonnées discrètes (`rho`, `phi`, `lambda`, `delta`), nous pouvons définir les opérateurs discrets de rétroprojection 2D et 3D dérivant des opérateurs continus (équations 1.9 et 1.13).

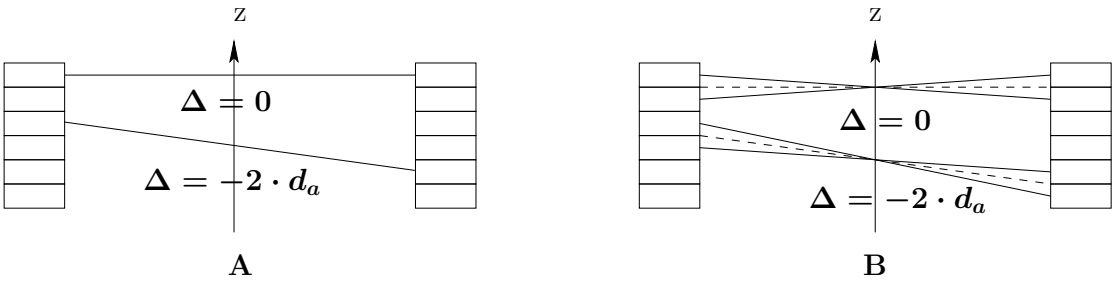


FIG. 1.22 – Compression axiale avec un span de 3 : (A) les sinogrammes "paires" ($\Delta = 0$ et $\Delta = 2 \cdot d_a$) correspondent aux sinogrammes directs ($\Delta = 0$ et $\Delta = 2 \cdot d_a$), (B) les sinogrammes "impaires" (les couples ($\Delta = -d_a, \Delta = +d_a$) et ($\Delta = -d_a, \Delta = -3 \cdot d_a$) en trait plein) sont sommés deux à deux pour former un sinogramme indirect ($\Delta = 0$ et $\Delta = -2 \cdot d_a$ en pointillés).

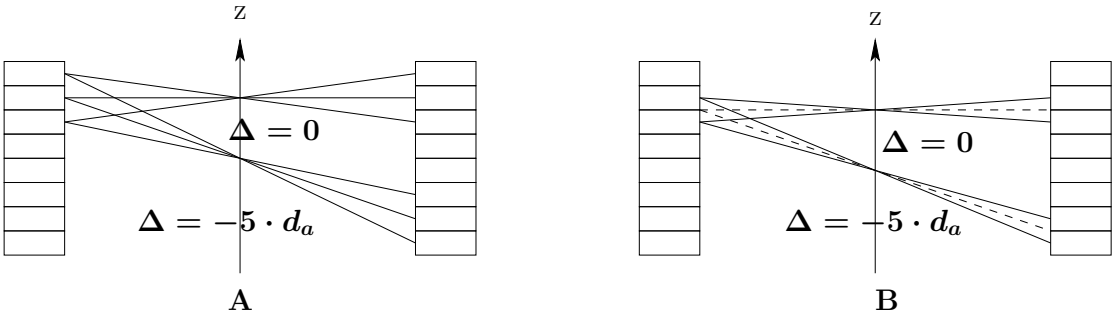


FIG. 1.23 – Compression axiale avec un span de 5 : (A) pour deux sinogrammes directs ($\Delta = 0$ et $\Delta = -5 \cdot d_a$), (B) pour deux sinogrammes indirects ($\Delta = 0$ et $\Delta = -5 \cdot d_a$)

5.4.1 Intégrale discrète

L'intégrale continue de l'opérateur de rétroprojection (voir équation **1.13**) est transcrite en une intégrale discrète de la manière suivante :

$$\boldsymbol{RPf(\vec{r})} = \sum_{\texttt{theta}=-\texttt{theta}_{\max}}^{+\texttt{theta}_{\max}} \sum_{\texttt{psi}=0}^{\texttt{psi}_{\max}} \boldsymbol{p(u_{\parallel}, v_{\parallel}, \psi_{\texttt{psi}}, \theta_{\texttt{theta}}) \cdot \cos(\theta_{\texttt{theta}}) \cdot \delta_{\theta} \cdot \delta_{\psi}}$$

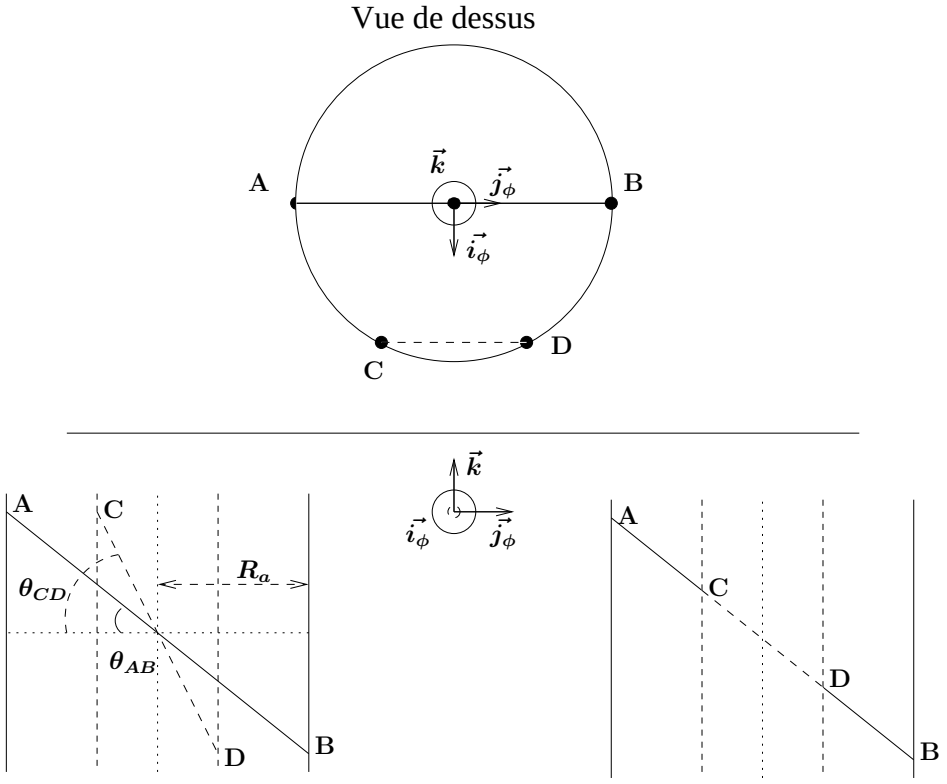
$$\text{avec} \begin{cases} \psi_{\texttt{psi}} = \texttt{psi} \cdot \boldsymbol{\delta_{\psi}} \\ \theta_{\texttt{theta}} = \texttt{theta} \cdot \boldsymbol{\delta_{\theta}} \end{cases} \tag{1.41}$$

Mais comme nous l'avons vu précédemment, les données du scanner ne correspondent pas au système de coordonnées $(u_{\parallel}, v_{\parallel}, \psi, \theta)$ mais à celui en $(\rho, \phi, \lambda, \Delta)$. Si les angles ϕ et ψ sont équivalents, ce n'est pas le cas entre l'angle d'inclinaison axiale des LORs θ et l'écart entre anneaux Δ. Afin d'obtenir une correspondance directe entre l'angle θ et

$\boldsymbol{\Delta}$, l'approximation suivante est faite :

$$\tan(\theta) = \frac{\Delta}{2\sqrt{R_a^2 - \rho^2}} \approx \frac{\Delta}{2 \cdot R_a} \tag{1.42}$$

Cette approximation correspond à placer les détecteurs non pas sur des arcs de cercle définis par les anneaux du scanner mais sur les axes $\boldsymbol{\rho}$ de projection (voir figure 1.24). Les *bins* d'un faisceau de LORs parallèles $\boldsymbol{F_{LOR}(\Delta, \lambda, \phi)}$ sont ainsi considérés coplanaires.



(a) Coupes Axiales sans Approximation (b) Coupes Axiales avec Approximation

FIG. 1.24 – Pour rapprocher le modèle d'acquisition de la transformée de Radon, une approximation angulaire est faite. Dans les schémas 1 et 2, des coupes axiales sont superposées pour différents $\boldsymbol{\rho}$ (trait plein, trait pointillé). Schéma (a) : pas d'approximation sur $\vec{V}$, le vecteur directeur des LORs. Schéma (b) : $\vec{V}$ est pris constant selon $\boldsymbol{\rho}$.

Après le changement de variable en $\boldsymbol{\Delta}$, un jacobien $\boldsymbol{J_\Delta}$ apparaît dans l'expression de l'intégrale discrète (voir équation 1.41).

$$RPf(\vec{r}) = \sum_{\texttt{delta}=0}^{\texttt{delta}_{max}-1} \sum_{\texttt{phi}=0}^{\texttt{phi}_{max}-1} p(\rho, \lambda, \phi_{\texttt{phi}}, \Delta_{\texttt{delta}}) \cdot J_{\Delta} \cdot \delta_{\phi} \cdot \delta_{\Delta}$$

$$\text{avec} \begin{cases} \phi_{\texttt{phi}} &= \texttt{phi} \cdot \delta_{\phi} \\ \Delta_{\texttt{delta}} &= \texttt{delta} \cdot \delta_{\Delta} \\ J_{\Delta} &= \dfrac{4 \cdot R_a^2}{(4 \cdot R_a^2 + \Delta^2)^{(3/2)}} \end{cases} \tag{1.43}$$

5.4.2 Rétroprojection *voxel-driven* avec interpolation linéaire

Afin de rétroprojeter des données de projection discrètes $(\texttt{rho}, \texttt{phi}, \texttt{lambda}, \texttt{delta})$ dans un volume discret $f(\texttt{xn}, \texttt{yn}, \texttt{zn})$, deux stratégies existent. Soit on rétroprojette toutes les données de projection dans un voxel à la fois en sommant les *bins* correspondant à ce voxel (*voxel-driven*), soit on rétroprojette *bin* après *bin* les données de projection dans le volume entier (*ray-driven*).

Nous nous intéresserons dans cette étude à la rétroprojection *voxel-driven* qui est la plus répandue en pratique et dans la littérature. Elle consiste à rechercher pour chaque pas d'intégration δ_{ϕ} et δ_{Δ}, la projection $(\texttt{rho}, \texttt{psi}, \texttt{lambda}, \texttt{delta})$ correspondant au voxel $(\texttt{xn}, \texttt{yn}, \texttt{zn})$. Les expressions de $\rho_{(x,y,z)}(\phi)$ et $\lambda_{(x,y,z)}(\phi, \lambda)$ sont alors nécessaires afin de désigner la LOR passant par le voxel (x, y, z) :

$$\begin{cases} \rho_{(x,y,z)}(\phi) &= x \cdot \cos\phi + y \cdot \sin\phi \\ \lambda_{(x,y,z)}(\phi, \Delta) &= z - \big(-x \cdot \sin\phi + y \cdot \cos\phi \big) \cdot \dfrac{\Delta}{2 \cdot R_a} \end{cases} \tag{1.44}$$

La projection $p(\rho_{(x,y,z)}, \lambda_{(x,y,z)}, \Delta_{\texttt{delta}}, \phi_{\texttt{phi}})$ est ensuite obtenue par interpolation bilinéaire à partir de données discrètes du scanner $\mathcal{S}_{\texttt{lambda,delta}}(\texttt{rho}, \texttt{phi})$.

5.4.3 Expression discrète de la rétroprojection 2D

Pour chaque voxel $(\texttt{xn}, \texttt{yn})$, la somme de *bins* suivante est effectuée :

$$f(\texttt{xn}, \texttt{yn}) = \sum_{\texttt{phi}=0}^{\texttt{phi}_{max}-1} \mathcal{S}(\texttt{rho}, \texttt{phi}) \cdot \delta_{\phi}$$

$$\text{avec} \begin{cases} \texttt{rho} &= \dfrac{\rho}{\delta_{\rho}} + \texttt{rho}_0 = \dfrac{x \cdot \cos\phi + y \cdot \sin\phi}{\delta_{\rho}} + \texttt{rho}_0 \\ &= \dfrac{\delta_{xy}}{\delta_{\rho}} \cdot \big((\texttt{xn} - \texttt{xn}_0) \cdot \cos\phi + (\texttt{yn} - \texttt{yn}_0) \cdot \sin\phi \big) + \texttt{rho}_0 \end{cases} \tag{1.45}$$

Le *bin* $\boldsymbol{\mathcal{S}}(\mathtt{rho},\mathtt{phi})$ est calculé par interpolation linéaire à partir de la valeur entière $\mathtt{rho_e}$ et décimale $\boldsymbol{\epsilon_\rho}$ de la coordonnée $\boldsymbol{\rho}$:

$$\begin{aligned}&\boldsymbol{\mathcal{S}}(\mathtt{rho},\mathtt{phi}) = (1-\boldsymbol{\epsilon_\rho})\cdot\boldsymbol{\mathcal{S}}(\mathtt{rho_e},\mathtt{phi}) + \boldsymbol{\epsilon_\rho}\cdot\boldsymbol{\mathcal{S}}(\mathtt{rho_e}+1,\mathtt{phi})\\ &\text{avec}\quad \mathtt{rho} = \mathtt{rho_e} + \boldsymbol{\epsilon_\rho}\end{aligned} \tag{1.46}$$

5.4.4 Expression discrète de la rétroprojection 3D

Pour chaque voxel $(\mathtt{xn},\mathtt{yn},\mathtt{zn})$, la somme de *bins* suivante est effectuée :

$$\boldsymbol{f}(\mathtt{xn},\mathtt{yn},\mathtt{zn}) = \sum_{\mathtt{delta}=\mathbf{0}}^{\mathtt{delta_{max}}-\mathbf{1}} \sum_{\mathtt{phi}=\mathbf{0}}^{\mathtt{phi_{max}}-\mathbf{1}} \boldsymbol{\mathcal{S}}_{\mathtt{lambda,delta}}(\mathtt{rho},\mathtt{phi})\cdot\boldsymbol{J_\Delta}\cdot\boldsymbol{\delta_\phi}\cdot\boldsymbol{\delta_\Delta}$$

$$\textbf{avec}\quad \begin{cases} \mathtt{rho} &= \dfrac{\boldsymbol{\rho}}{\boldsymbol{\delta_\rho}} + \mathtt{rho_0} = \dfrac{\boldsymbol{x}\cdot\cos\boldsymbol{\phi} + \boldsymbol{y}\cdot\sin\boldsymbol{\phi}}{\boldsymbol{\delta_\rho}} + \mathtt{rho_0} \\ &= \dfrac{\boldsymbol{\delta_{xy}}}{\boldsymbol{\delta_\rho}}\cdot\big((\mathtt{xn}-\mathtt{xn_0})\cdot\cos\boldsymbol{\phi} + (\mathtt{yn}-\mathtt{yn_0})\cdot\sin\boldsymbol{\phi}\big) + \mathtt{rho_0} \\ \mathtt{lambda} &= \dfrac{\boldsymbol{\lambda}}{\boldsymbol{\delta_\lambda}} + \mathtt{lambda_0} = \dfrac{\boldsymbol{z} - \big(-\boldsymbol{x}\cdot\sin\boldsymbol{\phi} + \boldsymbol{y}\cdot\cos\boldsymbol{\phi}\big)\cdot\dfrac{\boldsymbol{\Delta}}{\mathbf{2}\cdot\boldsymbol{R_a}}}{\boldsymbol{\delta_\lambda}} + \mathtt{lambda_0} \\ &= \dfrac{\boldsymbol{\delta_z}}{\boldsymbol{\delta_\lambda}}\cdot(\mathtt{zn}-\mathtt{zn_0}) + \mathtt{lambda_0} \\ &\quad + \dfrac{\boldsymbol{\delta_{xy}}}{\boldsymbol{\delta_\lambda}}\cdot\big((\mathtt{xn}-\mathtt{xn_0})\cdot\sin\boldsymbol{\phi} - (\mathtt{yn}-\mathtt{yn_0})\cdot\cos\boldsymbol{\phi}\big)\cdot\dfrac{\boldsymbol{\Delta}}{\mathbf{2}\cdot\boldsymbol{R_a}} \\ \boldsymbol{J_\Delta} &= \dfrac{\mathbf{4}\cdot\boldsymbol{R_a^2}}{(\mathbf{4}\cdot\boldsymbol{R_a^2} + \boldsymbol{\Delta^2})^{(3/2)}} \end{cases} \tag{1.47}$$

Le *bin* $\boldsymbol{\mathcal{S}}_{\mathtt{lambda,delta}}(\mathtt{rho},\mathtt{phi})$ est calculé par interpolation bilinéaire à partir des valeurs entières ($\mathtt{rho_e}$,$\mathtt{lambda_e}$) et décimales ($\boldsymbol{\epsilon_\rho}$,$\boldsymbol{\epsilon_\lambda}$) des coordonnées $\boldsymbol{\rho}$ et $\boldsymbol{\lambda}$:

$$\begin{aligned}\boldsymbol{\mathcal{S}}_{\mathtt{lambda,delta}}(\mathtt{rho},\mathtt{phi}) = {} & (1-\boldsymbol{\epsilon_\lambda})\cdot(1-\boldsymbol{\epsilon_\rho}) \cdot \boldsymbol{\mathcal{S}}_{\mathtt{lambda_e,delta}}(\mathtt{rho_e},\mathtt{phi}) \\ + {} & (1-\boldsymbol{\epsilon_\lambda})\cdot\boldsymbol{\epsilon_\rho} \cdot \boldsymbol{\mathcal{S}}_{\mathtt{lambda_e,delta}}(\mathtt{rho_e}+1,\mathtt{phi}) \\ + {} & \boldsymbol{\epsilon_\lambda}\cdot(1-\boldsymbol{\epsilon_\rho}) \cdot \boldsymbol{\mathcal{S}}_{\mathtt{lambda_e}+1,\mathtt{delta}}(\mathtt{rho_e},\mathtt{phi}) \\ + {} & \boldsymbol{\epsilon_\lambda}\cdot\boldsymbol{\epsilon_\rho} \cdot \boldsymbol{\mathcal{S}}_{\mathtt{lambda_e}+1,\mathtt{delta}}(\mathtt{rho_e}+1,\mathtt{phi})\end{aligned}$$

$$\text{avec}\quad \begin{cases} \mathtt{rho} = \mathtt{rho_e} + \boldsymbol{\epsilon_\rho} \\ \mathtt{lambda} = \mathtt{lambda_e} + \boldsymbol{\epsilon_\lambda} \end{cases} \tag{1.48}$$

6 Analogies avec la tomographie CT

Dans cette section, nous comparons la rétroprojection en faisceaux coniques utilisée en tomographie CT avec celle en faisceaux parallèles 3D utilisée en tomographie TEP.

6.1 Rétroprojection en faisceaux coniques

Les scanners CT possède un émetteur de rayons X qui tourne autour du patient en même temps que les récepteurs chargés de mesurer l'intensité des rayons X après leur passage dans le corps. A chaque position de la source sur son orbite autour du patient, un faisceau conique de rayons X traverse le patient (voir figure 1.25).

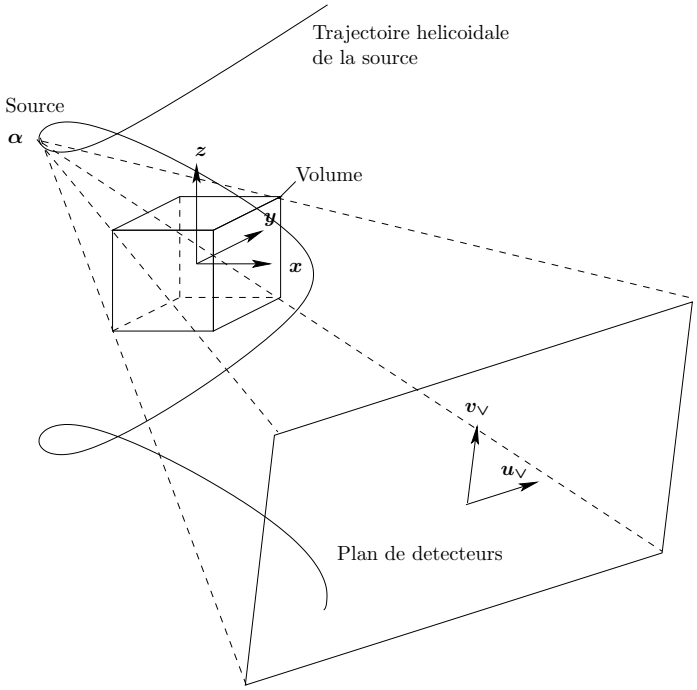


FIG. 1.25 – Acquisition en tomographie CT avec un faisceau conique sur les plans de détecteurs et une trajectoire hélicoïdale de la source.

L'algorithme de Feldkamp, Davis and Kress (FDK) est un algorithme de rétroprojection filtré largement utilisé en tomographie CT. Il permet de reconstruire le volume à partir des mesures en faisceaux coniques avec une source décrivant une orbite cylindrique autour du patient. Cette géométrie en faisceaux coniques demande lors de la rétroprojection une transformation perspective faisant intervenir la distance entre le volume et la source et celle entre la source et le plan de détecteurs.

L'équation 1.49 formule l'expression continue de l'algorithme FDK. Les coordonnées en faisceaux coniques $(\boldsymbol{u}_\vee, \boldsymbol{v}_\vee)$ sont calculées selon l'équation 1.50 à l'aide de matrices $\boldsymbol{c_{ij}}$ de dimensions 3*3 et fonction de $\boldsymbol{\alpha}$. Ces équations sont inspirées de celles de

[Kachelriess 07]

$$f_{\text{CT}}(\vec{r}) = \int p_{\text{CT}}\big(\alpha, u_{\vee}(\alpha,\vec{r}), v_{\vee}(\alpha,\vec{r})\big) \cdot w(\alpha,\vec{r})^2 \cdot d\alpha \tag{1.49}$$

$$\begin{cases} u_{\vee}(\alpha,\vec{r}) &= (c_{00}x + c_{01}y + c_{02}z + c_{03}) \cdot w(\alpha,\vec{r}) \\ v_{\vee}(\alpha,\vec{r}) &= (c_{10}x + c_{11}y + c_{12}z + c_{13}) \cdot w(\alpha,\vec{r}) \end{cases}$$

avec (1.50)

$$c_{ij} = c_{ij}(\alpha)$$
$$w(\alpha,\vec{r}) = \frac{1}{c_{20}\cdot x + c_{21}\cdot y + c_{22}\cdot z + c_{23}}$$

6.2 Reformulation de la rétroprojection en faisceaux parallèles 3D

Comme nous l'avons vu précédemment, la rétroprojection 3D en TEP utilise des faisceaux de LORs parallèles, correspondant à des projections sur des plans inclinés d'angle axial θ et d'angle transverse ψ (voir figure 1.8). Les projections sont repérés sur ces plans par les coordonnées $(u_{\parallel}, v_{\parallel})$. Ce système de coordonnées de projection en $(\theta, \psi, u_{\parallel}, v_{\parallel})$ est équivalent à celui en $(\Delta, \phi, \lambda, \rho)$, utilisé en pratique par les scanners TEP pour ordonner les *bins*. L'équation 1.51 illustre cette équivalence.

$$\begin{cases} \theta &\simeq \arctan\left(\frac{\Delta}{2\cdot R_a}\right) \\ \psi &= \phi - \frac{\pi}{2} \\ u_{\parallel} &= \rho \\ v_{\parallel} &= \cos(\theta)\cdot\lambda \end{cases} \tag{1.51}$$

Afin de comparer la rétroprojection en faisceaux parallèle utilisée en TEP avec la rétroprojection en faisceaux coniques, nous réutilisons ci-dessous l'équation 1.13 de la rétroprojection 3D :

$$f_{\text{PET}}(\vec{r}) = \int\int p_{\text{PET}}\big(\theta, \psi, u_{\parallel}(\phi,\vec{r}), v_{\parallel}(\psi,\theta,\vec{r})\big) d\psi d\theta \tag{1.52}$$

Les coordonnées en faisceaux parallèles $(u_{\parallel}, v_{\parallel})$ sont alors calculés selon l'équation 1.53 à l'aide d'une matrice a_{ij} de dimensions 2*3 et fonction de ψ et θ.

$$\begin{cases} u_{\parallel}(\psi,\vec{r}) &= a_{00}x + a_{01}y + a_{03} \\ v_{\parallel}(\psi,\theta,\vec{r}) &= a_{10}x + a_{11}y + a_{12}z + a_{13} \end{cases} \tag{1.53}$$

avec

$$a_{ij} = a_{ij}(\psi,\theta)$$

6.3 Comparaison entre rétroprojection en faisceaux parallèles et en faisceaux coniques

Bien que le calcul des coordonnées soit plus complexe pour la rétroprojection en faisceaux coniques à cause de la transformation perspective (le calcul des coordonnées demande une division par un poids sur les distances, $\boldsymbol{w(\alpha, \vec{r})}$), les calculs peuvent être considérés semblables à ceux de la rétroprojection en faisceaux parallèles. En effet, la somme sur le paramètre de trajectoire $\boldsymbol{\alpha}$ pour la tomographie CT est équivalent à la somme sur les angles $\boldsymbol{\psi}$ et les segments d'angle $\boldsymbol{\theta}$; et les calculs des coordonnées de projection $\boldsymbol{(u_\vee, v_\vee)}$ et $\boldsymbol{(u_\parallel, v_\parallel)}$ sont proches. Par ailleurs, les deux rétroprojection accèdent à chaque étape de sommation à une seule donnée de projection (4 avec l'interpolation bilinéaire). Ainsi, même si la séquence de données de projection accédées est légèrement différente en tomographie CT à cause de la transformation perspective et mise à part le fait que le nombre et la taille des plans de projection sont différents, les deux tomographies ont un schéma général d'accès mémoire pratiquement équivalent.

7 Conclusion

Nous avons décrit dans ce chapitre les mesures et les calculs définis par la tomographie TEP et nous avons observé qu'ils restent proches de ceux utilisés en tomographie CT.

D'une part, les calculs des différentes méthodes de reconstruction nécessitent un nombre important d'opérations correspondant en grande partie à ceux de la rétroprojection. Pour les méthodes analytiques (rétroprojection filtrée), cet opérateur représente l'étape principale de calcul comme l'a constaté [Heigl 07] avec **71%** du temps de reconstruction attribué à la seule étape de rétroprojection et **19%** seulement pour le filtrage, le reste correspondant aux pré et post traitements. Pour les algorithmes itératifs tel que l'algorithme EM, le calcul matriciel correspondant à l'étape de rétroprojection peut être approximé par une simple rétroprojection du sinogramme de comparaison $\boldsymbol{(y_i / \hat{y}_i)}$. Ainsi, la rétroprojection représente la moitié (en itératif) ou plus de la moitié (en analytique) des calculs en reconstruction TEP. Cet opérateur effectue pour $\boldsymbol{N^3}$ voxels du volume, la somme de $\boldsymbol{O(N)}$ éléments du sinogramme. Sa complexité est donc en $\boldsymbol{0(N^4)}$, ce qui représente un grand nombre d'opérations et donc un temps important de reconstruction. De plus, c'est la reconstruction 3D, plus coûteuse en calcul que la rétroprojection 2D qui sera utilisée en pratique car elle seule, permet d'améliorer la qualité de reconstruction grâce à l'utilisation de plusieurs segments du sinogramme 4D. La rétroprojection 3D représente ainsi une priorité pour l'effort d'accélération de la reconstruction tomographique TEP.

D'autre part, les mesures des scanners TEP sont de taille importante, et cela malgré un important effort de sous-échantillonnage des données (compressions axiales et transverses). Ainsi les données du sinogramme 4D représentent plusieurs dizaines de Mo (environ 20 Mo pour le scanner HR+ et 200 Mo pour le scanner HRRT). Une mémoire de type SDRAM devra donc être utilisée pour stocker l'ensemble des données du scanner. Or puisque chaque mise à jour d'un voxel (correspondant à une somme de l'équation

1.47) nécessite l'accès aux projections du voxel, des accès nombreux et dispersés dans la mémoire SDRAM de stockage du sinogramme devront être effectués. L'accès à la mémoire risque ainsi de constituer un goulot d'étranglement durant la reconstruction.

Dans la chapitre suivant, nous verrons les différentes techniques d'accélération décrites dans la littérature pour accélérer de manière générale les reconstructions tomographiques TEP et CT et en particulier les rétroprojections en faisceaux parallèles et en faisceaux coniques. Ces techniques apportent diverses solutions au problème posé par l'accès à la mémoire externe.

Chapitre 2

Accélération algorithmique, logicielle et matérielle en reconstruction tomographique

L'accélération de calcul peut être vue sous trois angles : algorithmique, logicielle et matérielle. Ses trois techniques sont bien souvent interdépendantes, chacune laissant plus ou moins de degrés de liberté aux autres. En général, le matériel est figé avec l'utilisation d'un processeur classique (CPU). Dans ce cas, les techniques d'accélération consistent à réduire la complexité de l'algorithme (accélération algorithmique), ou bien à écrire un code s'exécutant en un temps minimal sur le processeur (accélération logicielle). Lorsque l'accélération des calculs est une pré-occupation importante alors l'accélération matérielle est employée. Elle peut se limiter à la simple parallélisation de l'algorithme sur un réseau de PCs ou sur une machine multiprocesseurs ou bien encore au choix d'un processeur spécifique pré-existant comme le processeur IBM Cell ou les processeurs graphiques GPU. Mais elle peut aussi aboutir à la conception d'une architecture sur mesure lorsque les technologies ASIC et FPGA sont utilisées. A ce titre, un compromis entre la souplesse des solutions logicielles et la puissance de calcul des blocs matériels dédiés, est possible avec l'utilisation des SoPCs (System on Programmable Chip). Les SoPCs possèdent en effet à la fois de la logique programmable et des processeurs embarqués. Dans ce cas, un partionnement logiciel/matériel devra être mis en place pour définir les limites entre l'architecture matérielle et le processeur quitte à ce que ce dernier disparaisse totalement de l'architecture.

Dans ce chapitre, nous nous demanderons si les techniques d'accélération algorithmique et/ou logicielle de l'état de l'art utilisant un simple PC sont suffisantes ou non pour traiter en un temps satisfaisant les données provenant des scanners CT et TEP. Ensuite, nous exposerons les différentes stratégies de parallélisation à notre disposition pour obtenir un facteur supplémentaire d'accélération. Enfin nous passerons en revue plusieurs mises en oeuvre de ces stratégies sur différentes machines parallèles : réseaux de PC, machines multiprocesseurs, processeurs spécifiques (GPU et Cell) et les architectures matérielles de type FPGA/ASIC.

1 Accélération sur PC

Dans cette section, nous nous demanderons si le temps de reconstruction obtenu à l'aide d'un simple PC est suffisamment réduit par les techniques standards d'accélération logicielle et/ou algorithmique.

Après une présentation de la puissance de calcul qu'offre l'architecture des CPUs, nous exposerons les différentes techniques d'optimisations logicielles ou d'accélérations algorithmiques à notre disposition. Nous verrons ensuite que pour toutes ces techniques un compromis doit être trouvé entre l'accélération, la qualité de reconstruction et la complexité de mise en oeuvre. Nous confronterons enfin les accélérations ainsi obtenues aux besoins actuels et futurs en imagerie médicale TEP et CT.

1.1 Puissance de calcul des CPUs

Le débit de calcul théorique des processeurs, calculé en GFLOPS (*Giga FLoating point OPeration per Second*) est la mesure standard pour évaluer la puissance de calcul des processeurs. Par exemple, un Pentium 4 à 3.2 GHz atteint 3.2 GFLOPS et un Xeon double coeur à 3.06 GHz atteint 6.12 GFLOPS. Selon cette mesure, les performances des CPUs depuis plusieurs dizaines d'années doublent tous les deux ans, et sont ainsi proches de suivre la loi de Moore (doublement tous les 18 mois). Longtemps lié à l'augmentation de la fréquence d'horloge des CPUs, ce rythme de croissance est à présent obtenu par l'augmentation du nombre de coeurs par processeur. Ces importants débits de calcul sont obtenus notamment grâce à une architecture en pipeline (14 étages pour la dernière architecture Core d'Intel) permettant de réduire la durée d'un cycle. Dans cette étude, nous nous intéresserons aux CPUs ne possédant qu'une seule unité de calcul pour les flottants (*FPU*). Le débit maximum de calcul en flottant est alors simplement obtenu à partir de la fréquence d'horloge et du nombre de coeurs. Si on ajoute à ce calcul de performance, l'accélération potentielle obtenue grâce aux instructions vectorielles (*Streaming SIMD Extensions* d'*intel*) qui permettent de traiter 4 flottants simple précision par cycle, les performances théoriques du Pentium 4 atteignent alors 12.8 GFLOPS et celle du Xeon 24.48 GFLOPS.

Ce débit de calcul théorique toutefois n'est qu'une indication du potentiel de chaque architecture. Ainsi afin d'effectuer une véritable comparaison de performances, des *benchmarks* implémentent sur les principales machines existantes sur le marché, des algorithmes utilisés couramment ou spécifiquement. Ils obtiennent ainsi des mesures pratiques en GFLOPS de la puissance de calcul de chaque machine. Le *benchmarks* LINPACK teste par exemple, les performances pour la résolution d'équations linéaires. Afin de se rapprocher du débit de calcul maximum, les implémentations des *benchmarks* utilisent notamment comme techniques d'optimisations logicielles, le déroulage de boucle et les instructions vectorielles. Le déroulage de boucle permet non seulement de réduire le nombre de Load/Store mais aussi de profiter au mieux des ressources parallèles de calcul en réduisant l'effet des latences Load/Store et d'obtention de résultats des opérateurs pipelinés. Elle est prise en charge par le compilateur ou définie explicitement par le développeur. Par ailleurs, pour les calculs se prêtant bien à une exécution via des instructions

vectorielles ([**?**],[Yu 01]), une accélération significative peut être obtenue (facteur 4 pour des instructions portant sur des registres 128 bits soit 4 floats). Cette dernière technique est toutefois loin d'être couramment utilisée. En effet, sa prise en charge par les compilateurs de type *gcc* n'étant pas encore très développée, elle est ainsi souvent laissée à la charge du développeur.

1.2 Accélération de la rétroprojection

Après une présentation rapide des techniques d'accélération logicielle classiquement utilisées, nous exposerons les principales techniques d'accélération algorithmique utilisées pour réduire la complexité de l'opérateur de rétroprojection.

1.2.1 Accélération logicielle

Outre le déroulage de boucles et l'utilisation des instructions vectorielles, la manière la plus sûre pour accélérer l'exécution d'un algorithme sur CPU est encore de réduire la séquence élémentaire d'instruction contenue dans les boucles de calcul. La manière la plus simple d'arriver à ce but est de réutiliser au maximum les résultats des opérations déjà effectuées. En pratique, cela correspond soit à un pré-traitement sur les données, soit à la réutilisation de résultats intermédiaires dans les boucles de calcul.

[Knaup 07] effectue ainsi plusieurs pré-traitements des données pour la reconstruction en tomographie CT. En rétroprojection 2D, il sur-échantillonne au préalable les données afin d'utiliser une simple interpolation au plus proche voisin à la place d'une interpolation linéaire. Il réduit ainsi par 2 la séquence de calcul. Le coût de ce pré-calcul (estimé à 5-10 % du temps de rétroprojection total) est amorti en comparaison de l'exécution en ligne de l'interpolation linéaire. Un compromis doit bien entendu être trouvé pour que le taux de sur-échantillonnage appliqué aux données de projection ne soit pas trop élevé afin qu'il ne fasse pas trop baisser la précision de reconstruction. En rétroprojection 3D avec une géométrie à faisceaux coniques, [Knaup 07] effectue une transformation géométrique des données : les plans de projection deviennent parallèles et font face au volume. Elle permet ainsi d'éliminer certains termes de la séquence de calcul.

Les algorithmes incrémentiels sont un bon exemple de réutilisation des résultats intermédiaires dans les boucles de calculs. Nous pouvons citer l'implémentation par [Egger 98] de l'algorithme incrémentiel de rétroprojection de [Cho 90]. En remarquant simplement que $\boldsymbol{\rho_{(x,y)} = x * \cos\phi + y * \sin\phi = \rho_{(x-1,y)} + \cos\phi}$, cet algorithme calcule la coordonnée $\boldsymbol{\rho_{(x,y)}}$ en incrémentant de $\boldsymbol{\cos\phi}$ la coordonnée $\boldsymbol{\rho_{(x-1,y)}}$ du pixel précédemment traité. Ainsi le nombre d'opérations et notamment de multiplications est considérablement réduit.

Un autre facteur à prendre en compte afin de réduire le temps de reconstruction sont les temps de communication avec les éléments externes aux CPUs. Il faut veiller notamment à ce que l'accès à la mémoire externe ne devienne pas un goulot d'étranglement, c'est-à-dire que l'exécution des calculs ne soit pas suspendu par l'attente en lecture ou en écriture de données. Pour remédier à ce problème, [Knaup 07] crée une localité spatiale et temporelle en inversant l'ordre des boucles et reconstruit des sous-volumes à partir des

sous-espaces de projection correspondants. Il réduit ainsi le nombre de défaut de cache, et donc le temps d'accès aux données. Au final en diminuant la séquence de calcul et en optimisant les accès mémoire, [Knaup 07] réduit d'un ou deux ordres de magnitude le temps de reconstruction des rétroprojections à faisceaux parallèles 2D (réduction de 5.2 s à 68 ms) et à faisceaux coniques 3D (réduction de 1.93 h à 3.2 mn).

1.2.2 Accélération algorithmique

Nous nous intéressons ici à la rétroprojection 2D d'une image $\boldsymbol{N^2}$ à partir de $\boldsymbol{P}$ projections. La complexité de la rétroprojection filtrée classique est en $\boldsymbol{O(P \log_2 N)}$ pour le filtrage rampe et en $\boldsymbol{O(P \cdot N^2)}$ pour la rétroprojection. $\boldsymbol{P}$ étant du même ordre de grandeur que $\boldsymbol{N}$, la rétroprojection filtrée est dit en $\boldsymbol{O(N^3)}$. Les techniques d'accélération algorithmique présentées dans cette partie réduisent la complexité en $\boldsymbol{O(N^2 \log N)}$.

La réduction de la complexité est obtenue soit par un sous-échantillonnage des données, soit par la définition d'opérateurs discrets utilisant des méthodes *divide and conquer*. Une condition à la première option est d'effectuer une décomposition (hiérarchique ou non) dans l'espace de Fourier [Rodet 02a], dans l'espace de projection [Bresler 07] ou bien encore dans l'espace image [Basu 00]. Il faut en effet se placer dans un espace de données justifiant un sous-échantillonnage sans conséquence pour la qualité de reconstruction. La deuxième option utilise la FFT lors de la reconstruction directe dans l'espace de Fourier, ou bien l'opérateur discret de rétroprojection défini par [Brady 98]. Les algorithmes d'accélération de la rétroprojection filtrée sont décrits de manière synthétique dans le tableau 2.1 et présentés plus en détail dans les paragraphes suivants.

Décomposition non hiérarchique dans l'espace de Fourier

L'algorithme MCFBP (*Multi Channel Filtered BackProjection*) de [Rodet 02a] décompose les données dans l'espace de Fourier en sous bandes carrées $\boldsymbol{B_{\nu_x} \cdot B_{\nu_y} = B_{\nu}^2}$. La somme des $\boldsymbol{B_{\nu}^2}$ imagettes issues de la rétroprojection des sous-sinogrammes, est suréchantillonnée pour former l'image finale (voir figure 2.1). Cette décomposition non hiérarchique ne permet pas à elle seule de réduire la complexité de la rétroprojection. C'est la réduction du nombre de projections $\boldsymbol{P}$ d'un facteur $\boldsymbol{B_{\nu}}$ pour chaque rétroprojection, qui permet d'obtenir une complexité en $\boldsymbol{O(PN^2/B_{\nu})}$ pour la rétroprojection. Si bien que c'est le filtrage avec une complexité en $\boldsymbol{O(B_{\nu} PN \log N)}$ qui prend alors le pas sur l'expression de la complexité.

[Rodet 02a] observe en pratique que le temps total diminue avec le nombre de canaux car même si le temps de filtrage augmente, le temps de la rétroprojection diminue fortement avec le nombre de canaux. Mais il observe également que la qualité de reconstruction se dégrade assez rapidement avec le nombre de canaux fréquentiels car la sélection dans le domaine fréquentiel des carrés associés à chaque canal n'est pas exacte et crée des effets de blocs. Pour contre balancer cette limitation d'accélération par le nombre de canaux fréquentiels, [Rodet 02a] fait remarquer que la rétroprojection sur chaque canal fréquentiel peut être accélérée par d'autres techniques d'accélération comme celles de [Bresler 07] ou de [Basu 00] et améliorer ainsi encore le facteur d'accélération.

Algorithmes	Principe d'accélération	Accélération
	Etapes des calculs	
MCFBP [Rodet 02b]	Décomposition non hiérarchique en composantes fréquentielles avec une résolution des sinogrammes selon la taille $\boldsymbol{B_\nu}$ de la sous-bande.	$\boldsymbol{O(B_\nu PN \log N)}$
	(1) Sélection et sous échantillonnage des sous sinogrammes correspondant à une sous bande $\boldsymbol{B_\nu^2}$ par filtrage dans l'espace de Fourier. (2) Rétroprojection des sous sinogrammes. (3) Recomposition de l'image complète à partir des sous composantes fréquentielles.	
SBFHBP [Bresler 07]	Décomposition hiérarchique en angle dans l'espace de projection avec une résolution multi-niveau des sinogrammes selon l'amplitude de la direction lente de projection.	$\boldsymbol{O(N^2 \log P)}$
	(0) Rétroprojections des P sous sinogrammes sous échantillonnés correspondant à un seul angle de projection. Puis opération de transvection (opération affine 2D) appliquée sur les P rétroprojections partielles $\boldsymbol{I_0}$. Puis récursivement (R1) Regroupement par 3 des images partiellement rétroprojetées $\boldsymbol{I_{n-1}}$. (R2) Sur échantillonnage de l'image partiellement rétroprojetée $\boldsymbol{I_n}$. (R3) Opération de transvection appliquée sur $\boldsymbol{I_n}$.	
FHBP [Basu 00]	Décomposition hiérarchique dans l'espace image avec une résolution multi-niveau des sinogrammes selon la taille de la sous image.	$\boldsymbol{O(PN \log N)}$
	Récursivement (R1) Sélection et sous échantillonnage du sinogramme correspondant à un quart d'image $\boldsymbol{I_{n-1}}$. (R2) Rétroprojection des sous sinogrammes. (R3) Recomposition de l'image $\boldsymbol{I_n}$ complète à partir des quarts d'images $\boldsymbol{I_{n-1}}$.	
ADRT inverse [Brady 98]	Décomposition hiérarchique en rétroprojection partielle (partage hiérarchique des sommes partielles).	$\boldsymbol{O(N^2 log(N))}$
	Récursivement (R1) Construction du linogramme de l'image $\boldsymbol{I_n}$ à partir des deux sous linogrammes des demi images $\boldsymbol{I_{n-1}}$.	
FRA [Matej 04]	Décomposition hiérarchique en TF inverse partielle (principe de la FFT).	$\boldsymbol{O(N^2 \log N)}$ + gridding
	(1) TF 1D des projections correspondant à un angle pour obtenir la TF de l'image en grille polaire. (2) Interpolation de la grille polaire en grille cartésienne. (3) Calcul de la TF inverse par FFT .	

TAB. 2.1 – Etat de l'art de l'accélération des algorithmes de rétroprojection

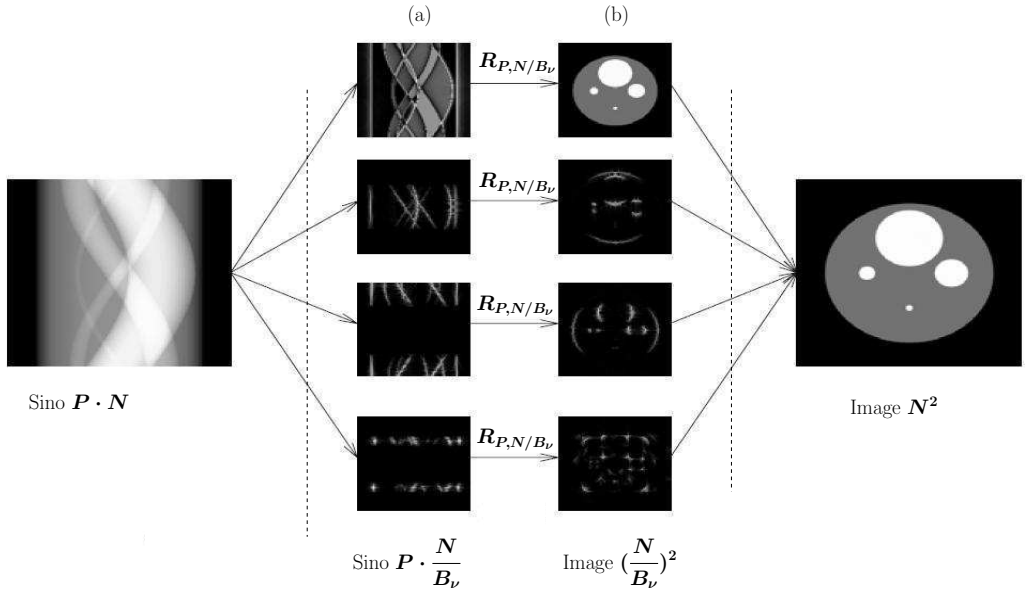


FIG. 2.1 – Algorithme MCFBP de rétroprojection filtrée multi canaux [Rodet02]. (a) décomposition fréquentielle et sous échantillonnage, (b) addition des imagettes et sur échantillonnage

Décomposition hiérarchique dans l'espace de projection

L'algorithme SBFHBP (*Shear Based Fast Hierarchical BackProjection*) de [Bresler 07] calcule d'abord $\boldsymbol{P}$ images correspondantes aux $\boldsymbol{P}$ rétroprojections partielles calculées par simple épandage des $\boldsymbol{P}$ *viewgrams*. Puis en $\boldsymbol{\log_3 P}$ étapes successives de réduction sommation, il aboutit au volume final (voir figure 2.2). Des transformations géométriques linéaires (transvection $\boldsymbol{T^i_{phi}}$ de l'étape i pour l'angle phi) sont effectuées sur chaque rétroprojection partielle. En effet, des opérations de rotations et de mises à l'échelle sont nécessaires avant chaque réduction sommation car lors de la toute première étape, les *viewgrams* sont rétroprojettés dans l'espace image directement en plaçant le *viewgram* et l'image dans le même repère.

Tel quel l'algorithme de rétroprojection possède $\boldsymbol{\log_3 P}$ étapes de calcul inégales en nombre d'opérations. Pour atteindre une complexité en $\boldsymbol{O(N^2)}$ à chaque niveau de cette décomposition hiérarchique de la rétroprojection, [Bresler 07] sous-échantillonne les rétroprojections partielles selon la direction lente de projection. Une complexité en $\boldsymbol{O(N^2 \cdot log(P))}$ est alors obtenue. La légitimité du sous-échantillonnage provient de la différence de résolution existant entre les directions dites lentes et rapides de projection (voir figure 2.3). Ainsi [Bresler 07] rétroprojette lors de la première étape les données de projection sur des images $\boldsymbol{N * N/3^{L-1}}$ avec L le nombre d'étapes de calcul. Pour obtenir une image finale de taille $\boldsymbol{N^2}$, il sur-échantillonne d'un facteur 3 selon la direction lente à chaque étape de calcul (voir figure 2.2).

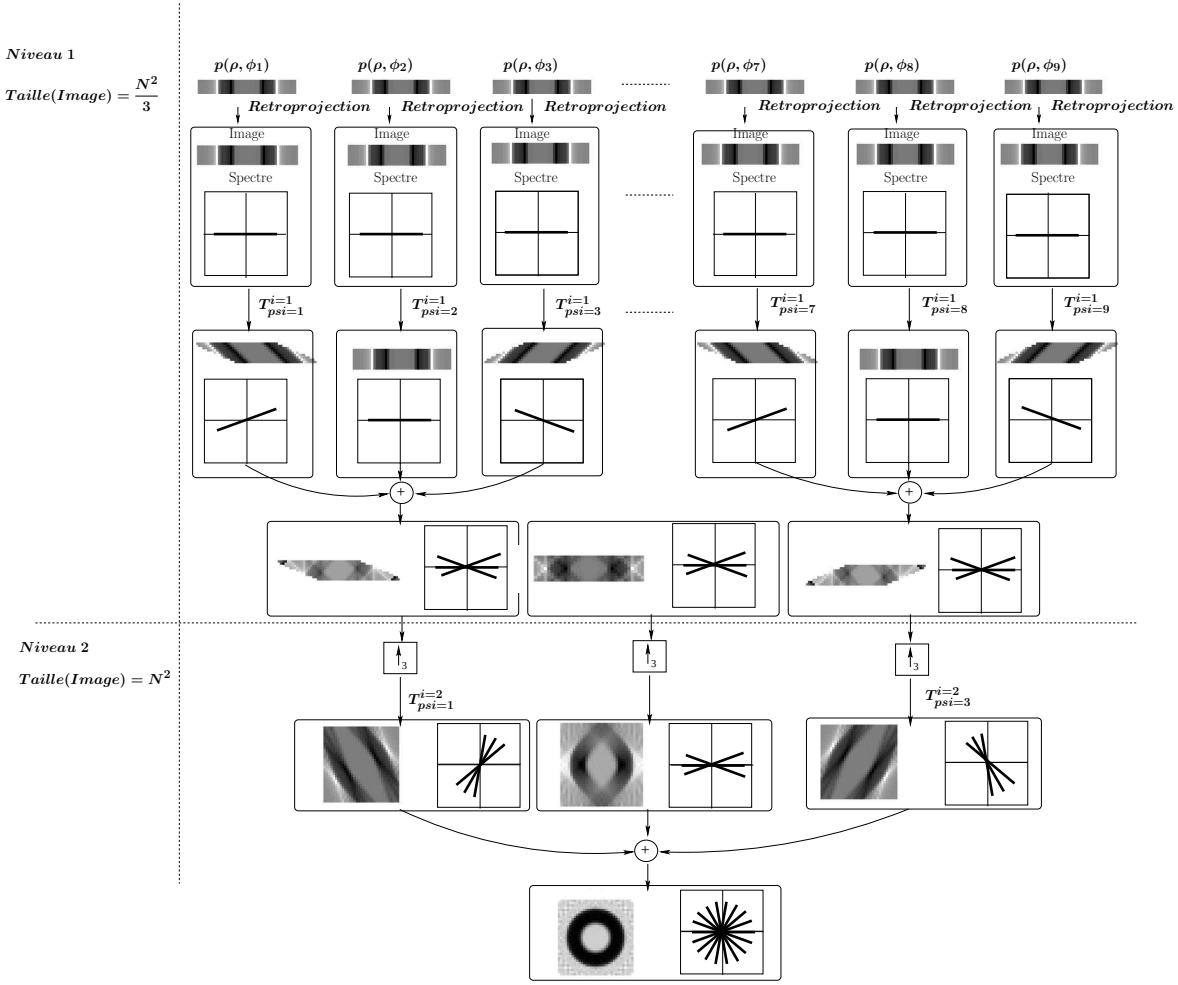


FIG. 2.2 – Décomposition hiérarchique dans l'espace de projection selon l'algorithme de [Bresler 07]. Les spectres des rétroprojections partielles sont représentés pour chaque étape de la rétroprojection à coté des images.

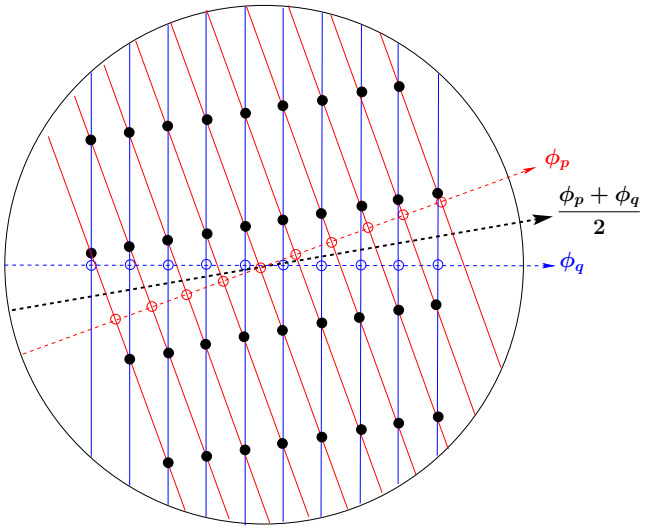


FIG. 2.3 – Deux rétroprojections correspondantes aux angles $\boldsymbol{\phi_p}$ et $\boldsymbol{\phi_q}$ sont représentées respectivement en rouge et en bleu. Les valeurs des projections sont en ronds vides et les valeurs de l'image issues de la rétroprojection des deux angles en ronds pleins. La densité d'échantillons selon la direction rapide de projection (axe d'angle $\boldsymbol{(\phi_p + \phi_q)/2}$) est plus importante que celle selon la direction lente de projection (orthogonale à l'axe d'angle $\boldsymbol{(\phi_p + \phi_q)/2}$).

Décomposition hiérarchique dans l'espace image

L'algorithme FHBP (*Fast Hierarchical BackProjection*) de [Basu 00] sépare les données de projection en quatre sinogrammes correspondants chacun à un quart d'image (voir figure 2.4). Il effectue pour cela des opérations de décalage ($\boldsymbol{\hat{Z}_i}$) et de troncature d'un facteur 2 ($\boldsymbol{\hat{K}_{N/2}}$). Ensuite après la rétroprojection de ces quatre sinogrammes et le décalage dans l'espace image ($\boldsymbol{Z_i}$) des quatre imagettes, l'image finale est reconstruite. Appliqué récursivement, ce principe permet de décomposer la rétroprojection en $\boldsymbol{\log_2 N}$ étapes. Cette décomposition hiérarchique de la rétroprojection dans l'espace image offre la possibilité d'effectuer un sous-échantillonnage d'un facteur 2 des données de projection (opération $\boldsymbol{A_P}$) avant la rétroprojection sans aucune perte de précision. Un sur-échantillonnage est effectué après chaque rétroprojection afin d'obtenir une image de taille $\boldsymbol{N^2}$. L'opération de rétroprojection est ainsi réduite d'un facteur 2 à chaque étape. Une complexité en $\boldsymbol{O(PN \log N)}$ de la rétroprojection est ainsi atteinte.

Le sous-échantillonnage est théoriquement expliqué par la forme en papillon du support de la transformée de Fourier de la transformée de Radon (voir figure 2.5). La TF a une bande limite $\boldsymbol{B_\rho}$ selon $\boldsymbol{\nu_\rho}$, et une bande limite égale à $\boldsymbol{B_f \cdot B_\rho}$ selon $\boldsymbol{\nu_\phi}$ avec $\boldsymbol{B_f}$ la bande limite de la transformée de Fourier de l'image. Lorsque l'on réduit la taille de l'image d'un facteur 2, le papillon rétrécit d'un facteur 2 selon l'axe $\boldsymbol{\nu_\phi}$. La transformée de Radon est alors sur-échantillonnée d'un facteur 2 supplémentaire. C'est ce principe qui est utilisé par [Basu 00] pour effectuer théoriquement sans perte le sous-échantillonnage d'un facteur 2 des données de projection à chaque étape. Toutefois, en pratique l'utilisation d'interpolateurs ne respecte pas toutes les conditions théoriques, et crée ainsi une

dégradation de la qualité de reconstruction.

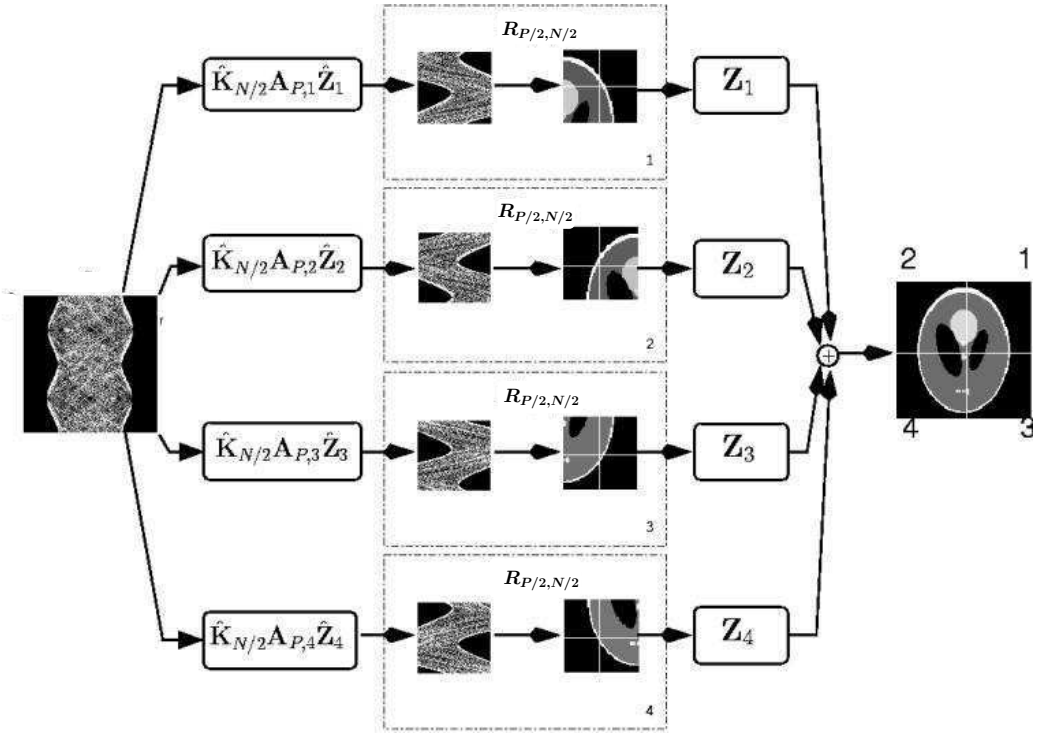


FIG. 2.4 – Décomposition hiérarchique de la rétroprojection dans l'espace image selon l'algorithme de [Basu 00].

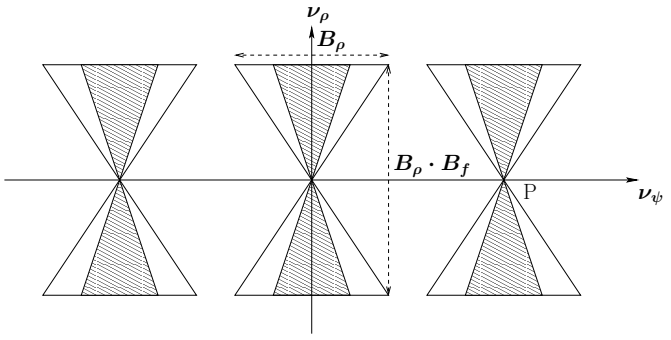


FIG. 2.5 – Support en papillon de la TF de la transformée de Radon

Utilisation d'un opérateur de rétroprojection discret

L'algorithme ADRT (*Approximate Discrete Radon Transform*) de [Brady 98] est une approximation discrète de la transformée de Radon continue. Il discrétise à la fois l'espace image avec une grille 2D de pixels mais également l'espace de projection en associant chaque donnée de projection à une ligne constituée d'une séquence de pixels de l'image

(voir figure 2.6). Les données de projection ne sont donc plus organisées en *viewgrams* ou en sinogrammes mais en linogrammes. Les lignes discrètes $\boldsymbol{l(h, s)}$ sont définies par leur coordonnée à l'origine $\boldsymbol{h}$ et leur pente $\boldsymbol{s}$.

A partir de cette modélisation des données, [Press 06] définit un rétroprojecteur qui est l'inverse de l'ADRT. La rétroprojection est décomposée alors de manière hiérarchique en $\mathbf{log_2}\, \boldsymbol{N}$ étapes comme illustrée sur la figure 2.6. Lors de la première étape, l'algorithme part du linogramme complet pour former deux demi-linogrammes qui correspondent aux linogrammes des deux demi-images gauche et droite de l'image entière. Chaque valeur associée à une ligne des demi-linogrammes est alors la somme des valeurs de deux lignes du linogramme complet. En effet, sur la moitié de l'image deux lignes parcourent le même chemin comme représentées sur la figure 2.6 où les lignes rouges et vertes ont la même séquence de pixels sur la demi-image gauche. Ainsi les valeurs des lignes rouges et vertes sont sommées et associées à une ligne du linogramme de gauche. Les valeurs des lignes noires et bleues sont également regroupées et associées à une seule ligne. Appliquée récursivement, cette division des linogrammes en deux aboutit à des fractions de linogrammes qui correspondent à des colonnes de pixels. Et ainsi, la valeur attachée à un pixel est bien la somme de toutes les lignes passant par le pixel, c'est-à-dire la somme des valeurs des lignes rouges, bleues, vertes et noires de la figure 2.6. La complexité de la rétroprojection est en $\boldsymbol{O(N^2 \log N)}$, avec $\boldsymbol{N^2}$ opérations par étape.

1.3 Accélération des algorithmes itératifs

Deux techniques sont classiquement utilisées pour accélérer les algorithmes itératifs. Premièrement, la matrice système est générée en ligne afin d'éviter son stockage sur un disque dur au temps d'accès trop important. Deuxièmement, la convergence assez lente des algorithmes EM est accélérée grâce à l'utilisation des algorithmes OSEM (*Ordered Subset EM*).

1.3.1 Génération en ligne de la matrice système

La taille trop importante de la matrice système $\boldsymbol{H = (h_{ij})}$ (jusqu'à quelques Tera octets) interdit son stockage intégral en mémoire RAM. Deux stratégies existent pour contourner ce problème : soit une matrice système compressée est accédée en mémoire externe puis décompressée "en ligne" c'est-à-dire pendant l'exécution ([Ortuno 06], [Scheins 06], [Herraiz 06]), soit les coefficients $\boldsymbol{h_{ij}}$ sont calculés "en ligne".

Si l'utilisation d'une matrice système compressée requiert l'obtention préalable d'une matrice système exacte par mesures réelles ([Panin 06]), par simulation monte-carlo (180 jours de calcul pour [Herraiz 06]) ou par pré-calcul analytique ([Scheins 06]), ce n'est pas le cas du calcul en ligne. Ainsi, la manière la plus répandue d'obtention de la matrice système est de calculer "en ligne" les coefficients par lancer de rayon (*ray-tracing*) pour l'étape de projection des algorithmes itératifs (la contribution $\boldsymbol{h_{ij}}$ du voxel j à la LOR i est modélisée par la longueur du segment d'interception de la LOR avec le voxel) et par interpolation bilinéaire pour l'étape de rétroprojection.

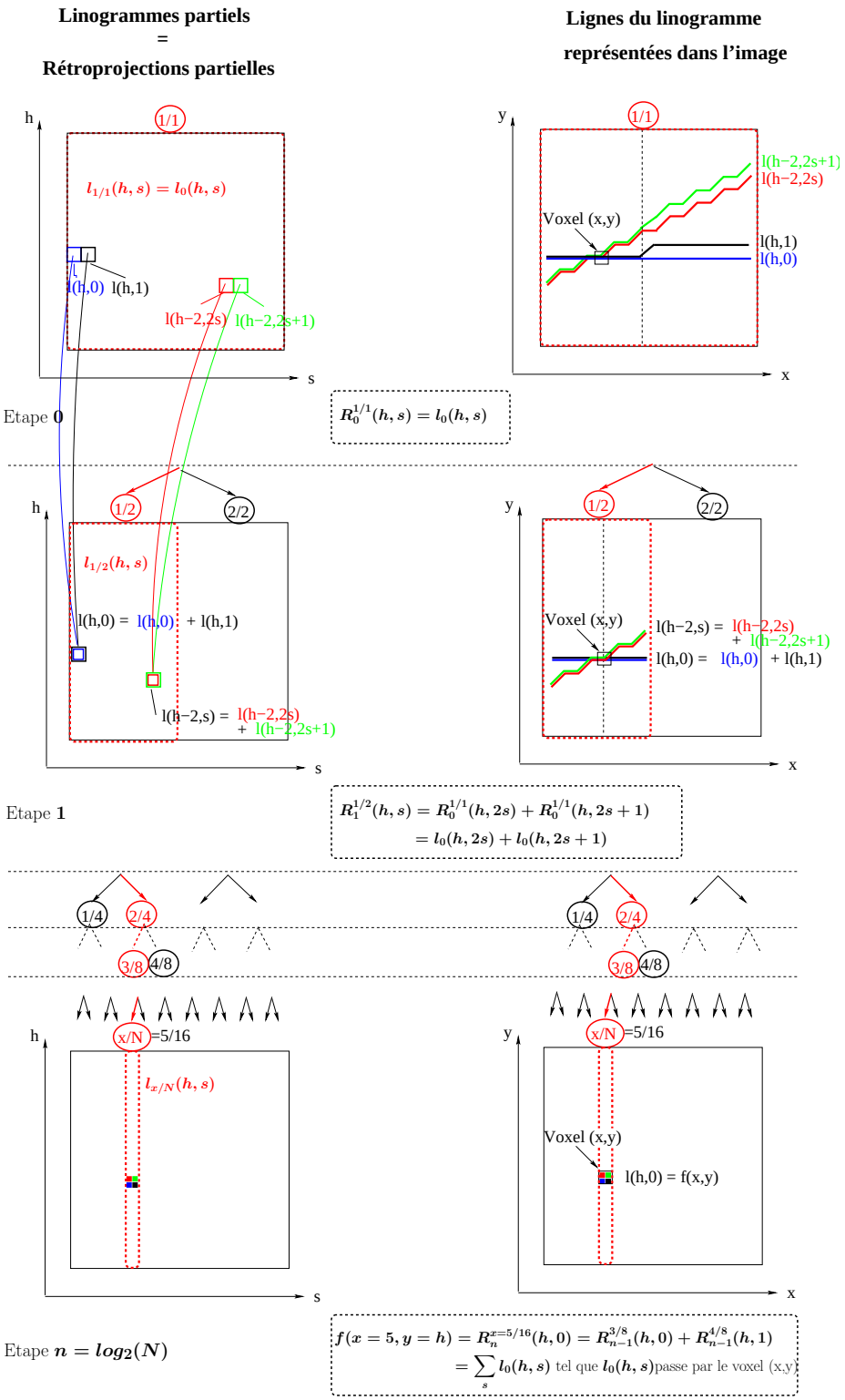


FIG. 2.6 – Décomposition hiérarchique dans un espace de projection composé de lignes discrètes. La rétroprojection partielle de l'étape i $\boldsymbol{R_i^{j/2^i}(h,s)}$ correspond au linogramme partiel $\boldsymbol{l_{j/2^i}(h,s)}$. Ce dernier est la j ième fraction du linogramme original $\boldsymbol{l_0}$ découpé en $\boldsymbol{2^i}$ sous linogrammes. Le calcul récursif des linogrammes partiels aboutit à l'obtention de l'image rétroprojetée organisée en N colonnes de pixels.

D'autres techniques plus originales de calcul en ligne de la matrice système existent comme celles de [Kadrmas 05] et de [Kudrolli 02]. [Kadrmas 05] effectue les projections et rétroprojections par des rotations du volume qu'il décompose en multiplications matricielles. Cette technique a l'avantage d'inclure, sans surcoût de calcul, les corrections (atténuations, événements aléatoires et diffusés). [Kudrolli 02] effectue en ligne une simulation monte-carlo simplifiée qu'il qualifie d'échantillonnage stochastique. Les temps de reconstruction ainsi obtenus sur PC sont présentés en annexe D.

1.3.2 Accélération de la convergence des algorithmes EM

Afin de réduire la nombre d'opérations lors de chaque mise à jour du volume, l'algorithme OS-EM de [Hudson 94] utilise seulement une partie des données de projection. [Hudson 94] ordonne l'espace de projection en $\boldsymbol{L}$ sous-ensembles (*subsets*) qui sont utilisés chacun leur tour lors de $\boldsymbol{L}$ sous-itérations de l'itération courante $\boldsymbol{n}$. Lors du calcul du coefficient $\boldsymbol{C^{(n,l)}}$ de mise à jour du volume, les données de projection du *subset* $\boldsymbol{S_l}$ sont estimées puis comparées aux mesures $\boldsymbol{y_i}$ avant d'être rétroprojetées dans le volume entier :

$$\boldsymbol{\lambda_j^{(n,l)} = \frac{\lambda_j^{(n,l-1)}}{N_j^l} C_j^{(n,l)}} \quad \textbf{avec} \quad \left\{ \begin{array}{l} \boldsymbol{C_j^{(n,l)} = \sum_{i \in S_l} h_{ij} \left(\frac{y_i}{\sum_{j'=1}^{J} h_{ij'} \lambda_{j'}^{(n,l-1)}} \right)} \\ \boldsymbol{N_j^l = \sum_{i \in S_l} h_{ij}} \end{array} \right. \tag{2.1}$$

Même si l'algorithme OSEM n'est pas vraiment un algorithme EM et que sa convergence n'est pas théoriquement prouvée, il se révèle en pratique l'un des meilleurs algorithmes itératifs pour la tomographie TEP. Il permet en effet d'accélérer (pour un nombre modéré de *subsets*), d'un facteur 10 le temps de calcul de l'algorithme EM et cela pour une qualité de reconstruction quasiment identique. Si l'augmentation du nombre de *subsets* permet d'accélérer la reconstruction, elle cause également une détérioration de la qualité de reconstruction. Ainsi au-delà d'une dizaine de *subsets*, le volume devient de plus en plus bruité. Une autre précaution à prendre est de choisir les éléments de chacun des *subsets* avec la plus grande distance angulaire possible. Par exemple en OSEM2D pour $\boldsymbol{N_\phi}$ projections et L *subsets*, chaque *subset* devra contenir $\boldsymbol{N_{subset} = N_\phi / L}$ *bins* de LORs espacés de $\boldsymbol{\pi / N_{subset}}$.

1.4 Compromis entre accélération, qualité de reconstruction et complexité de mise en oeuvre

Les techniques d'accélération présentées jusqu'ici demandent en pratique d'être employées avec soin car à partir d'un certain degré d'accélération, elles ne permettent plus une qualité de reconstruction équivalente à celle de l'algorithme original. De plus, leurs mises en oeuvre peut s'avérer difficile pour être réellement utilisées en pratique.

1.4.1 Efficacité en pratique des techniques d'accélérations algorithmiques

Les techniques d'accélérations algorithmiques présentées pour la rétroprojection 2D offrent une accélération théorique de $\boldsymbol{N/\log_2 N}$, ce qui correspond pour une image $\mathbf{256^2}$ à un facteur 32. Ce facteur théorique n'est toutefois pas atteint en pratique car comme le fait remarquer [Basu 00], il faut utiliser certains réglages techniques qui correspondent essentiellement :

- au nombre de points utilisés pour l'interpolation de la grille polaire en une grille cartésienne pour l'algorithme FRA (*Fourier Reconstruction Algorithms*) ;
- au nombre d'échantillons selon la direction lente de projection pour l'algorithme SBFHBP de [Bresler 07] ;
- au nombre d'étapes de calcul pour lesquelles un sous-échantillonnage est effectué pour l'algorithme FHBP de [Basu 00] ;
- au nombre de canaux fréquentiels pour l'algorithme MCFBP de [Rodet 02a].

Les facteurs d'accélération obtenus pour différentes paramétrisations des algorithmes FRA, MI et FHBP sont présentés dans le tableau 1.4.1. Si les algorithmes MI et FHBP permettent d'obtenir une accélération d'un ordre de magnitude pour une qualité de reconstruction proche de l'algorithme original, les algorithmes FRA et MCFBP dégradent un peu plus la qualité de reconstruction et offrent une accélération moindre. Dans un autre contexte en itératif 2D, [Hamill 03] utilise l'ADRT de [Brady 98] pour le projecteur et le rétroprojecteur de l'algorithme EM. Il obtient une accélération d'un ordre de magnitude par rapport à la méthode standard EM et une qualité de reconstruction semblable à celle obtenue avec la méthode OSEM.

Référence	Algorithme	Image	Accélération	Erreur $\boldsymbol{L^1}$	Erreur $\boldsymbol{L^2}$ normalisée	Qualité
[Basu 00]	FRA	$\mathbf{256^2}$	$\mathbf{\simeq 10}$		**2.5**	++
[Basu 00]	FHBP	$\mathbf{256^2}$	$\mathbf{\simeq 10}$		**1.16**	++++
		$\mathbf{256^2}$	$\mathbf{\simeq 20}$		**1.4**	+++
[Bresler 07]	MI	$\mathbf{512^2}$	$\mathbf{\simeq 10}$		**1.05**	++++
		$\mathbf{512^2}$	$\mathbf{\simeq 20}$		**1.1**	++++
[Rodet 02a]	MCFBP	$\mathbf{512^2}$	**2.3**	**0.74%**		++
		$\mathbf{512^2}$	**3.4**	**2%**		+

TAB. 2.2 – Facteurs d'accélération de la rétroprojection filtrée pour différents algorithmes. L'erreur quadratique ($\boldsymbol{L^2}$) par rapport au volume d'origine est normalisée avec l'erreur obtenue pour la rétroprojection filtrée. L'erreur absolue ($\boldsymbol{L^1}$) est calculée par rapport à l'image reconstruite par rétroprojection filtrée.

1.4.2 Mise en oeuvre des techniques d'accélération algorithmique

Les algorithmes accélérés modifient en profondeur l'organisation des données et ainsi demandent bien souvent une manipulation supplémentaire des données ce qui se traduit

par une implémentation plus complexe. De plus, certaines techniques d'optimisations logicielles efficacement utilisées lors de l'implémentation de l'algorithme original peuvent s'avérer inadaptées lors de l'implémentation de l'algorithme accéléré. Ainsi pour la rétroprojection, les techniques d'accélération algorithmique sont peu utilisées en pratique. L'algorithme original est accéléré simplement par les optimisations logicielles présentées en début de cette étude (algorithme incrémentiel, déroulage de boucles, introduction de localités spatiales et temporelles...). Par contre, l'algorithme OSEM dont son implémentation reste proche de celle de l'algorithme EM, est devenu un algorithme incontournable pour la reconstruction TEP. Il permet une accélération d'un ordre de magnitude des algorithmes itératifs.

1.5 Une accélération insuffisante sur les machines standards

Les besoins en puissance de calcul des équipes médicales radiologiques sont intimement liés aux capacités d'acquisition des scanners. Afin de savoir si les PCs seront à l'avenir une solution technologique satisfaisante, nous avons comparé en échelle logarithmique, les besoins en calcul avec la puissance offerte par les CPUs pour la tomographie TEP (voir figure 2.7) et CT (voir figure 2.8). A partir d'une simple mesure du temps de reconstruction obtenue à l'aide d'une implémentation PC optimisée, nous extrapolons les performances de calcul que peuvent offrir les PCs pour la reconstruction TEP et CT pour les prochaines années. Nous nous basons pour cela sur la loi de Moore (en théorie doublement des performances des CPUs tous les 18 mois et en pratique tous les 2 ans).

1.5.1 Tomographie TEP

En tomographie TEP, le nombre de calculs nécessaire à la reconstruction du volume est lié aux nombres de LORs du scanner. Par exemple en rétroprojection 3D, $\boldsymbol{O(N^4)}$ calculs sont nécessaires pour reconstruire un volume à partir de $\boldsymbol{O(N^3)}$ LORs. De manière plus générale, le logarithme du nombre de calculs est proportionnel au logarithme du nombre de LORs du scanners. Ainsi, nous pouvons directement comparer en échelle logarithmique l'évolution des performances en calcul des CPUs avec l'évolution du nombre LORs acquises par les scanners TEP (voir figure 2.7). La mesure à partir de laquelle nous avons extrapolé les performance des PCs, correspond au temps de reconstruction obtenu à l'aide des exécutables de la librairie *open source* STIR ([Thielemans 06]) sur un Pentium 4 (1 mn en 3D-RP et 1 mn 30s en OSEM).

Nous pouvons tout d'abord remarquer que les scanners TEP cliniques ont vu, depuis leur création dans les années 70 jusqu'à l'arrivée du scanner haute résolution HRRT dédié à l'étude du cerveau, leur capacité d'acquisition doubler tous les 18 mois ([Brasse 04]). Mais depuis la fin des années 90, ce rythme soutenu de croissance s'est essoufflé : actuellement les scanners cliniques pour corps entier n'ont pas plus de 25 000 cristaux. Parallèlement, les scanners pré-cliniques ou $\boldsymbol{\mu}$TEP grâce à un rythme soutenu de croissance jusqu'à aujourd'hui (doublement tous les 18 mois), ont rattrapé le retard technologique qu'il avait initialement avec les scanners cliniques. Si nous comparons à présent, l'évolution des scanners TEP avec celle des CPUs, nous observons un écart assez grand

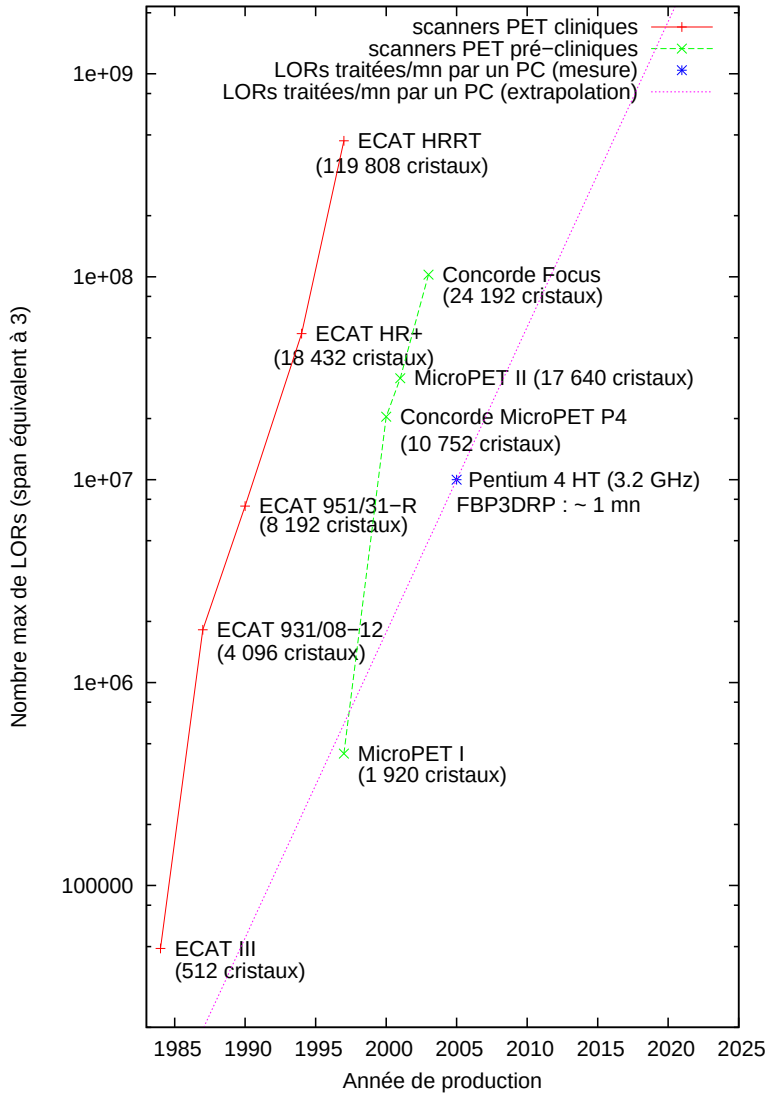


FIG. 2.7 – Evolution comparée des scanners TEP avec celle des CPUs

entre besoins et offres en performance de calcul. On peut estimer à 15 ans le retard de la technologie de calcul des CPUs par rapport à la technologie d'acquisition des scanners TEP. En effet, il aura fallu attendre 2005 pour voir un CPU comme le Pentium 4 traiter dans un temps de l'ordre de la minute, des données correspondant à un scanner datant de 1990. Et pour obtenir un temps de reconstruction de l'ordre de la minute pour les scanners cliniques corps entier actuels et pour le scanner HHRT, il faudra attendre respectivement les années 2012 et 2017. Rappelons également qu'actuellement, il faut près de 16 h avec un PC pour reconstruire des volumes acquis avec le HRRT avec les algorithmes itératifs ([Jones 06]).

1.5.2 Tomographie CT

En tomographie CT, le graphe de la figure 2.8 compare le nombre de coupes transverses acquises par le scanner avec le nombre de coupes transverses par seconde reconstruites par les PCs. L'évolution des performances des PCs a été extrapolée à partir du temps obtenu par la société Exxim ([Exxim 07]) pour l'algorithme de rétroprojection FDK. Nous pouvons observer que la croissance de la capacité d'acquisition des scanners CT est légèrement plus importante que celle des performances en calcul des PCs. Ainsi le retard des PCs existant dès les années 90 ne se résorbe pas. Nous pouvons évaluer ce retard à une dizaine d'années. En effet, un CPU datant de 2007 permet de reconstruire en temps réel les données d'un scanner datant de 1997.

1.5.3 Conclusion

Bien que la puissance de calcul des processeurs classiques augmente constamment et malgré les importants efforts pour optimiser les algorithmes, les solutions logicielles standards sont actuellement insuffisantes pour satisfaire les contraintes de reconstruction en temps réel. Ainsi, l'accélération matérielle est appelé à réduire le fossé technologique existant entre les systèmes d'acquisition et de reconstruction.

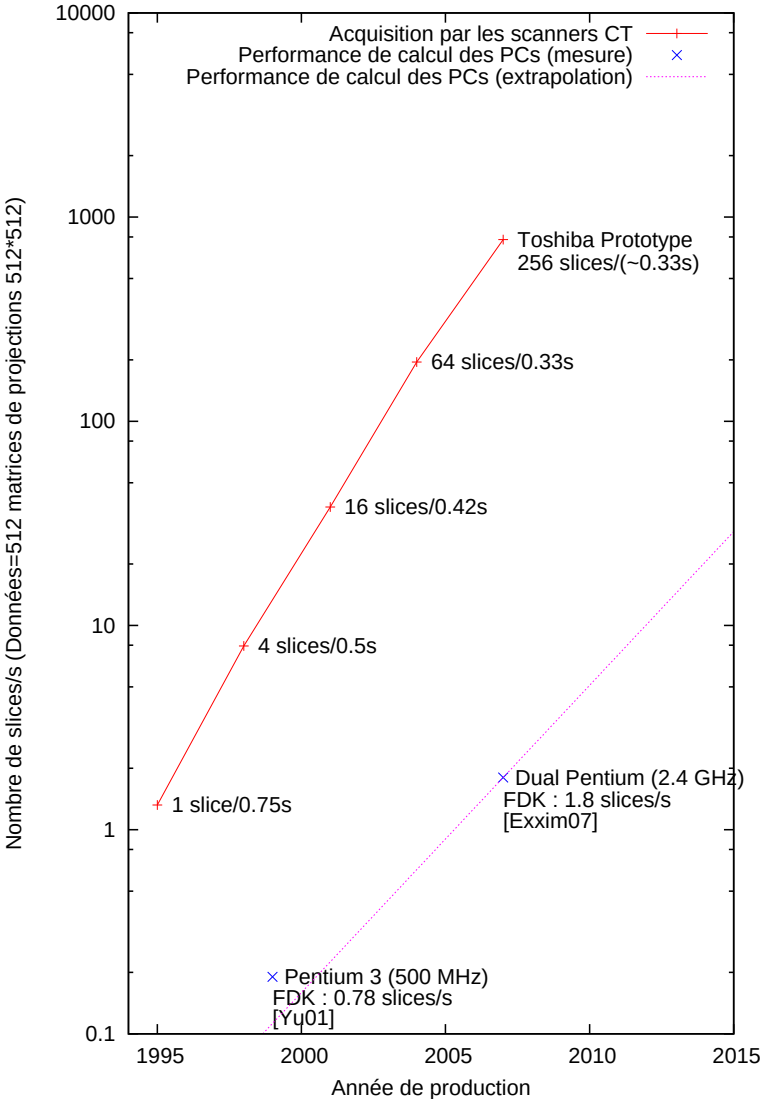


FIG. 2.8 – Evolution comparée des scanners CT avec celle des CPUs (données sur les scanners CT extraites en partie de [Kalender 05])

2 Parallélisation de la reconstruction

L'enjeu de toute stratégie de parallélisation est d'utiliser le plus efficacement possible la puissance de calcul offerte par les différents de noeuds de calculs d'une machine parallèle. Nous présenterons ainsi tout d'abord une métrique permettant d'évaluer cette efficacité de parallélisation. Puis nous décrirons les stratégies générales de parallélisation adaptées à la reconstruction tomographique. Enfin, nous présenterons plus en détail les stratégies destinées aux algorithmes de rétroprojection filtrée 3D et aux algorithmes itératifs 3D.

2.1 Efficacité de parallélisation

La loi d'Amdahl exprime le gain de performance que l'on peut espérer en augmentant le nombre de noeuds de calcul. En théorie, le gain est linéairement proportionnel au nombre d'unités de traitement pour un code 100% parallélisable et une répartition de charge (*load balancing*) idéale entre unités de calcul. Pour les algorithmes étudiés ici, le gain en performance ne subit qu'un léger fléchissement de cette loi linéaire, n'ayant qu'une faible fraction de code non-parallélisable. Cependant, la communication des données peut devenir un facteur limitant de l'effort de parallélisation.

Si une synchronisation de données est nécessaire, le coût en communication augmente alors assez rapidement avec le nombre d'unités de calcul. Le gain atteint alors un seuil de performance lorsque la réduction linéaire du coût de calcul (si le problème est de grande taille) est trop lourdement pénalisée par le coût des communications globales. A partir d'une modélisation analytique des coûts en communication et en calcul, on peut obtenir une bonne prédiction de l'évolution du rapport entre ces deux coûts et décider ainsi alors du nombre de noeuds de calcul adapté au problème. Dans la suite de ce document, l'efficacité de la parallélisation sera simplement mesurée par la ratio $\eta_{/\!/}$:

$$\eta_{/\!/} = \frac{\text{facteur d'accélération}}{\text{nombre d'unités de traitement}} \tag{2.2}$$

2.2 Stratégies de parallélisation

On qualifie en général le parallélisme soit de contrôle où plusieurs tâches sont effectuées en même temps sur plusieurs unités de calcul, soit de donnée où une même tâche est répétée sur un ensemble de données réparties sur les différents noeuds de calculs. En théorie, le parallélisme de données n'est qu'un cas particulier du parallélisme de contrôle, mais il possède des propriétés spécifiques permettant une adéquation efficace avec les architectures des machines parallèles. En reconstruction tomographique, un parallélisme de données s'applique tout naturellement. Mais selon le type d'algorithme et le grain de parallélisation, cette parallélisation demande ou non des étapes de synchronisation de données entre les unités de calculs.

A un grain élevé de parallélisation, la parallélisation des algorithmes 2D et 3D ne nécessitent pas de coûteuses étapes de synchronisation des données. C'est ce que font [Kontaxakis 02] et [Jones 06], lorsqu'ils conçoivent leur système de reconstruction destiné

à un service clinique en TEP dynamique. Dans ce cas, la tâche globale correspond à toutes les acquisitions de la journée des différents patients composées chacune de 20-30 *frames*, et les sous-tâches à la reconstruction d'une *frame*. Et ainsi l'exécution des sous tâches sur les différents noeuds de calcul se fait sans synchronisation de données.

La parallélisation des algorithmes de reconstruction à un grain plus fin de parallélisation, peut se faire sans synchronisation de donnée dans le cas 2D mais doit se faire avec en 3D. En effet, les algorithmes de reconstruction (rétroprojection filtrée ou algorithmes itératifs) effectuent d'abord un traitement sur les données de projection puis une rétroprojection dans l'espace image. Il existe ainsi une dépendance entre espace de projection et espace image. Or si en 2D, une coupe 2D ne dépend que d'un sinogramme 2D et vice versa, ce n'est pas la cas en 3D où une partie de l'espace image dépend de l'ensemble de l'espace de projection et vice versa. Ainsi les algorithmes 2D (rétroprojection 2D et algorithmes itératifs 2D) peuvent se décomposer en sous-tâches indépendantes correspondant chacune à la reconstruction d'une coupe 2D du volume à partir d'un sinogramme 2D. Pour les algorithmes 3D, le découpage en sous tâches des algorithmes 3D se fait soit à l'aide d'une Décomposition de l'Espace Image (DEI) soit à l'aide d'une Décomposition de l'Espace de Projection (DEP) [Chen 90]. Avec une décomposition DEI, la charge de calcul affectée à un noeud de calcul correspond au traitement d'un sous-volume. Si chaque noeud a alors besoin d'accéder à une partie uniquement de l'espace image (son sous-volume), tous ont besoin d'accéder à l'ensemble de l'espace de projection. Avec une décomposition DEP, la charge de calcul affectée à un noeud de calcul correspond au traitement d'un sous-espace de projection. Si chaque noeud a alors besoin d'accéder uniquement à une partie de l'espace de projection (son sinogramme), tous ont besoin d'accéder à l'ensemble de l'espace image.

2.3 Parallélisation de la rétroprojection filtrée 3D

Les algorithmes de rétroprojections filtrés se prêtent bien à une parallélisation avec une décomposition dans l'espace image (DEI) où chaque noeud de calcul traite un sous-volume. En effet, de cette manière le traitement séquentiel des *viewgrams* est respecté, avec l'exécution des trois étapes de calculs illustrées sur la figure 2.9 : (P) Pré-traitements du *viewgram* (pondération, filtrage...), (D) Distribution des *viewgrams* pré-traités vers les $\boldsymbol{n_{proc}}$ noeuds de calcul, et (R) Rétroprojection de ce *viewgram* sur les $\boldsymbol{n_{proc}}$ sous-volumes reconstruits en parallèle.

L'accélération de l'étape de rétroprojection (R) est bien plus essentielle que celle de l'étape de filtrage. En effet, les efforts de parallélisation de l'étape de filtrage par [Gregor 02] ou de masquage de (D) la communication par le réseau de la projection $\boldsymbol{i}$ avec (P) le traitement de la projection $\boldsymbol{i+1}$ par le noeud de calcul par [Laurent 98], n'améliore que faiblement le temps de reconstruction. Et lorsqu'une parallélisation de l'étape de filtrage (P) est mise en place, [Ni 06] conclut qu'il est préférable d'attendre que tous les noeuds de calcul aient terminé l'étape de filtrage avant de débuter l'étape de rétroprojection (R). En effet, la charge de calcul de l'étape de filtrage (P) est bien distribuée dans un réseau homogène. De plus, la synchronisation est ainsi simple à mettre en oeuvre et l'on peut concentrer l'effort de parallélisation sur l'étape de rétroprojection

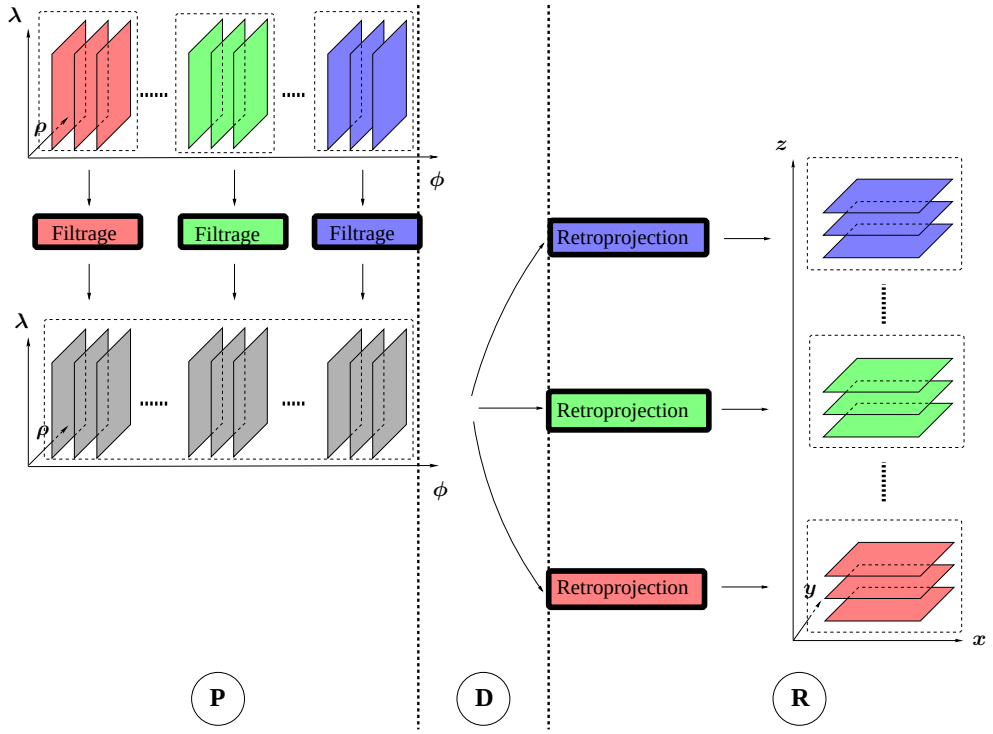


FIG. 2.9 – Parallélisation de la rétroprojection filtrée 3D

comme le fait [Brasse 05] dont la charge de calcul est bien plus importante que celle de l'étape filtrage. Toutefois dans un réseau hétérogène, il devient peut-être plus intéressant de distribuer de manière plus élaborée la charge de calcul de l'étape de filtrage comme le fait [He 06].

2.4 Parallélisation des algorithmes itératifs

Dans cette section, nous présenterons et comparons tout d'abord les deux stratégies classiques de parallélisation : celle avec Décomposition dans l'Espace de Projection (DEP) et celle avec Décomposition dans l'Espace Image (DEI). Nous exposerons enfin quelques propositions originales de parallélisation qui tentent de supprimer les étapes de synchronisations de données.

2.4.1 Parallélisations DEP et DEI

Nous illustrons les parallélisations DEP et DEI respectivement sur les figures 2.10 et 2.11. La parallélisation DEP consiste tout d'abord à Distribuer le volume sur tous les noeuds de calcul (étape D), à Projeter sur chaque noeud de calcul ce volume sur une partie des *viewgrams* constituant l'espace de projection (étape P) puis à Rétroprojeter sur chaque noeud ces *viewgrams* dans l'espace image (étape R) et enfin à effectuer la Somme

des rétroprojections partielles (étape S). A l'opposé la parallélisation (DEI) consiste à Projeter sur chaque noeud de calcul un sous volume (étape P), à Sommer tous ces projections partielles projections (étape S), puis à Distribuer la projection complète ainsi obtenue sur chaque noeud de calcul (étape D) et enfin à effectuer la Rétroprojection du sous-volume associé à chaque noeud de calcul (étape R).

Pour les deux parallélisations DEP et DEI, à une étape de Somme des résultats partiels (étape S) succède une étape de Distribution de cette somme sur les différents noeuds de calcul de la machine parallèle (étape D). Mais si pour la parallélisation DEP, les données communiquées correspondantes à des sous-volumes, pour la parallélisation DEI, les données échangées correspondent à des *viewgrams*. Or puisque la taille des données de projection sont plus importantes que la taille du volume (le nombre de données doit être plus grand que le nombre d'inconnues), la parallélisation DEP minimise en théorie les communications et donc le temps de reconstruction.

2.4.2 Parallélisations minimisant la synchronisation de données

Les parallélisations DEP et DEI ont le désavantage d'effectuer une décomposition soit de l'espace de Projection soit de l'espace Image mais pas des deux en même temps. [Fritzsche 05], [Keesing 06] et [Deng 06a] ont mis en place des stratégies de parallélisations originales effectuant une décomposition double à la fois dans l'espace de projection et dans l'espace image. La reconstruction peut alors être découpée en traitement de paires (sous-volume, sous-espace de projection) indépendantes les unes des autres. Il n'existe alors plus de coûteuses étapes de synchronisations de données.

Parallélisation pour une acquisition faiblement 3D (quasi 2D)

[Fritzsche 05] effectue un partionnement partiel de l'espace image et de l'espace de projection en tomographie électronique pour une acquisition 2D/3D. Cette technique de tomographie se prête bien à un partionnement double car même si les coupes 2D partagent des données de projection en commun, la dépendance de données reste limitée. Ainsi pour la projection, une rangée de détecteurs est déterminée par 2 ou 3 coupes 2D et pour la rétroprojection, une coupe transverse est déterminée par deux ou trois rangées de détecteurs. Il attribue alors à chaque noeud de calcul une partition du plan de projection (des rangées de détecteurs) et les coupes 2D qui s'y projettent. Les coupes transverses à cheval entre deux partitions du plan de projection sont dupliqués : dans un noeud, la coupe transverse est reconstruite et dans l'autre elle est seulement utilisée lors de l'étape de projection. Cette redondance des données communes entre deux partitions du plan de projection permet de réduire les communications de données : (1) communication des rangées de détecteurs adjacentes après la projection, et (2) communication des coupes transverses adjacentes après rétroprojection. Pour d'autres tomographies avec une acquisition réellement 3D, une donnée de projection touche un plus grand nombre de coupes transverses qu'en reconstruction 2D. Ainsi, la technique de redondance de données de Fritzsche coûterait peut-être trop chère dans ce cas.

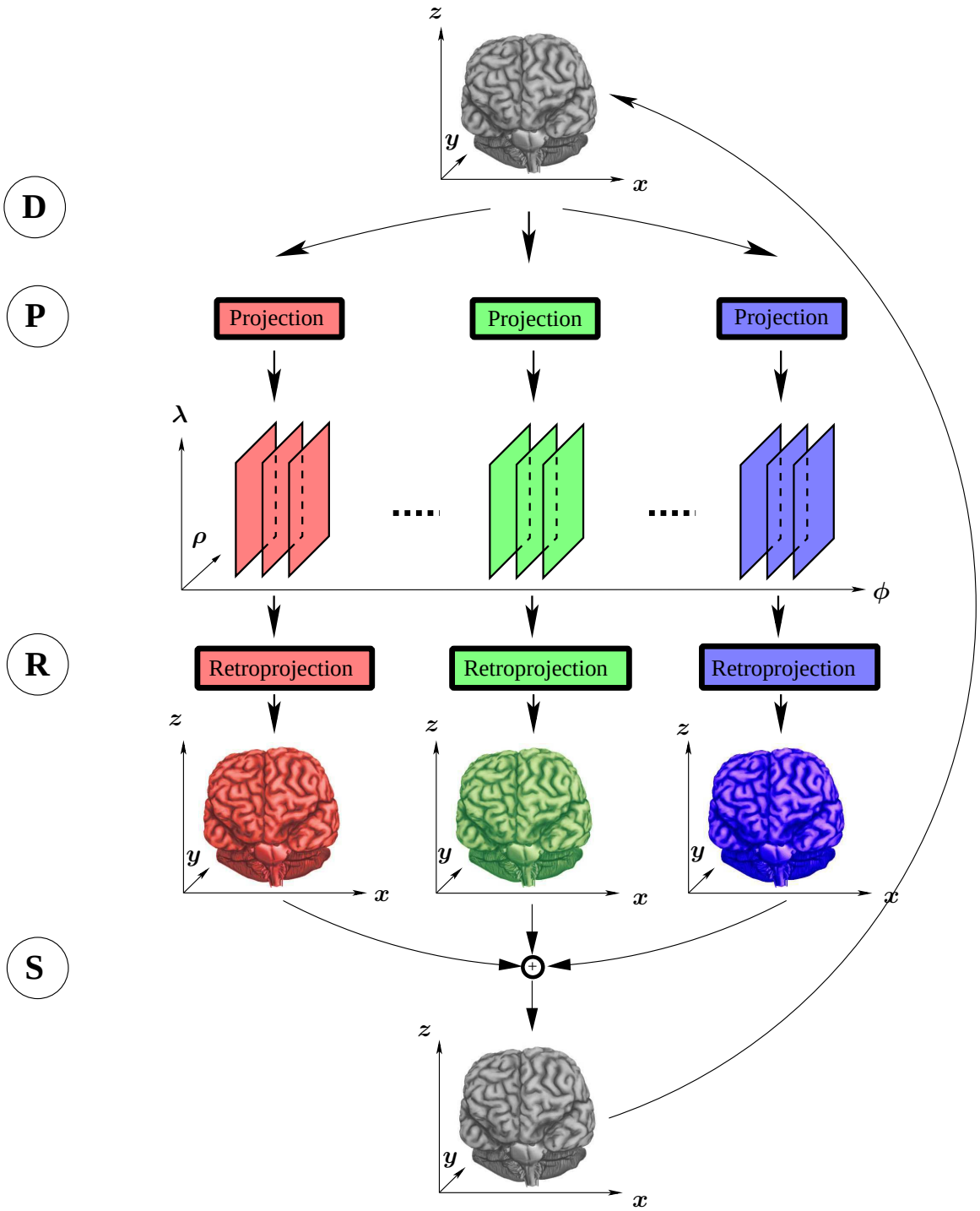


FIG. 2.10 – Parallélisation des algorithmes itératifs 3D avec une Décomposition dans l'Espace de Projection (DEP)

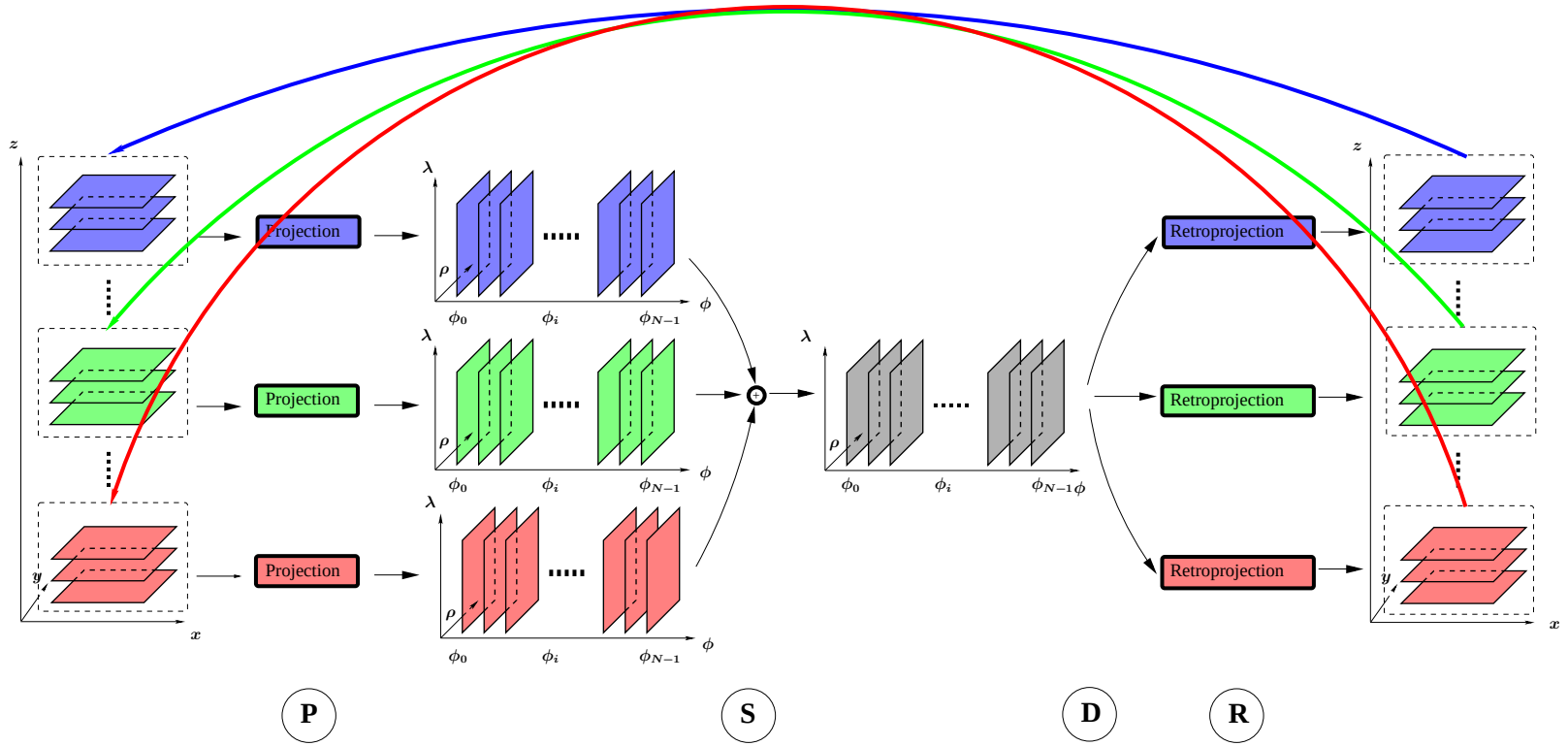


FIG. 2.11 – Parallélisation des algorithmes itératifs 3D avec une Décomposition dans l'Espace Image (DEI)

Parallélisation augmentant la vitesse de convergence

Avec une idée assez proche de celle de [Fritzsche 05], [Keesing 06] réussit à effectuer une partition totale à la fois de l'espace image et de l'espace de projection qui de plus permet d'augmenter la convergence d'un algorithme itératif en tomographie CT. Le volume est découpé en $\boldsymbol{n_{sous-volume}}$ sous-volumes, constitués chacun de $\boldsymbol{n_{coupe}}$ coupes transverses. Les coupes transverses d'un sous-volume sont assez espacées axialement pour ne pas correspondre à des données de projection communes. Ces sous-ensembles de coupes transverses indépendantes sont reconstruits l'un après l'autre sur la machine parallèle. C'est le traitement des coupes transverses d'un sous-volume qui est distribué sur les différents noeuds de calcul. A un processeur est affecté la mise à jour d'une coupe transverse ($\boldsymbol{c_0}$) et aux autres processeurs, les autres coupes transverses du même sous-volume ($\boldsymbol{c_n}$, $\boldsymbol{c_{2n}}$...). [Keesing 06] qualifie de sous itération une mise à jour des coupes transverses indépendantes d'un sous-volume. En effet, le volume n'est entièrement mis à jour qu'après le traitement de tous les sous-volumes de coupes transverses. Les sous-itérations suivantes effectuent de même pour les autres sous-volumes de coupes transverses indépendantes ($\boldsymbol{c_1}$, $\boldsymbol{c_{n+1}}$, $\boldsymbol{c_{2n+1}}$...) à la différence qu'elles se font à présent à partir de coupes transverses voisines ($\boldsymbol{c_0}$, $\boldsymbol{c_n}$, $\boldsymbol{c_{2n}}$...) déjà mises à jour. C'est ainsi que Keesing arrive à augmenter la vitesse de convergence puisqu'il est plus avantageux d'utiliser des coupes transverses voisines dont l'estimation est déjà améliorée. Le deuxième avantage de cette méthode est que sur une machine à mémoire partagée, les coupes transverses n'ont pas besoin d'être communiquées. Au final, Keesing obtient une accélération limite de 20 pour 32 processeurs ($\boldsymbol{\eta_{//} = 0.62}$) sur une SGI Altix 3000.

Parallélisation d'un algorithme local

[Deng 06a] a développé un algorithme itératif local en tomographie CT hélicoïdale, qui effectue une décomposition totale à la fois de l'espace de projection et de l'espace image. Son algorithme local permet de déterminer le sous-espace de projection nécessaire et suffisant à la reconstruction d'un sous-volume. Chaque noeud est alors affecté à la reconstruction d'un sous-volume à partir du sous-espace associé. Il n'existe plus alors de lourde étape de synchronisation de données et ne subsiste plus qu'une collecte des sous-volumes par un noeud maître à la fin de la reconstruction. Ainsi, les performances de la parallélisation augmentent linéairement avec la taille du réseau et n'atteignent pas un seuil comme c'est la cas pour les parallélisations DEI et DEP. Toutefois même si l'efficacité de parallélisation reste quasi constante plus le nombre de processeurs augmente, elle reste tout de même assez faible ($\boldsymbol{0.3 < \eta_{//} < 0.4}$). Ainsi sur une machine HPC (High Performance Computing) avec 16 bi-Opterons, une accélération de seulement 9.5 ($\boldsymbol{\eta_{//}}$=0.3) est ainsi obtenue. Cette faible performance peut s'expliquer par la décomposition de l'espace de projection en sous-espaces de projection non disjoints. Il existe en effet une redondance de traitement des données de projection. De plus, la répartition de charge déterminée par l'algorithme itératif local à partir du découpage en sous-volumes est complexe et non-régulière. Cette répartition de charge non-idéale crée alors des temps d'inactivité sur certains noeuds de calcul. A noter également que la parallélisation de l'algorithme local crée des effets de bord aux frontières des sous-volumes.

3 Accélérations sur machines parallèles

Dans cette section, nous nous intéresserons à l'accélération de la reconstruction sur différents types de machines parallèles : machines parallèles classiques (machines multiprocesseurs, clusters de PCs, processeurs multi-coeurs), les processeurs vectoriels (machine Cray), les processeurs spécifiques (processeur Cell et GPU) et enfin les architectures matérielles (FPGA/ASIC). Après une présentation des caractéristiques architecturales de ces machines, nous exposerons en détail pour chacune d'entre elles, la manière dont sont implémentés les algorithmes analytiques et itératifs de reconstruction ainsi que les différents facteurs d'accélération obtenus. Les temps de reconstruction des différentes implémentations sont regroupées en annexes C et D.

3.1 Architectures des machines parallèles

On classifie généralement les machines parallèles selon la centralisation ou la distribution du contrôle et de la mémoire. Pour le contrôle, on distingue les machines vectorielles ou SIMD (Single Instruction Multiple Data) des machines scalaires ou MIMD (Multiple Instruction Mutiple Data). Les premières sont contrôlées par un seul flot d'instructions qui s'exécute sur chaque unité de calcul, elles traitent ainsi des vecteurs de $\boldsymbol{n_{proc}}$ données. Les deuxièmes sont contrôlées par un flot d'instructions par processeur et traitent ainsi $\boldsymbol{n_{proc}}$ données scalaires. Pour la mémoire, on qualifie les architectures parallèles de machines à Mémoire Partagée (MP) ou de machines à Mémoire Distribuée (MD) selon que les accès en mémoire des processeurs se fassent sur une mémoire commune ou exclusivement sur la mémoire propre à chaque unité de traitement.

Les caractéristiques architecturales de processeurs classiques (Pentium 4 et Xeon) communément utilisés pour former les machines multiprocesseurs sont présentées dans le tableau 2.3. Dans ce tableau figurent également deux processeurs spécifiques : le processeur Cell et le processeur graphique GTX8800 de Nvidia.

3.1.1 Machines MIMD

Les réseaux d'interconnexion des machines multiprocesseurs MIMD sont définis par leur topologie, leur routage et leur flot de données. Différentes topologies hiérarchiques ou non de réseaux sont possibles : dynamiques en multi-étages ou statiques en bus, anneau, grille, tore 2D/3D, hypercube... Selon que la mémoire soit partagée ou distribuée, les coûts de communications diffèrent.

Machines à mémoire partagée (MP)

Les machines à mémoire partagée avec un bus commun ont l'avantage de ne pas avoir à communiquer explicitement leurs données entre noeuds de calcul. En effet, la mémoire centrale rend implicite la synchronisation de données du moment que l'exécution des *threads* est correctement synchronisée. Toutefois, les conflits pour l'accès à la mémoire centrale, pénalisent la parallélisation lors de l'exécution. Sur ces machines, la parallélisation se fait avec un modèle de programmation de type *openMP* (*Open Multi-Processing*).

Machines à mémoire distribuée (MD)

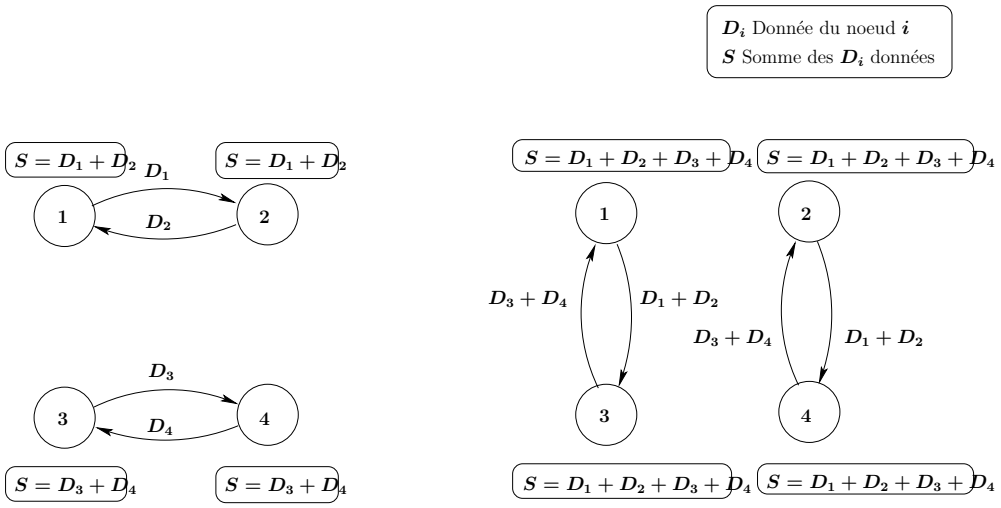
Les machines à mémoire distribuée utilisent un réseau de communication pour rendre accessible à tout noeud du réseau les données locales des autres processeurs. Les temps de ces communications dépendent entre autres de la topologie, de la bande-passante, du protocole de communication et de la charge du réseau. Dans la suite, nous ferons au sein des machines à mémoire partagée, la distinction entre les machines multiprocesseurs et les réseaux de PCs. En effet, les coûts et protocole de communication étant plus lourds pour les réseaux de PCs, elles n'offrent pas les mêmes possibilités que les machines multiprocesseurs plus performantes.

Pour les machines multiprocesseurs ou les réseaux de PCs, la réception, l'envoi, le *broadcast* et la collecte de données se font par passage de messages généralement à l'aide de la bibliothèque *MPI* (*Message Passing Interface*). Deux possibilités existent pour la synchronisation de données. Soit un maître ou client découpe la tâche en sous-tâches qu'il distribue à tous les esclaves ou serveurs et centralise l'envoie et la réception des données. C'est cette stratégie de communication qui est mise en place pour les réseaux de PCs. Soit tous les processeurs exécutent le même programme mais sur des données différentes, on parle alors de *Single Program Multiple Data (SPMD)*. C'est cette dernière stratégie qui est généralement employée sur les machines multiprocesseurs.

Pour le schéma maître/esclave, la synchronisation se fait en deux étapes : le maître collecte d'abord les résultats partiels des différents esclaves puis redistribue le résultat global. Pour réduire le nombre d'opérations de sommation lors de la synchronisation de données, différentes techniques de réduction/sommation peuvent être utilisées. Une réduction/sommation linéaire ou en arbre binaire [Jones 99] peut ainsi s'appliquer à l'étape de réception et/ou à celle d'envoi de données. Pour le schéma *SPMD*, les étapes de collecte et de redistribution des données se font en une seule même étape. Généralement, une topologie hiérarchique en anneaux ([Picard 98, Jones 05]) semblable à celle illustrée sur la figure 2.12, est utilisée. L'efficacité d'un tel réseau ne varie pas linéairement avec le nombre de processeurs. Les performances optimales sont atteintes pour $\boldsymbol{n_{proc} = 2^i}$: la synchronisation se fera alors avec i étapes de synchronisation entre paires de noeuds.

	Pentium 4 HT *Prescott 2M (640)*	Xeon dual core *Woodcrest (5160)*	IBM Cell BE	Nvidia G80 *8800 (GTX)*
ARCHITECTURE				
Type	**Simple coeur**	**Multi-coeur**	**Cell**	**GPU**
Nombre de proc.	1	2	PPE + 8*SPE	16 Blocks
Fréquence	3.2 GHz	3 GHz	3.2 GHz	1.35 GHz
PERFORMANCE DE CALCUL				
Trait. Vectorielle	1 (4)	1 (4)	4	8
FLOP/cycle	1	1	2 (1 MAD/cycle)	2 (1 MAD/cycle)
Perf. théorique	3.2 GFlops	6 GFlops	204.8 GFlops	345.6 GFlops
Evolution	*$\sim$1.4/an			*$\sim$2.2/an
MEMOIRE EXTERNE				
Fréquence mémoire	200 MHz	333 MHz	400 MHz	900 MHz
type de DRAM	QDR	QDR	XDR	GDDR3
Taille du bus	128 bit	128 bit	8 voies de 8 bits	384 bit
Débit théorique	6.4 Go/s	10.6 Go/s	25.6 Go/s	86.4 Go/s
MEMOIRE CACHE ET MEMOIRE LOCALE				
L1 Instructions	12 Kμop	32 Ko/coeur	32 Ko (PPE)	8* ?
L1 Données	16 Ko	32 Ko/coeur	32 Ko (PPE)	8*8 Ko
L2	2 Mo	4 Mo en commun	512 Ko (PPE)	6* ?
Mémoire locale			256 Ko/SPE soit 2 Mo	16 Ko/Block soit 256 Ko
OCCUPATION EN SURFACE				
Transistors	169 mil.	291 mil.	234 mil.	681 mil.
Technologie	90 nm	65 nm	90 nm	90 nm
Surface	135 mm^2	144 mm^2	221 mm^2	484 mm^2

TAB. 2.3 – Caractéristiques architecturales d'un Pentium 4, d'un Xeon double coeur, du Cell et du GTX 8800



(a) $\mathbf{1}^{re}$ étape (b) $\mathbf{2}^{me}$ étape

FIG. 2.12 – Synchronisation des données SPMD d'une machine MD (4 noeuds) avec une topologie en anneau hiérarchique.

3.1.2 Processeurs vectoriels

Les supercalculateurs vectoriels de type *Cray* permettent la manipulation et le traitement de tableaux de vecteurs (64 vecteurs de 64 bits pour le Cray J90). Devenues populaires dans les années 70/80, ces machines étaient utilisées pour les calculs scientifiques intensifs. Son architecture est en effet particulièrement intéressante pour les calculs matriciels. Cependant, le rapport performance/coût et les progrès technologique des CPUs rendent aujourd'hui plus confidentiels l'utilisation de ces processeurs/supercalculateurs.

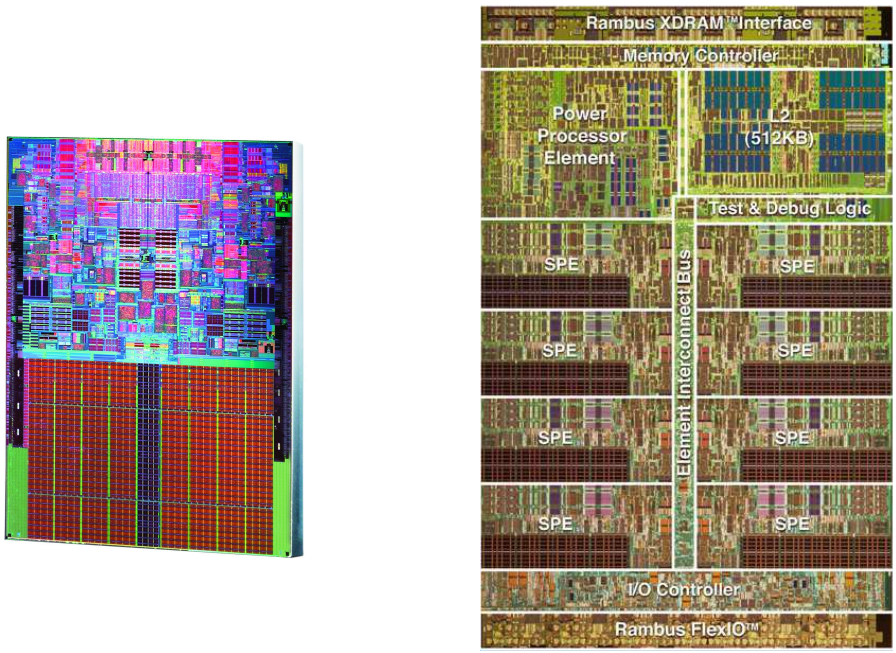
3.1.3 Le processeur Cell

Le *Cell* dont l'architecture est représentée sur la figure 2.14, est un processeur *general purpose* composé d'un coeur principal, le *PowerPC Processing Element* (PPE) et de 8 coeurs spécifiques, les *Synergistic Processing Element* (SPE). Le PPE est un CPU classique avec des caches de niveau L1 (64 Ko) et L2 (512 Mo). Il joue le rôle de chef d'orchestre, lançant l'exécution des calculs sur les 8 SPEs. L'*Element Interconnect Bus* comporte 4 anneaux 128 bits supportant des transferts multiples. Il relie tous les composants du *cell* entre eux : PPE, SPEs, cache L2 et contrôleurs mémoire et entrées/sorties.

Chaque SPE possède une unité de calcul vectoriel, le *Synergistic exécution Unit* (SXU), un contrôleur mémoire, *Memory Flow Controller* (MFC) et 256 Ko de mémoire locale haute vitesse à 3.2 GHz, *Local Store* (LS). Un SPE correspond à un processeur vectoriel indépendant possédant 128 registres 128 bits, 4 unités de calcul en virgule flottante double précision et 4 unités de calcul entiers. Il peut exécuter 2 instructions par

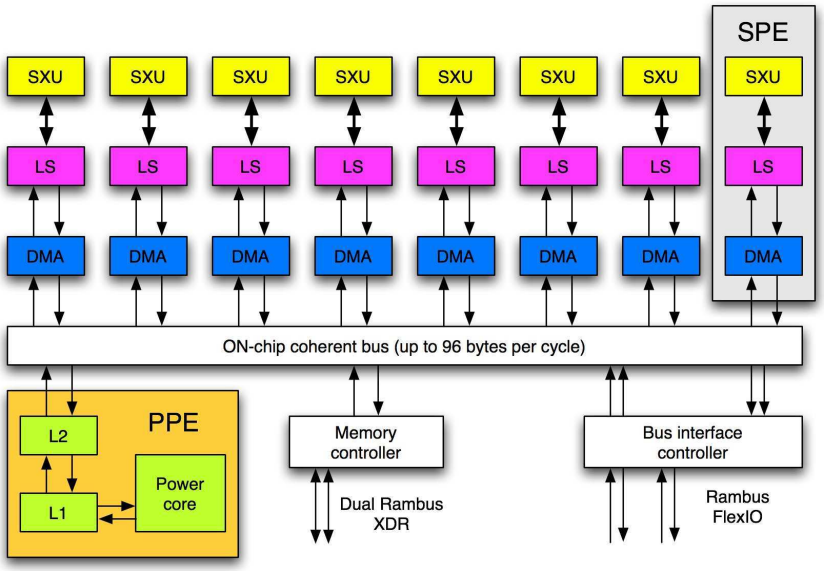
cycle grâce à ses deux pipelines. Le pipeline pair exécute les multiplications-additions (*mad*) et le pipeline impair exécute les *load* et *store*. Ainsi, les performances théoriques de calcul du Cell atteignent 204.8 GFlops. Les SPEs ne possèdent pas de mémoire cache. Toutefois, les concepteurs du Cell ([Sakamoto 05]) laissent la possibilité au développeur de masquer les temps de transferts mémoire entre la mémoire principale du PPE et les mémoires locales des SPEs grâce des techniques de synchronisation pilotées par le PPE utilisant les DMAs (*Direct Memory Access*). La mémoire locale des SPEs de 256 Ko (celle-ci contient également le code, le tas et la pile) possède une organisation en *buffers* circulaires de type FIFO afin d'être adaptée aux transferts de données par paquets. Le découpage des données en paquet est à la charge du développeur. L'accès à la mémoire externe se fait via le *Memory Interface Controler* (MIC). Ce double contrôleur mémoire XDR (XDRAM) offre un très haut débit de 25,6 Go par seconde.

La figure 2.13 compare la puce du Cell à celle d'un Pentium 4. On peut observer qu'une grande partie de la puce du Pentium 4 est destinée à la mémoire cache de niveau 2 (113 millions de transistors pour les 2 Mo de cache soit 67 % du nombre total de transistors). Le Cell possède en effet moins de mémoire par coeur : les SPEs ont une mémoire locale propre (256 Ko) et seul le PPE a un cache de niveau L2 (512 Ko).

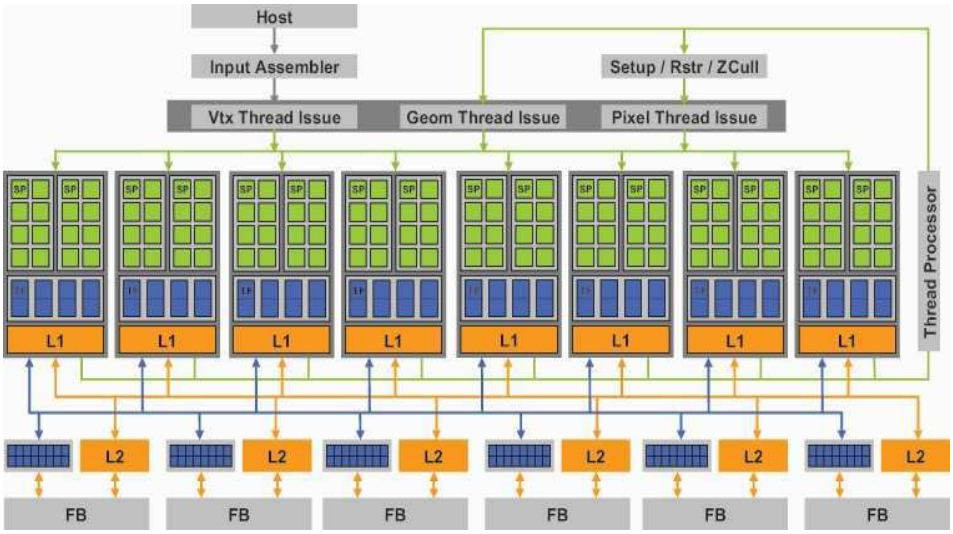


(a) Xeon dual-core (b) Cell BE

FIG. 2.13 – Puces du Xeon double coeur (a) et du Cell (b).



(a) Cell BE



(b) GTX 8800

FIG. 2.14 – Architectures du Cell (a) et du GTX 8800 (b).

3.1.4 Les *Graphics Processing Units* (GPU)

Les GPUs sont conçus à l'origine pour effectuer du rendu de polygones texturés, autrement dit à projeter une scène 3D composée de polygones sur une grille 2D de pixels correspondant à l'écran. Mais l'introduction d'une plus grande précision de calcul (en 2002/2003) et d'une plus grande programmabilité (en 2004/2005), ont rendu plus accessibles les ressources matérielles des GPUs. Ces machines sont devenues ainsi une solution attrayante pour l'accélération des calculs scientifiques en général. On parle ainsi de *General Purpose GPU* (GPGPU).

A l'origine un pipeline graphique Comme illustré sur la figure 2.15, le rendu de polygones texturés s'effectue via le pipeline graphique des GPUs. Les trois étages principaux de ce pipeline sont : (1) la transformation géométrique des sommets des polygones avec le *vertex shader*, (2) la transformation des polygones en une image pixelisée avec le *rasterizer* et (3) le traitement des pixels dans le *fragment shader*. Le *vertex shader* transforme les sommets (ou *vertex*) du polygone positionné dans l'espace 3D vers le plan de l'écran 2D à l'aide d'une matrice de projection (perspective ou non). Le *rasterizer* transforme le polygone projeté sur l'écran en une série de pixels ou *fragments*. Il calcule également les attributs de chaque pixel (notamment les indices de la texture 2D attachée au polygone de la scène 3D) à partir des attributs des vertex par une interpolation bilinéaire câblée. Enfin le *fragment shader* traite un à un les pixels de l'image. C'est à ce moment là que la texture 2D peut être plaquée sur les pixels à partir de leurs indices de texture.

Mutations en une machine parallèle multiprocesseurs Dans la perspective de transformer le pipeline graphique en une machine parallèle, Nvidia a unifié l'architecture des processeurs du *vertex shader* et du *fragment shader* de sa dernière génération de GPU (G80). Ces GPUs deviennent ainsi des machines multiprocesseurs et le développement se fait via une interface de programmation haut niveau (CUDA : *Compute Unified Device Architecture*) qui facilite la parallélisation du code sur les *stream processors* (128 sur le GTX 8800). Le GTX 8800 dont l'architecture est représentée sur la figure 2.14, possède 16 blocs de traitement composés d'un processeur vectoriel avec 8 *streaming processors* (SP) et d'unités de préchargement des textures (TA : *Texture Addressing*) et de filtrage des textures (TF : *Texture Filtering*). Chaque SP possède un multiplieur additionneur (*mad*) pour des flottants simple précision. Le GTX 880 atteint ainsi une performance théorique de calcul de 345.6 GFlops. Les applications graphiques étant particulièrement sensibles aux temps d'accès mémoire, le GTX 8800 possède un débit mémoire important (86.4 Go/s), 16 Ko de mémoire locale par bloc de traitement, ainsi que de nombreux caches mémoire en lecture seule (un cache L1 par couple de blocs de traitement et un cache L2 par banc mémoire).

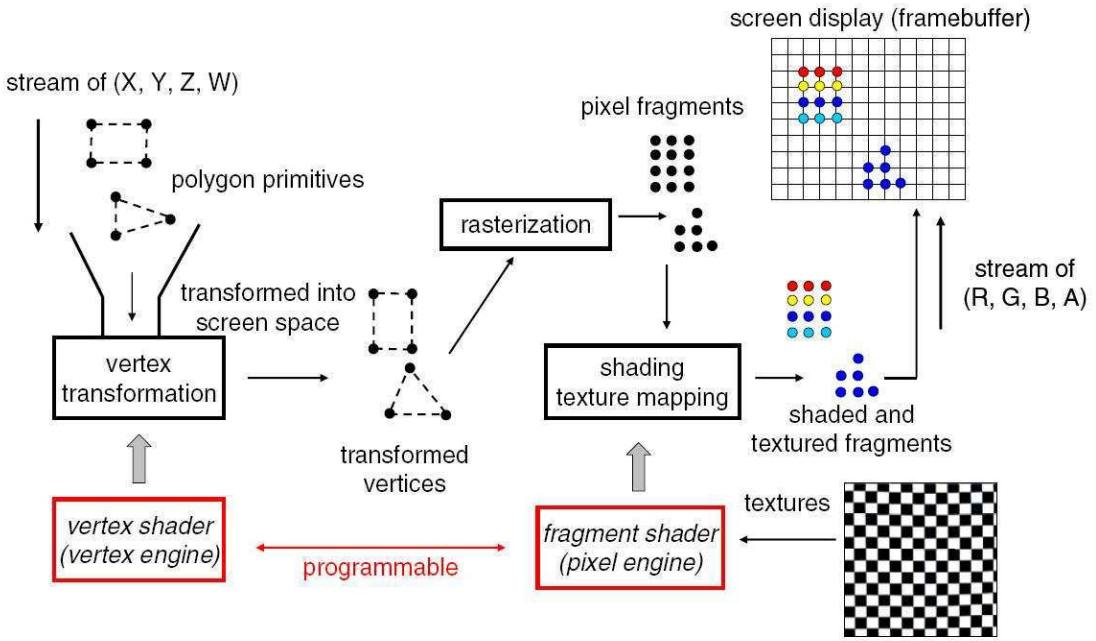


FIG. 2.15 – Pipeline graphique des GPUs ([Xu 07])

3.1.5 Architectures matérielles FPGA/ASICs

Si les ASICs (*Application-Specific Integrated Circuit*) sont des circuits intégrés conçus sur mesure pour une ou plusieurs applications, les FPGAs (*Field-Programmable Gate Array*) sont des circuits logiques programmables. Certes les ASICS offrent des performances plus intéressantes que les FPGAs en terme de fréquence de fonctionnement, de surface (jusqu'à plusieurs millions de portes logiques) et de consommation, mais vu leurs coûts de production initiaux importants, les ASICs sont une solution économiquement envisageable seulement pour les gros volumes de production ($> 100\ 000$).

Grâce à leur nombreuses cellules logiques élémentaires librement assemblables, les FPGAs sont une solution technologique plus flexible, et depuis le début des années 80 de plus en plus mature. Elle représente à présent un bon compromis entre performance et coût de développement (une carte de développement coûte moins de 1 000 euros). C'est une solution adaptée au prototypage ou aux besoins en petit quantité de circuits intégrés spécifiques. Parmi tous les fabricants de FPGA (*Actel, Atmel, Altera, Xilinx* ...), nous nous intéresserons plus particulièrement aux FPGAs de la société Xilinx. Ses FPGAs utilisent des cellules SRAM pour reprogrammer chacun de ses CLBs (Configurable Logic Block) mais également pour reconfigurer le schéma d'interconnexion des CLBs. Des multiplieurs matériels et des IPs de communication sont notamment présents sur ces circuits. Les Virtex de Xilinx sont déclinés en trois gammes de produits :

– les LX dédiés à un usage générique, sont composés essentiellement de logique programmable,

– les FX dédiés au traitement du signal, se voient attribuer une plus grande proportion de multiplieurs accumulateurs (ils sont un concurrencent direct des DSPs),
– les SX dédiés aux systèmes embarqués, possèdent des processeurs embarqués comme les PPCs.

La famille des Virtex 4 FX, sortie en 2004, est composée selon les modèles de 10 000 à 140 000 portes logiques équivalentes, de 32 à 192 multiplieurs accumulateurs 18 bits, de 10 à 125 Ko de mémoire RAM et de 1 à 2 PPCs . Ce circuit peut tourner jusqu'à 550 MHz mais la fréquence de fonctionnement dépend avant tout de l'architecture du bloc matériel implémenté, appelé IP (Intellectual Property).

3.1.6 Classification des machines étudiées

En nous appuyant sur les caractéristiques architecturales de chacune de ces machines, nous avons placé sur la figure 2.16 les différentes machines parallèles figurant dans cette étude, selon leur degré de distribution du contrôle et de la mémoire. Les architectures FPGA/ASIC ne figurent pas dans cette classification étant donne qu'idéalement ils couvrent tout le spectre du graphe. Nous justifions ci dessous les proportions de distribution du contrôle et de la mémoire que nous avons attribuées à chacune de ces machines.

– Les machines MIMD ont un contrôle entièrement distribué dans l'hypothèse où les instructions vectorielles de type SSE des processeurs actuels ne sont pas utilisées. Outre les machines multiprocesseurs à mémoire partagée et celles à mémoire distribués, nous avons ajouter les machines multiprocesseurs à Mémoire Logiquement Distribuée (MLP) qui grâce à un unique système d'exploitation, ont la possibilité de rendre accessible à chaque processeur, les mémoires des autres processeurs. Toutefois, selon qu'une donnée soit sur la mémoire locale ou sur la mémoire d'un autre processeur, le temps d'accès ne sera pas le même. On parle ainsi de *Non Uniform Memory Access* (NUMA). Nous rajoutons également à cette classification, les machines multiprocesseurs à Mémoire Hybride (MH) qui correspondent aux réseaux de processeurs symétriques (*Symmetric multiprocesseurs*) possédant 2, 4 voire 8 coeurs identiques. La mémoire est alors considérée à mi chemin entre une mémoire partagée et une mémoire distribuée.
– Les machines vectorielles ont un contrôle très faiblement distribué (4 unités traitant des tableaux de 64 vecteurs de taille 64 bits pour la machine Cray J90) et une mémoire partagée.
– Le Cell a un contrôle au 2/3 distribué (8 unités traitant des vecteurs de taille 4 flottants) et une mémoire essentiellement distribuée avec les 256 Ko de mémoire local que dispose chaque SPE et partagée dans une moindre mesure avec les accès indirects à la mémoire partagée constitués par les transferts mémoire du PPE vers les SPEs.
– Les GPUs ont un contrôle au 2/3 distribué (16 unités traitant des vecteurs de taille 8 flottants) et une mémoire essentiellement partagée avec l'accès à la mémoire vidéo (750 Mo) et dans une moindre mesure distribuée avec l'accès aux mémoires locales (16 Ko).

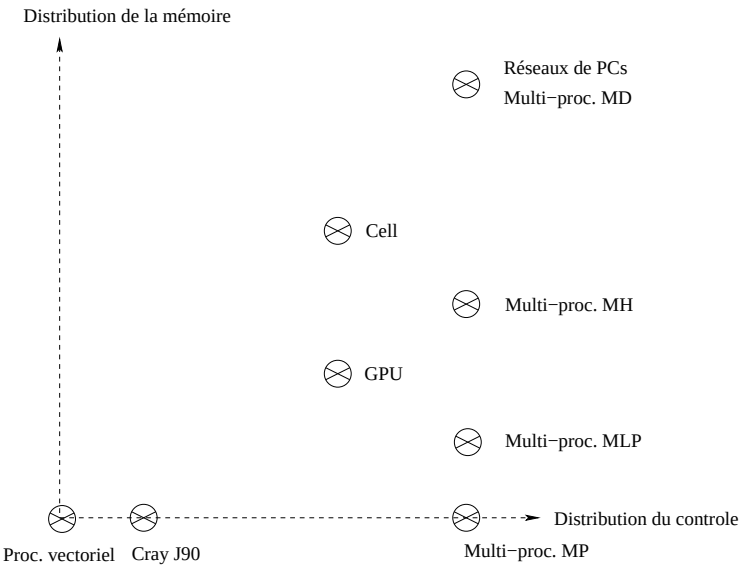


FIG. 2.16 – Classification des machines parallèles selon la distribution du contrôle et de la mémoire.

3.2 Accélération sur machines MIMD

Nous passons ici en revue les accélérations obtenues sur machines MIMD selon le type de mémoire : partagée, distribuée sur un réseaux performant (machines multiprocesseurs), distribués sur un réseau peu performant (cluster de PCs), logiquement partagée et hybride.

3.2.1 Machines multiprocesseurs à Mémoire Partagée (MP)

La mémoire centrale des machines MIMD à mémoire partagée rend inutile la Distribution de donnée (étape D) vers chaque noeud de calcul. Ainsi pour la parallélisation DEP, le volume n'a pas besoin d'être renvoyé vers tous les noeuds de calcul puisque chacun y a accès via la mémoire centrale. Mais le talon d'Achille de ces machines se situe au niveau du bus mémoire. En effet, les conflits d'accès mémoire pénalisent fortement le potentiel de parallélisation de ces machines. Ainsi [**?**] observe lors de la parallélisation DEP d'un algorithme mode-liste OSEM en PET 3D, que les performances chutent assez vite sur une machine MP avec 8 Opterons. L'accélération est de 3.5 pour 4 processeurs ($\boldsymbol{\eta_{/\!/} = 0.87}$) et seulement de 5 pour 8 processeurs ($\boldsymbol{\eta_{/\!/} = 0.62}$). De la même manière, [Smith 07] effectue une parallélisation DEP sur une machine MP constituée de 4 Opterons double coeurs pour l'algorithme OSEM en mode liste à l'aide d'une directive de compilation *openMP* sur la boucle sur les LORs. Il obtient un facteur d'accélération de seulement 4.8 pour 8 coeurs ($\boldsymbol{\eta_{/\!/} = 0.6}$).

3.2.2 Machines multiprocesseurs à Mémoire Distribuée (MD)

Les machines multiprocesseurs à mémoire distribuée ont été largement utilisées pour accélérer la rétroprojection 3D et les algorithmes itératifs 3D.

Rétroprojection 3D

[Laurent 98] a observé une bonne parallélisation de l'algorithme FDK sur plusieurs types de machines. Que ce soit sur une ferme de 16 processeurs Alpha (réseau d'interconnexion Ethernet à 200 Mb/s), sur la machine parallèle SP1 d'IBM de 32 processeurs (réseau multi-étage à 20 Mb/s), ou bien encore sur une machine massivement parallèle Cray T3D avec 64 noeuds double coeurs (réseau en tore 3D), l'accélération de la reconstruction reste quasi linéairement proportionnelle au nombre de processeurs. Pour la reconstruction de volume de taille $\mathbf{128^3}$, l'auteur obtient les perfomances suivantes : $\boldsymbol{\eta_{/\!/}}$=0.95 pour les 16 processeurs de la ferme d'AXP, $\boldsymbol{\eta_{/\!/}}$=0.9 pour les 32 processeurs du SP1, $\boldsymbol{\eta_{/\!/}}$=1.38 pour les 128 processeurs du Cray T3D. Comparées avec une machine SUN classique de l'époque, ces machines permettent ainsi respectivement des facteurs d'accélération de 30, 100 et 500.

[Deng 06b, Ni 06, He 06] ont effectué la parallélisation de l'algorithme de Katsevitch sur différentes machines MD. A partir d'une modélisation des coûts de traitement et de communication dans un réseau homogène de machines MD de taille assez grande, [Ni 06] obtient un seuil théorique $\boldsymbol{S}$ fonction du temps de filtrage par élément de projection, du temps de transfert d'un octet, du temps de rétroprojection d'un voxel et du nombre de voxels et d'éléments de projection :

$$\boldsymbol{S = \frac{t_{filtrage} + k \cdot t_{retro}}{k \cdot t_{transfert}}} \text{ avec } \boldsymbol{k = \frac{N_{voxel}}{N_{proj}}} \tag{2.3}$$

Ce modèle de prédiction de performance a été validé lors de la parallélisation de l'algorithme sur le TeraGrid de la NCSA (National Center for Supercomputing Applications). Ce centre possède des machines figurant dans le top 500 des supercalculateurs les plus performants au monde. Pour la reconstruction d'un volume $\mathbf{256^3}$ et **3501** plans de projection de taille $\mathbf{300 * 40}$, le seuil de 150 est atteint dès 150 processeurs ($\boldsymbol{\eta_{/\!/}}$=1). Pour un volume $\mathbf{512^3}$ et **7001** plans de projection de taille $\mathbf{500 * 70}$, un seuil de 300 est atteint dès 300 processeurs ($\boldsymbol{\eta_{/\!/}}$=1). A noter que comme [Laurent 98], [Deng 06b] observe des performances sur-linéaires à partir d'un dizaine de processeurs. En effet, les performances sur un seul processeur sont pénalisées par une allocation mémoire trop grande pour un simple processeur, alors que celles sur plusieurs processeurs ne sont pas limitées par ces problèmes d'allocation mémoire puisque distribuées sur les $\boldsymbol{n_{proc}}$ processeurs.

Nous pouvons conclure donc que la parallélisation des algorithmes de rétroprojection filtrée se passent très bien sur machines parallèles MIMD. Le gain en performance augmente linéairement avec le nombre de processeurs. Mais cela reste vrai tant que le problème est de taille assez grande. C'est ce qu'observe [Deng 06b] avec une machine composée de 16 bi-Opterons pour une taille fixe des données de projection : l'accélération limite est de 11 pour 16 processeurs ($\boldsymbol{\eta_{/\!/}}$=0.69) lors de la reconstruction d'un volume

$\mathbf{128^3}$ et de 26 pour 32 processeurs ($\boldsymbol{\eta_{/\!/}}$=0.81) lors de la reconstruction d'un volume de $\mathbf{256^3}$.

Algorithmes itératifs 3D

La stratégie de parallélisation (DEP ou DEI) est souvent choisie pour des raisons arbitraires ou peu justifiées. Toutefois, des optimisations comme le masquage des communications ou la prise en compte des temps de lecture des données du scanner peuvent mieux éclairer le choix de la stratégie de parallélisation.

[Laurent 98] part du constat que les données de projection sont plus importantes que le volume pour préférer une parallélisation DEP pour l'algorithme *Block ART* en tomographie CT cone-beam. Comme pour la parallélisation de la rétroprojection filtrée, il a effectué des mesures sur une ferme de 16 processeurs Alpha, sur une machine parallèle SP1 d'IBM de 32 processeurs et sur une machine massivement parallèle Cray T3D avec 64 noeuds double coeurs. [Laurent 98] évalue le pouvoir d'accélération pour un volume de $\mathbf{64^3}$ à 14 pour la ferme de 16 processeurs Alpha ($\boldsymbol{\eta_{/\!/}}$=0.87), 23 pour les 32 processeurs du SP1 ($\boldsymbol{\eta_{/\!/}}$=0.72) et 100 pour les 128 noeuds de calcul du Cray T3D ($\boldsymbol{\eta_{/\!/}}$=0.79). Comparées avec une machine SUN4, ces machines permettent respectivement des facteurs d'accélération de 34, 43 et 53 pour un volume $\mathbf{64^3}$.

[Jones 03] compare les deux stratégies de parallélisation en OSEM 3D avec un cluster homogène de 9 Xeons double coeur. La synchronisation de données est faite non pas avec un schéma maître/esclave mais en *SPMD* avec une topologie hiérarchique en anneau (voir 3.1.1). [Jones 03] estime que malgré une synchronisation de données plus importante due à la taille des données plus importante que la taille du volume, la parallélisation DEI est plus avantageuse car elle autorise le masquage de la communication par le réseau de la projection $\boldsymbol{S(\phi_i)}$ avec la projection $\boldsymbol{P(\phi_{i+1})}$ par les noeuds de calcul. Ce masquage des communications est possible car le temps de traitement de l'étape de Projection est supérieure aux temps de communication des données de projection. Expérimentalement, il observe alors que malgré le handicap d'une étape de synchronisation supplémentaire, la parallélisation DEI offre des performances légèrement meilleures à la parallélisation DEP. L'étape de synchronisation supplémentaire que [Jones 03] rajoute, correspond à la recopie de tous les sous-volumes reconstruits sur chaque noeud de calcul après l'étape de rétroprojection. Il obtient ainsi avec 9 noeuds (18 processeurs), une accélération d'un facteur 16.8 ($\boldsymbol{\eta_{/\!/} = 0.93}$) pour la parallélisation DEI et 16.6 ($\boldsymbol{\eta_{/\!/} = 0.92}$) pour la parallélisation DEP. Ainsi s'il on se passe de l'étape de recopie des sous volumes qui n'est pas primordiale, nous pouvons conclure qu'au vu des résultats de [Jones 03], la parallélisation DEI semble préférable à la parallélisation DEP.

[Schellmann 07] n'arrive pas aux même conclusions que [Jones 03]. En effet, il prédit avec un modèle analytique puis observe sur un cluster de 200 Xeons double coeurs (mais il n'en utilise qu'un coeur), que la parallélisation DEP surpasse largement la parallélisation DEI pour l'algorithme OSEM en mode liste. Ce désaccord entre [Schellmann 07] et [Jones 03] s'explique par la prise en compte par [Schellmann 07] des coûts de lecture des données $\boldsymbol{y_i}$ acquis par le scanner. Ces dernières sont en effet lues à chaque itération après l'étape de projection lors de la comparaison des projections estimées $\boldsymbol{\hat{y}_i}$ avec les données

réelles $\boldsymbol{y_i}$ (voir chapitre 1). [Schellmann 07] obtient ainsi des facteurs d'accélération pour les parallélisations DEP et DEI respectivement de 25 ($\boldsymbol{\eta_{/\!/} = 0.78}$) et 7 ($\boldsymbol{\eta_{/\!/} = 0.22}$) pour 32 processeurs et 42 ($\boldsymbol{\eta_{/\!/} = 0.65}$) et 6 ($\boldsymbol{\eta_{/\!/} = 0.09}$) pour 64 processeurs. En conclusion, pour [Schellmann 07] la parallélisation DEP et DEI offrent des performances assez proches lorsque le nombre de noeuds de calcul est limité (inférieur à 10) mais au-delà, la parallélisation DEP est préférable.

3.2.3 Réseaux de PCs

De nombreuses études ([Munz 98, Vollmar 00, Vollmar 02, Shattuck 02, Kontaxakis 02, Li 04b, Brasse 05, He 06]) ont cherché à constituer des machines parallèles MIMD aux performances compétitives en ré-utilisant tout simplement les PCs standards disponibles dans un service clinique.

Algorithmes itératifs 2D

Pour ces machines, la parallélisation par tâche donne de bonnes performances pour les réseaux homogènes : [Vollmar 02] obtient ainsi un facteur d'accélération de 21.3 avec 7 noeuds de 4 Xeons ($\boldsymbol{\eta_{/\!/}}$=0.89) et [Munz 98] obtient une accélération d'un facteur 25 avec un cluster de 30 stations ($\boldsymbol{\eta_{/\!/}}$=0.83). Pour les réseaux hétérogènes, l'accélération se retrouve assez vite limitée à partir d'une dizaine de processeurs comme l'observe [Vollmar 00]. Ce dernier atteint ainsi une accélération limite de 8 avec 10 machines ($\boldsymbol{\eta_{/\!/}}$=0.8).

Rétroprojection 3D

[Brasse 05] construit ainsi un système de reconstruction en ligne pour la tomographie CT constitué d'un réseau homogène de PCs (cluster *Beowulf*). Les pré-traitements (étape P) se font sur 3 PCs en série, la rétroprojection sur 8 PCs en parallèle, et la collecte des sous-volumes reconstruits sur un dernier PC. Il réussit ainsi à traiter en temps réel le flot de *viewgrams* sortant du système d'acquisition. Toutefois, l'efficacité d'accélération obtenue n'est pas élevée : accélération d'un facteur 5 avec ces $\mathbf{3 + 8 + 1 = 12}$ PCs ($\boldsymbol{\eta_{/\!/}}$=0.41) et prédiction d'accélération limite de 9.4 avec $\mathbf{3 + 16 + 1 = 20}$ PCs ($\boldsymbol{\eta_{/\!/}}$=0.47). [He 06] observe des performances équivalentes pour la parallélisation de l'algorithme de Katsevitch avec un réseau hétérogène de PCs *peer-to-peer* où chaque PC peut devenir le maître et les autres PCs devenant ses esclaves. Il obtient un facteur limite d'accélération limite de 10 avec 16 PCs ($\boldsymbol{\eta_{/\!/}}$=0.62). Dans son réseau de PCs, le maître gère une liste de données de projection de différentes tailles et une liste de processus qu'il distribue selon la vitesse de traitement des esclaves. Cela lui permet de ne pas être trop pénalisé par l'hétérogénéité du réseau.

Algorithmes itératifs 3D

Pour les algorithmes itératifs 3D, la parallélisation sur clusters de PCs atteint ses limites dès 10 PCs. [Shattuck 02] pour la parallélisation DEP en PET 3D de l'algorithme itératif PCG (*Preconditioned Conjugate Gradient*) avec un cluster homogène de

bi-Pentium 3 (un maître connecté à 8 esclaves), obtient ainsi une accélération de 5 avec 9 processeurs ($\boldsymbol{\eta}_{/\!/}$=0.55). [Li 04b] et [He 06] effectuent également une parallélisation DEP mais pour un algorithme EM en tomographie CT hélicoïdale sur des clusters hétérogènes de PCs. [Li 04b] atteint une limite d'accélération de 5 dès 8 machines ($\boldsymbol{\eta}_{/\!/}$=0.62) et [He 06] une limite de 3 avec 10 machines ($\boldsymbol{\eta}_{/\!/}$=0.3). Dans cette étude, l'architecture du réseau ne semble pas bénéficier de techniques avancées de réduction/sommation puisque les esclaves envoient chacun leur tour leur mise à jour du volume au maître. Les communications augmentent donc linéairement avec le nombre de processeurs. On peut expliquer également la moins bonne performance obtenue par [He 06] avec l'algorithme EM (accélération de 3) qu'avec l'algorithme de Katsevitch (accélération de 10) par le nombre plus important de communications globales (1 contre 3).

3.2.4 Machines multiprocesseurs à Mémoire Logiquement Partagée (MLP)

Si les machines à Mémoire Partagée (MP) donnent d'assez mauvaises performances, les machines à Mémoire Logiquement Partagée (MLP) donnent de meilleurs résultats proches de ceux des machines MD. Ainsi [Kole 05] compare pour l'algorithme itératif OSC (*Ordered Subset Convex*) en tomographie CT, les performances d'une parallélisation DEP sur une machine MLP avec celles d'une parallélisation DEI sur une machine MD. Tout comme [Jones 03], il observe globalement peu de différences entre les deux stratégies : accélération de 33 avec 40 processeurs ($\boldsymbol{\eta}_{/\!/} = \mathbf{0.82}$) pour la machine MP, et une accélération de 25 avec 32 processeurs ($\boldsymbol{\eta}_{/\!/} = \mathbf{0.78}$) pour la machine MD. Il en conclut que vu le coût élevé des machines MP ou MLP, la parallélisation DEI sur machines MD reste la meilleure solution. Toutefois si une machine MLP est à disposition, la parallélisation DEP sur machines MLP peut être préférée pour sa plus grande simplicité de programmation à l'aide d'*openMP*.

3.2.5 Machines multiprocesseurs à Mémoire Hybride (MH)

Avec les nouveaux processeurs multi-coeurs, deux niveaux de parallélisation apparaissent sur les réseaux multiprocesseurs : (1) au niveau du noeud de calcul qui peut être considéré comme une machine à Mémoire Partagée (MP) exécutant plusieurs *threads* en parallèle sur les différents coeurs du processeur et (2) au niveau du réseau qui peut être considéré comme une machine à Mémoire Distribuée (MD). [Jones 04] observe ainsi qu'une programmation mixte *OpenMP/MPI* semble la plus appropriée pour tirer profit du caractère hybride de ces clusters de machines multi-coeurs. Le parallélisme de *threads* est obtenu avec une programmation *openMP*, et le parallélisme sur le réseau se fait classiquement à l'aide de la librairie *MPI*. Toutefois, il optimise ainsi les performances de seulement **10%** par rapport à une programmation *MPI* classique.

3.2.6 Conclusion concernant les machines MIMD

Comme nous pouvons le constater sur le graphe 2.17, l'efficacité de parallélisation est variable selon le type d'algorithme implémenté et le type de machines MIMD utilisées.

Tout d'abord, nous pouvons constater que les clusters de PC permettent de reconstruire de manière satisfaisante les algorithmes 2D (et ce depuis une dizaine d'années) et de manière nettement insuffisante, les algorithmes de reconstruction 3D. Pour les algorithmes de rétroprojection 3D, la parallélisation DEI atteint ses limites assez rapidement avec une efficacité $\boldsymbol{\eta_{/\!/}}$ aux alentours de 0,6 pour 20 noeuds de calcul. Pour les algorithmes itératifs 3D, une limite de $\boldsymbol{\eta_{/\!/} = 0,6}$ est atteinte dès une dizaine de processeurs. Si les algorithmes itératifs 2D se parallélisent mieux que les algorithmes itératifs 3D, c'est tout simplement parce qu'ils demandent beaucoup moins d'étapes de synchronisation de données qu'en 3D. Or les réseaux de PCs utilisent des techniques de synchronisation de données bien peu efficaces avec un maître et plusieurs esclaves. De plus, si le réseau est hétérogène, la répartition des tâches devient difficile et alourdit immanquablement la stratégie de parallélisation.

De même, les machines à mémoire partagée offrent des performances décevantes. Pour les algorithmes itératifs 3D, elles sont intéressantes pour une taille de 4 noeuds ($\boldsymbol{\eta_{/\!/}}$ compris entre 0,7 et 0,9), moins intéressante pour 8 noeuds ($\boldsymbol{\eta_{/\!/} = 0,6}$) et véritablement inefficaces pour des tailles supérieures à 8 noeuds. Les conflits d'accès à la mémoire centrale constituent la raison majeure de ces résultats décevants.

A l'inverse, nous pouvons observer que les machines à mémoire distribuée offrent un bon, voire un très bon support pour la parallélisation des algorithmes de reconstruction. Ainsi, la parallélisation DEI des algorithmes de rétroprojection filtrée 3D sur ces machines fonctionnent très bien : l'efficacité de parallélisation $\boldsymbol{\eta_{/\!/}}$ est supérieure à 0,9 pour des machines d'une taille de 10 à 300 processeurs. Pour les algorithmes itératifs, les performances sur les machines à mémoire distribuée sont légèrement inférieures à celles observées pour la rétroprojection 3D, mais restent tout de même satisfaisante. Pour les algorithmes itératifs 2D, une parallélisation par tâche est bien adaptée au problème en permettant une accélération efficace ($\boldsymbol{\eta_{/\!/}}$ compris entre 0,8 et 0,9) pour des machines ayant jusqu'à 20 à 30 noeuds. Pour les algorithmes itératifs 3D, l'efficacité est comprise entre 0,8 et 0,9 pour une taille d'une dizaine de noeuds et entre 0,7 et 0,8 pour une taille allant jusqu'à 100 noeuds. Concernant la technique de parallélisation à adopter pour ces algorithmes, les stratégies DEP et DEI offrent de prime abord des performances équivalentes, mais compte tenu du temps de lecture des données du scanner, la parallélisation DEP est significativement plus intéressante que la parallélisation DEI. Les techniques de parallélisation plus originales visant à éliminer les étapes de synchronisation des données souffrent jusqu'à présent des effets secondaires de la décomposition simultanée de l'espace image et de l'espace de projection : redondance de traitement et complexité accrue des algorithmes.

Ainsi en pratique, lorsqu'une machine parallèle est utilisée c'est une machine à mémoire distribuée qui est choisie. Pour illustrer cette conclusion sur les machines MIMD, nous pouvons donner en exemple les travaux [Jones 06] qui a utilisé un cluster de 32 Xeons (double coeur) pour accélérer les reconstructions du scanner HRRT du centre

de recherche en TEP dynamique de Johns Hopkins à Baltimore (USA). Il utilise deux niveaux de parallélisation : une parallélisation par tâche avec la distribution de la reconstruction des 30 frames sur les 4 mini-clusters composés chacun de 8 processeurs et une parallélisation DEI pour la reconstruction d'une frame sur chaque mini-cluster. Ainsi il utilise la machine parallèle à son optimum. En effet, pour 8 noeuds la parallélisation DEI offre une efficacité $\boldsymbol{\eta_{/\!/}}$ égale à 1.0. Au final les temps des différentes étapes de traitement des données HRRT formées de **6367** sinogrammes de taille $\mathbf{256 * 288}$ sont de : 1.5 min/*frame* pour la transformation du mode liste en mode histogramme, 26s/*frame* pour la correction en atténuation, 28 min/*frame* pour les corrections en aléatoire et diffusé, et de 31 min/*frame* pour la reconstruction OSEM (accélérée d'un facteur 30).

3.3 Accélération sur processeurs vectoriels

Le pouvoir d'accélération des processeurs vectoriels de type Cray a fait l'objet d'une étude sur une machine CrayJ90 dans les années 90 par [Lecomte 98] pour l'algorithme FDK (rétroprojection à faisceaux coniques) et pour un algorithme algébrique en itératif 2D. Cette machine vectorielle est constituée de 4 blocs de 4 processeurs à 100 Mhz. La puissance de calcul de cette machine provient de ces processeurs hors normes. Ils possèdent une unité scalaire et une unité vectorielle pouvant traiter simultanément 64 vecteurs de 64 bits.

3.3.1 Rétroprojection

La rétroprojection n'étant pas un calcul matriciel pur et simple, il se prête mal à priori à l'implémentation sur processeur vectoriel. [Laurent 98] a ainsi observé la faible efficacité de la parallélisation classique de l'algorithme FDK sur un processeur vectoriel, la Maspar de DEC. Cette machine avec son architecture en tore 2D de 32*32 processeurs élémentaires exécutant tous la même instruction, n'a été que 2.5 fois plus rapide qu'un processeur SUN4. Selon l'auteur, cela est dû à sa faible capacité de mémoire et aux nombreuses phases d'inactivité lors des calculs séquentiels.

Toutefois grâce à une adaptation de l'algorithme de rétroprojection, il est possible de le paralléliser de manière satisfaisante. [Lecomte 98] a ainsi parallélisé l'algorithme incrémentiel de rétroprojection 3D TEP de [Cho 90] sur une machine CRAY J90. L'algorithme *ray-driven* de [Cho 90] traite de manière séquentielle les *viewgrams* en les déposant l'un après l'autre dans le volume. La parallélisation sur les 16 processeurs se fait sur les boucles des voxels $\boldsymbol{f(x,y,z)}$ selon x. La parallélisation se déroule bien à l'exécution avec une accélération d'un facteur 14 pour 16 processeurs ($\boldsymbol{\eta_{/\!/}}$=0.87). De plus, 25 % des 200 Mflops de chaque processeur sont utilisés. L'auteur ne donne pas de détails sur le déroulement des calculs vectoriels, mais on peut penser qu'ils se font sur la boucle des voxels $\boldsymbol{f(x,y,z)}$ selon z, celle-ci étant la boucle la plus interne. Une opération vectorielle consiste alors à déposer le vecteur de *bins* $\boldsymbol{B_{\phi,\Delta}(x,y) = \{bin(\rho_{x,y}, \lambda_{x,y,z}, \phi, \Delta)/z \in 0...N_z\}}$ dans le vecteur de voxels $\boldsymbol{V(x,y)}$ égal à $\boldsymbol{\{f(x,y,z)/z \in 0...N_z\}}$. On peut penser également que le calcul des coordonnées $\boldsymbol{\rho_{x,y}, \lambda_{x,y}}$ se fait de manière scalaire puis le calcul de $\boldsymbol{\lambda_{x,y,z}}$ de manière vectorielle.

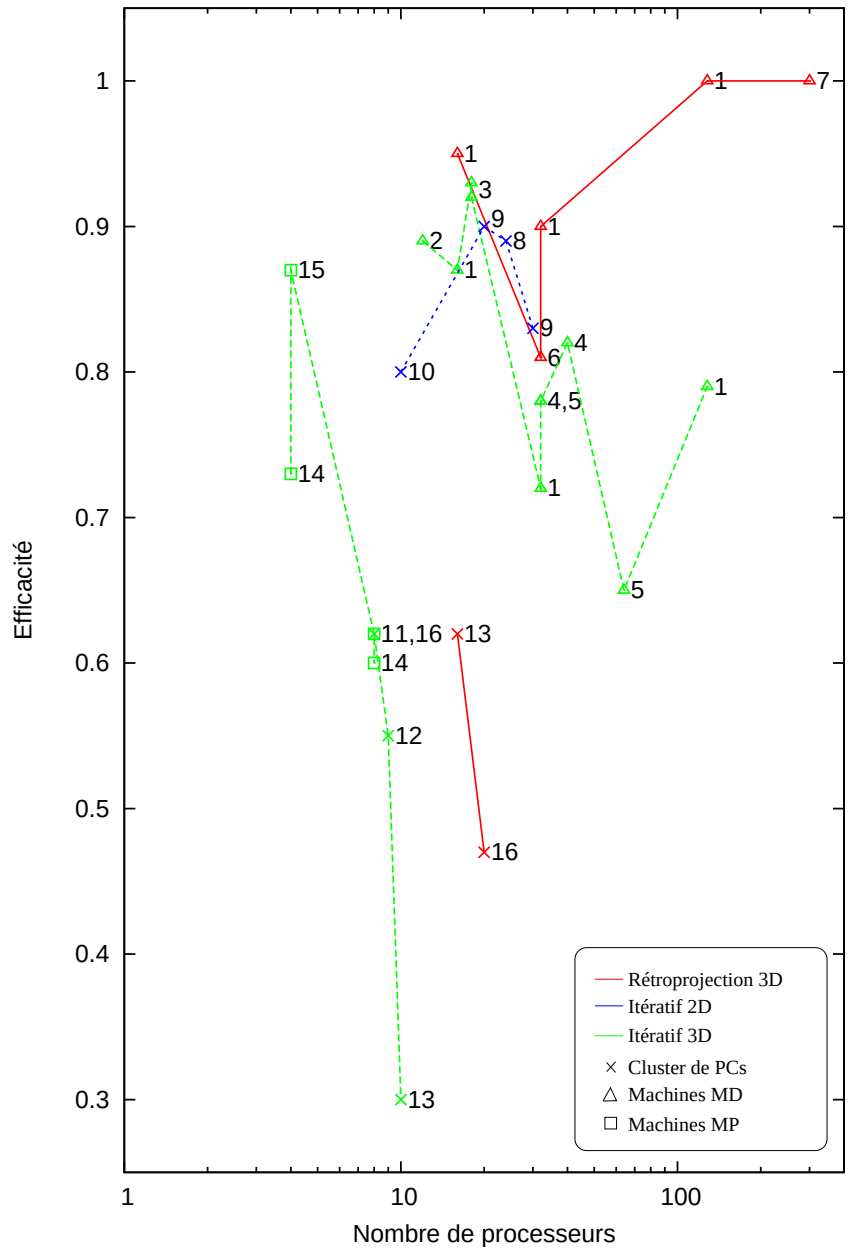


FIG. 2.17 – Efficacités de parallélisation $\eta_{//}$ pour les algorithmes de rétroprojection filtrée (en rouge), itératifs 2D (en bleu) et itératifs 3D (en verte) sur cluster de PCs (croix), sur machines à mémoire partagée (carrés) et sur machines à mémoire distribuée (triangles). Réferences : 1 [Laurent 98], 2 [Jones 99], 3 [Jones 03], 4 [Kole 05], 5 [Schellmann 07], 6 [Deng 06a], 7 [Ni 06], 8 [Vollmar 02], 9 [Munz 98], 10 [Vollmar 00], 11 [Li 04a], 12 [Shattuck 02], 13 [He 06], 14 [Smith 07], 15 [**?**], 16 [Brasse 05]

3.3.2 Algorithmes itératifs

Particulièrement adaptée aux calculs matriciels, [Lecomte 98] a également utilisé la machine vectorielle CRAY J90 pour accélérer un algorithme algébrique 2D de reconstruction en tomographie TEP. Les calculs matriciels de l'algorithme du gradient conjugué implique la matrice système $\boldsymbol{H}$ et le vecteur $\boldsymbol{f}$ représentant le volume 3D. [Lecomte 98] effectue d'abord le pré-calcul de la matrice 2D $\boldsymbol{H^t H}$, qui est la même pour chaque coupe transverse transverse. Ensuite à chaque itération, il effectue la multiplication de cette matrice $\boldsymbol{H^t H}$ avec le vecteur $\boldsymbol{f}$ qui représente **90%** des calculs. La parallélisation dans le plan transverse accélère de 14.6 le calcul matriciel de $\boldsymbol{H^t H f}$ pour 16 processeurs. Au final, le facteur d'accélération est de 13.2 pour 16 processeurs ($\boldsymbol{\eta_{//}}$=0.82) en ajoutant les parties séquentielles de l'algorithme. Le calcul vectoriel se fait selon z et chaque processeur exécute alors 137.8 MFlops sur les 200 MFlops théoriques.

3.3.3 Conclusion concernant les machines vectorielles

Les bons résultats obtenus sur les machines vectorielles par [Lecomte 98] montrent que les algorithmes de reconstruction offrent, outre un bon potentiel de parallélisation, la possibilité d'utiliser les calculs vectoriels pour accélérer leur exécution. Ainsi, **70%** des 200 MFlops des processeurs de la machine Cray utilisés par [Lecomte 98] sont atteints pour les algorithmes itératifs 2D et **25%** le sont pour la rétroprojection filtrée. Au final, l'accélération obtenue avec ces machines n'est pas à la hauteur de leur prix trop important pour être utilisées dans un service clinique. Toutefois, des solutions technologiques telles que le Cell ou les GPUs dont la puissance de calcul étaient considérées il y a quelques années encore comme bien inférieure à celle des bêtes de calcul que constituent les machines Cray, ont bien grandi depuis. Ils permettent aujourd'hui à des coûts bien inférieurs l'utilisation efficace d'opérateurs vectoriels de taille plus réduites (vecteur de 4 ou 8 flottants).

3.4 Accélération sur Cell

La rétroprojection (algorithme FDK en cone-beam 3D) a été implémentée sur le processeur Cell par trois équipes de recherche [Bockenbach 07],[Sakamoto 07] et [Scherl 07b]. Tous les trois ont choisi la même stratégie de parallélisation avec une décomposition DEI (recommandée pour la rétroprojection comme nous l'avons vu précédemment) et la même stratégie mémoire avec un préchargement logiciel (imposée par l'organisation mémoire du Cell).

Lors de la parallélisation DEI de la rétroprojection sur le Cell, chaque SPE reconstruit un bloc de voxels à partir du sous-espace de projection correspondant. Les projections de chaque sous-volume sur le plan de détecteurs sont chargées les unes après les autres dans la mémoire locale (256 Ko). Afin de masquer le temps de chargement de ces projections de la mémoire principale vers les mémoires locales, une technique de *double buffering* est utilisée. Le PPE se charge du transfert de la projection $\boldsymbol{\phi_{t+1}}$ du sous-volume via le DMA et la mémoire locale stocke ainsi le sous-volume et les deux projections $\boldsymbol{\phi_t}$ et $\boldsymbol{\phi_{t+1}}$

en plus du code, du tas et de la pile. Ainsi le découpage en blocs du volume doit se faire en fonction de la taille de la mémoire locale de chaque SPE (256 Ko). [Bockenbach 07] découpe le volume en cubes de $\mathbf{32^3}$ voxels, [Scherl 07b] utilise, lui, des blocs $\mathbf{32^2 * 8}$. En effet, il observe expérimentalement que ces blocs avec une hauteur moins importante, offrent une meilleure localité spatiale et temporelle.

Outre ce pré-chargement logiciel du à l'absence de mémoire cache des SPEs, les implémentations sur Cell tirent profit de l'architecture adaptée au calcul intensif des *Synergistic execution Unit* (SXU) avec ses opérateurs vectoriels et ses deux pipelines. [Scherl 07b] mesure l'exécution de 1.75 instruction par cycle par *SPE* pour la rétroprojection à faisceaux coniques sans interpolation. Cela reste proche des deux instructions par cycle théorique. Toutefois, pour la plupart des opérations flottantes, la proportion de *load* et *store* est supérieure à celle des multiplications-additions (*mad*). Par exemple, [Bockenbach 07] en rétroprojection à faisceaux coniques, compte pour chaque SPE **6** cycles d'exécution pour la seule opération d'accumulation de la mise à jour du vecteur de voxels (4 flottants) avec 5 *loads* et 1 *store* sur le pipeline impair et 1 *mad* sur le pipeline pair. Ainsi, l'efficacité de calcul est de **1.5** C/Op (Cycles par opération) au lieu du $\mathbf{1/4 = 0.25}$ C/Op théorique : seulement $\mathbf{1/6 = 16.7\%}$ des performances théoriques du *cell* sont utilisées. Au final, [Bockenbach 07] mesure 4.72 cycles par SPE pour la mise à jour d'un voxel (les 3.22 cycles supplémentaires correspondant au calcul des coordonnées de projection).

Après alignement des reconstructions en rétroprojection à faisceaux coniques de [Bockenbach 07] et [Scherl 07b] à un jeu de données de référence (volumes de $\mathbf{512^3}$ et 512 projections de $\mathbf{512^2}$), [Bockenbach 07] et [Scherl 07b] obtiennent des temps semblables de l'ordre de 25 s. [Bockenbach 07] utilise les techniques d'optimisation *software* de [Knaup 07] (voir 1.2.1) en effectuant un pré-calcul sur les données de projection pour éliminer le calcul de l'interpolation bilinéaire et rendre parallèles les plans de projection au volume. Il divise ainsi par deux le temps de reconstruction. Par rapport à la version CPU optimisée, le Cell accélère ainsi d'un facteur 14 la rétroprojection FDK. Contrairement à [Bockenbach 07], [Scherl 07b] effectue le filtrage en plus de la rétroprojection : un SPE pour le filtrage et les sept autres pour la rétroprojection.

[Knaup 06] faisant partie comme [Bockenbach 07] de l'équipe de Kachelriess à l'IMP, il a implémenté l'algorithme itératif OSC en tomographie CT sur les 2 Cells d'une carte *dual Cell Blade*. Un Cell est utilisé pour l'initialisation effectuant une reconstruction FDK et l'autre Cell est utilisé pour l'exécution des itérations OSC. Une parallélisation DEI semble adoptée : chaque SPE effectue la reconstruction d'un sous-volume. Une réduction-sommation des projections des sous-volumes semble être effectuée sur le PPE qui centralise l'accès au volume et aux données de projection. Après 4 itérations constituées chacune de 16 subsets, le volume est reconstruit en 3.2 mn.

3.5 Accélération sur GPU

Pour les GPUs actuels, nous pouvons faire la distinction entre la vision purement graphique adaptant le pipeline graphique pour la projection et la rétroprojection, et la vision multiprocesseur parallélisant les calculs sur les *streamming processors* utilisant le

GPU comme un GPGPU. Si la dernière est plus classique, la première l'est moins. Son principe est pourtant simple : la projection et la rétroprojection peuvent être décomposées en une succession de plaquages de textures 2D associées a des polygones sur des images 2D pixelisées ([Cabral 94]) .

Nous présenterons tout d'abord les implémentation de projecteurs et de rétroprojecteurs sur le pipeline graphique. Nous nous intéresserons ensuite aux implémentations GPGPU de la rétroprojection et constaterons que ces dernières sont sur le point d'être utilisées en routine clinique. Nous aborderons enfin le problème posé par la qualité de reconstruction sur les GPUs.

3.5.1 Projecteurs sur pipeline graphique

L'implémentation classique de la projection ([Mueller 00]) sur GPU, projette le volume tranche par tranche. Une rotation d'un angle ϕ des polygones correspondant aux coupes transverses est d'abord effectuée, puis la texture de la tranche est plaquée sur le plan des détecteurs en respectant une projection perspective pour les tomographies à faisceaux coniques, et une projection orthogonale pour les tomographies à faisceaux parallèles. L'intégrale des LORs à travers le volume se fait alors en accumulant les projections de chaque coupe transverse dans le *framebuffer*. Le pas d'intégration n'est pas constant et dépend de ϕ, il faut donc appliquer un facteur correction à l'accumulation des projections. Cette technique demande de stocker en mémoire de texture deux volumes découpés en coupes transverses : un selon (x, z) et un autre selon (y, z). Selon l'angle ϕ de projection, un des deux volumes est utilisé pour la projection. Une méthode plus coûteuse mais plus exacte, consiste à redéfinir les coupes transverses pour chaque angle de projection, comme le fait [Tita 07]. Tita calcule ainsi par interpolation tri-linéaire du volume les coupes transverses qui font face au plan de détecteurs avant d'accumuler toutes ces coupes transverses pour obtenir la projection du volume sur le plan de détecteurs.

Contraint par l'ordonnancement chronologique des LORs en mode-liste, [**?**] pour une reconstruction OSEM 3D à partir de données PET en mode liste, n'effectue pas la projection en groupant les LORs appartenant à un même plan de projection mais il effectue de manière indépendante la projection de chaque LOR. Le pipeline graphique ne traite alors plus un flot de coupes transverses à projeter sur un plan de détecteur mais un flot de LORs à intégrer selon le volume. La projection se fait en deux étapes : (1) échantillonnage de chaque LOR à travers le volume par un plaquage de la texture 3D du volume sur les LORs, (2) sommation des échantillons stockés dans des textures temporaires pour obtenir la valeur d'un *bin*. Un noyau gaussien est appliqué lors de l'échantillonnage pour modéliser les LORs par des Tubes de Réponse (TOR) gaussiens.

3.5.2 Rétroprojecteurs sur pipeline graphique

La rétroprojection d'un plan de détecteurs dans le volume est effectuée classiquement sur GPU tranche par tranche, le GPU ne pouvant effectuer que des *rasterizations* 2D. La figure 2.18 représente comment la rétroprojection 3D à faisceaux coniques sur

GPU est implémentée par Mueller ([Xu 07]). Les fragments du *framebuffer* sont liés aux pixels de la coupe 2D par une vue orthographique. Les coordonnées de projection des sommets de la coupe transverse sont calculées à partir d'une matrice de projection perspective chargée dans le *vertex shader*. Les coordonnées de projection des sommets du polygone (c'est-à-dire la tranche 2D) sont alors des attributs du polygone. Ils sont utilisés ensuite pour calculer par interpolation bilinéaire les coordonnées de projection de tous les fragments du polygone (c'est-à-dire chaque pixel de la tranche 2D) lors de l'étape de *rasterization*. Cette méthode est tout à fait valide puisque les coordonnées de projection et les coordonnées du volume sont linéairement liées. La division par le poids $\boldsymbol{w_\phi}$ correspondant à la vue perspective, permet ensuite d'obtenir les indices de textures autrement dit les coordonnées dans le plan de projection. Le *fragment shader* calcule ensuite la valeur de projection $\boldsymbol{bin_\phi}$ de chaque fragment à partir des indices de texture avec l'interpolateur bilinéaire câblé de texture. Enfin, la coupe transverse contenue dans le *framebuffer* est accumulée avec la coupe transverse stockée en mémoire de texture.

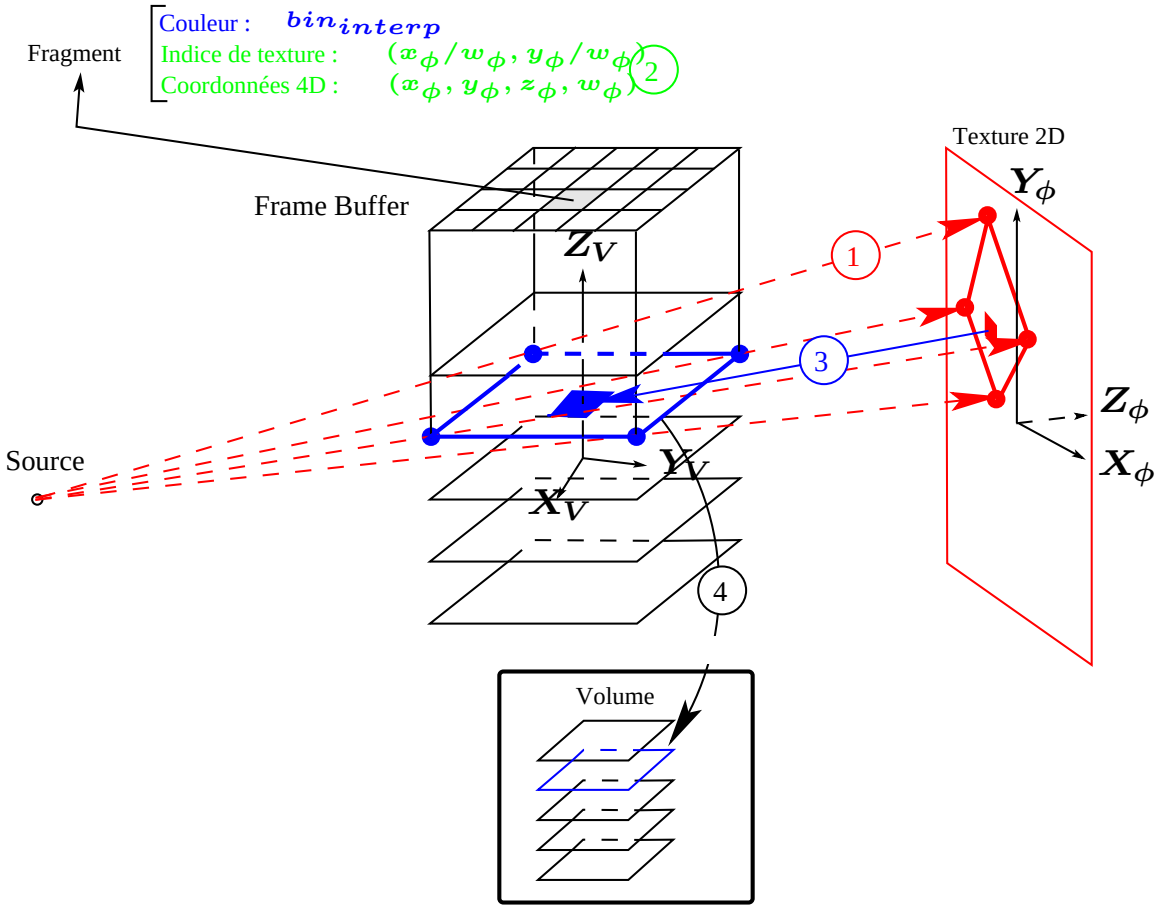
Cette dernière méthode utilise lors du plaquage de textures, le polygone du plan de détecteurs correspondant exactement à la coupe transverse (ce polygone est obtenu par la projection des sommets de la coupe transverse sur le plan des détecteurs). [Chidlow 03] en OSEM 2D, lui, définit comme polygone la rangée de détecteurs qu'il projette ensuite sur la coupe transverse avant d'effectuer la *rasterisation*. [Xu 04] étend cette méthode en CT cone-beam 3D en définissant comme polygone les quatre ou cinq rangées de détecteurs situés dans le champs de vue de chaque coupe transverse. Cette méthode permet de réduire les transferts de textures entre le GPU et le CPU.

[**?**] définit comme polygones, les LORs à projeter sur les coupes transverses $\boldsymbol{(x, y)}$ du volume, pour une reconstruction OSEM 3D à partir de données TEP en mode liste. Comme pour la projection, il effectue un traitement par LORs afin de respecter le stockage des données de projection en mode liste. Il projette tout d'abord par groupe, les LORs (10 000) à l'aide d'une matrice de projection orthographique. Ensuite le polygone projeté sur la coupe transverse est pixelisé, puis chaque pixel de cette LOR est pris en compte ou non pour la rétroprojection de la coupe transverse selon la profondeur (coordonnée axiale $\boldsymbol{z}$) de la coupe transverse traitée. Il traite quatre coupes transverses en parallèle en utilisant les quatre canaux RGBA du pipeline graphique.

3.5.3 Rétroprojecteurs sur GPGPU

L'introduction de la programmabilité des *vertex shaders* et des *fragment shaders*, a ouvert le champs à une implémentation plus classique de la rétroprojection découplée des contraintes posées par le pipeline graphique comme l'a fait [Despres 07] avec l'implémentation de FFTs rapides. [Schiwietz 07] a ainsi pu effectuer une rétroprojection filtrée entièrement sur GPU : pré-traitement sur les données de projection (lissage des courbures et filtrage rampe), rétroprojection (même méthode que [Xu 07]) et post-traitement sur le volume (suppression de différents types d'artefacts). Au final, il obtient une reconstruction 3.5 fois plus rapide qu'un Xeon dual core optimisé avec SSE2/3.

Afin de bénéficier au maximum de la localité spatiale et temporelle, deux stratégies d'accès mémoire existent : soit les accès aux données de projection se font via des textures



1 Projection des sommets de la coupe 2D (*vertex shader*)

2 Calcul des indices de texture de chaque voxel (*rasterizer*)

3 Plaquage de la texture 2D pour chaque voxel (*fragment shader*)

4 Accumulation

FIG. 2.18 – Rétroprojection sur GPU selon la méthode de [Mueller 07]

2D et ainsi à l'exécution la reconstruction est accélérée grâce au cache 2D de texture, soit ils sont définis explicitement dans le code avec un préchargement dans la mémoire partagée (comme [Kachelriess 07] le fait avec le *Cell*) et les transferts mémoire sont masqués avec les calculs durant l'exécution. Des temps de reconstruction de [Yang 07], nous pouvons conclure que la première stratégie est préférable. Afin de ne pas prendre en compte l'accélération de l'interpolateur matériel par rapport à l'interpolation *software*, nous comparons dans l'étude de [Yang 07], la rétroprojection suivant la première stratégie

et effectuant une interpolation linéaire effectuée avec l'interpolateur matériel (8.8 ms), avec celle suivant la deuxième stratégie et effectuant une interpolation au plus proche voisin (19.2 ms). Cette différence de performance (facteur 2) entre les deux stratégies mémoire indique qu'il est préférable de laisser gérer les transferts mémoire par le cache 2D (si l'accès est seulement en lecture, celui-ci étant limité aux accès en lecture) plutôt que par le développeur. En effet, le préchargement mémoire demande une maîtrise de la synchronisation des *threads*. Et celle-ci par exemple n'est pas explicitée de manière optimale dans le code de [Yang 07].

La facilité de programmation de CUDA a toutefois un désavantage : il n'est pas toujours possible d'utiliser toutes les ressources matérielles du pipeline graphique. Ainsi, [Xu 07] observe qu'avec une implémentation multiprocesseurs, un facteur 3 d'accélération est perdu par rapport à une implémentation purement graphique. En effet, le calcul des coordonnées de projection avec son implémentation CUDA est effectué explicitement pour chaque pixel et ne se fait plus avec l'interpolateur bilinéaire du *rasterizer* à partir des coordonnées de projection des quatre sommets de la coupe transverse, comme cela est fait avec le pipeline graphique.

3.5.4 GPGPU : vers une utilisation en routine clinique

Des travaux récents ([Tita 07],[Riabkov 07],[Vaz 07]) utilisent les GPUs comme accélérateur matériel pour la reconstruction en tomofluoroscopie, technique d'imagerie médicale CT temps réel sur scanner *C-arm*. Ainsi, [Tita 07] a implémenté l'algorithme itératif SART sur le pipeline graphique du GPU Nvidia GeForce 7950. Il effectue une reconstruction d'un volume $\mathbf{256^3}$ à partir de 90 projections en un temps de l'ordre de la minute pour trois itérations. Moins de trois secondes après qu'une image de projection soit acquise, une mise à jour du volume est effectuée prenant en compte cette nouvelle image. L'accès aux images de projection suit la règle suivante : une nouvelle image acquise par le scanner est traitée en priorité puis mise dans une FIFO de traitement. [Vaz 07] (société *Barco*) obtient aussi de très bonnes performances pour une reconstruction FDK sur C-ARM. Il reconstruit en moins de 20 secondes un volume $\mathbf{512^3}$ à partir de 1000 projections. L'auteur insiste sur la possibilité offerte par les GPUs d'autoriser une visualisation du volume mis à jour sans risque majeur de ralentir la reconstruction. C'est en effet une des forces des GPUs dont l'architecture a été conçue pour afficher en temps réel le contenu du *framebuffer* notamment pour les jeux vidéos.

La mémoire RAM des GPU (768 Mo sur la Nvidia 8800 GTX) n'a pas une taille suffisante pour contenir les données manipulées classiquement en tomographie CT (volume $\mathbf{512^3}$ et 360 projections $\mathbf{1024^2}$). Il faut donc effectuer des transferts mémoire entre la mémoire GPU et la mémoire CPU. [Riabkov 07] (*GE Healthcare*) pour une reconstruction FDK, présente ainsi les deux stratégies de stockage mémoire : soit on stocke dans la mémoire de texture tout le volume et on transfert dynamiquement les images de projections, soit on stocke toutes les projections sur la mémoire GPU et on transfert dynamiquement les coupes transverses du volume. La première stratégie effectue la rétroprojection des plans de projection un à un dans tout le volume, alors que la deuxième effectue la rétroprojection des coupes transverses du volumes, l'un après l'autre en ayant

à disposition toutes les données de projection. La première a le désavantage de devoir attendre toutes les données de projection avant de démarrer la reconstruction. De plus, le volume étant de taille plus petite que les données de projection (le nombre d'inconnues est inférieur au nombre d'équations). La deuxième stratégie est donc préférable.

3.5.5 Les GPUs et la qualité de reconstruction

Les premières études visant à utiliser les GPUs pour la reconstruction tomographique [Cabral 94, Mueller 00, Chidlow 03] se heurtaient à une précision de calcul insuffisante (virgule fixe sur 8 ou 12 bits) du pipeline graphique pour atteindre une accélération satisfaisante. Une extension de la précision était alors nécessaire en répartissant les 16 bits d'une donnée sur les 4 canaux RGBA du pipeline graphique. Et il fallait également effectuer les opérations d'accumulation sur le CPU et non sur le GPU afin d'éviter les débordements. Dès l'introduction dans le pipeline graphique d'une précision accrue (virgule flottante sur 16 ou 24 bits), [Kole 04] et [Xu 05] obtenaient une perte de précision minimale voire négligeable en comparaison de la précision en virgule flottante sur 32 bits. Pouvant ainsi effectuer la reconstruction entièrement sur le GPU sans être pénalisés par de nombreux transferts mémoire CPU/GPU, ils obtiennent un facteur d'accélération d'un ou deux ordres de magnitude en tomographie CT avec des algorithmes itératifs.

C'est avec les nouvelles générations de GPU que l'utilisation des flottants simple précision a été généralisée même pour le *rasterizer* matériel. Auparavant, il fallait effectuer une reconstruction avec des précisions hybrides afin de bénéficier pleinement des ressources matérielles des GPUs. Ainsi [Mueller 06] utilisait le *rasterizer* matériel (entier 8 bits) pour la rétroprojection et l'additionneur flottant pour l'accumulation. Avec deux passes de *rasterization* pour traiter des données 16 bits, il obtenait une accélération supplémentaire d'un facteur 14. La génération actuelle des GPUs offre une précision simple flottante (32 bits) et bientôt une précision double flottante (64 bits). Même si la précision flottante des GPUs ne respecte pas les standards IEEE, elle semble suffisante pour la reconstruction tomographique. Toutefois même si tous les opérateurs du GPU manipulent des flottants, ils n'effectuent pas toutes leurs opérations avec une précision flottante. Par exemple, l'interpolateur bilinéaire de texture utilise des poids codés en virgule fixe sur 9 bits ([Yang 07]). Ainsi si l'on souhaite bénéficier du *rasterizer* matériel, il faudra veiller à ce que cela ne dégrade pas trop la qualité de reconstruction.

3.6 Accélération sur ASICs et FPGAs

Les systèmes de reconstruction développés par la plupart des sociétés en imagerie médicale (GE Healthcare, Siemens AG Medical solutions, Terarecon, Mercury) utilisent du matériel dédié (FPGA/ASIC) pour l'accélération des noyaux durs des algorithmes de reconstruction : le filtrage et la rétroprojection. A la différence des étapes de pré et post traitements, elles constituent des algorithmes stables, non affectées par l'évolution des scanners. Malgré leur coût élevé en temps de développement, ces architectures sur mesure justifient leur emploi comme coprocesseur permettant une accélération bien plus importante que celle offerte par les solutions "grand marché" (*off-the-shelf*).

3.6.1 Architectures ASICs

Les relatives faibles perspectives qu'offre le marché de la reconstruction tomographique en imagerie médicale, justifient difficilement l'utilisation de matériels entièrement dédiés (*full custom*) comme les ASICs. Ainsi le processeur XTrillion, de la société [Terarecon], est dédié à un champs plus large que la seule rétroprojection en CT comme par exemple la tomographie industrielle ou le rendu volumique. Le XTrillion 3.0 (2006) grâce à ses 8 processeurs vectoriels possédant chacun 6 ALUs, atteint un pic de performance de 16 GFLOPS pour une fréquence de fonctionnement de 333 MHz. Par rapport aux processeurs classiques, il permet une accélération d'un facteur 2 à 3 tout en offrant une réduction en consommation d'un facteur 6. Les systèmes de reconstruction proposés par Terarecon, possèdent jusqu'à 32 processeurs XTrillions. Nous pouvons toutefois remarquer que l'utilisation d'ASICs dédiés a été essayée par GE Healthcare ([DeMan 07]) avec sa carte *Image Generator* qui était composée de 8 unités de calcul constituées chacune d'un DSP dédié au pré-traitement et d'un ASIC de rétroprojection. Cependant, cette carte date des débuts des années 90, or depuis une dizaine d'années la technologie ASIC a laissé la place aux FPGAs dans le domaine de la reconstruction tomographique.

3.6.2 Architectures FPGAs

Les cartes FPGAs s'intègrent bien dans les systèmes de reconstruction classique via des connections PCI avec les PCs hôtes. Si le *Global Recon Engine* (GRE) de GE Healthcare ([DeMan 07]) datant de 2003 et le système de reconstruction de Mercury ([Goddard 02]) datant de 2002 utilisent des cartes mono-FPGA, la carte de l'*ImagePro* de Siemens ([Heigl 07]) est constituée de 9 FPGAs. Un FPGA central est dédié au contrôle et à l'étape de filtrage, les huit autres forment deux anneaux indépendants de 4 FPGAs chacun dédiés à la rétroprojection.

Les efficacités de reconstruction atteintes par les architectures FPGAs de [Li 04b], [Leeser 05] et [Heigl 07], sont proches du maximum soit une opération par cycle pour chaque pipeline (voir tableau 2.4). Ces performances équivalentes ont été obtenues à l'aide de stratégies d'accès mémoire et de parallélisation différentes.

	[Leeser 05]	[Li 04b]	[Heigl 07]
ARCHITECTURE			
Génération de FPGA	Virtex 2000E (1999)	Virtex 2 Pro (2002)	Virtex 4 (2004)
Nombre de FPGA	1	1 VP125	8 SX35 +1 SX55
Nombre de pipelines par FPGA	16	8	16
Nombre d'étages du pipeline	7	64	?
Fréquence de fonctionnement	65 MHz	150 MHz	200 MHz
MEMOIRE			
Quantité de mémoire	$\boldsymbol{mem_V = 2 \cdot 36}$ Mo $\boldsymbol{mem_P = 2 \cdot 36}$ Mo	$\boldsymbol{mem_V = 1 \cdot}$? Mo $\boldsymbol{mem_P = 1 \cdot}$? Mo	1 Go/FPGA
Débit mémoire	$\boldsymbol{deb_V = 2 \cdot}$ 260 Mo/s $\boldsymbol{deb_P = 2 \cdot}$ 260 Mo/s	$\boldsymbol{deb_V = 1}$ Go/s $\boldsymbol{deb_P = 1}$ Go/s	?
PARAMETRES DE RECONSTRUCTION			
Rétroprojection	Parallel-beam 2D	Cone-beam 3D (FDK)	Cone-beam 3D (FDK)
Parallélisme	Projection sous-volume=$\mathbf{512^2}$	Volume sous-volume=$\mathbf{16^3}$	Volume sous-volume=$\mathbf{16^3}$
Volume	$\mathbf{512^3}$	$\mathbf{512^3}$	$\mathbf{440 \cdot 512^2}$
Projection	$\mathbf{1024^2}$	$\mathbf{300 \cdot 512^2}$	$\mathbf{543 \cdot 1240 \cdot 960}$
Nombre d'opérations	$\mathbf{131 \cdot 10^9}$	$\mathbf{40.3 \cdot 10^9}$	$\mathbf{62.6 \cdot 10^9}$
TEMPS DE RECONSTRUCTION			
Temps de reconstruction	128 s	33.5 s	3.5 s
Cycles/Opération	0.06	0.12	0.01
Cycles/Opération/pipeline	0.97	1.0	1.43

TAB. 2.4 – Rétroprojection sur FPGA

Stratégies d'accès mémoire

Si les accès à la mémoire externe ne bloquent pas le pipeline pour les architectures de [Li 04b], [Leeser 05] et [Heigl 07], c'est grâce à l'emploi de mémoire *ping pong* pour masquer le temps de chargement des données de projection, comme [Bockenbach 07] le fait sur le Cell. De plus, l'organisation des systèmes de mémoires (externes et internes) est faite sur mesure avec l'utilisation de mémoires externes indépendantes pour l'accès au sinogramme et au volume afin de libérer au maximum l'accès aux mémoires externes, mais également avec l'emploi d'autant de bancs mémoires internes que de données accédées lors de l'opération d'interpolation. Sur ce dernier point, des différences existent entre les architectures de [Li 04b] et de [Leeser 05] : si [Li 04b] pour une interpolation tri-linéaire préfère dupliquer les données dans les 8 bancs mémoire plutôt que d'utiliser un système complexe d'adressage, [Leeser 05] peut se contenter pour une interpolation simplement linéaire, d'aiguiller les données dans deux bancs mémoire selon la parité des adresses mémoire.

Stratégies de parallélisation

Les architectures de [Li 04b] et [Leeser 05] diffèrent également sur la stratégie de parallélisation. Alors que [Li 04b] et [Heigl 07] effectuent en parallèle la reconstruction de N sous-volumes de taille $\mathbf{16^3}$ voxels ce qui correspond à un parallélisme DEI (voir figure 2.19), [Leeser 05] effectue la reconstruction d'une coupe transverse en rétroprojetant en parallèle N projections d'un pixel ce qui correspond à un parallélisme DEP (voir figure 2.20). Les derniers étages des N pipelines de [Leeser 05] effectuent l'accumulation de ces N données de projection.

Ces deux stratégies de parallélisation impliquent des stratégies de pré-chargements de données différentes. D'un coté, [Li 04b] effectue le pré-chargement des N projections des sous volumes $\boldsymbol{V_i}$ reconstruits en parallèle : $\boldsymbol{P_{\phi_1}(V_1)}$, $\boldsymbol{P_{\phi_1}(V_2)}$ ... $\boldsymbol{P_{\phi_1}(V_N)}$. De l'autre, [Leeser 05] pré-charge les N projections $\boldsymbol{P_{\phi_i}}$ de la coupe transverse : $\boldsymbol{P_{\phi_1}(coupe)}$, $\boldsymbol{P_{\phi_2}(coupe)}$, ... $\boldsymbol{P_{\phi_N}(coupe)}$. La stratégie de [Leeser 05] est automatique, il charge toutes les projections correspondantes à une coupe transverse alors que celle de [Li 04b] est plus ciblée, il ne charge que les données de projection nécessaires en utilisant le processeur PPC embarqué sur le Virtex pour calculer les sommets des projections des sous-volumes. Si le taux de réutilisation des données est équivalent pour les deux stratégies, [Li 04b] peut adapter la taille des données à pré-charger en modifiant la taille du sous-volume. La taille des données à pré-charger pour [Leeser 05] est contrainte par la taille des coupes transverses $\mathbf{512^2}$.

Précision de calcul

[Leeser 05] a effectué une étude approfondie de la qualité de reconstruction obtenue en implémentant la rétroprojection avec une précision de calcul en virgule fixe, caractéristique des architectures FPGAs. En codant sur 9 bits les données de projection, il obtient ainsi seulement 0.015 % d'erreur relative par rapport à une reconstruction en virgule flottante.

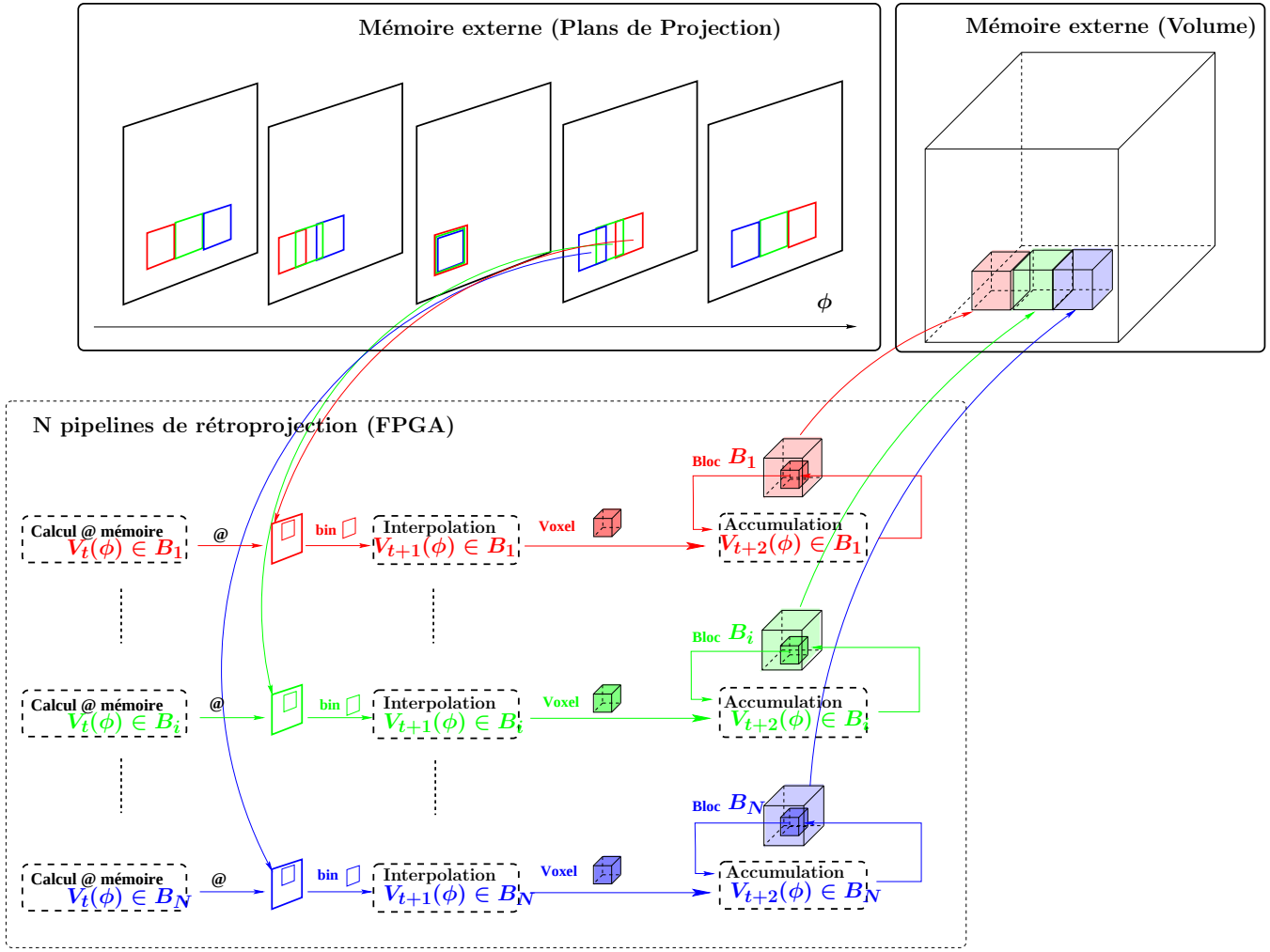


FIG. 2.19 – Parallélisation DEI de la rétroprojection filtrée 3D pour une architecture matérielle : les blocs de voxels $\boldsymbol{B_1}$, $\boldsymbol{B_2}$ ... $\boldsymbol{B_N}$ sont traités en parallèle. Un voxel $\boldsymbol{V}$ de chaque bloc est mis à jour avec l'accumulation de la projection correspondant à un angle $\boldsymbol{\phi}$ à chaque coup d'horloge (ce voxel est noté $\boldsymbol{V(\phi)}$).

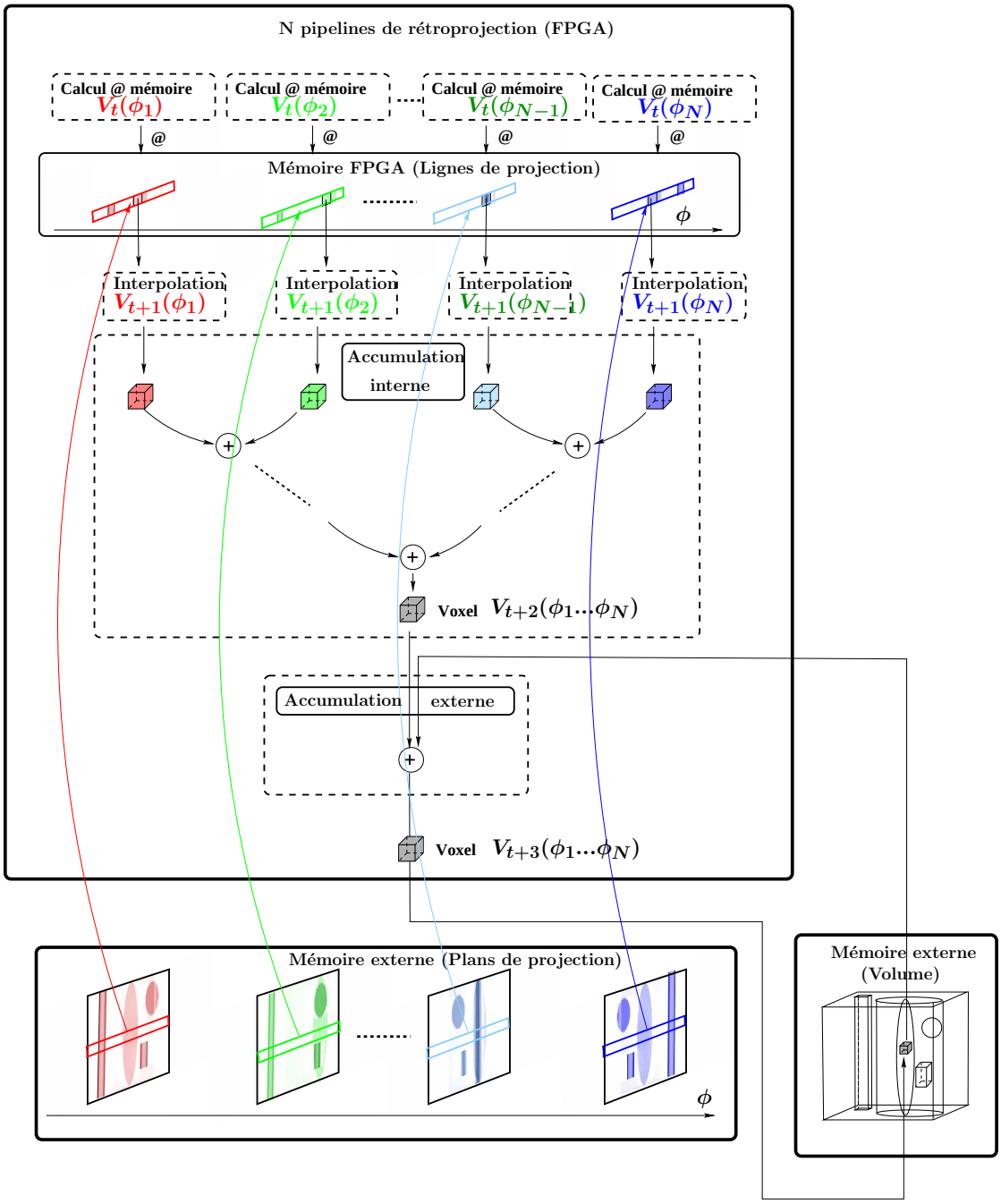


FIG. 2.20 – Parallélisation DEP de la rétroprojection filtrée 3D pour une architecture matérielle : les plans de projections $\boldsymbol{P_{\phi_1}}$, $\boldsymbol{P_{\phi_2}}$ .. $\boldsymbol{P_{\phi_N}}$ sont traités en parallèle. Après accumulation interne des voxels partiellement reconstruits ($\boldsymbol{V(\phi_1)}$, $\boldsymbol{V(\phi_2)}$ ... $\boldsymbol{V(\phi_N)}$), un voxel reconstruit est obtenu à chaque cycle d'horloge.

4 Conclusion

Nous avons démontré dans ce chapitre que le seul recours aux techniques d'accélération logicielle et algorithmique sur un PC standard pour accélérer les reconstructions tomographiques TEP et CT est insuffisant. Nous avons ensuite constaté que la parallélisation sur machines multiprocesseurs à mémoire distribuée est efficace, et permet ainsi une accélération significative de la reconstruction 2D et 3D. Enfin, nous nous avons décrit les accélérations obtenues avec trois architectures matérielles plus spécifiques : le processeur Cell, les processeurs graphiques et les architectures matérielles de type FPGA/ASIC.

Premier constat, les architectures spécifiques (Cell, GPU et FGPA) constituent à elles seules, des unités de calcul plus performantes qu'une dizaine ou une vingtaine de noeuds de calcul d'une machine multiprocesseur. Cela reste toutefois insuffisant pour satisfaire les contraintes temps réel comme illustré sur la figure 2.21 en tomographie CT. Ainsi la constitution d'une machine parallèle constituée de plusieurs unités de calcul performantes (Cell, GPU ou FPGA) permettrait de réduire l'écart avec la contrainte temps réel. Une stratégie de parallélisation semblable à celle utilisée pour les machines multiprocesseurs semble toute indiquée. Par ailleurs, si trois architectures (Cell, GPU et FPGA) ont bel et bien émergé ces dernières années, nous pouvons d'ores et déjà constater une hiérarchie des performances. Ainsi pour la rétroprojection, un GPU est plus rapide que le Cell (3 fois plus d'après [Scherl 07a]) et qu'une architecture FPGA comme celle de [Heigl 07] (8 fois plus rapide comme cela apparaît sur la figure 2.21).

Deuxième constat, deux stratégies d'accès mémoire ont été utilisées dans toutes les implémentations présentées jusqu'à présent. Elles s'appuient toutes sur la nécessité de charger les données utiles à l'unité de calcul dans une mémoire proche de l'unité de calcul. Si l'architecture possède un cache mémoire, alors le développeur laisse le cache s'occuper des accès à la mémoire externe. Sinon le développeur doit mettre en place une stratégie de pré-chargement "ad hoc" afin de rendre transparent le temps d'accès à la mémoire externe. Idéalement avec cette stratégie, les données de projection nécessaires à l'instant t+1 sont chargées à partir de la mémoire externe, pendant que les données de projection nécessaires à l'instant t sont en train d'être rétroprojetées. Dans ce contexte, d'une part les processeurs classiques et graphiques sont basés sur une utilisation intensive de leur cache mémoire ([Yang 07] a observé que cette option est plus pertinente pour le GPU qu'un code effectuant une pré-chargement mémoire), d'autre part les stratégies d'accès mémoire du processeur Cell et des architectures FPGA sont à la charge du développeur. Si pour le processeur Cell, le PPE et les DMAs sont prévus pour mettre en place un pré-chargement de manière logicielle vers les SPEs, pour les FPGAs en revanche tout reste à inventer.

Dans le chapitre suivant, nous présenterons une architecture matérielle pour la rétroprojection TEP 3D destinée à être implémentée sur un SoPC. Elle utilisera le potentiel de parallélisation de la rétroprojection et mettra en place une stratégie de pré-chargement mémoire qui s'adapte dynamiquement aux requêtes du module de rétroprojection.

.

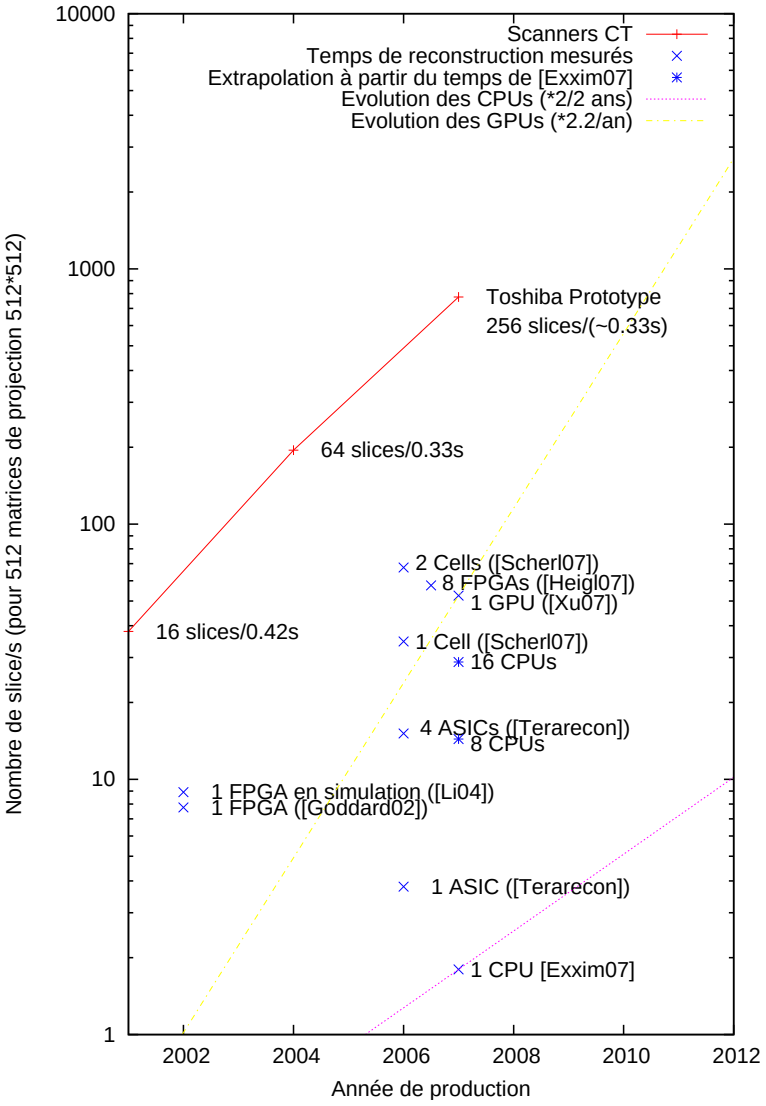


FIG. 2.21 – Accélération de la rétroprojection en faisceaux coniques en tomographie CT (algorithme FDK). Cette figure a été complétée à partir de la figure 2.8 présentant l'évolution comparée des scanners CT avec celle des CPUs.

Deuxième partie

ArchiTEP 3P : une architecture matérielle pour la rétroprojection 3D en imagerie TEP

Notations pour la partie II

Efficacité en nombre de cycles par opération :	
Efficacité de la Rétro-Projection	$\boldsymbol{t_{RP/op}}$
Coût en nombre de cycles	
Coût de calcul d'une opération	$\boldsymbol{C_{calcul}(op)}$
Coût d'un accès à la mémoire externe pour une opération	$\boldsymbol{C_{mem}(op)}$
Coût moyen d'un accès à la mémoire externe pour une opération	$\boldsymbol{\hat{C}_{mem}(op)}$
Coût d'un accès à la mémoire externe pour un *bin*	$\boldsymbol{C_{mem}(bin)}$
Coût moyen d'accès à une ligne mémoire en SDRAM	$\boldsymbol{\hat{C}_{SDRAM}(ligne)}$
Coût moyen d'un accès à la mémoire externe pour un *bin*	$\boldsymbol{\hat{C}_{mem}(bin)}$
Coût moyen d'un défaut de cache pour un *bin*	$\boldsymbol{\hat{C}_{defaut}(bin)}$
Coût moyen d'une mise en cache d'un *bin*	$\boldsymbol{\hat{C}_{cache}(bin)}$
Taux caractérisant le cache mémoire	
Taux d'utilisation d'un cache mémoire	$\boldsymbol{\eta_{cache}}$
Taux de défaut d'un cache mémoire	$\boldsymbol{\eta_{defaut}}$
Taux de surcharge d'un cache mémoire	$\boldsymbol{\eta_{surcharge}}$
Taux de réutilisation idéal des données	$\boldsymbol{\eta_{ideal}}$
Taux de réutilisation idéal des données pour le segment $\boldsymbol{\Delta}$	$\boldsymbol{\eta_{\Delta}}$
Taux de réutilisation idéal des données pour le segment $\boldsymbol{\Delta}$ et l'angle $\boldsymbol{\phi}$	$\boldsymbol{\eta_{\Delta,\phi}}$
Grandeurs caractéristiques des ressources mémoire	
Latence de la mémoire SDRAM	$\text{Latence}_{\boldsymbol{SDRAM}}$
Latence moyenne d'accès au bus mémoire	$\hat{\text{Latence}}_{\boldsymbol{bus}}$
Débit du bus mémoire	$\text{debit}_{\boldsymbol{bus}}$
Stockage mémoire et localité spatiale	
Nombre moyen de *bins* utiles par page mémoire accédée	$\boldsymbol{\hat{N}_{bin}(page)}$
Nombre moyen de *bins* utiles par ligne mémoire accédée	$\boldsymbol{\hat{N}_{bin}(ligne)}$
Nombre moyen de *bins* utiles par mot mémoire accédé	$\boldsymbol{\hat{N}_{bin}(mot)}$
Nombre de mots mémoire dans une ligne mémoire	$\boldsymbol{N_{mot}(ligne)}$

Découpage du volume en bloc de voxels

Dimension transverse d'un bloc	$\boldsymbol{T_{bloc}}$
Hauteur d'un bloc	$\boldsymbol{H_{bloc}}$
Ratio entre dimension axiale et transverse d'un bloc	$\boldsymbol{r_{bloc}} = \dfrac{\boldsymbol{H_{bloc}}}{\boldsymbol{T_{bloc}}}$

Dimensions de la projection d'un bloc de voxels

Largeur (dimension $\boldsymbol{\rho}$)	$\boldsymbol{l_\rho}$
Longueur (dimension $\boldsymbol{\lambda}$)	$\boldsymbol{L_\lambda}$

Chapitre 3

Adéquation Algorithme, Architecture et Données

Nous avons pu constaté dans le chapitre précédent que la mise en place d'une stratégie d'accès mémoire efficace est une préoccupation majeure lors de l'accélération de la rétroprojection à cause du goulot d'étranglement constitué par l'accès à la mémoire externe. D'une part, les processeurs classiques et graphiques utilisent le plus intelligemment possible le cache mémoire à leur disposition. Celui-ci automatise la recherche des données manquantes en mémoire externe lors des défauts de cache (avec ou sans mécanisme de préchargement du cache mémoire). D'autre part, le processeur Cell et les architectures matérielles SoPC utilisent une mémoire *ping pong* qui effectue un préchargement "ad hoc" des données prochainement accédées par l'unité de calcul.

Dans ce chapitre, nous chercherons à définir une architecture matérielle SoPC constituée de plusieurs pipelines de rétroprojection offrant ainsi un important débit de calcul, et d'un cache mémoire effectuant un préchargement mémoire s'adaptant dynamiquement aux requêtes du module de rétroprojection. Le cache utilisé aura un double objectif : optimiser le débit mémoire en entrée du pipeline de rétroprojection, et rester le plus générique possible pour s'adapter à toutes applications effectuant un parcours mémoire continue dans des espaces mémoire n dimensionnels. Pour la rétroprojection, ce cache devra permettre le suivi de la sinusoïde 3D afin d'éviter au maximum les défauts de cache et réduire ainsi le coût d'accès à la mémoire. Nous focaliserons notre étude sur l'accélération de la rétroprojection à faisceaux parallèles 3D utilisée en tomographie TEP. Si des architectures FPGA ou ASICs ont été proposées pour la rétroprojection à faisceaux parallèles 2D comme celle de [Leeser 05], aucune à notre connaissance n'a été proposée pour la rétroprojection à faisceaux parallèles 3D.

Nous présenterons tout d'abord la problématique principale de cette étude avec une formalisation analytique des effets du goulot d'étranglement que constitue l'accès à la mémoire externe sur une architecture matérielle constituée de plusieurs pipelines de rétroprojection. Nous donnerons également quelques métriques permettant d'évaluer la pertinence d'un cache mémoire effectuant un préchargement de données. Puis dans la perspective de mise en place d'une stratégie de préchargement mémoire dynamique,

nous chercherons à augmenter la localité spatiale et temporelle des accès aux données en jouant sur l'ordre des boucles de l'algorithme et l'organisation des données en mémoire de stockage. Enfin nous proposerons une architecture matérielle 3P (Pipelinée, Préchargée et Parallélisée) destinée à être implémentée sur un SoPC.

1 Au-delà du mur mémoire

Dans cette section, après une brève description des boucles de calcul de la rétroprojection 3D, nous mettrons en évidence la nécessité de chercher un bon équilibre entre le débit de calcul et le débit mémoire pour atteindre une efficacité optimale de reconstruction. Nous étudierons ensuite le mode d'accès aux données qu'offrent les mémoires de type SDRAM communément utilisées comme mémoires externes. Nous donnerons enfin quelques axes de réflexion pour évaluer l'efficacité des caches mémoire destinés à masquer les accès à la mémoire externe.

1.1 Boucles de calcul de l'algorithme de rétroprojection 3D

Le code générique correspondant à l'algorithme de rétroprojection 3D (voir algorithme 1) fait apparaître trois boucles de calcul imbriquées : une sur l'espace image selon $\vec{r}$ avec trois sous boucles selon `xn`,`yn` et `zn`, une autre sur les segments $\boldsymbol{\Delta}$ et une dernière sur les angles de projection $\boldsymbol{\phi}$. Il n'y a aucune contrainte à priori sur l'ordre de ces boucles. Le corps de ces boucles est constitué d'une séquence assez courte de calcul correspondant à la mise à jour d'un voxel pour un segment et un angle de projection donnés. Cette séquence de calcul inclut le calcul des index des coordonnées de projection $\boldsymbol{\rho(\phi, \vec{r})}$ et $\boldsymbol{\lambda(\phi, \Delta, \vec{r})}$, l'interpolation bilinéaire avec l'accès à 4 *bins* et l'accumulation du *bin* interpolé pour mettre à jour le voxel traité.

Nous pouvons tout d'abord qualifier la rétroprojection d'algorithme massivement parallèle. En effet, il n'existe aucune dépendance forte de données. Ainsi chaque voxel peut être reconstruit de manière indépendante. L'unique contrainte de calcul se situe au niveau des boucles $\boldsymbol{\phi}$ et $\boldsymbol{\Delta}$ pour la mise à jour d'un voxel avec la somme de toutes les projections d'un voxel. La séquence de calcul se prête ainsi très bien à une implémentation sous forme d'un pipeline avec trois étages : calcul des coordonnées, interpolation bilinéaire et accumulation. Concernant les accès aux données, chaque mise à jour de voxel nécessite quatre accès en lecture pour obtenir les quatre *bins* utilisés lors de l'interpolation bilinéaire et un accès en lecture/écriture pour accumuler le *bin* interpolé à $\boldsymbol{f(\vec{r})}$. Nous verrons dans la suite de cette étude que ce sont ces accès mémoire qui constituent l'obstacle majeur à l'implémentation logicielle ou matérielle de l'algorithme de rétroprojection.

Algorithm 1 Code standard de l'algorithme de rétroprojection 3D

for $(\mathtt{xn}, \mathtt{yn}, \mathtt{zn}) = (\mathbf{0}, \mathbf{0}, \mathbf{0})$ to $(\mathtt{xn}_{\mathtt{max}} - \mathbf{1}, \mathtt{yn}_{\mathtt{max}} - \mathbf{1}, \mathtt{zn}_{\mathtt{max}} - \mathbf{1})$ **do**
  for $\mathtt{delta} = \mathbf{0}$ to $\mathtt{delta}_{\mathtt{max}} - \mathbf{1}$ **do**
    for $\mathtt{phi} = \mathbf{0}$ to $\mathtt{phi}_{\mathtt{max}} - \mathbf{1}$ **do**

// Calcul des coordonnées

$$\begin{aligned} \mathtt{rho}(\mathtt{phi}) &= \frac{\boldsymbol{\delta_{xy}}}{\boldsymbol{\delta_{\rho}}} \cdot \big((\mathtt{xn} - \mathtt{xn}_0) \cdot \mathbf{cos}\,\boldsymbol{\phi} + (\mathtt{yn} - \mathtt{yn}_0) \cdot \mathbf{sin}\,\boldsymbol{\phi}\big) + \mathtt{rho}_0 \\ \mathtt{lambda}(\mathtt{phi}, \mathtt{delta}) &= \frac{\boldsymbol{\delta_{xy}}}{\boldsymbol{\delta_{\lambda}}} \cdot \big((\mathtt{xn} - \mathtt{xn}_0) \cdot \mathbf{sin}\,\boldsymbol{\phi} - (\mathtt{yn} - \mathtt{yn}_0) \cdot \mathbf{cos}\,\boldsymbol{\phi}\big) \cdot \frac{\boldsymbol{\Delta}}{\mathbf{2} \cdot \boldsymbol{R_a}} + \\ &\quad \frac{\boldsymbol{\delta_z}}{\boldsymbol{\delta_{\lambda}}} \cdot (\mathtt{zn} - \mathtt{zn}_0) + \mathtt{lambda}_0 \end{aligned}$$

// Interpolation bilinéaire

$$\begin{aligned} \mathrm{bin}_{\boldsymbol{interp}} = & \quad (\mathbf{1} - \boldsymbol{\epsilon_{\lambda}}) \cdot (\mathbf{1} - \boldsymbol{\epsilon_{\rho}}) \cdot \boldsymbol{\mathcal{S}}_{\mathtt{lambda}_e, \mathtt{delta}}(\mathtt{rho}_e, \mathtt{phi}) \\ & + (\mathbf{1} - \boldsymbol{\epsilon_{\lambda}}) \cdot \boldsymbol{\epsilon_{\rho}} \cdot \boldsymbol{\mathcal{S}}_{\mathtt{lambda}_e, \mathtt{delta}}(\mathtt{rho}_e + \mathbf{1}, \mathtt{phi}) \\ & + \boldsymbol{\epsilon_{\lambda}} \cdot (\mathbf{1} - \boldsymbol{\epsilon_{\rho}}) \cdot \boldsymbol{\mathcal{S}}_{\mathtt{lambda}_e + \mathbf{1}, \mathtt{delta}}(\mathtt{rho}_e, \mathtt{phi}) \\ & + \boldsymbol{\epsilon_{\lambda}} \cdot \boldsymbol{\epsilon_{\rho}} \cdot \boldsymbol{\mathcal{S}}_{\mathtt{lambda}_e + \mathbf{1}, \mathtt{delta}}(\mathtt{rho}_e + \mathbf{1}, \mathtt{phi}) \end{aligned}$$

avec $\begin{cases} \mathtt{rho} = \mathtt{rho}_e + \boldsymbol{\epsilon_{\rho}} \\ \mathtt{lambda} = \mathtt{lambda}_e + \boldsymbol{\epsilon_{\lambda}} \end{cases}$

// Accumulation

$$\boldsymbol{f}(\mathtt{xn}, \mathtt{yn}, \mathtt{zn}) += \boldsymbol{J_{\Delta}} \cdot \mathrm{bin}_{\boldsymbol{interp}}$$

    end for
  end for
end for

1.2 Balance entre le débit de calcul et le débit mémoire

L'efficacité de reconstruction, définie par le temps moyen de traitement par opération élémentaire ($\hat{t}_{RP}/op$), dépend du bon équilibrage entre le coût de calcul ($C_{calcul}(op)$) et le coût des accès aux données ($C_{mem}(op)$). La manière non optimale d'effectuer une opération est d'amorcer l'accès aux données uniquement quand le besoin s'en fait sentir. Le temps de latence pour l'accès aux données s'ajoute alors au temps de calcul :

$$\hat{t}_{RP}/op = \frac{1}{N_{op}} \sum_{op} C_{calcul}(op) + C_{mem}(op) \tag{3.1}$$

Etant donné qu'il n'y a pas de dépendance de données, les calculs nécessités par l'opération en cours de traitement peuvent se faire en parallèle avec l'accès aux données nécessaires à la prochaine opération. Ce préchargement des données permet alors de masquer le temps d'accès aux données lorsque celui-ci reste inférieur au temps de calcul. L'efficacité de reconstruction est alors égale à :

$$\hat{t}_{RP}/op = \frac{1}{N_{op}} \cdot \sum_{op} max(C_{calcul}(op), C_{mem}(op)) \tag{3.2}$$

Une architecture en pipeline de l'algorithme de rétroprojection, aura un coût de calcul d'un cycle par opération (mise à jour d'un voxel). Et si l'accumulation se fait via une mémoire locale alors le coût mémoire ne dépendra que du temps d'accès aux *bins* (et du temps d'écriture du voxel reconstruit tout à la fin de la reconstruction). Ce coût est fonction du nombre de *bins* nécessaires à la mise à jour d'un voxel, $N_{bin/op}$ (quatre pour la rétroprojection 3D avec interpolation bilinéaire) et du coût moyen d'accès à un *bin* $\hat{C}_{mem}(bin)$:

$$\begin{cases} C_{calcul}(op) &= 1 \\ \hat{C}_{mem}(op) &= N_{bin/op} \cdot \hat{C}_{mem}(bin) \end{cases} \tag{3.3}$$

Malgré la mise en place d'une technique de préchargement des données, une architecture pipeline de rétroprojection restera très sensible aux coûts d'accès aux données comme l'indique l'équation suivante :

$$\hat{t}_{RP}/op = \frac{1}{N_{op}} \cdot \sum_{op} N_{bin/op} \cdot \hat{C}_{mem}(bin) \tag{3.4}$$

1.3 Accès à la SDRAM

La taille des données du scanner étant importante (de 10 Mo à 1Go selon les scanners), une mémoire externe à la puce de reconstruction de type SDRAM est utilisée pour stocker les sinogrammes. Ces mémoires sont organisées en rangées et colonnes de données. En mode *burst*, l'accès aux données se fait de manière groupée par mots mémoire contigus. L'accès au premier mot se fait avec un temps de latence important correspondant au temps de positionnement du numéro de rangée et du numéro de la première colonne. Par

contre, le temps d'accès aux mots mémoire suivants situés sur la même rangée mais sur les colonnes suivantes, se fera plus rapidement puisqu'un simple incrément du numéro de colonne suffit. L'unité de traitement est liée à la mémoire SDRAM via un bus mémoire. Le temps d'accès dépendra donc également de l'accès à ce bus mémoire. Au final, le coût moyen d'accès en SDRAM à une ligne mémoire $\hat{\boldsymbol{C}}_{\boldsymbol{SDRAM}}(\boldsymbol{ligne})$ est fonction de :

- $\hat{\text{Latence}}_{\boldsymbol{bus}}$: Latence moyenne d'accès au bus mémoire. Elle est liée au taux d'occupation du bus mémoire.
- $\text{debit}_{\boldsymbol{bus}}$: Débit du bus mémoire. Il correspond au nombre de mots mémoire délivrés par cycle. Il s'agit d'une constante des ressources mémoire.
- $\text{Latence}_{\boldsymbol{SDRAM}}$: Latence d'accès à la mémoire SDRAM pour le premier mot d'un *burst*. Il s'agit d'une constante des ressources mémoire.

Ce coût moyen s'exprime de la manière suivante :

$$\hat{\boldsymbol{C}}_{\boldsymbol{SDRAM}}(\boldsymbol{ligne}) = \text{Latence}_{\boldsymbol{SDRAM}} + \hat{\text{Latence}}_{\boldsymbol{bus}} + \frac{\boldsymbol{N_{mot}}(\boldsymbol{ligne}) - \boldsymbol{1}}{\text{debit}_{\boldsymbol{bus}}} \tag{3.5}$$

La latence des mémoires SDRAM actuelles est de l'ordre de la dizaine de ns ($\simeq 30$ ns). Selon la fréquence de l'unité de traitement, cela correspond à quelques cycles d'horloge (7 cycles @ 200 Mhz) ou à une centaine de cycles (100 cycles @ 3 Ghz). Puisque comme nous l'avons vu précédemment, les architectures pipelines sont très sensibles aux temps d'accès à la mémoire (voir équation 3.4), le temps de reconstruction est très lourdement pénalisé, d'un ou deux ordres de magnitude, par l'accès à la mémoire SDRAM.

1.4 Utilisation d'un cache mémoire

Les caches mémoire sont utilisés pour permettre un amortissement voire un masquage complet des coûts d'accès à la mémoire externe. Situés sur la puce à proximité de l'unité de traitement, ils délivrent en quelques cycles d'horloge leurs données. Toutefois, leurs tailles restent limitées pour les processeurs classiques à 16 ou 32 Ko pour le cache L1 et quelques Mo pour le cache L2 (4 Mo au maximum). Ainsi une stratégie d'accès mémoire doit être mise en place pour avoir en cache les données les plus pertinentes.

Les stratégies d'accès mémoire s'appuient sur la localité spatiale et temporelle des requêtes aux caches. Une forte localité spatiale des données demandées permet lors des mises à jour de cache de charger en plus de la donnée demandée au cache, les données adjacentes à celle-ci. Ainsi, lorsqu'une données de 4 octets est demandée aux caches des CPUs, c'est une ligne de cache de 64 octets qui est chargée. Mais la taille du cache étant limitée, seules les données fréquemment utilisées sont conservées et les autres doivent laissées leur place aux nouvelles données demandées. Ainsi les données doivent posséder une forte localité temporelle afin de rendre efficace l'utilisation du cache.

Nous donnons, ici, quelques grandeurs caractérisant la pertinence des stratégies mémoire à base de cache mémoire.

1.4.1 Grandeurs caractérisant l'efficacité d'un cache mémoire

L'efficacité d'une stratégie mémoire utilisant un cache mémoire peut être caractérisée par les grandeurs suivantes :

- $\boldsymbol{\eta_{ideal}}$: Taux de réutilisation idéale des données. Il constitue une propriété intrinsèque de l'algorithme implémenté et caractérise la localité géométrique de l'algorithme. Bien exploitée, cette localité permet une forte localité spatiale et temporelle.
- $\boldsymbol{\eta_{defaut}}$: Taux de défauts du cache. Il correspond à la proportion de requêtes au cache non satisfaites. Les défauts se produisent lorsque la donnée demandée n'est pas en cache.
- $\boldsymbol{\eta_{surcharge}}$: Taux de surcharge du cache. Il correspond à la proportion de données inutilement chargées en cache.
- $\boldsymbol{\hat{N}_{bin}(ligne)}$: Nombre de données élémentaires contenues dans une ligne mémoire demandée à la mémoire SDRAM. Il caractérise l'arrangement spatial des données dans la mémoire de stockage mémoire.
- $\boldsymbol{\hat{N}_{bin}(mot)}$: Nombre de données élémentaires contenues dans un mot mémoire. Il caractérise l'arrangement spatial des données dans la mémoire de stockage mémoire.
- $\boldsymbol{\hat{C}_{SDRAM}(ligne)}$: Coût moyen d'accès en SDRAM à une ligne mémoire. Il dépend du taux d'occupation du bus et des caractéristiques temporelles de la mémoire SDRAM.

1.4.2 Taux d'utilisation du cache

Idéalement, toute donnée demandée par l'unité de traitement doit être directement accessible via le cache et donc délivrée en quelques cycles d'horloge. Le taux $\boldsymbol{\eta_{ideal}}$, taux maximal de réutilisation des données que peut atteindre la meilleure stratégie d'accès mémoire, correspond au ratio suivant :

$$\boldsymbol{\eta_{ideal}} = \frac{\text{Nombre de lectures nécessaires}}{\text{Nombre de bins nécessaires}} \tag{3.6}$$

Toutefois, en pratique, la stratégie d'accès mémoire mise en place est pénalisée par le taux de défaut $\boldsymbol{\eta_{defaut}}$, correspondant à la proportion des lectures en défaut de cache et par le taux de surcharge du cache $\boldsymbol{\eta_{surcharge}}$, caractérisant la proportion de données inutilement chargées en cache. Ces deux ratios sont exprimés dans l'équation 3.7.

$$\begin{aligned} \boldsymbol{\eta_{defaut}} &= \frac{\text{Nombre de lecture en défaut de cache}}{\text{Nombre de lectures nécessaires}} \\ \boldsymbol{\eta_{surcharge}} &= \frac{\text{Nombre de bins mis en cache}}{\text{Nombre de bins nécessaires}} \end{aligned} \tag{3.7}$$

Ainsi en pratique, le taux moyen de réutilisation des données mises en cache $\boldsymbol{\eta_{cache}}$ est fonction de la localité spatiale et temporelle intrinsèque de l'algorithme $\boldsymbol{\eta_{ideal}}$, du

taux de défaut du cache $\boldsymbol{\eta_{defaut}}$, et du taux de surcharge du cache $\boldsymbol{\eta_{surcharge}}$. Il s'exprime sous la forme suivante :

$$\begin{aligned} \boldsymbol{\eta_{cache}} &= \frac{\text{Nombre de lecture en cache}}{\text{Nombre de bin mis en cache}} \\ &= \frac{\boldsymbol{\eta_{ideal}}}{\boldsymbol{\eta_{surcharge}}} \cdot (\mathbf{1} - \boldsymbol{\eta_{defaut}}) \end{aligned} \tag{3.8}$$

1.4.3 Coût mémoire avec utilisation d'un cache

Lorsqu'un cache mémoire est utilisé, le coût d'accès moyen à une donnée est fonction du coût global de mise en cache des données et du coût global des défauts :

$$\hat{\boldsymbol{C}}_{\boldsymbol{mem}}(\boldsymbol{bin}) = \frac{\text{Coûts de la mise en cache des données} + \text{Coûts des défauts}}{\text{Nombre de lectures nécessaires}} \tag{3.9}$$

Ce coût mémoire peut s'exprimer à l'aide des coûts locaux de défauts $\hat{\boldsymbol{C}}_{\boldsymbol{defaut}}(\boldsymbol{bin})$ et de mise en cache $\hat{\boldsymbol{C}}_{\boldsymbol{cache}}(\boldsymbol{bin})$ et des taux d'utilisation $\boldsymbol{\eta_{cache}}$ et de défauts $\boldsymbol{\eta_{defaut}}$ du cache :

$$\hat{\boldsymbol{C}}_{\boldsymbol{mem}}(\boldsymbol{bin}) = (\mathbf{1} - \boldsymbol{\eta_{defaut}}) \cdot \frac{\hat{\boldsymbol{C}}_{\boldsymbol{cache}}(\boldsymbol{bin})}{\boldsymbol{\eta_{cache}}} + \boldsymbol{\eta_{defaut}} \cdot \hat{\boldsymbol{C}}_{\boldsymbol{defaut}}(\boldsymbol{bin})$$

$$\begin{cases} \hat{\boldsymbol{C}}_{\boldsymbol{cache}}(\boldsymbol{bin}) &= \dfrac{\text{Coûts de la mise en cache des données}}{\text{Nombre de lectures en cache}} \\ \hat{\boldsymbol{C}}_{\boldsymbol{defaut}}(\boldsymbol{bin}) &= \dfrac{\text{Coûts des défauts}}{\text{Nombre de lectures en défaut}} \end{cases} \tag{3.10}$$

En se basant sur le modèle de temps d'accès à la SDRAM vu en 1.3, le coût de mise en cache d'une donnée élémentaire $\hat{\boldsymbol{C}}_{\boldsymbol{cache}}(\boldsymbol{bin})$ peut s'exprimer en fonction du nombre de *bins* contenus dans un mot mémoire $\boldsymbol{N_{bin}(mot)}$ (4 au maximum pour des *bins* codés sur 16 bits et un bus mémoire de taille 64 bits), du nombre moyen de *bins* contenus dans les lignes mémoire accédées en SDRAM $\hat{\boldsymbol{N}}_{\boldsymbol{bin}}(\boldsymbol{ligne})$ et de la latence et du débit caractéristiques des ressources mémoire ($\text{debit}_{\boldsymbol{bus}}$,$\hat{\text{Latence}}_{\boldsymbol{bus}}$,$\text{Latence}_{\boldsymbol{SDRAM}}$) :

$$\begin{aligned} \hat{\boldsymbol{C}}_{\boldsymbol{cache}}(\boldsymbol{bin}) &= \frac{\hat{\boldsymbol{C}}_{\boldsymbol{SDRAM}}(\boldsymbol{ligne})}{\hat{\boldsymbol{N}}_{\boldsymbol{bin}}(\boldsymbol{ligne})} \\ &= \frac{\hat{\text{Latence}}_{\boldsymbol{bus}} + \text{Latence}_{\boldsymbol{SDRAM}}}{\hat{\boldsymbol{N}}_{\boldsymbol{bin}}(\boldsymbol{ligne})} + \left(\frac{\mathbf{1}}{\hat{\boldsymbol{N}}_{\boldsymbol{bin}}(\boldsymbol{mot})} - \frac{\mathbf{1}}{\hat{\boldsymbol{N}}_{\boldsymbol{bin}}(\boldsymbol{ligne})}\right) \cdot \frac{\mathbf{1}}{\text{debit}_{\boldsymbol{bus}}} \end{aligned} \tag{3.11}$$

2 Introduction de localité spatiale et temporelle

Nous montrons dans cette section que l'algorithme de rétroprojection possède une forte localité géométrique. Bien utilisée, cette localité est synonyme de forte localité spatiale et temporelle. Nous évaluerons cette localité géométrique, puis nous verrons que couplée avec un arrangement spatial adéquat des données dans la mémoire externe, elle permet de réduire significativement le coût moyen d'accès aux données.

2.1 Augmentation de la localité géométrique η_{ideal}

2.1.1 Reconstruction par bloc

La projection d'un bloc 3D de voxels pour un segment $\boldsymbol{\Delta}$ et un angle de projection $\boldsymbol{\phi}$ donnés, correspond à un plan 2D de *bins*. De fait, de nombreux voxels ont des données de projections ou *bins* en commun. Ainsi, si la reconstruction d'un bloc de voxel se fait par mises à jour successives (boucle sur $\boldsymbol{\phi}$ et $\boldsymbol{\Delta}$), la localité spatiale et temporelle s'en trouve renforcée. Il suffit pour cela de couper en deux la boucle sur le volume de l'algorithme 1, avec une boucle externe sur les blocs du volume et une boucle interne qui parcourt les voxels du bloc comme décrit dans le pseudo-code de l'algorithme 2.

Algorithm 2 Réordonnancement des boucles avec une reconstruction par blocs de voxels

```
for n = 0 to n_max do
  for Delta = 0 to Delta_max do
    for phi = 0 to phi_max do
      for r_min(n) to r_max(n) do
        f(r) = f(r) +
        bin(phi, Delta, rho(phi, r)
        lambda(phi, x))
      end for
    end for
  end for
end for
```

Le taux de réutilisation de donnée $\boldsymbol{\eta}$ d'un bloc de voxel est alors fonction des dimensions du bloc. Nous évaluons dans la suite ce taux de réutilisation $\boldsymbol{\eta_\Delta}$ lors de la rétroprojection 3D des données contenues dans un segment $\boldsymbol{\Delta}$ fixé dans un bloc de voxel de dimension transverse $\boldsymbol{T_{bloc}}$ et de hauteur $\boldsymbol{H_{bloc}}$. Par ailleurs, nous utiliserons la grandeur $\boldsymbol{r_{bloc}}$, ratio entre la dimension axiale $\boldsymbol{H_{bloc}}$ et la dimension transverse $\boldsymbol{T_{bloc}}$ d'un bloc :

$$\boldsymbol{r_{bloc}} = \frac{\boldsymbol{H_{bloc}}}{\boldsymbol{T_{bloc}}} \tag{3.12}$$

2.1.2 Estimation de η_{Δ}

Selon l'algorithme 2, lors de la rétroprojection d'un segment $\boldsymbol{\Delta}$, il faut accéder à une page mémoire constituée de $\texttt{phi}_{\texttt{max}}$ plans mémoire. Les plans mémoire, projections $\boldsymbol{p_{\phi,\Delta}}$ d'un bloc de voxel, ont pour largeur $\boldsymbol{l_{\rho}}$ selon la dimension $\boldsymbol{\rho}$ et pour longueur $\boldsymbol{L_{\lambda}}$ selon la dimension $\boldsymbol{\lambda}$. Ainsi la page mémoire à accéder lors de la rétroprojection du segment $\boldsymbol{\Delta}$ dans le bloc a pour taille :

$$
\begin{aligned}
\boldsymbol{N_{\text{bin}}(page)} &= \sum_{\texttt{phi}=\mathbf{0}}^{\texttt{phi}_{max}-1} \boldsymbol{l_{\rho} * L_{\lambda}} \\
\text{avec} \quad \begin{cases} \boldsymbol{l_{\rho}} & = \mathbf{T_{bloc}} \cdot \left(|\cos(\phi)| + |\sin(\phi)| \right) \\ & = \boldsymbol{\sqrt{2}} \cdot \mathbf{T_{bloc}} \cdot \cos|\phi - \frac{\pi}{4}| \\ \boldsymbol{L_{\lambda}} & = \mathbf{H_{bloc}} + \mathbf{T_{bloc}} \cdot \left(|\cos\phi| + |\sin\phi| \right) \cdot \frac{\boldsymbol{\Delta}}{\mathbf{2} \cdot \boldsymbol{R_a}} \\ & = \mathbf{T_{bloc}} \cdot \left(\boldsymbol{r_{bloc}} + \frac{\boldsymbol{\Delta}}{\boldsymbol{\sqrt{2}} \cdot \boldsymbol{R_a}} \cdot \cos|\phi - \frac{\pi}{4}| \right) \end{cases}
\end{aligned}
\tag{3.13}
$$

Le taux de réutilisation $\boldsymbol{\eta_{\Delta,\phi}}$ pour un segment $\boldsymbol{\Delta}$ et un angle $\boldsymbol{\phi}$ donnés, correspond au ratio entre le nombre de bins demandés, c'est-à-dire le nombre de voxels d'un bloc lorsqu'il n'y a pas d'interpolation bi-linéaire, et le nombre de bins contenus dans le plan mémoire projection du bloc de voxels. Son expression est ainsi égal à :

$$
\begin{aligned}
\boldsymbol{\eta_{\Delta,\phi}} &= \frac{\boldsymbol{N_{voxel}(bloc)}}{\boldsymbol{N_{bin}(plan)}} \\
&= \frac{\boldsymbol{r_{bloc}} \cdot \mathbf{T_{bloc}^3}}{\boldsymbol{l_{\rho}} \cdot \boldsymbol{L_{\lambda}}} \\
&= \frac{\mathbf{T_{bloc}}}{\boldsymbol{\alpha_{\phi}} + \boldsymbol{\alpha_{\phi}^2} \cdot \boldsymbol{\beta_{bloc,\Delta}}}
\end{aligned}
\tag{3.14}
$$

$$
\text{avec} \quad \begin{cases} \boldsymbol{\alpha_{\phi}} = \boldsymbol{\sqrt{2}} \cdot \cos|\phi - \frac{\pi}{4}| \\ \boldsymbol{\beta_{bloc,\Delta}} = \dfrac{\boldsymbol{\Delta}}{\mathbf{2} \cdot \boldsymbol{R_a} \cdot \boldsymbol{r_{bloc}}} \end{cases}
$$

Le taux moyen de réutilisation $\boldsymbol{\eta_{\Delta}}$ pour un segment donné est alors égal à :

$$
\boldsymbol{\eta_{\Delta}} = \frac{\mathbf{1}}{\texttt{phi}_{\texttt{max}}} \cdot \sum_{\texttt{phi}=\mathbf{0}}^{\texttt{phi}_{max}} \boldsymbol{\eta_{\Delta,\phi}} \tag{3.15}
$$

Nous pouvons approximer l'expression de $\boldsymbol{\eta_{\Delta}}$ par la moyenne entre le maximum et le minimum pris par $\boldsymbol{\eta_{\Delta,\phi}}$. Le maximum et le minimum de $\boldsymbol{\eta_{\Delta,\phi}}$ correspondent respectivement à $\boldsymbol{\phi} = \{\mathbf{0}, \frac{\boldsymbol{\pi}}{\mathbf{2}}\}$ et à $\boldsymbol{\phi} = \{\frac{\boldsymbol{\pi}}{\mathbf{4}}, \frac{\mathbf{3} \cdot \boldsymbol{\pi}}{\mathbf{4}}\}$. Ainsi $\boldsymbol{\eta_{\Delta}}$ a alors pour expression :

$$\boldsymbol{\eta_{\Delta}} \simeq \frac{\max(\boldsymbol{\eta_{\Delta,\phi}}) + \mathrm{Min}(\boldsymbol{\eta_{\Delta,\phi}})}{\mathbf{2}}$$

avec

$$\begin{cases} \mathrm{Max}(\boldsymbol{\eta_{\Delta,\phi}}) = \dfrac{\mathbf{T_{bloc}}}{\mathbf{1} + \boldsymbol{\beta_{bloc,\Delta}}} \\ \mathrm{Min}(\boldsymbol{\eta_{\Delta,\phi}}) = \dfrac{\mathbf{T_{bloc}}}{\sqrt{\mathbf{2}} + \mathbf{2} \cdot \boldsymbol{\beta_{bloc,\Delta}}} \end{cases} \tag{3.16}$$

Nous avons évalué ce taux de réutilisation des données $\boldsymbol{\eta_{\Delta}}$ pour les segments $\mathbf{\{+2, -2\}}$, $\mathbf{\{+1, -1\}}$ et 0 définis par le scanner HR+ avec un *span* de 9. Nous avons tout d'abord observé que pour maximiser le taux de réutilisation des données, un bloc en forme de parallélépipède doit avoir une dimension transverse plus importante que sa dimension axiale. Les graphes 3.1, 3.2, 3.3 représentent respectivement le $\boldsymbol{\eta}$ moyen pour les segments 0, $\mathbf{\{+2, -2\}}$ et pour l'ensemble des segments (chaque segment représente $\mathbf{20\%}$ des calculs) en fonction du ratio $\boldsymbol{r_{bloc}}$ et de la taille du bloc égal à $\boldsymbol{T_{bloc}^2 \cdot H_{bloc}}$. Pour le segment 0, plus $\boldsymbol{r_{bloc}}$ est proche de zéro, plus grand est le taux de réutilisation des données. Par contre pour le segment 2, le meilleur taux de réutilisation est atteint pour $\boldsymbol{r_{bloc} = 0.1}$. Ainsi en moyenne, $\boldsymbol{\eta}$ est maximal pour $\boldsymbol{r_{bloc} = 0.05}$. Toutefois, ces valeurs obtenues pour $\boldsymbol{\eta}$ ne tiennent pas compte de l'interpolation bilinéaire qui requiert un plus grand nombre de données. Ainsi l'interpolation selon $\boldsymbol{\lambda}$, la dimension axiale, nécessite un plan mémoire supplémentaire. Pour amortir le coût de ce plan supplémentaire, il faut donc augmenter la dimension axiale du bloc de voxel. C'est pourquoi dans la suite de cette étude, nous avons opté pour un ratio $\boldsymbol{r_{bloc}}$ égal à 0.2 au lieu de 0.05.

Le graphe 3.4 présente l'évolution de $\boldsymbol{\eta_{\Delta}}$ pour différentes tailles de blocs parallélépipèdiques avec un ratio $\boldsymbol{r_{bloc} = 0.2}$. Par exemple, le bloc de taille 768 voxels a pour dimensions $\mathbf{16^2 \cdot 3}$ et a un taux de réutilisation de donnée de 13 pour le segment 0, de 10 pour les segments $\mathbf{\{+1, -1\}}$ et de 8 pour les segments $\mathbf{\{+2, -2\}}$. Ainsi, le taux de réutilisation moyen $\boldsymbol{\eta}$ est d'environ 9. Pour la rétroprojection 3D avec interpolation bilinéaire, ce taux de réutilisation des données doit être multiplié par 4. En effet, le nombre de lectures est 4 fois plus important à cause de l'interpolation bilinéaire mais le nombre de données chargées en cache reste inchangé. Ainsi pour un bloc de taille $\mathbf{16^2 \cdot 3}$, le taux de réutilisation peut être évalué à 36 en moyenne pour une rétroprojection 3D-RP effectuée à partir de sinogrammes provenant d'un scanner HR+.

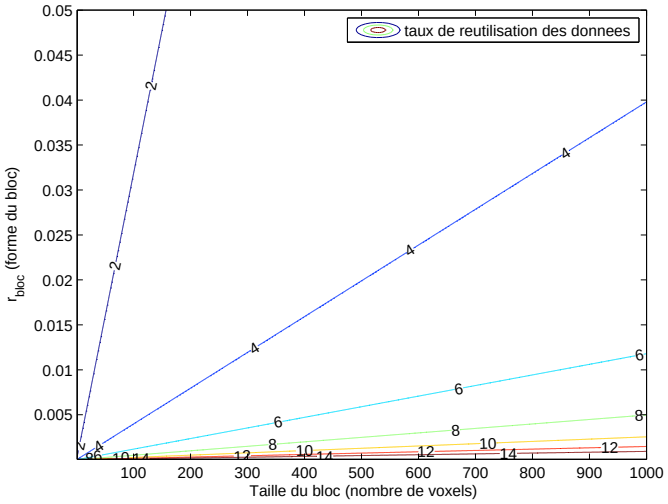


FIG. 3.1 – Taux de réutilisation $\boldsymbol{\eta}$ pour la rétroprojection du segment 0, en fonction de la forme ($\boldsymbol{r_{bloc}}$) et de la taille du bloc

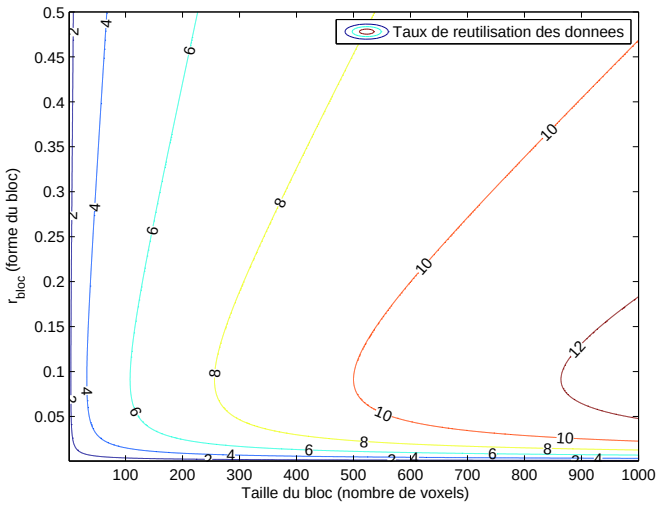


FIG. 3.2 – Taux de réutilisation $\boldsymbol{\eta}$ pour la rétroprojection des segments +2 et -2, en fonction de la forme ($\boldsymbol{r_{bloc}}$) et de la taille du bloc

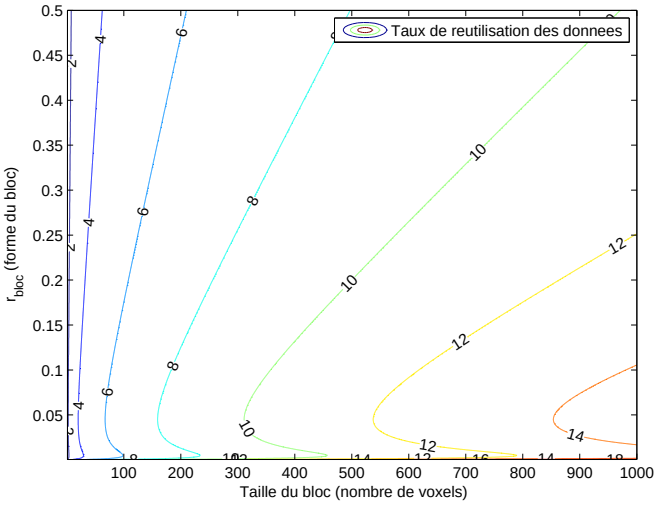


FIG. 3.3 – Taux de réutilisation $\boldsymbol{\eta}$ pour une rétroprojection complète (pour tous les segments), en fonction de la forme ($\boldsymbol{r_{bloc}}$) et de la taille du bloc

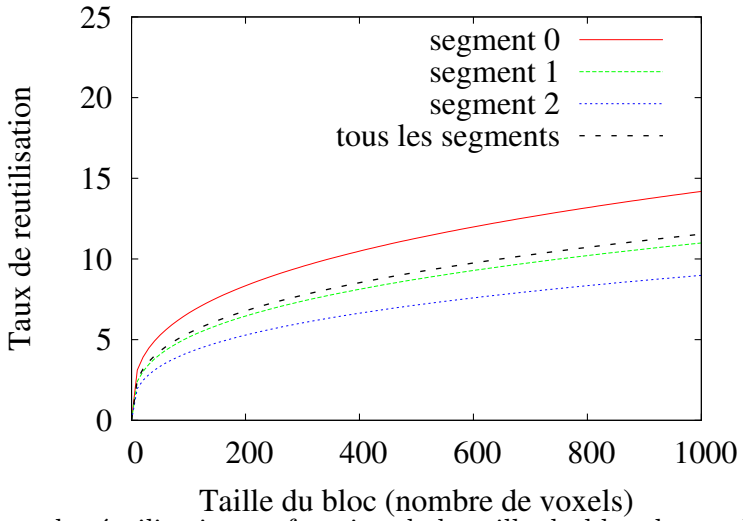


FIG. 3.4 – Taux de réutilisation en fonction de la taille du bloc de voxels reconstruit

2.2 Augmentation de la localité spatiale dans la mémoire de stockage

L'organisation des données dans la mémoire doit être en adéquation avec le parcours mémoire qui est effectué dans l'espace 4D des données du scanner durant la reconstruction. Ce parcours mémoire diffère selon l'ordre des boucles en $\boldsymbol{\Delta}$ et $\boldsymbol{\phi}$. Nous examinerons ici les différents parcours mémoire possible avec deux objectifs : augmenter le taux de réutilisation des données et faciliter un mécanisme de prédiction du parcours mémoire.

2.3 Boucle interne selon $\boldsymbol{\Delta}$

Lorsque la boucle en $\boldsymbol{\Delta}$ est plus interne que la boucle en $\boldsymbol{\phi}$, la reconstruction peut se décomposer en $\texttt{phi}_{\texttt{max}}$ boucles selon $\boldsymbol{\Delta}$. Pour chacune de ces boucles, l'angle $\boldsymbol{\phi}$ est fixé et donc $\boldsymbol{\rho(\phi)}$ également. Ensuite, pour chaque valeur de $\boldsymbol{\Delta}$, il faut déterminer la coordonnée $\boldsymbol{\lambda(\Delta)}$, qui a pour expression :

$$\boldsymbol{\lambda(\Delta)} = -\frac{-\boldsymbol{x}\cdot\sin\boldsymbol{\phi} + \boldsymbol{y}\cdot\cos\boldsymbol{\phi}}{\boldsymbol{2\cdot R_a}}\cdot\boldsymbol{\Delta} + \boldsymbol{z} \tag{3.17}$$

L'équation de $\boldsymbol{\lambda(\Delta)}$ correspond à une droite affine selon $\boldsymbol{\Delta}$. Les accès successifs à la mémoire pour reconstruire un voxel tracent alors une ligne dans la donnée 2D correspondant à un $\boldsymbol{\phi}$ et à un $\boldsymbol{\rho}$ fixés. C'est pourquoi, nous appellerons ces données 2D, linogrammes $\text{lino}_{\texttt{phi,rho}}(\texttt{delta}, \texttt{lambda})$. La figure 3.5 illustre le parcours mémoire effectué en deux étapes lors de la reconstruction d'un voxel : d'abord sélection des linogrammes puis parcours de ces linogrammes 2D.

2.4 Boucle interne selon $\boldsymbol{\phi}$

Lorsque la boucle en $\boldsymbol{\phi}$ est plus interne que la boucle en $\boldsymbol{\Delta}$, la reconstruction s'effectue en $\texttt{Delta}_{\texttt{max}}$ boucles selon $\boldsymbol{\phi}$. Pour chacune de ces boucles, le segment $\boldsymbol{\Delta}$ est fixé. Ensuite, pour chaque valeur de $\boldsymbol{\phi}$, il faut déterminer les valeurs de $\boldsymbol{\rho(\phi)}$ et $\boldsymbol{\lambda(\phi)}$ qui ont pour expression :

$$\begin{cases} \boldsymbol{\rho(\phi)} &= \boldsymbol{x}\cdot\cos\boldsymbol{\phi} + \boldsymbol{y}\cdot\sin\boldsymbol{\phi} \\ \boldsymbol{\lambda(\phi)} &= \boldsymbol{z} - \big(-\boldsymbol{x}\cdot\sin\boldsymbol{\phi} + \boldsymbol{y}\cdot\cos\boldsymbol{\phi}\big)\cdot\dfrac{\boldsymbol{\Delta}}{\boldsymbol{2\cdot R_a}} \end{cases} \tag{3.18}$$

Les équations de $\boldsymbol{\rho(\phi)}$ et $\boldsymbol{\lambda(\phi)}$ correspondent à une sinusoïde 3D dessinée dans le segment $\boldsymbol{\Delta}$. C'est pourquoi, nous appellerons également ces segments, sinogrammes $\text{sino}_{\texttt{delta}}(\boldsymbol{\phi}, \boldsymbol{\lambda}, \boldsymbol{\rho})$. La figure 3.6 illustre le parcours mémoire ainsi effectué en deux étapes lors de la reconstruction d'un voxel : d'abord sélection des sinogrammes 3D puis parcours de ces sinogrammes.

2.5 Choix du parcours mémoire

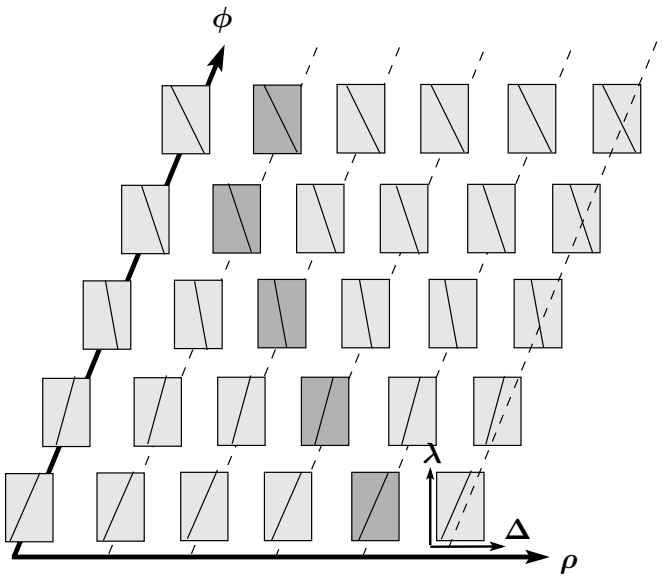
Aux premiers abords, une reconstruction avec une boucle interne en $\boldsymbol{\Delta}$ trace un parcours mémoire plus facilement exploitable par un cache mémoire pourvu d'un mécanisme de préchargement. En effet, il est à priori plus simple de suivre une ligne tracée dans

un espace mémoire 2D, qu'une sinusoïde dans un espace mémoire 3D. Toutefois deux remarques viennent contredire ce premier constat.

Premièrement, en pratique la boucle selon $\boldsymbol{\Delta}$ est bien plus courte que celle selon $\boldsymbol{\phi}$. Par exemple, le scanner HR+ utilise en moyenne 5 segments (boucle selon $\boldsymbol{\Delta}$) et une centaine d'angles de projection (boucle selon $\boldsymbol{\phi}$). Or un mécanisme de préchargement est plus efficacement utilisé lors d'un parcours mémoire long. En effet, les éventuels coûts d'initialisation dus aux changements d'espace mémoire, sont ainsi plus facilement amortis. Dans ce contexte, une boucle interne en $\boldsymbol{\Delta}$, dessine plusieurs parcours mémoire (une centaine pour le scanner HR+) trop courts (de longueur 5 pour le scanner HR+). A l'inverse, une boucle interne en $\boldsymbol{\phi}$, dessine quelques parcours mémoire (5) atteignant une longueur critique (environ 100 accès) pour être efficacement suivis par un mécanisme de préchargement en amortissant les coûts d'initialisation.

Deuxièmement, comme nous l'avons vu précédemment, une reconstruction par blocs est fortement recommandée pour bénéficier au maximum de la localité géométrique intrinsèque de l'algorithme de rétroprojection. Or ces blocs de voxels demandent pour chaque valeur de $\boldsymbol{\phi}$ et de $\boldsymbol{\Delta}$, d'accéder à un plan mémoire de largeur $\boldsymbol{l_\rho}$ et de longueur $\boldsymbol{L_\lambda}$ (voir équation 3.13). Ainsi d'une part, une boucle interne selon $\boldsymbol{\Delta}$ demandera pour la reconstruction d'un bloc de parcourir en parallèle des lignes de largeur $\boldsymbol{L_\lambda}$ dans $\boldsymbol{l_\rho}$ linogrammes 2D. D'autre part, une boucle interne selon $\boldsymbol{\phi}$ demandera de parcourir une sinusoïde de base rectangle de dimension $\boldsymbol{l_{\rho * \lambda}}$ dans un unique sinogramme 3D. Dans un cas, les accès sont dispersés dans l'espace des données, et dans l'autre cas ils sont plus compacts.

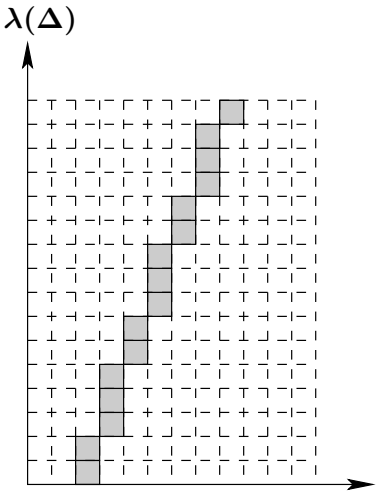
En conclusion, une reconstruction par blocs avec une boucle interne selon $\boldsymbol{\phi}$ sera préférée car elle constitue la stratégie mémoire la plus adaptée à l'utilisation d'un cache mémoire intégrant un mécanisme de préchargement des accès mémoire. La reconstruction sera effectuée segment après segment. Ainsi pour le scanner HR+ et ses cinq segments, nous effectuerons cinq rétroprojections successives, le volume reconstruit correspondra à la somme des cinq volumes partiellement reconstruits. Pour le segment 0 ($\boldsymbol{\Delta = 0}$) la reconstruction correspondra à une rétroprojection 2D, et pour les segments {+1,-1} ($\boldsymbol{\Delta = span * \delta_\Delta}$) et {+2,-2} ($\boldsymbol{\Delta = 2 * span * \delta_\Delta}$), la rétroprojection correspondra à une rétroprojection 3D. En effet, si pour le segment 0 ($\boldsymbol{\Delta = 0}$) la reconstruction d'un voxel nécessite la lecture d'un seul sinogramme 2D et donc un parcours mémoire 2D (voir figure 3.7), pour les segments {+1,-1} ($\boldsymbol{\Delta = span * \delta_\Delta}$) et {+2,-2} ($\boldsymbol{\Delta = 2 * span * \delta_\Delta}$), la reconstruction d'un voxel nécessite la lecture de plusieurs sinogrammes 2D et donc un parcours mémoire 3D avec un déplacement selon la dimension $\boldsymbol{\lambda}$ (voir figure 3.8).



$$\rho(\phi) = x \cdot \cos\phi + y \cdot \sin\phi$$

$$\lambda(\Delta) = -\frac{-x \cdot \sin\phi + y \cdot \cos\phi}{2 \cdot R_a} \cdot \Delta + z$$

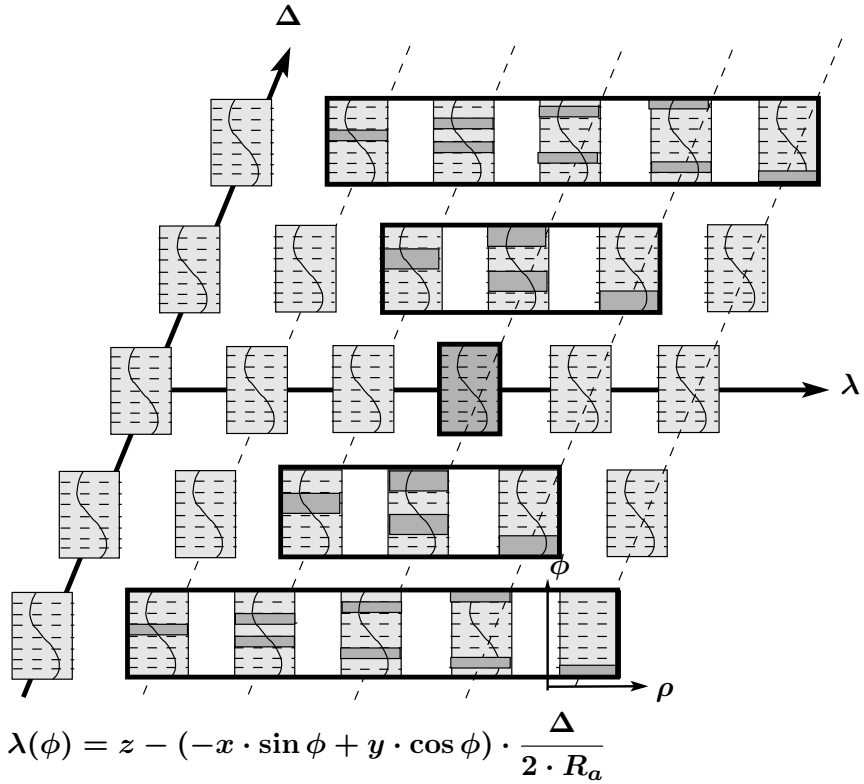
(a) Première étape : Sélection du linogramme $\text{lino}_{\texttt{phi,rho}}$ correspondant à l'angle ϕ traité.



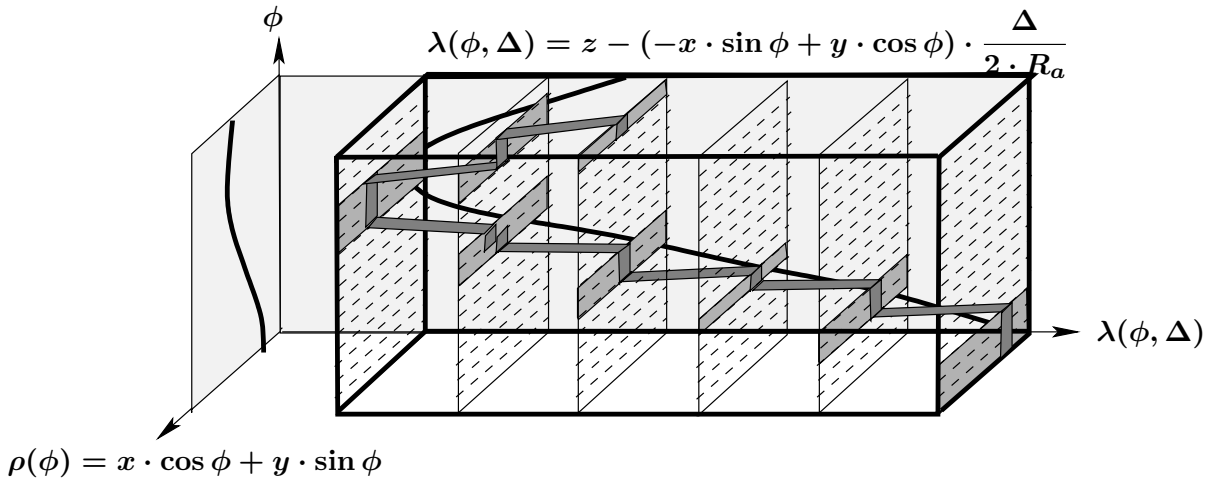
$$\lambda(\Delta) = -\frac{-x \cdot \sin\phi + y \cdot \cos\phi}{2 \cdot R_a} \cdot \Delta + z$$

(b) Deuxième étape : Parcours mémoire dans le linogramme 2D $\text{lino}_{\texttt{phi,rho}}(\texttt{delta}, \texttt{lambda})$.

FIG. 3.5 – Parcours mémoire en deux étapes avec la boucle la plus interne selon Δ



(a) Première étape : Sélection du sinogramme 3D correspondant au segment Δ traité.



(b) Deuxième étape : Parcours mémoire dans le sinogramme 3D $\text{sino}_{\texttt{delta}}(\texttt{phi}, \texttt{lambda}, \texttt{rho})$.

FIG. 3.6 – Parcours mémoire en deux étapes avec la boucle la plus interne selon ϕ

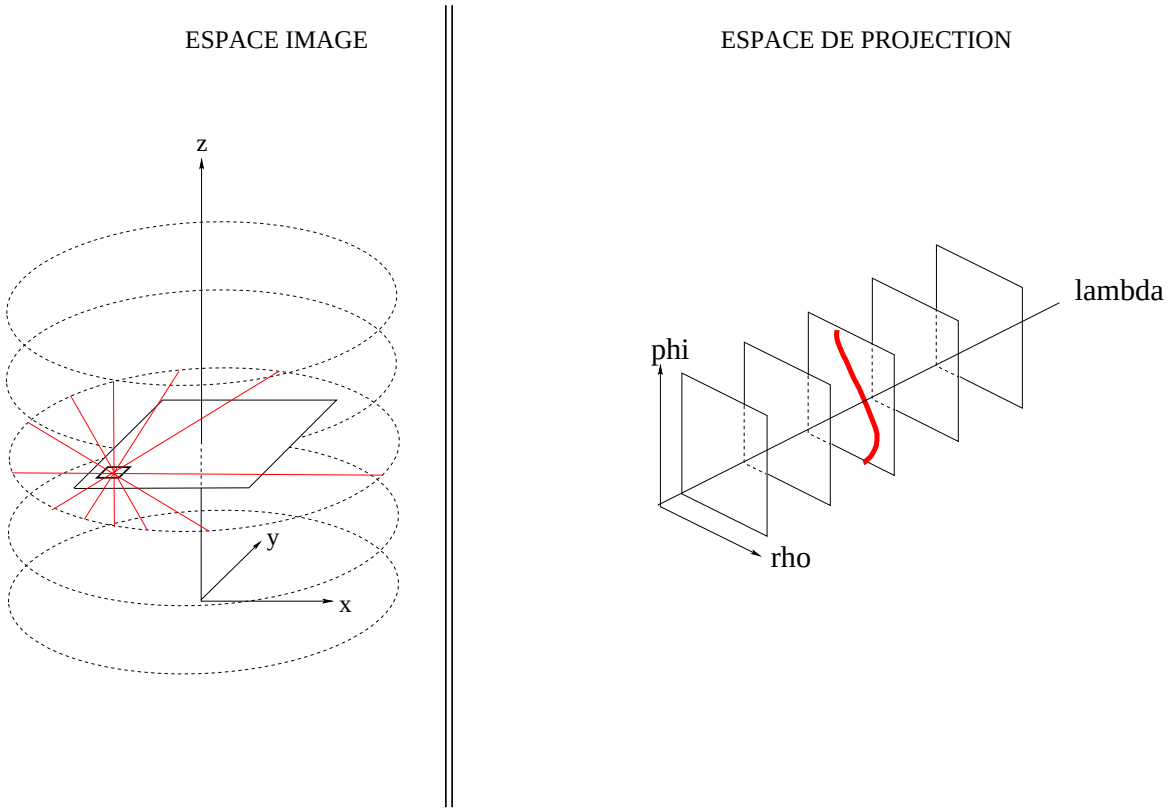


FIG. 3.7 – Correspondance entre un voxel du volume et ses données de projection pour le segment 0. Les données de ses sinogrammes (appelés “direct”) correspondent aux paires de détecteurs d’un même anneau (LORs intra-anneau).

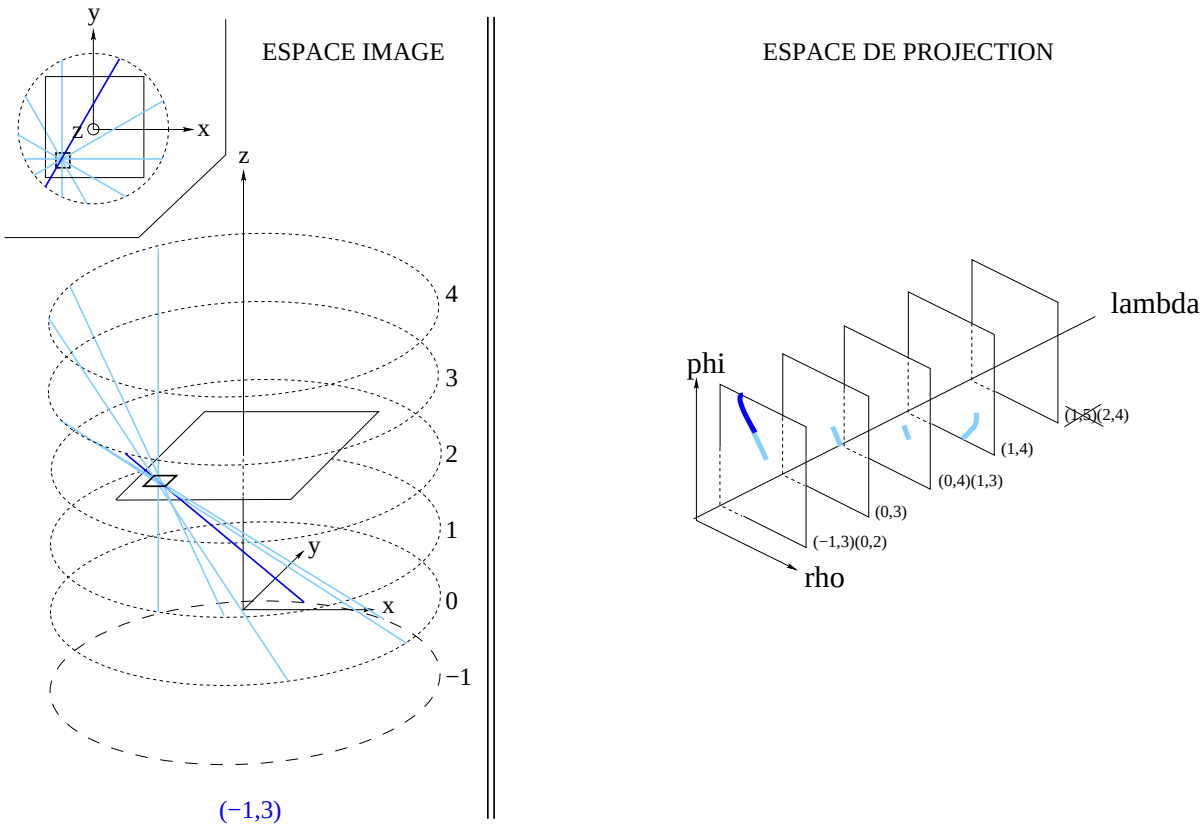


FIG. 3.8 – Correspondance entre données de projection et un voxel du volume pour un segment autre que le segment 0. Les données de ses sinogrammes (appelés “indirects”) correspondent aux paires de détecteurs appartenant à deux anneaux différents (LORs inter-anneau). Par exemple, la LOR en bleu foncé relit un détecteur de l’anneau -1 (anneau virtuel utilisé pour l’algorithme 3D-RP) avec l’anneau 3.

3 Architecture 3P : Pipelinée, Préchargée dynamiquement et Parallélisée

Nous présentons ici, une architecture matérielle faite sur mesure pour la rétroprojection TEP 3D et dédiée à être intégrée dans un SoPC (*System on Programmable Chip*). Trois principes caractérisent cette architecture 3P avec tout d'abord la mise en place d'un pipeline de rétroprojection 3D, puis l'utilisation d'un cache 3D-AP afin d'effectuer le Préchargement des accès à la mémoire externe, et enfin la Parallélisation des unités de traitement avec la mise en place d'une hiérarchie mémoire. Cette architecture permet d'effectuer la rétroprojection d'un bloc de voxels à partir des données d'un seul segment. Elle effectue ainsi un parcours mémoire en sinusoïde 3D.

3.1 Pipeline : une opération par cycle

Le pipeline présenté sur la figure 3.9 intègre les trois étapes de calcul nécessaires à la mise à jour d'un voxel : le calcul des coordonnées de projection $\texttt{rho(phi, xn, yn)}$ et $\texttt{lambda(phi, Delta, xn, yn, zn)}$ (étages A et B du pipeline), le calcul de l'interpolation bilinéaire (étages C, D et E du pipeline) et l'accumulation du *bin* interpolé à la densité $\boldsymbol{f}^*\texttt{(xn, yn, zn)}$ (étage F du pipeline). Le pipeline comporte en tout six étages de calcul afin de minimiser le chemin critique.

3.1.1 Flot de données et flot de contrôle

Le flot de données traverse tous les étages du pipeline : coordonnées de projection dans le bloc de calcul des coordonnées, *bins* dans le bloc d'interpolation et voxels reconstruits dans le bloc d'accumulation. Un flot de contrôle accompagne ce flot de données. Il correspond au numéro du voxel (indice $\boldsymbol{n}$) et à l'angle `phi` traités. Une machine à état en début de pipeline implémente les boucles sur les coordonnées des voxels et sur l'angle `phi`.

L'accès aux 4 *bins* nécessaires à l'interpolation bilinéaire est fait à l'aide d'une interface mémoire contrôlant le cache mémoire et permettant de bloquer le pipeline selon la disponibilité des données en cache. Un flot de contrôle à contre sens met en oeuvre ce blocage si nécessaire. Par ailleurs, une FIFO entre les étages de calcul des coordonnées de projection et ceux de calcul de l'interpolation bilinéaire, permet d'amortir une demande d'arrêt du pipeline.

3.1.2 Partionnement logiciel/matériel

Les calculs ont été adaptés pour minimiser ceux pris en charge par le matériel afin de minimiser la surface occupée du FPGA. Seules les opérations répétitives sont effectuées en matériel, les autres sont pré-calculées par le processeur du SoPC. Plusieurs valeurs précalculées sont alors passées en paramètre par le processeur vers l'architecture matérielle. Les équations correspondantes aux étages A et B de calcul des coordonnées $\boldsymbol{\rho}$ et $\boldsymbol{\lambda}$ (voir équations 1.47 du chapitre 1) ont été ainsi adaptées pour une implémentation en

matériel (voir equation 3.19). Les deux tableaux `Tcos[phi]` et `Tsin[phi]` sont contenus dans une mémoire RAM indexée par `phi`. Les coordonnées des voxels $\mathtt{xn}'$, $\mathtt{yn}'$ et $\mathtt{zn}'$ sont incrémentés dans la machine à état en haut de la chaîne de traitement, à partir des coordonnées du voxel origine du bloc de voxels et de la taille du bloc.

Calculs effectués en matériel (étages A et B de l'architecture 3P) :

$$\begin{cases} \mathtt{rho} &= \mathtt{rho}_0 + \left(\mathtt{xn}' * \mathtt{Tcos[phi]} + \mathtt{yn}' * \mathtt{Tsin[phi]}\right) \\ \mathtt{lambda} &= \left(\mathtt{lambda}_0 + \mathtt{zn}'\right) - \left(\mathtt{KDelta} * \left(\mathtt{yn}' * \mathtt{Tcos[phi]} - \mathtt{xn}' * \mathtt{Tsin[phi]}\right)\right) \end{cases}$$

Calculs effectués en logiciel (processeur du SoPC) :

$$\begin{cases} \mathtt{KDelta} &= \mathtt{span} \cdot \mathtt{Delta} \cdot \dfrac{\boldsymbol{\delta_\rho}}{\boldsymbol{R_a}} \\ \mathtt{Tcos[phi]} &= \dfrac{\boldsymbol{\delta_{xy}}}{\boldsymbol{\delta_\rho}} \mathbf{cos}(\mathtt{phi} \cdot \dfrac{\boldsymbol{\pi}}{\mathtt{phi}_{\boldsymbol{max}}}) \\ \mathtt{Tsin[phi]} &= \dfrac{\boldsymbol{\delta_{xy}}}{\boldsymbol{\delta_\rho}} \mathbf{sin}(\mathtt{phi} \cdot \dfrac{\boldsymbol{\pi}}{\mathtt{phi}_{\boldsymbol{max}}}) \\ \mathtt{xn}' &= \mathtt{xn} - \mathtt{xn}_0 \\ \mathtt{yn}' &= \mathtt{yn} - \mathtt{yn}_0 \\ \mathtt{zn}' &= \mathtt{zn} - \mathtt{zn}_0 \end{cases} \tag{3.19}$$

3.1.3 Calculs en virgule fixe

La technologie FPGA ne permet pas actuellement d'implémenter les opérations en virgule flottante avec un coût raisonnable en surface occupée. C'est pourquoi dans le cadre de cette étude, tous les calculs sont implémentés en virgule fixe. Ainsi, la détermination de la précision nécessaire pour chaque calcul a fait l'objet d'une étude préalable à l'aide d'une version logicielle *bit-true* effectuant au bit près les mêmes calculs en virgule fixe que l'architecture matérielle. Par exemple, pour le rétroprojection 2D, douze chiffres après la virgule sont suffisants pour coder le cosinus et le sinus comme l'illustre la figure 3.1.3. Pour tous les calculs, un compromis a été ainsi recherché entre le nombre de bits nécessaires pour coder les opérandes,(synonyme de surface occupée sur le FPGA) et la précision des résultats de chaque opération.

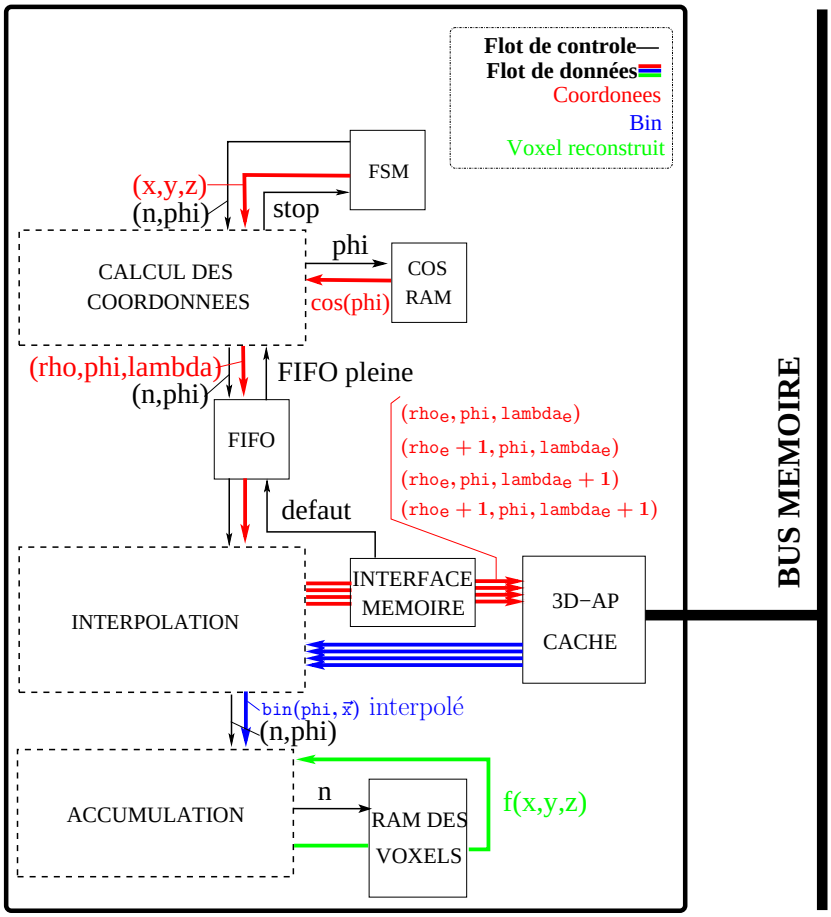


FIG. 3.9 – Pipeline de l'architecture 3P

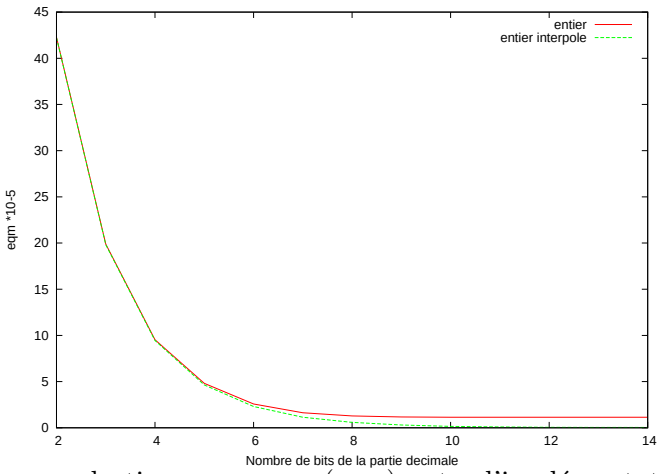


FIG. 3.10 – Erreur quadratique moyenne (eqm) entre l'implémentation en virgule fixe (avec et sans l'interpolation linéaire) et l'implémentation en virgule flottante (avec interpolation linéaire) pour la rétroprojection 2D en fonction du nombre de chiffres de la partie décimale utilisée pour coder $\boldsymbol{cos(\phi)}$ et $\boldsymbol{sin(\phi)}$

3.2 Préchargement mémoire : Utilisation du cache 3D-AP

Le cache n Dimensionnel Adaptatif et Prédictif (cache nD-AP) développé au sein du laboratoire Gipsa-lab par [Mancini 04] a été utilisé pour masquer la latence de la mémoire externe. Son architecture a été adaptée pour permettre un accès simultané aux quatre *bins* nécessaires pour le calcul de l'interpolation bilinéaire.

3.2.1 Principe général du cache nD-AP

Le cache nD-AP met en oeuvre une stratégie mémoire générique permettant le préchargement de toute séquence d'accès mémoire continue contenue dans un espace mémoire n dimensionnel. Le cache est situé sur la puce à proximité d'IPs de traitement de données nécessitant de nombreux accès en mémoire externe. Dans le cadre de cette étude, nous avons utilisé un cache 2D-AP pour la rétroprojection 2D et un cache 3D-AP pour la rétroprojection 3D. Comme illustrées sur la figure 3.11, les requêtes au cache par l'IP utilisateur (ici le pipeline de rétroprojection) sont faites en terme de coordonnées spatiales (en `phi`, `rho` et `lambda`). Si les données demandées sont en cache, ces requêtes sont servies immédiatement (en un cycle d'horloge). Sinon elles constituent un défaut et doivent attendre que le cache soit disponible pour traiter ce défaut.

Afin de prédire quelles seront les prochaines requêtes provenant de l'IP utilisateur, le cache effectue une analyse statistique des requêtes. Il se charge alors d'effectuer le transfert de ces données potentiellement utiles par l'IP utilisateur depuis la mémoire externe vers la mémoire cache sur la puce. Par ailleurs, ces transferts de données sont totalement masqués pour l'IP utilisateur. En effet, pendant que de nouvelles données sont chargées depuis la mémoire externe lors de la mise à jour du cache, les données communes entre l'actuelle et la prochaine zone mémoire mise en cache, restent entièrement disponibles. Les mises à jour d'un cache sont traitées de manière prioritaire par rapport aux traitements des défauts. Le traitement d'un défaut ne peut interrompre la mise à jour du cache.

3.2.2 Paramétrisation du mécanisme de prédiction du cache 3D-AP

Le fonctionnement détaillé du cache nD-AP est décrit dans le cas 2D en figure 3.13. Le mécanisme de prédiction du cache nD-AP doit être adapté au parcours mémoire particulier de l'IP utilisateur. Un jeu de cinq paramètres pour chaque dimension spatiale est utilisé dans le cadre de cette étude. Pour la rétroprojection 3D, la configuration des zones du cache (voir figure 3.12) requiert quinze paramètres au total pour les 3 dimensions $\boldsymbol{\phi}$, $\boldsymbol{\rho}$ et $\boldsymbol{\lambda}$. Ces paramètres sont les suivants :

- *Fréquence de coupure $\boldsymbol{f_c}$ et fréquence d'échantillonnage $\boldsymbol{f_{ech}}$ de l'analyseur statistique* : la coordonnée moyenne $(\boldsymbol{\phi_M}, \boldsymbol{\rho_M}, \boldsymbol{\lambda_M})$ des précédentes requêtes est calculée à l'aide d'un filtre IIR du premier ordre configuré par ces deux fréquences.
- *Taille de la zone de cache $\boldsymbol{t_{cache}}$* : cette zone mémoire est déclarée comme disponible à l'interface mémoire du pipeline de rétroprojection. Le cache nD-AP permet de déterminer dynamiquement la taille de la zone en cache via le calcul de la variance des coordonnées spatiales des requêtes. Toutefois, afin de limiter le nombre

de paramètres du cache à fixer, nous déterminons, dans le cadre de cette étude, statiquement la zone en cache via le paramètre $\boldsymbol{t_{cache}}$.
- *Taille de la zone de garde* $\boldsymbol{g_{cache}}$: à chaque fois que la coordonnée moyenne est en dehors de la zone de garde, cette zone est mise à jour par un déplacement selon une dimension de la zone de cache dans l'espace mémoire globale.
- *Vitesse de déplacement du cache* $\boldsymbol{v_{cache}}$: elle doit être mise en fonction de la vitesse de déplacement des accès mémoire dans chaque dimension spatiale.

3.2.3 Cache 3D-AP "mosaïque"

Le cache nD-AP a été adapté pour l'accès simultané aux 4 *bins* nécessaires à l'interpolation bilinéaire. La figure 3.14 illustre de manière simplifiée l'architecture du cache 3D-AP "mosaïque". Le contrôle du cache récupère des données 64 bits de la mémoire externe via le bus système. Ces données qui correspondent à 4 *bins* codés sur 16 bits, sont aiguillées vers les 4 bancs mémoire a, b, c et d selon la parité des coordonnées `rho` et `lambda` des *bins*. Lors de la demande des données par l'unité de rétroprojection au cache, quatre cas peuvent se produire selon la parité des coordonnées $\texttt{rho}_\texttt{e}$ et $\texttt{lambda}_\texttt{e}$ comme illustrés sur la figure 3.15. Le pipeline de rétroprojection possède un *switch* qui permet de réordonner les données selon la parité de $\texttt{rho}_\texttt{e}$ et $\texttt{lambda}_\texttt{e}$ en entrée des étages de calcul de l'interpolation bilinéaire.

3.3 Parallélisation : Mise en place d'une hiérarchie mémoire

Afin d'augmenter la puissance de calcul, plusieurs pipelines de rétroprojection sont utilisés en parallèle. Chacun des pipelines a pour tâche de reconstruire un bloc de voxels et possède un cache mémoire 3D-AP qui le nourrit en données. Mais l'exécution en parallèle de ces $\boldsymbol{n}$ pipelines de rétroprojection risque d'augmenter sensiblement le taux d'occupation du bus mémoire. C'est pourquoi une hiérarchie mémoire a été mise en place dans le cache 3D-AP comme illustrée sur la figure 4.20. Dans cette nouvelle architecture du cache 3D-AP, un cache racine est ajouté et a pour but de nourrir en données les caches feuilles liés aux unités de rétroprojection. Cette hiérarchie mémoire permet de réduire le nombre d'accès à la mémoire externe car de la même manière que des voxels voisins accèdent à des données de projection voisines, des blocs voisins accèdent à des données de projection voisines. Nous réutilisons ici une fois de plus la forte localité géométrique intrinsèque de l'algorithme de rétroprojection. Les pipelines de rétroprojection fonctionnent en phase : lorsque la FIFO d'un des pipelines est pleine, les autres pipelines sont arrêtés.

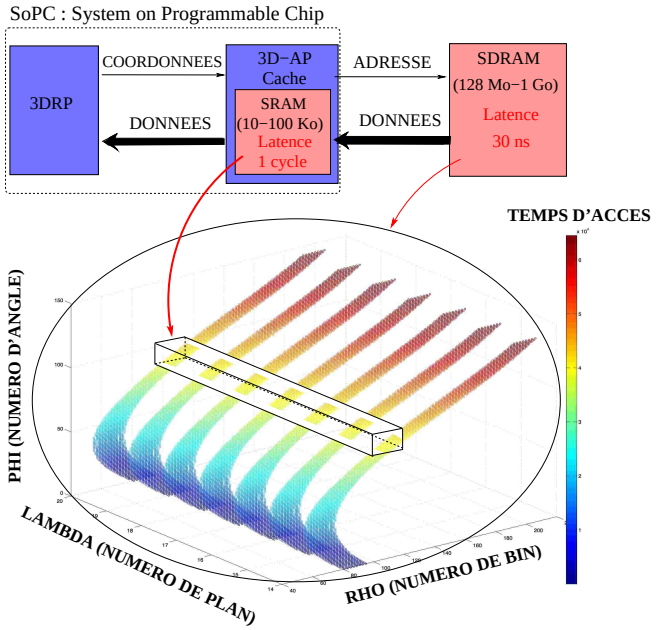


FIG. 3.11 – Principe du cache 3D-AP utilisé pour suivre la sinusoïde 3D des parcours mémoire de l'IP de rétroprojection 3D

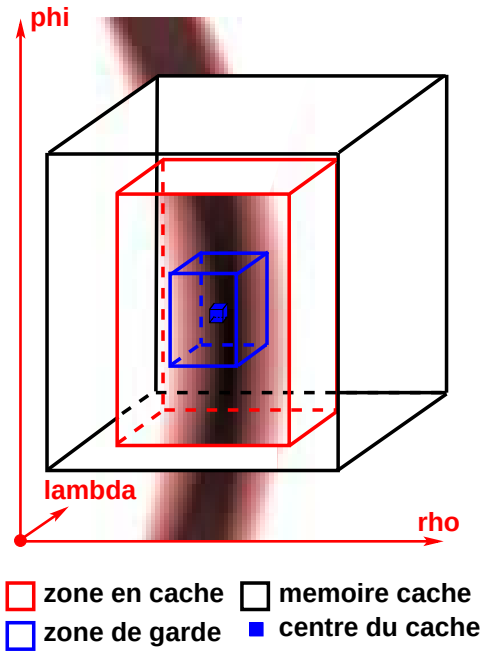
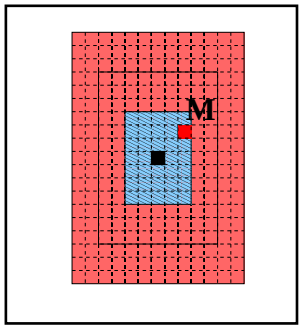
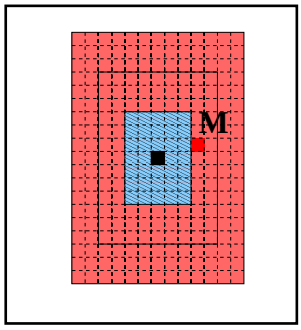


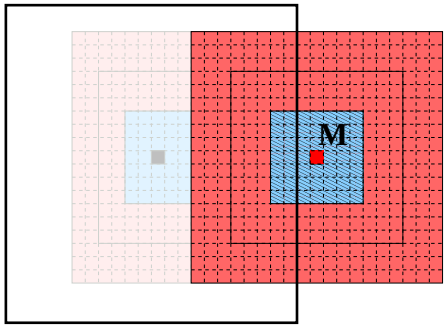
FIG. 3.12 – Zones du cache 3D-AP



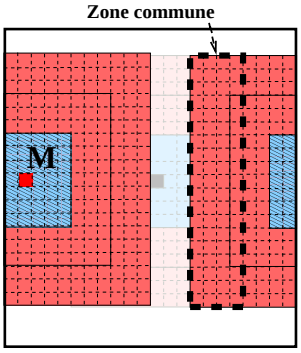
(a) La moyenne des coordonnées est calculée par l'analyseur statistique via un filtre IIR paramétré par les fréquences f_{ech} et f_c. Si cette moyenne M est dans la zone de garde de taille g_{cache}, aucune mise à jour du cache n'est effectuée.



(b) Lorsque la moyenne M des coordonnées est en dehors de la zone de garde, une mise à jour du cache est effectuée.



(c) Le déplacement de la zone de cache dans l'espace mémoire est effectué selon la vitesse de déplacement v_{cache} fixée au préalable par l'utilisateur. La taille de la nouvelle zone de cache est soit déterminée statiquement par le paramètre t_{cache}, soit déterminée dynamiquement à partir du calcul par l'analyseur statistique de la variance des coordonnées accédées.



(d) Un mécanisme en "tapis roulant" permet d'écrire la nouvelle zone dans la mémoire cache. Les données communes entre l'ancienne et la nouvelle zone de cache sont disponibles lors du transfert depuis la mémoire externe des nouvelles données.

FIG. 3.13 – Fonctionnement du cache 2D-AP

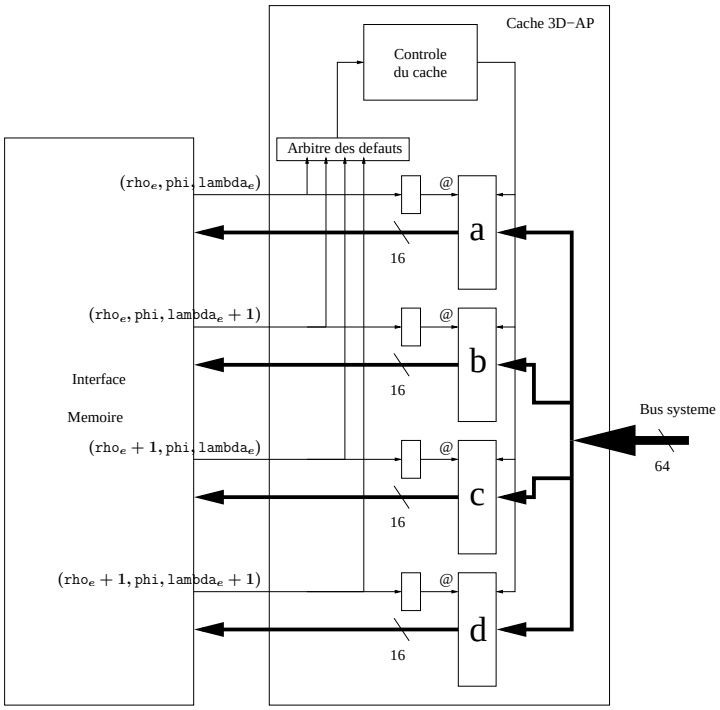


FIG. 3.14 – Architecture du cache 3D-AP adaptée à l'interpolation bilinéaire avec les quatre bancs mémoire internes a, b ,c et d

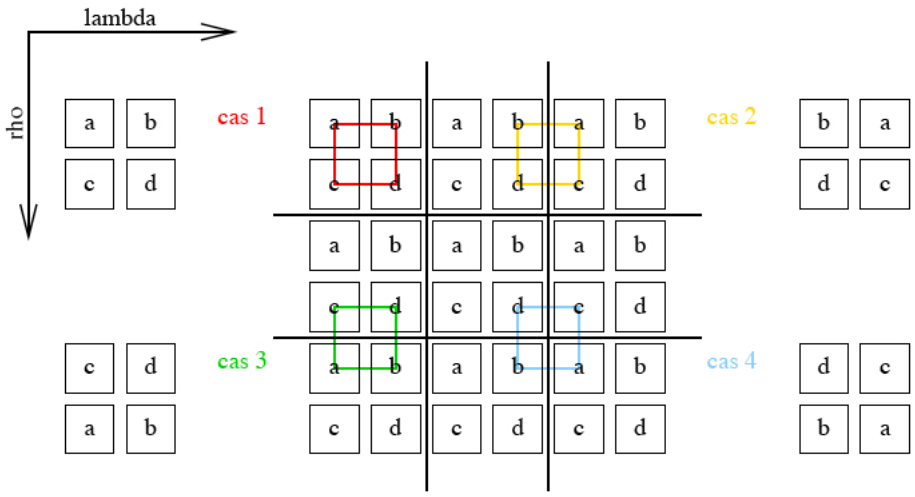
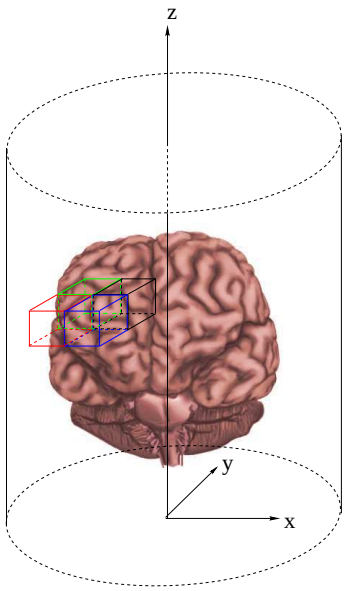
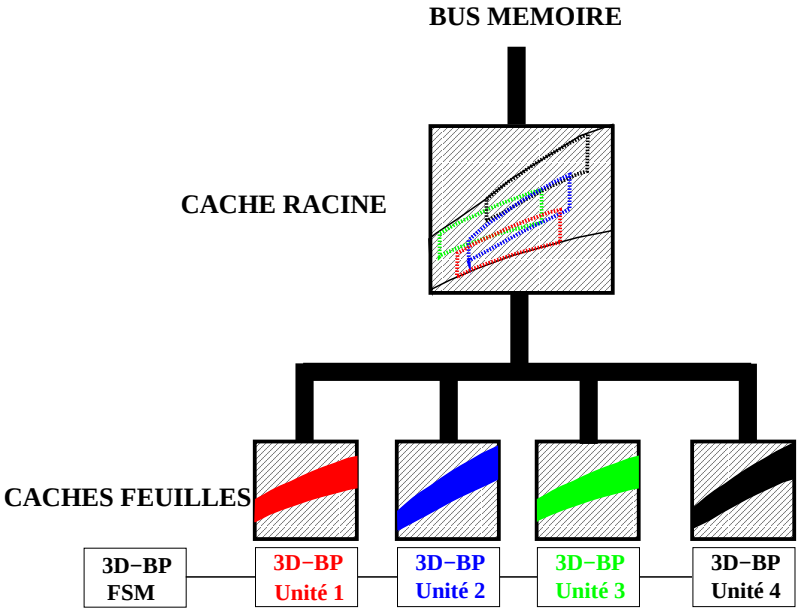


FIG. 3.15 – Les quatre types de requêtes de l'unité de rétroprojection selon le stockage des *bins* sur les bancs mémoire du cache 3D-AP



(a)



(b)

FIG. 3.16 – Architecture parallèle : (a) découpage du volume en blocs de voxel, et (b) architecture hiérarchique du cache 3D-AP

4 Conclusion

Nous avons dans ce chapitre proposé une architecture matérielle destinée à être implémentée sur un SoPC. La meilleure adéquation possible entre l'algorithme, l'architecture et le stockage des données a été recherchée.

Dans un premier temps, puisque l'algorithme de rétroprojection se résume à des boucles imbriquées de calcul sans dépendance forte de données, une architecture en pipeline a été tout naturellement choisie. Dans un second temps, afin de ne pas pénaliser les pipelines de rétroprojection par l'important temps de latence des mémoires externes de type SDRAM, une stratégie originale d'accès mémoire a été mise en place. Cette stratégie mémoire s'appuie tout d'abord sur un réordonancement des boucles de calcul. En effet la rétroprojection du volume se fait en bloc de voxels afin de bénéficier au maximum de la localité géométrique des données de projection. Ainsi un équilibre est obtenu entre l'important débit de calcul atteint par les n pipelines de rétroprojection (n cycles/Op.), et le débit mémoire existant entre l'unité de rétroprojection et le cache mémoire. Ce débit mémoire est significativement amélioré grâce à l'important taux de réutilisation des données mises en cache obtenu avec la reconstruction en bloc. Cette stratégie mémoire s'appuie également sur le suivi du parcours mémoire dessiné lors de la rétroprojection d'un bloc par l'analyseur statistique du cache 3D-AP. Un stockage mémoire approprié en segment permet de faciliter le suivi du parcours mémoire qui correspond alors à une sinusoïde 3D. Au final, grâce à l'équilibre atteint entre débit de calcul et débit mémoire et grâce au suivi de la sinusoïde 3D par le cache 3D-AP, le temps d'accès à la mémoire externe sera totalement masqué pour l'unité de rétroprojection.

Dans le chapitre suivant, nous présentons les performances atteintes par notre architecture 3P. Nous évaluerons avec une attention particulière la pertinence de l'utilisation du cache 3D-AP pour accélérer la rétroprojection 3D.

Chapitre 4

Performances de l'architecture 3P

Les trois principes qui ont guidé la conception de l'architecture 3P (architecture en Pipeline, Préchargement mémoire et Parallélisation), ont pour objectif de tirer profit au maximum des ressources matérielles des cartes de type SoPC. Dans ce chapitre, nous présentons les performances atteintes par notre architecture 3P pour la rétroprojection TEP 3D. Après une description de l'implémentation sur une plateforme SoPC, nous présenterons et analyserons la qualité de reconstruction, le comportement du cache 3D-AP et l'efficacité de reconstruction.

1 Implémentation sur une plateforme SoPC

Nous décrivons ici tout d'abord la plateforme SoPC utilisée avec les modules annexes autour de l'architecture 3P, puis l'occupation des ressources matérielles des Virtex par notre architecture 3P.

1.1 Plateforme SoPC utilisée

L'architecture 3P a été implémentée sur une plateforme SoPC disposant d'une interface PCI avec une station PC hôte afin de permettre les échanges de données avec le système sur la carte (voir figure 4.1). Cette dernière contient entre autres : un Virtex 2 Pro de Xilinx (VP30), deux mémoires externes (une SDRAM de 32 Mo et une SRAM de 2 Mo) et un bridge PCI, réalisé sur un FPGA Spartan. La transmission de données entre le PC et la carte via la mémoire SRAM a nécessité la mise en place d'un système de synchronisation. Le SoPC (Virtex 2 Pro VP30) est constitué d'un processeur PPC et d'une matrice FPGA avec plus de 30 000 portes logiques équivalentes. Le processeur PPC est en charge de la synchronisation des calculs et des communications avec l'architecture 3P implémentée sur la matrice FPGA.

En plus des éléments incontournables de l'architecture d'un système (bus PLB et OPB, contrôleurs mémoire...), est implémenté sur la matrice FPGA un module d'échange de données entre l'architecture 3P et le processeur PPC. C'est via ce module que les paramètres de configuration de l'architecture 3P (taille du bloc, paramètres du cache,

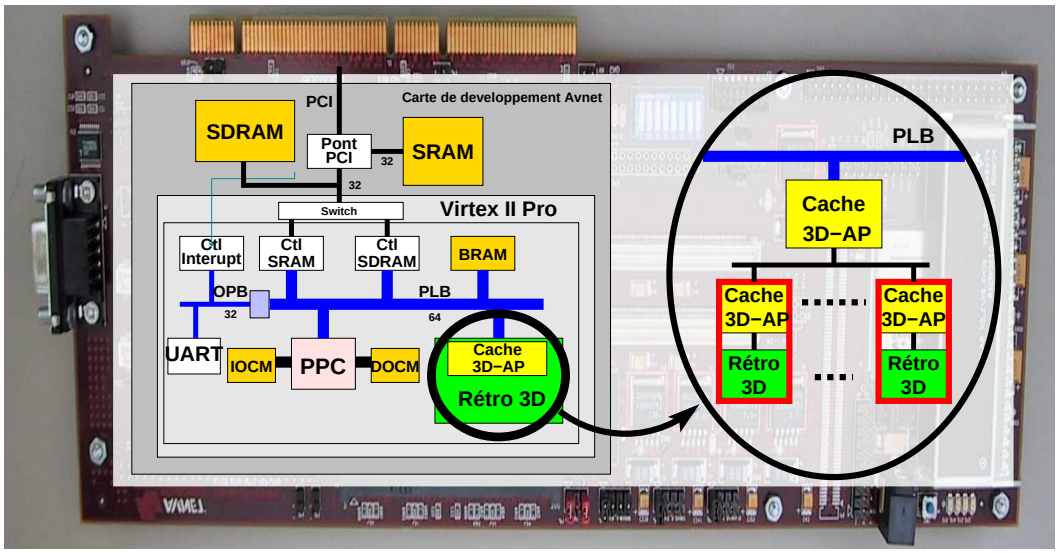


FIG. 4.1 – Carte de développement Avnet

constantes de calcul...), les valeurs des voxels reconstruits et les signaux de synchronisation entre le PPC et l'architecture 3P sont échangés. Par ailleurs, une interface PLB entre l'architecture 3P et le bus PLB permet au cache d'accéder directement et efficacement à la mémoire externe. Au sein de cette interface, une hiérarchisation des données est faite en lignes, plans et pages mémoire en phase avec le fonctionnement propre du cache 3D-AP.

1.2 Ressources matérielles utilisées

Les ressources matérielles utilisées par l'architecture 3P pour une taille de bloc de $\mathbf{16^2 \cdot 3}$ voxels sont présentées dans le tableau **4.1**. L'architecture permet une fréquence de fonctionnement de 35 MHz sur un Virtex 2 Pro. Le contrôle de la rétroprojection et le contrôle du cache racine étant partagés entre les $\boldsymbol{n}$ unités de l'architecture 3P, le coût additionnel d'une unité est de seulement 800 slices et de 12 multiplieurs. Les tailles des caches mémoire feuille et racine sont respectivement de 2 Ko et 18 Ko. Ainsi jusqu'à 9 unités de rétroprojection peuvent être mise sur un virtex 2 Pro VP30. Aux slices utilisés pour la rétroprojection 3D et le cache 3D-AP, il faut ajouter ceux de l'interface PLB (environ 10 % pour 9 unités) et du système autour (bus mémoire, contrôleur mémoire...).

Un Virtex 4 (V4FX60 ou V4FX100) permettrait de mettre jusqu'à 13 unités. Dans ce cas c'est le nombre de multiplieurs qui limite le nombre d'unités et non le nombre de slices comme pour le Virtex 2 Pro. Dans la suite, nous avons utilisé 1, 4 ($\mathbf{2 \cdot 2 \cdot 1}$ blocs de voxels) ou 8 unités ($\mathbf{2 \cdot 2 \cdot 2}$ blocs de voxels). Nous avons préféré reconstruire des macros blocs constitués de $\mathbf{2 \cdot 2 \cdot 2}$ blocs plutôt que de $\mathbf{3 \cdot 3 \cdot 1}$ blocs afin d'avoir une hauteur de macro-bloc plus importante. Cela permet tout d'abord de bénéficier d'un plus grand taux de réutilisation étant donné le plan supplémentaire que nécessite l'interpolation linéaire selon la dimension $\boldsymbol{\lambda}$ (voir chapitre 3). Par ailleurs une hauteur de macro-bloc

importante implique une taille de cache plus importante selon la dimension axiale $\boldsymbol{\lambda}$. Or le cache est plus difficile à paramétrer lorsqu'une dimension de la zone de cache est très petite (inférieure à 3).

	1 unité	**4 unités**	**9 unités**
Rétroprojection 3D			
slices CLB	573 (**4.2%**)	1817 (**13.3%**)	3924 (**28.6%**)
Multiplieurs	12 (**9%**)	48 (**35%**)	108 (**79%**)
Cache 3D-AP			
slices CLB	672 (**4.9%**)	2830 (**20.6%**)	4804 (**35.1%**)
RAMs	2 Ko (**0.6%**)	24 Ko (**7.8%**)	36 Ko (**11.7%**)
Rétroprojection 3D + Cache 3D-AP			
slices CLB	1245 (**9.1%**)	4637 (**32.9%**)	8728 (**63.7%**)

TAB. 4.1 – Ressources matérielles utilisées par l'architecture 3P de rétroprojection 3D pour une fréquence de fonctionnement de 35 Mhz. Entre parenthèses est indiqué le pourcentage d'occupation des ressources du Virtex 2 Pro VP30

2 Qualité de reconstruction

L'architecture 3PA-PET correspond à une version en arithmétique à virgule fixe de l'algorithme de rétroprojection Voxel driven avec Interpolation Bilinéaire (VIB). De plus, les données du sinogramme et du volume sont codées en entier sur respectivement 16 bits et 32 bits, alors que ces données sont à l'origine des réels en virgule flottante (le sinogramme après filtrage ne correspond plus au simple comptage d'événements sur chaque paire de détecteurs).

Afin d'évaluer la qualité de reconstruction de notre implémentation matérielle sur FPGA de la rétroprojection 3D, nous avons utilisé une version logicielle "bit-true" de notre architecture, qui effectue au bit près les mêmes opérations en virgule fixe que l'architecture 3PA-PET. Nous présentons tout d'abord les données et critères utilisés puis nous comparons cette rétroprojection en virgule fixe avec la rétroprojection en virgule flottante.

2.1 Données utilisées

Les données utilisées proviennent du logiciel de reconstruction STIR développé par [Thielemans 06] dans la suite du projet PARAPET[1]. STIR est un logiciel Open Source multiplateforme permettant la reconstruction 2D et 3D à l'aide d'algorithmes analytiques (rétroprojection filtrée) et itératifs (ML-EM). Il est utilisé pour la recherche mais également en clinique. Nous avons tout d'abord généré des volumes numériques, appelés phantoms, à l'aide de STIR comme par exemple le phantom de la figure 4.2, ou bien à l'aide de Matlab comme le phantom Shepp Logan de la figure 4.3. Nous avons ensuite simulé l'acquisition des données en utilisant le projecteur de STIR. Enfin, nous avons effectué la correction d'arc, les filtrages (filtre rampe en 2D et de Colsher en 3D) et la complétion des sinogrammes pour l'algorithme 3D-RP (estimation du volume par rétroprojection 2D puis projection du volume en utilisant des anneaux virtuels du scanner). Ces corrections ont été effectuées à l'aide d'exécutables créés à partir du code C++ de STIR .

Les jeux de données utilisés (volume 3D et sinogramme 4D) ainsi générés correspondent à l'acquisition d'un scanner HR+ (32 anneaux avec un span de 9 soit 5 segments). Le volume a une taille de $\mathbf{128^2 \cdot 63}$. Pour l'algorithme 3D-RP, les 5 segments sont constitués de sinogrammes 2D correspondant d'une part au processus d'acquisition et d'autre part à la complétion des données. Le sinogramme 4D possède ainsi 65 sinogrammes 2D (63 acquis + 2 complétés) pour le segment 0, 77 sinogrammes 2D (53 acquis + 24 complétés) pour les segments +1 et -1, et 91 sinogrammes 2D (35 acquis + 56 complétés) pour les segments +2 et -2. Les sinogrammes 2D sont de taille $\mathbf{269 \cdot 96}$ ce qui correspond à 96 angles de projection et 269 bins par angle ϕ (288 avant correction d'arc). Le nombre d'angles de projection qui est de 144 pour le format standard a du être réduit à 96 afin de pouvoir stocker le sinogramme dans les 32 Mo de la SDRAM de la carte de développement Avnet.

Par ailleurs, afin de pouvoir effectuer en un temps acceptable une reconstruction 3D à l'aide du simulateur VHDL de Modelsim, nous avons également généré avec STIR un jeux de données de taille réduite (quelques Ko). Ces mini volumes ont pour taille $\mathbf{9^2 \cdot 17}$ ou $\mathbf{16^2 \cdot 17}$. De plus, un format de scanner sur mesure a été défini à l'aide de STIR. Le mini-scanner ainsi défini, possède 9 anneaux, 5 segments et 18 angles de projection.

[1]Le projet PARAPET était un projet Européen dont l'objectif était d'améliorer les techniques de reconstruction volumique en tomographie par émission de positons (TEP), notamment en développant les approches itératives implémentées sur des architectures parallèles. STIR est basé sur les bibliothèques logicielles PARAPET développé dans le cadre du projet PARAPET.

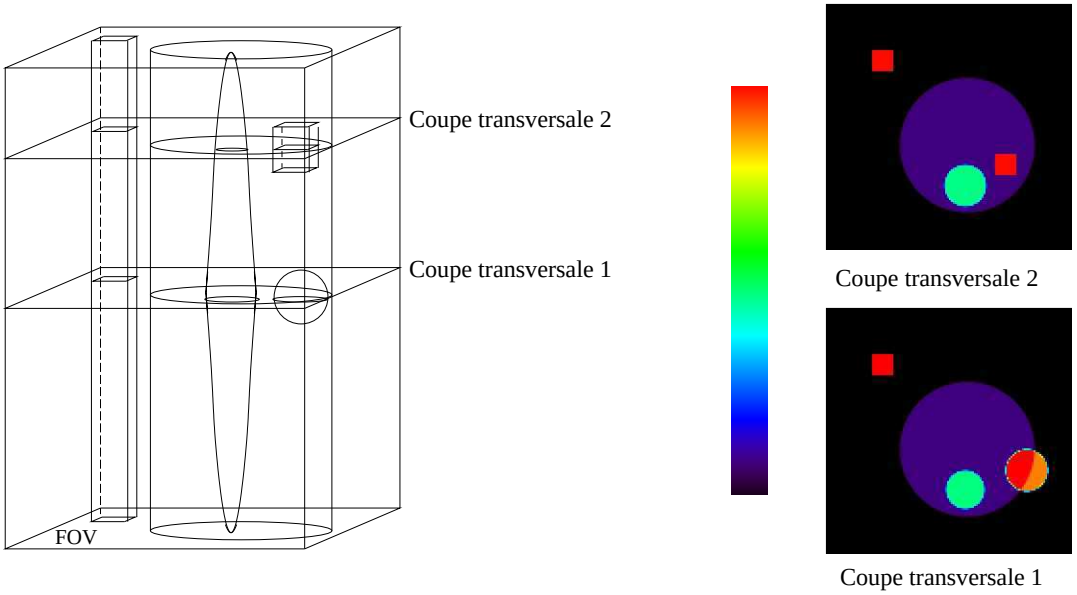


FIG. 4.2 – Le phantom001, phantom créé avec STIR, est représenté dans le champ de vu du scanner (FOV) par un parallélépipède de section carré. Le fantôme est composé d'un gros cylindre contenant une ellipsoïde, un cube et une sphère. Une barre de section carrée est située à l'extérieur du cylindre.

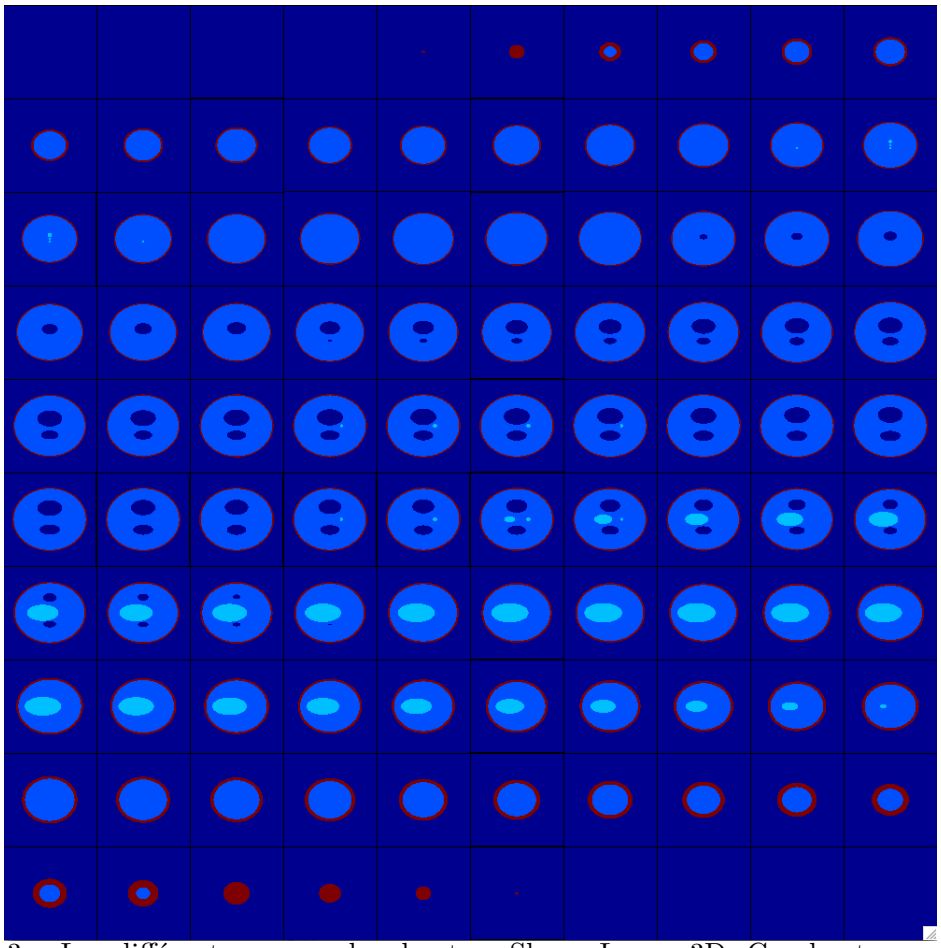


FIG. 4.3 – Les différents coupes du phantom Shepp Logan 3D. Ce phantom modélise grossièrement un cerveau et constitue un volume de référence en tomographie pour mesurer la qualité de reconstruction. Les hautes fréquences constituées par les écarts de niveau de gris brutaux entre l'extérieur, la fine enveloppe correspondant aux os du crâne et l'intérieur de cerveau, sont particulièrement difficiles à reconstruire

2.2 Critères d'évaluation

Pour évaluer la qualité de reconstruction, nous avons utilisé outre le simple critère visuel, deux métriques analytiques : l'Erreur Absolue Moyenne relative ($\boldsymbol{EAM_r}$) et le Pic du Ratio Signal sur Bruit (PRSB). Ces deux mesures comparent un volume $\boldsymbol{V}$ avec un volume de référence qui peut être le volume original ou bien un volume reconstruit avec une implémentation de référence. L'$\boldsymbol{EAM_r}$ calcule la moyenne de l'erreur relative entre les volumes $\boldsymbol{V}$ et $\boldsymbol{V_{ref}}$ (voir équation 4.1). Pour les voxels ayant des intensités inférieures à 1, cette erreur relative est simplement égale à la différence entre l'intensité des deux volumes.

$$\begin{aligned} \boldsymbol{EAM_r} &= \frac{\mathbf{1}}{\boldsymbol{i_{max}}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{i_{max}}} \boldsymbol{Err(i)} \\ \text{avec} & \\ \boldsymbol{Err(i)} &= \begin{cases} \dfrac{|\boldsymbol{V(i)} - \boldsymbol{V_{ref}(i)}|}{|\boldsymbol{V_{ref}(i)}|} & \text{si } |\boldsymbol{V_{ref}(i)} > \mathbf{1}| \\ |\boldsymbol{V(i)} - \boldsymbol{V_{ref}(i)}| & \text{sinon} \end{cases} \end{aligned} \tag{4.1}$$

Le PRSB correspond au ratio entre le maximum du volume de référence et la Racine de l'Erreur Quadratique Moyenne (REQM) entre les volumes $\boldsymbol{V}$ et $\boldsymbol{V_{ref}}$ (voir équation 4.2). Il donne une indication relative sur le niveau d'erreur (bruit) par rapport aux niveaux d'intensité du volume de référence.

$$\begin{aligned} \boldsymbol{PRSB} &= \mathbf{20} \cdot \boldsymbol{log_{10}} \frac{\boldsymbol{max(V_{ref})}}{\boldsymbol{REQM}} \\ \text{avec} & \\ \boldsymbol{REQM} &= \sqrt{\frac{\mathbf{1}}{\boldsymbol{i_{max}}} \cdot \sum_{\boldsymbol{i=1}}^{\boldsymbol{i_{max}}} (\boldsymbol{V(i)} - \boldsymbol{V_{ref}(i)})^{\mathbf{2}}} \end{aligned} \tag{4.2}$$

2.3 Comparaison entre les reconstructions en virgule fixe et en virgule flottante

Nous avons comparé la version logicielle en virgule fixe de la rétroprojection VIB avec les versions logicielles en virgule flottante du rétroprojecteur VIB et du rétroprojecteur de STIR (ray-driven avec interpolation bi-linéaire). Nous avons paramétré la précision de chaque calcul en virgule fixe. Les résultats présentés sont ceux obtenus avec une précision correspondant à un bon compromis entre le nombre de bits pour coder les nombres (synonyme de surface du FPGA occupé) avec la qualité de reconstruction. Les constantes intervenant lors d'opérations de multiplication ont été codées sur 18 bits (taille des multiplieurs des Virtex) et certaines constantes comme le cosinus ont fait l'objet d'études plus poussées afin de les coder de manière plus compacte (voir section 3.1.3 du chapitre 3). Nous avons tout d'abord validé l'équivalence entre la description VHDL de

notre architecture 3P et son implémentation logicielle "bit true" avec la reconstruction d'un petit volume $\mathbf{9^2 \cdot 17}$. Les volumes reconstruits avec ces deux implémentations sont identiques (voir figure 4.4).

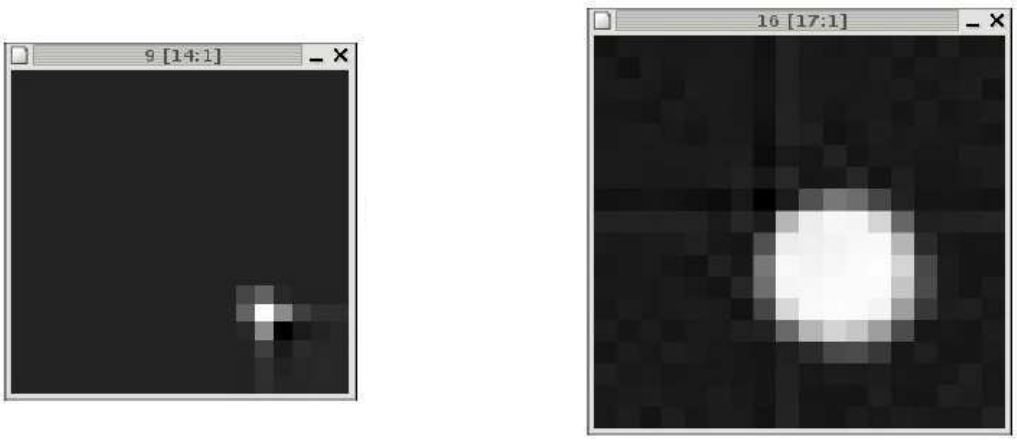
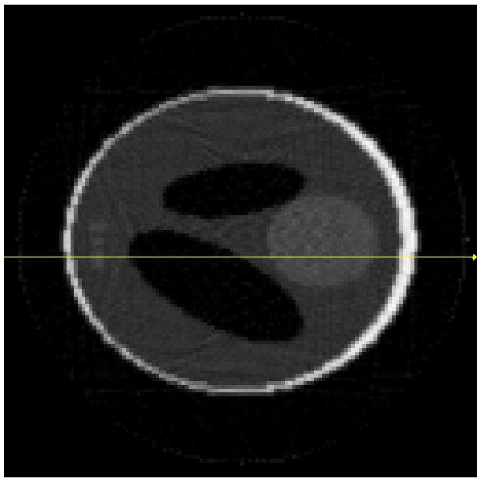


FIG. 4.4 – Mini volumes reconstruits en virgule fixe de manière équivalente avec l'implémentation "bit true" et la description VHDL (sous Modelsim)

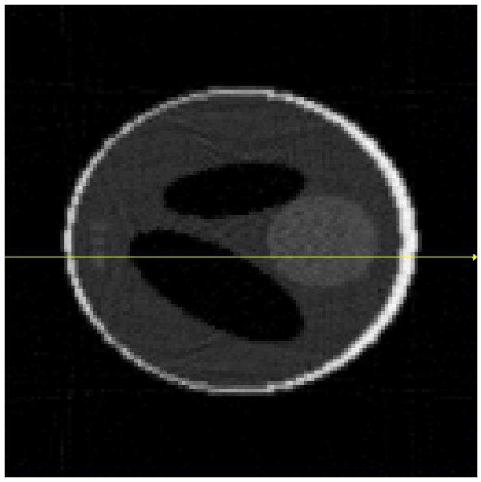
Sur la figure 4.5, nous pouvons constater que visuellement la reconstruction en virgule fixe est très proche de la reconstruction en virgule flottante. Les différences visibles entre les deux reconstruction sur le profil d'une ligne de la coupe 2D reconstruite, sont en partie dues aux différences entre les algorithmes de rétroprojection voxel-driven (VIB) et ray-driven (STIR). Nous avons calculé pour la rétroprojection de STIR et celles VIB en virgule fixe (VIB-fix) et en virgule flottante (VIB-float), le $\boldsymbol{EMA_r}$ et le $\boldsymbol{PRSB}$ en prenant le phantom original comme volume de référence (voir tableau 4.2). Toutes ces implémentations ont un $\boldsymbol{EMA_r}$ au alentour de $\mathbf{3,9\%}$ et un $\boldsymbol{PRSB}$ de 10,5 dB qui correspondent avant tout à l'erreur intrinsèque de l'algorithme de rétroprojection. Lorsque les données sont converties en entier sur 16 bits, l'augmentation de l'$\boldsymbol{EMA_r}$ de 0,1 % reste négligeable au vu de l'erreur de méthode.

Volumes comparés (Quantification de calcul)	**Type des données** (Quantification d'entrée)	$\mathrm{EAM_r}$	PRSB
STIR / original	float	**3.89 %**	10.5 dB
VIB-Float / original	float	**3.88 %**	10.5 dB
VIB-Fix / original	float	**3.88 %**	10.5 dB
VIB-Float / original	int16	**3.97 %**	10.5 dB
VIB-Fix / original	int16	**3.97 %**	10.5 dB

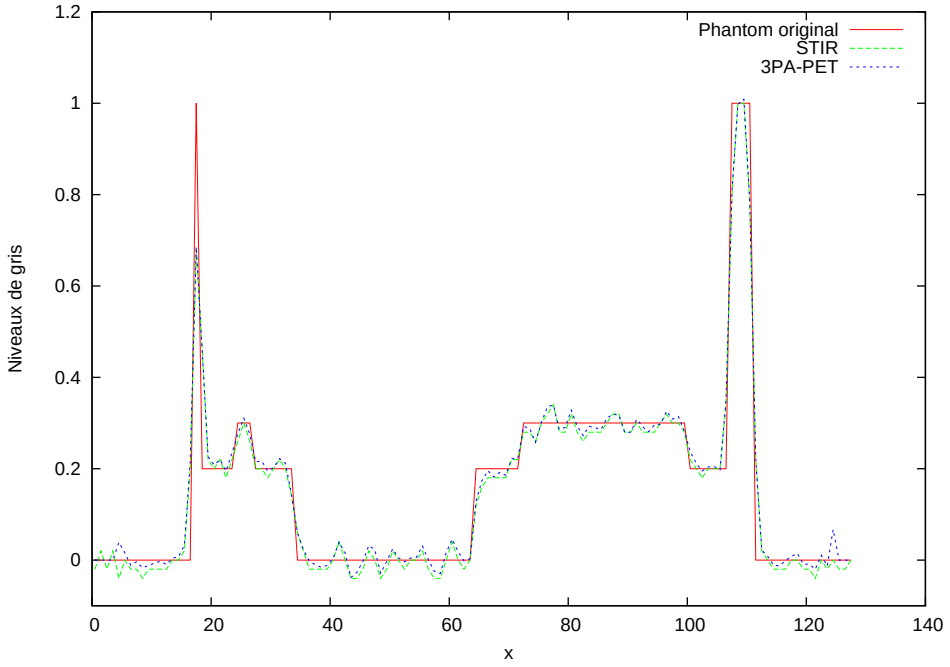
TAB. 4.2 – Qualité de reconstruction du phantom Shepp Logan



(a) Coupe 2D reconstruite par STIR



(b) Coupe 2D reconstruite par la version *software bit-true* de l'architecture 3P



(c) Comparaison du profil de la ligne tracée sur les coupes 2D (a) et (b)

FIG. 4.5 – Coupe du phantom Shepp Logan reconstruit par le rétroprojecteur 3D de STIR (a) et par la version logicielle "bit-true" de l'architecture 3P (b). Le profil de la ligne tracée en (a) et (b) des coupes originales, et reconstruites par STIR et 3P, est présenté en (c).

Afin d'évaluer plus précisément la perte de précision due à l'utilisation d'une arithmétique en virgule fixe, nous avons comparé pour le même algorithme VIB de rétroprojection, le volume reconstruit en virgule fixe avec le volume reconstruit en virgule flottante (voir figure 4.3). Lorsque les données utilisées sont des réels avec virgule flottante, l'écart correspond à un $\boldsymbol{EAM_r}$ de seulement $\mathbf{0,13\%}$ et un $\boldsymbol{PRSB}$ de 26.2 dB. Si nous utilisons des données de types différents (entier pour la version VIB-fix et réel pour la version VIB-float), l'écart correspond alors à un $\boldsymbol{EAM_r}$ de $\mathbf{1,1\%}$ et un $\boldsymbol{PRSB}$ de 19 dB. Nous pouvons conclure que l'erreur due à l'emploi d'une arithmétique en virgule fixe est quasi négligeable ($\mathbf{0,1\%}$) et que la conversion des données au format entier entraîne une légère perte de précision ($\mathbf{1\%}$).

Volumes comparés Quantification de calcul	**type des données** (Quantification d'entrée)	$\mathrm{EAM_r}$	PRSB
STIR / VIB-float	float	**0.35 %**	21.5 dB
VIB-fix / VIB-float	float	**0.13 %**	26.2 dB
VIB-fix / VIB-float	int16	**0.13 %**	23.0 dB
VIB-fix / VIB-float	int16/float	**1.1 %**	19.0 dB

TAB. 4.3 – Comparaison des différentes reconstructions du phantom Shepp Logan

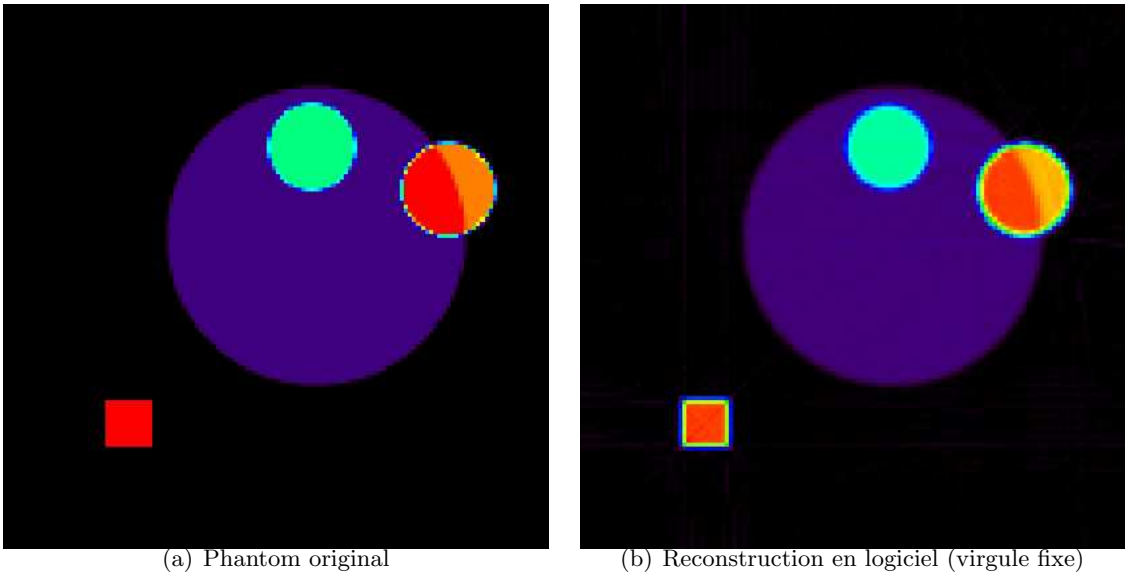
2.4 Volumes reconstruits sur la carte

Les volumes reconstruits sur la carte sont présentés sur la figure 4.6. Contrairement à ce que l'on avait observé en simulation sous Modelsim, des différences apparaissent entre les reconstructions en *software* en virgule fixe et en *hardware*.

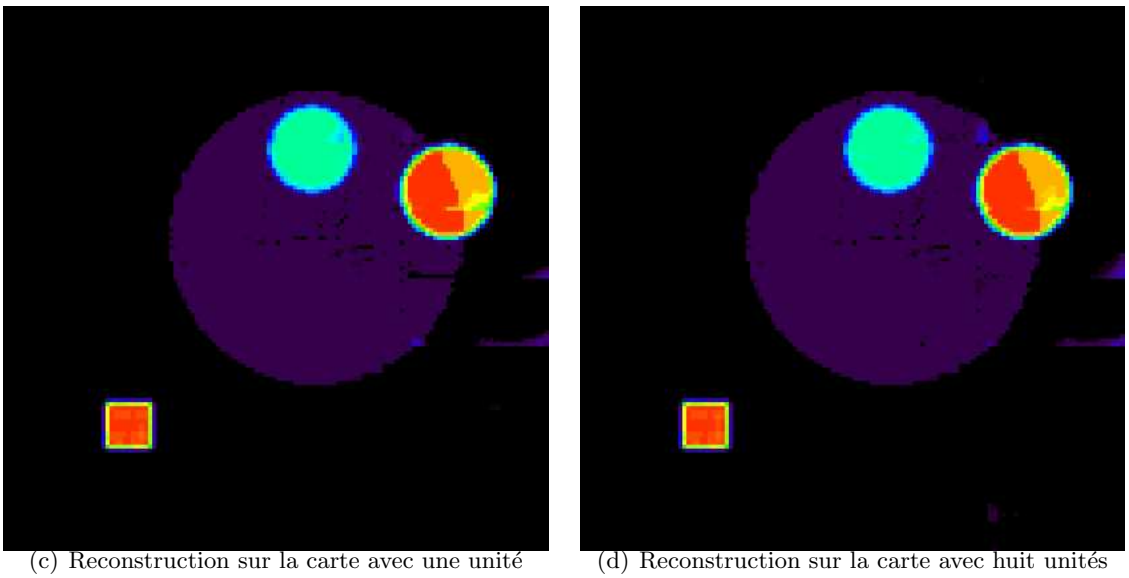
Tout d'abord, nous avons constaté que la densité du volume (valeur des voxels) est divisée par 2. Le volume original a une densité comprise 0 at 100, alors que le volume reconstruit à une intensité comprises entre 0 et 50. Cela doit provenir d'un mauvais placement de la virgule lors de calcul en virgule fixe (vraisemblablement lors de la multiplication par le jacobien). Par ailleurs, les zones reconstruites ne sont plus uniformes et on peut observer des pixels extérieurs au cylindre ayant une intensité non nulle. Cela s'explique par la plus petite dynamique du volume reconstruit (du à la division par 2) et également semble-t-il par certains bugs de la mémoire externe que nous avons observés sur la carte (un conflit entre les mémoires SRAM et SDRAM a été observé). Nous avons également observé des différences entre les reconstructions sur la carte à 20 Mhz et à 33 Mhz, qui pourrait s'expliquer par la sensibilité de la puce à l'élévation de température.

Toutefois, nous pouvons observer que les formes du volume sont correctes pour l'architecture avec une unité et pour l'architecture parallèle avec huit unités. Cela prouve que sur la carte le calcul des coordonnées est correct. Nous pouvons ainsi être assurés que les accès mémoire correspondent bien à ceux rétroprojection 3DRP et nous pouvons donc analyser du comportement du cache 3D-AP sur la carte. L'obtention sur la carte d'un volume qui correspondrait à la reconstruction logiciel en virgule fixe (comme nous

l'avons observé en simulation) demanderait d'effectuer de nouveaux essais (utilisation du réglage correct des constantes en virgule fixe, recherche des bugs mémoire...). Les adresses accédées en mémoire externe étant correctes, nous avons concentré nos efforts sur l'observation du cache 3D-AP et sur l'efficacité de reconstruction.



(a) Phantom original (b) Reconstruction en logiciel (virgule fixe)



(c) Reconstruction sur la carte avec une unité (d) Reconstruction sur la carte avec huit unités

FIG. 4.6 – Phantoms numériques originaux, reconstruits en *software* et en *hardware* sur la carte (rétroprojection 3DRP avec les 5 segments de données HR+)

3 Performances du cache 3D-AP

Nous étudions dans cette section le comportement du cache 3D-AP lors de la rétroprojection de blocs de $\mathbf{16^2 \cdot 3}$ voxels. Nous présentons tout d'abord la manière dont nous avons paramétré le cache 3D-AP, puis le comportement du cache en simulation et sur la carte.

3.1 Paramétrisation du cache

Le cache 3D-AP nécessite le réglage de 5 paramètres pour chaque dimension (voir chapitre 3). Il faut ainsi pour le cache 3D-AP hiérarchique, un jeu de paramètres pour le cache racine ($\mathbf{5 \cdot 3 = 15}$ paramètres) et un autre pour le cache feuille ($\mathbf{5 \cdot 3 = 15}$ paramètres). Nous avons utilisé un seul et même jeu de paramètres pour tous les segments et tous les blocs. La recherche des paramètres adéquats à la rétroprojection 3D, s'est effectuée en deux phases : la première en simulation et la deuxième sur la carte.

Le temps important nécessité par la simulation ne permet d'observer le comportement du cache seulement pour quelques blocs de voxels et pour quelques jeux de paramètres. Il faut donc passer sur la carte pour observer le comportement du cache sur l'ensemble des blocs du volume et des segments du sinogramme 4D. De plus, la rapidité d'exécution des calculs sur la carte permet d'effectuer une exploration plus importante de l'espace des paramètres et ainsi d'affiner les premiers jeux de paramètres du cache obtenus en simulation. Par ailleurs, les mesures provenant des simulations donnent des indications partielles sur le comportement dynamique du cache. C'est pourquoi, nous avons mesuré directement sur la carte plusieurs métriques afin de mieux appréhender le comportement du cache 3D-AP lors de la rétroprojection 3D.

Afin d'évaluer les performances de l'architecture 3PA-PET en fonction des caractéristiques des ressources mémoire, nous avons utilisé en simulation et sur la carte, un simulateur de bus mémoire dont la latence $\boldsymbol{l_{mem}}$ et le débit $\boldsymbol{d_{mem}}$ sont paramétrables. Ce bus délivre une ligne mémoire contenant $\boldsymbol{N_{mot}(ligne)}$ en un temps $\boldsymbol{t_{mem}}$ dépendant de la taille de la ligne mémoire accédée (voir équation 4.3) :

$$\boldsymbol{t_{mem} = l_{mem} + \frac{N_{mot}(ligne) - 1}{d_{mem}}} \tag{4.3}$$

3.2 Comportement du cache en simulation

En simulation sous modelsim, nous avons comparé les requêtes des unités de rétroprojection et le placement des zones du cache lors de la reconstruction de quelques blocs de voxels (plus ou moins excentrés dans le champ de vue du scanner) générant des parcours mémoire représentatifs (sinusoïdes plus ou moins courbes).

3.2.1 Suivi des coordonnées rho, lambda et phi

Les suivis des coordonnées rho, lambda et phi sont ainsi représentés sur les figures 4.7, 4.8 et 4.9 pour le cache feuille et sur les figures 4.10, 4.11 et 4.12 pour le cache racine.

Sur ces courbes, nous pouvons observer que la zone de cache englobe la quasi totalité des accès mémoire. Sur la figure 4.7, on peut observer que les accès mémoire selon rho dessinent des lignes brisées. A chaque changement de ligne du bloc (passage de y à y+1), on observe un saut brusque de la coordonnée rho.

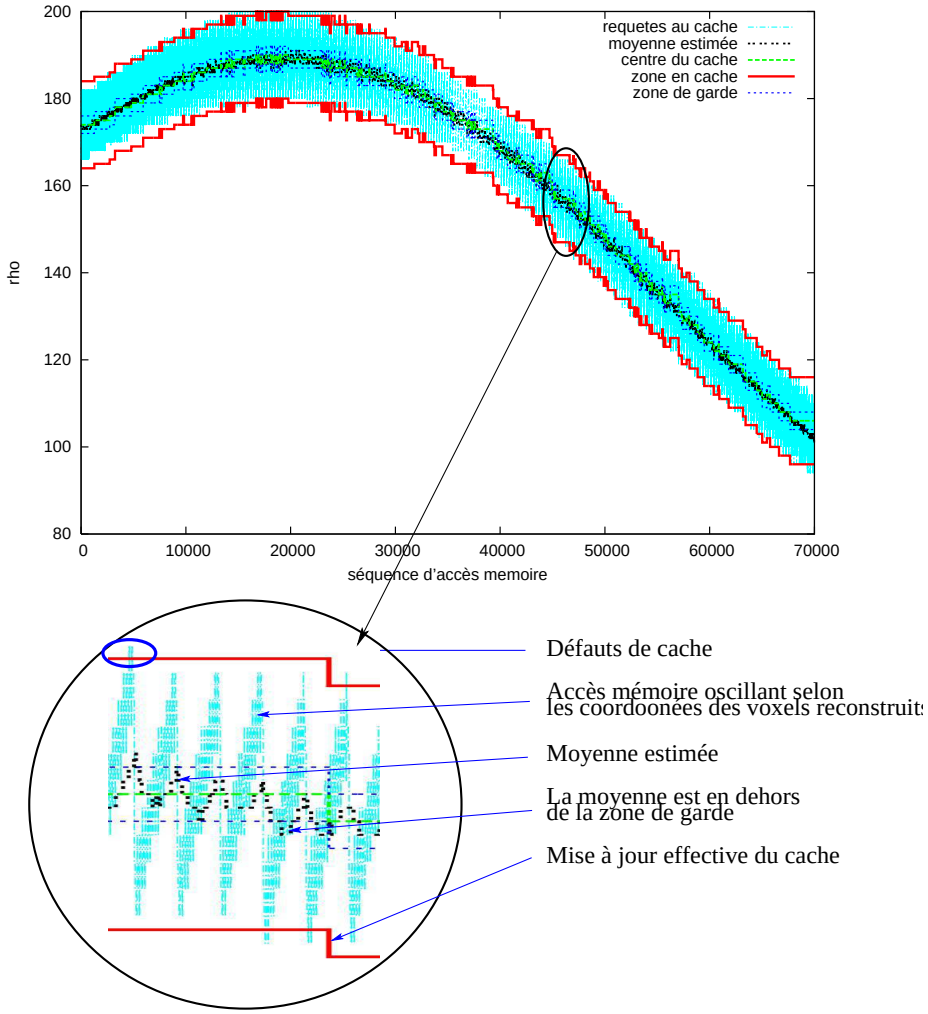


FIG. 4.7 – Suivi de la coordonnée rho par le cache feuille (8 unités, segment -2, $\boldsymbol{l_{mem}}$=5 cycles, $\boldsymbol{d_{mem}}$=8 octets/cycle).

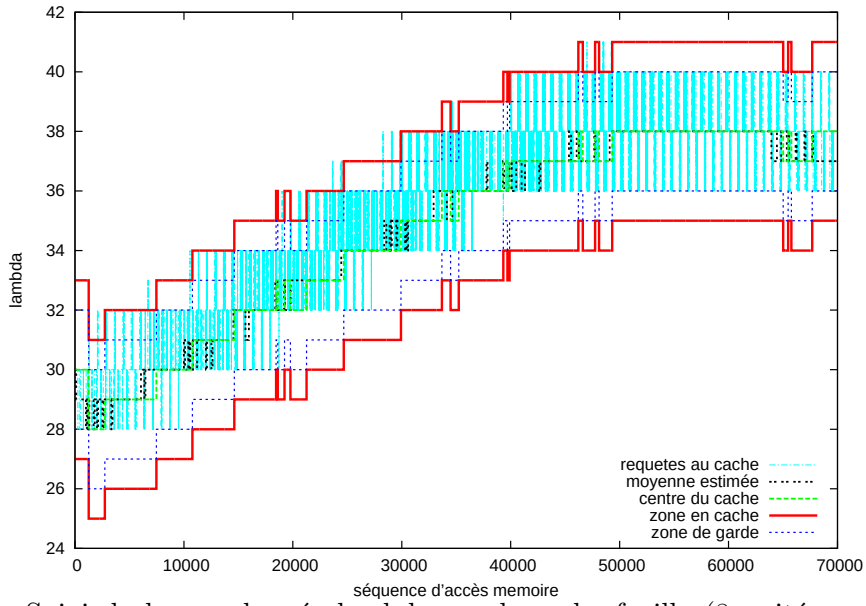


FIG. 4.8 – Suivi de la coordonnée lambda par le cache feuille (8 unités, segment -2, $\boldsymbol{l_{mem}}$=5 cycles, $\boldsymbol{d_{mem}}$=8 octets/cycle).

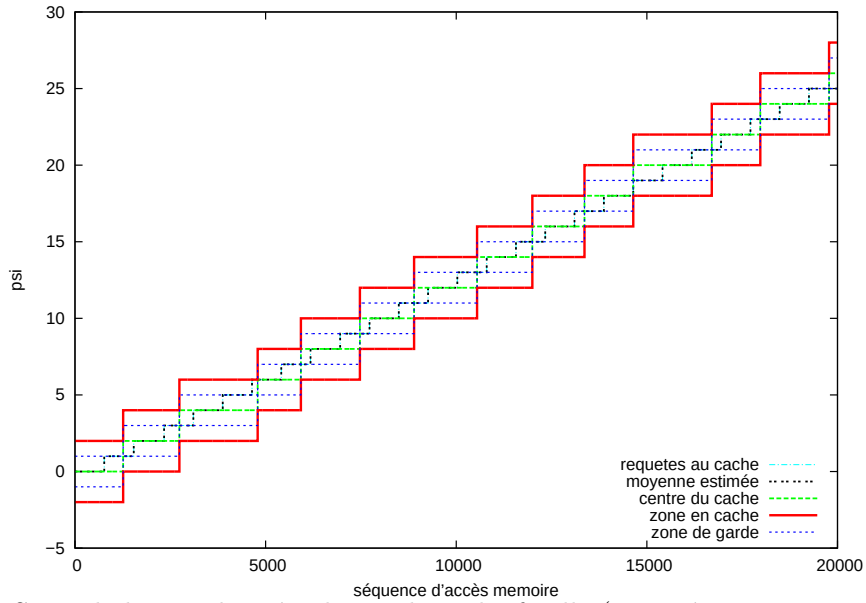


FIG. 4.9 – Suivi de la coordonnée phi par le cache feuille (8 unités, segment -2, $\boldsymbol{l_{mem}}$=5 cycles, $\boldsymbol{d_{mem}}$=8 octets/cycle).

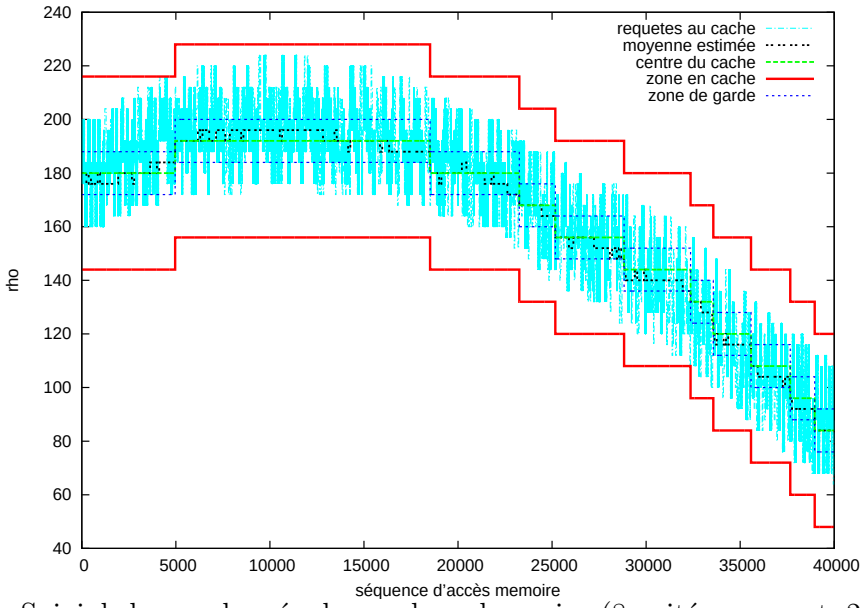


FIG. 4.10 – Suivi de la coordonnée rho par le cache racine (8 unités, segment -2, $\boldsymbol{l_{mem}}$=5 cycles, $\boldsymbol{d_{mem}}$=8 octets/cycle).

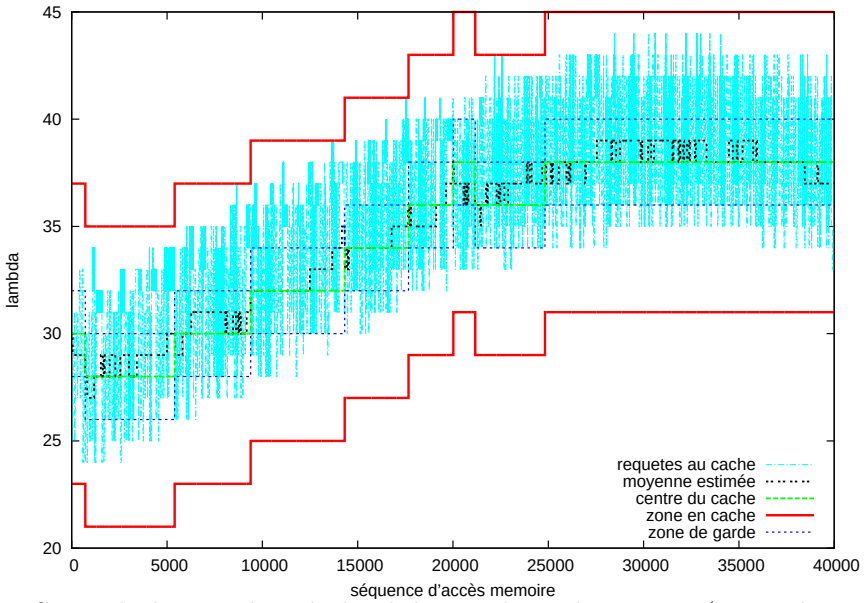


FIG. 4.11 – Suivi de la coordonnée lambda par le cache racine (8 unités, segment -2, $\boldsymbol{l_{mem}}$=5 cycles, $\boldsymbol{d_{mem}}$=8 octets/cycle).

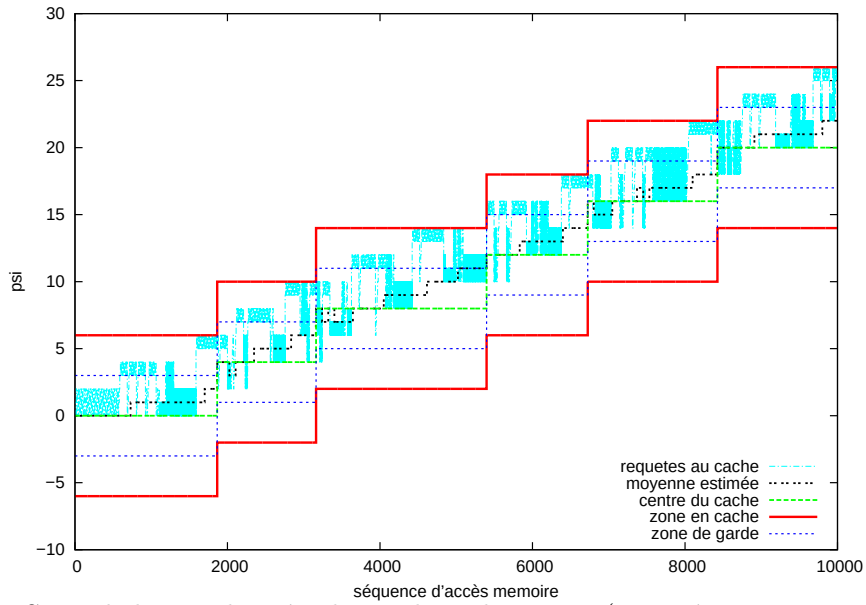


FIG. 4.12 – Suivi de la coordonnée phi par le cache racine (8 unités, segment -2, $\boldsymbol{l_{mem}}$=5 cycles, $\boldsymbol{d_{mem}}$=8 octets/cycle).

3.2.2 Surcharge du cache

Nous nous contenterons ici de donner quelques commentaires sur la taille de la zone en cache. Le tableau 4.4 compare ainsi la taille des caches feuilles et racines (4 et 8 unités) pour les dimensions $\boldsymbol{\rho}$ et $\boldsymbol{\lambda}$ avec la taille idéale des plans mémoire $\boldsymbol{l_\rho \cdot L_\lambda}$ nécessaires à la rétroprojection d'un bloc de voxel comme nous l'avons vu au chapitre 3. A partir de la taille des plans mémoire définis par le cache 3D-AP et de ceux idéalement nécessaires, nous avons calculé un taux de surcharge espéré du cache $\boldsymbol{E(\eta_{cache})}$.

Niveau du Cache	Nombre d'unités (Taille du bloc)	Numéro de segment	Plan mémoire ($\boldsymbol{l_\rho \cdot L_\lambda}$)		$\boldsymbol{E(\eta_{surcharge})}$
			idéal	cache	
Feuille	1 unité ($\mathbf{16^2 \cdot 3}$ voxels)	seg 0	$\mathbf{21 \cdot 4}$	$\mathbf{27 \cdot 7}$	2.3
		seg -2	$\mathbf{21 \cdot 6}$	$\mathbf{27 \cdot 7}$	1.5
Racine	4 unités ($\mathbf{32^2 \cdot 3}$ voxels)	seg 0	$\mathbf{40 \cdot 4}$	$\mathbf{76 \cdot 13}$	6.2
		seg -2	$\mathbf{40 \cdot 8}$	$\mathbf{76 \cdot 13}$	3.1
Racine	8 unités ($\mathbf{32^2 \cdot 6}$ voxels)	seg 0	$\mathbf{40 \cdot 7}$	$\mathbf{76 \cdot 15}$	4.0
		seg -2	$\mathbf{40 \cdot 10}$	$\mathbf{76 \cdot 15}$	2.9

TAB. 4.4 – Taille de la zone en cache pour 1, 4 et 8 unités

Le cache feuille a été configuré de manière à coller au plus près de la sinusoïde définie pour le segment -2 comme on peut le constater sur les figures 4.7 et 4.8 pour les coordonnées rho et lambda. Ainsi, le taux de surcharge espéré $\boldsymbol{E(\eta_{cache})}$ est de 1.5. Pour le segment 0 avec le même jeux de paramètres pour le cache, ce taux est plus élevé (2.3) car le cache est alors trop large en lambda pour suivre les sinusoïdes 3D. En effet, les sinusoïdes du segment 0 ne nécessitent pas de déplacement selon la dimension lambda. Pour le cache racine, nous avons préféré englober au plus large la sinusoïde 3D afin d'éviter au maximum les défauts de cache très coûteux car ils nécessitent alors un accès en mémoire externe. La taille plus importante du cache racine peut s'observer sur les figures 4.10 et 4.11 pour les coordonnées rho et lambda. Le taux de surcharge espéré $\boldsymbol{E(\eta_{surcharge})}$ du cache est alors beaucoup plus élevé. Ainsi pour 8 unités, il est de 2.9 pour le segment -2 et de 4.0 pour le segment 0.

Si la valeur $\boldsymbol{E(\eta_{surcharge})}$ donne une première indication du comportement futur du cache, elle ne correspond pas fidèlement au comportement réel du cache. En effet, le cache 3D-AP ne charge pas un plan mémoire $\boldsymbol{l_\rho \cdot L_\lambda}$ après l'autre comme le ferait une stratégie mémoire de type "ping pong". Les mises à jour du cache se font lors du dépassement de la zone de garde et avec un déplacement d'une taille fixe (voir chapitre 3). Contrairement aux dimensions rho et lambda, le déplacement en phi du cache feuille ne fait pas appel au calcul de la moyenne statistique des requêtes en phi. En effet, les requêtes en phi étant entièrement séquentielles (boucle sur phi), la dernière requête en phi suffit pour enclencher une mise à jour du cache (voir figure 4.9). Pour le cache racine, ce n'est pas le cas. En effet, il existe une légère désynchronisation entre les différentes unités de rétroprojection. Ainsi, la séquence des requêtes en phi des différents caches feuilles est plus dispersée (voir figure 4.12).

3.3 Comportement du cache sur la carte

Nous avons mesuré pour les caches feuille et racine plusieurs métriques inspirées de l'étude théorique sur les caches du chapitre 3. Nous présentons ici les mesures obtenues pour les segments 0 et -2 dont les accès mémoire dessinent des sinusoïdes 3D avec respectivement des déplacements selon 2 et 3 dimensions dans l'espace mémoire 3D.

3.3.1 Cache feuille

La figure 4.13 représente pour un cache non-hiérarchique et un cache feuille, le taux d'utilisation d'un *bin* mis en cache $\boldsymbol{\eta_{cache}}$, le coût moyen de mise en cache de 4 *bins* $\boldsymbol{\hat{C}_{cache}(4bins)}$, le taux de défaut $\boldsymbol{\eta_{defaut}}$ et le taux de surcharge du cache $\boldsymbol{\eta_{surcharge}}$.

Taux de réutilisation et de défaut pour le cache non-hiérarchique

Pour un cache non-hiérarchique avec un seul cache, on peut observer sur la figure 4.13 que le taux de réutilisation des données ne varie pas avec la latence mémoire : il est de 10.1 pour le segment 0 et de 9.2 pour le segment -2 (graphe a). Ce taux de réutilisation est à comparer avec le taux théorique de réutilisation des données qui idéalement est de 36 pour le segment 0 et 24 pour le segment -2 (graphe d). Cette différence s'explique par le taux de surcharge du cache assez important autour de 3,6 pour le segment 0 et 2,6 pour le segment -2 . Ce taux de surcharge est plus élevé que la valeur espérée $\boldsymbol{E(\eta_{surcharge})}$ par les mesures en simulation. En effet, le mécanisme de mise à jour du cache 3D-AP dessine une sinusoïde plus large que celle correspondante à une simple stratégie mémoire de type "ping pong". Le cache 3D-AP fait des mises à jour du cache plus larges car il effectue une prédiction pour plusieurs angles phi futurs, alors que la stratégie "ping pong" met à jour le cache uniquement pour l'angle suivant $\boldsymbol{phi + 1}$. Ce taux de réutilisation permet toutefois le masquage des transferts du cache car comme l'indique le graphe b, le coût de transfert en mémoire de 4 bins dont a besoin le pipeline à chaque cycle de calcul reste inférieur au taux de réutilisation. Il faut en moyenne entre 2 et 8 cycles pour mettre en cache un vecteur de 4 bins qui sera réutilisé en moyenne pendant 9 à 10 cycles de calcul. Enfin, le taux de défaut est faible autour de 0,1 % ce qui prouve que le cache réussit à suivre la sinusoïde 3D (graphe c).

Taux de réutilisation et de défaut pour le cache hiérarchique

Pour un cache hiérarchique, nous avons effectué ces mêmes mesures pour un des 4 caches feuilles d'une architecture parallèle avec 4 unités (avec 8 unités nous n'avons plus de ressources nécessaires sur le virtex 2 Pro pour placer les modules de mesures du cache). Sur la figure 4.13, nous pouvons observer tout d'abord que le taux de réutilisation est plus faible (entre 9 et 5) et surtout qu'il décroît avec la latence. Ce taux de réutilisation qui décroît s'explique par un nombre plus important de transferts mémoire entre les caches feuille et racine comme l'indique le taux de surcharge qui augmente avec la latence mémoire (compris entre 3 à 5 pour le segment -2). Le cache feuille ne suit plus aussi efficacement la sinusoïde 3D comme il le faisait sans la hiérarchie mémoire. Toutefois, s'il charge trop de données en cache, il obtient encore un faible taux de défauts autour

de 0,1 %. Le coût de mise en cache pour 4 unités varie peu avec la latence mémoire, il est compris entre 1,5 et 2 cycles. En effet, ce coût ne correspond plus à la recherche de données en mémoire externe mais au simple transfert entre les caches feuille et racine.

Coût des défauts

La figure 4.14 présente le coût d'un défaut en cache. Les défauts sont traités par le cache de manière non-prioritaire après la fin de mise à jour du cache. C'est pourquoi, nous faisons la distinction dans la suite de cette étude entre les défauts retardés qui attendent la fin d'une mise à jour du cache et ceux non retardés qui accèdent sans attendre au bus mémoire. Le coût d'un défaut non retardé augmente linéairement avec la latence pour un cache non hiérarchique (1 unité) comme illustré sur le graphe b. Pour un cache hiérarchique, les défauts non retardés au niveau feuille peuvent être retardés au niveau racine, c'est pourquoi on observe un coût plus important et non linéaire de ces défauts pour 4 unités (graphe b). Les défauts réellement non retardés coûtent quelques dizaines de cycles (entre 5 et 40 cycles selon la latence mémoire) ce qui est supérieur au coût de mise en cache d'une donnée. Ceci est du à l'accès non groupé en ligne mémoire des défauts, chaque défaut paye le prix de la latence mémoire. Alors que les accès à la mémoire pour la mise en cache d'une donnée bénéficient d'un coût partagé de la latence mémoire, par tous les *bins* de la ligne mémoire accédée. Les défauts retardés coûtent plusieurs centaines de cycles (entre 100 et 700 cycles selon la latence mémoire). C'est ce coût important des défauts retardés qui fait augmenter le coût moyen d'un défaut (graphe a). Toutefois, la proportion des défauts retardés reste inférieure à 7 % (graphe d). Ainsi au final pour une latence de 30 cycles, un défaut du cache coûte 75 cycles pour une unité et le double pour 4 unités.

Occupation du bus et traitement des défauts

Puisque les transferts du cache sont prioritaires sur les traitements des défauts, le temps de traitement des défauts est dépendant du temps de transfert du cache. Or le temps de traitement des défauts donne une indication directe sur l'efficacité de reconstruction car lorsqu'un défaut est traité le pipeline est en arrêt sinon il marche à plein régime. La figure 4.15 illustre via la représentation des taux d'occupation du bus par le cache (nombre de cycles de transfert du cache sur le nombre de cycles au total) et de traitement des défauts (nombre de cycles de traitement des défauts sur le nombre de cycles au total), la corrélation existante entre les temps de transfert du cache et de traitement des défauts. Plus le taux d'occupation du bus augmente, plus le temps de traitement des défauts augmente. Par exemple pour une unité, quand le taux d'occupation du bus atteint 60 % (latence de 15 cycles), la répercussion sur le retardement des défauts devient visible avec un accroissement net du temps de traitement des défauts avec la latence. Pour le cache hiérarchique avec 4 unités, le taux d'occupation du bus entre les caches feuilles et le cache racine est plus important puisque ce bus est partagé par les 4 caches feuilles. Ainsi, l'occupation du bus est déjà à plus de 50 % pour une latence faible. Ce taux d'occupation augmente quasi linéairement jusqu'à la saturation du bus à 80%.

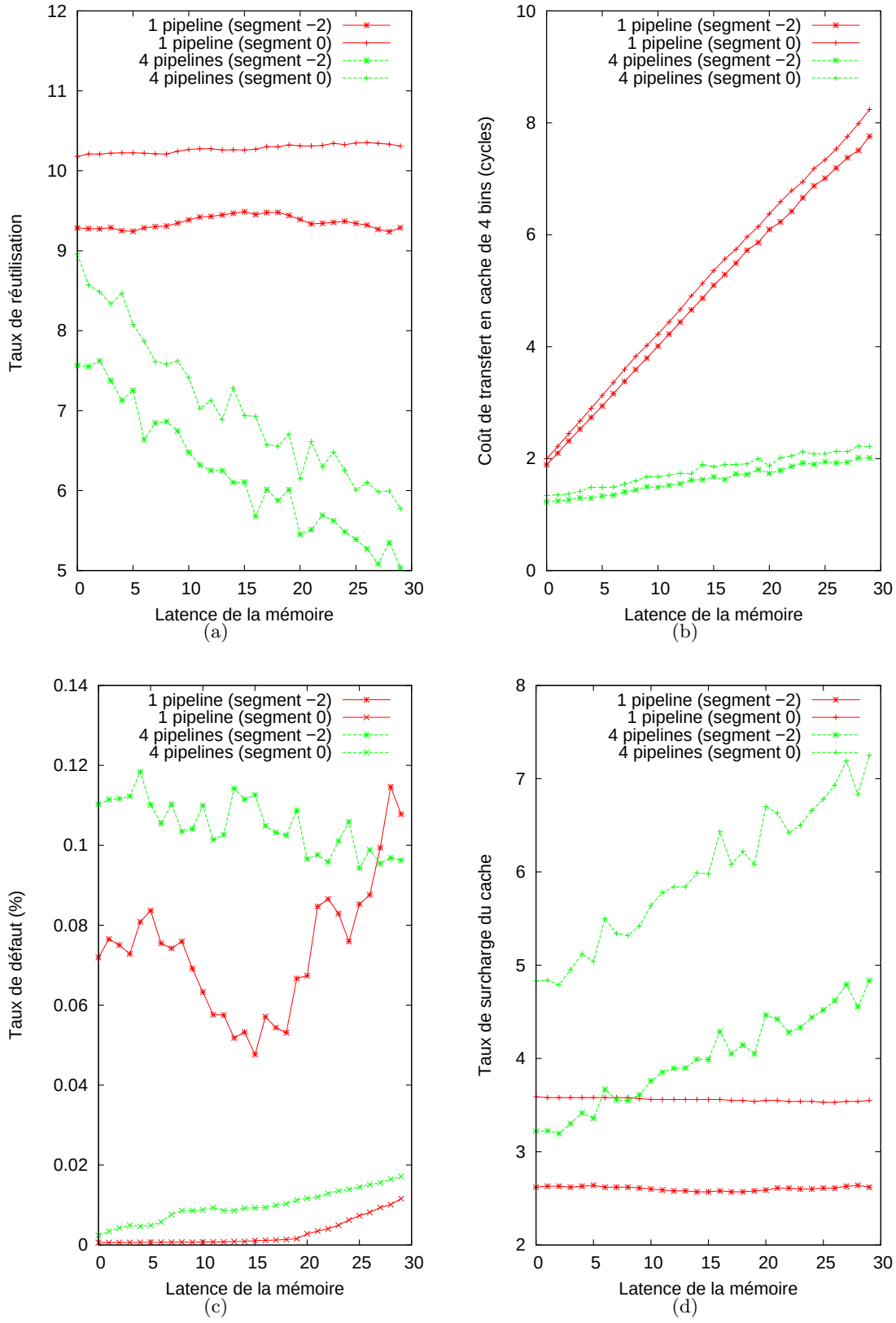


FIG. 4.13 – η_{cache} (a), $\hat{C}_{cache}(4bins)$ (b), η_{defaut} (c) et $\eta_{surcharge}$ (d) du cache feuille pour 1 et 4 unités (segments -2 et 0)

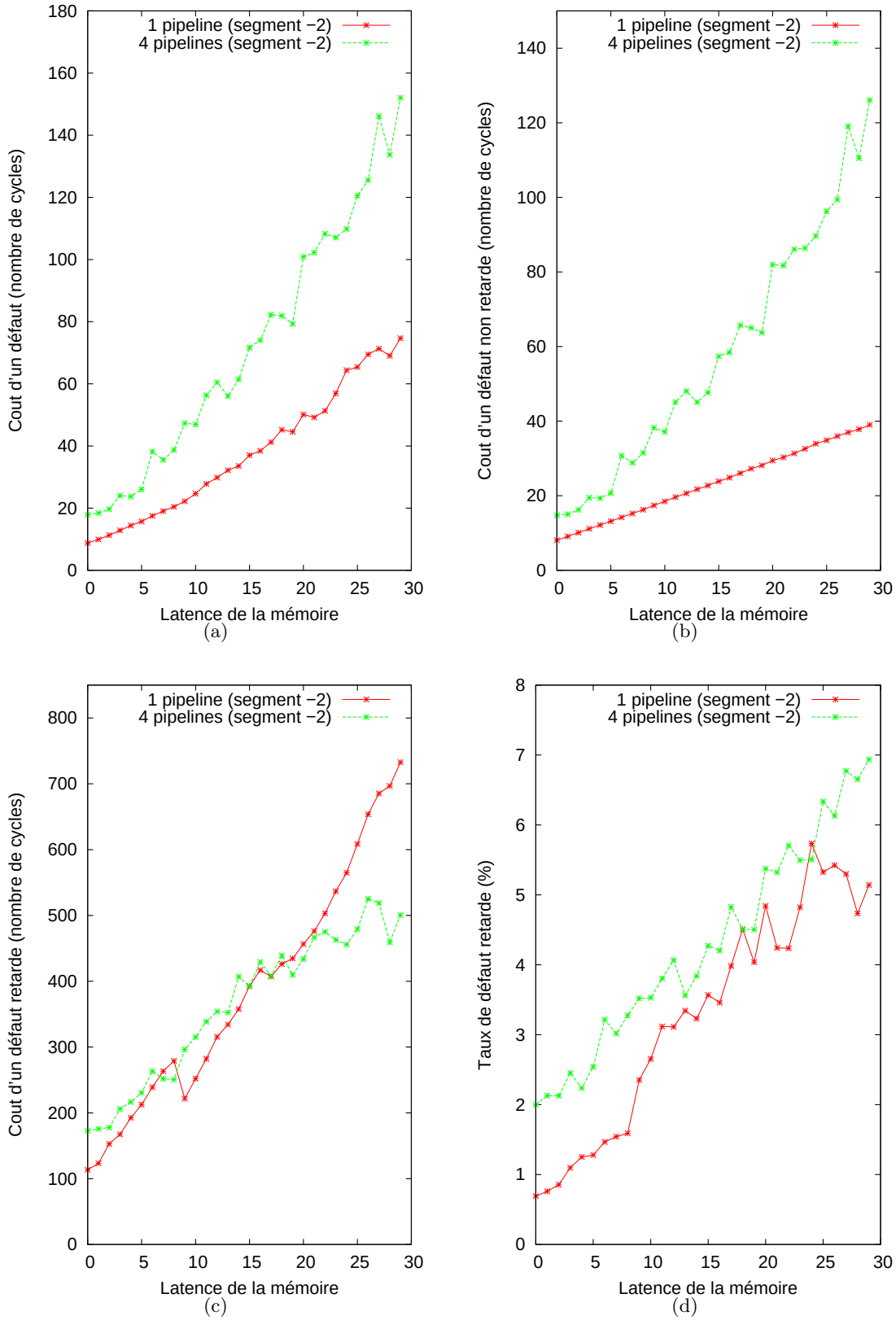


FIG. 4.14 – Coût des défauts (a), des défauts non retardés (b) et des défauts retardés (c) et la proportion de défauts retardés (d) du cache feuille pour 1 et 4 unités (segments -2)

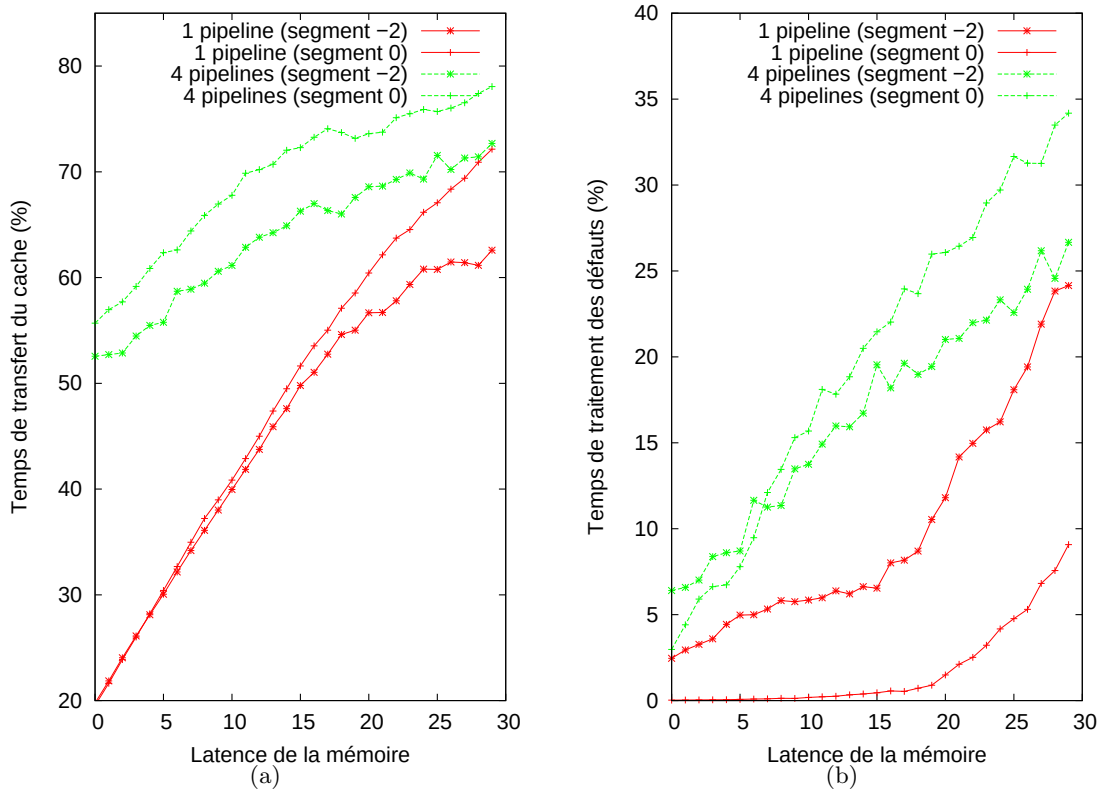


FIG. 4.15 – Taux de transfert du cache (a) et taux de traitement des défauts (b) au niveau feuille pour 1 et 4 unités (segments -2 et 0)

3.3.2 Cache racine

Taux de réutilisation et taux de défauts

Pour le cache racine, le taux d'utilisation des données mises dans le cache racine par les caches feuilles est légèrement supérieure à 1 pour 4 unités et proche de 2 pour 8 unités (voir figure 4.16). Cela prouve la pertinence de la hiérarchie mémoire qui arrive à bénéficier de la localité spatiale et temporelle entre blocs de voxels. Ce taux de réutilisation est obtenu malgré un fort de taux de surcharge du cache racine (égal à 4 pour le segment -2) car comme nous l'avons vu précédemment le cache a été paramétré avec une zone en cache assez large afin d'éviter au maximum les défauts. Le taux de défauts ne varie pas avec la latence, ce qui est un bon signe de la capacité du cache à suivre la sinusoïde 3D malgré une dégradation du temps d'accès à la mémoire externe. Ce taux de défaut est plus élevé que celui observé pour le cache feuille. Il est au alentour de 0,7 % pour 4 unités et de 0,2 % pour 8 unités. Au vu des courbes de simulation (voir figures 4.10, 4.11 et 4.12), ce taux de défaut n'est pas du au mauvais cadrage de la sinusoïde 3D mais au retard du cache par rapport aux requêtes. Les mises à jour du cache sont pertinentes mais demandent beaucoup de données à charger depuis la mémoire externe et donc un temps important de chargement. Ainsi des requêtes vont plus vite que le rythme de mise à jour du cache et sont alors assignés en défaut. Puisque la sinusoïde 3D d'un macro-bloc est plus large qu'un bloc de voxels, il faut transférer plus de données en cache. C'est ce que l'on peut observer sur la figure 4.18 (graphe a) avec une taux d'occupation du bus élevé. Cela a toutefois l'avantage de réduire le coût moyen de mise en cache d'une donnée car les accès par ligne mémoire de taille plus importante réduisent par deux ce coût par rapport au cache feuille. Ce coût évolue linéairement avec la latence mémoire et atteint 4 cycles pour une latence de 30 cycle (graphe b de la figure 4.16) contre 8 cycles pour le cache feuille (voir graphe b de la figure 4.13).

Traitements des défauts

Le temps de mise à jour plus conséquent du cache racine implique également que les défauts sont plus souvent et plus longtemps retardés. C'est ce qu'illustre la figure 4.17 avec une proportion de défauts retardés entre 10 % et 20 % et la figure 4.17 avec un coût de ces défauts qui varie entre 500 et 3000 cycles de manière quasi linéaire avec la latence mémoire. Les coûts des défauts non retardés sont invariants par rapport à ceux observés pour un cache non hiérarchique et de l'ordre de quelques dizaines de cycles selon la latence. Au final, le coût moyen d'un défaut au niveau du cache racine varie quasi linéairement avec la latence et est de l'ordre des centaines de cycles (entre 100 et 700 cycles). Les défauts coûtent donc chers. C'est ce que l'on peut observer sur la figure 4.18 (graphe b) avec un taux de traitement des défauts qui atteint près de 50 % pour une latence de 30 cycles.

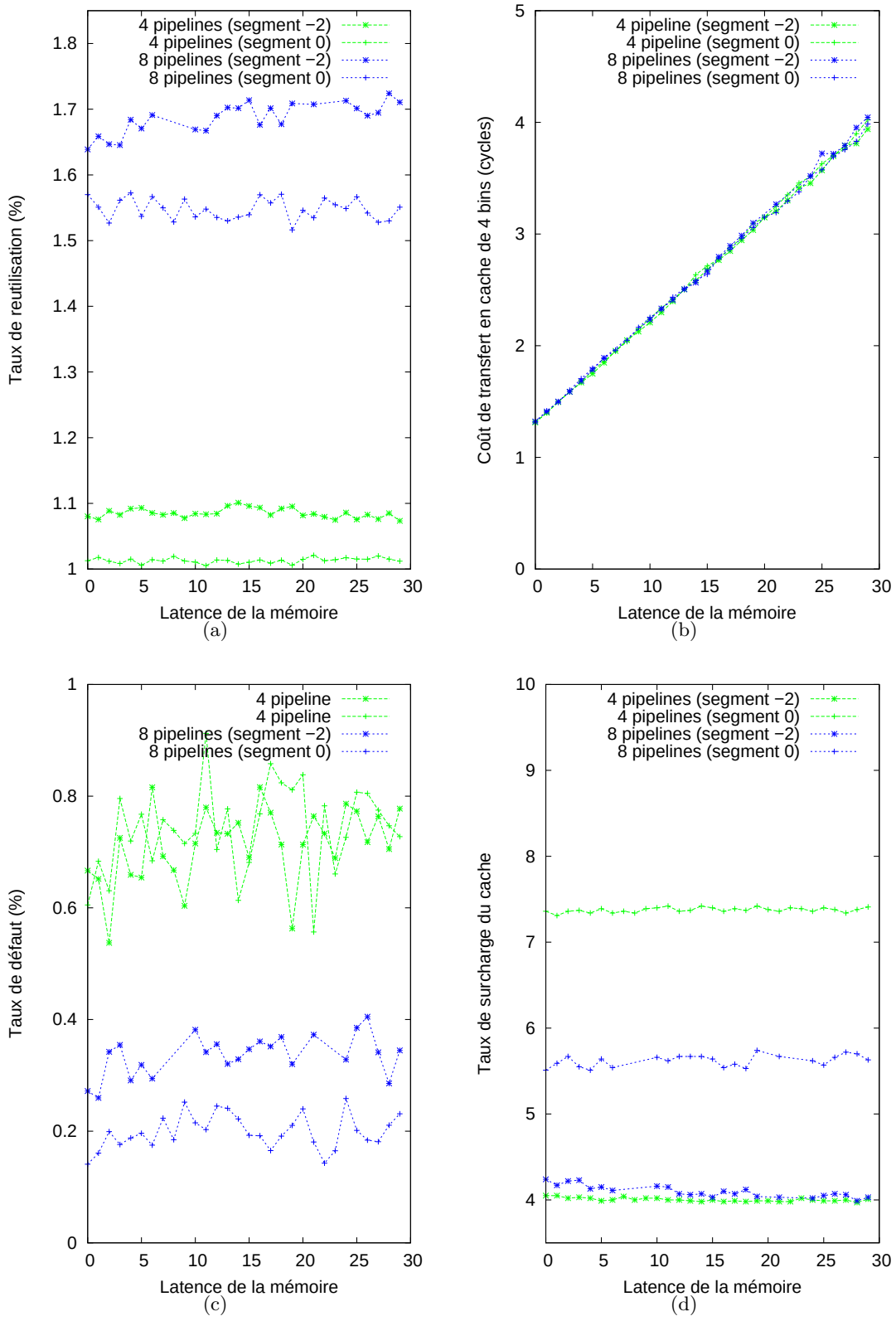


FIG. 4.16 – η_{cache} (a), $\hat{C}_{cache}(4bins)$ (b), $\eta_{surcharge}$ (c) et η_{defaut} (d) du cache racine pour 4 et 8 unités (segment -2 et 0)

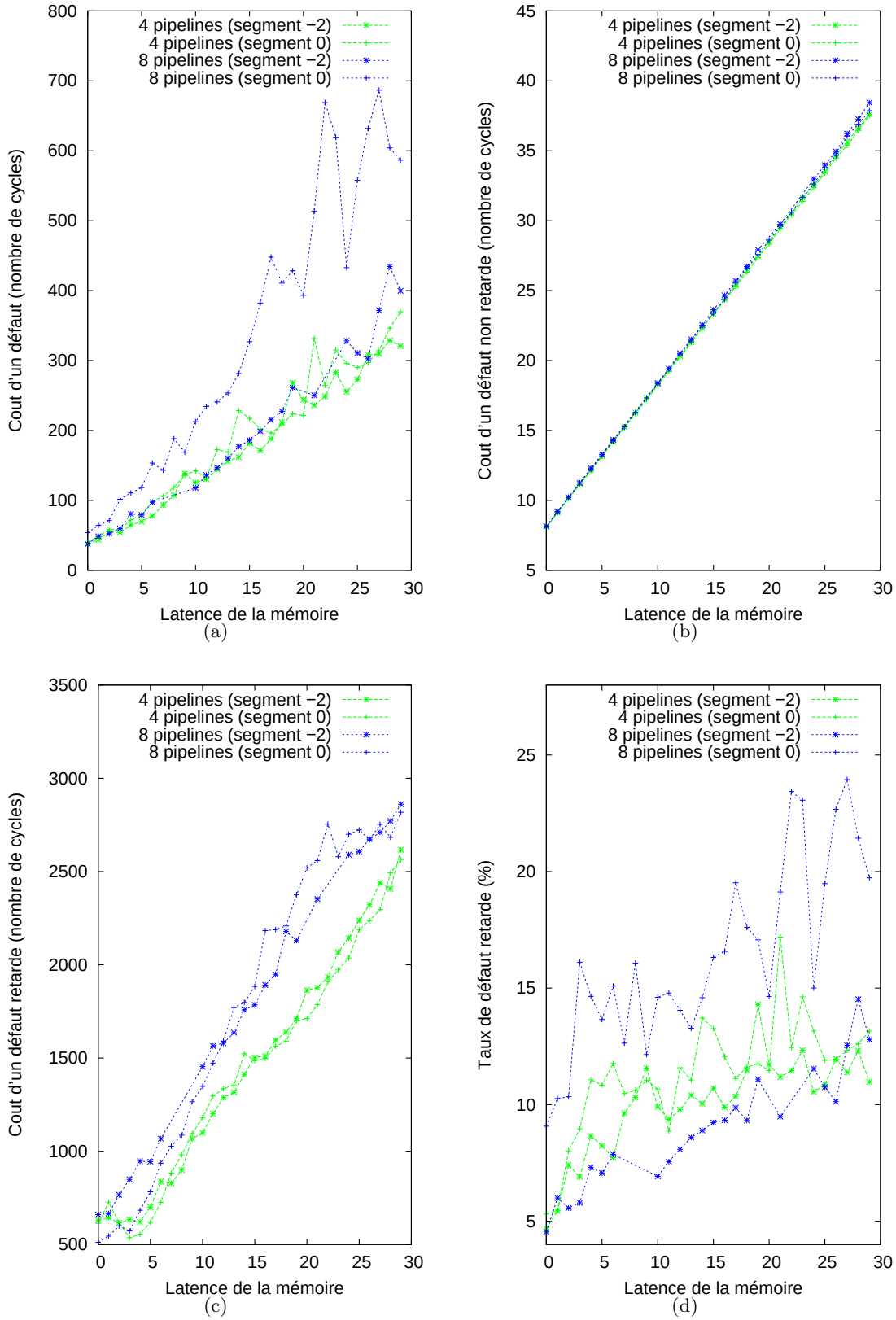


FIG. 4.17 – Coût des défauts (a), des défauts non retardés (b) et des défauts retardés (c), et proportion de défauts retardés (d) du cache racine pour 4 et 8 unités (segments -2 et 0)

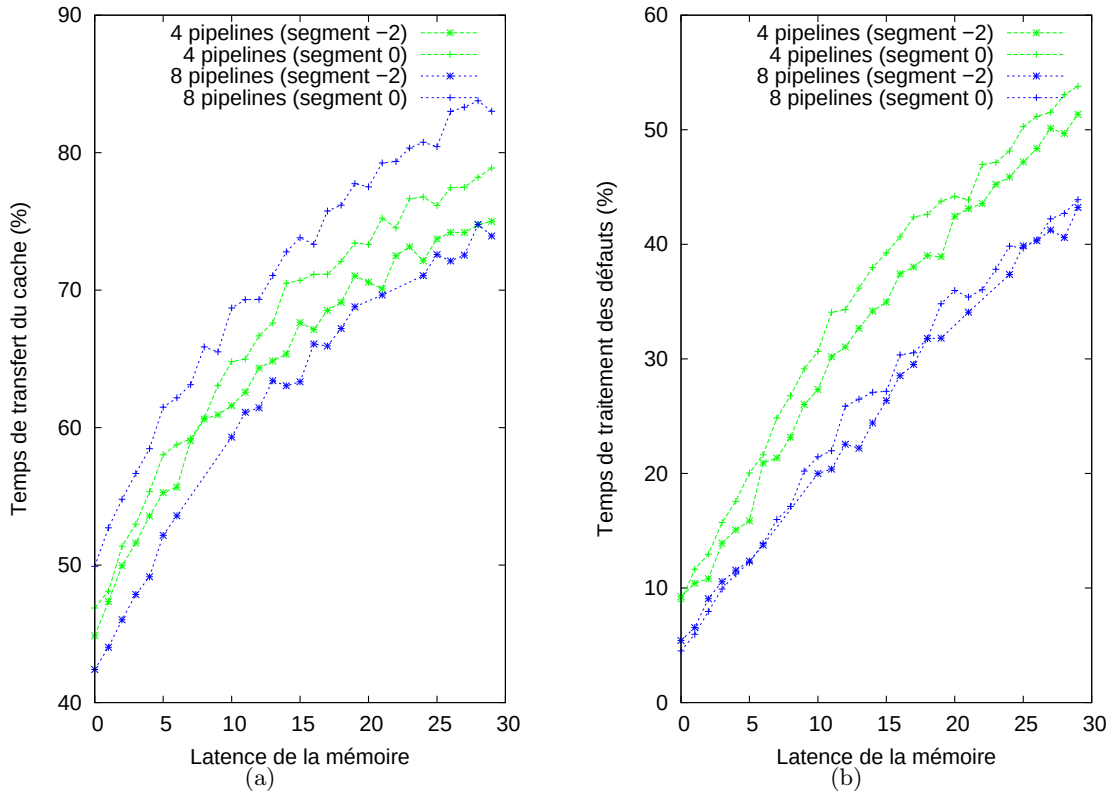


FIG. 4.18 – Taux d'occupation du bus mémoire (a) et taux de traitement des défauts (b) pour le cache racine pour 4 et 8 unités (segments -2 et 0)

4 Efficacité de reconstruction

Nous présentons ici l'efficacité de reconstruction mesurée sur la plateforme SoPC pour une, quatre et huit unités de rétroprojection de notre architecture 3P. Chaque unité de rétroprojection effectue la rétroprojection d'un bloc de $\mathbf{16^2 \cdot 3}$ voxels. L'efficacité de reconstruction est présentée en nombre de cycles par opération de mise à jour de voxel. Idéalement, chaque pipeline permet une efficacité de reconstruction d'un cycle par opération (voir chapitre 3).

4.1 Une unité de rétroprojection

Sur la figure 4.19, nous présentons l'efficacité de reconstruction obtenue pour une unité de reconstruction en fonction de la latence du bus mémoire.

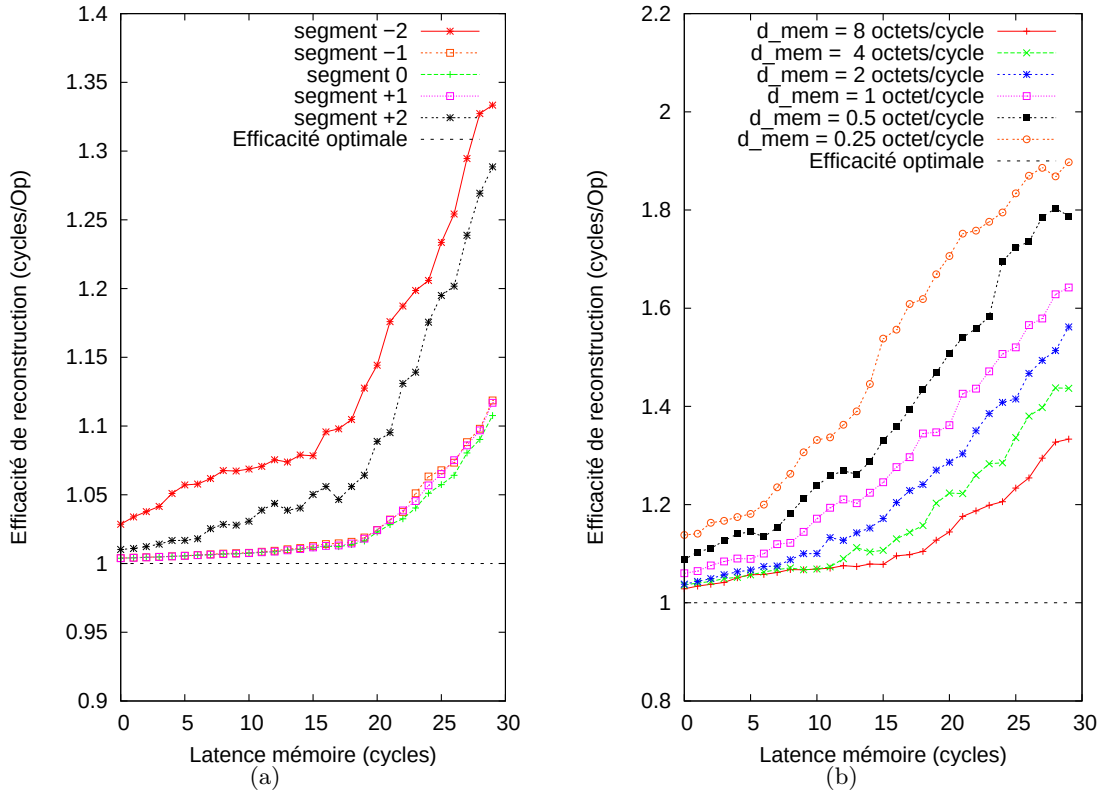


FIG. 4.19 – Efficacité de reconstruction pour une unité en fonction de la latence mémoire : (a) pour un débit mémoire de 8 octets/cycle et pour les différents segments du scanner HR+ et (b) pour le segment -2 et différents débits mémoire.

Sur le graphe (b), l'efficacité de reconstruction est représentée pour les différents segments du sinogramme 4D du scanner HR+. Nous pouvons observer tout d'abord la très bonne efficacité de reconstruction des segments 0 et $\mathbf{\{+1, -1\}}$ quasiment égale

à 1 cycle/op. pour une latence inférieure à 15 cycles. Pour les segments $\mathbf{\{+2, -2\}}$, si cette efficacité est également très proche de l'optimum de 1 cycle/op, on peut observer toutefois une sensibilité plus importante à la latence mémoire. En effet, le parcours mémoire dessiné lors de la rétroprojection du segment $\mathbf{\{+2, -2\}}$ est plus complexe à suivre que celui du segment 0. Car comme nous l'avons vu au chapitre 3, la reconstruction du segment 0 correspond à une rétroprojection 2D avec des valeurs de `lambda` fixes, alors que les reconstructions des autres segments correspondent à des rétroprojections 3D avec des valeurs de `lambda` variables (voir chapitre 3). C'est ce qui explique la plus grande difficulté du cache 3D-AP à répondre dans les temps aux requêtes du pipeline de rétroprojection. Toutefois malgré ce parcours mémoire 3D avec trois degrés de liberté ($\boldsymbol{\phi}$,$\boldsymbol{\rho}$,$\boldsymbol{\lambda}$), l'efficacité de reconstruction reste proche de l'optimum même pour de fortes latences et de faibles débits de la mémoire externe. C'est ce qu'illustre le graphe (a) de la figure 4.19, avec la reconstruction du segment -2 pour différents débits du bus mémoire. Le pipeline de rétroprojection est ainsi peu pénalisé par le goulot d'étranglement que constitue la mémoire externe.

4.2 Parallélisation

La figure 4.20 présente l'efficacité de rétroprojection par Unité de Traitement (UT) obtenue avec 1 unité seule et 4 et 8 unités en parallèle pour un débit de 8 octets/cycle et pour le segment -2 qui constitue le segment le plus pertinent pour tester les capacités du cache 3D-AP à suivre un parcours mémoire 3D. Pour une latence de 5 cycles, l'efficacité de reconstruction est ainsi de 1,05 cycle/op/UT pour 1 unité, 1,31 cycle/op/UT pour 4 unités et 1,85 cycles/op/UT pour 8 unités. Cela représente une efficacité de parallélisation de $\boldsymbol{\eta_{//} = 0,8}$ pour 4 unités (accélération d'un facteur 3,2) et de $\boldsymbol{\eta_{//} = 0,56}$ pour 8 unités (accélération d'un facteur 4,5).

Afin d'évaluer la sensibilité de l'architecture parallèle au paramétrage du cache, nous avons représenté en figure 4.21, l'efficacité de reconstruction obtenue avec 8 unités en parallèle selon la position du macro-bloc reconstruit. En effet, puisque la forme de la sinusoïde 3D à suivre est fonction de la position des blocs, nous pouvons ainsi tester la capacité du cache 3D-AP à suivre, avec un seul et même jeu de paramètres, différents parcours mémoire. On peut observer que l'efficacité de reconstruction obtenue pour les macro-blocs excentrés est inférieure à celle obtenue avec les macro-blocs centrés ([X,Y]=[2,2]). En effet, les macro-blocs excentrés dessinent une sinusoïde aux courbes plus prononcées que les macro-blocs du centre. C'est pourquoi le cache 3D-AP a alors plus de difficulté à suivre ce parcours mémoire. On peut également observer que le paramétrage du cache privilégie certaines positions à d'autres. Par exemple pour le segment -2, le macro-bloc [4,4] a une efficacité de reconstruction de 1.75 cycle/op/UT alors que le macro-bloc [2,1] a une efficacité de 2.35 cycles/op/UT. Ces différences de performance restent toutefois relatives et prouvent que le cache 3D-AP arrive à s'adapter à plusieurs types de parcours mémoire.

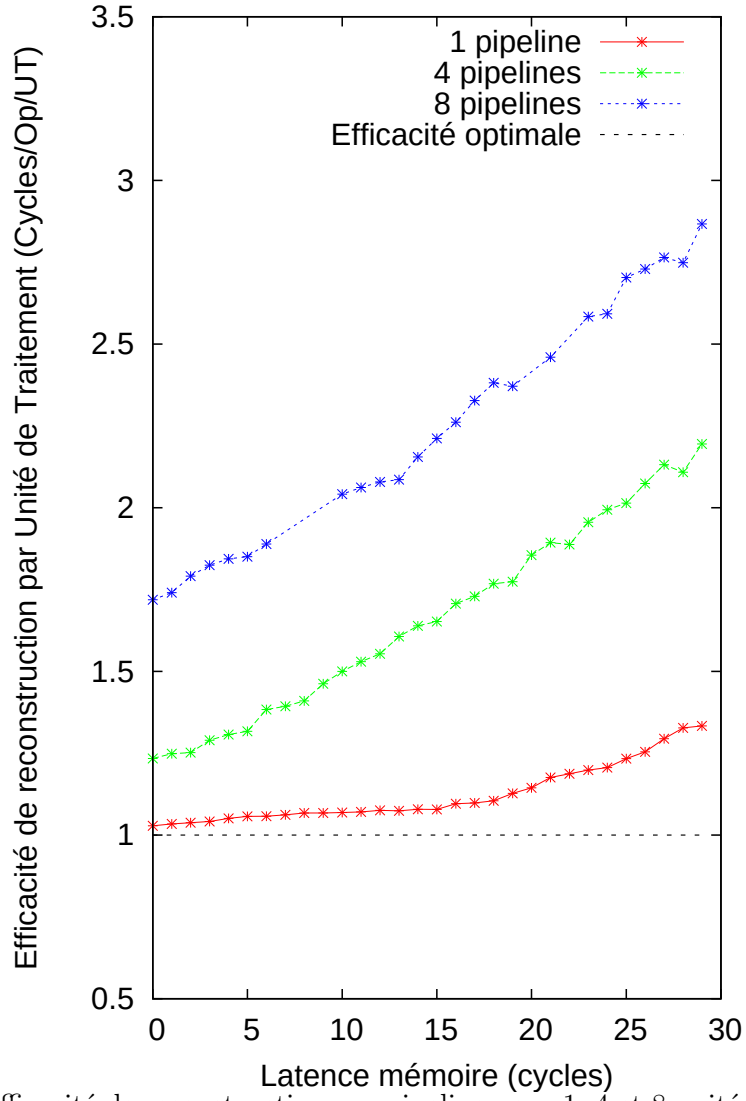
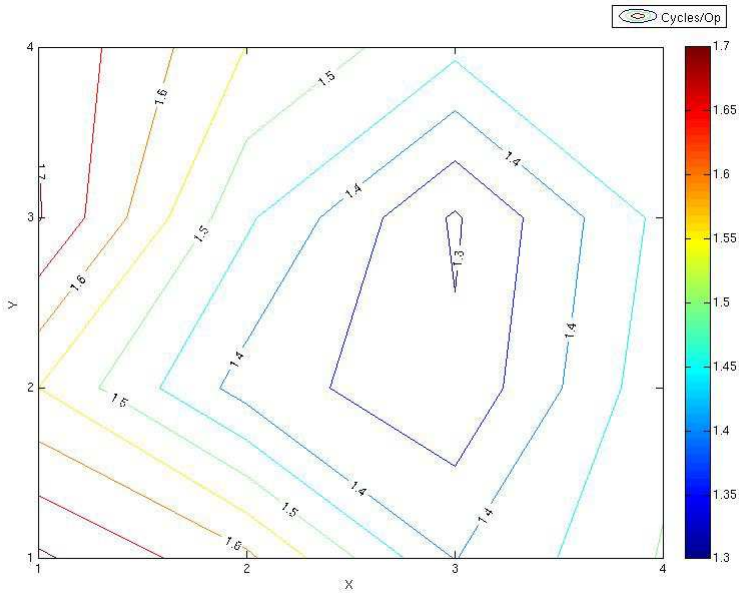
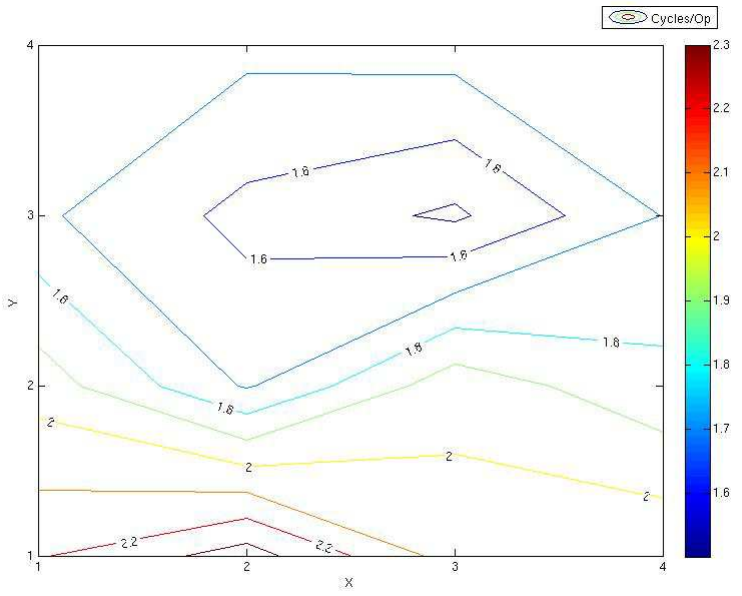


FIG. 4.20 – Efficacité de reconstruction par pipeline pour 1, 4 et 8 unités en fonction de la latence mémoire ($\boldsymbol{d_{mem}}$=8 octets/cycle, segment -2).



(a)



(b)

FIG. 4.21 – Efficacité de reconstruction par pipeline pour la rétroprojection des segments 0 (a) et -2 (b) avec 8 pipelines en parallèle en fonction de la position (X,Y) du macro-bloc reconstruit (constitué de $\mathbf{2 \cdot 2 \cdot 2}$ blocs de $\mathbf{16^2 \cdot 3}$ voxels).

5 Conclusion

Dans ce chapitre, nous avons décrit l'implémentation de notre architecture 3P sur une plateforme SoPC. Nous avons constaté que la rétroprojection 3D ainsi implémentée sur FPGA offre une très bonne qualité de reconstruction et une excellente efficacité de reconstruction (1 à 2 cycles par unité de rétroprojection). Cette efficacité est obtenue grâce au bon comportement du cache 3D-AP lors du suivi de la sinusoïde 3D dessinée dans l'espace 3D des données. Ce cache permet ainsi de ne plus limiter la reconstruction par le goulot d'étranglement que constitue la mémoire externe.

Les mesures en simulation et sur la carte, nous permettent d'affirmer que le cache 3D-AP suit correctement la sinusoïde 3D correspondant aux requêtes du module de rétroprojection. Le faible taux de défaut obtenu en est la preuve. Le cache mémoire ne perd pas le fil du parcours mémoire en donnant la priorité aux mises à jour du cache et non aux traitements des défauts. Une requête trop en avance par rapport au mécanisme de prédiction et de préchargement est mise en défaut et bloque l'unité de rétroprojection. Cela a bien entendu pour désavantage d'augmenter sensiblement le coût des défauts. Mais si les défauts n'étaient ainsi plus retardés, le cache risquerait alors de ne plus être en phase avec les requêtes. Celles-ci seraient alors servies directement en défaut, ce qui signifie avec un accès en mémoire externe et sans mise en cache des données. Ainsi la localité spatiale et temporelle intrinsèque à la rétroprojection par bloc de voxels seraient moins efficacement exploitée et le coût mémoire augmenterait sensiblement.

Malgré les obstacles que constituent la latence de la mémoire externe, un gain en performance est obtenu par la parallélisation. La hiérarchie mémoire du cache 3D-AP, s'est avérée ainsi pertinente. Avec 8 unités de rétroprojection, le taux de réutilisation des données est ainsi multiplié par un facteur 2. On observe toutefois que l'efficacité de reconstruction s'érode avec le nombre d'unités de rétroprojection mises en parallèle. Une raison majeure à cela est qu'une architecture parallèle fonctionnant à plein régime signifie un coût mémoire plus important (plus de données doivent être chargées pour nourrir les $\boldsymbol{n}$ unités de rétroprojection) et un coût de calcul moins important (le temps de calcul est divisé par le nombre d'unités de calcul). Ainsi l'équilibre entre débit de calcul et débit mémoire est plus difficile à atteindre. C'est ce qu'illustre la figure 4.22 qui présente le ratio entre le nombre de cycles nécessité par la mise en cache racine de la sinusoïde 3D du macro-bloc reconstruit et le nombre de cycles de calcul d'une architecture parallèle fonctionnant à plein régime. Idéalement, ce ratio devrait pour 1, 4 et 8 unités s'élever respectivement à **25%**, **40%** et **50%**. Mais en pratique avec le cache 3D-AP, nous avons mesuré pour 1, 4 et 8 unités un ratio plus important, égal à respectivement **88%**, **164%**, **210%**. Cela s'explique par la quantité plus importante de données chargées par le cache 3D-AP ($\boldsymbol{\eta_{surcharge} > 1}$), mais également par l'existence de conflits entre les caches feuilles pour l'accès au cache racine. Ces conflits peuvent être réduits en synchronisant de manière plus adéquate l'exécution de la rétroprojection sur les $\boldsymbol{n}$ unités de calcul. Cependant, ces conflits constitueront toujours une limite incontournable à la parallélisation de la rétroprojection.

En conclusion, le cache 3D-AP permet de délivrer un débit mémoire important à une

architecture parallèle malgré le goulot d'étranglement que constitue l'accès à la mémoire externe. De plus, si dans cette étude ce cache a été utilisé pour la seule application de rétroprojection, nous avons de bonnes raisons de penser que son utilisation serait également pertinente pour d'autres applications. Toutes les applications possédant une assez forte localité spatiale et temporelle intrinsèque et nécessitant un parcours mémoire continu dans un espace de données n dimensionnel, sont éligibles à l'emploi du cache nD-AP. En effet, si une certaine sensibilité de l'architecture au paramétrage du cache 3D-AP peut être observée, le cache 3D-AP permet à l'architecture 3P d'atteindre une bonne efficacité de reconstruction avec un seul et même jeu de paramètres, et cela malgré les différents parcours mémoire associés à une rétroprojection 3D complète (tous les segments et tous les blocs de voxels).

Dans le chapitre suivant, nous comparons le temps de reconstruction obtenu avec notre architecture 3P avec ceux obtenus avec des processeurs classiques et des processeurs graphiques. Nous évaluerons également l'efficacité de reconstruction de chaque architecture.

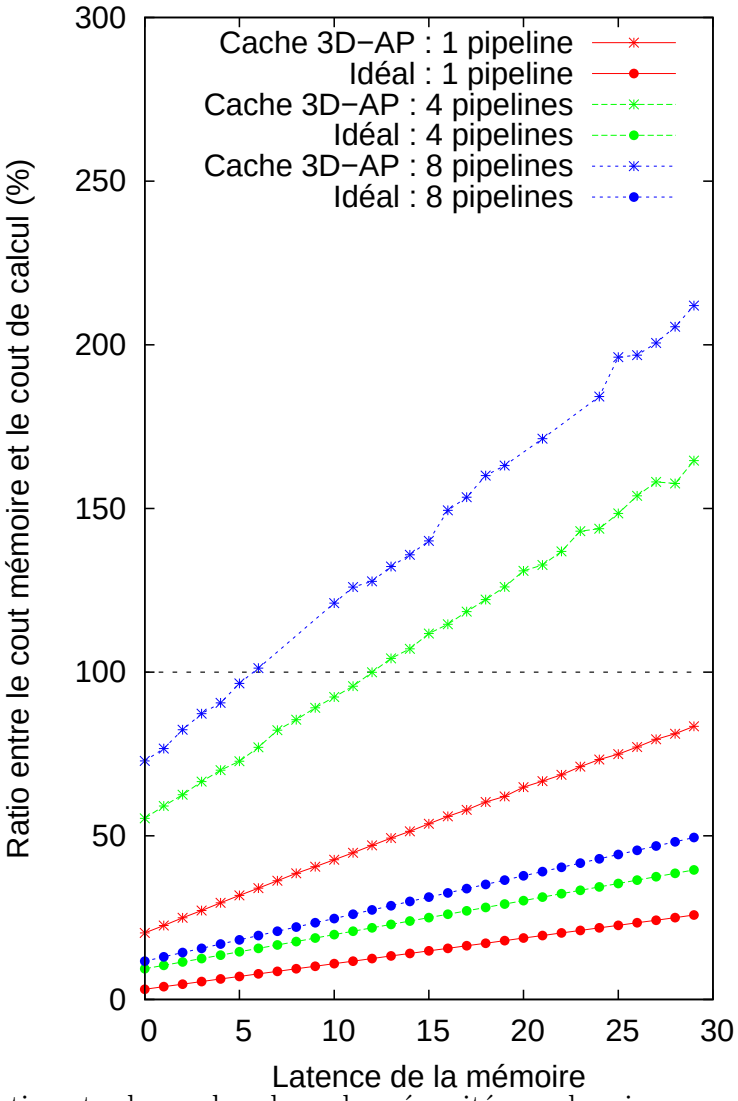


FIG. 4.22 – Ratio entre le nombre de cycles nécessités par la mise en cache racine de la sinusoïde 3D du macro-bloc reconstruit et le nombre de cycles de calcul d'une architecture parallèle fonctionnant à plein régime

Troisième partie

Etude comparative et intégration dans un système complet

Chapitre 5

Etude comparative sur CPU, GPU et FPGA

Ce chapitre a pour but de comparer les performances de notre architecture 3P avec celles obtenues sur processeurs classiques et graphiques. Nous détaillerons tout d'abord les implémentations optimisées sur deux processeurs classiques (Pentium 4 et Xeon), puis celles sur un processeur graphique (Nvidia GTS 8800). Nous porterons une attention particulière sur les stratégies d'accès mémoire de ces implémentations, basées chacune sur l'exploitation de la localité spatiale et temporelle intrinsèque de la rétroprojection par blocs de voxels. Enfin nous comparerons le temps et l'efficacité de reconstruction de ces implémentations avec ceux de notre architecture 3P implémentée sur un SoPC.

1 Implémentation sur CPU

Dans cette section, après une brève présentation des caches mémoire du Pentium 4 et du Xeon dual core utilisés, nous évaluerons l'impact de l'introduction de localité spatiale et temporelle sur ces processeurs. Nous observerons pour cela l'importance de l'ordre des boucles et nous proposerons un modèle analytique du temps de rétroprojection sur CPU. Ensuite, nous présenterons les optimisations logicielles que nous avons utilisées afin de réduire le nombre d'opérations. Enfin, nous présenterons la parallélisation du code de rétroprojection pour son implémentation sur un bi-Xeon dual core.

1.1 Caches mémoire du Pentium 4 et du Xeon dual core

Le Pentium 4 utilisé, le Prescott 2M 640 (voir tableau 2.3 du chapitre 2) possède un cache L1 de donnée de 16 Ko et un cache L2 de 2 Mo. Le Xeon dual core utilisé, le Woodcrest 5160 (voir tableau du 2.3 chapitre 2) possède 32 Ko de cache L1 par coeur et 4 Mo de cache L2 partagé par les deux coeurs. Nous donnons dans le tableau 5.1 les caractéristiques des caches de ces deux processeurs. Nous pouvons notamment remarquer la latence importante du cache L2 du Pentium 4 qui est un point faible de ce processeur.

Processeur (architecture)	Cache	Taille	*way associative*	Ligne de cache (octets)	Latence (cycles)
P4	L1	16 Ko	4	64 octets	3
(Netburst)	L2	2 Mo	8	128	28
Xeon dual core	L1	32 Ko/coeur	8	64	3
(Core)	L2	4 Mo partagé	16	64	14

TAB. 5.1 – Caractéristiques des caches mémoire du Pentium 4 (Prescott 2M [Mepham 04]) et du Xeon (Woodcrest 5160 [Gelas 06]) utilisés

1.1.1 Mémoire associative

Les caches des processeurs classiques sont N-associatifs. A une ligne de la mémoire globale correspond un emplacement dans chacun des N bancs mémoire. Ainsi afin de vérifier si une données est en cache, ces caches effectuent N comparaisons d'index mémoire (les *tags*), une dans chaque banc mémoire.

Les caches associatifs sont un bon compromis entre les caches à accès directs et les caches entièrement associatifs. Les premiers offrent une faible latence d'accès (puisque l'accès est direct sans comparaison), mais possède un fort risque de conflit (puisqu'à une ligne mémoire correspond un seul emplacement dans le cache). Les deuxièmes limitent les conflits (puisqu'une ligne mémoire peut être à n'importe quel emplacement mémoire), mais possèdent une latence importante (puisqu'il faut comparer le *tag* de la donnée demandée aux *tags* de toutes les lignes du cache).

1.1.2 Mécanismes de préchargements

Les mécanismes de préchargement des processeurs Intel sont de deux types : logiciels ou matériels. Dans le premier cas, c'est le programmeur qui peut grâce à des instructions spéciales précharger des données dans le cache L1 ou L2 (instructions *prefetchT1* et *prefetchT2*). Dans le deuxième cas, un préchargement automatique est déclenché selon un modèle prédéfini de prédiction par des modules matériels.

Les processeurs Intel de la 7ième génération (architecture NetBusrt) possèdent des mécanismes *hardware* de préchargement de données dans les caches L1 et L2 avec un horizon de prédiction très restreint. Seules les quelques lignes mémoire adjacentes à une ligne mémoire en cache peuvent être préchargées dans le cas où cette dernière est souvent accédée. Cela limite à 512 octets le champ d'action du *preftecher hardware* du Pentium 4.

Les processeurs de la 8ième génération comme le Xeon Woodcrest, possèdent un mécanisme de préchargement matériel plus évolué, l'*Intel Smart Memory Access* ([Doweck]). En plus du préchargement des lignes mémoire adjacentes à une ligne mémoire en cache souvent accédée (*prefetcherDCU* et *DPL*), ces processeurs possèdent un *IP prefetcher* dont le champ de prédiction touche l'ensemble de l'espace mémoire. Ce prefetcher enregistre l'historique de chaque instruction LOAD exécutée, indentifiée par le pointeur d'instruction ou IP (Instruction Pointer) dans une table. Puis à partir de la distance

entre les adresses mémoire des données précédemment demandées par une instruction LOAD (*constant stride*), les adresses mémoire des données potentiellement demandées aux prochaines exécutions par cette même instruction LOAD sont calculées. Le préchargement de ces données est ensuite effectué dans le cache L1 (le cache L2 n'est pas concerné par ce mécanisme de préchargement). Par ailleurs, ce préchargement des données n'est enclenché seulement quand le bus mémoire n'est pas engorgé. Cela permet de rendre prioritaire les accès mémoire réellement utiles liés à l'exécution du programme par rapport à ceux potentiellement utiles liés au mécanisme de prédiction.

1.1.3 Prédictions 1D, 2D et 3D lors de la rétroprojection

Lors de la rétroprojection, le Pentium 4 pourra précharger les bins adjacents dès le premier défaut observé sur une ligne de projection $\boldsymbol{l}_{\texttt{rho}}$. Ce préchargement s'effectue grâce (i) au chargement par ligne de cache (64 octets), et (ii) au préchargement des lignes de cache adjacentes si la ligne courante est souvent utilisée. Cela constitue une prédiction 1D (passage de `rho` à $\texttt{rho} + \mathbf{1}$). Par contre, il ne pourra pas précharger les lignes de projection $\boldsymbol{l}_{\boldsymbol{\rho}}$ d'un plan de projection $\boldsymbol{l}_{\boldsymbol{\rho}} \cdot \boldsymbol{L}_{\boldsymbol{\lambda}}$. En effet, la distance entre deux lignes de projection est supérieure à 512 octets. Cela ne lui permet donc pas d'effectuer cette prédiction 2D (passage de `lambda` à $\texttt{lambda} + \mathbf{1}$). De même, il ne pourra pas précharger le prochain plan de projection correspondant à l'angle $\texttt{phi} + \mathbf{1}$, autrement dit il ne pourra pas effecter de prédiction 3D (passage de `phi` à $\texttt{phi} + 1$).

Pour le Xeon, les prédictions 2D et 3D semblent difficiles à effectuer malgré l'*IP prefetcher*. En effet, les distances entre les dernières adresses accédées par l'instruction LOAD correspondront le plus souvent aux distances entre deux bins d'une même ligne de projection. Elles correspondent plus rarement à la distance entre deux lignes de projection consécutives, lorsqu'un changement de lignes de projection intervient. Afin d'effectuer les prédictions 2D et 3D, le cache doit ainsi faire la distinction entre les distances courtes (passage de `rho` à $\texttt{rho} + \mathbf{1}$) et les distances longues (passage de `lambda` à $\texttt{lambda} + \mathbf{1}$) et très longues (passage de `phi` à $\texttt{phi} + \mathbf{1}$).

Expérimentalement, nous avons observé l'absence de prédiction 3D sur le Xeon. Nous avons comparé pour cela une rétroprojection par bloc de voxels avec une séquence aléatoire de la boucle `phi` ($\texttt{phi} = \mathbf{7, 43, 91, 2...}$) et une autre avec une séquence ordonnée ($\texttt{phi} = \mathbf{0, 1, 2...}$). Cette dernière séquence devrait permettre une prédiction 3D. Or nous n'avons pas observé de différence en terme de temps de reconstruction. L'*IP prefetcher* échoue donc à prédire les déplacements selon la troisième dimension `phi`. Cela nous laisse supposer qu'il n'existe également pas de prédiction portant sur la deuxième dimension `lambda`.

1.2 Introduction de localité spatiale et temporelle

Comme nous l'avons vu au chapitre 3, la rétroprojection par bloc de voxel permet d'augmenter sensiblement la localité spatiale et temporelle. Pour mesurer l'impact de l'introduction de la localité spatiale et temporelle, nous avons implémenté sur un Pentium 4, deux versions du code de rétroprojection (voir les algorithmes 3 et 4). Le premier

possède une boucle externe sur tous les voxels du volume et une boucle interne sur les angles de projection (selon `delta` et `phi`). Le deuxième effectue la reconstruction par bloc de voxels de taille $\boldsymbol{Bx \cdot By \cdot Bz}$, chaque reconstruction de bloc possède alors une boucle externe sur les angles de projections (selon `delta` et `phi`) et une boucle interne sur les voxels du bloc. A noter que les codes v1 et v2 présentés utilisent des tableaux de constantes (`T0cos`, `T0sin`...) indexés par `phi`.

Algorithm 3 Code v1 de rétroprojection 3D

```
for zn = zn_0 to zn_0 + Bz − 1 do
  for yn = yn_0 to yn_0 + By − 1 do
    for xn = xn_0 to xn_0 + Bx − 1 do
      S = 0 // Mise à zéro de la somme S
      for delta = 0 to delta_max − 1 do
        S_delta = 0 // Mise à zéro de la somme partielle S_delta
        for phi = 0 to phi_max − 1 do
          // Calcul des coordonnées
          rho = xn · T0cos[phi] + yn · T0sin[phi] + rho_0
          lambda = zn + xn · T1sin[phi] + yn · T1cos[phi] + lambda_0
          // Calcul des coefficients d'interpolation
          ε_ρ = ...
          ε_λ = ...
          // Interpolation bilinéaire
          bin_interp = ...
          // Accumulation dans la somme partielle S_delta
          S_delta += bin_interp
        end for
        // Accumulation dans la somme S
        S += J_Δ · S_delta
      end for
    end for
  end for
end for
```

Algorithm 4 Code v2 de rétroprojection 3D

```
for delta = 0 to delta_max − 1 do

  // Mise à zéro du tableau S_delta
  for n = 0 to n_max − 1 do
    S_delta[n] = 0
  end for

  for phi = 0 to phi_max − 1 do
    n = 0 // Mise à zéro du compteur n
    for zn = zn_0 to zn_0 + Bz − 1 do
      for yn = yn_0 to yn_0 + By − 1 do
        for xn = xn_0 to xn_0 + Bx − 1 do
          // Calcul des coordonnées
          rho = xn · T0cos[phi] + yn · T0sin[phi] + rho_0
          lambda = zn + xn · T1sin[phi] + yn · T1cos[phi] + lambda_0
          // Calcul des coefficients d'interpolation
          ε_ρ = ...
          ε_λ = ...
          // Interpolation bilinéaire
          bin_interp = ...
          // Accumulation dans le tableau de somme partielle S_delta
          S_delta[n] += bin_interp
          n++ // Incrémentation du compteur n
        end for
      end for
    end for
  end for

  // Accumulation dans le tableau de somme S
  for n = 0 to n_max − 1 do
    S[n] = J_Δ · S_delta[n]
  end for
end for
```

1.3 Impact de l'ordre des boucles sur le temps de reconstruction

Afin d'évaluer l'impact de l'ordre des boucles sur le temps de reconstruction, nous comparons ici le temps de reconstruction obtenu pour la reconstruction 3D d'un volume $\mathbf{128^2 \cdot 63}$ avec les codes v1 et v2.

Sur le Pentium 4 qui ne possède pas de mécanisme de prédiction 2D, la mise en cache d'une ligne de projection est enclenché dès que les premiers voxels nécessitant cette ligne de projection en font la requête. Les voxels adjacents à ces premiers voxels peuvent ensuite accéder en cache à cette ligne de projection, à condition bien sûr que la ligne de projection soit encore en cache. Pour le code v2, les voxels adjacents aux premiers voxels (c'est-à-dire les voxels du blocs) accèdent à une ligne de projection juste après qu'elle soit mise en cache lors des requêtes des premiers voxels. En effet, la boucle selon les voxels est la plus interne. Pour le code v1, il faut attendre que les premiers voxels aient fait leur parcours mémoire en entier (boucle selon delta et phi) pour que les voxels adjacents puissent demander au cache une ligne de projection. Ainsi la localité temporelle du code v1 est dépendante de la longueur de la boucle selon les angles de projection (selon `phi` et `delta`). C'est ce que nous pouvons observer sur le graphe 5.1 qui représente le temps de reconstruction des codes v1 et v2 en fonction de la longueur de la boucle sur les angles (selon delta et phi). Nous avons ramené le temps de reconstruction au nombre de cycles nécessaire à la mise à jour d'un voxel. Le code v2 a ainsi une efficacité de reconstruction de 112 cycles/op qui est comme attendu insensible à la taille de la boucle en phi et delta. Afin d'exploiter la localité spatiale et temporelle lors de l'exécution de ce code, le cache mémoire doit être capable de contenir en entier un plan de projection. Or pour un bloc de $\mathbf{16^2 \cdot 3}$ voxels et pour des bins codés sur 4 octets, cela représente une taille moyenne de 400 octets. Dans ce but, le cache L1 suffit amplement.

Par ailleurs, pour le code v1 nous avons mesuré et représenté toujours sur le graphe 5.1, le temps de reconstruction pour différents ordres des boucles x,y et z. La boucle xyz désigne la boucle sur les voxels avec la boucle la plus externe selon x et celle plus interne interne selon z. Afin d'éliminer la localité spatiale et temporelle due à l'interpolation selon la dimension $\boldsymbol{\lambda}$, nous avons également mesuré le temps de reconstruction lorsque la boucle sur z se fait avec un pas de deux (z+=2). Nous désignons cette boucle par z#. Ainsi la boucle xyz#|angle génère le plus long laps de temps entre les requêtes de deux voxels adjacents possédant des données de projection communes. Ce laps de temps est d'ailleurs trop grand : lorsque la deuxième requête se produit le cache ne contient plus la donnée. Une boucle en zyx|angle permet de meilleures performances lorsque la boucle selon les angles de projection est réduite. Mais ces performances se dégradent graduellement lorsque la longueur de la boucle en `phi` et `delta` augmente. En effet, plus cette boucle s'allonge, plus le volume de données correspondant au parcours mémoire d'un voxel est grand. Et lorsque ce volume de données dépasse les capacités du cache L1 (16 Ko) puis celle du cache L2 (2 Mo), les données de projection demandées par le voxel suivant ne seront plus en cache. Ainsi pour une boucle selon `phi` et `delta` de longueur importante, la boucle en zyx|angle atteint la même efficacité de reconstruction (370 cycles/op) que la boucle en xyz#|angle.

Au final, l'introduction de localité spatiale et temporelle avec une reconstruction par

bloc de voxels permet un facteur 3 d'accélération sur un Pentium 4 par rapport à une reconstruction sans bloc.

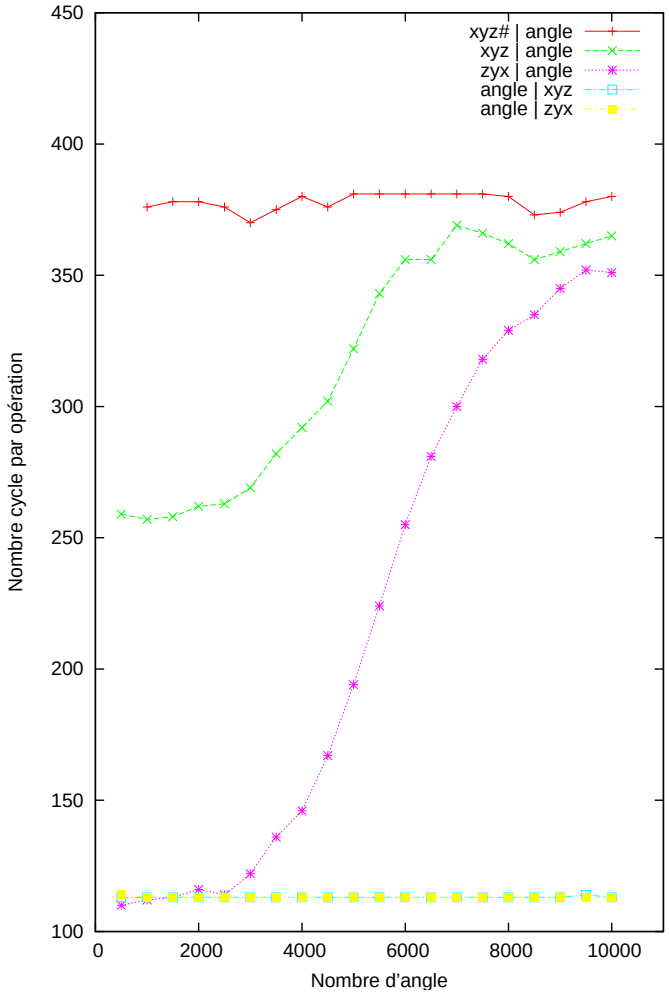


FIG. 5.1 – Temps de reconstruction sur Pentium 4 en fonction de la taille de la boucle en phi pour différents ordres des boucles de calcul

1.4 Modèle analytique du temps de rétroprojection sur Pentium 4

Lorsqu'un cache mémoire 1D sans préchargement de donnée comme celui du Pentium 4 est utilisé le coût mémoire important des premiers accès en SDRAM est amorti par la suite grâce aux accès ultérieurs en cache. C'est ce principe que nous avons traduit de manière analytique dans l'équation 5.1. Cette équation exprime l'efficacité moyenne (en cycles/op) lors d'une reconstruction d'un bloc de taille $\boldsymbol{T_{bloc}^2 \cdot 1}$ pour le segment 0 (cela correspond à une rétroprojection 2D). Pour chaque angle `phi`, la rétroprojection

nécessite t_{op} cycles de calcul et N_{bin} accès mémoire. Les accès aux données se font soit via la mémoire externe par ligne mémoire (64 octets mis en cache) avec une latence t_{SDRAM}, soit via le cache avec une latence t_{cache}.

$$t_{CPU/op} = \frac{N_{ligne} \cdot t_{SDRAM} + (N_{bin} - 1) \cdot t_{cache}}{N_{bin}} + t_{op}$$

avec :

$$\begin{cases} t_{SDRAM} & : \text{temps pour obtenir une ligne de données (64 octets) depuis la mémoire SDRAM} \\ t_{cache} & : \text{temps pour obtenir une donnée en cache} \\ t_{op} & : \text{temps de calcul nécessaire pour la mise à jour des voxels d'un bloc} \\ N_{ligne} & : \text{nombre de lignes mémoire mis en cache (64 octets)} \\ N_{bin} & : \text{nombre de bins demandés lors de la rétroprojection} \end{cases} \quad (5.1)$$

Nous avons pris pour latence du cache celle du cache L1 ($t_{cache} = 3$ cycles) car comme nous avons vu précédemment c'est ce cache qui est utilisé lors de la rétroprojection par bloc. Nous avons estimé le nombre moyen de *bins* nécessaires à $N_{bin} = T_{bloc} \cdot \frac{1+\sqrt{2}}{2}$. En effet, la projection d'un carré T_{bloc}^2 est une ligne de longueur comprise entre T_{bloc} et $\sqrt{2} \cdot T_{bloc}$ (diagonale du carré). Nous avons estimé le nombre de lignes mémoire accédées à l'aide d'une régression linéaire. Ainsi sans interpolation bilinéaire, une ligne de projection constituée d'un seul *bin* requiert l'accès à une seule ligne mémoire, et une ligne de projection de 16+1=17 *bins* (64+2=66 octets) requiert l'accès à deux lignes mémoire (2*64 octets). Pour la rétroprojection avec interpolation bilinéaire, nous avons multiplié par deux le nombre de lignes mémoire accédées. En effet, il faut alors accéder à deux plans de projection à cause de l'interpolation selon la dimension `lambda`. Nous avons ensuite calculé la latence de la mémoire externe t_{SDRAM} et le temps de calcul t_{op} à partir de deux mesures (pour un bloc d'un voxel et un bloc de 1024 voxels). Nous obtenons ainsi sans interpolation bilinéaire pour t_{SDRAM} 111 cycles et pour t_{op} 46 cycles, et avec interpolation bilinéaire pour t_{SDRAM} 121 cycles et t_{op} 104 cycles. Cela correspond à une latence mémoire de 35 ns, temps représentatif des latences des mémoire SDRAM. Sur le graphe 5.2, nous confrontons notre modèle analytique aux mesures avec et sans interpolation bilinéaire. Nous pouvons ainsi constater sur ce graphe que le modèle analytique correspond bien à la réalité des mesures.

Par ailleurs, nous pouvons estimer que lorsque la localité spatiale et temporelle de l'algorithme de rétroprojection est bien exploitée, le ratio entre le coût d'accès mémoire (9 cycles) sur le coût de calcul (104 cycles) est de 8,8 % (code v2 avec interpolation bilinéaire). Le fort poids du temps de calcul sur le temps total de reconstruction, nous invite à concentrer nos efforts d'optimisations logicielles sur la réduction du nombre d'opérations.

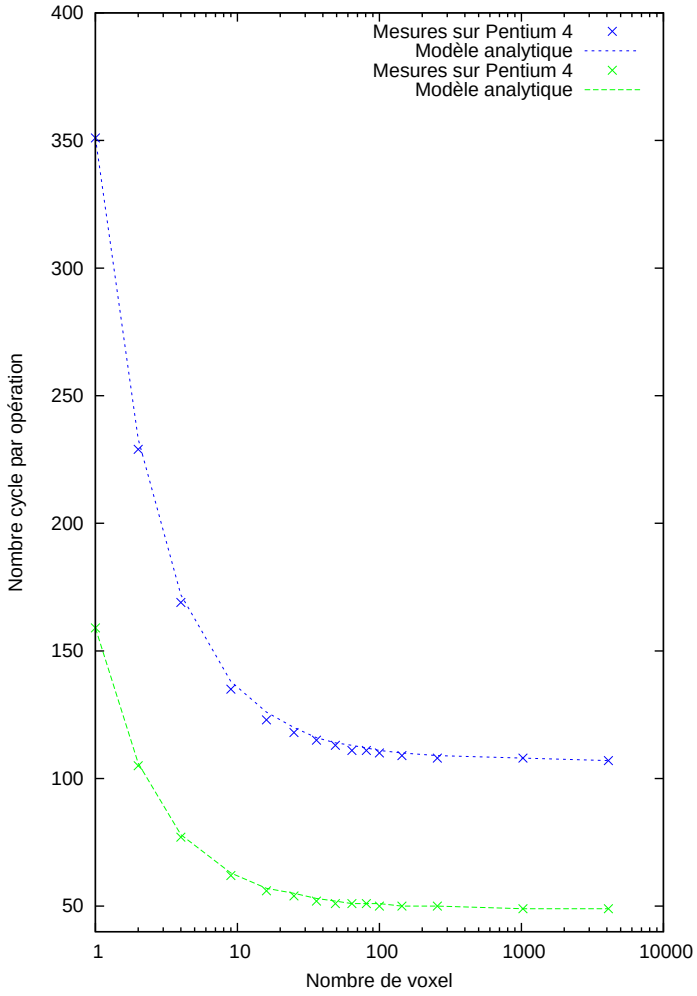


FIG. 5.2 – Temps de reconstruction sur Pentium 4 en fonction de la taille des blocs de voxels reconstruits

1.5 Réduction du nombre d'opérations

Afin d'accélérer le temps de reconstruction sur CPU, nous avons tenter de réduire au maximum le nombre d'opérations nécessaires au calcul des coordonnées de projection `rho` et `lambda` et des coefficients d'interpolation. En effet, la rétroprojection possède en plus d'une forte localité spatiale et temporelle des données accédées, une forte localité de calcul : les calculs de voxels adjacents sont très proches et donc factorisables. Le code v3 correspond à cet effort d'optimisation logicielle. Nous nous sommes tout d'abord inspiré de l'algorithme incrémentiel de [Cho 90] utilisé également par le rétroprojecteur de STIR. De plus, nous avons modifié l'ordre des boucles selon les voxels (`xn`, `yn` et `zn`). En effet, une boucle sur les voxels du bloc en xyz (boucle la plus externe en x et la boucle la plus interne en z) permet de factoriser plus de calculs. Ainsi, le calcul de `rho`, des coefficients

d'interpolation et en partie de celui de `lambda` est effectué une seule fois pour une colonne de voxels ($\texttt{zn} = \mathbf{0}\ldots\texttt{zn}_{\texttt{max}}$). Nous avons également utilisé des déroulages de boucles pour masquer les temps d'exécution des instructions LOAD.

Par ailleurs, nous avons testé l'utilisation des instructions vectorielles de type SSE (vecteur de 4 flottants soit 128 bits) pour effectuer les multiplications des bins avec les coefficients d'interpolation. Le résultat n'a pas été concluant. En effet, ces instructions nécessitent un réalignement des données sur 128 bits. Or les données de projection accédées en mémoire sont ordonnées selon les coordonnées de projection, et sont selon les cas alignées ou non sur 128 bits. Il faut donc au préalable transférer ces données dans des registres 128 bits. Du coup, cela rend moins efficace l'emploi des instructions vectorielles sur CPU.

1.6 Parallélisation

Nous avons parallélisé le code v3 sur le serveur de calcul disponible au laboratoire Gipsa-lab, un bi-Xeon dual core. Nous avons pour cela utilisé la librairie *pthread* qui permet de définir en C des *threads*. Nous avons fait correspondre à chaque *thread* la reconstruction d'un bloc de voxels et positionné une barrière de synchronisation après la reconstruction de $\boldsymbol{n_{proc}}$ blocs par les $\boldsymbol{n_{proc}}$ threads.

Algorithm 5 Code v3 de rétroprojection 3D

```
for delta = 0 to delta_max − 1 do

  // Mise à zéro du tableau S_delta
  for n = 0 to n_max − 1 do
    S_delta[n] = 0
  end for

  for phi = 0 to phi_max − 1 do
    n = 0 // Mise à zéro du compteur n
    // Initialisation de xcos et xsin
    xcos = (xn_0 − 1) · T0cos[phi]
    xsin = (xn_0 − 1) · T1sin[phi]

    for xn = xn_0 to xn_0 + Bx − 1 do
      // Incrémentation de xcos et xsin
      xcos = xcos + T0cos[phi]
      xsin = xsin + T1sin[phi]
      // Initialisation de rho et lambdaxy
      rho = xcos + (yn_0 − 1) · T1cos[phi] + rho_0
      lambdaxy = xsin + (yn_0 − 1) · T0sin[phi] + lambda_0

      for yn = yn_0 to yn_0 + By − 1 do
        // Incrémentation de rho et lambdaxy
        rho+ = T0sin[phi]
        lambdaxy+ = T1cos[phi]
        // Initialisation de lambda
        lambda = lambdaxy + zn0
        // Calcul des coefficients d'interpolation
        ε_ρ= ...
        ε_λ= ...

        for zn = zn_0 to zn_0 + Bz − 1 do
          // Incrémentation de lambda
          lambda+ = lambda
          // Interpolation bilinéaire
          bin_interp= ... ;
          // Accumulation dans le tableau de somme partielle S_delta
          S_delta[n]+ = bin_interp
          n++ // Incrémentation du compteur n
        end for
      end for
    end for
  end for

  // Accumulation dans le tableau de somme S
  for n = 0 to n_max − 1 do
    S[n] = J_Δ · S_delta[n]
  end for
end for
```

2 Implémentation sur GPU

Nous avons utilisé l'interface CUDA pour implémenter la rétroprojection 3D sur le processeur graphique 8800 GTS de Nvidia. Afin de bénéficier de la localité géométrique intrinsèque de l'algorithme de rétroprojection 3D, nous avons défini une grille de *threads* correspondant à une reconstruction par blocs de voxels. Dans cette grille, un *thread* correspond à la rétroprojection d'une colonne de voxels ($\texttt{xn,yn,zn} = \mathbf{0...}\texttt{zn}_{\texttt{max}} - \mathbf{1}$) et un bloc de *threads* à la rétroprojection d'un bloc de voxels de taille $\boldsymbol{B_{xy}^2} \cdot \mathbf{1}$. L'algorithme 6 présente une version du code (code v4) correspondant à cette stratégie de parallélisation.

A l'exécution, la reconstruction du volume se fait en parallèle sur les $\boldsymbol{n}$ processeurs vectoriels (il y en a 12 sur le GTS 8800), chacun traitant un bloc de voxels. Et la reconstruction des voxels d'un bloc se fait en parallèle sur les 8 processeurs élémentaires de chaque processeur vectoriel. L'ordonnancement des *threads* d'un bloc se fait automatiquement à l'exécution par le GPU. Les *threads* sont groupés par *warps* (32 *threads* au maximum) dont les *threads* s'exécutent parallèlement ou quasi parallèlement (pendant 4 cycles des unités de calcul pour un *warp* de 32 *threads*) sur les 8 processeurs élémentaires d'un processeur vectoriel. Un *scheduler* de *warps* du GPU permet d'alterner périodiquement les *warps* en train d'être exécutés. De cette manière l'exécution des différents *threads* d'un bloc parcourt la boucle en phi et z au même rythme. Ainsi les *threads* d'un bloc exécutent quasi-parallèlement la même opération d'accumulation (même `phi` et `zn` dans la boucle du thread) pour des voxels adjacents appartenant à une même coupe 2D du volume. Grâce à cet ordonnancement de *threads* et grâce au cache 2D de texture contenant les données de projection, le GPU pourra ainsi bénéficier pleinement de la même localité spatiale et temporelle des accès mémoire que celle mise en place sur notre architecture 3P et sur CPU. Concernant les mécanismes éventuels de préchargement de données du cache 2D de texture, nous n'avons pas trouvé d'informations de la part du concepteur Nvidia, nous permettant d'affirmer qu'il en possède. Nous pouvons seulement le supposer.

Les textures étant 2D et un segment du sinogramme 3D, nous avons rangé les données de projection selon le calcul d'adresse mémoire suivant :

$$\boldsymbol{@texture} = (\texttt{rho}, \texttt{lambda} + \texttt{phi} * (\texttt{lambda}_{\boldsymbol{max}} - \mathbf{1})) \tag{5.2}$$

Ainsi les interpolateurs 2D câblés en *hardware*, seront utilisés lors de l'étape d'interpolation bilinéaire selon les coordonnées `rho` et `lambda`.

Pour la reconstruction d'un volume $\mathbf{128^2 \cdot 63}$ voxels, nous avons défini $\mathbf{8^2}$ blocs de threads chacun de taille $\mathbf{16^2}$. Chaque thread effectue la rétroprojection de 63 voxels. Afin de réduire le nombre de calculs, nous avons inversé les boucles en `phi` et `zn` (code v5 correspondant à l'algorithme 7). De cette manière, le calcul de `rho` et celui partiel de `lambda` pour chaque angle `phi` se font une seule fois pour toute la colonne de voxel du *thread*.

Algorithm 6 Définition d'un thread pour la rétroprojection 3D sur GPU (code v4)

void Recon2D (float *volume, ...) {
// On obtient le numéro du bloc
$\boldsymbol{i_{bloc} = blockIdx.x}$
$\boldsymbol{j_{bloc} = blockIdx.y}$
// On positionne le bloc dans l'espace 3D
$\mathtt{xn}_{\mathtt{debut}} = \boldsymbol{i_{bloc} * blockDim.x} + \mathtt{xn}_0$
$\mathtt{yn}_{\mathtt{debut}} = \boldsymbol{j_{bloc} * blockDim.y} + \mathtt{yn}_0$
// On identifie les coordonnées transverses des voxels reconstruits
$\mathtt{xn} = \mathtt{xn}_{\mathtt{debut}} + \boldsymbol{threadIdx.x}$
$\mathtt{yn} = \mathtt{yn}_{\mathtt{debut}} + \boldsymbol{threadIdx.y}$

for $\mathtt{zn} = \mathtt{zn}_0$ to $\mathtt{zn}_0 + \mathtt{zn}_{\mathtt{max}} - \mathbf{1}$ **do**
  for $\mathtt{phi} = \mathbf{0}$ to $\mathtt{phi}_{\mathtt{max}} - \mathbf{1}$ **do**
    $\mathtt{rho} = \mathtt{xn} * \mathtt{T0cos[phi]} + \mathtt{yn} * \mathtt{T0sin[phi]} + \mathtt{rho}_0$
    $\mathtt{lambda} = \mathtt{zn} + \mathtt{xn} * \mathtt{T1sin[phi]} + \mathtt{yn} * \mathtt{T1cos[phi]} + \mathtt{lambda}_0$
    $\mathtt{volume}[\boldsymbol{xn, yn, zn}] = \boldsymbol{J_{\Delta} \cdot tex2D(tex0}, \mathtt{rho}, \mathtt{lambda} + \mathtt{phi} * \mathtt{lambda}_{\mathtt{max}})$
  end for
end for
}

Algorithm 7 Définition optimisée d'un thread pour la rétroprojection 3D sur GPU (code v5)

void Recon2D (float *volume, ...) {
// On obtient le numero du bloc
$\boldsymbol{i_{bloc} = blockIdx.x}$
$\boldsymbol{j_{bloc} = blockIdx.y}$
// On positionne le bloc dans l'espace 3D
$\mathtt{xn}_{\mathtt{debut}} = \boldsymbol{i_{bloc} * blockDim.x} + \mathtt{xn}_0$
$\mathtt{yn}_{\mathtt{debut}} = \boldsymbol{j_{bloc} * blockDim.y} + \mathtt{yn}_0$
// On identifie les coordonnées transverses des voxels reconstruits
$\mathtt{xn} = \mathtt{xn}_{\mathtt{debut}} + \boldsymbol{threadIdx.x}$
$\mathtt{yn} = \mathtt{yn}_{\mathtt{debut}} + \boldsymbol{threadIdx.y}$

for $\mathtt{phi} = \mathbf{0}$ to $\mathtt{phi}_{\mathtt{max}} - \mathbf{1}$ **do**
  $\mathtt{rho} = \mathtt{xn} * \mathtt{T0cos[phi]} + \mathtt{yn} * \mathtt{T0sin[phi]} + \mathtt{rho}_0$
  $\mathtt{lambda} = (\mathtt{zn}_0 - \mathbf{1}) + \mathtt{xn} * \mathtt{T1sin[phi]} + \mathtt{yn} * \mathtt{T1cos[phi]} + \mathtt{lambda}_0$
  for $\mathtt{zn} = \mathtt{zn}_0$ to $\mathtt{zn}_0 + \mathtt{zn}_{\mathtt{max}} - \mathbf{1}$ **do**
    $\mathtt{lambda} += \mathbf{1}$
    $\mathtt{volume}[\boldsymbol{xn, yn, zn}] += \boldsymbol{J_{\Delta} \cdot tex2D(tex0}, \mathtt{rho}, \mathtt{lambda} + \mathtt{phi} * \mathtt{lambda}_{\mathtt{max}})$
  end for
end for
}

3 Performances sur CPU/GPU/FPGA

3.1 Temps de reconstruction

Le tableau 5.2 présente les temps de rétroprojection 3D d'un volume $\mathbf{128^2 \cdot 63}$ mesurés sur un PC (Pentium 4), un serveur de calcul (bi-xeon dual core), un GPU (Nvidia 8800 GTS) et sur un SoPC (Virtex 2 Pro). Les données de projection utilisées correspondent à un sinogramme corrigé en arc d'un scanner HR+ (5 segments, 96 angles de projection, 269 coordonnées tangentielles). A partir de ce temps nous avons calculé, l'efficacité de reconstruction en cycles par opération élémentaire, c'est-à-dire par mise à jour de voxel.

Afin de comparer de manière équitable la technologie FPGA aux autres technologies, nous avons transposé les résultats obtenus sur le Virtex 2 Pro (fréquence de fonctionnement de 35 Mhz) sur un Virtex 4. En effet, la technologie du Virtex 4 est de la même génération que celle des CPUs et GPU utilisée dans cette étude. Ainsi avec un Virtex 4 et quelques améliorations de l'architecture du système, nous pouvons penser qu'une fréquence de fonctionnement de 200 MHz pourrait être atteinte. Nous avons également transposé les résultats obtenus sur le Virtex 2 Pro avec ceux que l'on obtiendrait avec la conception d'un circuit intégré de type ASIC. Nous avons évalué à 1.2 Ghz la fréquence d'un circuit ASIC basé sur notre architecture 3P. Cela correspond à la même fréquence de fonctionnement que la fréquence du *shader* du GTS 8800 de Nvidia. Les résultats à 200 Mhz et à 1.2 Ghz correspondent à ceux mesurés sur le Virtex 2 Pro avec une latence mémoire du simulateur de bus respectivement de 5 et 30 cycles, soit 25 ns.

Par ailleurs, nous avons également projeté les résultats obtenus sur Virtex 2 Pro avec un seul banc mémoire à ceux que l'on obtiendrait avec le même nombre de bancs mémoire que le GTS 8800, c'est-à-dire 5. Or puisque le débit mémoire serait alors multiplié par 5, une architecture avec $\mathbf{5 \cdot 8}$ unités de reconstruction et 5 bancs mémoire, offrirait la même efficacité de reconstruction par pipeline que celle obtenue avec un seul banc mémoire.

3.2 Comparaison CPU/GPU/FPGA

Sur CPU, nous pouvons tout d'abord remarquer que les optimisations logicielles permettent un facteur 3 d'accélération grâce à la reconstruction par bloc et d'un facteur 7 grâce à la réduction du nombre d'opération (algorithme incrémentiel et traitement par colonnes de voxels). Nous pouvons constater qu'un simple coeur de Xeon à fréquence comparable ($\simeq$ 3 Ghz), est deux fois plus rapide que le Pentium 4. Par ailleurs, la parallélisation sur le serveur de calcul se passe parfaitement bien : l'efficacité de parallélisation est de $\boldsymbol{\eta_{/\!/} = 1}$ sur un Xeon dual core et de $\boldsymbol{\eta_{/\!/} = 1}$ pour un bi-Xeon dual core.

Sur GPU, l'optimisation logicielle permet un facteur 2 d'accélération et le GPU atteint alors une efficacité de reconstruction de 13 cycles/op/UT que nous pouvons juger correct. Le ratio entre le temps de calcul et le temps des accès mémoire est assez grand pour que le mécanisme d'ordonnancement des *threads* permette le masquage automatique des accès à la mémoire vidéo du GPU. Ce masquage est d'autant plus efficace que la latence de la mémoire vidéo est comprise entre 200 à 500 cycles (source Nvidia). Ainsi grâce à sa plus grande force de calcul (96 UTs et des interpolations en *hardware*),

le processeur 8800 GTS de Nvidia est 10 fois plus rapide que notre architecture 3P implémentée sur un Virtex 4 et qu'un Xeon dual core, et 50 fois plus rapide qu'un Pentium 4.

Avec notre architecture 3P, nous obtenons la meilleure efficacité de reconstruction par Unité de Traitement (UT) avec 1,7 cycles/op/UT sur un Virtex avec 8 unités de rétroprojection et un seul banc mémoire. Cette architecture est ainsi 4 fois plus efficace qu'un coeur de Xeon, 7.5 fois plus qu'un GTS 8800 et près de 10 fois plus qu'un Pentium 4. Mais à cause de sa plus faible fréquence de fonctionnement (200 Mhz) et de ses faibles ressources de calcul (moins d'unités de traitement qu'un GPU) et d'accès à la mémoire (1 seul banc mémoire), la technologie FPGA ne parvient pas à concurrencer efficacement les processeurs graphiques malgré sa meilleure efficacité de reconstruction. Cependant, un circuit ASIC avec des ressources mémoire et une fréquence comparable à celle du GTS 8800 (5 bancs mémoire et freq.=1.2 Ghz), permettrait d'être deux fois plus rapide (avec 40 pipelines) qu'un GTS 8800 (avec 96 processeurs élémentaires), 50 fois plus qu'un Xeon dual core et 100 fois plus qu'un Pentium 4.

Afin de rendre compte des différences de débits mémoire, nous avons positionné les architectures étudiées (CPU,GPU, FPGA et ASIC) sur la figure 5.3 en fonction de leur efficacité de reconstruction (cycles/op) et du débit mémoire dont elles disposent (en Go/s). Nous avons placé également une architecture dite idéale. Elle correspond à une architecture matérielle (ASIC ou FPGA) qui grâce à un mécanisme de préchargement et une balance optimale entre débit de calcul et débit mémoire, masquerait entièrement la latence des accès mémoire. Chaque UT de cette architecture idéale effectue 1 opération par cycle et chargerait en cache uniquement les données nécessaires ($\boldsymbol{\eta_{surcharge} = 1}$) et les délivreraient à temps pour l'UT. Bien entendu, plus le nombre d'UTs augmente, plus le débit mémoire doit être important pour conserver une efficacité de reconstruction de 1 cycle/op/UT. Comme nous pouvons le constater sur ce graphe, l'architecture 3P offre les performances les plus proches de l'architecture idéale.

Version logicielle/matérielle	**UT (threads)**	**Temps**	**Cycles/Op**	
			/UT	**total**
Station de travail : Pentium 4				
core freq.=3.2 Ghz,$\boldsymbol{d_{mem}}$=6.4 Go/s				
STIR[1]	1	11.13 s	70.4	70.4
v1	1	54,7 s	355	355
v2	1	17,4 s	113	113
v3	1	2,5 s	16	16
Serveur de calcul : bi-Xeon dual core				
core freq.=3 Ghz,$\boldsymbol{d_{mem}}$=10.6 Go/s				
STIR[1]	1 (1)	5.74 s	34.5	34.5
v3	1 (1)	1.17 s	7,1	7,1
v3	2 (2)	583 ms	7,06	3,53
v3	4 (4)	294 ms	7,12	1,78
GPU : GTS8800				
shader freq.=1.2 Ghz,$\boldsymbol{d_{mem}}$=64 Go/s				
v4	96 (192)	99 ms	25.9	0.27
v5	96 (192)	50 ms	13,0	0.14
FPGA : Virtex 2 Pro				
freq.=35 Mhz,$\boldsymbol{d_{mem}}$=140 Mo/s,$\boldsymbol{l_{mem}}$=5 cycles (142 ns)				
3P	1	14,1 s	1	1
3P	4	4,4 s	1,25	0,31
3P	8	3,0 s	1,7	0,21
FPGA[2] : Virtex 4				
freq.=200 Mhz,$\boldsymbol{d_{mem}}$=0.8 Go/s,$\boldsymbol{l_{mem}}$=5 cycles (25 ns)				
3P	1	2,5 s	1	1
3P	4	774 ms	1,25	0,31
3P	8	526 ms	1,7	0,21
ASIC[3] : un banc mémoire				
freq.=1,2 Ghz,$\boldsymbol{d_{mem}}$=4,8 Go/s,$\boldsymbol{l_{mem}}$=30 cycles (25 ns)				
3P	1	499 ms	1.21	1,21
3P	4	214 ms	2,07	0,517
3P	8	135 ms	2,62	0,328
ASIC[3] : cinq bancs mémoire				
freq.=1,2 Ghz,$\boldsymbol{d_{mem}}$=24 Go/s,$\boldsymbol{l_{mem}}$=30 cycles (25 ns)				
VBI-fix	40	27 ms	2.62	0,065

[1] Temps normalisé à un volume $\mathbf{128^2 \cdot 63}$.
(STIR reconstruit des volumes cylindriques, ici de taille $\mathbf{64^2 \cdot \pi \cdot 63}$)
[2] Performances @35 Mhz transposées @200 Mhz ($\boldsymbol{l_{mem}}$=5 cycles)
[3] Performances @35 Mhz transposées @1,2 Ghz ($\boldsymbol{l_{mem}}$=30 cycles)

TAB. 5.2 – Temps de reconstruction comparées pour la rétroprojection TEP 3D pour un volume $\mathbf{128^263}$ à partir d'un sinogramme HR+ (5 segments, *span* de 9, 96 angles de projection). L'efficacité de reconstruction (cycles par mise à jour de voxel) est présenté pour l'architecture globale et par Unité de Traitement (UT).

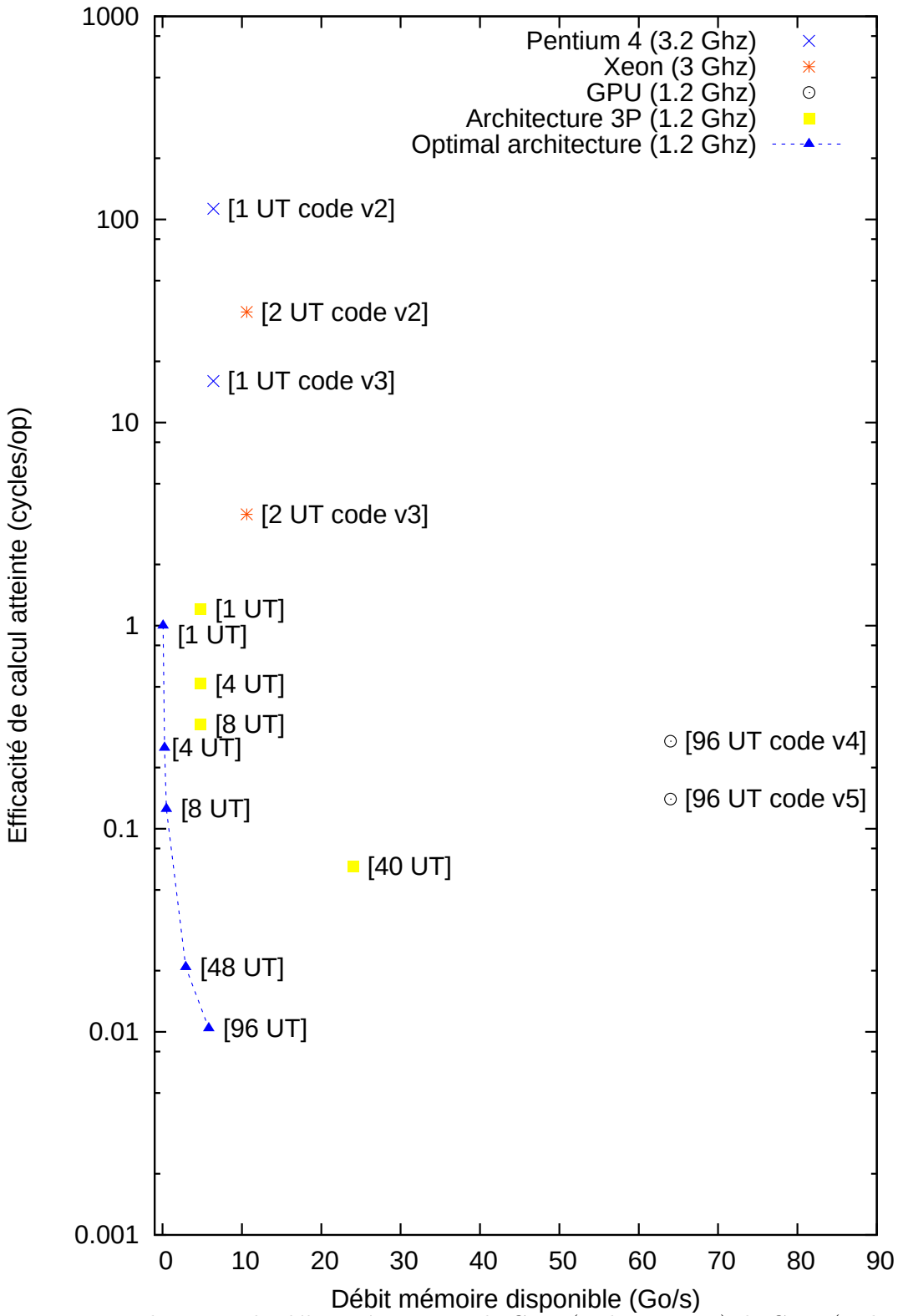


FIG. 5.3 – Exploitation du débit mémoire par le CPU (code v2 et v3), le GPU (code v4 et v5) et notre architecture 3P. Les résultats présentés pour les architectures idéale et 3P, sont obtenus avec une latence mémoire de 25 ns.

3.3 Comparaison entre le cache 3D-AP et le cache 1D du Pentium 4

Afin de comparer le comportement du cache 3D-AP et du cache 1D du Pentium 4 lors de l'exécution de la rétroprojection 3D, nous avons représenté sur la figure 2.2 l'efficacité de reconstruction mesurée avec un pipeline de rétroprojection nourri en données par un cache 3D-AP avec celle que l'on obtiendrait avec un pipeline de rétroprojection et un cache semblable à celui d'un Pentium 4.

Nous avons évalué les performances avec le cache 1D à l'aide de l'expression 5.1 de l'efficacité de reconstruction du pipeline ($\boldsymbol{\eta_{retro}}$) calculé pour la rétroprojection d'un bloc de voxels (un seul segment du sinogramme utilisé). Pour chaque angle `phi`, il faut faire une mise à jour pour chaque voxel ($\boldsymbol{N_{voxel}}$ opérations) et charger dans le cache le plan de projection correspondant au bloc de voxels. Nous avons pris un bloc de $\mathbf{16^2 \cdot 3}$ voxels, ce qui correspond en moyenne à une longueur du plan de projection de $\boldsymbol{L_\lambda = 5}$. Soit $\boldsymbol{N_{ligne}}(\text{plan}) = \mathbf{6}$ (+1 pour l'interpolation linéaire selon la dimension `lambda`). Par ailleurs, les lignes de cache contiennent 16 mots mémoire de 4 octets chacun et le débit $\boldsymbol{d_{mem}}$ est égal à 1 mot/cycle.

$$\boldsymbol{\eta_{retro}} = \frac{\boldsymbol{t}_{\text{calcul}} + \boldsymbol{t}_{\text{chargement du cache}}}{\boldsymbol{N_{op}}}$$

avec

$$\begin{cases} \boldsymbol{t}_{\text{calcul}} = \boldsymbol{N_{op}} \cdot \mathbf{1}\ \boldsymbol{cycle/op} = \boldsymbol{N_{voxel}} \cdot \boldsymbol{N}_{\texttt{phi}} \\ \boldsymbol{t}_{\text{chargement du cache}} = \boldsymbol{N}_{\texttt{phi}} \cdot \boldsymbol{N}_{\text{ligne}}(\text{plan}) \cdot \dfrac{\boldsymbol{N}_{\text{mot}}(\text{ligne de cache}) - \mathbf{1}}{\boldsymbol{d_{mem}}} \end{cases} \tag{5.3}$$

Pour des latences mémoire inférieures à 20 cycles, le cache 3D-AP permet des performances 10 à 20 % meilleures que le cache 1D. Le masquage de la latence mémoire grâce au mécanisme de préchargement du cache 3D-AP permet donc d'accélérer de manière efficace la rétroprojection.

Nous n'avons pas comparé ces deux caches lors de l'utilisation d'une architecture parallèle. Tout comme pour le cache 3D-AP, une architecture hiérarchique devrait être utilisée afin de réduire le nombre d'accès à la mémoire externe. Mais comme nous l'avons vu précédemment, lors de la parallélisation les conflits d'accès au cache racine par les caches feuilles dégradent de manière importante les performances. Or il est difficile de modéliser ces conflits. Ainsi l'estimation des performances obtenues avec un cache 1D hiérarchique n'a pas été effectuée. Toutefois, nous pouvons penser qu'un gain de performance du cache 3D-AP par rapport au cache 1D devrait être également observé sur une architecture parallèle grâce au préchargement des données de projection du macro-bloc dans le cache racine.

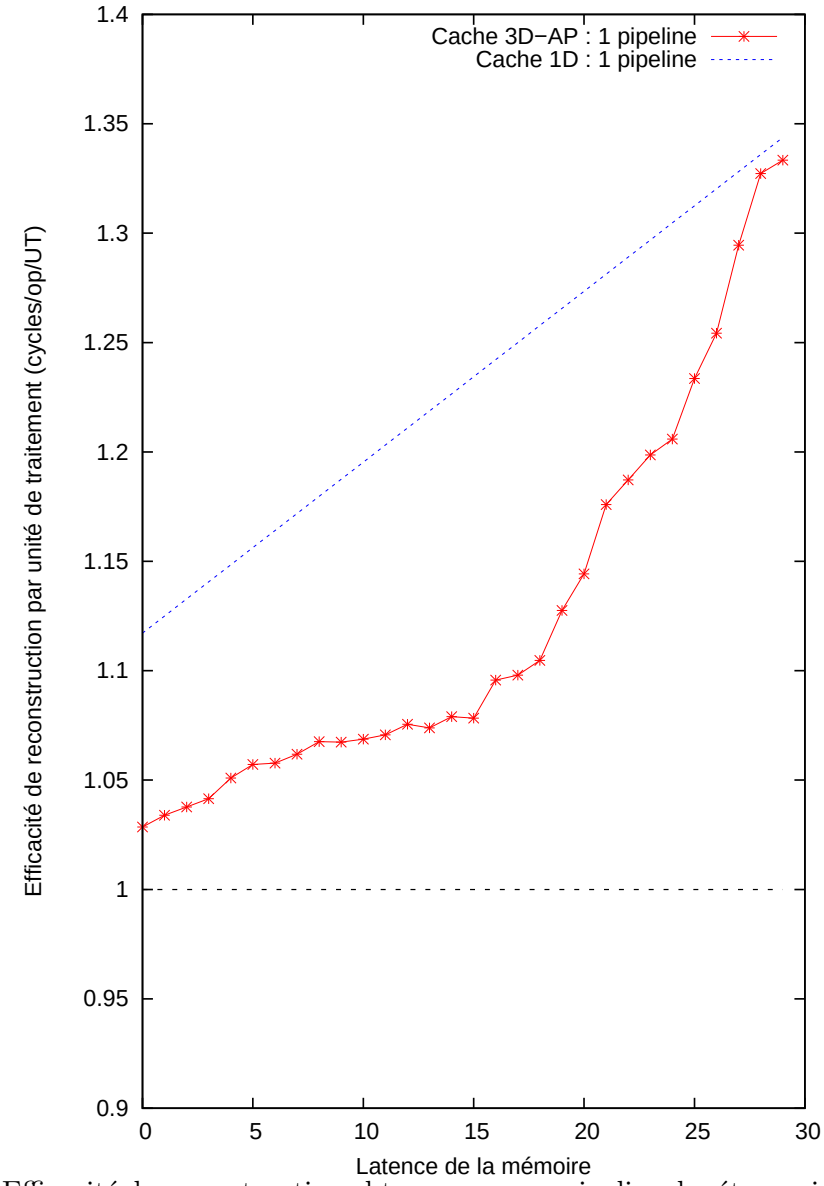


FIG. 5.4 – Efficacité de reconstruction obtenue pour un pipeline de rétroprojection couplé avec le cache 3D-AP et un cache 1D d'un CPU

4 Conclusion

Dans ce chapitre, nous avons comparé les performances obtenues avec des implémentations optimisées de la rétroprojection 3D sur processeurs classiques (Processeurs Intel) et graphiques (Geforce série 8 de Nvidia) avec celles obtenues avec notre architecture 3P.

Les processeurs graphiques grâce à leur fort potentiel de parallélisation constituent indéniablement la solution technologique actuelle la plus rapide. Ils permettent un facteur d'accélération d'un ordre de magnitude en comparaison des processeurs classiques et de notre architecture matérielle. La technologie FPGA quant à elle n'offre pas (ou n'offre plus depuis 2004) les ressources de calcul nécessaires pour concurrencer efficacement les processeurs graphiques dans le but d'accélérer les algorithmes massivement parallèles comme la reconstruction tomographique. Toutefois, nous avons démontré dans cette étude que notre architecture 3P offre le plus grand potentiel d'accélération grâce à une efficacité de reconstruction proche de l'optimal (1 cycle/op/UT). En effet lors de la parallélisation des unités de calcul, l'architecture 3P reste la plus robuste face au mur mémoire en exploitant au mieux le débit de la mémoire externe. Elle reste ainsi l'architecture la plus performante même si son efficacité de reconstruction s'érode avec le nombre d'unités de calcul ($\boldsymbol{\eta_{/\!/} = 0.56}$ pour 8 unités). Cette très bonne efficacité de reconstruction est synonyme de faible consommation. En imagerie médicale, ce facteur n'entre pas en ligne de compte. Cependant, les principes qui ont guidé la conception de notre architecture étant génériques (Pipeline, Préchargement avec le cache 3D-AP et Parallélisation), ils peuvent s'appliquer de manière pertinentes dans des domaines d'application où la faible consommation est activement recherchée.

Les trois architectures étudiées (CPU, GPU et architecture 3P) exploitent efficacement la localité spatiale et temporelle introduite avec la reconstruction par blocs grâce à l'utilisation de cache mémoire. Trois caches ont ainsi été étudiés : le cache 1D des CPUs, le cache 2D de texture des GPUs et le cache 3D-AP de notre architecture 3P. Ces caches permettent d'obtenir de bonnes performances sans la nécessité de mettre en place une stratégie mémoire "ad hoc". L'emploi de ces caches mémoire est en effet générique et ne nécessite pas d'effort supplémentaire pour le concepteur/développeur.

Les caches des CPUs ne permettent qu'un préchargement 1D des données de projection (les bins d'une ligne $\boldsymbol{l_\rho}$ de projection). Malgré la faible portée de leur mécanisme de préchargement, ces caches mémoire permettent un facteur important d'accélération (facteur 3 mesuré pour un Pentium 4 lors de la rétroprojection 3D). En effet, le coût d'accès à la mémoire externe pour chaque ligne de projection est amorti par les accès ultérieurs à cette ligne de projection en cache. Cela s'explique par le fort taux de réutilisation des données lors de la rétroprojection 3D d'un bloc (supérieur à 20).

Le cache 2D de texture des GPUs quant à lui permet un préchargement des données 2D, c'est-à-dire des plans mémoire $\boldsymbol{l_\rho \cdot L_\lambda}$. Grâce à ce préchargement (partiel ou total) des lignes d'un plan de projection et à un ordonnancement périodique des threads, le GPU arrive avec succès à masquer l'importante latence de la mémoire de texture (200 à

500 cycles) lors du chargement d'un plan mémoire dans le cache.

Enfin, le cache 3D-AP permet le préchargement des données 3D (chargement d'une page mémoire $\boldsymbol{l_\rho \cdot L_\lambda \cdot H_\phi}$) grâce au suivi des accès dans chacune des 3 dimensions de l'espace de projection. Ainsi les plans mémoire ($\boldsymbol{l_\rho \cdot L_\lambda}$) prochainement accédés sont préchargés dans le cache avant utilisation. Pour une architecture matérielle avec un pipeline de rétroprojection, le cache 3D-AP permet ainsi un gain en performance de 10 à 20 % par rapport à un cache 1D sans préchargement 2D/3D des données. L'emploi du cache 3D-AP pour augmenter le débit mémoire à l'entrée des pipelines de projection s'avère ainsi justifié.

Expérimentalement, nous avons donc démontré que le cache nD-AP offre une alternative efficace aux caches associatifs. Lorsqu'une forte localité spatiale et temporelle des index mémoire (c'est-à-dire des coordonnées dans l'espace nD) existe, le cache nD-AP permet de bénéficier à la fois de la faible latence offerte par les caches à accès direct (1 cycle), et de la limitation des conflits d'occupation de ligne de cache, offerte par les caches entièrement associatifs (dans ce cas, la limite correspond à la taille du cache). Les caches associatifs, de leur coté, ne parviennent qu'à un compromis entre l'accès direct et l'associativité totale avec une latence un peu plus importante (3 cycles pour le cache L1) et un risque accru de conflit.

Dans le chapitre suivant, nous présentons un système complet de reconstruction en tomographie TEP basé sur deux modules matérielles (FPGA ou ASIC) : un de projection et un de rétroprojection. Nous estimerons les performances que l'on peut espérer d'un tel système pour un algorithme analytique (3D-RP) et un autre itératif (3D-EM).

Chapitre 6

Conception d'un système complet de reconstruction tomographique

Dans ce chapitre, nous présentons un système complet de reconstruction en tomographique TEP, finalité du projet ArchiTEP. Ce système de reconstruction intègre le Rétroprojecteur $\boldsymbol{R_{VIB}}$ (Voxel driven avec Interpolation Bilinéaire) décrit dans les chapitres précédents ainsi que le Projecteur $\boldsymbol{P_{Lr}}$ par Lancer de rayon (*ray casting*) développé au sein du laboratoire Gipsa-lab par [Mancini 00].

Nous présenterons tout d'abord comment à partir d'une paire de Projection / Rétroprojection ($\boldsymbol{P/R}$), les reconstructions analytiques et itératives peuvent être effectuées. Nous décrirons ensuite l'architecture du système de reconstruction intégrant une paire $\boldsymbol{P/R}$ matérielle. Enfin, nous évaluerons la qualité de reconstruction et l'efficacité de reconstruction d'un tel système.

1 Reconstruction à l'aide d'une paire P/R

Comme nous l'avons vu au chapitre 1, la projection et la rétroprojection sont deux fonctions géométriques duales et une étape de filtrage doit être adjointe à la rétroprojection afin d'obtenir l'opérateur inverse de la projection. Nous mettrons tout d'abord en évidence la prédominance des étapes de projection et de rétroprojection lors de l'exécution des méthodes analytiques (3D-RP) et itératives (3D-EM) de reconstruction. Nous présenterons alors une paire $\boldsymbol{P/R}$ matérielle formée par le rétroprojecteur $\boldsymbol{R_{VIB}}$ implémenté via notre architecture 3P, et par le projecteur par lancer de rayon $\boldsymbol{P_{Lr}}$ de [Mancini 00]. Nous décrirons ce projecteur dont l'architecture a été conçue en respectant une méthodologie d'Adéquation Algorithme Architecture proche de celle présentée pour notre rétroprojecteur (voir chapitre 3). Enfin, nous décrirons la manière dont la paire $\boldsymbol{P_{Lr}/R_{VIB}}$ permet de calculer en ligne la matrice système $\boldsymbol{H}$ des algorithmes itératifs.

1.1 Etapes de projection et rétroprojection des méthodes de reconstruction

Les méthodes analytiques et itératives de reconstruction possèdent, toutes les deux, des étapes de projection et de rétroprojection. Pour l'algorithme analytique 3D-RP, ces deux étapes correspondent de manière explicite aux opérateurs géométriques de projection et de rétroprojection. Par contre pour l'algorithme itératif EM, ces étapes correspondent à des multiplications de matrices avec des vecteurs.

1.1.1 Algorithme 3D-RP

L'algorithme 3D de ReProjection (3D-RP) de [Kinahan 89] (voir chapitre 1) se décompose en 3 étapes de calcul :

($\mathbf{Rf_{2D}}$) Rétroprojection 2D précédée d'un filtrage rampe. Elle permet d'obtenir la première estimée du volume.

($\mathbf{P_{3D}}$) Projection 3D du volume estimé afin de compléter les données de projection.

($\mathbf{Rf_{3D}}$) Rétroprojection 3D précédée d'un filtrage de Colsher. La rétroprojection des données de projection complètes permet d'obtenir le volume final.

Comme nous l'avons remarqué au chapitre 2, les opérations de projection et de rétroprojection sont beaucoup plus gourmandes en ressources de calcul que les étapes de filtrage. C'est pourquoi l'accélération de cet algorithme nécessite avant tout une paire $\boldsymbol{P/R}$ rapide.

1.1.2 Algorithme EM

L'algorithme EM appliqué à la tomographie TEP [Shepp 82, Lange 84] est une technique itérative permettant de maximiser la vraisemblance de l'estimée de la densité du volume $\hat{\boldsymbol{f}}$ par rapport aux données $\boldsymbol{y}$ acquises par le scanner (voir chapitre 1). L'acquisition est modélisée par la matrice système $\boldsymbol{H} = (\boldsymbol{h_{ij}})$. A chaque itération, une nouvelle estimée $\hat{\boldsymbol{f}}^{(\boldsymbol{n+1})}$ est calculée à partir de l'estimée courante du volume $\hat{\boldsymbol{f}}^{(\boldsymbol{n})}$ (voir équation 6.1).

$$\hat{\boldsymbol{f}}_{\boldsymbol{j}}^{(\boldsymbol{n+1})} = \frac{\hat{\boldsymbol{f}}_{\boldsymbol{j}}^{(\boldsymbol{n})}}{\sum_{\boldsymbol{i}} \boldsymbol{h_{ij}}} \sum_{\boldsymbol{i}} \boldsymbol{h_{ij}} \frac{\boldsymbol{y_i}}{\sum_{\boldsymbol{k}} \boldsymbol{h_{ik}} \hat{\boldsymbol{f}}_{\boldsymbol{k}}^{(\boldsymbol{n})}} \tag{6.1}$$

Ce calcul se décompose en 5 étapes de calcul, partagées par l'ensemble des méthodes itératives de reconstruction :

($\mathbf{P_{3D}}$) Projection 3D du volume $\hat{\boldsymbol{f}}^{(\boldsymbol{n})}$ afin d'obtenir $\hat{\boldsymbol{y}}_{\boldsymbol{i}}^{(\boldsymbol{n})}$, la n$^{\boldsymbol{ime}}$ estimée des données : $\hat{\boldsymbol{y}}_{\boldsymbol{i}}^{(\boldsymbol{n})} = \sum_{\boldsymbol{k}} \boldsymbol{h_{ik}} \hat{\boldsymbol{f}}_{\boldsymbol{k}}^{(\boldsymbol{n})}$.

(CS) Comparaison des sinogrammes entre les données mesurées $\boldsymbol{y_i}$ et les données estimées $\hat{\boldsymbol{y}}_{\boldsymbol{i}}^{(\boldsymbol{n})}$. L'erreur de mesure $\boldsymbol{\epsilon}_{\boldsymbol{i}}^{(\boldsymbol{n})}$ est calculée par division pour l'algorithme EM : $\boldsymbol{\epsilon}_{\boldsymbol{i}}^{(\boldsymbol{n})} = \dfrac{\boldsymbol{y_i}}{\hat{\boldsymbol{y}}_{\boldsymbol{i}}^{(\boldsymbol{n})}}$.

(R_{3D}) Rétroprojection 3D $R_j^{(n)}$ de l'erreur de projection $\epsilon_i^{(n)}$: $R_j^{(n)} = \sum_i h_{ij}\epsilon_i^{(n)}$.

(N) Normalisation de la rétroprojection avec la multiplication du volume rétroprojeté $R^{(n)}$ par le volume de sensibilité s_j : $C_j^{(n)} = \frac{1}{\sum_i h_{ij}} \cdot R_j^{(n)} = \frac{1}{s_j} \cdot R_j^{(n)}$.

(MV) Mise à jour du volume $\hat{f}^{(n)}$ par le correctif $C^{(n)}$. Cette correction se fait par multiplication pour l'algorithme EM : $\hat{f}_j^{(n+1)} = \hat{f}_j^{(n)} \cdot C_j^{(n)}$.

Comme pour l'algorithme 3D-RP, les étapes de projection et de rétroprojection de l'algorithme EM constituent les principales étapes de calcul. Pour l'algorithme EM du logiciel de reconstruction STIR, elles représentent plus de **95%** du temps de calcul.

1.2 Description du projecteur par lancer de rayon

Le projecteur utilisé calcule chaque *bin* du sinogramme en lançant un rayon (correspondant à la LOR du bin) à travers le volume. L'utilisation de la géométrie projective appliquée au lancer de rayon permet de définir une architecture matérielle de ce projecteur, à la fois rapide et précise. Par ailleurs, cette architecture réduit le goulot d'étranglement que constitue l'accès à la mémoire externe, en utilisant comme pour le rétroprojecteur un cache 3D-AP.

1.2.1 La projection par lancer de rayon

La projection est définie analytiquement par la transformée de radon de f. Elle consiste à effectuer une intégrale linéaire du volume le long des LORs de l'espace de projection traversant le volume (voir équation 6.2). Nous utilisons les coordonnées de projection $(u_\parallel, v_\parallel, \psi, \theta)$ pour désigner les LORs (voir figure 1.8 du chapitre 1).

$$p_{\theta,\psi}(u_\parallel, v_\parallel) = \int_{-\infty}^{+\infty} f(x, y, z) \cdot d\vec{w}_\parallel \tag{6.2}$$

Une approche classique pour calculer cette intégrale linéaire est de "lancer le rayon" i puis de sommer l'intensité des voxels j traversés par ce rayon. Lors de cette somme, l'intensité des voxels est pondérée par la longueur l_{ij} de traversée du rayon dans les voxels ([Mancini 06]). Pour approcher l_{ij}, certains algorithmes proposent d'échantillonner régulièrement le rayon et de sommer les "morceaux" se trouvant dans le rayon d'action du voxel [Chidlow 03]. L'algorithme de lancer de rayon que nous utilisons calcule les points d'intersection du rayon (entrée et sortie) avec les "parois virtuelles" de chaque voxel traversé (voir figure 6.1). La longueur du segment peut ensuite être calculée simplement avec le théorème de Pythagore.

1.2.2 Obtention d'un sinogramme complet

Pour obtenir un sinogramme complet $\hat{y}^{(n)}$, il faut lancer toutes les LORs du scanner. Ainsi pour chaque LOR définie par le quadruplé $(\rho, \phi, \lambda, \delta)$, on calcule tout d'abord le

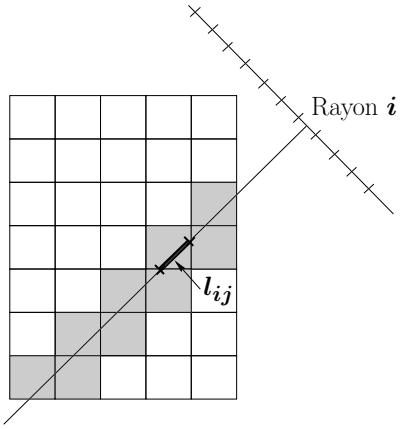


FIG. 6.1 – Lancer de rayon avec pondération de la distance traversée dans chaque voxel

vecteur directeur $\vec{V}$ et un point A appartenant à la LOR (voir équation 1.37 du chapitre 1). Une fois la LOR définie, on effectue le lancer du rayon correspondant à cette LOR. L'algorithme de projection se résume ainsi à la boucle sur les LORs de l'algorithme 8).

Algorithm 8 Boucle générique de projection géométrique

```
for delta = 0 to delta_max − 1 do
  for lambda = 0 to lambda_max − 1 do
    for phi = 0 to phi_max-1 do
      for rho = 0 to rho_max-1 do
        Calcul des coordonnées de A et V de la LOR
        Calcul de l'intégrale linéaire du volume le long de la LOR par lancer de rayon
      end for
    end for
  end for
end for
```

1.2.3 Adéquation Algorithme Architecture

L'architecture parallèle de projection 3D par lancer de rayon atteint un haut débit de calcul grâce aux lancers synchronisés de LORs adjacentes. Cette introduction de localité temporelle et spatiale lors des accès au volume permet une utilisation efficace d'un cache 3D-AP hiérarchique ([Mancini 06]).

La géométrie projective est utilisée afin de permettre un contrôle simple et efficace de la dynamique des données traitées et d'exprimer ainsi tous les calculs en virgule fixe. La géométrie projective est basée sur l'ajout d'une dimension à l'espace affine euclidien utilisé définissant l'espace des coordonnées homogènes. Ce changement d'espace nous permet de réduire la dynamique des coordonnées de tout point de l'espace affine à l'intervalle [-1,1[dans ce nouvel espace. En restant le plus possible dans cet espace, on

simplifie les opérateurs matériels (comme la division) et les calculs d'intersections grâce à la représentation en virgule fixe. Le temps de calcul est ainsi réduit.

1.3 Calcul en ligne de la matrice système par la paire de projection/rétroprojection

La matrice système étant de taille trop importante pour être stockée en mémoire externe (voir section 1.3.1 du chapitre 2), nous calculons en ligne ces coefficients à l'aide des opérateurs géométriques de projection et de rétroprojection.

Dans cette section, nous présenterons tout d'abord un modèle simplifié de l'acquisition (modèle *shift invariant*) qui permet d'exprimer sous la forme d'une intégrale volumique les coefficients h_{ij} de la matrice système. Puis, nous verrons que le projecteur P_{Lr} et le rétroprojecteur R_{VIB} constituent alors deux manières distinctes d'approcher ces intégrales volumiques. La paire P_{Lr}/R_{VIB} est ainsi qualifiée d'incohérente. Enfin, nous apporterons quelques réflexions sur l'impact de cette incohérence sur la convergence des algorithmes itératifs.

1.3.1 Expression de h_{ij} sous la forme d'une intégrale volumique

La matrice système **H** représente les contributions h_{ij} du voxel i pour le capteur j. Plus exactement, h_{ij} correspond à la probabilité qu'une annihilation apparaisse dans le voxel j et soit comptabilisée par le capteur i. Cette probabilité est obtenue en intégrant la sensibilité $s_i(\vec{x})$ du capteur i sur la fonction de densité de base $v_j(\vec{x})$ modélisant un voxel :

$$h_{ij} = \int s_i(\vec{x}) \cdot v_j(\vec{x}) \cdot d\vec{x} \tag{6.3}$$

La fonction de densité de base $v_j(\vec{x})$ correspond à la fonction $\text{Voxel}_j(\vec{x})$ modélisant le voxel j par des fonctions portes ($\prod_\delta$) et normalisée par le volume du voxel V_{voxel} :

$$v_j(\vec{x}) = \frac{\text{Voxel}_j(\vec{x})}{V_{voxel}} = \frac{\prod_{\delta_{xy}}(x - x_j) \cdot \prod_{\delta_{xy}}(y - y_j) \cdot \prod_{\delta_{z\lambda}}(z - z_j)}{V_{voxel}} \tag{6.4}$$

La sensibilité $s_i(\vec{x})$ des capteurs est exprimée à l'aide d'une fonction h appelé *kernel*. Dans la plupart des méthodes de reconstruction itératives, ce *kernel* est fonction des distances axiale ($|\lambda - \lambda_i|$) et transverse ($|\rho - \rho_i|$) du rayon $L\vec{O}R_i$ avec le voxel $\vec{x}$. Cette approximation correspond à un modèle idéal de réception invariant par déplacement par rapport à la LOR i ($L\vec{O}R_i$) passant par le centre du capteur i (*shift invariant Point spread function PSF*). Le *kernel* H peut être pris égal à une cloche gausienne ou bien à une fonction rectangulaire. Notre étude se limitera à une fonction rectangulaire correspondant au Tube de réponse $\text{TOR}_i(\vec{x})$. Ce parallélépipède a pour base le rectangle défini par la surface du capteur d'aire $A_{bin} = \delta_\rho \cdot \delta_{\lambda z}$ et a pour direction $L\vec{O}R_i$. La sensibilité du capteur i associée à ce modèle "shift invariant" en TOR a alors pour expression :

$$s_i(\vec{x}) = h(|\rho - \rho_i|, |\lambda - \lambda_i|) = \prod_{\delta_\rho}(\rho - \rho_i) \cdot \prod_{\delta_{z\lambda}}(\lambda - \lambda_i) \tag{6.5}$$

Avec une modélisation du volume en voxel et une modélisation *shift invariant* de l'acquisition, h_{ij} correspond alors au volume d'interaction V_{ij} entre le volume de réponse du capteur i (modélisé par $\mathrm{TOR}_i(\vec{x})$) et le volume du voxel j (modélisé par $\mathrm{Voxel}_j(\vec{x})$) :

$$h_{ij} = \frac{V_{ij}}{V_{voxel}} = \frac{\int \mathrm{TOR}_i(\vec{x}) \cdot \mathrm{Voxel}_j(\vec{x}) \cdot d\vec{x}}{V_{voxel}} \tag{6.6}$$

1.3.2 La paire P_{Lr}/R_{VIB} comparée au modèle d'acquisition *shift invariant* en TOR rectangulaire

Le calcul en ligne du volume d'interaction est approché de manière différente par le projecteur P_{Lr} et le rétroprojecteur R_{VIB}. Nous avons représenté sur la figure 6.2 dans le cas 2D, les surfaces d'interaction entre un pixel et les TORs des capteurs correspondant (i) au modèle *shift invariant*, (ii) au projecteur par lancer de rayon et (iii) au rétroprojecteur VIB.

Le volume d'interaction V_{ij} calculé par le projecteur P_{Lr} correspond à un parallélépipède ayant pour base la surface du capteur, d'aire $A_{capteur}$ et pour hauteur la longueur de traversée l_{ij} de la LOR_i dans le voxel j :

$$V_{ij} = l_{ij} \cdot A_{capteur} \tag{6.7}$$

Le volume d'interaction calculé par le rétroprojecteur R_{VIB} correspond au volume du voxel V_{voxel} pondéré par un coefficient d'interpolation bilinéaire $C_{interp}(\epsilon_\rho, \epsilon_\lambda)$:

$$V_{ij} = C_{interp}(\epsilon_\rho, \epsilon_\lambda) \cdot V_{voxel} \tag{6.8}$$

Comme nous pouvons le constater sur la figure 6.2, les volumes d'interaction calculés par le projecteur et le rétroprojecteur correspondent à une simplification du modèle *shift invariant*. Par exemple pour le projecteur, les événements d'un voxel selon une direction d'émission (pour un couple angle psi et theta) ne touchent qu'un seul capteur alors qu'avec le modèle *shift invariant*, ils peuvent toucher plusieurs capteurs. Par ailleurs, le projecteur et le rétroprojecteur calculent différemment le volume d'interaction et donc les coefficients h_{ij} de la matrice système.

L'incohérence entre projecteur et rétroprojecteur est susceptible d'avoir à priori un impact sur la convergence des algorithmes itératifs comme l'algorithme EM. Pour palier à cette incohérence, la paire P/R *distance-driven* de [DeMan 04] peut être utilisée. Toutefois, en pratique l'incohérence de la paire P_{Lr}/R_{VIB} ne pose pas de problème de convergence pour les algorithmes itératifs. En effet, la paire P_{Lr}/R_{VIB} a été validée de manière empirique depuis de nombreuses années. De plus, [Zeng 00] a étudié l'influence d'une paire P/R "non cohérente" sur la convergence de l'algorithme EM. Son étude sur les conditions de convergence des paires P/R a porté sur la non-négativité des valeurs

propres des matrices de projection et de rétroprojection. Il est arrivé à la conclusion que peu importe si la paire est cohérente ou incohérente, ce qui compte c'est (i) la convergence initiale lors des premières itérations, (ii) de choisir une paire de $\boldsymbol{P/R}$ rapide, et (iii) d'utiliser des méthodes de régularisation qui guident ou stoppent le processus itératif. Ces règles pratiques se justifient par la nature "mal posé" (*ill condition nature*) du problème inverse. Cela explique pourquoi les solutions parfaitement convergentes sont trop souvent bruitées et donc non désirables. Notre système de reconstruction utilisera donc telle quelle la paire $\boldsymbol{P_{Lr}/R_{VIB}}$ pour effectuer les étapes de projection et de rétroprojection de l'algorithme itératif EM.

2 Architecture du système de reconstruction

La technologie SoPC permet un partionnement logiciel/matériel très utile lors de l'implémentation de chaque algorithme. Les étapes importantes, gourmandes en ressources de calcul sont ainsi implémentées en matériel sur la matrice FPGA et les étapes de transition entre les projections et rétroprojections, de complexité algorithmique moindre sont implémentées en logiciel sur le processeur du système sur puce ou sur le processeur plus performant du PC hôte. De cette manière, notre paire $\boldsymbol{P_{Lr}/R_{VIB}}$ matérielle peut être intégrée dans un système complet de reconstruction utilisant la technologie SoPC. Nous décrivons dans la suite, le partionnement logiciel/matériel correspondant aux architectures des algorithmes analytiques et itératifs.

2.1 Reconstruction 3D-RP

Pour la reconstruction analytique, seule la paire $\boldsymbol{P_{Lr}/R_{VIB}}$ est implémentée sur la matrice FPGA du système programmable sur puce. L'étape de filtrage de complexité algorithmique moindre par rapport à la rétroprojection et à la projection, est effectuée sur le PC hôte. Un système simple de synchronisation entre le PC hôte et le processeur du système sur puce permet la transition entre les étapes de filtrage (filtrage rampe de l'étape $\boldsymbol{Rf_{2D}}$ et filtrage de colsher de l'étape $\boldsymbol{Rf_{3D}}$) et les étapes de projection/rétroprojection.

2.2 Reconstruction 3D-EM

Pour la reconstruction itérative, des IPs implémentant des fonctions basiques (comparaison, multiplication, division) sont ajoutés à ceux de projection et rétroprojection. L'architecture du système comporte deux flots de données : un en voxels, et un autre en LORs (données de projection). Ces deux flots correspondent chacun à une chaîne de modules matériels implémentant une séquence d'étapes de l'algorithme EM. Les deux chaînes de traitement sont représentées sur la figure 6.3.

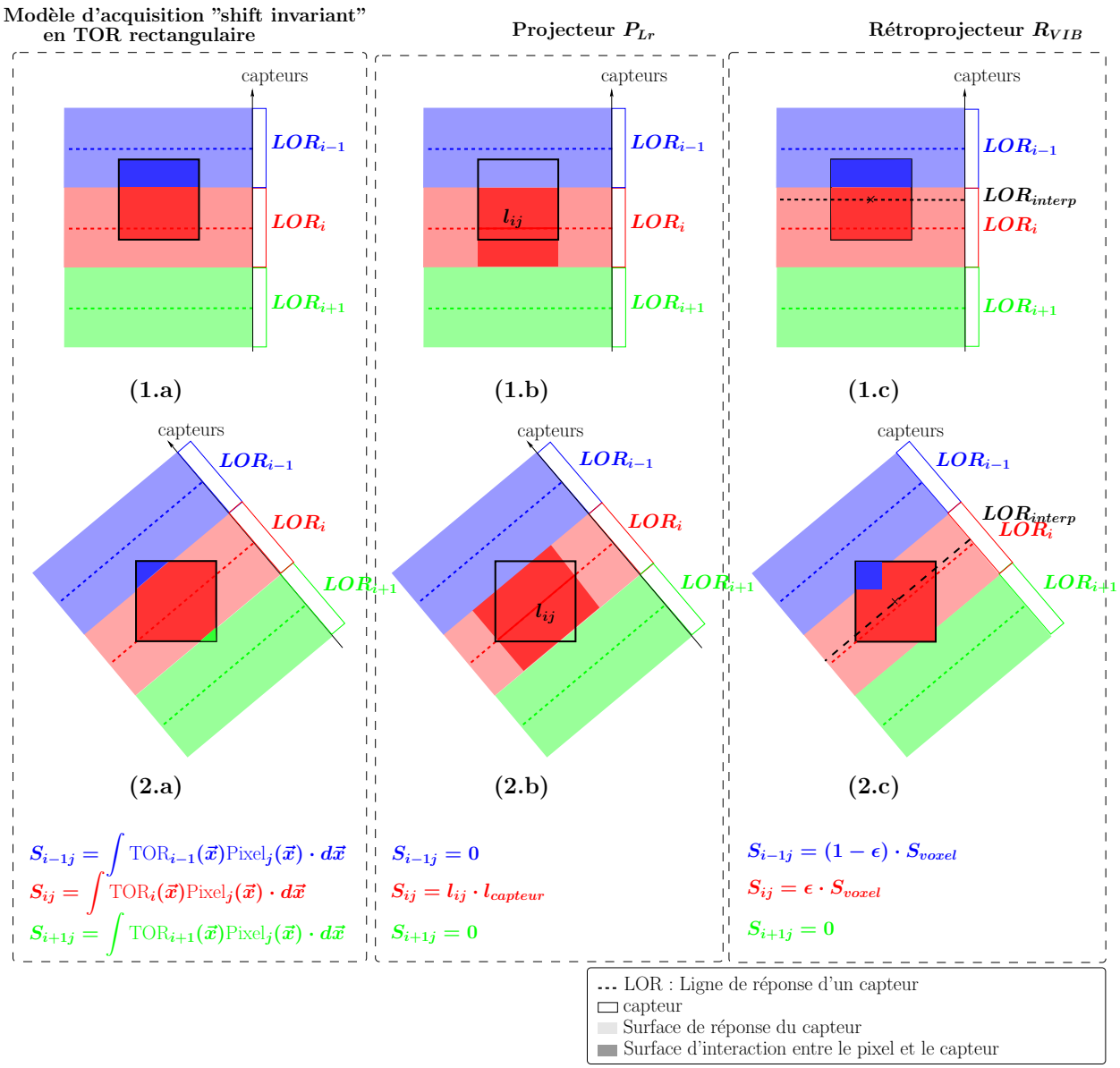


FIG. 6.2 – Surface d'interaction entre le pixel $\boldsymbol{j}$ et les capteurs $\boldsymbol{i-1}$, $\boldsymbol{i}$ et $\boldsymbol{i+1}$ pour l'angle $\boldsymbol{\phi = 0}$ en (1) et l'angle $\boldsymbol{\phi \# 0}$ en (2). Sont représentés en (a) les surfaces d'interaction du modèle d'acquisition "shift invariant" en TOR rectangulaire, en (b) celles calculées par le projecteur et en (c) celles calculées par le rétroprojecteur

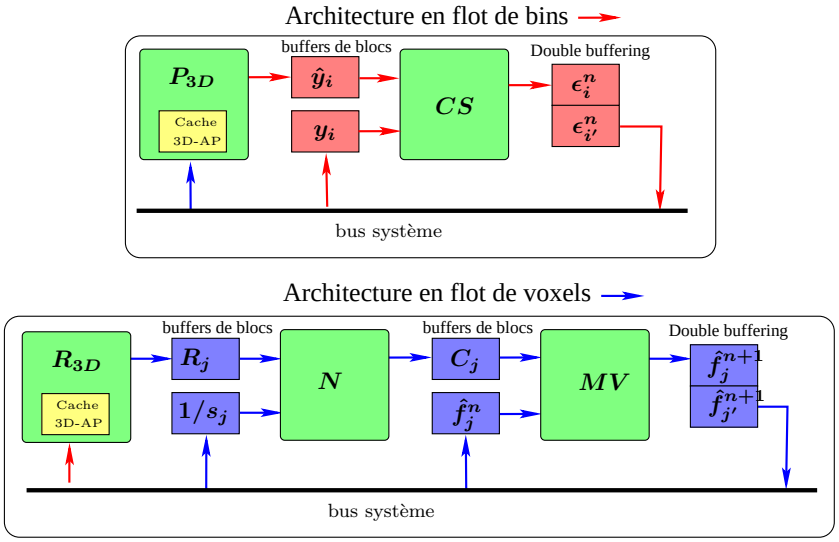


FIG. 6.3 – Architectures en flot de voxels et en flot de bins pour la reconstruction 3D-EM.

Les données sont traitées par bloc (blocs de LORs pour le flot en LORs et blocs de voxels pour le flot en voxels). Un buffer entre chaque module matériel permet de stocker les blocs de données transitant dans la chaîne de traitement. Le dernier module de la chaîne utilise la technique de *double buffering* afin d'effectuer en parallèle le traitement d'un bloc de données et l'écriture en mémoire externe du dernier bloc traité.

2.2.1 Architecture avec un flot de LORs

Cette architecture est constituée d'un projecteur 3D $\mathbf{P_{3D}}$ et d'un module de comparaison de sinogramme **CS**. Ce dernier correspond à une division et peut être implémenté à l'aide d'un pipeline offrant un débit de calcul d'une division par cycle. Ainsi, l'étape **CS** est principalement limitée lors de son exécution par les accès à la mémoire externe. Cette étape requiert en effet la lecture d'un bloc de donnée de mesures $\boldsymbol{y_i}$ et l'écriture d'un bloc de comparaison de sinogrammes $\boldsymbol{\epsilon_i^{(n)}}$. Toutefois dans son ensemble, l'architecture est avant tout limitée par l'étape $\mathbf{P_{3D}}$.

2.2.2 Architecture avec un flot de voxels

Cette architecture est constituée d'un rétroprojecteur 3D $\mathbf{R_{3D}}$, d'un module de normalisation de la rétroprojection **N** et d'une mise à jour de volume **MV**. Les multiplieurs matériels des puces Virtex peuvent être efficacement utilisés pour atteindre un débit de calcul d'une multiplication par cycle. Comme pour l'étape **CS**, les étapes **N** et **MV** sont limitées principalement par les accès à la mémoire externe. En effet, l'étape **N** requiert la lecture d'un bloc de voxels du volume de sensibilité $\boldsymbol{s_j}$, et l'étape **MV** requiert la lecture d'un bloc de voxels de l'estimée courante du volume $\boldsymbol{\hat{f}^{(n)}}$ et l'écriture d'un bloc

de voxels du volume reconstruit $\hat{\boldsymbol{f}}^{(n+1)}$. Mais dans son ensemble, l'architecture est avant tout limitée par l'étape $\mathbf{R_{3D}}$.

3 Qualité de reconstruction

Dans cette section, nous validerons de manière logicielle notre paire $\boldsymbol{P/R}$, afin de nous assurer que l'utilisation d'une paire $\boldsymbol{P_{Lr}/R_{VIB}}$ permet la convergence de l'algorithme EM et que la qualité de reconstruction reste proche des standards cliniques.

3.1 Validation logicielle de la paire P_{Lr}/R_{VIB}

Nos projecteurs et rétroprojecteurs, opérateurs discrets avec une arithmétique en virgule fixe, diffèrent des opérateurs géométriques continus définis au chapitre 1. De plus comme nous l'avons vu précédemment, la paire $\boldsymbol{P_{Lr}/R_{VIB}}$ est incohérente ce qui peut avoir des conséquences sur la convergence de l'algorithme EM. C'est pourquoi afin de valider la pertinence de notre paire $\boldsymbol{P/R}$, nous avons développé des versions logicielles des algorithmes 3D-RP et 3D-EM utilisant une paire $\boldsymbol{P_{Lr}/R_{VIB}}$. Si le rétroprojecteur de cette version logicielle utilise une arithmétique à virgule fixe (réplique exacte de celle utilisée par le rétroprojecteur matériel sur FPGA), ce n'est pas le cas pour le projecteur qui utilise une arithmétique à virgule flottante. Ainsi la paire $\boldsymbol{P_{Lr}/R_{VIB}}$ logicielle n'est pas à 100 % "*bit true*" et la qualité de reconstruction obtenue avec cette paire n'est pas exactement celle que l'on obtiendrait avec notre système de reconstruction sur cartes SoPC. Toutefois, nous avons observé précédemment au chapitre 4 que le rétroprojecteur $\boldsymbol{R_{VIB}}$ est relativement robuste au bruit de calcul avec un $\boldsymbol{EMA_r}$ de **0.13%** entre les arithmétiques à virgule fixe et à virgule flottante. Or puisque la projection et la rétroprojection ont une dynamique de calcul proche, nous pouvons espérer une robustesse équivalente du projecteur face au bruit de calcul. Les travaux de Florian Deboissieu durant son stage ([Deboissieu 06]), ont confirmé cette hypothèse. L'algorithme EM avec une paire $\boldsymbol{P_{Lr}/R_{VIB}}$ s'est avéré dans l'ensemble peu sensible à l'injection de bruits de quantification dans le volume et/ou dans le sinogramme et de bruits de calcul (notamment pour le calcul des coordonnées (A,$\vec{\boldsymbol{V}}$) de la LOR lors de la projection).

3.2 STIR comme référence de reconstruction

Nous avons utilisé le logiciel STIR (version 2.0 [Thielemans 06]) comme référence de reconstruction, afin de nous assurer de la validité de notre paire $\boldsymbol{P_{Lr}/R_{VIB}}$. STIR utilise également une paire $\boldsymbol{P/R}$ pour son implémentation des algorithmes 3D-RP et 3D-EM. Sa paire $\boldsymbol{P/R}$ est légèrement différente de la notre. Elle est faite d'un projecteur à lancer de rayon, variante de l'algorithme de [Siddon 85] et du rétroprojecteur ray-driven de [Egger 98], dérivé de l'algorithme incrémentiel de [Cho 90]. Par ailleurs, contrairement à notre méthode de reconstruction EM, STIR utilise une étape de régularisation lors de sa reconstruction itérative. Son algorithme itératif OSMAPOSL est bien un algorithme EM (avec Ordered Subset pour accélérer la reconstruction) mais après chaque itération

un filtrage passe bas (filtres de Metz de [Jacobson 00]) est effectué pour réduire le bruit du à la nature "mal posé" du problème inverse.

3.3 Convergence de l'algorithme EM avec la paire P_{Lr}/R_{VIB}

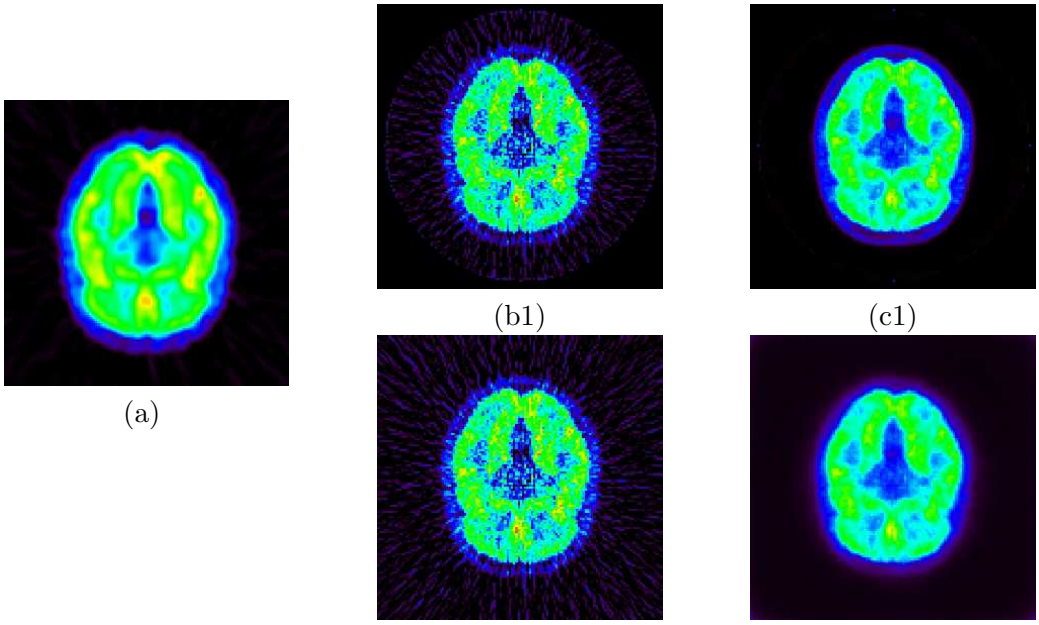
Afin de nous assurer de la convergence de l'algorithme EM, nous avons calculé après chaque itération l'Ecart Absolu Moyen relatif (EAM_r) entre le phantom Shepp Logan (voir chapitre 4) reconstruit avec la paire P_{Lr}/R_{VIB} et le phantom original. Nous avons utilisé comme volume initial f_0 soit un volume blanc (tous les voxels ont pour valeur 1), soit un volume reconstruit par rétroprojection filtrée 2D (Rf_{2D}). Comme illustré sur la figure 6.5 en (a), l'algorithme EM converge jusqu'à 25/30 itérations sans Rf_{2D} préalable et jusqu'à 3 itérations avec Rf_{2D}. Pour l'algorithme OSEM avec 8 *subsets*, la convergence est obtenue dès la troisième itérations sans Rf_{2D} (soit 24 sous itérations) et dès la première itération avec Rf_{2D}. L'algorithme OSMAPOSL de STIR converge plus rapidement et grâce à son filtre de régularisation entre chaque itération, il continue à améliorer la qualité de reconstruction du volume comme l'illustre la figure 6.5 en (b). A noter que l'algorithme EM avec la paire P_{Lr}/R_{VIB} diverge de manière importante quelques itérations après avoir offert un volume de bonne qualité. Il faut donc stopper à temps l'algorithme EM avant qu'il diverge et que le volume reconstruit soit ainsi "perdu".

3.4 Reconstruction de qualité quasi "clinique"

La figure 6.4 présente un exemple de volume reconstruit avec notre paire P_{Lr}/R_{VIB} et par STIR. Le volume correspond à un phantom numérique d'une acquisition IRM d'un cerveau. A chaque zone du cerveau a été attribuée une densité de radiotraceurs FDG à l'aide d'un modèle fonctionnel. Puis une pseudo-acquisition HR+ a été effectuée à l'aide du simulateur monte carlo PET SORTEO de [Reilhac 05] (17 études sont ainsi disponibles sur la base de donnée en ligne de [**?**]). Nous avons ensuite effectué à l'aide du logiciel bkproj3D de Siemens, les différentes corrections nécessaires pour s'approcher d'une qualité de reconstruction de type "clinique" : normalisation, atténuation et diffusé. A noter que pour l'algorithme 3D-RP, la correction en arc et les étapes de filtrage (filtre rampe et filtre de Colsher) ont été effectué à l'aide d'exécutables générés à partir du code de STIR. Un organigramme en annexe E présente l'ensemble des outils logiciels utilisés lors des reconstructions analytiques et itératives.

Comme l'illustre la figure 6.4, qualitativement les volumes reconstruits avec notre paire de projection/rétroprojection sont proches de ceux reconstruits avec STIR. Afin d'évaluer de manière quantitative les volumes reconstruits, nous avons calculé l'Ecart Absolu Moyen relatif (EAM_r voir chapitre 4) entre nos volumes et ceux de STIR (voir tableau 3.4). Pour l'algorithme 3D-RP, l'écart est faible entre les deux implémentations avec un EAM_r de seulement $\mathbf{0,63\%}$. Par contre, pour l'algorithme EM l'écart est un plus important avec un EAM_r de $\mathbf{2.42\%}$. Cette différence est due à l'utilisation d'étapes de régularisation par l'algorithme OSMAPOSL de STIR qui pourraient aisément être effectuées sur le PC hôte lors de son implémentation sur cartes SoPC.

En conclusion, nous pouvons considérer la paire $\boldsymbol{P_{Lr}/R_{VIB}}$ comme tout à fait valide pour être utilisée pour les algorithmes 3D-RP et 3D-EM. Une version matérielle de cette paire permet d'obtenir une qualité de reconstruction proche de celle qu'offre des outils de reconstruction "clinique" comme STIR ($\boldsymbol{EMA_r < 1\%}$).

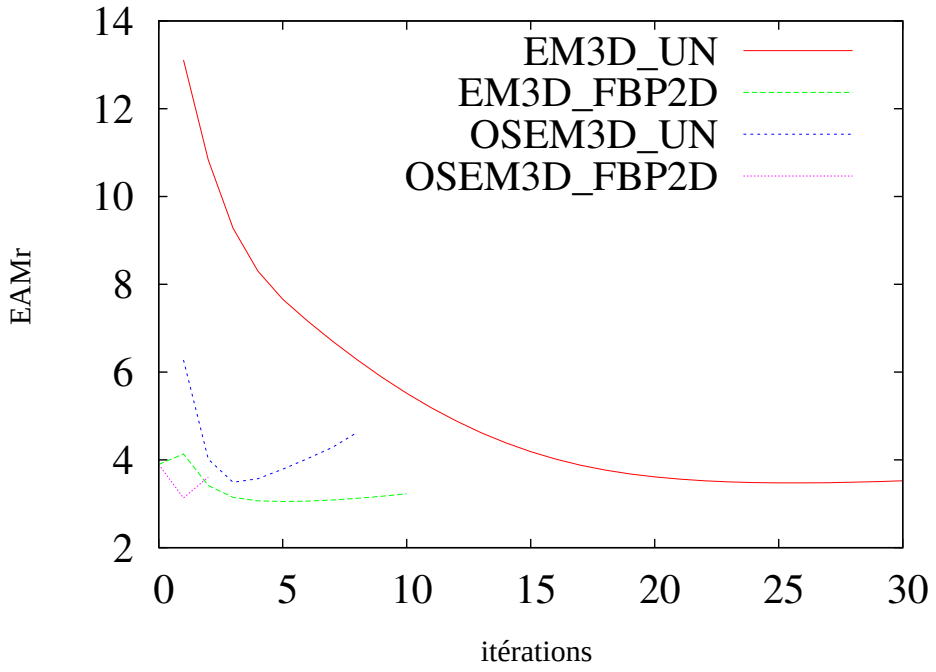


(a) (b1) (c1) (b2) (c2)

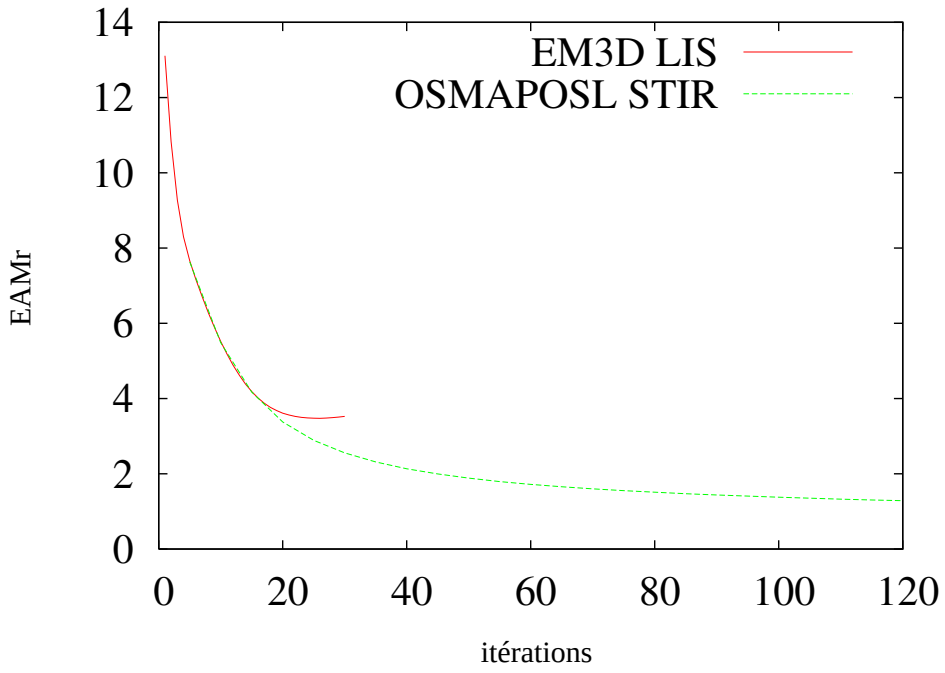
FIG. 6.4 – Sont présentés ci-dessus : le phantom d'émission FDG original (a), les reconstructions 3D-RP par STIR (b1) et notre version logicielle (b2), et les reconstructions itérative avec l'algorithme OSMAPOL de STIR (c1) et par notre algorithme EM après 30 itérations et une initialisation avec un volume "blanc" (c2). Un phantom numérique de cerveau associé avec un modèle FDG fonctionnel a été utilisé par le simulateur monte-carlo PET-SORTEO pour générer les données PET correspondant à une pseudo acquisition HR+ d'un cerveau.

	3D-RP($\boldsymbol{P_{Lr}/R_{VIB}}$) par rapport à 3D-RP(STIR)	3D-EM($\boldsymbol{P_{Lr}/R_{VIB}}$) par rapport à OSMAPOSL(STIR)
$\boldsymbol{EAM_r}$	**0.63%**	**2.42%**

TAB. 6.1 – $\boldsymbol{EMA_r}$ entre les reconstructions avec la paire $\boldsymbol{P_{Lr}/R_{VIB}}$ et avec STIR



(a) Convergence des algorithmes EM et OSEM avec la paire $\boldsymbol{P_{Lr}/R_{VIB}}$. La première estimée $\boldsymbol{f_0}$ est soit un volume blanc avec tous ses voxels à 1 ("_UN") soit un volume reconstruit par rétroprojection filtrée 2D ("_FB2D")



(b) Convergence de l'algorithme EM avec la paire $\boldsymbol{P_{Lr}/R_{VIB}}$ et de l'algorithme OSMAPOSL de STIR

FIG. 6.5 – Convergence de l'algorithme EM : l'Ecart Absolu Moyen relatif $\boldsymbol{EAM_r}$ entre le phantom de Shepp Logan reconstruit et l'original a été calculé à chaque itération

4 Efficacité de reconstruction

Les temps de reconstruction du système de reconstruction sur cartes SoPC présentés dans cette section correspondent aux mesures effectuées sur carte pour le projecteur $\boldsymbol{P_{Lr}}$ [Mancini 06] et le rétroprojecteur $\boldsymbol{R_{VIB}}$ (voir chapitre 4) et aux estimations de performances des IPs additionnels nécessaires pour l'algorithme EM. Après avoir comparé ces performances avec celles obtenues avec le logiciel STIR, nous tirerons les conclusions sur l'efficacité globale de cette architecture utilisant une paire $\boldsymbol{P_{Lr}/R_{VIB}}$ matérielle.

4.1 Performances espérées

Notre estimation se base sur l'utilisation d'un Virtex fonctionnant à 200 Mhz. Nous avons calculé pour chaque étape, le temps de traitement total pour la reconstruction d'un volume $\mathbf{128^2 \cdot 63}$ à partir de données HR+ en multipliant l'efficacité de reconstruction (cycles/op), le nombre d'opération par élément (Nb op/Elt) et le nombre d'éléments (voxels ou LORs selon les cas). Nous avons rajouté un surcoût de **20%** du temps total pour prendre en compte le coût des communications à chaque transition entre étapes de calcul. Les estimations pour les algorithmes 3D-RP et 3D-EM sont présentées respectivement sur les tableaux 6.2 et 6.3.

	Elt	Nb Elt	Nb Op/Elt	Cycles/Op	Temps
($\boldsymbol{R_{2D}}$) Rétroprojection 2D	Voxels	$\mathbf{10^6}$	144	0.25 (8)	$\sim$ 185 ms
($\boldsymbol{P_{3D}}$) Projection 3D	LORs	$\mathbf{6,3 \cdot 10^6}$	$\sim$ 200	0.5 (8)	$\sim$ 3.2 s
($\boldsymbol{R_{3D}}$) Rétroprojection 3D	Voxels	$\mathbf{10^6}$	$\mathbf{144 \cdot 5 = 720}$	0.25 (8)	$\sim$ 930 ms
TOTAL sans filtrage	$\sim$ 4.3 s + 20 % = 5.2 s				

TAB. 6.2 – Evaluation du temps et de l'efficacité de notre système de reconstruction pour l'algorithme 3D-RP sans les étapes de filtrage

	Elt	Nb Elt	Nb Op/Elt	Cycles/Op	Temps/Iter.
($\boldsymbol{P_{3D}}$) Projection 3D	LORs	$\mathbf{9.3 \cdot 10^6}$	$\sim$ 200	0.5 (8)	$\sim$ 4.65 s
(CS) Comparaison des Sinogrammes	LORs	$\mathbf{9.3 \cdot 10^6}$	1	1.25 (1)	$\sim$ 70 ms
($\boldsymbol{R_{3D}}$) Rétroprojection 3D	Voxels	$\mathbf{10^6}$	720	0.25 (8)	$\sim$ 930 ms
(N) Normalisation	Voxels	$\mathbf{10^6}$	1	1.25 (1)	$\sim$ 10 ms
(MV) Mise à jour du Volume	Voxels	$\mathbf{10^6}$	1	1.25 (1)	$\sim$ 10 ms
TOTAL	$\sim$ 5.6s + 20 % = 6.7 s				

TAB. 6.3 – Evaluation du temps et de l'efficacité de notre système de reconstruction pour une itération de l'algorithme 3D-EM.

Pour la rétroprojection, un élément correspond à un voxel reconstruit et une opération à l'accumulation d'un bin. Pour la projection, un élément correspond à une LOR et une opération au calcul de l'intersection d'une LOR avec un voxel. Les données HR+ sont constituées de 5 segments, 144 angles de projection et 288 bins par ligne de projection (269 après correction d'arc pour l'algorithme analytique). Pour l'algorithme 3D-RP, les données de projection sont formées de 239 sinogrammes correspondant à l'acquisition et de 162 sinogrammes générés par projection 3D. Pour l'algorithme 3D-EM, les données de projection sont constituées uniquement des 239 sinogrammes acquis.

Nous avons utilisé pour cette estimation, un rétroprojecteur et un projecteur avec chacun 8 unités de traitement. Un tel système est envisageable à l'aide d'une carte possédant deux Virtex 4 FX60, un pour la projection et un autre pour la rétroprojection. L'efficacité serait ainsi de l'ordre de 0.25 cycles/Op pour le rétroprojecteur (voir chapitre 4 et 5) et de 0.5 cycles/op pour le projecteur ([Mancini 06]). Les IPs **CS**, **N** et **MV** sont limitées par l'accès à la mémoire externe. Pour un bus mémoire avec une fréquence de 200 Mhz et une largeur de 64 bits, le débit mémoire est de 4 données par cycle (les voxels ou les bins sont codés sur 16 bits). Les étapes **CS**, **N** et **MV** requièrent ainsi 0.25 cycle/op pour la lecture en mémoire externe (les écritures éventuelles sont masquées grâce au *double buffering*) et 1 cycle/op pour l'exécution de l'opération (multiplication ou écriture). Ainsi ces IPs ont une efficacité de 1.25 cycles/op.

4.2 Facteurs d'accélération espérés

Nous présentons côte à côte, sur les tableaux 6.4 et 6.5, les temps de reconstruction espérés de notre architecture et ceux mesurés sur un Pentium 4 avec le logiciel STIR. Puisque STIR reconstruit des volumes cylindriques (dans ce cas précis, de taille $\mathbf{64^2 \cdot \pi \cdot 63}$ voxels), nous avons mis à l'échelle ces temps pour qu'ils correspondent à la reconstruction d'un volume $\mathbf{128^2 \cdot 63}$.

Une accélération d'un facteur 7.5 peut être espérée pour l'algorithme 3D-RP et d'un facteur 3.7 pour l'algorithme 3D-EM. Plus intéressant, l'architecture matérielle serait 120 fois plus efficace que le logiciel STIR pour l'algorithme 3D-RP et 60 fois plus pour l'algorithme 3D-EM. Le tableau 6.6 présente les facteurs d'accélération et les gains en efficacité de reconstruction (ratio entre l'efficacité de reconstruction de STIR et celle de notre système) obtenus pour chacun des projecteurs. Il apparaît ainsi clairement que le rétroprojecteur permet une accélération plus importante que le projecteur. Cela s'explique par la plus grande complexité à paralléliser efficacement le projecteur 3D. En effet, le lancer d'un groupe de LORs doit être correctement synchronisé afin de conserver la plus grande localité temporelle et spatiale possible lors des traversées du volume. De plus, contrairement à notre projecteur, STIR exploite les symétries (au moins 8) afin de réduire le nombre de lancers de rayons.

Algorithme	Matériel	Temps	Proj/Rétro	Cycles/Voxels
3D-RP (STIR)	P4 (3.2 GHz)	38.6 s	**49%/50%**	120 000
3D-RP	V4 (200 MHz)	$\sim$ 5.2 s	**74%/25%**	$\sim$ 1 000

TAB. 6.4 – Temps et Efficacité mesurés de la reconstruction analytique (3D-RP) logicielle de STIR sur un Pentium 4 et ceux espérés de notre système de reconstruction (algorithme 3D-RP). Ces temps de reconstruction sont évalués sans les étapes de filtrage.

Algorithme	Matériel	Temps	Proj/Rétro	Cycles/Voxels
OSMAPOSL (STIR)	P4 (3.2 GHz)	24.6 s/Iter.	**54%/45%**	76 300
EM	V4 (200 MHz)	$\sim$ 6.7 s/Iter.	**82%/16%**	$\sim$ 1 300

TAB. 6.5 – Temps et Efficacité mesurés de la reconstruction itérative (OSMAPOSL) logicielle de STIR sur un Pentium 4 et ceux espérés de notre système de reconstruction (algorithme EM).

Nous pouvons observer que notre rétroprojecteur permet un gain en efficacité de reconstruction moins important pour l'algorithme 3D-EM : le facteur d'accélération est de 17.5 pour l'algorithme 3D-RP et de 12 pour l'algorithme 3D-EM. Cela est du au nombre de LORs utilisé pour l'algorithme 3D-EM moins important (239 sinogrammes en 3D-EM contre 239+162 en 3D-RP). Or puisque son rétroprojecteur est *ray driven*, cela réduit du coup le nombre d'opérations pour l'algorithme 3D-EM. Alors que pour notre rétroprojecteur *voxel-driven* le nombre de voxels à traiter est le même en 3D-RP et en 3D-EM, et donc le nombre d'opérations reste constant. Par ailleurs, le projecteur de STIR est plus efficace en 3D-EM qu'en 3D-RP. Cela est du à l'exploitation de la symétrie axiale qui est plus importante pour les données de projection utilisées en 3D-EM. Pour toutes ces raisons, le facteur d'accélération de notre architecture est deux fois moins important en 3D-EM qu'en 3D-RP.

	Facteur d'accélération			Gain en efficacité		
	Proj	Rétro	Total	Proj	Rétro	Total
3D-RP	6	17.5	7.5	95	300	120
3D-EM	3	12	3.7	50	200	60

TAB. 6.6 – Facteur d'accélération et gains en efficacité entre notre système matériel de reconstruction et l'implémentation logicielle de STIR.

5 Conclusion

Dans ce chapitre, un système de reconstructions analytiques et itératives en tomographie TEP sur carte SoPCs a été présenté et évalué en terme de qualité et d'efficacité de reconstruction.

Ce système intègre une paire matérielle de projection par lancer de rayon et de rétroprojection par interpolation bilinéaire. Cette paire incohérente permet une convergence rapide de l'algorithme EM et offre une qualité de reconstruction proche des standards cliniques avec une erreur moyenne inférieure à 1%. Le facteur d'accélération espéré de cette paire matérielle sur un Virtex 4 est de 7 pour une reconstruction 3D-RP et de 3.7 pour une reconstruction 3D-EM par rapport aux outils logiciels classiques. Mais plus prometteur, cette paire offre une efficacité de deux ordres de magnitude plus importante pour les deux familles d'algorithmes. Cette architecture démontre ainsi un fort potentiel d'accélération et elle laisse espérer des facteurs d'accélération d'un ou deux ordres de magnitude grâce aux prochaines générations de FPGA ou bien encore via une implémentation ASIC.

Conclusion

Les travaux de thèse exposés dans ce manuscrit ont abouti à la conception d'une architecture matérielle efficace de rétroprojection 3D en imagerie TEP. Ces travaux participent ainsi à l'effort d'accélération matérielle de la reconstruction tomographique, indispensable à la diffusion à plus grande échelle des progrès des techniques d'imagerie médicale.

Dans la première partie, nous avons constaté que les systèmes de reconstruction constitués d'un simple PC accuse un retard technologique d'une dizaine d'année par rapport aux systèmes d'acquisition. Afin de bâtir des systèmes de reconstruction répondant aux besoins croissants en puissance de calcul de la reconstruction tomographique, deux voies de développement sont conjointement empreintées : la conception de noeuds (ou puces) de calcul performants et la constitution d'un réseau formé par plusieurs noeuds de calcul.

La première voie s'appuie d'une part sur le développement de processeurs spécifiques performants tels le Cell ou les GPUs qui bénéficient de la forte activité de l'industrie des jeux vidéos, et d'autre part des technologies FPGA/ASIC permettant de concevoir des architectures "sur mesure". Les architectures de la dernière génération des GPUs (série 8 du fabricant Nvidia) s'avèrent particulièrement adaptés aux calculs scientifiques. Elles intègrent sur une même puce plusieurs unités de calcul vectorielles, et constituent ainsi un bon compromis entre les architectures purement SIMD ou MIMD. Elles offrent les meilleurs facteurs d'accélération, de l'ordre d'un voire deux ordres de magnitude. Les architectures matérielles de type FPGA/ASIC offrent également des performances intéressantes. Toutefois nous avons pu constater que depuis 2004 leur performances sont en léger retrait par rapport à celle des processeurs graphiques. Par ailleurs, les stratégies mémoire de ces architectures, permettant de faire face au goulot d'étranglement que constitue la mémoire externe, sont basés soit sur l'utilisation d'un cache mémoire (cache 1D des CPU et cache 2D des GPUs), soit sur la mise en place d'un stratégies mémoire "ad hoc" (préchargements logiciels pour le Cell et préchargements matériels pour les architectures FPGA).

La deuxième voie permet d'obtenir un gain additionnel en performance quasi linéaire au nombre de noeuds de calcul. En effet, l'efficacité de parallélisation des algorithmes analytiques et itératifs sur machines multiprocesseurs à mémoire distribuée est très satisfaisante jusqu'à plusieurs dizaines voire centaines de noeuds de calcul ($\boldsymbol{\eta_{/\!/} > 0.8}$).

Dans la deuxième partie, nous avons décrit les choix algorithmiques, architecturaux et d'organisation des données effectués afin d'atteindre une efficacité optimum de reconstruction. Cette démarche d'adéquation algorithme, architecture et données, a été validée en terme de qualité et d'efficacité de reconstruction par l'implémentation sur un SoPC. Trois principes ont guidé la conception de cette architecture 3P : architecture en Pipeline, Préchargement mémoire et Parallélisation.

Tout d'abord, la rétroprojection 3D étant un algorithme massivement parallèle, le choix d'une architecture en pipeline s'avère tout naturel. Cependant afin de bénéficier efficacement de l'important débit de calcul offert par un pipeline de calcul (1 opéra-

tion/cycle), une stratégie d'accès mémoire visant à masquer la forte latence d'accès à la mémoire externe (5 cycles à 200 Mhz) a été mise en place. C'est pourquoi, nous avons intégré à notre architecture, un cache 3D Adaptatif et Prédictif permettant d'augmenter le débit d'accès à la mémoire externe. L'équilibre entre le débit de calcul et le débit mémoire est ainsi atteint pour une unité de rétroprojection. L'efficacité de reconstruction est alors proche de l'optimum (un cycle par opération). Dans ce but, une forte localité spatiale et temporelle est introduite en réordonnançant les boucles de calcul (reconstruction par bloc). Le fort taux de réutilisation des données mis en cache ainsi obtenu (compris entre 20 et 40) permet le masquage complet des préchargements mémoire commandées par l'analyseur statistique du cache 3D-AP. Au prix d'un taux de surcharge de l'ordre de trois (sur trois données chargées en cache, une est utile), le cache 3D-AP suit ainsi avec succès la sinusoïde 3D dessinée dans l'espace 3D des données de projection. Le taux de défaut du cache observé est très faible de l'ordre de 0,1%. Enfin, la parallélisation des unités de rétroprojection a permis d'augmenter significativement le pouvoir d'accélération de l'architecture 3P. Une hiérarchie mémoire a été utilisée afin de bénéficier de la localité spatiale et temporelle des données communes accédées par les différents blocs de voxels reconstruits. Un taux de réutilisation de 2 a été ainsi observé au niveau du cache racine de l'architecture 3P avec 8 unités de calcul en parallèle. Malgré les conflits d'accès au cache racine par les caches feuilles, des accélérations d'un facteur 3.2 et 4.5 ont été respectivement obtenues avec 4 et 8 unités, ce qui correspond à des efficacités de parallélisation respectives de $\boldsymbol{\eta_{/\!/} = 0.8}$ et $\boldsymbol{\eta_{/\!/} = 0.56}$.

Dans la troisième partie, nous avons comparé dans un premier temps les performances obtenues avec notre architecture de rétroprojection 3D à celles obtenues avec des processeurs classiques et graphiques. Dans un second temps, nous avons intégré notre rétroprojecteur 3D à un système complet de reconstruction.

Dans le chapitre 5, nous avons pu constater que les processeurs graphiques grâce à leur fort potentiel de parallélisation permettent un facteur d'accélération d'un ordre de magnitude en comparaison des processeurs classiques et de notre architecture matérielle. En effet, la technologie FPGA n'offre pas (ou n'offre plus depuis 2004) les ressources de calcul nécessaires pour concurrencer efficacement les processeurs graphiques dans le but d'accélérer les algorithmes massivement parallèles comme la reconstruction tomographique. Toutefois, notre architecture 3P offre le plus grand potentiel d'accélération grâce à une efficacité de reconstruction proche de l'optimal (1 cycle/op/UT) et d'un ordre de magnitude plus efficace que les processeurs classiques et graphiques. Notre architecture tire profit au maximum des ressources d'accès à la mémoire externe grâce au cache 3D-AP. En effet, lorsque les caches des processeurs classiques et les caches des GPUs n'effectuent respectivement qu'une prédiction 1D ou 2D des accès mémoire, le cache 3D-AP effectue quant à lui une prédiction 3D. Par ailleurs, nous avons constaté expérimentalement que les caches 1D se contentent d'amortir le coût des premiers accès aux données lorsqu'une prédiction 2D ou 3D permettrait un masquage des accès mémoire. Nous avons ainsi évalué à 10 à 20 %, le gain en performance du cache 3D-AP par rapport aux caches classiques.

Dans le chapitre 6, notre rétroprojecteur matériel a été couplé avec un module matériel de projection par lancer de rayon. afin de former une paire matérielle de projection/rétroprojection. Cette paire permet de construire un système complet de reconstruction sur une carte SoPC. Ce système effectue les reconstructions analytiques (algorithme 3D-RP) et itérative (algorithme 3D-EM) à l'aide de quelques étapes de calcul effectuées en logiciel sur le processeur du PC hôte ou bien à l'aide de modules matériels basiques. Si cette paire implémentée sur un SoPC avec une fréquence à 200 Mhz permettrait un facteur d'accélération assez limité (entre 3.5 et 7) par rapport à une solution logicielle standard, elle offre une efficacité de deux ordres de magnitude plus importante pour une qualité de reconstruction plus que correcte (erreur moyenne inférieure à **1%**). Cette architecture démontre ainsi un fort potentiel d'accélération et elle laisse espérer des facteurs d'accélération d'un ou deux ordres de magnitude grâce aux prochaines générations de FPGA ou bien encore via une implémentation ASIC.

Perspectives

La parallélisation des calculs constitue un levier essentiel d'accélération. Dans cette étude, nous nous sommes concentré sur la parallélisation sur une puce SoPC et nous avons obtenu une parallélisation satisfaisante. Toutefois un gain additionnel en performance est encore possible à l'aide d'une meilleure stratégie de parallélisation des unités de calcul sur une puce SoPC mais également via la mise en place d'une parallélisation des calculs sur une carte multi SoPCs.

Nous avons pu constater que la parallélisation sur une puce est limitée par l'accès par les caches feuilles du cache 3D-AP au cache racine. Deux voies de développement sont envisageables. Premièrement, une meilleure synchronisation des unités de rétroprojection réduisant les conflits d'accès, permettrait un gain en performance. Deuxièmement, une architecture vectorielle sans hiérarchie mémoire permettrait de contourner le problème lié aux conflits d'accès au bus mémoire. Le pipeline de cette architecture vectorielle n'effectuerait plus la mise à jour d'un seul voxel mais celle d'un vecteur de voxel (`xn` et `yn` fixés et `zn`=0 ... $\texttt{zn}_{max} - 1$). L'accès aux données se ferait par vecteurs de bins (`rho` et `phi` fixés et `lambda`=0 ... $\texttt{lambda}_{max} - 1$). Le pipeline effectuerait toujours une reconstruction par blocs (ici de vecteurs de voxels), afin de bénéficier de la localité spatiale et temporelle des données. Le degré de parallélisation de cette architecture serait élevée avec un pipeline vectoriel traitant simultanément $\texttt{zn}_{max}$ voxels ($\texttt{zn}_{max}$=63 pour les volumes standards en TEP). Un cache 2D-AP serait suffisant puisque les accès se feraient alors dans l'espace 2D (`phi`,`rho`) des vecteurs de bins. Lors de la rétroprojection avec interpolation bilinéaire, quatre vecteurs de bins (c'est-à-dire quatre accès au cache vectoriel) permettrait ainsi de faire en parallèle la mise à jour d'un vecteur de voxels. L'accès aux bins regroupés dans un vecteur, éviterait ainsi les conflit d'accès aux données entre unités de calculs.

La parallélisation sur une carte multi SoPC est un objectif majeur du projet Archi-TEP. La carte conçue par le laboratoire LPSC possède sept Virtex 4 : un central jouant le rôle de chef d'orchestre et six dédiés au calcul. Comme nous l'avons exposé dans la

première partie de ce manuscrit, les stratégies de parallélisation DEI ou DEP pour les algorithmes itératifs offrent une efficacité semblable lorsque une dizaine de noeuds de calcul est utilisée (au-delà la stratégie DEI est préférable). Ainsi, une stratégie DEI ou DEP pourra être choisie de manière arbitraire sur la carte multi SoPCs du projet Archi-TEP.

Notre étude s'est essentiellement focalisée sur la tomographie TEP. Mais comme nous l'avons souligné en début de ce manuscrit, la rétroprojection à faisceaux coniques utilisée en tomographie CT et celle à faisceaux parallèles utilisée en tomographie TEP, comportent une séquence de calculs et d'accès mémoire proche. Ainsi notre architecture 3P de rétroprojection s'appuyant sur la technologie flexible des SoPCs, est tout à fait adaptable pour la reconstruction tomographique CT. Par ailleurs, les méthodes de reconstructions IRM s'appuyant sur le théorème de la coupe centrale sont susceptibles d'être implémentée sur une architecture semblable à celle proposée dans cette étude.

Les travaux de recherche menés durant cette thèse ont validé l'emploi du cache nD-AP dans le cadre bien défini de la rétroprojection 3D. Cette application a constitué un premier véritable *testbench* du cache nD-AP. Validée en pratique, l'utilisation du cache nD-AP est à présent envisageable pour un large champs d'application. Toute application effectuant un parcours mémoire continue dans un espace n dimensionnel avec une forte localité spatiale et temporelle, est éligible à l'emploi du cache nD-AP. La forte efficacité de calcul ainsi obtenue, d'un ordre de magnitude supérieure à celle des CPUs et GPUs, est synonyme d'une basse consommation. Le cache nD-AP offre ainsi une solution technologique pertinente lorsque la consommation est une contrainte forte.

Quatrième partie

Annexes

Annexe A

Correction en arc

La correction d'arc (correction géométrique) est réalisée dans les systèmes TEP modernes pour prendre en compte l'échantillonnage non-uniforme sur $\boldsymbol{\rho}$. En effet, avec la géométrie circulaire du scanner, le pas d'échantillonnage $\boldsymbol{\delta_\rho}$ diminue graduellement en allant du centre vers les bords du scanner selon $\boldsymbol{i_\phi}$ (voir équation A.1). Or nos modélisations sont basées sur la transformée de Radon qui serait elle plutôt échantillonnée régulièrement suivant $\boldsymbol{\rho}$.

$$\boldsymbol{\rho} = \mathbf{sin}(\mathtt{rho} * \boldsymbol{\delta_\phi}) \tag{A.1}$$

Ainsi la correction d'arc se charge d'estimer des valeurs de projections équidistantes à partir des données mesurées espacées non-uniformément. Pour cela, on définit un pas d'échantillonnage $\boldsymbol{\delta_\rho} \leq \boldsymbol{\delta_{\rho_{max}}}$ et un nombre de capteur $\boldsymbol{\rho_{max}} \leq \dfrac{\boldsymbol{\phi_{max}}}{\mathbf{2}}$. Par exemple, pour le scanner ECAT HR+ et avec la correction d'arc du logiciel STIR, le nombre de capteurs est réduit de $\dfrac{\mathtt{phi}_{\mathtt{max}}}{\mathbf{2}} = \mathbf{288}$ à $\mathtt{rho}_{\mathtt{max}} = \mathbf{269}$. L'équation de discrétisation de $\boldsymbol{\rho}$ devient donc :

$$\boldsymbol{\rho} = \mathtt{rho} * \boldsymbol{\delta_\rho} \quad \text{avec} \quad -\frac{\mathtt{rho}_{\mathtt{max}} - \mathbf{1}}{\mathbf{2}} \leq \mathtt{rho} < \frac{\mathtt{rho}_{\mathtt{max}} - \mathbf{1}}{\mathbf{2}} \tag{A.2}$$

Plusieurs méthodes sont proposées par Buchert [Buchert 00]. Celle qui nous intéresse est la correction d'arc par ré-échantillonnage (aussi appelée *overlapping interpolation* [Hamill 02]). Il s'agit de répartir les données d'acquisition sur une échelle uniforme suivant $\boldsymbol{\rho}$. Pour réaliser la correction d'arc, imaginons que les données d'acquisition soient réparties dans des cases de taille différentes, et les données corrigées en arc dans des cases de taille égales. Maintenant on aligne ces cases l'une en dessous de l'autre. Chaque case corrigée en arc est remplie avec les cases non-corrigées la chevauchant pondérées par la "proportion de chevauchement". Par exemple, si deux cases non-corrigées rentrent dans une case corrigée, cette dernière recevra la somme des deux cases non-corrigées. Si seulement $\mathbf{1/3}$ de la case non-corrigée chevauche la case corrigée, il sera attribué à cette dernière $\mathbf{1/3}$ de la première ajoutée aux autres cases chevauchantes. La totalité des données d'acquisition est ainsi préservée.

Annexe B

Compression span et Michelogram

La compression axiale (*span*) regroupent plusieurs sinogrammes en un seul. Le sinogramme résultant de la somme des sinogrammes représente un plan. Le span est la somme du nombre de sinogrammes d'un plan direct et de celui d'un plan indirect. Le nombre de sinogrammes d'un plan direct est toujours impaire et celui d'un plan indirect paire. Par exemple, pour un span de 3=1+2, les plans directs sont constitués d'un seul sinogramme et les plans indirectes de deux sinogrammes. Pour un span de 9=5+4, comme pour le ECAT HR+, les plans directs sont constitués de 5 sinogrammes et les plans indirects de 4 sinogrammes.

Pour simplifier la représentation des sinogrammes selon $(\boldsymbol{\lambda}, \boldsymbol{\Delta})$, on utilise souvent un "michelogram". Le michelogram représente l'ensemble des sinogrammes associées chacun à un couple d'anneaux. Le michelogram du ECAT HR+ avec un span de 9 est représenté en figure B.1. Sur ce michelogram, sont représentées en noir les sinogrammes acquis par le scanner et en rouge ceux estimés par projection 3D lors de l'étape de complétion des données de l'algorithme 3D-RP. C'est l'ensemble des sinogrammes qui sont rétroprojetés lors de la dernière étape de l'algorithme 3D-RP. Pour les algorithmes itératifs, seuls les sinogrammes acquis (en noir sur la figure) sont utilisés.

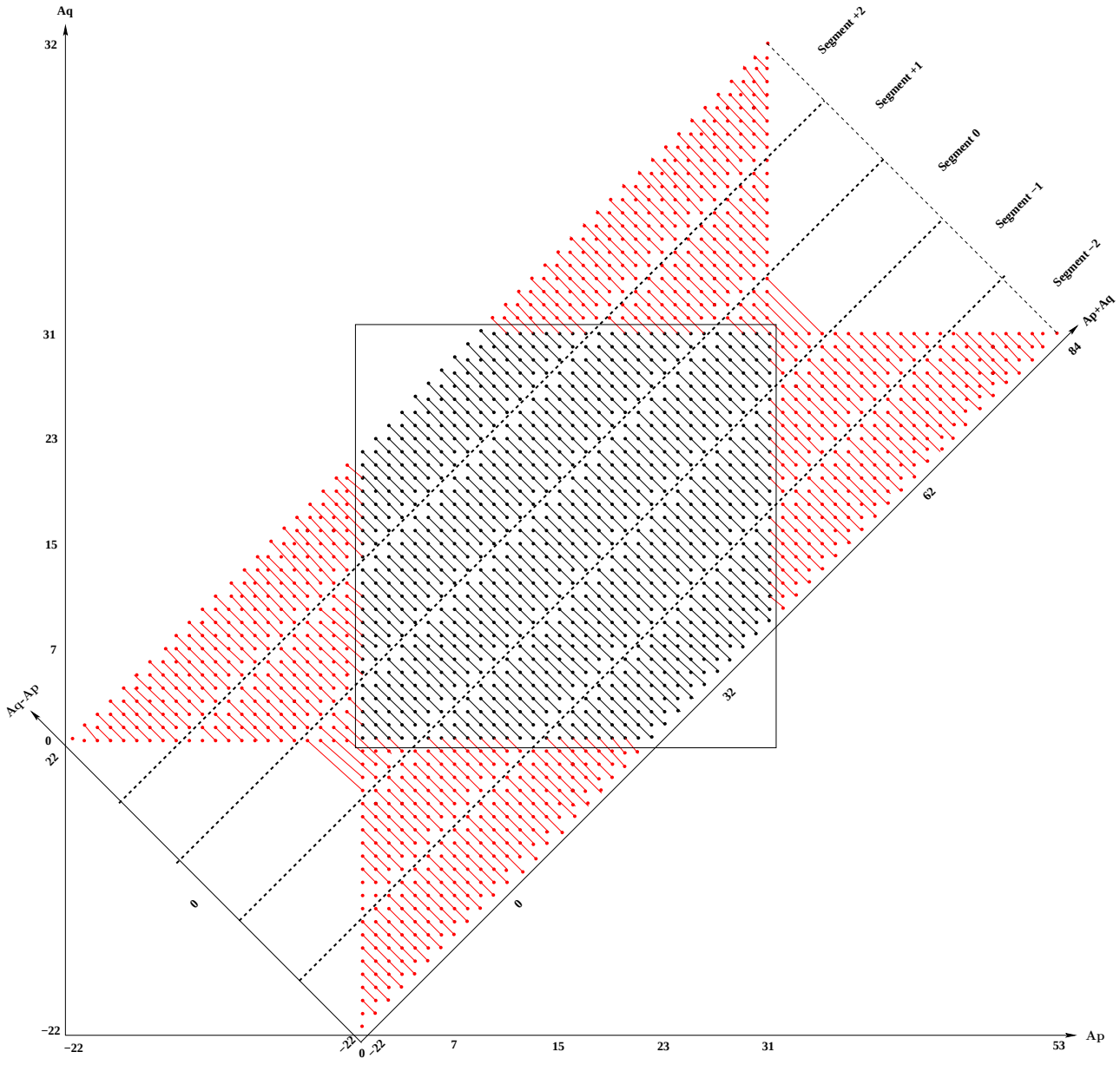


FIG. B.1 – Michelogram du scanner HR+. Avec un span de 9, 5 segments avec 239 sinogrammes acquis (en noir) et 162 sinogrammes estimées (en rouge) ont ainsi définis. Le michelogram représente les plans selon les indices d'anneaux $\boldsymbol{A_p}$ et $\boldsymbol{A_q}$. Nous avons fait apparaître également les axes $\boldsymbol{\lambda} = \dfrac{\boldsymbol{A_p} + \boldsymbol{A_q}}{\boldsymbol{2}}$ et $\boldsymbol{\Delta} = \boldsymbol{A_p} - \boldsymbol{A_q}$.

Annexe C

Temps de reconstruction pour la rétroprojection

Les tableaux C et C regroupent les différents de temps de reconstruction présentés dans la littérature, pour respectivement la rétroprojection 2D à faisceaux parallèles (tomographie TEP ou CT) et la rétroprojection 3D à faisceaux coniques (tomographie CT).

	Contexte			Accélération matérielle		
	Interp.	Commentaire	Temps CPU	Architecture	Précis.	Temps (/CPU)
[Schiwietz 06]	avec	avec iFFT	22.6 s	GPU	f32	176 ms (*128)
[Leeser 05]	avec	16 pipelines	4.66 s	FPGA	i09	125 ms (*37)
	avec	Virtex 2000E				
	avec	(65 MHz)				
[Xue 06]	avec		7.13 s	FPGA	i32	273 ms (*26)
	avec			GPU	i32	295 ms (*24)
	avec			GPU	i16	143 ms (*51)
[Yang 07]	sans	CUDA		GPU	f32	19.2 ms
	avec	CUDA		GPU	f32	24.6 ms
	avec	CUDA+TEX		GPU	i09	8.8 ms
[Kachelriess 07]	sans	version CPU	4.1 s	Cell	f32	6.1 ms (*672)
	avec	non optimisée	5.2 s	Cell	f32	7.9 ms (*658)
[Kachelriess 07]	sans	version CPU	68 ms	Cell	f32	6.1 ms (*11)
	avec	optimisée	93 ms	Cell	f32	7.9 ms (*12)

TAB. C.1 – Temps de reconstruction pour la rétroprojection 2D à faisceaux parallèles

	Contexte			Accélération matérielle			
	Interp.	Commentaire	Temps CPU	Archi	Impl.	Précis.	Temps (/CPU)
FDK SANS FILTRAGE							
[Riabkov 07]	avec			GPU		f32	12.8 s
[Yang 07]	avec			GPU	CUDA+TEX	f32	6.7 s
[Kachelriess 07]	avec	Xeon 3 GHz	1.93 h	Cell	Directe	f32	25.5 s (*255)
	avec	(version CPU non opt.)		Cell	Hybride	f32	12.7 s (*511)
[Kachelriess 07]	avec	Xeon 3 GHz	3.21 mn	Cell	Directe	f32	25.5 s (*7)
	avec	(version CPU hybride)		Cell	Hybride	f32	12.7 s (*14)
[Heigl 07]	avec	dual core Intel	3.3 mn	FPGA	8+1 V4 (200 MHz)	i ?	3.8 s (*51)
FDK AVEC FILTRAGE							
[Yu 01]	avec	Pentium 3 (500 MHz)	51.1 mn				
[Exxim 07]	sans	Dual Pentium 3 GHz	1.9 mn				
	avec		3.8 mn				
[Terarecon]	avec	1 Dual Xeon 3 GHz	5.9 mn	ASIC	1 XTrillion 3.0 (333 MHz)	f32	2.2 mn (*2.3)
	avec	4 Dual Xeons 3 GHz	2.6 mn	ASIC	4 XTrillions 3.0 (333 MHz)	f32	33.3 s (*4.3)
[Laurent 98]	avec	SUN4 (16.67 MHz)	110.9 h	// MD	16 CPUs	f32	3.7 h (*30)
	avec			// MD	SP1 (32 CPUs)	f32	1.1 h (*100)
	avec			// MD	Cray T3D	f32	13.3 mn (*500)
[Goddard 02]	avec			FPGA	1 carte	i16	66.0 s
	avec			FPGA	2 cartes	i16	36.5 s
[Li 04b]	avec			FPGA	SIMULATION V2 Pro (150 MHz)	i ?	57.2 s
[Brasse 05]	avec	Dual Xeon 3 GHz	44.4 mn	// MD	12 dual-Xeons 3 GHz	f32	2.8 mn (*16)
[Vaz 07]	avec	Temps Transferts		GPU	Q1 Directe	f32	11.9-19.0 s
	avec	CPU/GPU inclus		GPU	Q2 Limitée	f32	7.2-11.3 s
[Xu 07]	avec			GPU	CUDA	f32	35.3 s
	avec			GPU	AG	f32	12.7 s
	avec			GPU	AG (EFK)	f32	9.7 s
[Scherl 07b]	sans			Cell		f32	14.6 s
	sans			dual Cell Blade		f32	7.5 s
	avec			Cell		f32	26.0 s
	avec			dual Cell Blade		f32	13.2 s
[Heigl 07]	avec	dual core Intel ? GHz	3.6 mn	FPGA	8+1 V4 (200 MHz)	i ?	9.0 s (*24)
[Schiwietz 07]	avec	Xeon dual 3 GHz (pré et post traitements)	8.4 mn	GPU		f32	2.6 mn (*3.2)
KATSEVICH AVEC FILTRAGE							
[He 06]	avec	V=$\mathbf{128^3}$, P=$\mathbf{701 * 150 * 60}$	1.9 mn	// MD	16 PCs hétérogènes	f32	10.9 s (*10)
[Deng 06a]	avec	V=$\mathbf{512^3}$, P=$\mathbf{3501 \cdot 500 \cdot 70}$	1.7 h	// MD	16 bi-Opterons	f32	3.0 mn (*32)
[Ni 06]	avec	V=$\mathbf{512^3}$, P=$\mathbf{3501 \cdot 500 \cdot 70}$	1.7 h	// MD	300 processeurs!!!	f32	20.0 s (*300)

TAB. C.2 – Temps de reconstruction pour la rétroprojection 3D à faisceaux coniques

Annexe D

Temps de reconstruction pour les algorithmes itératifs

Les tableaux D et D regroupent les différents de temps de reconstruction présentés dans la littérature, pour respectivement les algorithmes itératifs 3D en tomographie CT et ceux en tomographie TEP.

	Contexte					Accélération matérielle		
	Algo	Iter	Volume	Projection	CPU (Proj/Retro)	Archi	Commentaires	T (/CPU)
[Laurent 98]	ART	1	$\mathbf{64^3}$	$\mathbf{64^3}$	20.2 mn (Calcul de H en ligne) SUN4 16.67 MHz	// MD // MD // MD	16 CPUs SP1 (32 CPUs) Cray T3D	$\mathbf{35.6}\boldsymbol{s}$ (*34) $\mathbf{28.2}\boldsymbol{s}$ (*43) $\mathbf{22.9}\boldsymbol{s}$ (*53)
[?]	EM	1	$\mathbf{12 * 252^2}$	$\mathbf{310 \cdot 15 \cdot 12}$	3.7 mn (Calcul de H en ligne)	// MD	8 PCs (*Peer2Peer*)	45 s (*4.9)
[He 06]	OSEM	10 (10)	$\mathbf{128^3}$	$\mathbf{701 \cdot 150 \cdot 60}$	16.4 mn (Calcul de H en ligne)	// MD	10 PCs (*Peer2Peer*)	5.3 mn (*3.0)
[Gregor 07]	PSIRT	2 (45)	$\mathbf{1022 * 512^2}$	$\mathbf{1022 \cdot 512^2}$	Calcul H en ligne (interp. tri-linéaire)	// MD	20 bi-dual Opt. (1.8 GHz)	10.3 mn
[Keesing 06]	AM-OS	1 (320)	$\mathbf{320 * 512^2}$	$\mathbf{320 \cdot}$ $\mathbf{1160 \cdot 672}$	$\sim$3 200 h (H pré-calculée) CPU 900 MHz	// MLP	32 CPUs	$\sim \mathbf{160}\boldsymbol{h}$ (*20)
[Kole 05]	OSC OSC	1 (10) 1 (10)	$\mathbf{400 * 512^2}$	$\mathbf{400 * 512^2}$	59 h (*Ray-tracing*) Itanium2 1.3 GHz	// MLP // MD	32 Itanium2 40 Itanium2	2.4 h (*25) 1.8 h (*33)
[Kole 04]	OSC OSC	10 (50) 10 (50)	$\mathbf{256^3}$	$\mathbf{256^3}$	120 h (*Ray-tracing*) Athlon 2.0 GHz	GPU GPU	sans interp. Bi-lin.	34 mn (*149) 48 mn (*222)
[Knaup 06]	OSC	4	$\mathbf{512^3}$	$\mathbf{512^3}$		Cell		3.2 mn
[Tita 07]	SART	3	$\mathbf{256^3}$	$\mathbf{90 * 568^2}$		GPU		$\sim$1 mn

TAB. D.1 – Temps de reconstruction pour les algorithmes itératifs 3D en tomographie CT

	Contexte					Accélération matérielle		
	Algo	Iter	Volume	Projection	CPU (Proj/Retro)	Archi	Commentaires	T (/CPU)
[Shattuck 02]	PCG	30	$\mathbf{63 \cdot 128^2}$	$\mathbf{9.9 \cdot 10^6}$	1.6 h (H pré-calculée)	// MD	9 P3 0.9 GHz	19 mn (*5.0)
[Herraiz 06]	OSEM	3 (50)	$\mathbf{62 * 175^2}$	$\mathbf{28.8 \cdot 10^6}$	1.6 h (H=150 Mo) Opteron 1.8 GHz	// MD	1+8 Opterons 1.8 GHz	15 mn (*6.5)
[Jones 06]	OSEM	1 (16)	$\mathbf{207 * 256^2}$	$\mathbf{469 \cdot 10^6}$	$\sim$16 h (*Ray-tracing/VBI*) dual Xeons 3.4 GHz	// MD	32 dual Xeons 3.4 GHz	31 mn (*$\sim$30)
[Schellmann 07]	OSEM	1 (10)	$\mathbf{280 * 150^2}$	$\mathbf{10 \cdot 10^6}$	4.2 mn (*Ray-tracing*) Xeon 3.2 GHz	// MD	64 Xeons 3.2 GHz // PSD	6 s (*42)
[Schellmann 07]	OSEM	1 (10)	$\mathbf{280 * 150^2}$	$\mathbf{10 \cdot 10^6}$	5.3 mn (*Ray-tracing*) Xeon 3.2 GHz	// MD	64 Xeons 3.2 GHz // ISD	54 s (*6)
[Chidlow 03]	EM	80	$\mathbf{128^3}$	$\mathbf{10^6}$	14,1 mn (*Ray-tracing/VBI*)	GPU		1.6 mn (*8.7)
	OSEM	10 (8)			2.2 mn (*Ray-tracing/VBI*) Athlon 2.0 GHz	GPU		30.8 s (*4.2)
[?]	2D OSEM	2 (32)	$\mathbf{128^3}$	$\mathbf{28.8 \cdot 10^6}$	2 mn (*Ray-tracing*)			
	OSEM	2 (32)			1.5 h (*Ray-tracing*)	GPU	Tri-lin.	2.1 mn (*44)
	OSEM	2 (32)			1.5 h (*Ray-tracing*)	GPU	TOR gaus.	5.7 mn (*16)
	OSEM	2 (32)			15 h (*Ray-tracing* 10X)	GPU	Tri-lin.	2.1 mn (*441)
	OSEM	2 (32)			15 h (*Ray-tracing* 10X)	GPU	TOR gaus.	5.7 mn (*162)
[Smith 07]	liste-OSEM	2 (3)	$\mathbf{150^3}$	$\mathbf{5.8 \cdot 10^6}$	22 mn (*Ray-tracing*) Mono Opteron 2.6 GHz	// MP	4 dual Opteron	4.6 mn (*4.8)
[Ortuno 06]	OSEM	1 (10)	$\mathbf{140 * 112^2}$	$\mathbf{1.3 \cdot 10^6}$	12.9 mn (H=596 Mo) P4 3.2 GHz			
[Scheins 06]	OSEM	1 (12)	$\mathbf{64 * 116^2}$	$\mathbf{4.0 \cdot 10^6}$	1.52 mn (H=0.9 Go) Pentium 3.0 Ghz			
[Kadrmas 05]	lor-OSEM	4 (14)	$\mathbf{81 \cdot 128^2}$	$\mathbf{14.9 \cdot 10^6}$	1.8 mn (*Rotate-and-Slant*) Opteron 2.4 GHz			
[Kudrolli 02]	EM	25	$\mathbf{60 \cdot 120^2}$	$\mathbf{23.6 \cdot 10^6}$	5.8 h (*Stochastic*) Pentium 433 MHz			

TAB. D.2 – Temps de reconstruction pour les algorithmes itératifs 3D en tomographie PET

Annexe E

Outils logiciels du projet ArchiTEP

Les figures E.1, E.2, E.3 et E.4 présentent la chaîne de traitement pour effectuer les reconstructions analytiques et itératives. Les outils logiciels proviennent de SIEMENS, de STIR, de la librairie PET-TURKU et de la librairie logicielle du projet ArchiTEP. Voila ci dessous, la liste des outils logiciels utilisés :

SIEMENS :	*bkproj* du CERMEP (pré corrections des données)
STIR :	*FBP3DRP* (rétroprojection filtrée 3DRP) *FBP2D* de (rétroprojection filtrée 2D) *FBP3DRP_précorrection* (projection 3D) *FBP2D_précorrection* (correction d'arc et filtrage rampe) *filtrage_colsher* *ifheaders_for_ecat7* (format ECAT -> format Interfile)
ArchiTEP :	*retro3d* (rétroprojection 3D) *projLIS_3DRP* (projection 3D) *Algo_IT* (algorithme EM et OSEM)
PET-TURKU :	*ecat2flo* (format ECAT -> format raw)

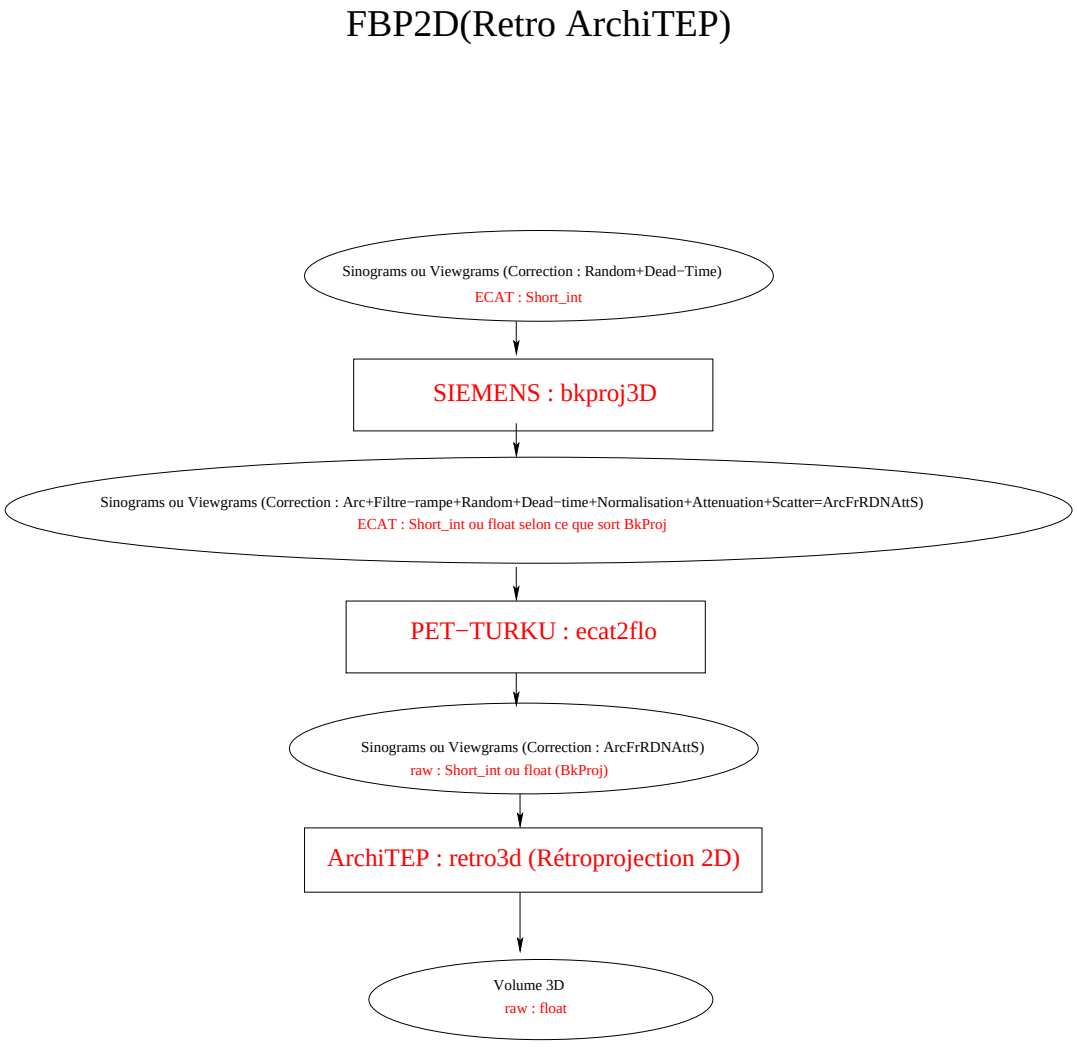


FIG. E.1 – Organigramme des outils logiciels utilisés pour la rétroprojection filtrée 2D

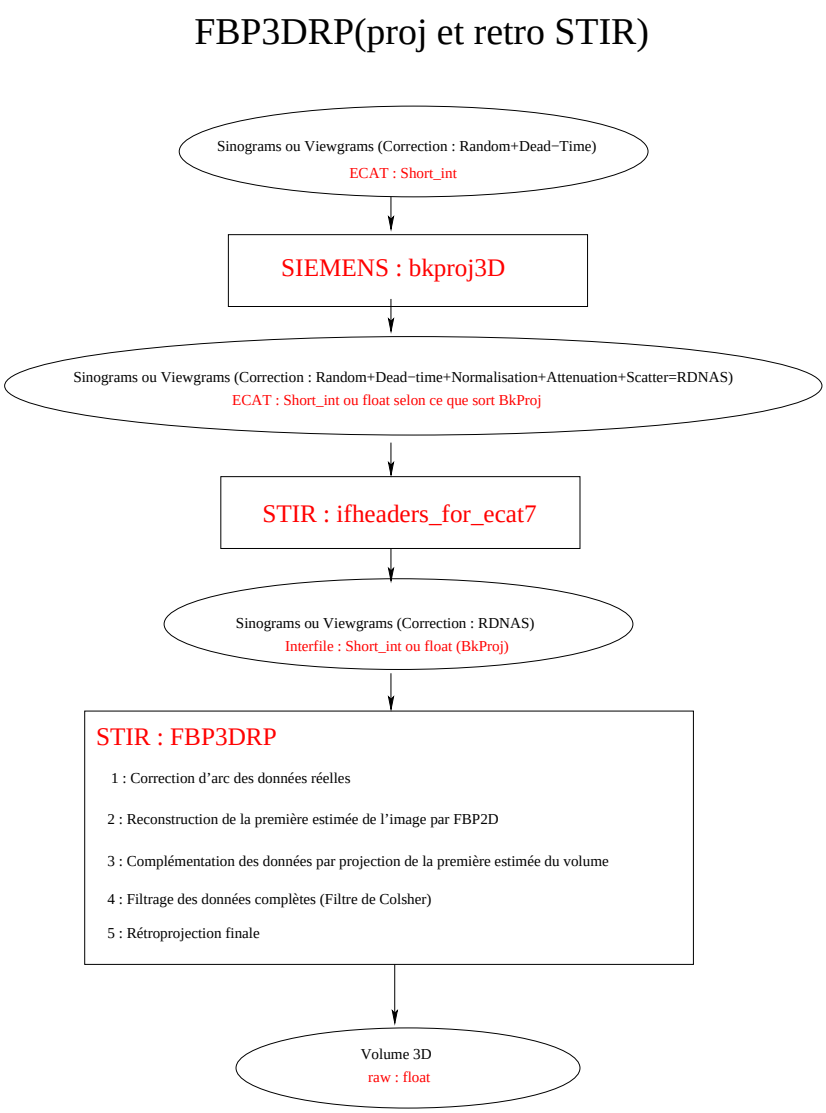


FIG. E.2 – Organigramme des outils logiciels utilisés pour l'algorithme 3DRP (projection et rétroprojection avec STIR)

FBP3DRP(proj STIR,retro ArchiTEP) FBP3DRP(proj et retro ArchiTEP)

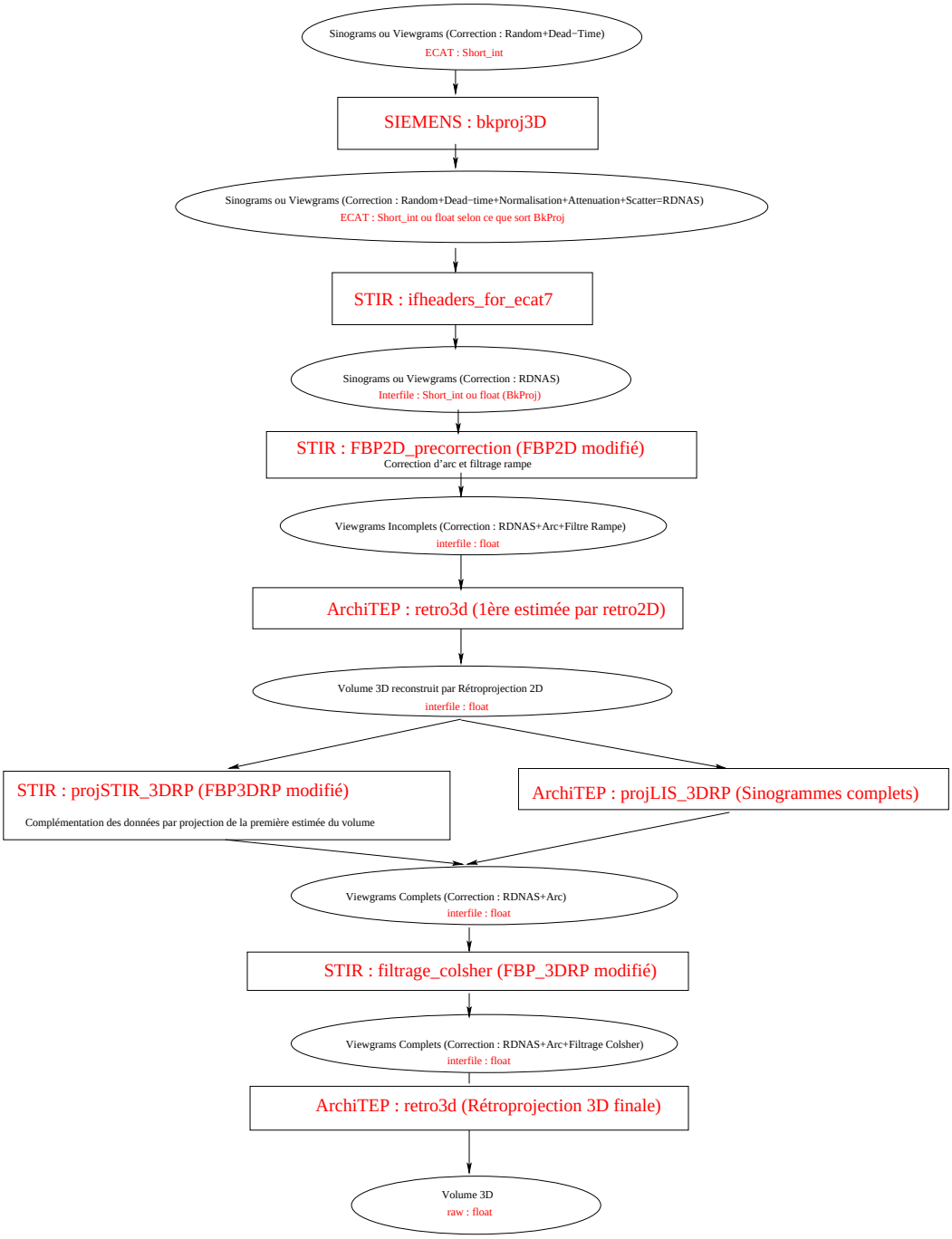


FIG. E.3 – Organigramme des outils logiciels utilisés pour l'algorithme 3DRP (projection avec STIR ou ArchiTEP et rétroprojection avec ArchiTEP)

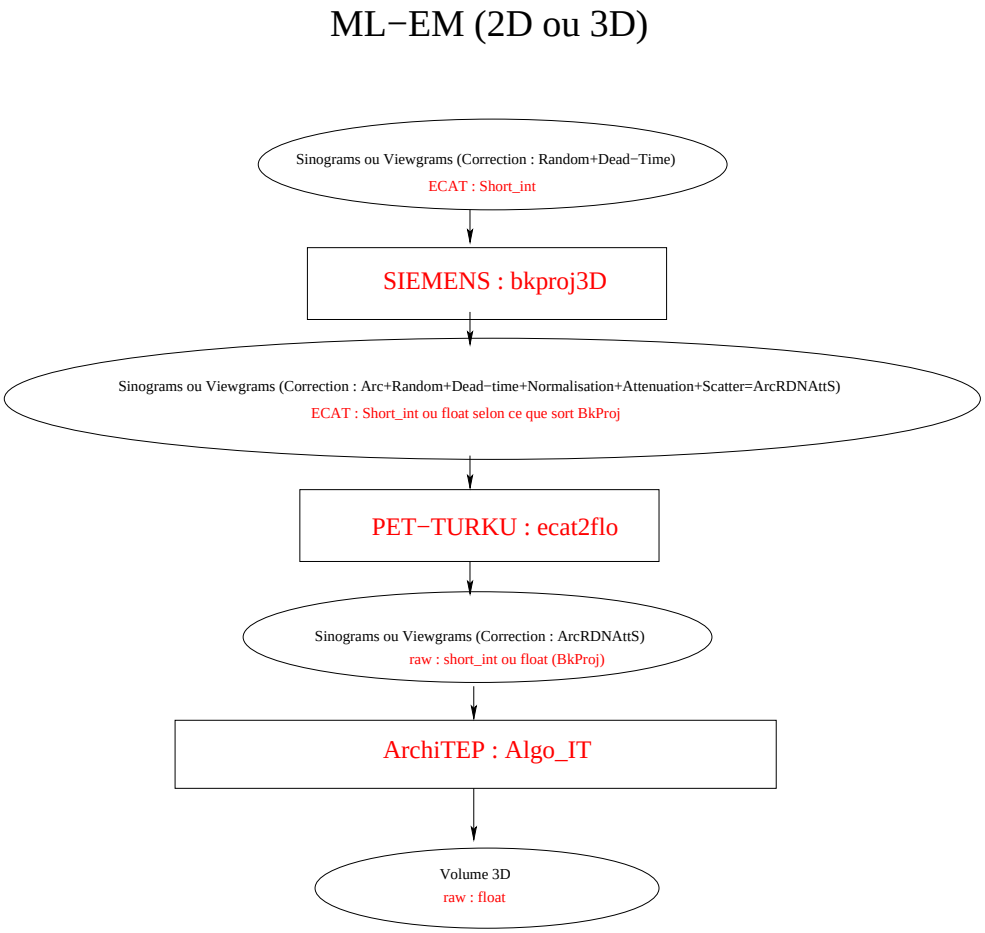


FIG. E.4 – Organigramme des outils logiciels utilisés pour les algorithmes itératifs

Annexe F

Publications

Revues internationales

- **Algorithm Architecture Adequacy for High Speed 3D Tomography**
  N. Gac, S. Mancini, M. Desvignes and D. Houzet
  (accepté en Septembre 2008) EURASIP Journal on Embedded Systems

Conférences internationales

- **A Hardware Projector/Backprojector Pair for 3D PET Reconstruction**
  N. Gac, S. Mancini, M. Desvignes, F. Deboissieu and A. Reilhac
  (Physics of Medical Imaging conference proceedings, vol. 6913, SPIE Digital library San Diego, United States, February 2008 (Poster)

- **Algorithm Architecture Adequacy for High Speed 3D Tomography**
  N. Gac, S. Mancini, M. Desvignes and D. Houzet
  ECSI Design and Architecture for Signal and Image Processing (DASIP) Grenoble, France, November 2007 (présentation orale)

- **Hardware/software 2D-3D backprojection on a SoPC platform**
  N. Gac, S. Mancini et M. Desvignes
  Proceedings of the 2006 ACM symposium on Applied computing, vol.4, pp. 222-228 Dijon, France, April 2006 (présentation orale)

Revue nationale

- **Application d'un cache 2D prédictif à l'accélération de la rétroprojection TEP 2D**
  S. Mancini, **N. Gac**, M. Desvignes, O. Bourrion et O. Rosseto
  Traitement du Signal, 2006, vol. 23, n 5-6-NS, p. 391-404

Conférences nationales

- **Une paire matérielle projecteur/rétroprojecteur pour la reconstruction TEP 3D**
  N. Gac, S. Mancini et M. Desvignes
  2ème Colloque du GDR System On Chip - System In Package (SOC-SIP)
  Paris, France, Juin 2008 (poster)

- **Rétroprojection 2D sur plateforme SoPC**
  S. Mancini, **N. Gac** et M. Desvignes
  20ème Colloque sur le traitement du signal et des images (GRETSI),
  pp. 1060-1063
  Louvain-La-Neuve, Belgique, Septembre 2005 (présentation orale)

- **Rétroprojection 2D sur plateforme SoPC, premiers résultats**
  S. Mancini, **N. Gac** et M. Desvignes
  Journées Sciences, Technologies et Imagerie pour la Médecine (JSTIM'05),
  pp. 132-136
  Nancy, France, Mars 2005 (Poster)

- **Etude d'un cache 2D adaptatif et prédictif pour le traitement d'image**
  S. Mancini, **N. Gac** et M. Desvignes
  Journées Francophones sur l'Adéquation Algorithme Architecture (jFAAA'05),
  pp. 260-265
  Dijon, France, Janvier 2005 (présentation orale)

Bibliographie

[Basu 00] S. Basu & Y. Bresler. O(Nog) filtered backprojection reconstruction algorithm for tomography. Image Processing, IEEE Transactions on, vol. 9, no. 10, pages 1760–1773, Oct. 2000.

[Bockenbach 07] O. Bockenbach, S. Schuberth, M. Knaup & M. Kachelriess. High Performance 3D Image Reconstruction Platforms ; State of the Art, Implications and Compromises. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, 2007.

[Brady 98] Martin L. Brady. A Fast Discrete Approximation Algorithm for the Radon Transform. SIAM J. Comput., vol. 27, no. 1, pages 107–119, 1998.

[Brasse 04] D. Brasse, P.E. Kinahan, R. Clackdoyle, M. Defrise, C. Comtat & D.W. Townsend. Fast fully 3-D image reconstruction in PET using planograms. Medical Imaging, IEEE Transactions on, vol. 23, no. 4, pages 413–425, April 2004.

[Brasse 05] D. Brasse, B. Humbert, C. Mathelin, M.-C. Rio & J.-L. Guyonnet. Towards an inline reconstruction architecture for micro-CT systems. Physics in Medicine and Biology, vol. 50, pages 5799–5811, December 2005.

[Bresler 07] Y. Bresler & A.K. George. Shear-Based Fast Hierarchical Backprojection for Parallel-Beam Tomography. Medical Imaging, IEEE Transactions on, vol. 26, no. 3, pages 317–334, March 2007.

[Buchert 00] R. Buchert, K. H. Bohuslavizki, H. Fricke, J. Mester & M. Clausen. Performance evaluation of PET scanners : testing of geometric arc correction by off-centre uniformity measurement. Eur J Nucl Med, vol. 27, no. 1, pages 83–90, Jan 2000.

[Cabral 94] Brian Cabral, Nancy Cam & Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In VVS '94 : Proceedings of the 1994 symposium on Volume visualization, pages 91–98, New York, NY, USA, 1994. ACM Press.

[Chen 90] C.M. Chen, S.-Y. Lee & Z.H. Cho. Parallelization Of The EM Algorithm For 3D PEt Image Reconstruction. In Nuclear Science Sym-

posium, 1990. Conference record : Including Sessions on Nuclear Power Systems and Medical Imaging Conference, 1990 IEEE, pages 1550–1555, 22-27 Oct. 1990.

[Chidlow 03] K. Chidlow & T.Moller. Rapid Emission Tomography Reconstruction. In Proc. Int. Work. Volume Graphics (VG'03), Tokyo, Japan, July 2003.

[Cho 90] Z.H. Cho, C.M. Chen & S. Lee. Incremental algorithm-a new fast backprojection scheme for parallel beam geometries. Medical Imaging, IEEE Transactions on, vol. 9, no. 2, pages 207–217, June 1990.

[Deboissieu 06] F. Deboissieu. Reconstruction itérative 3d tep. Master's thesis, Master SIPT, 2006.

[DeMan 04] B. DeMan & S. Basu. Distance-driven projection and backprojection in three dimensions. Physics in Medicine and Biology, vol. 49, pages 2463–2475, June 2004.

[DeMan 07] B. DeMan, R. Nilsen & E. Drapkin. High Performance Image Reconstruction and Implementation. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, pages 13–16, 2007.

[Dempster 77] Arthur P. Dempster, Nan M. Laird & Donald B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. Journal of the Royal Statistical Society, vol. 39, no. 1, pages 1–38, 1977.

[Deng 06a] J. Deng, H. Yu, T. He, S. Zhao, L. Wang & G. Wang. A Parallel Implementation of the Katsevich Algorithm for 3-D CT Image Reconstruction. The Journal of Supercomputing, vol. 38, no. 1, pages 35–47, October 2006.

[Deng 06b] J. Deng, H. Yu, J. Ni, L. Wang & G. Wang. Parallelism of iterative CT algorithm based on local reconstruction. In Developments in X-Ray Tomography V. Edited by Bonse, Ulrich. Proceedings of the SPIE, Volume 6318, pp. 63181P (2006)., volume 6318 of Presented at the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference, August 2006.

[Despres 07] P. Despres, M. Sun, B. H. Hasegawa & S. Prevrhal. FFT and cone-beam CT reconstruction on graphics hardware. In Proc. SPIE Vol. 6510 [Medical Imaging 2007 : Physics of Medical Imaging], 2007.

[Doweck] Jack Doweck. Intel Smart Memory Access and the Energy-Efficient Performance of the Intel Core Microarchitectures. white paper.

[Egger 98] M.L. Egger & C. Morel. Execution times of five reconstruction algorithms in 3D positron emission tomography. Physics in Medicine and Biology, vol. 43, pages 703–712, March 1998.

[Exxim 07] Exxim. http ://www.exxim-cc.com, 2007.

[Fahey 02] F.H. Fahey. Data Acquisition in PET Imaging. Journal of Nuclear Medicine Technolog, vol. 30, pages 39–49, 2002.

[Fritzsche 05] P. Fritzsche, A. Ripoll, E. Luque, J.-J. Fernandez & I. Garcia. A performance prediction for iterative reconstruction techniques on tomography. In Parallel, Distributed and Network-Based Processing, 2005. PDP 2005. 13th Euromicro Conference on, pages 92–99, 9-11 Feb. 2005.

[Gelas 06] J. De Gelas. Intel Woodcrest, AMD's Opteron and Sun's UltraSparc T1 : Server CPU Shoot-out. http ://www.anandtech.com/IT/showdoc.aspx ?i=2772, 2006.

[Goddard 02] I. Goddard & M. Trepanier. High-Speed Cone-Beam Reconstruction : An Embedded Systems Approach. In Proc. SPIE Medical Imaging Conf., pages 483–491, February 2002.

[Gregor 02] J. Gregor, S. S. Gleason, M. J. Paulus & J. Cates. Fast Feldkamp reconstruction based on focus of attention and distributed computing. International Journal of Imaging Systems and Technology, vol. 12, no. 6, pages 229–234, 2002.

[Gregor 07] J. Gregor, T. Benson & G. Bosilca. Distributed multi-core implementation of SIRT with application to cone-beam micro-CT. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, 2007.

[Hamill 02] J. Hamill & T. Bruckbauer. Iterative reconstruction methods for high-throughput PET tomographs. Physics in Medicine and Biology, vol. 47, pages 2627–2636, August 2002.

[Hamill 03] J. Hamill, C. Michel & P. Kinahan. Fast PET EM reconstruction from linograms. IEEE Tr. Nucl. Sci., vol. 50, no. 5, pages 1630–1635, October 2003.

[He 06] Tao He, Jun Ni & Ge Wang. A Heterogeneous Windows Cluster System for Medical Image Reconstruction. In Computer and Computational Sciences, IMSCCS '06., volume 1, pages 410–415, 20-24 June 2006.

[Heigl 07] B. Heigl & M. Kowarschik. High-Speed Reconstruction for C-Arm Computed Tomography. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, pages 25–28, 2007.

[Herraiz 06] J.L. Herraiz, S. Espana, J.J. Vaquero, M. Desco & J.M. Udias. FIRST : Fast Iterative Reconstruction Software for (PET) tomography. Physics in Medicine and Biology, vol. 51, pages 4547–4565, September 2006.

[Hudson 94] H.M Hudson & R.S. Larkin. Accelerated Image Reconstruction Using Ordered Subsets of Projection Data. IEEE Tr. Med. Imaging, vol. 13, no. 4, pages 601–609, December 1994.

[Jacobson 00] M. Jacobson, R. Levkovitz, A. Ben-Tal, K. Thielemans, T. Spinks, D. Belluzzo, E. Pagani, V. Bettinardi, M. C. Gilardi, A. Zverovich &

G. Mitra. Enhanced 3D PET OSEM reconstruction using inter-update Metz filtering. Physics in Medicine and Biology, vol. 45, pages 2417–2439, August 2000.

[Jones 99] H. Jones, G. Mitra, D. Parkinson & T. Spinks. A parallel implementation of the maximum likelihood method in positron emission tomography image reconstruction. Comput. Stat. Data An., vol. 31, no. 4, pages 417–439, October 1999.

[Jones 03] J.P. Jones, W.F. Jones, F. Kehren, D.F. Newport, J.H. Reed, M.W. Lenox, K. Baker, L.G. Byars, C. Michel & M.E. Casey. SPMD cluster-based parallel 3D OSEM. Nuclear Science, IEEE Transactions on, vol. 50, no. 5, pages 1498–1502, Oct. 2003.

[Jones 04] M.D. Jones & R. Yao. Parallel programming for OSEM reconstruction with MPI, OpenMP, and hybrid MPI-OpenMP. In Nuclear Science Symposium Conference Record, 2004 IEEE, volume 5, pages 3036–3042, 16-22 Oct. 2004.

[Jones 05] J.P. Jones, W.F. Jones, J. Everman, V. Panin, C. Michel, F. Kehren, Jun Bao, J. Young & M.E. Casey. Impact of a high-performance communication network on cluster-based parallel iterative reconstruction. In Nuclear Science Symposium Conference Record, 2005 IEEE, volume 4, pages 2273–2277, 23-29 Oct. 2005.

[Jones 06] Judson P. Jones, Arman Rahmim, Merence Sibomana, Andrew Crabb, Ziad Burbar, Charles B. Cavanaugh, Christian Michel & Dean F. Wong. Data Processing Methods for a High Throughput Brain Imaging PET Research Center. In Nuclear Science Symposium Conference Record, 2006. IEEE, volume 4, pages 2224–2228, Oct. 2006.

[Kachelriess 07] Marc Kachelriess, Michael Knaup & Olivier Bockenbach. Hyperfast parallel-beam and cone-beam backprojection using the cell general purpose hardware. Medical Physics, vol. 34, no. 4, pages 1474–1486, April 2007.

[Kadrmas 05] D. J. Kadrmas. Rotate-and-slant projector for fast LOR-based fully-3D iterative PET reconstruction. In Proc. Intl. Mtg. on Fully 3D Image Recon. in Rad. and Nuc. Med, pages 411–415, 2005.

[Kalender 05] W.A. Kalender. CT : the unexpected evolution of an imaging modality. European Radiology Supplements, vol. 15, pages d21–d24, 2005.

[Keesing 06] D.B. Keesing, J.A. O'Sullivan, D.G. Politte & B.R. Whiting. Parallelization of a fully 3D CT iterative reconstruction. In Biomedical Imaging : Macro to Nano, 2006. 3rd IEEE International Symposium on, pages 1240–1243, 6-9 April 2006.

[Kinahan 89] P.E. Kinahan & J.G. Rogers. Analytic 3D image reconstruction using all detected events. IEEE Tr. Nucl. Sci., vol. 36, no. 1, pages 964–968, February 1989.

[Kinahan 04] P. E. Kinahan, M. Defrise & R. Clackdoyle. Emission tomography : the fundamentals of pet and spect, chapitre Analytic image reconstruction methods. Elsevier Academic Press, 2004.

[Knaup 06] M. Knaup, W. A. Kalender & M. Kachelriess. Statistical Cone-Beam CT Image Reconstruction using the Cell Broadband Engine. In IEEE Medical Imaging Conference Record, 2006.

[Knaup 07] M. Knaup & M. Kachelriess. Acceleration Techniques for 2D Parallel and 3D Perspective Forward and Backprojections. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, 2007.

[Kole 04] J.S. Kole, M.A. Viergever & F.J. Beekman. Evaluation of accelerated iterative X-ray CT image reconstruction using floating point graphics hardware. In Nuclear Science Symposium Conference Record, 2004 IEEE, volume 6, pages 3940–3944, 16-22 Oct. 2004.

[Kole 05] J S Kole & F J Beekman. Parallel statistical image reconstruction for cone-beam x-ray CT on a shared memory computation platform. Physics in Medicine and Biology, vol. 50, no. 6, pages 1265–1272, 2005.

[Kontaxakis 02] G. Kontaxakis. Iterative Image Reconstruction for Clinical PET Using Ordered Subsets Median Root Prior and web-based Interface. Mol. Imaging Biol., vol. 4, no. 3, pages 219–233, May 2002.

[Kudrolli 02] H. Kudrolli, W. Worstell & V. Zavarzin. SS3D-Fast fully 3D PET iterative reconstruction using stochastic sampling. Nuclear Science, IEEE Transactions on, vol. 49, no. 1, pages 124–130, Feb. 2002.

[Lange 84] K. Lange & R. Carson. EM reconstruction algorithms for emission and transmission tomography. J. Comput. Assist. Tomogr., vol. 8, no. 2, pages 306–316, April 1984.

[Laurent 98] C. Laurent, F. Peyrin, J.M. Chassery & M.Amiel. Parallel image reconstruction on MIMD computers for three dimensional cone beam tomography. Parallel Comput., vol. 24, no. 9-10, pages 1461–1479, September 1998.

[Lecomte 98] Jean-François Lecomte. Etude comparative de méthodes de reconstruction en imagerie à émission de positon 3D : Influence de la statistique d'émission, impléménantation et optimisation sur calculateur et optimisation sur calculateur parallèle. PhD thesis, Université de Caen, 1998.

[Leeser 05] Miriam Leeser, Srdjan Coric, Eric Miller, Haiqian Yu & Marc Trepanier. Parallel-beam backprojection : an FPGA implementation optimized for medical imaging. J. VLSI Signal Process. Syst., vol. 39, no. 3, pages 295–311, 2005.

[Lewitt 03] R.M. Lewitt & S. Matej. Overview of Methods for Image Reconstruction From Projections in Emission Computed Tomography. Proc. IEEE, vol. 91, pages 1588–1611, oct 2003.

[Li 04a] J. Li, C. Papachristou & R. Shekhar. A Reconfigurable SoC Architecture and Caching Scheme for 3D Medical Image Processing. In Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM'04), Napa, USA, April 2004.

[Li 04b] Jianchun Li, R. Shekhar & C. Papachristou. A "brick" caching scheme for 3D medical imaging. In Biomedical Imaging : Macro to Nano, 2004. IEEE International Symposium on, volume 1, pages 563–566, 15-18 April 2004.

[Mancini 00] S. Mancini. Architectures matérielles pour la synthèse d'image par lancer de rayon. PhD thesis, Ecole Nationale Supérieure des Télécommunications, January 2000.

[Mancini 04] S. Mancini & N. Eveno. An IIR based 2D adaptive and predictive cache for image processing. In Design of Circuits and Integrated Systems, page 85, Bordeaux, France, 2004.

[Mancini 06] S. Mancini & M. Desvignes. Ray Casting on a SOPC : Algorithm and Memory Hierarchy Trade-Off. In Sixth International Conference on Computer and Information Technology (CIT 2006), 20-22 September 2006, Seoul, Korea, page 180, 2006.

[Matej 04] S. Matej, J.A. Fessler & I.G. Kazantsev. Iterative tomographic image reconstruction using Fourier-based forward and back-projectors. Medical Imaging, IEEE Transactions on, vol. 23, no. 4, pages 401–412, April 2004.

[Mepham 04] D. Mepham. The next Pentium 4 processor, Prescott arrives. http ://www.hardwareanalysis.com/content/article/1686.8/, 2004.

[Mueller 00] K. Mueller & R. Yagel. Rapid 3-D cone-beam reconstruction with the simultaneous algebraic reconstruction technique (SART) using 2-D texture mapping hardware. Medical Imaging, IEEE Transactions on, vol. 19, no. 12, pages 1227–1237, Dec. 2000.

[Mueller 06] K. Mueller & Fang Xu. Practical considerations for GPU-accelerated CT. In Biomedical Imaging : Macro to Nano, 2006. 3rd IEEE International Symposium on, pages 1184–1187, 6-9 April 2006.

[Mueller 07] K. Mueller, F ; Xu & N. Neophytou. Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography ? In Proc. SPIE Vol. 6498 [Computational Imaging V], 2007.

[Munz 98] F. Munz. NOW Based Parallel Reconstruction of Functional Images. In Proc. Int. Parallel Processing Symp. and Symp. Parallel and Distributed Processing (IPPS/SPDP'98), pages 210–214, Orlando, USA, April 1998.

[Ni 06] J. Ni, T. He, X. Li & Ge Wang. Review of Parallel Computing Techniques for Computed Tomography Image Reconstruction. Current Medical Imaging Reviews, vol. 2, no. 4, pages pp. 405–414, 2006.

[Ortuno 06] Juan E. Ortuno, Jose L. Rubio, Pedro Guerra, George Kontaxakis & Andres Santos. Multi-grid 3D-OSEM reconstruction technique for high resolution rotating-head PET scanners. In Nuclear Science Symposium Conference Record, 2006. IEEE, volume 4, pages 2215–2218, Oct. 2006.

[Panin 06] V.Y. Panin, F. Kehren, H. Rothfuss, D. Hu, C. Michel & M.E. Casey. PET reconstruction with system matrix derived from point source measurements. Nuclear Science, IEEE Transactions on, vol. 53, no. 1, pages 152–159, Feb. 2006.

[Picard 98] Y. Picard, V. Selivanov, M. Verreault & R. Lecomte. Optimizing communications for parallel ML-EM image reconstruction on large clusters of processors. In Nuclear Science Symposium, 1998. Conference Record. 1998 IEEE, volume 3, pages 1574–1580, 8-14 Nov. 1998.

[Press 06] W.H. Press. Discrete Radon transform has an exact, fast inverse and generalizes to operations other than sums along lines. Proceedings of the National Academy of Sciences of the United States of America, vol. 103, no. 51, pages 19249–19254, 2006.

[Qi 98] J. Qi & R.M. Leahy et al. Fully 3D Bayesian image reconstruction for the ECAT EXACT HR+. IEEE Tr. Nucl. Sci., vol. 45, no. 3, pages 1096–1103, June 1998.

[Reilhac 05] A. Reilhac, G. Batan, C. Michel, C. Grova, J. Tohka, D.L. Collins, N. Costes & A.C. Evans. PET-SORTEO : validation and development of database of Simulated PET volumes. Nuclear Science, IEEE Transactions on, vol. 52, no. 5, pages 1321–1328, Oct. 2005.

[Riabkov 07] D. Riabkov, X. Xue, D. Tubbs & A. Cheryauka. Accelerated cone-beam backprojection using GPU-CPU hardware. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, pages 68–71, 2007.

[Rodet 02a] T. Rodet, P. Grangeat & L. Desbat. Multichannel algorithm for fast 3D reconstruction. Phys. Med. Biol., vol. 47, no. 15, pages 2659–2671, August 2002.

[Rodet 02b] Thomas Rodet. Algorithmes rapides de reconstruction en tomographie par compression des calculs. PhD thesis, TIMC-IMAG, Institut National Polytechnique de Grenoble, oct 2002.

[Sakamoto 05] M. Sakamoto, H.Nishiyama, H.Satoh, S.Shimizu, T. Sanuki, K.Kamijoh, A. Watanabe & A. Asahara. An Implementation of the Feldkamp Algorithm for Medical Imaging on Cell. Rapport technique, IBM, 2005.

[Sakamoto 07] Masaharu Sakamoto & Masana Murase. Parallel Implementation for 3-D CT Image Reconstruction on Cell Broadband Engine. In Multimedia and Expo, 2007 IEEE International Conference on, pages 276–279, 2-5 July 2007.

[Scheins 06] J.J. Scheins, F. Boschen & H. Herzog. Analytical calculation of volumes-of-intersection for iterative, fully 3-D PET reconstruction. Medical Imaging, IEEE Transactions on, vol. 25, no. 10, pages 1363–1369, Oct. 2006.

[Schellmann 07] M. Schellmann & S. Gorlatch. Comparison of Two Decomposition Strategies for Parallelizing the 3D List-Mode OSEM Algorithm. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, 2007.

[Scherl 07a] H. Scherl, S. Hoppe, F. Dennerlein, G. Lauritsch, W. Eckert, M. Kowarschik & J. Hornegger. On-the-fly-Reconstruction in Exact Cone-Beam CT using the Cell Broadband Engine Architecture. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, pages 29–32, 2007.

[Scherl 07b] Holger Scherl, Holger Scherl, Benjamin Keck, Markus Kowarschik & Joachim Hornegger. Fast GPU-Based CT Reconstruction using the Common Unified Device Architecture (CUDA). In Benjamin Keck, editeur, IEEE Nuclear Science Symposium Conference Record NSS '07, volume 6, pages 4464–4466, 2007.

[Schiwietz 06] T. Schiwietz, T.-c. Chang, P. Speier & R. Westermann. MR image reconstruction using the GPU. In Proceedings of the SPIE, volume 6142, pages 1279–1290, March 2006.

[Schiwietz 07] T. Schiwietz, S. Bose, J. Maltz & R. Westermann. A fast and high-quality cone beam reconstruction pipeline using the GPU. In Proc. SPIE Vol. 6510, 2007.

[Shattuck 02] D.W. Shattuck, J. Rapela, E. Asma, A.Chatziioannu, J.Qi & R.M. Leahy. Internet2-based 3D PET image reconstruction using a PC cluster. Phys. Med. Biol., vol. 47, no. 15, pages 2785–2795, August 2002.

[Shepp 82] L.A. Shepp & Y. Vardi. Maximum likelihood reconstruction for emission tomography. IEEE Tr. Med. Imaging, vol. 1, pages 113–122, October 1982.

[Siddon 85] R. L. Siddon. Fast calculation of the exact radiological path for a three-dimensional CT array. Med Phys, vol. 12, no. 2, pages 252–255, 1985.

[Smith 07] M.F. Smith, S. Majewski & R.R. Raylman. Fully 3D Iterative List-Mode PEM-PET Image Reconstruction on a Multiprocessor Computer. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, 2007.

[Terarecon] Terarecon. http ://www.terarecon.com/.

[Thielemans 06] Kris Thielemans, Sanida Mustafovic & Charalampos Tsoumpas. STIR : Software for Tomographic Image Reconstruction Release 2. In

Nuclear Science Symposium Conference Record, 2006. IEEE, volume 4, pages 2174–2176, Oct. 2006.

[Tita 07] R. Tita & T.C. Lueth. Online Iterative Reconstruction with the use of the Graphical Processing Unit (GPU). In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, 2007.

[Toft 96] Peter Toft. The Radon Transform : Theory and Implementation. PhD thesis, Technical University of Denmark, Copenhagen, 1996.

[Vaz 07] M.S. Vaz, M. McLin & A. Ricker. Current and Next Generation GPUs for Accelerating CT Reconstruction : Quality, Performance, and Tuning. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, 2007.

[Vollmar 00] S. Vollmar, K. Wienhard, M. Lercher, C. Knoss, C. Michel, J. Treffert, M. Schmand, M.E. Casey, D. Newport, P. Luk, J. Bao, R. Nutt & W.D. Heiss. BeeHive : cluster reconstruction of 3-D PET data in a Windows NT network using FORE. In Nuclear Science Symposium Conference Record, 2000 IEEE, volume 2, pages 15/213–15/215, 15-20 Oct. 2000.

[Vollmar 02] S. Vollmar, C. Michel, J.T. Treffert, D.F. Newport, M. Casey, C. Knoss, K.Wienhard, X. Liu, M.Defrise & W.-D. Heiss. HeinzelCluster accelerated reconstruction for FORE and OSEM3D. Phys. Med. Biol., vol. 47, no. 15, pages 2651–2658, August 2002.

[Xu 04] F. Xu & K. Mueller. Ultra Fast 3D filtered back projection on commodity graphics hardware. In Proc. IEEE Int. Symp. Biomedical Imaging (ISBI'04), pages 571–574, Arlington, USA, April 2004.

[Xu 05] Fang Xu & K. Mueller. Accelerating popular tomographic reconstruction algorithms on commodity PC graphics hardware. Nuclear Science, IEEE Transactions on, vol. 52, no. 3, pages 654–663, June 2005.

[Xu 07] Fang Xu & Klaus Mueller. Real-time 3D computed tomographic reconstruction using commodity graphics hardware. Physics in Medicine and Biology, vol. 52, no. 12, pages 3405–3419, 2007.

[Xue 06] X. Xue, A. Cheryauka & D. Tubbs. Acceleration of fluoro-CT reconstruction for a mobile C-arm on GPU and FPGA hardware : a simulation study. vol. 6142, pages 1494–1501, March 2006.

[Yang 07] H. Yang, M. Li, K. Koizumi & H. Kudo. Accelerating Backprojections via CUDA Architecture. In 9th International Meeting on Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine, pages 52–55, 2007.

[Yu 01] R. Yu, R. Ning & B. Chen. High-speed cone-beam reconstruction on PC. vol. 4322, pages 964–973, July 2001.

[Zeng 00] G.L. Zeng & G.T. Gullberg. Unmatched projector/backprojector pairs in an iterative reconstruction algorithm. Medical Imaging, IEEE Transactions on, vol. 19, no. 5, pages 548–555, May 2000.