UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Algoritmos Paralelos de Reconstrucción de Imágenes TAC sobre Arquitecturas Heterogéneas

TESIS DOCTORAL

Presentada por: Liubov Alexandrovna Flores

Dirigida por: Prof. Vicente Vidal Gimeno

Prof. Gumersindo Verdú Martín

Septiembre 2015

A mi familia-
la fuente de apoyo e inspiración

Agradecimientos

Al finalizar este trabajo, me doy cuenta de la magnitud de la participación de personas e instituciones que han facilitado las cosas para que este trabajo llegue a un término feliz. Por esto, es para mí un verdadero placer utilizar este espacio para ser justo expresando mis agradecimientos.

Quisiera expresar mi agradecimiento especial para mis tutores Prof. Vicent Vidal y Prof. Gumersindo Verdú. Su apoyo y confianza en mi trabajo y su capacidad de guiar mis ideas ha sido un aporte invaluable no solamente en el desarrollo de esta tesis, sino también en mi formación como investigadora. Les agradezco también el haberme facilitado siempre los medios suficientes para llevar a cabo todas las actividades propuestas durante el desarrollo de esta tesis. Muchas gracias.

Agradezco infinitamente a la Dra. Patricia Mayo y al Dr. Francisco Rodenas por su disponibilidad y paciencia en las discusiones que siempre redundaban beneficiosamente.

Quiero expresar mi gratitud a todo el personal de la Universidad Politécnica de Valencia por su apoyo a mi trabajo diario en el Departamento de Sistemas Informáticos y Computación (DSIC).

Mis sinceros agradecimientos a Estíbaliz Parcero. Su participación, aunque corta, ha enriquecido el trabajo realizado.

Quisiera destacar el constante apoyo de mi familia durante el desarrollo de este trabajo y expresarles mi enorme agradecimiento.

Abstract

In medicine, the diagnosis based on computed tomography (CT) imaging is fundamental for the detection of abnormal tissues by different attenuation values on X-ray energy, which frequently are not clearly distinguished for the radiologist. Different methods have been developed to reconstruct images. In this work we analyse and compare analytical and iterative methods to resolve the reconstruction problem.

Today, in practice, the reconstruction process is based on analytical methods and one of the most widely used algorithms is known as Filtered back projections (FBP) algorithm. This algorithm implements the inverse Radon Transform, which is a mathematical tool used in Biomedical Engineering for the reconstruction of CT images.

From the very beginning of the development of scanners, it was important to reduce the scanning time, to improve the quality of images and to reduce the reconstruction time of images. Today's technology provides powerful systems, multiprocessor and multicore processor systems, that provide the possibility to reduce the reconstruction time.

In this work, we analyze the FBP based on the inverse Radon Transform and its relation to the Fourier Transform, with the aim to achieve better performance while using resources of a system in an optimal way. This algorithm uses parallel projections, is simple, robust, and the results could be extended for a variety of situations.

In many applications, the set of projection data needed for the reconstruction, is incomplete due to the physical reasons. Consequently, it is possible to achieve only approximated reconstruction. In this conditions, the images reconstructed with analytical methods have a lot of artefacts in two and three dimensions.

Iterative methods are more suitable for the reconstruction from a limited number of projections in noisy conditions. Their usage may be important for the functionality of portable scanners in emergency situations. However, in practice, these methods are less used due to their high computational cost. In this work, the reduction of the execution time is achieved by performing the parallel implementation on multi-core and many-core systems of such iterative algorithms as SART, MLEM and LSQR.

The iterative methods have become a hot topic of interest because of their capacity to resolve the reconstruction problem from a limited number of projections. This allows the possibility to reduce the radiation dose during the data acquisition process. At the same time, in the reconstructed images appear undesired artefacts.

To resolve the problem effectively, we have adopted the LSQR method with soft threshold filtering technique and the fast iterative shrinkage-thresholding algorithm for computed tomography imaging and present the efficiency of the method named LSQR-STF-FISTA.

The reconstruction methods are analysed through the reconstructions from simulated and real projection data. Also, the quality of the reconstructed images is compared with the aim of drawing conclusions regarding the studied methods.

We conclude from this study that iterative methods are capable to reconstruct images from a limited number of dataset at a low computational cost.

Resumen

En medicina, el diagnóstico basado en imágenes de tomografía axial computerizada (TAC) es fundamental para la determinación de anormalidades a través de diferentes valores de atenuación de la energía de rayos-X, las cuales, frecuentemente, son difíciles de ser distinguidas por los radiólogos. Se han desarrollado diferentes técnicas de reconstrucción de imagen. En este trabajo analizamos y comparamos métodos analíticos e iterativos para resolver de forma eficiente el problema de reconstrucción.

Hoy, en la práctica, el proceso de reconstrucción de imagen se basa en algoritmos analíticos entre los cuales, el algoritmo de retroproyección filtrada 'filtered backprojection' (FBP) es el más conocido. Este algoritmo se usa para implementar la Transformada de Radon inversa que es una herramienta matemática cuya utilización principal en Ingeniería Biomédica es la reconstrucción de imágenes TAC.

Desde el comienzo del desarrollo de escáneres ha sido importante reducir el tiempo de escaneo, mejorar la calidad de imagen y reducir el tiempo de reconstrucción. La tecnología de hoy ofrece potentes sistemas con varios procesadores y núcleos que posibilitan reducir el tiempo invertido en la reconstrucción de imágenes.

En este trabajo se analiza el algoritmo FBP basado en la Transformada de Radon inversa y su relación con la Transformada de Fourier con el objetivo de optimizar su cálculo aprovechando al máximo los recursos del sistema. Este algoritmo se basa en proyecciones paralelas y se destaca por su simplicidad y robustez, y permite extender los resultados a una variedad de situaciones.

En muchas aplicaciones el conjunto de proyecciones necesarias para la reconstrucción puede ser incompleto por razones físicas. Entonces, la única posibilidad es realizar una reconstrucción aproximada. En estas condiciones, las imágenes reconstruidas por los algoritmos analíticos en dos o tres dimensiones son de baja calidad y con muchos artefactos.

Los métodos iterativos son más adecuados para la reconstrucción de imágenes cuando se dispone de un menor número de proyecciones en condiciones más ruidosas. Su uso puede ser importante para el funcionamiento en escáneres portátiles en condiciones de urgencia en cualquier lugar. Sin embargo, en la práctica, estos métodos son menos usados por su alto coste computacional. En este trabajo presen-

tamos el estudio y diversas implementaciones paralelas que permiten bajar el coste computacional de tales métodos iterativos como SART, MLEM y LSQR.

Los métodos iterativos se han convertido en un tópico de gran interés para muchos vendedores de sistemas de TAC clínicos por su capacidad de resolver el problema de reconstrucción con un número limitado de proyecciones. Esto proporciona la posibilidad de reducir la dosis radiactiva en los pacientes durante el proceso de adquisición de datos. Al mismo tiempo, en la reconstrucción aparecen artefactos no deseados.

Para resolver el problema en forma efectiva y eficiente, hemos adaptado el método LSQR con el método de filtrado 'Soft Threshold Filtering' y el algoritmo de aceleración 'Fast Iterative Shrinkage-thresholding Algorithm' para TAC. La eficiencia y fiabilidad del método nombrado LSQR-STF-FISTA se presenta en este trabajo.

Los métodos de reconstrucción de imágenes se analizan mediante la reconstrucción a partir de proyecciones simuladas y reales, comparando la calidad de imagen reconstruida con el objetivo de obtener conclusiones respecto a los métodos usados.

Basándose en este estudio, concluimos que los métodos iterativos son capaces de reconstruir imágenes con el conjunto limitado de proyecciones con un bajo coste computacional.

Resum

En medicina, el diagnòstic basat en imatges de tomografia axial compueritzada (TAC) és fonamental per a la determinació d'anormalitats a través de diferents valors d'atenuació de l'energia de rajos-X, les quals, freqüentment,són difícils de ser distingides pels radiòlegs. S'han desenvolupat diferents tècniques de reconstrucció d'imatge. En aquest treball analitzem i comparem mètodes analítics i iteratius per a resoldre el problema de reconstrucció.

Avui, en la pràctica, el procés de reconstrucció d'imatge es basa en algorismes analítics entre els quals, l'algorisme de retroproyección filtrada 'filtered backprojection' (FBP) és el més conegut. Aquest algorisme s'usa per a implementar la Transformada de Radon inversa que és una eina matemàtica la utilització principal de la qual en Enginyeria Biomèdica és la reconstrucció d'imatges TAC.

Des del començament del desenvolupament dels lectors òptics ha sigut important reduir el temps d'escanege, millorar la qualitat d'imatge i reduir el temps de reconstrucció. La tecnologia d'avui ofereix potents sistemes amb diversos processadors i nuclis que possibiliten reduir el temps invertit en la reconstrucció d'imatges.

En aquest treball s'analitza l'algorisme FBP basat en la Transformada de Radon inversa i la seua relació amb la Transformada de Fourier amb l'objectiu d'optimitzar el seu càlcul aprofitant al màxim els recursos del sistema. Aquest algorisme es basa en projeccions paral·leles i es destaca per la seua simplicitat i robustesa, i permet estendre els resultats a una varietat de situacions.

En moltes aplicacions el conjunt de projeccions necessàries per a la reconstrucció pot ser incomplet per raons físiques. Llavors, l'única possibilitat és realitzar una reconstrucció aproximada. En aquestes condicions, les imatges reconstruïdes pels algorismes analítics en dues o tres dimensions són de baixa qualitat i amb molts artefactes.

Els mètodes iteratius són més adequats per a la reconstrucció d'imatges quan es disposa d'un menor nombre de projeccions en condicions més sorolloses. El seu ús pot ser important per al funcionament en escáneres portàtils en condicions d'urgència en qualsevol lloc. No obstant açò, en la pràctica, aquests mètodes són menys usats pel seu alt cost computacional. En aquest treball presentem l'estudi i diverses implementacions paral·leles que permeten baixar el cost computacional de

tals mètodes iteratius com SART, MLEM i LSQR.

Els mètodes iteratius s'han convertit en un tòpic de gran interès per a molts venedors de sistemes de TAC clínics per la seua capacitat de resoldre el problema de reconstrucció amb un nombre limitat de projeccions. Açò proporciona la possibilitat de reduir la dosi radioactiva en els pacients durant el procés d'adquisició de dades. Al mateix temps, en la reconstrucció apareixen artefactes no desitjats.

Per a resoldre el problema en forma efectiva i eficient, hem adaptat el mètode LSQR amb el mètode de filtrat 'Soft Threshold Filtering' i l'algorisme d'acceleració 'Fast Iterative Shrinkage-thresholding Algorithm' per a TAC. L'eficiència i fiabilitat del mètode nomenat LSQR-STF-FISTA es presenta en aquest treball.

Els mètodes de reconstrucció d'imatges s'analitzen mitjançant la reconstrucció a partir de projeccions simulades i reals, comparant la qualitat d'imatge reconstruïda amb l'objectiu d'obtenir conclusions respecte als mètodes usats.

Basant-se en aquest estudi, concloem que els mètodes iteratius són capaços de reconstruir imatges amb el conjunt limitat de projeccions amb un baix cost computacional.

Índice general

Índice de figuras

Lista de símbolos

$\mathbf{A}$	matriz
A^T	matriz transpuesta
A_i	fila i de $\mathbf{A}$
A_i^+	suma de filas de $\mathbf{A}$
A_j^+	suma de columnas de $\mathbf{A}$
a_{ij}	elemento de $\mathbf{A}$
x	vector
x_j	elemento de x
p	vector
p_i	elemento de p
N	número de columnas de $\mathbf{A}$
M	número de filas de $\mathbf{A}$
$cols$	vector de columnas de $\mathbf{A}$
$rows$	vector de filas de $\mathbf{A}$
$values$	vector de valores de $\mathbf{A}$
NNZ	número de elementos no ceros en $\mathbf{A}$
NNZ_i	número de elementos no ceros en la fila i de $\mathbf{A}$
NNZ_j	número de elementos no ceros en la columna j de $\mathbf{A}$
I	matriz identidad
$\mu(x, y)$	Distribución de coeficiente lineal de atenuación
$p_\phi(r)$	Proyección para un ángulo ϕ a la distancia r
$p(\phi, r)$	Conjunto de proyecciones
J	Jacobiano de transformación de coordenadas rectangulares a polares
I_0	Intensidad de rayo no atenuada
$I_\phi(r)$	Intensidad de rayo para un ángulo ϕ a una distancia r
$\Re\{f(x, y)\}$	Transformada de Radon de $f(x, y)$
$\Re^{-1}\{p(r, \phi)\}$	Transformada Inversa de Radon de $p(r, \phi)$
$F(k_x, k_y)$	Transformada 2D de Fourier
$P(k, \phi)$	Transformada 1D de Fourier
$iterMax$	número máximo de iteraciones

Abreviaciones

ART	Algebraic Reconstruction Technique
AMD	Advances Micro Devices
BLAS	Basic Linear Algebra Subprogramms
CAP	Computación de Altas Prestaciones
CT	Computed Tomography
CG	Gradiente Congugado
CPU	Central Processing Unit
COO	Formato Coordenado Ordenado por Filas
CSR	Compact Sparse Row
CSC	Compact Sparse Column
CUDA	Compute Unified Device Architecture
DICOM	Digital Imaging and Communications in Medicine
FBP	Filtered Backprojection
FLOPS	FLoating-point Operations Per Second
FOV	Field of View
FISTA	Fast Iterative Shrinkage-Thresholding Algorithm
FT	Fourier Transform
GPUs	Graphical Processing Units
GB	GigaByte
kB	KiloByte
LSQR	Least Square QR method
MAE	Mean Absolute Error
MISD	Multiple Input Single Data
ML-EM	Maximum Likelihood Emission Tomography
ML-TR	Maximum Likelihood Transmission Tomography
MRI	Magnetic Resonance Imaging
MPP	Massively Parallel Processing
MPI	Massage Passing Interface
MSE	Mean Square Error
NUMA	Non-Uniform Memory Access

OpenMP	Open Multi-Processing
PET	Positron Emission Tomography
PETSc	Portable, Extensible Toolkit for Scientific Computation
PSNR	Peak Signal-to-Noise Ratio
SART	Simultaneous Algebraic Reconstruction Technique
SIRT	Simultaneous Iterative Reconstruction Technique
SISD	Single Input Single Data
SIMD	Single Input Multiple Data
SSIM	Structural Similarity Index Measure
STF	Soft Threshold Filtering
SMP	Simmetric Multiprocessing
SMR	System Matrix
SPECT	Single Photon Emission Tomography
TAC	Tomografía Axial Computarizada
TV	Total Variance
UMA	Uniform Memory Access
WTD	Weighted Total Diference

1 Introducción

1.1. Motivación

La tomografía axial computarizada (TAC) o tomografía de rayos-X es una técnica fundamental en el diagnóstico médico basado en imagen, que también tiene numerosas aplicaciones industriales. En TAC, las imágenes que corresponden a cortes interiores de un objeto, se obtienen a partir de las proyecciones tomadas por un escáner. Este procedimiento se denomina reconstrucción de imágenes tomográficas. Una descripción bastante detallada de la tomografía computerizada se puede encontrar en [2].

El problema de reconstrucción consiste en determinar la estructura interna del objeto basándose en datos experimentales del mismo objeto. Varios medios, incluido rayos-X, rayos gamma, electrones, protones, ondas de sonido y señales de resonancia magnética fueron usados para estudiar objetos cuyo tamaño varían desde moléculas complejas estudiadas con los microscopios electrónicos hasta objetos distantes estudiadas por radioastrónomos utilizando radioseñales. Numerosas aplicaciones donde el problema de recostrucción juega un papel principal están descritas por Stanley R. Deans en [1].

Los avances tecnológicos y teóricos han promovido un interés continuo al desarrollo de los diferentes métodos de reconstrucción y sus implementaciones. Desde el comienzo del desarrollo de escáneres ha sido importante reducir el tiempo de escaneo, disminuir en la medida de lo posible el número de rotaciones necesario para realizar una prueba TAC, mejorar la calidad de imagen y reducir el tiempo de reconstrucción.

Hoy en día, el desarrollo de arquitecturas paralelas, principalmente los procesadores multinúcleo y sistemas clusters, posibilita el desarrollo de nuevos algoritmos de reconstrucción que permiten explotar las características particulares de estas

plataformas. En la implementación de estos algoritmos, se puede plantear cómo optimizar su ejecución para reducir los costes computacionales empleando la arquitectura disponible.

La motivación de este trabajo es la investigación de los algoritmos de reconstrucción de imagen existentes, el diseño de nuevos algoritmos, sus eficientes implementaciones paralelas, y su adaptación a la reducción de vistas y como consecuencia a la reducción de dosis radiactiva en pacientes.

1.2. Objetivos

El concepto de Arquitectura de Sistema Heterogéneo se aplica a sistemas compuestos por diferentes tipos de PCs y máquinas con múltiples procesadores conectados mediante redes. Estos sistemas usan múltiples tipos de procesadores (normalmente CPUs y GPUs), en el mismo circuito integrado, para conseguir las mejores prestaciones de ambos. Por una parte, el procesamiento general de la GPU, que aparte de sus bien conocidas capacidades gráficas en 3D, también puede realizar cálculos matemáticos intensivos con conjuntos de datos muy grandes, y por otra, las CPUs que pueden realizar tareas tradicionales.

Este trabajo tiene como objetivo diseñar, implementar y evaluar algoritmos paralelos para resolver de forma eficiente el problema de reconstrucción de imágenes en TAC sobre arquitecturas actuales tales como procesadores multi-cores, sistemas clusters y GPUs. Este objetivo general puede refinarse en los siguientes objetivos específicos:

- Estudio de métodos analíticos de reconstrucción de imagen, en particular, el método basado en la utilización de la Transformada Inversa de Radon.
- Implementación paralela del algoritmo basado en la Transformada Inversa de Radon usando OpenMP y el algoritmo de la Transformada Rápida de Fourier.
- Estudio y análisis de métodos iterativos de reconstrucción de imagen.
- Estudio y análisis del método de Siddon de generación de la matriz del sistema de ecuaciones lineales.

- Implementación paralela de algoritmos iterativos de reconstrucción de imágenes sobre arquitecturas multi-core y GPUs.
- Utilización de librerías en la implementación de algoritmos iterativos de reconstrucción de imagen.
- Análisis de prestaciones y escalabilidad de los algoritmos.
- Comparación de la calidad de imágenes reconstruidas por métodos analíticos e iterativos.
- Estudio de la posibilidad de reconstrucción de imágenes médicas por menor número de proyecciones con el objetivo de reducir la dosis de radiación en pacientes.

1.3. Estado del arte

El problema de reconstrucción por proyecciones fue resuelto por primera vez por Johan Radon en 1917. En su trabajo [1] Radon desarrolla un método analítico de reconstrucción de la imagen de un objeto por sus proyecciones. Sucesivos investigadores en diversas áreas desconocían el trabajo de Radon y por eso existen muchos redescubrimientos de los resultados de Radon hasta los años 70.
Los avances tecnológicos y teóricos han promovido un interés continuo en los diferentes métodos de reconstrucción y sus implementaciones. Estos métodos se pueden clasificar en: métodos analíticos, métodos algebraicos y métodos estadísticos de reconstrucción.

Una de las áreas más amplias de aplicación del método de Radon es la Tomografía Axial Computarizada (TAC). En TAC, los rayos-X se usan para obtener un conjunto de datos necesarios para generar la imagen del interior de un objeto. El conjunto de proyecciones paralelas se conoce como la Transformada de Radon de la imagen, y la inversa de la transformada representa la misma imagen. Las Figuras 1.1 y 1.2 ilustran la idea de cómo se generan las proyecciones paralelas y proyecciones 'fanbeam'. En la Figura 1.1, rayos-X se emiten de varias fuentes y se registran por detectores después de atravesar un objeto generando proyecciones paralelas. En la

Figura 1.2 los rayos-X se emiten de una sola fuente y atraviesan un objeto en forma de un cono. Esta forma de proyecciones se conoce como proyecciones 'fanbeam'.

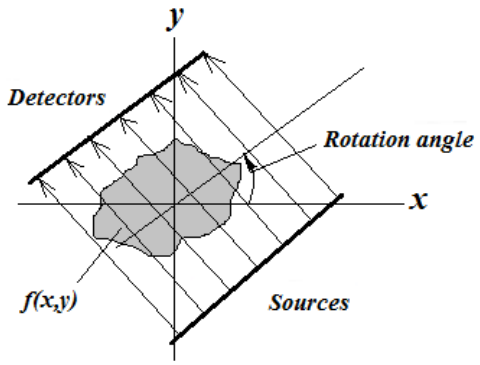

Figura 1.1: Proyecciones paralelas.

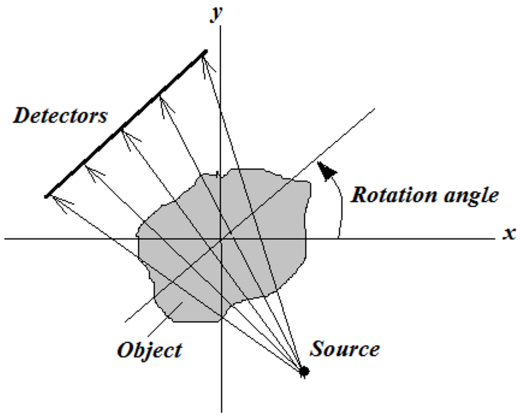

Figura 1.2: Proyecciones fanbeam.

Actualmente, la reconstrucción en TAC se basa en algoritmos analíticos entre los cuales el algoritmo de retroproyección filtrada FBP (Filtered Back Projections) es uno de los más conocidos, e.g. [3], [4]. El algoritmo es de bajo coste computacional, reconstruye imágenes de buena calidad, pero requiere una completitud de datos. En este trabajo se analiza el algoritmo que implementa la Transformada Inversa de Radon en la reconstrucción de imagen, representada por sus proyecciones paralelas, con el objetivo de determinar el grado de paralelización y escalabilidad del algoritmo para optimizarlo.

En muchas aplicaciones el conjunto de proyecciones necesarias para la reconstrucción puede ser incompleto por razones físicas. Entonces, la única posibilidad de reconstruir la imagen del objeto es realizar una reconstrucción aproximada. En estas condiciones, las imágenes reconstruidas por los algoritmos analíticos en dos o tres dimenciones son de baja calidad y con muchos artefactos [11]. Los métodos algebraicos y estadísticos resuelven el problema de reconstrucción en forma iterativa. En el enfoque algebraico (ART), la reconstrucción se reduce a la resolución de sistema de ecuaciones derivadas de la matriz de un sistema físico. Este sistema

de ecuaciones puede ser resuelto en forma iterativa. El número de variables del sistema de ecuaciones es igual al número de vóxeles en el volumen. El método ART fue aplicado al proceso de reconstrucción por Gordon, Bender and Herman y se conoce como el algoritmo de Kaczmarz [6]. Existen varias implementaciones de la técnica algebraica ART, como SIRT (simultaneous iterative reconstruction) y SART (simultaneous algebraic reconstruction) que están orientadas a una reconstrucción más precisa numéricamente y con una mejor calidad en las reconstrucciones [4], [5].

Existen muchas publicaciones sobre métodos iterativos estadísticos de reconstrucción. En estos métodos, primero se construye el modelo matemático (la función objetivo que incluye el ruido y modela el proceso de reconstrucción) y a partir de una aproximación inicial de la solución, en varios pasos de forma repetitiva, se llega a la imagen reconstruida. Este modelo se basa generalmente en el algoritmo 'expectation minimization' (ML-EM) que es un método iterativo para maximizar la probabilidad de ocurrir los parámetros en el modelo [7], [8], [10]. ML-EM está diseñado para la tomografía de emisión. ML-TR se usa para la tomografía de transmisión y trata los sinogramas como las realizaciones de una distribución de Poisson [8], [9].

Los algoritmos basados en métodos iterativos son capaces de proporcionar imágenes de mayor contraste y precisión en condiciones adversas y con un menor número de proyecciones [11]-[13]. En los exámenes de TAC es común escontrarse con proyecciones incompletas, en estos casos los métodos algebraicos son más adecuados para la reconstrucción de imágenes [14]-[16]. Los métodos iterativos son menos sensitivos al conjunto incompleto de datos. Adicionalmente, los artefactos de reconstrucción pueden ser reducidos o incluso, eliminados. Para casos prácticos, esto significa una mejor calidad en las imágenes reconstruidas. El mayor inconveniente de los métodos iterativos es su alto coste computacional. Por esta razón, en la práctica no son usados en forma amplia y siguen siendo un objeto de estudio. Actualmente, debido al crecimiento del poder computacional, los métodos iterativos son un foco de investigación en la reconstrucción de imágenes. Su uso puede ser importante en escáneres portátiles en condiciones de urgencia [17] y en estudios 3D.

Aunque se usa en medicina nuclear (gamma - camera, single photon emission computed tomography (SPECT), positron emission tomography (PET)), la reconstrucción iterativa no se difundió bastante en TAC. La principal razón de ello es que el conjunto de datos en TAC es mucho mayor que en medicina nuclear y la reconstrucción iterativa se hace muy costosa. La aceleración de la reconstrucción iterativa es una área de investigación activa [18]. Stone *et al.* describe el algoritmo acelerado en unidades gráficas de procesamiento (GPUs) para imágenes de resonancia magnética (MRI) [19]. Ellos reconstruyen imágenes de 128^3 voxels en, aproximadamente, un minuto. Johnson y Sofer en [20] proponen un método paralelo para las aplicaciones de tomografía de emisión (emission tomography) que es capaz de explotar la dispersidad y simetría de un modelo y demuestran que su esquema de paralelización es aplicable a la mayoría de algoritmos iterativos de reconstrucción. El tiempo requerido para la reconstrucción de imágenes de 128x128x23 voxels es de más de 3 minutos. Pratx *et al.* muestran los resultados de la reconstrucción en PET usando GPUs [21]. El tiempo requerido en una sola tarjeta GPU para la reconstrucción de una imagen de 160^3 es 8.8 segundos. La implementación multi GPU de las reconstrucciones tomográficas [22] acelera la reconstrucción de imágenes de 350x350x9 hasta 67 segundos en una tarjeta GPU y hasta 32 segundos con cuatro GPUs.

Al parecer, el tamaño de las imágenes reconstruidas y el tiempo de reconstrucción sigue siendo un problema. En uno de nuestros trabajos, hemos analizado la implementación paralela de la reconstrucción iterativa de imagen utilizando la librería PETSc [23]. En esta tesis hemos puesto como objetivo utilizar todo el poder computacional masivo de las GPUs para la reconstrucción de imágenes de mayor resolución sin perder la calidad. En los capítulos posteriores presentamos la descripción y validación de la implementación paralela de algoritmos iterativos basada en las arquitecturas actuales.

Un examen TAC implica la exposición del sujeto a una dosis de radiación que en exceso puede provocar efectos no deseados en la salud [27], [28]. El aumento de la radiación recibida en los exámenes médicos produce una preocupación [24]- [26]. En

aplicaciones prácticas, es deseable que las imágenes TAC de buena calidad se reconstruyan con un tiempo reducido de la expósición a radiación del paciente durante la adquisición de datos, es decir, es deseable la reducción del número de rotaciones nesesario para realizar una prueba TAC. En consecuencia, surge un problema de reconstrucción con un menor número de proyecciones. En los años recientes, han surgido algoritmos basados en la teoría de 'compress sensing' orientados a la resolución del problema de reconstrucción con un número limitado de proyecciones [30], [66]. Chen *et al.* [31] propone el método PICCS para la reconstrucción dinámica de imágenes TAC. Yu y Wang [32] analizan la tomografía computarizada en base de la teoría de 'compressed sensing'. En [33], Rudin *et al.* se propone el algoritmo basado en la variación total para eliminar el ruido en las imágenes. La reconstrucción con menor número de proyecciones es objeto de estudios en los trabajos de Herman y Davidi [34] y Wang y Jiang que proponen un subconjunto ordenado de datos para la reconstrucción [35].

En esta tesis, uno de los principales objetivos es utilizar todo el poder masivo computacional de las GPUs para la reconstrucción de imágenes de mayor resolución y sin perder la calidad. Con el objetivo de reducción de la dosis radiactiva en pacientes proponemos la resolución del problema de reconstrucción con menor número de vistas aplicando el método LSQR [36].

1.4. Estructura del documento

El resto del estudio desarrollado en esta tesis doctoral está organizado de la siguiente forma:

En el **Capítulo 2** se describen los conceptos principales de los sistemas de altas prestaciones actuales sobre los cuales están orientados los métodos desarrollados en el trabajo que se presenta. También se describen las herramientas de hardware utilizadas en los experimentos, las librerías y los entornos utilizados en la programación. Por último, se presentan las métricas empleadas para evaluar la calidad en las imágenes reconstruidas.

En el **Capítulo 3** se describen conceptos fundamentales de la tomografía axial computarizada.

El **Capítulo 4** se centra en los métodos analíticos de reconstrucción de imágenes. Se describen los componentes fundamentales del proceso de reconstrucción. Estos componentes son proyecciones que se toman mediante el proceso de escaneo. A continuación se decriben los aspectos matemáticos del método de reconstrución basado en la transformada inversa de Radon empleada en el proceso de la reconstrucción. Por último, se analiza la implementación paralela de este método.

El **Capítulo 5** se dedica a los métodos iterativos de reconstrucción de imágenes que representan el objetivo central de los estudios de este trabajo. Se describen las ventajas que presentan estos métodos y también la razón principal por la cual los métodos iterativos no son de uso comercial amplio actualmente. También se estudia los métodos iterativos tales como SART, MLEM y LSQR y se evalúan las implementaciones secuenciales y paralelas de estos algoritmos. El capítulo se concluye con el análisis y implementación de las formas optimizadas de estos algoritmos sobre las estructuras multicore.

El **Capítulo 6** se ha dedicado al análisis e implementación de los algoritmos iterativos en arquitecturas altamente paralelas - unidades de procesamiento gráfico (GPUs). Los métodos estudiados se aplican a la reconstrucción en 3D y se presentan los resultados obtenidos en la reconstrucción.

En el **Capítulo 7** se presenta el estudio de la reconstrucción de imágenes para el conjunto limitado de proyecciones. Se presenta el método LSQR combinado con la técnica de filtrado y aceleración que permite acelerar la convergencia del proceso de reconstrucción y al mismo tiempo preservar el contorno del objeto y mejorar la calidad de la imagen.

El estudio se concluye con el **Capítulo 8** donde se presentan las conclusiones del trabajo realizado y se plantean ideas para futuro desarrollo.

2 Herramientas de sistemas de computación de altas prestaciones

La estructura de hardware o arquitectura determina qué posibilidades hay en mejorar el rendimiento de un sistema comparado con el sistema con un procesador simple. Otro factor importante es la capacidad de generar un código eficiente, código que se va a ejecutar en una plataforma dada. En este capítulo describimos arquitecturas hardware y máquinas que pueden ser consideradas como herramientas CAP.

2.1. Clasificación de computadores de altas prestaciones

La clasificación de computadores de altas prestaciones dada por Flynn [37], se basa en la forma de manipular instrucciones y datos y se divide en cuatro clases de arquitecturas [38]:

- **Máquinas SISD.** Sistemas convencionales con una CPU que ejecutan instrucciones en forma secuencial. Actualmente, muchos servidores tienen más de una CPU y cada una puede ejecutar instrucciones que no están relacionadas. Estos sistemas actúan sobre diferentes conjuntos de datos.
- **Máquinas SIMD.** Estos sistemas a menudo tienen un número grande de unidades de procesamiento que ejecutan la misma instrucción sobre diferentes conjuntos de datos. De esta forma, una instrucción manipula un conjunto de datos en paralelo. Las máquinas con procesadores vectoriales se consideran como máquinas SIMD.
- **Máquinas MISD.** En la práctica no se ha construido este tipo de máquinas, aunque de forma teórica, en estas máquinas ejecutan diferentes instrucciones

sobre un único conjunto de datos.

- **Máquinas MIMD.** En estas máquinas se ejecutan varias instrucciones en paralelo sobre diferentes conjuntos de datos. La diferencia con las máquinas SIMD consiste en que las instrucciones y datos están relacionados y representan diferentes partes de la misma tarea. Sistemas MIMD pueden ejecutar múltiples subtareas en paralelo con el objetivo de disminuir el tiempo total de ejecución.

2.2. Sistemas con memoria compartida

Estos sistemas tienen múltiples CPU que comparten la misma memoria a la cual acceden de la misma forma. Estos sistemas pueden ser SIMD o MIMD. Un simple-CPU procesador vectorial puede considerarse como SIMD, y modelos multi-CPU de estas máquinas son ejemplos de MIMD. Los esquemas de sistemas con memoria compartida se presentan en las Figuras 2.1 y 2.2.

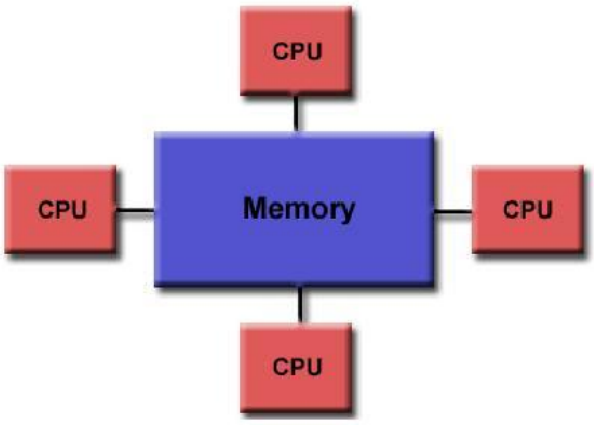

Figura 2.1: Memoria compartida UMA.

Los sistemas UMA (Uniforme Memory Access) tienen las siguientes caraterísticas:

- Tienen procesadores idénticos.
- Los tiempos de acceso a memoria son iguales para todos los procesadores.
- Sistemas de cache coherentes. Si un procesador modifica una variable en la memoria compartida, todos los procesadores saben de esto.

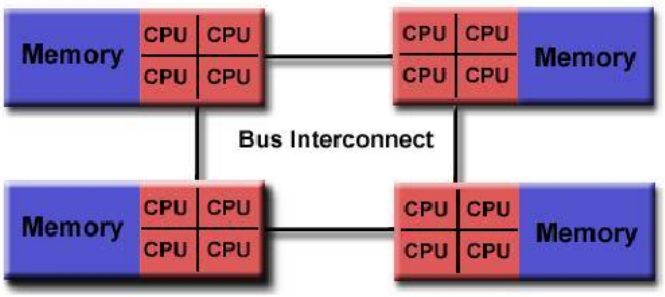

Figura 2.2: Memoria compartida NUMA.

Características de sistemas NUMA (Non-Uniform Memory Access)

- No todos los procesadores tienen tiempo de acceso a memorias iguales.
- Los accesos a través de interconexiones son más lentos.

Entre las ventajas de los sistemas con memoria compartida se puede subrayar:

- Rápido intercambio de datos debido a la proximidad de memoria a CPUs.
- Facilidad de programación.

Entre las desventajas de estos sistemas son:

- Falta de escalabilidad entre memoria y CPUs. El aumento de CPUs aumenta el trafico entre CPUs y memoria.
- Responsabilidad del programador por el acceso 'correcto' a la memoria global.

2.3. Sistemas con memoria distribuida

En los sistemas con memoria distribuida cada CPU tiene su propia memoria asociada. Las CPUs se conectan a través de una red y pueden intercambiar datos entre sus memorias. Estos sistemas también puede ser SIMD o MIMD. Los MIMD de memoria distribuida usan redes de interconexión de topologías muy variadas. Un sistema con memoria distribuida se presenta en la Figura 2.3.
Se destacan por las siguientes características:

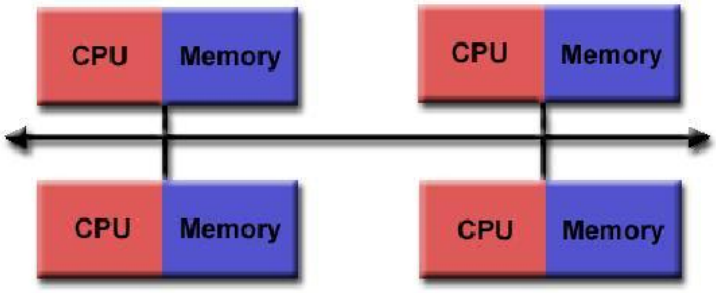

Figura 2.3: Memoria distribuida.

- Necesitan una red de comunicación para conectar la memoria entre los procesadores.
- Los procesadores tienen su memoria local. No hay memoria global. Cambios en la memoria local no tienen efecto en la memoria de otros procesadores.
- El programador define de forma explícita cómo y cuándo un procesador accede a la memoria de otro.

Las ventajas principales de estos sistemas :

- La memoria es escalable con el número de procesadores.
- Cada procesador accede a su memoria rápidamente.
- Pero la programación es más dificil.

2.4. Clusters de sistemas

Los sistemas clusters representan una colección de PCs/Estaciones de trabajo (workstations) que están conectadas con una red local. Representan una opción atractiva por el bajo coste de hardware y software, y por la posibilidad de tener control sobre el sistema. Hoy existen variedad de redes de comunicación en clusters que se distinguen por sus características como latencia, ancho de banda y coste.

2.5. Procesadores multi-núcleos

Los procesadores multi-núcleos (**multi-core processor**) son procesadores que contienen dentro de su circuito integrado a varios núcleos o unidades de procesamiento de instrucciones. Un procesador multi-núcleo puede repartir los procesos entre sus varios núcleos para su posterior ejecución. Los cores están integrados en un chip llamado multiprocesador.
Un procesador 'dual-core' tiene dos núcleos (e.g. AMD Phenom II X2, Intel Core Duo), un procesador 'quad-core' tiene cuatro núcleos (e.g. AMD Phenom II X4, la línea core Intel 2010, i3, i5, and i7), y un procesador 'hexa-core' tiene seis núcleos (e.g. AMD Phenom II X6, Intel Core i7 Extreme Edition 980X).

Un **'many-core' procesador** es un procesador que tiene un número de núcleos bastante grande de tal forma que las técnicas tradicionales de multiprocesamiento no son efectivas debido a la congestión que se genera a la hora de proporcionar las instrucciones y datos a los procesadores. El número límite de núcleos que establece el paso de un 'multi-core' a un 'many-core' procesador se mide en docenas de núcleos. Pasado este límite, la tecnología '*network on chip*' (NOC) resulta ser la más ventajosa.
El objetivo de NOC es diseñar el sistema de comunicación entre núcleos. Las topologías comunes para interconectar núcleos incluyen bus, anillo, grid bidimensionales, crossbar. La teoría y métodos de redes de interconección aplicadas a la comunicación de chips llevan a una mejora notable en la escalabilidad y la eficiencia de sistemas de diseño complejos.
Las aplicaciones que sacan más provecho de los procesadores multi-núcleo son aquellas que pueden generar muchos hilos de ejecución (threads) como las aplicaciones de audio/vídeo, cálculo científico, juegos, tratamiento de gráficos. Sólo cuando se ejecuta una sola aplicación que no sea paralelizable (no se pueda descomponer en hilos), es cuando no se aprovecha el potencial de procesamiento que permiten estos procesadores.

El primer procesador multi-núcleo que apareció en el mercado fue el IBM Power 4 en el año 2000. Actualmente, Intel y AMD ofrecen procesadores de dos (Core 2Duo),

cuatro (Intel Core i7) y hasta 10 núcleos (Familia de procesadores Intel Xeon E7) [39], [40] que posibilitan la ampliación de rendimiento para las aplicaciones fundamentales y más exigentes con los datos. Intel ha desarrollado un 80-core procesador que es capaz de transferir un terabyte de datos por segundo. El Duo chip transfiere solo 1.66 gigabytes de datos por segundo. El 80-core chip va a representar un incremento de rendimiento enorme sobre los procesadores existentes.
Las arquitecturas 'many-core' proporcionan un poder de procesamiento de datos enorme en la forma de paralelismo masivo SIMD. El número de núcleos en un chip crece rápido, y como resultado, se puede dar una nueva interpretación a la ley de Moore: 'el número de núcleos se duplica cada 18 meses'.

El esquema principal de procesador AMD Athlon se presenta en la Figura 2.4.

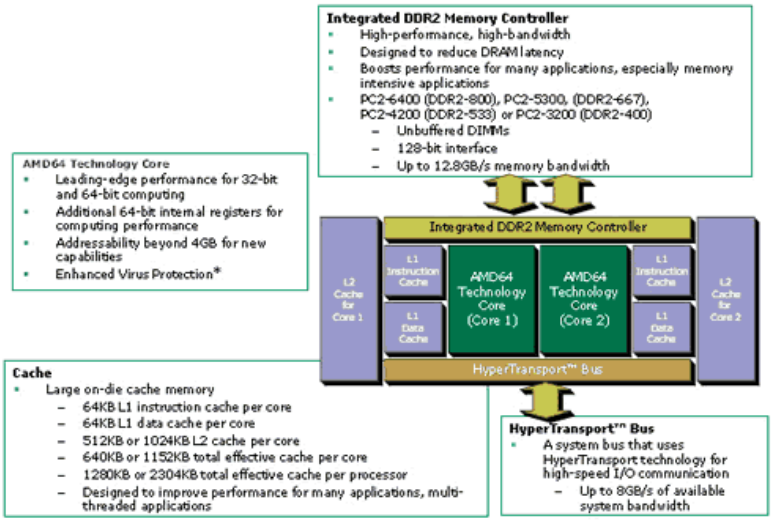

Figura 2.4: Arquitectura del procesador AMD Athlon.

2.6. Sistemas Multiprocesadores

Sistemas Multiprocesadores Simétricos (SMP). El término SMP, sistema multiprocesador simétrico, se refiere a la arquitectura hardware del sistema multiprocesador y al comportamiento del sistema operativo que utiliza dicha arquitectura. Un SMP es un computador con las siguientes características:

- Tiene dos o más procesadores similares de capacidades comparables.
- Los procesadores comparten la memoria principal y la Entrada/Salida, y están interconectados mediante un bus u otro tipo de sistema de interconexión, de manera que el tiempo de acceso a memoria es aproximadamente el mismo para todos los procesadores.
- Todos los procesadores pueden desempeñar las mismas funciones (de ahí el término simétrico).
- El sistema está controlado por un sistema operativo que posibilita la interacción entre los procesadores y sus programas.

El diagrama de bloques de un sistema multiprocesador se presenta en la Figura 2.5.

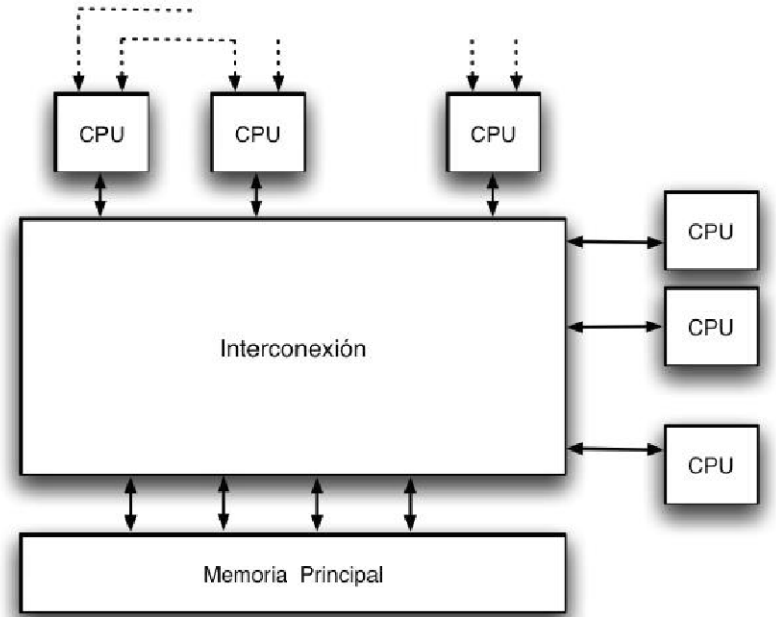

Figura 2.5: Diagrama de bloques de un sistema multiprocesador.

En SMP todos los procesadores pueden realizar las mismas funciones, un fallo en un procesador no hará que el computador se detenga. Se pueden aumentar las prestaciones del sistema añadiendo más procesadores.

Arquitectura MPP. Representa una máquina paralela que consta de varias unidades de procesamiento básicamente independientes. En efecto, cada una de estas unidades, conocida como 'nodo', es prácticamente un ordenador en sí misma, contando con su propio procesador, memoria no compartida, y que se comunica con las demás unidades de procesamiento a través de un canal provisto exclusivamente para este propósito. Este tipo de máquinas se conocen como computadores masivamente paralelos o máquinas MPP (Massively Parallel Processing, procesamiento masivamente paralelo).

Para que esta organización redunde en un mayor desempeño, se requiere colaboración entre los nodos. Como se mencionó, una máquina MPP debe contar con un canal que permita a los nodos comunicarse entre sí, a fin de intercambiar datos y coordinar sus operaciones. Ya que el objetivo principal de una máquina MPP es obtener un alto rendimiento, se busca que este canal de comunicaciones sea lo más eficiente posible, en términos tanto de ancho de banda como de tiempo de latencia. Sin embargo, el tener varias secciones de memoria independientes complica la programación en este tipo de arquitecturas.

2.7. GPU: altamente paralelo, multihilo, procesador multicore

La unidad de procesamiento gráfico (GPU), inventada por NVIDIA en 1999, es el procesador paralelo más potente hoy día [41]. Para satisfacer la demanda en los gráficos 3D de alta resolución en tiempo real, la GPU se ha desarrollado como un procesador multicore con multihilo y de alto paralelismo. Es un procesador de tremenda potencia para los cálculos con simple y doble precisión. La GPU es un procesador con una gran ancho de banda de memoria como se ilustra en las Figuras 2.6 y 2.7.

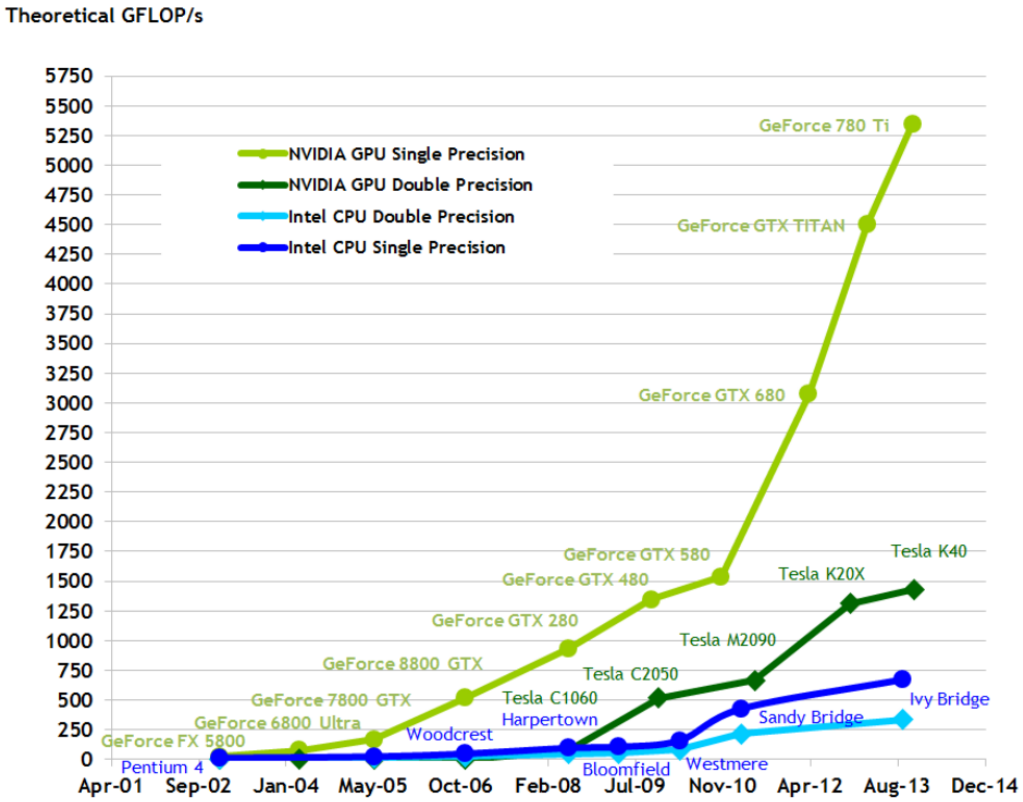

Figura 2.6: Operaciones por segundo para CPU y GPU [41].

La razón de la discrepancia entre CPU y GPU en la capacidad de cálculo con simple precisión es que la GPU está diseñada especialmente para los cálculos intensivos, de tal forma que hay más transistores dedicados al procesamiento de datos, como se ilustra en la Figura 2.8.

Los sistemas con GPU son especialmente adecuados para las aplicaciones donde el mismo programa se ejecuta sobre varios conjuntos de datos en paralelo. En muchas aplicaciones donde se procesan enormes conjuntos de datos, puede usarse el modelo de programacion paralelo de datos (data-parallel programming model) para acelerar cálculos. En efecto, muchos algoritmos, desde algoritmos de procesamiento de señal o algoritmos de simulaciones físicas y hasta los algoritmos en biología computacional, se aceleran por el procesamiento paralelo de datos.

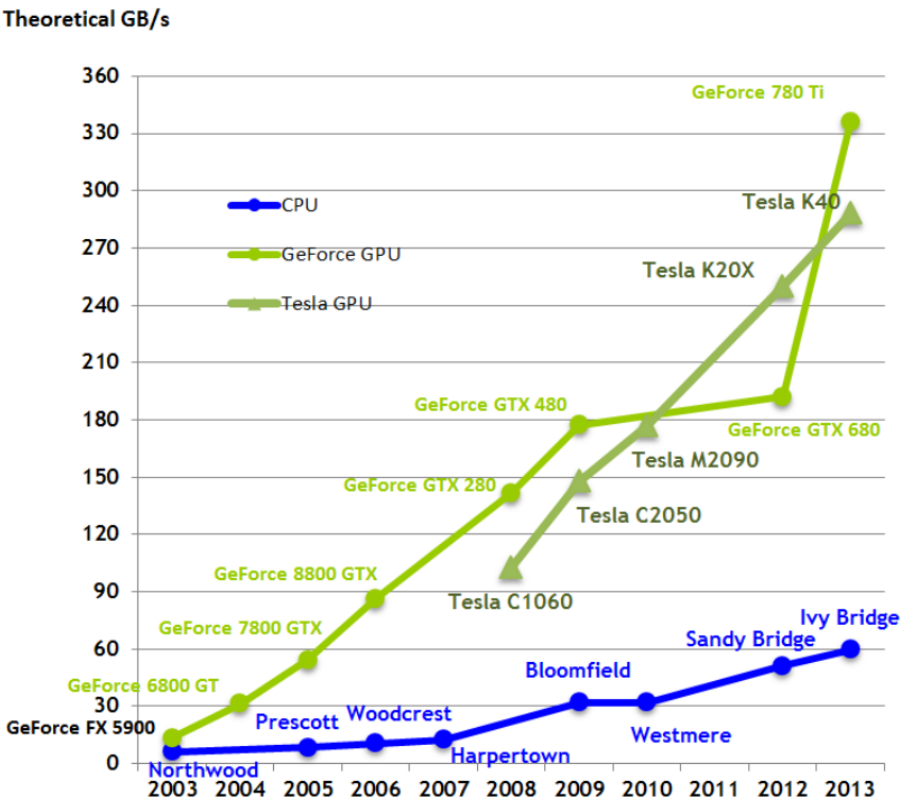

Figura 2.7: Ancho de banda de memoria para CPU y GPU [41].

NVIDIA introdujo dos tecnologías principales - la arquitectura G80 [43] (primero fue introducida en GPUs GeForce 8800®, Quadro FX 5600® y Tesla C870®) y CUDA [42], una arquitectura software y hardware que permite programar GPUs con los lenguajes de programación de alto nivel. El programador ahora puede escribir programas en C con la extensión CUDA dirigidos a explotar el procesador en forma paralela y masiva. Esta forma de programación se denomina Computación GPU (GPU Computing).

El modelo de programación CUDA explota de forma efectiva las capacidades paralelas de los GPUs. CUDA extiende *C* permitiendo al programador definir funciones *C*, llamados kernels, que se ejecutan N veces en paralelo por N diferentes CUDA threads. GeForce 8800 era el primer producto que inició el modelo de Computación GPU. Introducido en 2006, permitió a los programadores usar toda la potencia de

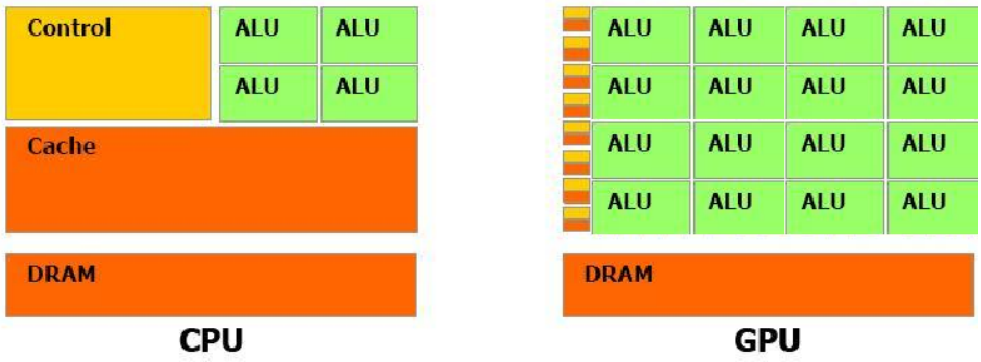

Figura 2.8: La GPU incluye más transistores para procesamiento de datos.

las GPUs sin necesidad de aprender un nuevo idioma de programación. La arquitectura **Fermi** [43] es la más significante desde entonces. Las características principales mejoradas en Fermi son:

- rendimiento mejorado con doble precisión - muchas aplicaciones requieren rendimiento alto con doble precisión
- verdadera jerarquía Cache - algunos algoritmos eran incapaces de usar la memoria compartida
- aumento de memoria compartida para acelerar la ejecución de las aplicaciones
- más rápidas operaciones atómicas

2.8. Herramientas hardware

En este apartado se describen brevemente los sistemas utilizados en los experimentos analizados en este trabajo. Se trata de dos sistemas ubicados en el departamento de Sistemas Informáticos y Computación de la Universidad Politécnica de Valencia.
El primero es Gpu.dsic.upv.es. El sistema se caracteriza por el procesador de 2.6GHz y dos unidades de procesamiento gráfico. Las tarjetas GPU tienen las siguientes propiedades:

- Son tarjetas TESLA K20c no integradas

- Pico de rendimiento de operaciones en coma flotante de precisión simple: 3.52 TFlops
- Pico de rendimiento de operaciones en coma flotante de doble precisió: 1.17 TFlops
- Memoria Global: 5GB
- Memoria Constante: 64 kB
- Número de Multiporocesadores: 13 con 192 núcleos / MP
- Número total de núcleos: 2496
- Memoria Compartida por multiprocesador: 49 kB

El segundo sistema utilizado es KAHAN. KAHAN es un cluster de computación de tamaño pequeño [44]. Se compone de:

- Un frontend con un procesador Intel Core 2 Duo a 3GHz, con 4 GB de memoria.
- Seis nodos biprocesador conectados mediante una red Infiniband.
- Cada nodo consta de:
  - 2 procesadores AMD Opteron 16 Core 6272, 2.1GHz, 16MB
  - 32GB de memoria DDR3 1600
  - Disco 500GB, SATA 6 GB/s
  - Controladora InfiniBand QDR 4X (40Gbps, tasa efectiva de 32Gbps)
- Los nodos están interconectados mediante una Infiniband.
- En total: 12 procesadores, 192 núcleos, 192 GB.

El pico teórico del rendimiento del sistema KAHAN es de 2304 GFlops (10^9 float point operations per second) y se calcula por la fórmula:

Performance in GFlops = (number of CPU cores) x (CPU instruction per cycle) x (CPU speed in GHz) = 192 x 4 x 3 = 2304 GFlops.

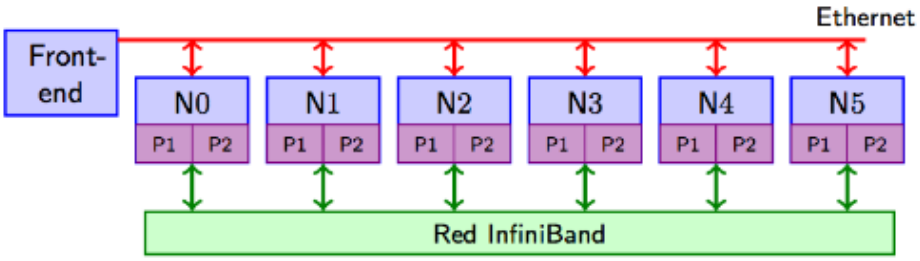

Figura 2.9: El Cluster Kahan.dsic.upv.es.

2.9. Herramientas software

En este apartado se describen los modelos de programación paralela y herramientas software utilizadas en los experimentos.

2.9.1. Modelos de programación paralela

- **Modelo de Memoria Compartida.** En este modelo las tareas comparten espacio de direcciones, que ellos leen y escriben asíncronamente. Diferentes mecanismos, como '*locks/semaphors*' pueden usarse para controlar el acceso a la memoria compartida. El desarrollo de programas es simplificado. Sin embargo, la mayor desventaja consiste en la posibilidad de crear tráfico cuando varios procesadores usan el mismo dato.

- **Modelo de Hilos .** En este modelo un proceso puede tener varios pasos de ejecución. El programa principal carga todo lo necesario para la ejecución, ejecuta en forma secuencial algunas instrucciones, y después crea tareas (hilos) que se ejecutan simultáneamente. Cada hilo tiene datos locales y puede comunicarse con otros a través de memoria global. Esto requiere instrucciones de sincronización para asegurar el acceso correcto a la memoria. Los threads pueden ser cancelados y creados de nuevo, sólo el programa (thread) principal se queda presente hasta que la aplicación se completa.

  Generalmente, este modelo de programación está asociado con arquitecturas de memoria compartida. Desde el punto de vista de programación, la implementación consta de un conjunto de directivas insertadas en el código secuencial o paralelo y el programador es responsable de determinar todo el

paralelismo. Un estándar de implementación de hilos es OpenMP.

OpenMP.

La librería OpenMP [45] permite añadir paralleliso al código secuencial en sistemas con memoria compartida. En forma simple se crean hilos y se controlan concurrencias de estos basándose en directivas del compilador. Es una librería portable. Existe implementaciones en C/C++ y Fortran.
Otra característica interesante de la librería es la posibilidad de explotar el paralelismo a nivel vectorial en un bucle. La vectorización significa el prosesamiento de elementos múltiples de un array al mismo tiempo. Para la realización de este prosesamiento, en los procesadores se puede utilizar unidades vectoriales con instrucciones SIMD (Single Instruction Multiple Data).

Algunas directivas proporcionadas por OpenMP4.0 que en forma explicita soportan construcciones de la programación vectorial son las siguientes:

- *omp simd* marca un bucle a vectorizar.
- *omp declare simd* declara un función que puede ser llamada desde bucle vectorizado.
- *omp parallel for simd* marca un bucle que se divide entre hilos y se vectoriza.

- **Modelo de Paso de Mensajes.** El modelo de paso de mensajes tiene las siguientes características:
  - Conjunto de tareas que puede residir en la misma unidad física o en varias máquinas, usa su propia memoria local para cálculos.
  - Las tareas intercambian datos a través de comunicación enviando y recibiendo mensajes.
  - Desde el punto de vista de programación, las implementaciones del modelo de paso de mensajes consisten en una librería de subrutinas insertadas en el código. El programador es responsable de determinar todo el paralelismo.

- **Librería MPI** es el estándar de la implementación de modelo de programación de Paso de Mensajes en la actualidad [46].

2.9.2. Librería PETSc

PETSc (Portable Extensive Toolkit for Scientific computation) es un conjunto de herramientas para la solución paralela numérica de sistemas de ecuaciones lineales dispersos utilizado en sistemas de alto rendimiento [47]. PETSc consta de un conjunto de librerías (similares a clases en C++). Cada librería manipula una familia particular de objetos (por ejemplo, matrices) y operaciones sobre estos objetos.
Algunos de los módulos de PETSc incluyen:

- vectores paralelos
- matrices dispersas paralelas con varios formatos de almacenamiento
- métodos de subespacios de Krylov
- precondicionadores
- los métodos de resolución de sistemas lineales y no lineales
- soporte para las tarjetas NVIDIA GPU.

Cada módulo representa una interfaz abstracta (una secuencia de llamadas) e implementaciones usando una estructura particular de datos. De esta forma PETSc ofrece códigos efectivos para diferentes fases de resolución de sistemas de ecuaciones y genera un ambiente agradable para la modelación de aplicaciones científicas y el rápido diseño de algoritmos. Usando la librería, el usuario puede incorporar 'solvers' y estructuras de datos personalizados.
Todos los programas de PETSc usan **MPI (Message Passing Interface)** estándar para las comunicaciones. Las librerías posibilitan la personalización y extensión de algoritmos y sus implementaciones. PETSc posibilita el uso de paquetes externos como Matlab y otros, se puede emplear desde Fortran, C, C++ y en la mayoría de sistemas basados en UNIX.

Soporte para las tarjetas NVIDIA GPU. Los métodos algebraicos de PETSc de resolución de sistemas de ecuaciones se puede usar en los sistemas NVIDIA GPU. Se ha introducido una nueva subclase de vectores con sus correspondientes métodos de manipulación. Adicionalmente, una subclase de matrices dispersas ejecuta las operaciones de producto matriz-vector en GPUs, y los métodos paralelos de resolución de sistemas de ecuaciones lineales que utilizan GPUs funcionan para todas las operaciones de producto entre vectores y matrices.
El uso de GPUs proporciona una técnica de resolución alternativa de alto rendimiento y de bajo coste computacional.
La infraestructura de PETSc crea la base para las aplicaciones de gran escala.

2.9.3. Librerías CUBLAS y CUSPARSE

La utilización de GPU, con el enorme potencial de cómputo paralelo, puede elevar considerablemente la eficiencia del algoritmo. El modelo de programación CUDA [48] permite resolver muchos problemas complejos computacionalmente de un modo más eficiente que en una CPU. Las librerías CUBLAS [49] y CUSPARSE [50] proporcionan al usuario el acceso a los recursos computacionales de unidades de procesamiento gráfico de NVIDIA (GPUs). La librería CUBLAS es la implementación de la librería BLAS (Basic Linear Algebra Subprograms) para GPUs. Para usar la librería, la aplicación tiene que colocar las matrices y vectores necesarios en en la memoria de GPU, rellenarlos con datos , hacer llamadas a la secuencia de las funciones CUBLAS deseadas, y después transferir los resultados de la memoría GPU a host. La librería CUBLAS proporciona funciones de ayuda para la realización de transferencia de datos entre GPU y host. La librería CUSPARSE contiene un conjunto de subrutinas básicas de álgebra lineal usadas para las operaciones con matrices dispersas y está diseñada para ser empleada desde C o C++. Estos subrutinas incluyen operaciones entre vectores y matrices en formato disperso y denso, así como las rutinas de conversión de diferentes formatos para matrices.

2.9.4. Librería BLAS

BLAS (Basic Linear Algebra Subprograms). Colección de rutinas para realizar operaciones básicas a bloques sobre matrices densas y vectores [51]. El Nivel 1

BLAS realiza operaciones vectoriales, el Nivel 2 Blas realiza las operaciones entre matrices y vectores, y el Nivel 3 Blas realiza operaciones entre matrices.
El BLAS es portable y eficiente, y por esto es usado en el desarrollo de software de algebra lineal de más alto nivel, como LAPACK por ejemplo.

2.9.5. Librería FFTW

FFTW es la librería de subrutinas para calculos de la Transformada de Fourier discreta (DFT) en una o más dimensiones para datos reales y complejos [52]. La librería también proporciona las rutinas para calcular la transformadas discretas de seno y coseno.
La última versión 3.3.4 de FFTW proprociona la rutina de paralelización de código para las plataformas con hilos OpenMP y también la versión MPI para la memoria distribuida.

2.10. Métricas de evaluación de calidad de imágenes

Para la evaluación de calidad de las imágenes reconstruidas se han utilizado las siguientes métricas:

- **Error Cuadrático Medio** (MSE).
  MSE mide la diferencia entre dos imágenes a comparar por la siguiente formula:

$$MSE = \frac{1}{nxm} \sum_{i=1}^{m} \sum_{j=1}^{n} \left[I_1(i,j) - I_2(i,j) \right]^2 , \tag{2.1}$$

  donde m y n denotan el tamaño de la imagen en píxeles.

- **Peak Signal-to-Noise Ratio** ($PSNR$).
  Se utiliza para definir la relación entre la máxima energía posible de una señal y el ruido que afecta la señal. El uso más habitual del PSNR es como medida cuantitativa de la calidad de la reconstrucción de imágenes. El PSNR se define como :

$$PSNR = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right) , \tag{2.2}$$

donde MAX_I es el valor máximo posible de píxel en la imagen. Cuando éstos se representan usando B bits por muestra, $MAX_I = 2^B - 1$.

- **Error Absoluto Medio** (MAE):

$$MAE = \frac{1}{nxm} \sum_{i=1}^{m} \sum_{j=1}^{n} |[I_1(i,j) - I_2(i,j)|. \tag{2.3}$$

- **Índice de Similitud Estructural** ($SSIM$).

  Es un método para medir la similitud entre dos imágenes ([73]). El SSIM considera la degradación en la imagen como un cambio en su información estructural. La métrica SSIM se calcula en varias ventanas x y y de tamaño NxN de la imagen por la siguiente formula:

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}, \tag{2.4}$$

  donde x y y son ventanas de I_1 y I_2 que van a ser comparadas; μ_x, μ_y denotan valores promedios de x y y; σ_x^2, σ_y^2 denotan varianzas de x y y; σ_{xy} representa la covarianza de x y y; c_1=$(k_1L)^2$, c_2=$(k_2L)^2$ denotan dos variables para estabilizar el denominador; L - el el rango de valores de píxeles, $k_1 = 0.01$, $k_2 = 0.03$.

  Esta métrica se calcula en varias ventanas de la imagen. El resultado de SSIM es un valor decimal entre -1 y 1; el valor 1 es alcanzable en el caso de dos conjuntos de datos idénticos.

2.11. Métricas de evaluación de algoritmos

Para describir las medidas de evaluación de los algoritmos paralelos desarrollados en este trabajo introducimos algunas definiciones.

Un sistema paralelo se considera como un conjunto de una arquitectura paralela y el algoritmo paralelo ejecutado en esta arquitectura.
El tiempo de ejecución de un algoritmo en un sólo procesador se llama el tiempo secuencial y lo denotaremos por T_1. T_1 representa el número de operaciones ejecutadas por un sólo procesador. El tiempo de ejecución del algoritmo paralelo en p

procesadores idénticos se llama el tiempo de ejecución paralelo, se denota por T_p y representa el número de operaciones ejecutadas en forma paralela por cada procesador. El tiempo de ejecución paralelo incluye un 'overhead' debido al coste de creación/destrucción/conmutación de procesos/hilos, coste de comunicación entre ellos, las esperas, bloqueos, etc. El 'overhead' total se denota por T_0.

Se puede deducir que:

$$pT_p = T_1 + T_0 \Rightarrow T_p = \frac{T_1 + T_0}{p} \tag{2.5}$$

El **SpeedUp** (S) del sistema es la relación entre el tiempo de ejecución en un sólo procesador y el tiempo de ejecución en procesadores múltiples y se define como

$$S = \frac{T_1}{T_p} = \frac{pT_1}{T_1 + T_0}. \tag{2.6}$$

La **Eficiencia** (E) del sistema paralelo expresa el grado de utilización del sistema multiprocesador y se define como

$$E = \frac{S}{p} = \frac{T_1}{T_1 + T_0} = \frac{1}{1 + \frac{T_0}{T_1}}. \tag{2.7}$$

El SpeedUp y la Eficiencia límite caracterizan el comportamiento del sistema al hacer crecer el tamaño del problema (W) hasta infinito y se expresan por las ecuaciones siguientes:

$$S_{limit} = \lim_{W \to \text{ínf}} \frac{T_1}{T_p} = p. \tag{2.8}$$

$$E_{limit} = \lim_{N \to \text{ínf}} \frac{S}{p} = 1. \tag{2.9}$$

El tamaño del problema (W) se expresa en términos del número total de operaciones básicas en el problema y está relacionado con el tamaño de datos de entrada. Si sustituimos T_1 por W, la eficiencia puede ser expresada por la ecuación:

$$E = \frac{1}{1 + \frac{T_0}{W}}. \tag{2.10}$$

Analizando la ecuación (2.10), se observa que si W se mantiene constante y p aumenta, entonces la eficiencia E disminuye debido a 'overhead' total que crece al

aumentar el número de procesadores p. Si W aumenta mientras p se mantiene constante, entonces para sistemas escalables la eficiencia crece. Esto sucede porque para un número de procesadores dado, T_0 crece más lento que $O(W)$. Para estos sistemas la eficiencia puede mantenerse a nivel deseado (entre 0 y 1) incrementando el número de procesadores p y el tamaño del problema W simultáneamente.

La **Escalabilidad** de un sistema paralelo es la capacidad del par algoritmo-máquina de mantener su eficiencia cuando se incrementa en la misma proporción el tamaño del problema y el número de procesadores del sistema.
Despreciando el 'overhead', la expresión para la eficiencia 2.7 se puede reescribir de la siguiente forma:

$$E = \frac{S_p}{p} = \frac{T_1}{T_1 + pT_p - T_1} = \frac{1}{1 + \frac{pT_p - T_1}{T_1}}, \tag{2.11}$$

o, con $T_1 = W$

$$E = \frac{1}{1 + \frac{pT_p - W}{W}} = \frac{1}{1 + I(W,p)}, \tag{2.12}$$

donde $I(W,p) = \frac{pT_p - W}{W}$ se conoce como la función de isoeficiencia. El valor pequeño de $I(W,p)$ implica la alta escalabilidad del sistema paralelo. En contrario, el valor grande de la función isoeficiencia indica la poca escalabilidad del sistema.
A efectos prácticos se establece la siguiente definición de escalabilidad: Dado un sistema paralelo HN con p procesadores para un trabajo M, y un sistema HN', con p' procesadores para un trabajo M'; se puede decir que HN es un sistema escalable si cuando el sistema es ampliado desde HN a HN', es posible seleccionar un problema M' tal que la eficiencia de HN y HN' permanezcan constantes.

Las fórmulas (2.8) y (2.9) expresan el SpeedUp y la Eficiencia teórica de un sistema. En la práctica, estas caractéricticas van a tener valores menores debido a las condiciones reales del sistema.

3 Tomografía Axial Computarizada

3.1. Visión general

En medicina, el diagnóstico basado en la tomografía axial computarizada (TAC) es fundamental para la detección de anormalidades por diferente atenuación de rayos-X, las cuales son difíciles de distinguir por los radiólogos.
El inicio en investigación sobre imágenes médicas (medical imaging) se remonta al año 1895 cuando Wilhelm Conrad Rontgen descubrió los rayos-X. Él mostró que los huesos podrían ser visualizados al atravesarlos por los rayos-X y, en consecuencia de este descubrimiento, recibió el primer premio Nobel de Física en 1901. Así apareció la radiografía que es una de las modalidades principales para la obtención de imágenes médicas. Desde entonces fueron inventadas diferentes modalidades médicas de obtención de imagen.

En medicina nuclear, se usan 'gamma-camera', 'positron emission tomography' (PET), 'single photon emission computed tomography' (SPECT) donde se miden los fotones de rayo gamma emitidos desde el interior del cuerpo humano. En las imágenes por resonancia magnética (MRI), se utilizan pulsos magnéticos para visualizar la capa interior de un objeto. En la ecografía se utiliza ultrasonido.
En TAC, se usa una fuente externa de rayos-X y se utilizan las propiedades de atenuación de un objeto atravesado por los rayos-X para reproducir la estructura interna de una capa del objeto calculando los valores de atenuación que se corresponden con el nivel de gris del píxel.

3.2. Principios básicos de Tomografía Computarizada

La tomografía axial computarizada representa una forma de obtener imágenes de secciones transversales de un objeto. Sus principios básicos se describen en [4], [5]. Para obtener la imagen de una capa interior, se utilizan rayos-X que atraviesan el objeto. En la Figura 3.1 se presenta la geometría paralela (a) y la de 'fan beam' (b) de flujo de rayos-X.

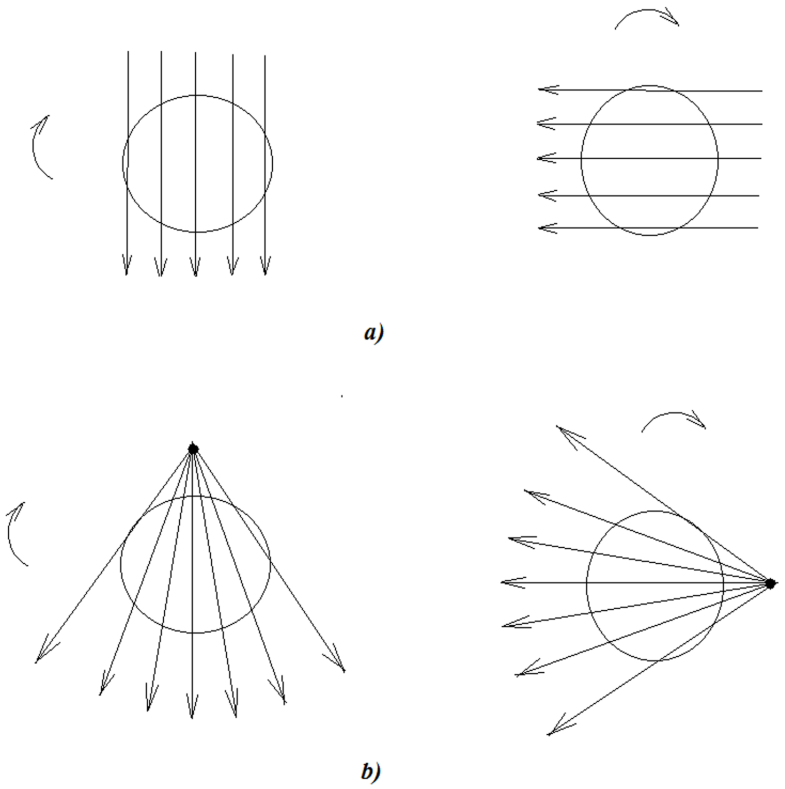

Figura 3.1: Proceso básico de escaneo en TAC: a) geometría paralela, b) geometría 'fan beam'.

El proceso de escaneo se repite para diferentes ángulos. Los rayos-X atenuados se capturan por detectores. De esta forma se obtiene un conjunto de datos que se utilizan para obtener la imagen de la estructura interna del objeto escaneado.

La Figura 3.2 representa la geometría paralela del proceso de escaneo del corte ho-

rizontal de un objeto. Sea μ(x,y) la distribución del coeficiente lineal de atenuación en el plano (x,y). $\mu = 0$ fuera del círculo llamado 'Field of View' (FOV) que se produce rotando la fuente alrededor del centro del objeto.

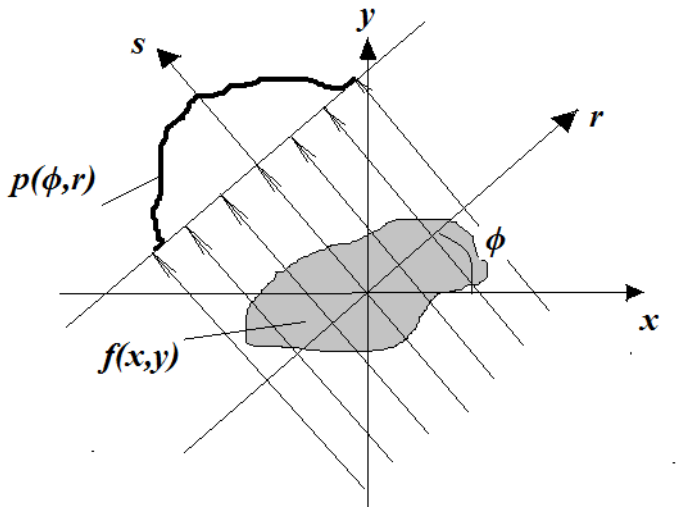

Figura 3.2: Proyection $p(\phi, r)$ para un ángulo ϕ.

Rotando la fuente de los rayos-X se define el sistema de coordenadas (r, s) y la relación entre las dos sistemas está dada por las fórmulas de transformación:

$$\begin{bmatrix} r \\ s \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}, \quad \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} r \\ s \end{bmatrix} \tag{3.1}$$

y el Jacobiano es:

$$J = \begin{vmatrix} \cos\phi & \sin\phi \\ -\sin\phi & \cos\phi \end{vmatrix} = 1. \tag{3.2}$$

Para un ángulo ϕ, la intensidad de rayos-X como función de la distancia del centro del objeto se define por la formula 3.3 y se presenta en la Figura 3.3(a).

$$I_\phi(r) = I_0 \exp^{-\int_L \mu(r\cos\phi - s\sin\phi, r\sin\phi + s\cos\phi)ds} \tag{3.3}$$

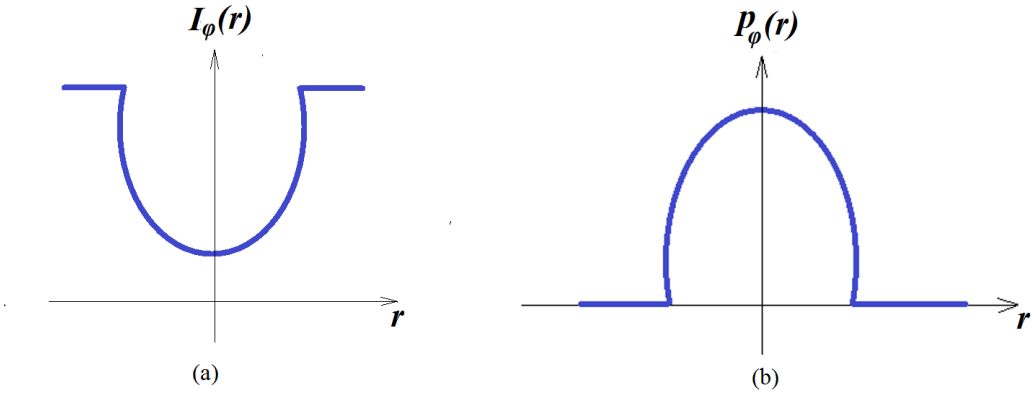

Figura 3.3: La intensidad y proyección de $\mu(x, y)$ para un ángulo como función de r.

Cada intensidad $I_\phi(r)$ se transforma en la proyección $p_\phi(r)$ representada en la Figura 3.3(b) y definida por la ecuación:

$$p_\phi(r) = -ln\frac{I_\phi(r)}{I_0} = \exp^{-\int_L \mu(r\cos\phi - s\sin\phi, r\sin\phi + s\cos\phi)ds}. \tag{3.4}$$

Está claro que $p_\phi(r) = 0$ para $\mid r \mid \geq$ FOV / 2.

Un conjunto de tales proyecciones medidas en el intervalo $[0 - 2\pi]$ resulta en el conjunto $\mathbf{p(r,\phi)}$ y representa un sinograma que se muesta en la Figura 3.4.

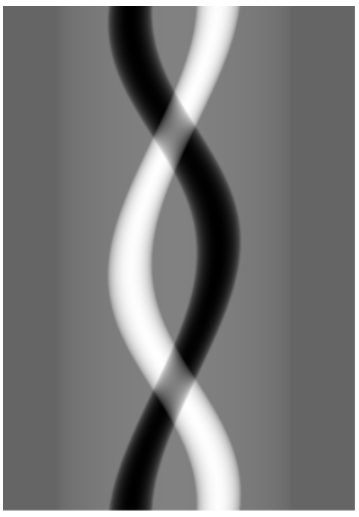

Figura 3.4: Un sinograma formado por el conjunto de proyecciones.

Cuando se considera la geometría paralela, es suficiente medir las proyecciones en [0,π].
La transformación de una función $f(x, y)$ en sus proyecciones paralelas $\mathbf{p(r,}\phi\mathbf{)}$ se llama la Transformada de Radon que se puede representar por la siguiente ecuación:

$$\mathbf{p(r,}\phi\mathbf{)} = \Re\{f(x, y)\} = \int_{-\infty}^{\infty} f(r\cos\phi - r\sin\phi, r\sin\phi + s\cos\phi)ds. \tag{3.5}$$

La Inversa de la Transformada de Radon de $f(x, y)$ representa la misma función:

$$f(x, y) = \Re^{-1}\{\mathbf{p(r,}\phi\mathbf{)}\}. \tag{3.6}$$

A diferencia de la radiografía donde los valores atenuados del rayo se superimponen a lo largo del rayo, en el TAC, los algoritmos de reconstrucción reconstruyen los valores de atenuación en cada píxel de la imagen. En escáneres modernos, una imagen se representa por 512x512 píxeles, donde el valor de cada píxel corresponde a nivel de color gris en la imagen.

El problema de reconstrucción de una función por sus proyecciones fue formulado por primera vez por Johann Radon in 1917 y en 1972 apareció el primer escáner desarrollado por Godfrey N. Hounsfield. Desde entonces, durante varias décadas los escáneres pasaron varias generaciones en su desarrollo: de escáneres estáticos hasta escáneres espirales. En los capítulos posteriores presentamos diferentes métodos de reconstrucción y analizaremos su implementación en diversas arquitecturas.

4 Métodos analíticos de reconstrucción

4.1. Introducción

Actualmente, en la mayoría de los escáneres comerciales el proceso de reconstrucción de imagen se basa en algoritmos analíticos entre los cuales, el algoritmo de retroproyección filtrada 'filtered backprojection' (FBP) es el más conocido [3], [4]. Este algoritmo se usa para implementar la Transformada Inversa de Radon que es una herramienta matemática cuya utilización principal en Ingeniería Biomédica es la reconstrucción de imágenes de TAC [1].
En TAC, la información sobre la imagen original se da en forma de un conjunto de proyecciones, que se conoce como la transformada de Radon de la imagen, y la inversa de la transformada proporciona la solución directa de problema de reconstrucción y representa la misma imagen.

FBP representa la solución analítica del problema de reconstrucción. En caso de las proyecciones ideales, es decir en caso de un número infinito de datos con rayos-X infinitamente finas, sin ruido, etc, la solución del problema de reconstrucción es ideal. Desafortunadamente, en la práctica se obtiene un número finito de medidas con un número finito de rotaciones de escáner. Además, las medidas contienen ruido debido a condiciones físicas reales de medición. Estas discrepancias dan lugar a artefactos en la imagen reconstruida. Usualmente, estos errores son relativamente pequeños, y el FBP proporciona resultados satisfactorios. Pero en situaciones extremas, por ejemplo, en presencia de objetos de alta atenuación, o en situaciones con conjunto de proyecciones no completos, los artefactos emborrosan la imagen.
Cuando se usa el método FBP, algunos artefactos se pueden reducir aplicando filtros a los datos iniciales, es decir al conjunto de proyecciones.

4.2. Aspectos matemáticos de reconstrucción de imagen

4.2.1. Relación con la transformada de Fourier

La Transformada de Radon está relacionada con la Transformada de Fourier por medio del Teorema de 'Slices' [3].
Sea $F(k_x, k_y)$ la transformada 2D de Fourier (2D FT) de $f(x, y)$:

$$F(k_x, k_y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y) \exp^{-2\pi i(k_x x + k_y y)} \, dxdy. \tag{4.1}$$

Sea $P(k, \phi)$ la transformada 1D de Fourier de $\mathbf{p(r,}\phi\mathbf{)}$:

$$P(k, \phi) = \int_{-\infty}^{\infty} \mathbf{p(r,}\phi\mathbf{)} \exp^{-2\pi i(k-r)} \, dr. \tag{4.2}$$

La función $P(k, \phi)$ está definida en la red polar y la función $F(k_x, k_y)$ está definida en la red rectangular, como se presenta en la Figura 4.1.

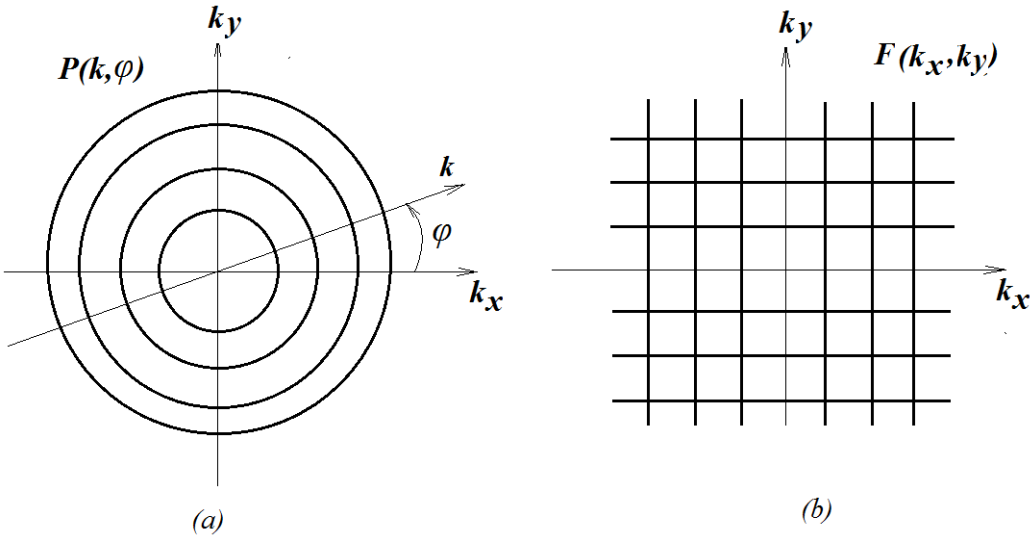

Figura 4.1: $P(k, \phi)$ está definida en la red polar (a) y $F(k_x, k_y)$ está definida en la red rectangular (b)

Por el **Teorema de 'Slices'**:

$$P(k, \phi) = F(k_x, k_y), \quad con \quad k_x = k \cos \phi \quad y \quad k_y = k \sin \phi. \tag{4.3}$$

Es decir, la transformada 1D de Fourier de la transformada de Radon de una función $f(x, y)$ es la 2D FT de esta función. Entonces, es posible calcular $f(x, y)$ para cada punto (x, y) basándose en sus proyecciones $p_\phi(r)$, variando ϕ entre $(0, \pi)$.

La versión polar de la inversa de 2D FT (4.1) considerando las relaciones 3.1 proporciona la solución del problema de reconstrucción en la siguiente forma:

$$f(x, y) = \int_0^\pi \int_{-\infty}^{\infty} P(k, \phi)|k| e^{i2\pi kr} dk d\phi, \tag{4.4}$$

con $r = x \cos\phi + y \sin\phi$.
En (4.4) $P(k, \phi)$ es la transformada 1D de Fourier de las proyecciones, $|k|$ es el determinante del Jacobiano al pasar de sistema de coordenadas rectangulares a polares y se denomina filtro rampa. El producto $P(k, \phi)|k|$ se obtiene multiplicando las proyecciones por $|k|$ en el espacio de Fourier. En el método FBP, el filtro rampa se usa para filtrar las proyecciones antes de realizar la reconstrucción para obtener proyecciones filtradas (ideales) en el espacio de frecuencias. Después de ser filtradas las proyecciones se retroproyectan y superponen para producir la imagen.

La fórmula (4.4) es la base del cálculo de la Transformada Inversa de Radon en la reconstrucción del TAC y representa la relación entre las proyecciones y la imagen inicial.
El proceso de reconstrucción por proyecciones con el algoritmo FBP se puede visualizar en la Figura 4.2 donde se muestra como se superponen las proyecciones filtradas para obtener la intensidad resultante en un píxel de la imagen.

4.3. Algoritmo FBP y arquitecturas multicore

Los pasos principales en la reconstrucción de imagen por el método FBP consisten en:

- aplicar la Transformada de Fourier a las proyecciones
- fitrar las proyecciones
- aplicar la Transformada de Fourier inversa

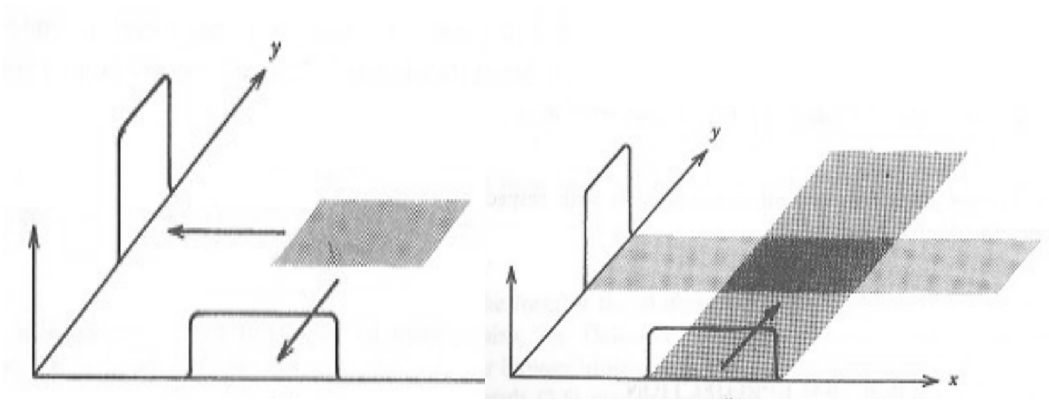

Figura 4.2: Reconstruccion por dos proyecciones: las proyecciones filtradas se superponen para formar la imagen

- formar la imagen sumando las proyecciones filtradas correspondientes a cada ángulo y para cada píxel de la imagen.

Muchas tareas del proceso de reconstrucción son independientes, por ejemplo la Transformada de Fourier se aplica a las columnas de la matriz de entrada, o para un ángulo, la suma de las proyecciones en cada píxel . Por lo tanto, se puede aprovechar las arquitecturas con memoria compartida para reducir el tiempo de reconstrucción. En la siguiente sección analizamos una posible paralelización de este algoritmo.

4.4. Paralelización

Las proyecciones tomadas por el escáner se dan en forma de una matriz PR_{MxN} donde M y N se corresponden con el número de filas y columnas en PR. Las columnas corresponden a los ángulos bajo los cuales se toman las proyecciones. Las filas se corresponden con los detectores en el escáner. Los coeficientes de la matriz de proyecciones representan las intensidades de los rayos-X registradas por los detectores al atravesar el objeto.

El diagrama de flujo del algoritmo se presenta en la Figura 4.3. El algoritmo consta de los siguientes módulos:

- módulo principal: Image Reconstruction

- módulo: Backproject
- módulo: Filter Projections

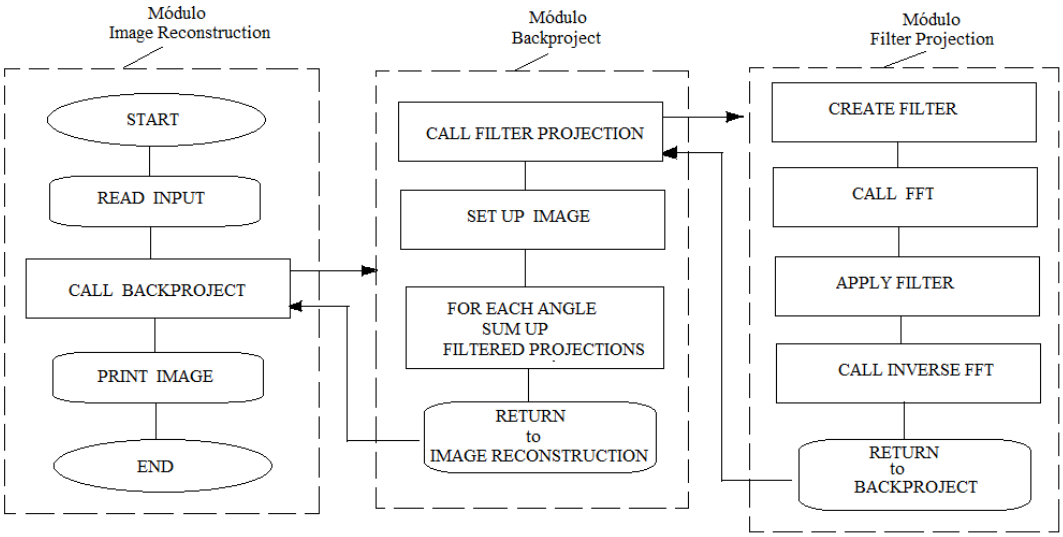

Figura 4.3: Diagrama de flujo del algoritmo FBP.

Descripción de los módulos.

- Módulo Image Reconstruction:
  - Leer Datos de entrada
  - Llamar Backproject
  - Generar el archivo de salida
- Módulo Backproject:
  - Llamar Filter projections para filtrar las proyecciones PR (**paso 9 del Algoritmo 4.1**)
  - Se hace una copia de las proyecciones filtradas (**paso 10**)
  - Inicializar la imagen a reconstruir BPR (**paso 11**)
  - Definir Grid rectangular (**pasos 13:19**)
  - Inicializar BPIa - la imagen actual que corresponde al ángulo actual (**paso 20**)

- Para cada ángulo se determinan los píxeles por los cuales pasa la proyección (**pasos 22:37**)
- Registrar los píxeles de pasos (**pasos 22:37**) en la imagen actual BPIa (**pasos 39:43**)
- Se actualiza y se normaliza la imagen final BPR (**pasos 44:50**)

- Módulo Filter projections:
  - Se determina el orden del filtro y se construye el Hamming filter (**pasos 51:68**)
  - Se inicializa la matriz FPR que representa las proyecciones filtradas (**paso 69**)
  - Las proyecciones PR se registran en FPR en orden de columna mayor(**pasos 70:74**)
  - Se aplica la transformada rápida de Fourier a las proyecciones (**pasos 75:78**)
  - Las proyecciones se filtran en el espacio de Fourier (**pasos 79:84**)
  - Se aplica la transformada inversa de Fourier a las proyecciones filtradas (**pasos 85:88**)
  - Las proyecciones se normalizan y se registran en orden de columna mayor (**pasos 89:94**)

A continuación se presenta el algoritmos FBP en forma de pseudocódigo de cada módulo.

Algoritmo 4.1: Image Reconstruction

- **Entrada:** PR_{MxN} - matriz de proyeciones.
- **Salida:** $Image_{MxM}$ - matriz que representan las intensidades de la imagen reconstruida.

01: Leer datos de entrada
02: Llamar **Backproject()**
03: Almacenar la imagen en el archivo de salida:

04: for $i = 1 : M$
05: for $j = 1 : M$
06: $write\ Image[i][j]$
07: end for
08: end for

Algoritmo 4.2: Backproject()

- **Entrada:** PR_{MxN}, ángulos.
- **Salida:** $Image_{MxM}$ - matriz-imagen.

01: Llamar **Filter projections()** $\Longrightarrow$ PR [ ]
Set up image
02: Convertir los ángulos a radianes:
03: cblas-dscal(theta size, pi/180, theta, 1)
04: Inicializar la imagen previa:
05: ImagenPrevia = 0

Set up las matrices con las coordenadas de las proyecciones
06: $midindex = (M + 1)/2$
07: **# pragma omp parallel**
08: for $i = 1 : M$
09: for $j = 1 : M$
10: $p_{xpr}[i][j] = j + 1 - (M + 1)/2$
11: $p_{ypr}[i][j] = i + 1 - (M + 1)/2$
12: end for
13: end for

Para cada ángulo escribir los índices de las proyecciones filtradas
14: **# pragma omp parallel for**
15: for $z = 1 : N$
16: # pragma omp parallel for private (j)
17: for $i = 1 : M$

18: for $j = 1 : M$
19: $pfiltIndex[z][i][j] = floor(0{,}5+midindex+p_{xpr}[i][j]*sin(theta[z])-$
20: $-p_{ypr}[i][j] * cos(theta[z])) - 1$
21: end for
22: end for

Corregir los índices que están fuera de la imagen:

23: $k = 0$
24: for $i = 1 : M$
25: for $j = 1 : M$
26: if $pfiltIndex[z][i][j] >= 0$ && $pfiltIndex[z][i][j] < M$
27: $pspota[z][k] = i * M + j$
28: $pnewfiltIndex[z][k] = pfiltIndex[z][i][j]$
29: $k = k + 1$
30: end if
31: end for
32: end for

Determinar el píxel de la imagen por el cual pasa la proyección

33: # pragma omp parallel for private(i,i1,i2)
34: for $i = 1 : k$
35: $i1 = pspota[z][i]$
36: $i2 = pnewfiltIndex[z][i]$
37: $ImagenPrevia[z][i1] = pfiltPR[i2][z]$
38: end for

Sumar las proyecciones para formar la imagen

39: for $i = 1 : M * M$
40: Imagen[i] =Imagen[i]+ ImagenPrevia[z][i]
41: end for
42: end for /* End of the angle loop */

Algoritmo 4.3: Filter Projections()

- **Entrada:** PR_{MxN}.
- **Salida:** PR_{MxM}

Determinar el orden del filtro

01: $a = floor(\log(2 * M)/\log(2))$
02: if $(pow(2, a) < (2 * M))$
03: $\quad a = a + 1$
04: end if
05: order = máx$(64, pow(2, a))$ /* order del filtro

Crear filtro Hamming

\# pragma omp parallel for
06: for $i = 1 : order/2$
07: $\quad H1[i] = 2 * i/order$
08: $\quad w[i] = 2 * pi * i/order$
09: $\quad H1[i] = H1[i] * (0{,}54 + 0{,}46 * \cos(w[i]/d))$
10: end for
11: cblas-dcopy (order/2+1, H1, 1, H2, 1)
12: k = order/2
13: for $i = order/2 - 1 : 1$
14: $\quad H2[k + 1] = H[i]$
15: $\quad k = k + 1$
16: end for

Crear la matriz transpuesta de las proyecciones

17: cblas-dscal(order*PRcols, 0, FPR, 1)
18: for $i = 1 : N$
19: $\quad$ for $j = 1 : M$
20: $\quad\quad pFPR[i][j] = pPR[j][i]$
21: $\quad$ end for
22: end for

Aplicar la transformada de Fourier a las proyecciones

23: # pragma omp parallel for
24: for $i = 0 : N$
25: call FFT (FPR) $\Longrightarrow$ out[ ]
26: end for

Filtrar las proyecciones en el espacio de Fourier:

27: # pragma omp parallel for private(j)
28: for $i = 1 : N$
29: for $j = 1 : order$
30: $pout[i][j] = pout[i][j] * H2[j]$
31: end for
32: end for

Aplicar la inversa de la transformada de Fourier a las proyecciones filtradas

33: # pragma omp parallel for
34: for $i = 0 : N$
35: call inverse FFT (out) $\Longrightarrow$ FPR[ ]
36: end for
37: for $i = 1 : N$
38: for $j = 1 : M$
39: $pPR[j][i] = pFPR[i][j]/order$
40: end for
41: end for
42: end of Filter-projections

4.5. Evaluación de coste

Utilizaremos las métricas descritas en la sección 2.11 del capítulo 2 para evaluar el coste del algoritmo FBP. Vamos a suponer que el algoritmo se ejecuta en un sistema con memoria compartida.

Coste del módulo Backproject.

Sea N el número de ángulos, MxM el tamaño de la imagen a reconstruir y p el número de procesos.

- Tiempo secuencial:
  $T_1 = \sum_{i=1}^{M}\sum_{j=1}^{M} 8 + \sum_{z=1}^{N}\left[\sum_{i=1}^{M}\sum_{j=1}^{M} 6 + \sum_{i=1}^{M}\sum_{j=1}^{M} 5 + \sum_{i=1}^{M*M} 1\right].$

  $T_1 = (8 + 12N)M^2.$

- Tiempo paralelo:
  $T_p = \sum_{i=1}^{\frac{M}{p}}\sum_{j=1}^{M} 8 + \sum_{z=1}^{\frac{N}{p}}\left[\sum_{i=1}^{M}\sum_{j=1}^{M} 6 + \sum_{i=1}^{M}\sum_{j=1}^{M} 5 + \sum_{i=1}^{M*M} 1\right].$

  $T_p = \frac{1}{p}M^2(8 + 12N).$

- SpeedUp límite teórico del sistema: $S = \lim_{N\to\text{ínf}} \frac{T_1}{T_p} = p.$

- Eficiencia límite teórica del sistema: $E = \lim_{N\to\text{ínf}} \frac{S}{p} = 1.$

Coste del módulo Filter projections.

Sea *order* - el orden del filtro, M y N - el número de filas y columnas en la matriz de proyecciones.

- Tiempo secuencial: $T_1 = \sum_{i=1}^{order/2} 10 + \sum_{i=1}^{N} callFFTP + \sum_{i=1}^{N}\sum_{j}^{order} 1 + \sum_{i=1}^{N} inverseFFTP + \sum_{j=1}^{N}\sum_{i=1}^{M} 1.$

$$T_1 \approx 10 * order + 2N + N * order + NM.$$

- Tiempo paralelo: $T_p = \sum_{i=1}^{order/2p} 10 + \sum_{i=1}^{N/p} callFFTP + \sum_{i=1}^{N/p} \sum_{j}^{order} 1 + \sum_{i=1}^{N/p} inverseFFTP + \sum_{j=1}^{N/p} \sum_{i=1}^{M} 1.$

$$T_p \approx 10\frac{order}{p} + 2\frac{N}{p} + \frac{N}{p}order + \frac{N}{p}M.$$

- SpeedUp: $S = \frac{T_1}{T_p} = p,$ Eficiencia: $E = \frac{S}{p} = 1.$

4.6. Resultados experimentales

El algoritmo desarrollado en el apartado anterior, como se ha analizado y como muestra la fórmula (4.4), es paralelizable ya que la reconstrucción se realiza mediante las proyecciones que son independientes. En este trabajo se trató de analizar el grado de paralelización para aprovechar al máximo los recursos del sistema.
El algoritmo se testó en el sistema KAHAN. Se han adquirido imágenes digitales en formato DICOM, que es el formato que se está implantando en el campo de la radiología médica, de tamaños 6.46MB, 13.3MB, 28.7MB. Para la reconstrucción de las imágenes se usaron las proyecciones paralelas sintéticas de estas imágenes generadas en Matlab en el rango de 0 a 180 grados.

Se han analizado el algoritmo secuencial y paralelo, obteniendo los tiempos de ejecución, variando el número de hilos OpenMP (procesos) p relacionado con los núcleos y procesadores del sistema, y el tamaño de problema N que está relacionado con el número de detectores del escáner y el número de ángulos en el proceso de adquisición de datos.
Los tiempos de reconstrucción (en segundos) de imágenes de diferentes tamaños se resumen en la Tabla 4.1.
En la Figura 4.4 se observa la dependencia del tiempo de ejecución del algoritmo FBP en función del tamaño de problema N y número de procesos p. En la Figura 4.5 se presenta la variación de SpeedUp y Eficiencia en función del número de pro-

$p \setminus N$	367x90	367x180	729x180	1453x180
1	2.2	4.3	14.6	62.5
2	1.1	2.1	7.6	30.9
4	0.6	1.1	4.1	14.9
8	0.3	0.7	2.5	9.1
16	0.2	0.4	1.4	5.7
32	0.1	0.2	0.9	3.3
64	0.1	0.2	0.8	2.9

Tabla 4.1: Tiempo de reconstrucción con FBP en un nodo del cluster KAHAN

cesos y el tamaño del problema N en KAHAN.

Analizando los resultados de Tabla 4.1 y Figura 4.5 acerca del grado de paralelismo y eficiencia del algoritmo en el sistema estudiado, se observa que:

- En el KAHAN, el tiempo óptimo se logra con 32 procesos. El tiempo de reconstrucción de imagen con 32 procesos disminuye en 19.5 veces comparado con el tiempo necesario para el mismo cálculo con un solo proceso, lo que lleva a una eficiencia del 61 % del sistema.

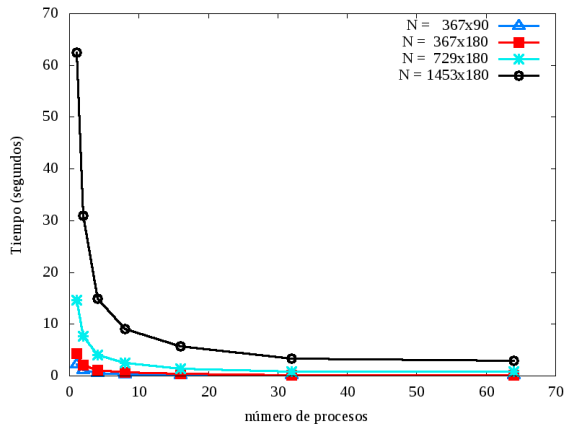

Figura 4.4: Variación de tiempo de ejecución del algoritmo FBP en un nodo de KAHAN

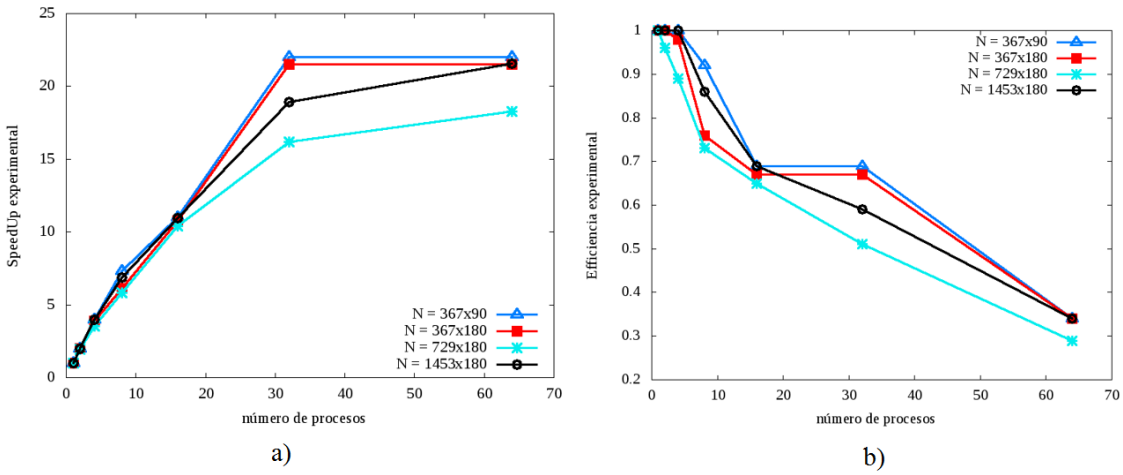

Figura 4.5: Variación de SpeedUp (a) y de Eficiencia (b) experimentales de la ejecución del algoritmo FBP.

A continuación, en las figuras 4.6- 4.8, se presenta a modo de ejemplo una imagen sintética reconstruida a partir de 180 proyecciones mediante el algoritmo secuencial y el paralelo en el sistema estudiado. En la implementación de este algoritmo analítico, ambas imágenes reconstruidas coinciden y visualmente ambos métodos secuencial y paralelo obtienen una reconstrucción similar a la imagen original.

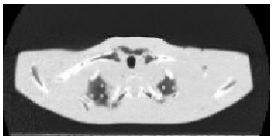

Figura 4.6: Imagen original

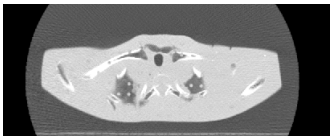

Figura 4.7: Reconstrucción secuencial

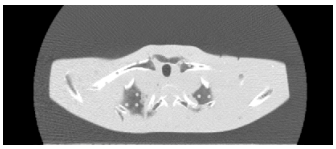

Figura 4.8: Reconstrucción paralela

4.7. Conclusiones

Los resultados muestran que el algoritmo de retroproyección filtrada es paralelizable y las condiciones óptimas de ejecución en una arquitectura con memoria compartida se consiguen cuando el algoritmo se ejecuta en forma paralela, utilizando el número de procesos igual al número máximo de hilos en un nodo del sistema. La utilización de varios nodos no aumenta el speedUp y disminuye la eficiencia del

sistema. Esto se debe a los costos de transmisión de datos entre nodos de un cluster.

5 Reconstrucción iterativa

5.1. Introducción

En el **Capítulo 4** se ha analizado el algoritmo FBP de reconstrucción de imágenes. El algoritmo se basa en la transformada inversa de Radon y representa la solución directa del problema de reconstrucción. En condiciones ideales (proyecciones sin ruido, con número de infinito de proyecciones), FBP representa la reconstrucción ideal.

Los métodos iterativos proponen una forma diferente de reconstrucción. Aquí, el proceso empieza con la representación discreta de datos de medición (proyecciones) y de la imagen a reconstruir. El problema de reconstrucción se lleva a cabo de la siguiente forma: se define la función objeto y después esta función se optimiza.
Se proponen varios tipos de función objeto y varios métodos de optimización. Usualmente, todo se reduce a un algoritmo iterativo en el cual se actualiza la función objeto. En este trabajo hemos incluido los siguientes algoritmos iterativos: la técnica de reconstrución algebraica simultánea (SART),' Maximum Likelihood' para la tomografía de emisión (MLEM), y el método Least Square QR (LSQR), todos ellos se describen en las secciones posteriores.

5.2. Métodos iterativos: ventajas y desventajas

Aunque en la primera reconstrucción tomográfica por proyecciones se utilizaron métodos iterativos, en la actualidad el proceso de reconstrucción en escáneres clínicos se basa en algoritmos analíticos.
Sin embargo, los métodos iterativos representan una opción dominante debido a dos razones. Primero, los métodos analíticos necesitan una colección de datos completa,

lo que no siempre es posible. Segundo, estos métodos no proporcionan reconstrucción óptima en condiciones ruidosas, e.g. [14]. En condiciones de ruido, los métodos iterativos permiten la reconstrucción de imágenes de más alto contraste y precisión con un menor número de proyecciones que los métodos basados en la transformada de Fourier [11], [12].
En TAC, es común encontrar proyecciones incompletas, proyecciones no igualmente espaciadas. En estos casos los métodos iterativos reconstruyen imágenes de mejor calidad [15], [16].
Una de las líneas de investigación donde puede ser utilizada esta metodología está relacionada con equipos tomográficos portátiles [17]. Este tipo de escáneres pueden ser usados para realizar exámenes de urgencia en cualquier lugar. Ellos no producen datos igualmente espaciados, y, por esta razón, la reconstrucción iterativa es más apropiada en estos equipos.

Sin embargo, la mayor desventaja de los métodos iterativos es su alto costo computacional. En este trabajo, mostraremos y analizaremos la implementación eficiente de algoritmos iterativos en la reconstrucción de imágenes sobre arquitecturas heterogéneas actuales.

5.3. Aspectos matemáticos de reconstrucción iterativa

Suponemos que la matriz x es la versión digital de una imagen $f(x, y)$ como se ilustra en la Figura 5.1.
Esto significa que la imagen $f(x, y)$ puede ser aproximada por los elementos de la matriz cuyos elementos corresponden a las intensidades de la imagen. Asumimos que la imagen encaja en una región cuadrada y puede ser aproximada por la matriz x[1] con las dimensiones nxn. La proyección p_k tomada bajo un ángulo ϕ se presenta en la Figura 5.1 y puede ser calculada por la fórmula (5.1)

$$\sum_{i=1}^{m} \sum_{j=1}^{n^2} a_{ij}(k, \phi_r) x_j = p_{k,\phi_r}. \tag{5.1}$$

En (5.1), los valores $a_{ij}(k, \phi_r)$ representan la contribución de cada píxel de la imagen

[1] La matriz x queda almacenada en un array unidimensional

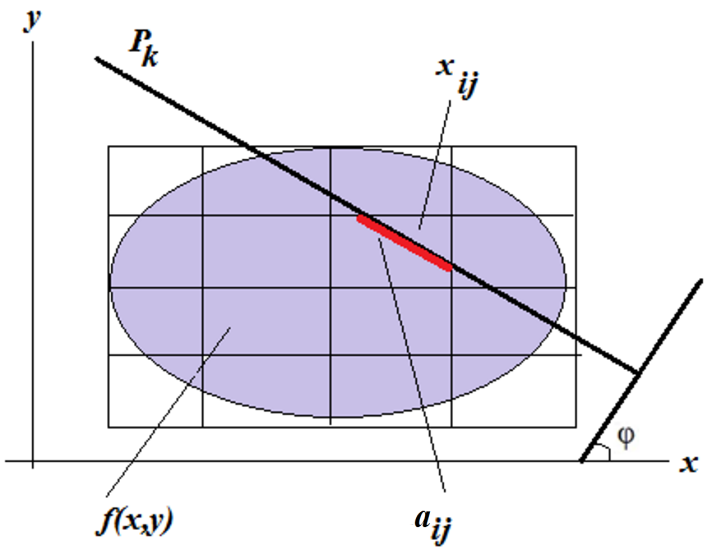

Figura 5.1: Proyección P_k para un ángulo ϕ.

a cada proyección y dependen de la proyección k y del ángulo ϕ_r ; x_j son intensidades incógnitas de la imagen (atenuaciones); p_{k,ϕ_r} son las proyecciones tomadas por el escáner.

En forma matricial la ecuación (5.1) está dada por la fórmula (5.2)

$$\mathbf{A}x = p, \tag{5.2}$$

donde $\mathbf{A}$ tiene dimensiones mxn^2, m representa número de proyecciones y es igual al producto del número de detectores y ángulos bajo los cuales se toman las proyecciones. La matriz $\mathbf{A}$ está formada por los elementos $a_{ij}(k, \phi_r)$ y se conoce como matriz de sistema y, en general, puede ser no cuadrada.

Los algoritmos de reconstrucción por proyecciones que se basan en métodos algebraicos se conocen como algoritmos iterativos. Básicamente, estos algoritmos de reconstrucción de imagen son esquemas para resolver sistemas de ecuaciones de la forma (5.2). El caso más simple, es el de una matriz de cuatro píxeles (2x2), como se muestra en la Figura 5.2. En este caso, hay cuatro proyecciones que van a generar un sistema de cuatro ecuaciones con cuatro incógnitas.

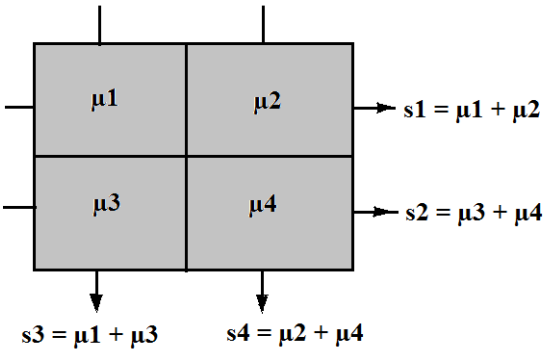

Figura 5.2: Sistema de 4 ecuaciones y 4 incógnitas.

La extensión a una matriz de 3x3 con nueve incógnitas puede ser resuelta con doce valores de medición (Figura 5.3).

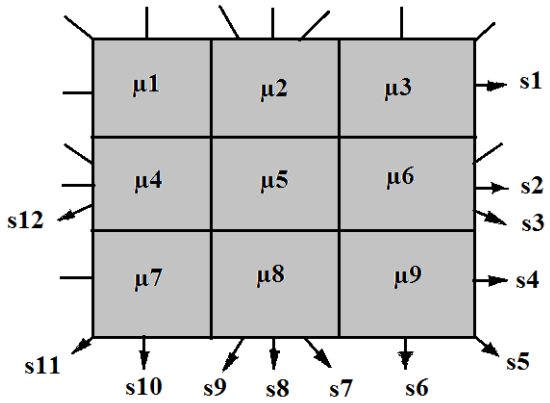

Figura 5.3: Sistema de 12 ecuaciones y 9 incógnitas.

Las dimensiones de la matriz **A** crecen proporcionalmente con la resolución de la imagen que se va a reconstruir y al número de proyecciones, elevando de esta forma el coste computacional. Los computadores de hoy están equipados con procesadores multi-cores. Esta tecnología permite paralelizar el cómputo asignando cada parte independiente a un proceso lo que posibilita un manejo de recursos de sistema más eficiente. En este trabajo, mostraremos diferentes mecanismos para la implementa-

ción paralela de algoritmos de reconstrucción utilizados para reducir el tiempo de cómputo.

Para un ángulo dado, asumimos que el número de proyecciones varía de 1 a m (m es igual al número de detectores). Si tomamos k diferentes ángulos, entonces en (5.2) p es el vector-columna con $m_{\times}k$ elementos, x es una matriz-columna con n^2 elementos

$$p = \begin{bmatrix} p_{11} \\ ... \\ p_{m1} \\ ... \\ p_{mk} \end{bmatrix}, \; x = \begin{bmatrix} x_{11} \\ x_{12} \\ ... \\ x_{nn} \end{bmatrix}. \tag{5.3}$$

Y la matriz **A** es una matriz rectangular con las dimensiones $mk_{\times}n^2$

$$A = \begin{bmatrix} a_{11}(11) & a_{12}(11) & ... & a_{nn}(11) \\ ... & ... & ... & ... \\ a_{11}(m1) & a_{12}(m1) & ... & a_{nn}(m1) \\ ... & ... & ... & ... \\ a_{11}(mk) & a_{12}(mk) & ... & a_{nn}(mk) \end{bmatrix}. \tag{5.4}$$

La matriz **A** puede ser calculada de diferentes formas. En este trabajo hemos usado el método de Siddon [53] para calcular los elementos de la matriz en una grid rectangular. Se muestra que el método proporciona buenos resultados en la generación de los elementos de la matriz de sistema [54]. El algoritmo de Siddon se describe a continuación.

5.4. Construcción de la matriz del sistema: método de Siddon

Para los cálculos de los pesos a_{ij} de la matriz **A** Robert Siddon propuso un algoritmo cuya idea básica consiste en hacer los pesos proporcionales a la longitud de línea que atraviesa el píxel.

El método se ilustra en la Figura 5.4 donde se presenta un rayo que atraviesa una imagen discretizada entre los puntos P_1 y P_2.

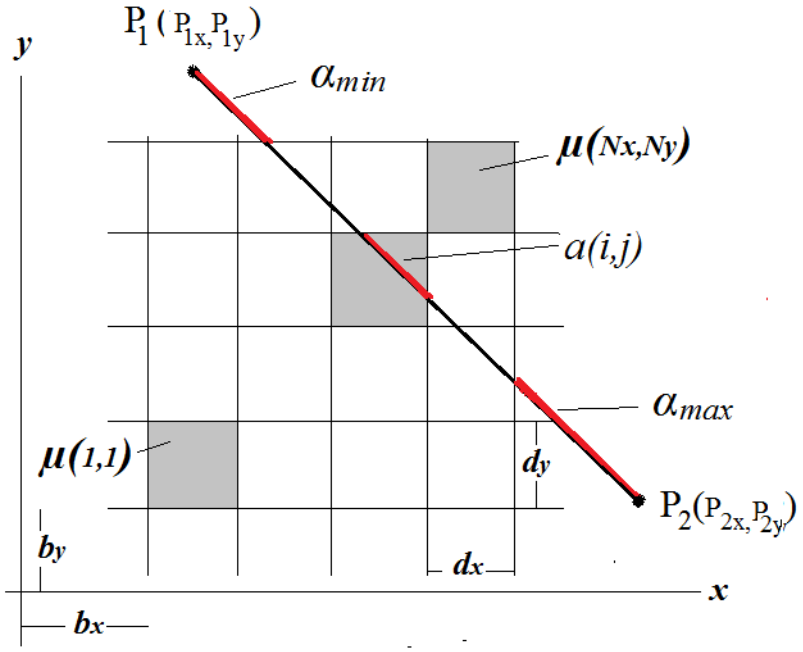

Figura 5.4: Ilustración del algoritmo de Siddon.

Notaciones.

Los factores de peso del píxel (i, j) se denotan por $a(i, j)$ y son iguales a la longitud de la intersección del rayo con el píxel (i, j). En la Figura 5.4, N_x y N_y representan el número de planos perpendiculares a los ejes x y y, α_{min} y α_{max} denotan la distancia entre el punto P_1 y el primer plano interceptado (α_{min}) y la distancia entre el último plano interceptado y el punto P_2 (α_{max}).

El valor de a_{ij} para una proyección se presenta también en la Figura 5.5.

De esta forma, la integral de línea del punto P_1 al punto P_2 en forma discreta puede ser expresada mediante la fórmula (5.5)

$$p_{12} = \sum_{i,j} a(i, j)\mu(i, j). \tag{5.5}$$

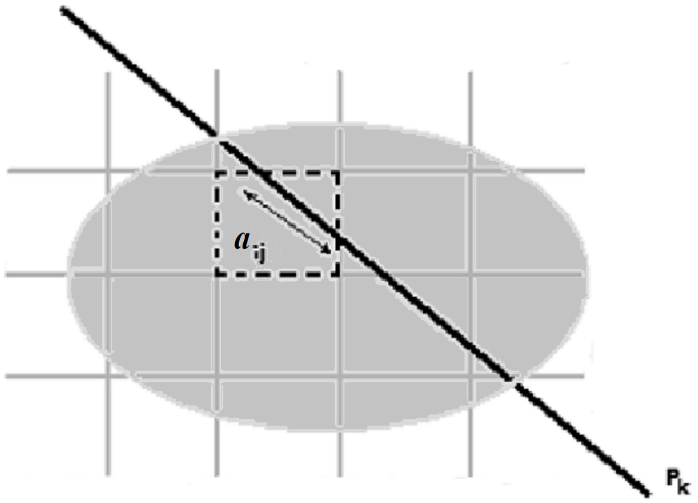

Figura 5.5: El valor a_{ij} de la proyeccion p_{k,ϕ_r}.

En (5.5) $\mu(i,j)$ representa el valor de píxel que corresponde a la intensidad de la imagen (atenuación). Sería muy ineficiente evaluar la ecuación (5.5) para todos los índices i, j, pues la mayoría de los valores de $a(i,j)$ son ceros. La mejor forma es seguir el rayo al pasar del punto P_1 a P_2.

Se usa la forma paramétrica del rayo:

$$p_{12} = \left\{ \begin{array}{rcl} p_x(\alpha) & = & p_{1x} + \alpha(p_{2x} - p_{1x}) \\ p_y(\alpha) & = & p_{1y} + \alpha(p_{2y} - p_{1y}) \end{array} \right\}. \tag{5.6}$$

En (5.6) $\alpha \in [0,1]$ para puntos entre P_1 y P_2, y $\alpha \notin [0,1]$ para otros puntos.

En este trabajo, el algoritmo de Siddon se emplea para generar los elementos de la matriz del sistema (5.2) **A** que simula el proceso de escaneo de la imagen. La matriz se calcula una vez y está asociada a las características del escáner empleado. Una vez generada para una imagen de resolución determinada, la matriz se usa para la resolución iterativa del sistema 5.2. En la Figura 5.6 se muestra el esquema del proceso de escaneo de una imagen. La fuente de rayos-X gira alrededor del centro

del objeto emitiendo rayos cuya intesidad atenuada se detecta por detectores del escáner.

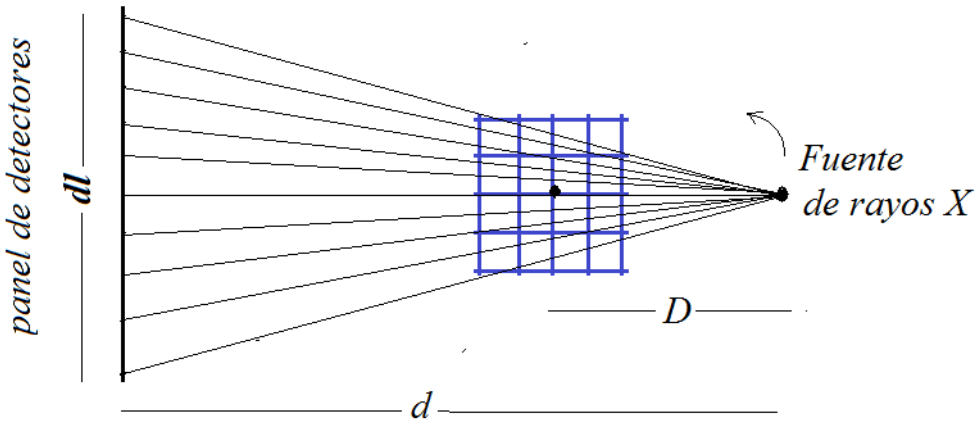

Figura 5.6: El esquema de escaneo de un objeto por rayos$-X$.

Algoritmo de Siddon

Entrada: N_d - el número total de detectores, Nz - el número de ángulos en que se toman las medidas, D - distancia de la fuente de rayos-X hasta el centro de la imagen, d - distancia de la fuente al panel de detectores, dl - longitud del panel de detectores.

Salida:

En la salida el algoritmo genera la matriz del sistema A_{Nx3} almacenada en el formato coordenado ordenado por filas disperso (COO):

- En la primera columna se registra el número de fila del elemento no nulo de la matriz **A**. Las filas estan ordenadas de menor a mayor.
- En la segunda columna se indica el número de columna del píxel con el valor no cero.
- En la tercera columna se indica el valor del elemento no nulo. NNZ indica el número total de elementos no nulos.

En la Figura 5.7 se presenta el diagrama de flujo del algoritmo de Siddon y, a continuación, se detalla las partes principales de este método.

Algoritmo 5.4.1: siddonMain() {

01: leer datos de entrada
02: calcular coordenadas de la fuente: $Ps_x[]$, $Ps_y[]$
03: empezar el bucle angular:
04: for $phi = 1 : Nz$
05: calcular coordenadas $Px[]$, $Py[]$ de detectores
06: elegir un par de puntos Fuente-Detector: $P1$, $P2$
07: $P1_x = Ps_x[phi]$
08: $P1_y = Ps_y[phi]$
09: for $detector = 1 : Nd$
10: $P2_x = Px[detector]$
11: $P2_y = Py[detector]$
12: calcularLongitudes ($P1_x, P1_y, P2_x, P2_y$)
13: endfor
14: endfor
15: return 0
16: }

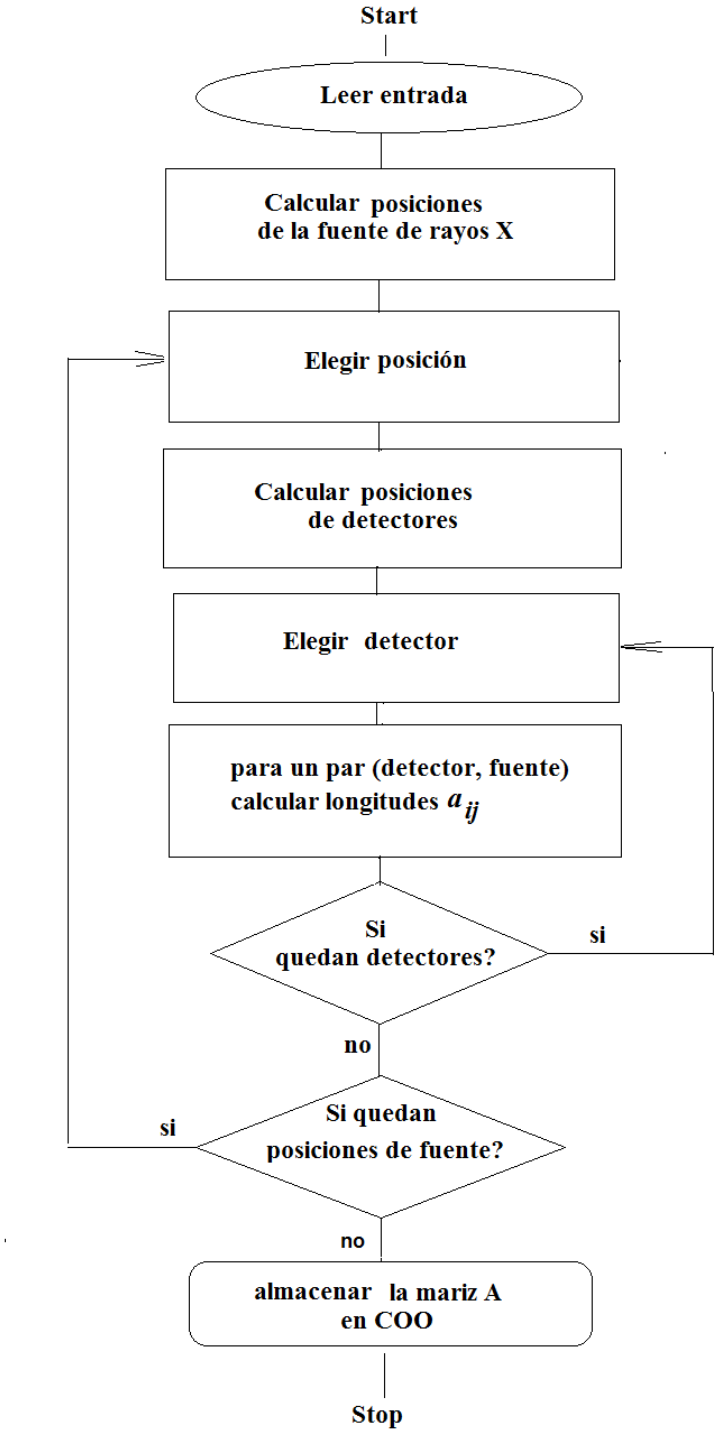

Figura 5.7: El diagrama de flujos del algoritmo de Siddon.

Algoritmo 5.4.2: calcularLongitudes() {

01: Calcular los parámetros α correspondientes a la intersección de $i-th$ x-plano y $i-th$ y-plano con el rayo que pasa de P_1 a P_2 por las fórmulas (5.7)

$$\begin{aligned} \alpha_x(i) &= \frac{(b_x+id_x)-P_{1x}}{P_{2x}-P_{1x}} \\ \alpha_y(i) &= \frac{(b_y+id_y)-P_{1y}}{P_{2y}-P_{1y}}. \end{aligned} \tag{5.7}$$

02: Determinar los puntos de entrada ($\alpha = \alpha_{min}$) y salida ($\alpha = \alpha_{max}$) por las fórmulas (5.8)

$$\begin{aligned} \alpha_{min} &= \text{máx}(\alpha_{xmin}, \alpha_{ymin}) \\ \alpha_{max} &= \text{máx}(\alpha_{xmax}, \alpha_{ymax}) \end{aligned} \tag{5.8}$$

con

$$\begin{aligned} \alpha_{xmin} &= \text{mín}(\alpha_x(0), \alpha_x(N_x-1)) \\ \alpha_{xmax} &= \text{máx}(\alpha_x(0), \alpha_x(N_x-1)) \\ \alpha_{ymin} &= \text{mín}(\alpha_y(0), \alpha_y(N_y-1)) \\ \alpha_{ymax} &= \text{máx}(\alpha_y(0), \alpha_y(N_y-1)). \end{aligned} \tag{5.9}$$

03: Calcular el número del primer i_{min} y el último i_{max} x-planos interceptados con el rayo después que este entra en el espacio de píxeles.

Para los puntos $P_{1x} < P_{2x}$ usar fórmulas (5.10), para los puntos $P_{1x} > P_{2x}$ usar (5.11)

$$\begin{aligned} \alpha_{min} = \alpha_{xmin} &\rightarrow i_{min} = 1 \\ \alpha_{min} \neq \alpha_{xmin} &\rightarrow i_{min} = \lceil \phi_x(\alpha_{min}) \rceil \\ \alpha_{max} = \alpha_{xmax} &\rightarrow i_{max} = N_x - 1 \\ \alpha_{max} \neq \alpha_{xmax} &\rightarrow i_{min} = \lfloor \phi_x(\alpha_{max}) \rfloor \end{aligned} \tag{5.10}$$

$$\begin{aligned} \alpha_{min} = \alpha_{xmin} &\rightarrow i_{max} = N_x - 2 \\ \alpha_{min} \neq \alpha_{xmin} &\rightarrow i_{max} = \lfloor \phi_x(\alpha_{min}) \rfloor \\ \alpha_{max} = \alpha_{xmax} &\rightarrow i_{min} = 0 \\ \alpha_{max} \neq \alpha_{xmax} &\rightarrow i_{min} = \lceil \phi_x(\alpha_{max}) \rceil . \end{aligned} \tag{5.11}$$

Las definiciones de $\phi_x(\alpha)$ están dadas por la fórmula (5.12)

$$\phi_x(\alpha) = \frac{P_x(\alpha) - b_x}{d_x}. \tag{5.12}$$

Para el primer y el último y-planos interceptados por el rayo, se usan fórmulas similares.

04: Calcular dos conjuntos $\alpha_x\,[...]$ y $\alpha_y\,[...]$ con los valores de parámetros de los puntos de intersección del rayo con los planos x e y respectivamente.
Si $P_{1x} < P_{2x}$ el primer conjunto está dado por la ecuación (5.13), en caso contrario, por la ecuación (5.14)

$$\alpha_x\,[i_{min}...i_{max}] = (\alpha_x(i_{min}), \alpha_x(i_{min}+1), ..., \alpha_x(i_{max})) \tag{5.13}$$

$$\alpha_x\,[i_{max}...i_{min}] = (\alpha_x(i_{max}), \alpha_x(i_{max}-1), ..., \alpha_x(i_{min})). \tag{5.14}$$

05: Ordenar los conjuntos en orden ascendente y reemplazar los valores dobles por uno solo. El conjunto resultante $\alpha_{xy}\,[0, ..., N_v]$ contiene los valores paramétricos de todos los puntos de intersección.

06: Usando el conjunto α_{xy} calculamos las coordenadas de los píxeles intersectados por las fórmulas (5.15) y las longitudes del rayo, que pasa por este píxel, por las fórmulas (5.16) para todos los $m \in [1, ..., N_v]$

$$\begin{aligned} i_m &= \left\lfloor \phi_x \left(\tfrac{\alpha_{xy}[m]+\alpha_{xy}[m-1]}{2} \right) \right\rfloor \\ j_m &= \left\lfloor \phi_y \left(\tfrac{\alpha_{xy}[m]+\alpha_{xy}[m-1]}{2} \right) \right\rfloor \end{aligned} \tag{5.15}$$

$$a(i_m, j_m) = (\alpha_{xy}[m] - \alpha_{xy}[m-1])D^*. \tag{5.16}$$

En (5.16) D^* representa la distancia cartesiana entre los puntos P_1 y P_2. Los valores $a(i,j)$ representan los factores de peso de un píxel en la matriz del sistema.
El sistema de ecuaciones (5.2) puede ser sobredeterminado o subdeterminado. Los sistemas sobredeterminados contienen más información sobre el proceso de escaneo de la imagen y, por lo tanto, las imágenes reconstruidas son menos ruidosas.

5.4.1. Siddon - versión paralela

La idea de la paralelización del algoritmo de Siddon consiste en la construcción de la matriz del sistema para cada ángulo por separado. La región paralela empieza en el bucle externo (paso 04 del Algoritmo 1). Se generan hilos, cada uno de los cuales genera una matriz para un ángulo. La implementación de la versión paralela se basa en modo de programación OpenMP.

Entrada: N_d - número de detectores, Nz - número de vistas (ángulos) en que se toman las medidas, D - distancia de la fuente de los rayos -X hasta el centro de la imagen, d - distancia de la fuente al panel de detectores, dl - longitud del panel de detectores, *start*, *stop* - el rango de ángulos para generar las matrices.

Salida: conjunto de matrices del sistema en formato COO

Algoritmo 5.4.3: siddonParalelo()

01: leer datos de entrada
02: calcular coordenadas de la fuente: $Ps_x[]$, $Ps_y[]$
03: paralelizar el bucle angular:
04: # pragma omp parallel for
05: for $phi = start : stop$
06: calcular coordenadas $Px[]$, $Py[]$ de detectores
07: elegir un par de puntos Fuente-Detector: $P1$, $P2$
08: $P1_x = Ps_x[phi]$
09: $P1_y = Ps_y[phi]$
10: for $sensor = 1 : NumSensors$
11: $P2_x = Px[sensor]$
12: $P2_y = Py[sensor]$
13: calcularLongitudes ($P1_x, P1_y, P2_x, P2_y$)
14: endfor
15: endfor

El método de Siddon genera la matriz en el formato compacto COO ordenado por filas. La matriz resultante del tamaño [3xN] consiste de tres columnas de elementos: [rows columns values] con la información de elementos no ceros. La matrices en este formato se utilizarán en los experimentos para estudiar el comportamiento de los algoritmos iterativos en las secciones posteriores.

5.4.2. Estudio de la matriz del sistema

La matriz del sistema **A** se define por la fórmula (5.4). En la práctica, **A** es una matriz rectangular y de alta dispersidad.

El número de columnas de **A** se define por la resolución de la imagen (e.g. 256x256 píxeles), y el número de filas es igual al producto del número de detectores por número de ángulos para los cuales se toman las proyecciones. Por lo tanto, las dimensiones de **A** crecen proporcionalmente a la resolución de la imagen y al número de ángulos.

La estructura de la matriz **A** que simula el proceso de escaneo en el rango de 0 a 360 grados con un paso angular de 9 grados para una imagen de 128x128 píxeles se visualiza en la Figura 5.8.

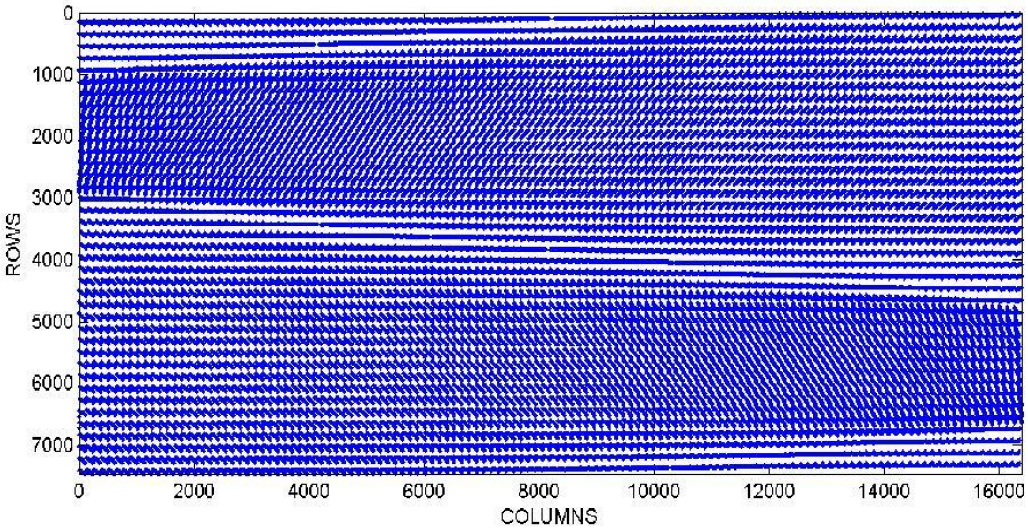

Figura 5.8: La estructura de la matriz del sistema.

La imagen aumentada de **A** se presenta en la Figura 5.9.

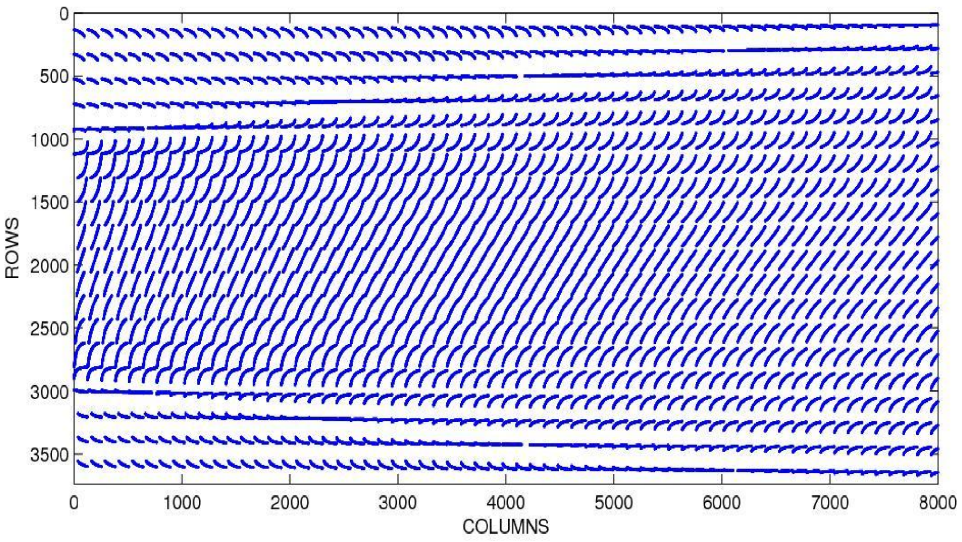

Figura 5.9: Visión aumentada de **A**.

Una parte de la matriz **A** con 1500 filas y 2500 columnas se presenta en la Figura 5.10.

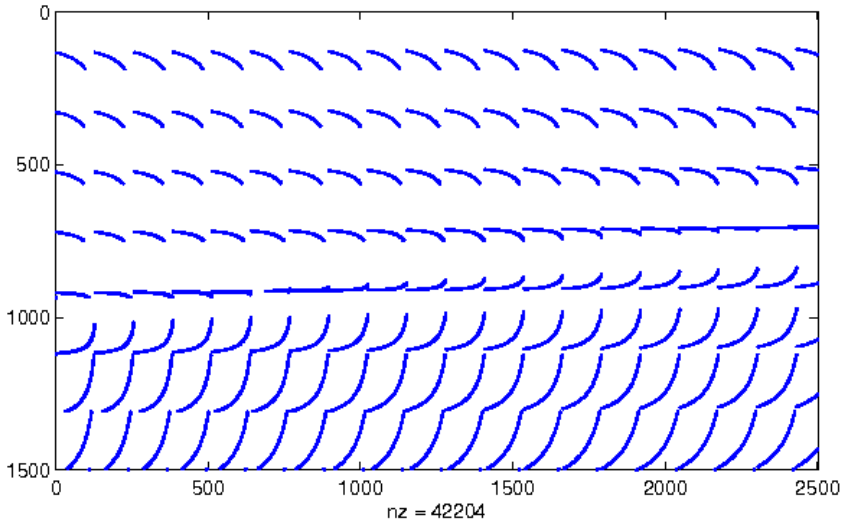

Figura 5.10: La matriz **A** con 1500 filas y 2500 columnas.

Se puede concluir que, como se esperaba, la matriz del sistema **A** es una matriz de alta dispersidad y el uso de los patrones simétricos en la estructura de datos puede minimizar el tamaño de la matriz almacenada.

5.4.3. Estructura simétrica de bloques de la matriz del sistema

En este apartado describimos los resultados de la reconstrucción aprovechando los patrones simétricos en la estructura de datos de la matriz del sistema y el análisis respecto la generación de la matriz.

Durante la adquisisción de datos en la tomografía axial computarizada, un escáner da la vuelta en el rango 0 - 360 grados como se muestra en la Figura 5.11. En este rango existen bloques simétricos en la estructura de datos (como se ve en la Figura 5.8) lo que permite reducir el número de vistas en la reconstrucción de la matriz y de esta forma reducir el tamaño de la memoria utilizada por el sistema.

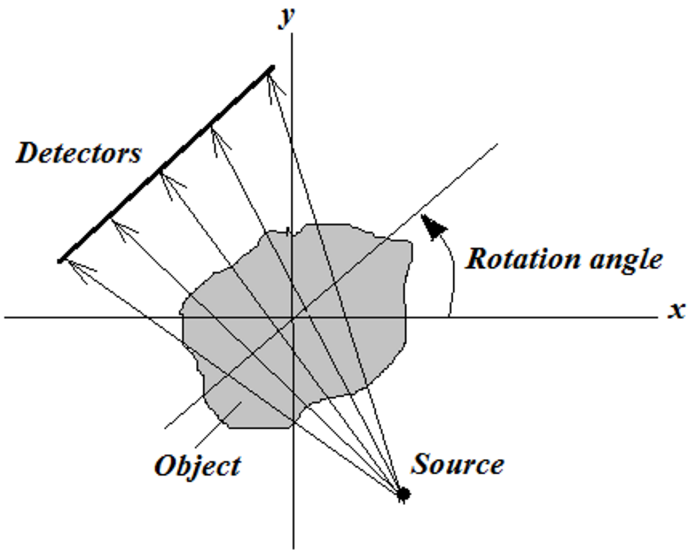

Figura 5.11: El proceso de adquisición de proyecciones.

Las simetrías por bloques se muestran en una visión general en la Figura 5.8 y más detallado en las figuras 5.12 - 5.14.

Suponemos que la posición inicial de la fuente está en $(0, \pi/2)$. Entonces, en el rango 90-180, la simetría es respecto a la recta $y = -x$, en el rango 90-270, respecto al eje x, y en el rango 90-450, respecto al eje y.

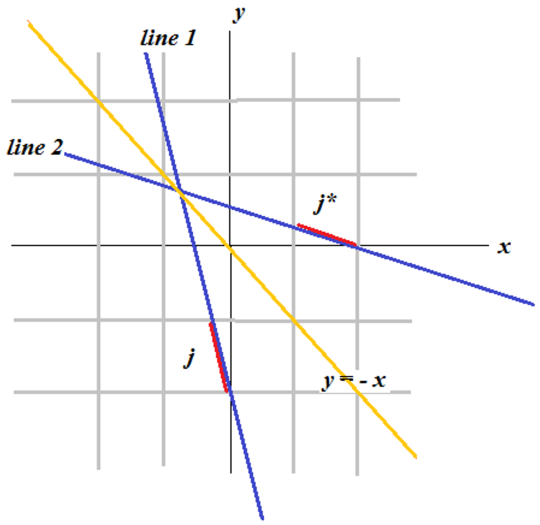

Figura 5.12: La simetría de proyecciones en el rango 90-180 es respecto a la recta $y = -x$.

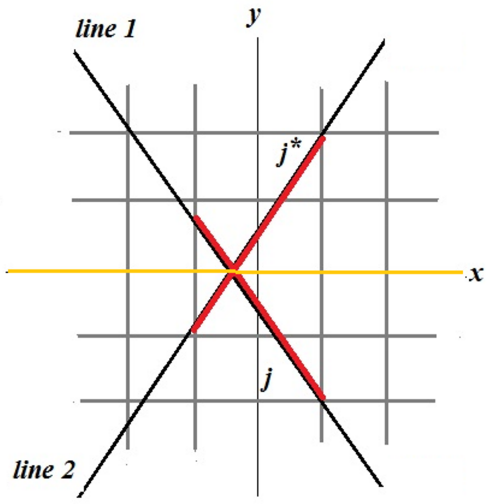

Figura 5.13: La simetría de proyecciones en el rango 90-270 es respecto al eje x.

Esto significa que la matriz del sistema puede ser generada en 1/8 parte de la circunferencia (en el rango de 0-45 grados, por ejemplo de 90 a 135) y, durante el proceso de reconstrucción, extendida a otros rangos.

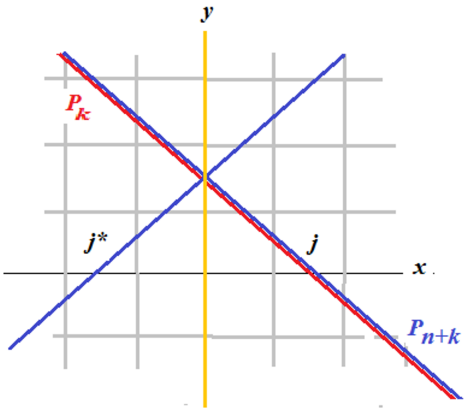

Figura 5.14: La simetría de proyecciones en el rango 90-450 es respecto al eje y.

La relación entre los elementos de la matriz.

Sea $X = \{ x_1 ... x_N \}$ - es un vector imagen con N=nxn. Para un píxel j la relación con el píxel $j*$ es la siguiente:

- Rango 90-180 (0-90): las filas y columnas se intercambian
  $fila = int(j, n)$
  $columna = mod(j, n)$
  entonces el píxel $j*$ de la proyección 'line2' es (ver Figura 5.12) :
  $j* = columna * n + fila$

- Rango 180-270 (ver Figura 5.13)
  $fila = int(j, n)$
  $columna = mod(j, n)$
  $j* = (n - row) * n + columna$

- Rango 270 - 450
  Las proyecciones en este rango coinciden con las proyecciones del rango 90-270,

por ejemplo, la proyeccion $P_{n+k} = P_k$. También se puede ver como paralelos respecto eje y (Figura 5.14).

$fila = int(j, n)$
$columna = mod(j, n)$
$j* = fila * n + (n - 1) - columna$
Utilizando esta relación se puede obtener la matriz correspondiente a este rango.

Utilizando las simetrías descritas en el párrafo, se puede generar las matrices en diferentes rangos previamente teniendo generada la matriz básica en el rango 90-135.

Algoritmo 5.4.4: GenerarRango90-180()
Entrada: $MatrizBasica_{90-135}$
Salida: $Matriz_{90-180}$

01: leer $MatrizBasica_{90-135}$
02: $totalM0 = total0$ // elementos correspondientes a 90 grados
03: $k = total$ elementos en $Matriz_{90-135}$
03: Completar el rango 135-180
04: $j = 1$
05: $r1 = rows(total - 1)$
06: for $i = total - 1 : -1 : totalM0$
07: if $(rows(i) = r1$
08: $rows(k) = rows(total - 1) + j$ // nueva fila
09: end if
10: else $j = j + 1$
11: $rows(k) = rows(total - 1) + j$ //nueva fila
12: $r1 = rows(i)$
13: end else
14: // nueva columna
15: rowtemp = columns(i) / n
16: coltemp = columns(i) % n

17: columns(k) = coltemp * n + rowtemp
18: $values(k) = values(i)$ nuevo valor
19: k = k + 1
20: end for

Los algoritmos que generan las matrices en los rangos 90-180 y 90-350 son similares al Algoritmo 5.4.4, excepto la fila 17.

Algoritmo 5.4.5: GenerarRango90-270()
Entrada: $MatrizBasica_{90-135}$
Salida: $Matriz_{90-270}$

17: columns(k) = ((n - 1) - rowtemp) * n + colmntemp

Algoritmo 5.4.6: GenerarRango90-350()
Entrada: $Matriz_{90-135}$
Salida: $Matriz_{90-450}$

17: columns(k) = rowtemp * n + (n-1) - columntemp

En la Figura 5.15 se presentan imágenes reconstruidas con las matrices generadas en diferentes rangos. Como se observa, la imagen reconstruida con 1/8 parte de la matriz del sistema (en el rango 0-45) es idéntica a la imagen reconstruida con la matriz generada en el rango completo (0-360). La utilización de 1/8 parte de la matriz permite reducir el uso de memoria del sistema.

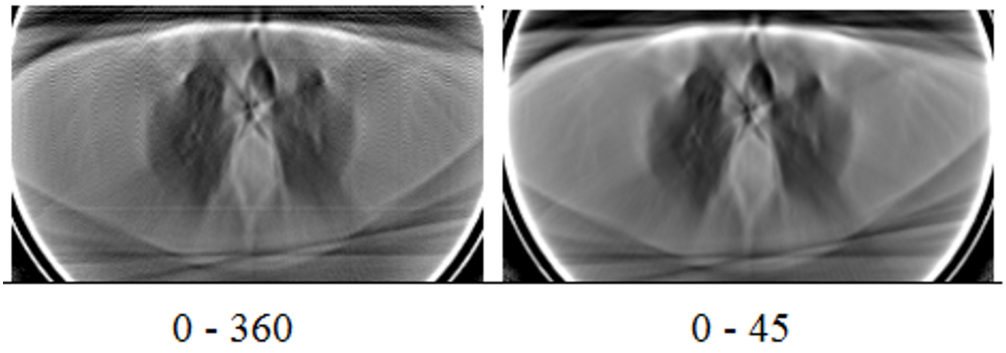

Figura 5.15: Las reconstrucciones con las matrices generadas en los rangos: 0-360 y 0-45 grados.

5.4.4. Tiempo de generación y tamaño de la matriz del sistema

Utilizando la simetría de la estructura de datos, se puede ahorrar el uso de memoria del sistema y, también disminuir el tiempo de reconstrucción debido a que leer una matriz completa es más costoso que utilizar la matriz mínima (en el rango 0-45) para completarla internamente. La comparativa de estas magnitudes para una matriz del sistema en el formato COO con 256 detectores y 50 vistas se resume en la tabla 5.1.

Rango de Matriz	*Tamaño*	*Tiempo de generación*	*Tiempo de reconstrución*
0 - 360	154 MB	1379 sec	2.25 msec
0 - 180	77 MB	680 sec	1.13 msec
0 - 90	39 MB	340 sec	0.68 msec

Tabla 5.1: Las características de las matrices generadas en diferentes rangos

5.4.5. Simulación de proyecciones

La reducción del número de vistas durante el escaneo convierte el problema de reconstrucción de imágenes (5.2) en un problema mal condicionado. En consecuencia, en la imagen reconstruida aparecen artefactos. La calidad de la imagen reconstruida

depende mucho de la matriz del sistema que simula el proceso de adquisición de datos.

El valor de la intensidad de un rayo i que va desde una fuente S de rayos-X se registra por un detector p_i. Este rayo atraviesa un píxel j como se muestra en la Figura 5.16 a).

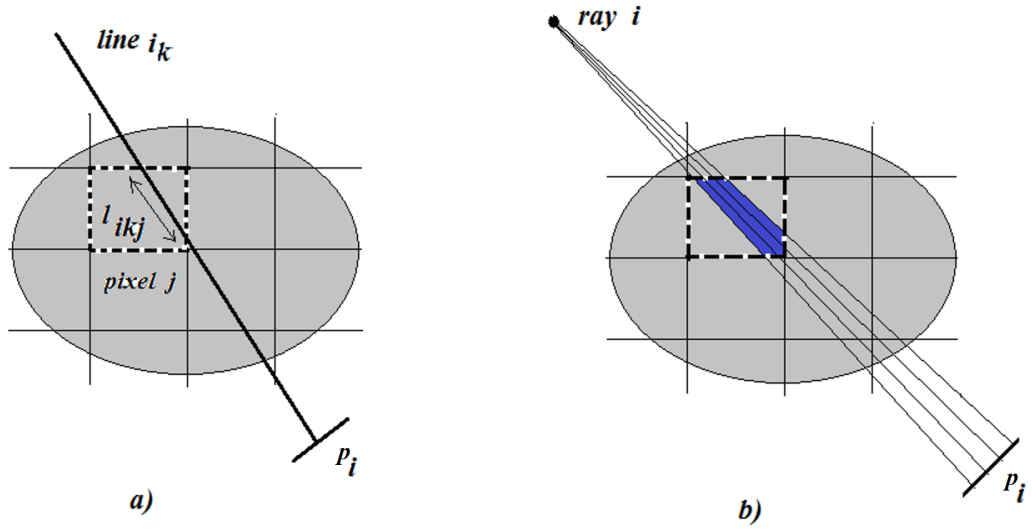

Figura 5.16: Simulación de rayos X registrados por un detector.

Los elementos de la matriz representan la proporción del rayo i que atraviesa el píxel j. Este rayo se puede representar por una línea. Pero esto va llevar a una simulación muy aproximada. La forma más adecuada, es representar un rayo como un conjunto de líneas y construir la matriz más adecuada. El área intersectada por un rayo y el píxel da el elemento correspondiente de la matriz. La simulación se presenta en la Figura 5.16(b).

Un elemento a_{ij} de la matriz A puede ser expresado por la ecuación 5.17

$$a_{ij} = \sum_{k=1}^{n} l_{ikj} \tag{5.17}$$

donde n es el número de líneas por rayo.

Llegados a este punto podríamos preguntarnos ¿cuántas líneas deben ser tomadas para simular un rayo? Para este estudio heurístico, hemos utilizado el siguiente

experimento. Hemos hecho simulaciones con diferente número de líneas por detector y analizado cómo se comporta la norma del error $||p-Ax||$. En nuestro experimento, la longitud del panel de detectores era igual a 50 cm, el número de detectores 256. Entonces la longitud de un detector era 1.9 cm. La Figura 5.17 muestra como varía el error en función de número de líneas por detector. Se puede concluir que el error se estabiliza para 20 líneas por detector, ó 5 líneas por mm de detector.

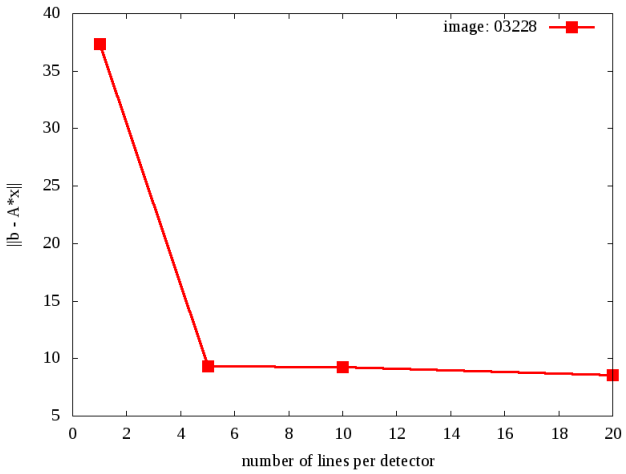

Figura 5.17: El error en la reconstrucción como función de número de líneas por detector.

Para confirmar esta conclusión, hemos generado las matrices con diferente número de líneas por un detector y en un rango limitado de 0 - 180 grados. En la Figura 5.18 se presentan los resultados de reconstrucción con estas simulaciones.

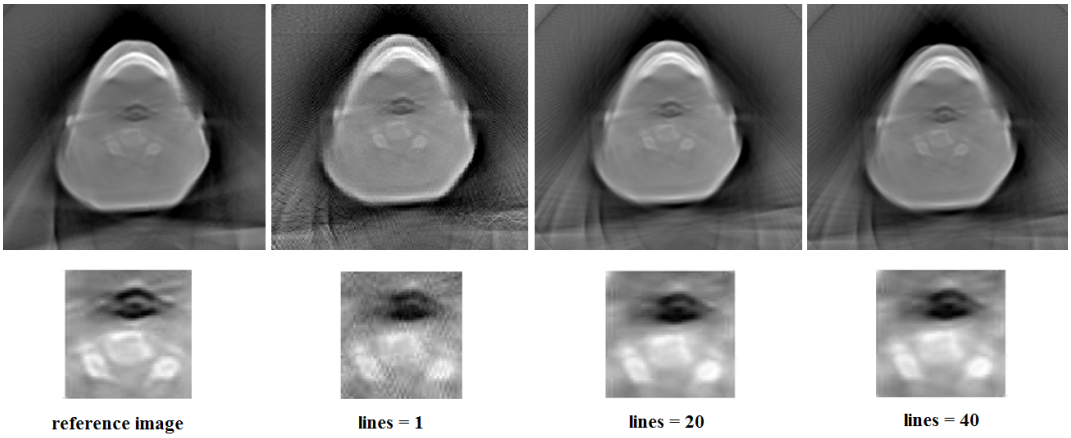

Figura 5.18: Las reconstrucciones con las matrices simuladas con 1, 20 y 40 líneas en el rango 0 - 180 grados y sus vistas aumentadas.

Observando estas imágenes se ve que hay diferencia entre la segunda y la tercera imagen, y no se observa diferencia entre las imagenes reconstruidas con 20 y 40 líneas.

5.4.6. Conclusión

La matriz del sistema 5.2 simula el proceso de adquisición de datos durante el escaneo de un paciente y afecta la calidad de la imagen a reconstruir.
Un rayo X se puede simular con un conjunto de líneas, tomando 5 líneas por mm de detector para lograr mayor calidad en la imagen reconstruida.

Para resolver el problema de reconstrucción es suficiente generar la matriz del sistema en el rango 0-45 y extenderla a otros rangos (0-180, 0-360) durante el proceso de reconstrucción. Esto permite utilizar recursos del sistema de forma más eficiente.

5.5. La técnica de reconstrución algebraica simultánea

El método algebraico para resolver el problema de reconstrucción se reduce al sistema lineal de ecuaciones:

$$Ax = p, \tag{5.18}$$

donde la matriz del sistema A simula el procesamiento de la tomografía axial computerizada y sus coeficientes representan la proporción del rayo i que atraviesa el píxel j de la imagen; x es la matriz columna cuyos elementos representan las intensidades de la imagen a reconstruir; la matriz columna p representa las proyecciones recolectadas por el escáner. En esta técnica, el problema de reconstrucción de la estructura interna de un objeto se reduce a la resolución del sistema (5.18) en términos de las proyecciones medidas.

La técnica de reconstrución algebraica simultanea (SART) se considera como un algoritmo de reconstrucción clásico en la tomografía computarizada. Existen muchas formulaciones del método. La iteración en SART que hemos empleado es como sigue ([55]):

$$x^{k+1} = x^k + \lambda^k \frac{1}{A_i^+} \sum_{i=1}^{M} \frac{a_{ij}}{A_j^+} r_i, \tag{5.19}$$

donde x^{k+1}, x^k representan el vector imagen en la iteración $k+1$ y k respectivamente, A_i denota la fila i-ésima de la matriz A, $r_i = p_i - A_i x^k$ representa la componente i-ésima del vector residuo, p_i denota la componente i-ésima de vector de las proyecciones, A_i^+ y A_j^+ denotan las sumas por filas y por columnas de los elementos de la matriz A respectivamente.

El proceso empieza con una aproximación inicial de x, generalmente igual a cero, y se repite hasta safisfacer un criterio de parada. El parámetro λ toma valor entre 0 y 1, y representa el término regularizador de la solución. El SART es fácilmente adaptable a las peculiaridades de cada una de las matrices del sistema que se va a calcular y la incorporacion en el algoritmo de información 'a priori' es muy sencilla. Es un algoritmo que ha demostrado ser muy robusto numéricamente [55].

Analizando la ecuación (5.19), se puede notar que hay partes de cómputo que se puede realizar de forma independiente de las iteraciones. Estas operaciones son: cálculo de suma de elementos de filas y columnas de la matriz A (A_i^+, A_j^+), normalización de elementos de la matriz A (a_{ij}/A_j^+). A continuación, se presentan los algoritmos que realizan estas operaciones.

5.5.1. SART: versión secuencial

Algoritmo 5.5.1: sumarFilas() - Suma de elementos de una fila de A

- **Entrada**: matriz A en COO
- **Salida**: vector sumaFilas

```
01:  for j = 1 : N
02:    suma = 0
03:    for k = 1 : NNZ
04:      if cols[k] == j
05:        suma ⇐ suma + values[k]
06:      endif
07:    endfor
08:    sumaFilas[j] ⇐ suma
09:  end for
```

Algoritmo 5.5.2: sumarColumnas() - Suma de elementos de una columna de A

- **Entrada**: matriz A en COO
- **Salida**: vector sumaColumnas

```
01:  for i = 1 : M
02:    suma = 0
03:    for k = 1 : NNZ
04:      if rows[k] == i
05:        suma ⇐ suma + values[k]
06:      endif
07:    endfor
08:    sumaColumnas[i] ⇐ suma
09:  end for
```

Algoritmo 5.5.3: calcularValues2()- cálculo de a_{ij}/A_i^+ - coeficientes normalizados de A

- **Entrada**: matriz A en COO (rows, values)
- **Salida**: vector values2 []

01: for $k = 1 : NNZ$
01: ro = rows[k]
02: values2[k] $\Leftarrow$ values[k] / sumaColumnas[ro]
03: endfor

Algoritmo 5.5.4: calcularProducto Matriz-Vector()

- **Entrada**: matriz A en COO , vector x
- **Salida**: vector V - resultado del producto

01: for $k = 1 : NNZ$
02: $ro = rows[k]$
03: $co = cols[k]$
04: $V[ro] \Leftarrow values[k] * x[co]$
05: end for

La parte $\sum_{i=1}^{M} \frac{a_{ij}}{A_j^+} r_i$ de la ecuación (5.19) se puede calcular de dos formas. Una forma es directa y se presenta en la función **calcularNumerador1()**. La otra forma presentada en **calcularNumerador2()**, consiste en utilizar los coeficientes de A normalizados (**values2**) y precalculados previamente.

Algoritmo 5.5.5: calcularNumerador1() - para j dado

- **Entrada**: valor j, vector V, matriz en COO: rows, cols, values
- **Salida**: numerador

01: $numerador = 0$
02: for $k = 1 : NNZ$
03: if $(cols[k] == j)$
04: ro = rows[k]
05: $numerador \Leftarrow values[k] * (p[ro] - V(ro))/sumaColumnas[ro]$

06: endif
07: endfor
08: return *numerador*

Algoritmo 5.5.6: calcularNumerador2() - para j dado

- **Entrada**: valor j, vector V, rows, cols, values2
- **Salida**: numerador

01: $numerador = 0$
02: for $k = 1 : NNZ$
03: if $(cols[k] == j)$
04: ro = rows[k]
05: $numerador \Leftarrow$ **values2[k]** $* (p[ro] - V[ro])$
06: endif
07: endfor
08: return *numerador*

Algoritmo 5.5.7: updateX()- actualiza los valores de x

- **Entrada**: valor j
- **Salida**: valor x(j) actualizado

01: x(j)$\Leftarrow$ x[j] + λ^k * numerador / sumaFilas [j]

Dos formas diferentes de calcular el numerador en (5.19)(**calcularNumerador1()** y **calcularNumerador2()**) dan lugar a dos versiones de SART.

Algoritmo 5.5.8: SART - versión 1

- **Entrada**: Matriz A_{MxN} en COO, vector p_{1xM} - vector de proyecciones.
- **Salida**: Vector imagen x_{1xN}.

/** Inicialización
01: $x = 0; \lambda = 0{,}1;$

02: **sumarFilas()**
03: **sumarColumnas()**
04: for $iter = 1 : iterMax$
05: **calcularProducto Matriz-vector()**
06: for $j = 1 : N$
07: numerador $\Leftarrow$ **calcularNumerador1(j)**
08: updateX(j)
09: endfor
10: endfor

Algoritmo 5.5.9: SART - versión 2

/** Inicialización
01: $x = 0; \lambda = 0{,}1;$
02: **sumarFilas()**
03: **sumarColumnas()**
04: **calcularValues2()**
05: for $iter = 1 : iterMax$
06: **calcularProducto Matriz-vector()**
07: for $j = 1 : N$
08: numerador $\Leftarrow$ **calcularNumerador2(j)**
09: updateX(j)
10: endfor
11: endfor

5.5.2. SART - versión paralela

Las operaciones en un proceso de reconstrucción de una imagen son operaciones de píxel, independientes de otros píxeles. La versión paralela de SART se presenta para la arquitectura multi-core utilizando el modelo de programación OpenMP. En la iteración k, un hilo trabaja con un solo píxel de la imagen a reconstruir. Cuando todos los hilos terminan su trabajo, empieza la siguiente iteración.

Algoritmo 5.5.10: SART: versión paralela 1

- **Entrada:** Matriz A_{MxN} en COO, vector b_{1xM} - vector de proyecciones.
- **Salida:** Vector x_{1xN} cuyos valores representan las intensidades de la imagen reconstruida.

```
/** Inicialización
01:  x = 0; λ = 0,1;
02:  sumarFilas()
03:  sumarColumnas()
04:  for iter = 1 : iterMax
05:      calcularProducto Matriz-vector()
06:      # pragma omp parallel for
07:      for j = 1 : N
08:          numerador ⇐ calcularNumerador1(j)
09:          updateX(j)
10:      endfor
11:  endfor
```

Algoritmo 5.5.11: SART: versión paralela 2

```
/** Inicialización
01:  x = 0; λ = 0,1;
02:  sumarFilas()
03:  sumarColumnas()
04:  calcularValues2()
05:  for iter = 1 : iterMax
06:      calcularProducto Matriz-vector()
07:      # pragma omp parallel for
08:      for j = 1 : N
09:          numerador ⇐ calcularNumerador2(j)
10:          updateX(j)
11:      endfor
```

12: endfor

5.5.3. Evaluación de coste

A continuación, se evalúa el coste del algoritmo SART para un sistema con arquitectura multi-core.
La iteración SART dada por la ecuación (5.19), se puede reescribir en la forma siguiente:

$$x_j^{k+1} = x_j^k + \lambda^k \frac{1}{funcion3} \sum_{i=1}^{M} \frac{a_{ij}}{funcion2}(p_i - funcion1) \tag{5.20}$$

donde $j = 1...N$ es el número de píxeles en la imagen, M es el número de filas en la matriz A. Sea p el número de procesos, NNZ es el número de elementos no ceros en la matriz A en formato COO, NNZ_i y NNZ_j corresponden con el número de elementos no ceros en la fila i y columna j de la matriz A. Tomando en cuenta que $NNZ \gg M$ y $NNZ \gg N$ evaluaremos los costes de **funcion1**, **funcion2** y **funcion3**.

Coste de la funcion1 (algoritmo 5.5.1) - costeF1

$$costeF1 = \sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij}x \Rightarrow \sum_{i=1}^{M} 2NNZ_i = 2NNZ \tag{5.21}$$

Coste de la funcion2 (algoritmo 5.5.2) - costeF2

$$costeF2 = \sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij} => \sum_{i=1}^{M} NNZ_i = NNZ \tag{5.22}$$

Coste de la funcion3 (algoritmo 5.5.3) - costeF3

$$costeF3 = \sum_{i=1}^{M} a_{ij} => NNZ_j \tag{5.23}$$

- Tiempo secuencial:
  $T_1 = \sum_{j=1}^{N} (NNZ_j + 2NNZ + NNZ) \approx 4NNZ$

- Tiempo paralelo:
  $T_p = \sum_{j=1}^{N/p}(NNZ_j + 2NNZ + NNZ) \approx \frac{4NNZ}{p}$

- SpeedUp límite teórico del sistema: $S = \lim_{N\to\text{ínf}} \frac{T_1}{T_p} = p$

- Eficiencia límite teórica del sistema: $E = \lim_{N\to\text{ínf}} \frac{S}{p} = 1$

5.5.4. Resultados experimentales

En este apartado presentamos algúnos resultados experimentales al comparar dos versiones de SART (versión paralela 1 y 2, como versiones más eficientes). Los algoritmos se han ejecutado en el cluster Kahan. Para una imagen de 256x256 píxeles reconstruida con 50 proyecciones, en la Tabla 5.2 se resume el tiempo (en segundos) por una iteración utilizando diferente número de hilos OpenMP.

# hilos OpenMP	SART versión 2	SART versión 1
1	3612	3703
2	1851	1899
4	504	515
16	303	305
24	219	221
32	170	169
64	220	215

Tabla 5.2: Tiempo de reconstrucción (en segundos) por una iteración de SART: versión 1 y 2 en función de número de procesos.

Se observa que la versión 2 de SART requiere un poco menos tiempo. Sin embargo, incluso una sola iteración de SART es bastante costosa.
La eficiencia de las versiones 1 y 2 de SART se presenta en las figuras 5.19 y 5.20. Se observa que los algoritmos son más eficientes cuando se ejecutan en un sólo nodo con total de 32 hilos. En estas condiciones se crece el SpeedUp del sistema y la

eficiencia se mantiene al nivel de 70 por ciento. Al aumentar el número de procesos las prestaciones se empeoran. Este resultado se debe al elevado coste de transmición de gran cantidad de datos entre nodos.

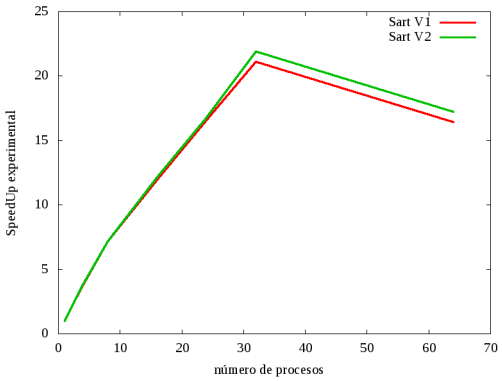

Figura 5.19: Comparación de Speed Up.

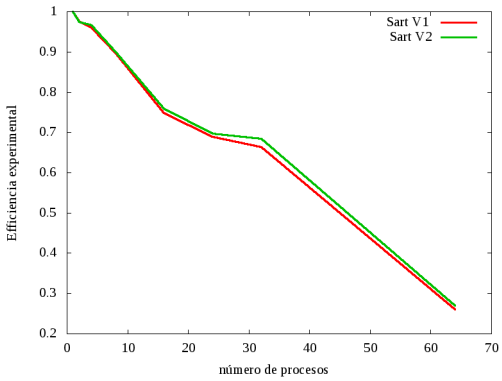

Figura 5.20: Comparación de Eficiencia.

Para obtener cierta calidad en la imagen se requieren muchas más iteraciones como se observa en la Figura 5.27.

5.5.5. Conclusiones

El algoritmo SART en COO es muy costoso. Las operaciones de cómputo utilizadas en el proceso de reconstrucción son de tipo píxel-vóxel, tienen pocas dependencias y son ejecutadas en bucles grandes. Las plataformas apropiadas para este tipo de operaciones son procesadores vectoriales o arquitecturas masivamente paralelas como unidades de procesamiento gráfico (GPUs). La implementación de SART en GPUs se presenta en el capítulo 6.

5.6. MLEM - algoritmo de maximum likelihood para la tomografía de emisión

El algoritmo 'Maximum Likelihood Expectation Maximization' (MLEM) es un método de reconstrucción de imágenes para la tomografía de emisión. En el MLEM, primero se define la función objetivo y después, la función se optimiza [56]. Se considera que el ruido en las proyecciones sigue la distribución de Poisson.

MLEM busca la solución (imagen) óptima según los datos experimentales. La idea básica de este algoritmo es como sigue: dado un conjunto de datos experimentales b, encontrar la distribución del coeficiente lineal de atenuación μ que maximiza la probabilidad $P(\mu/b)$.
Las iteraciones en MLEM se definen de la siguiente forma:

$$x_j^{k+1} = x_j^k \sum_{i=1}^{M} \frac{p_i a_{ij}}{\sum_{j=1}^{N} x_{ij}^k a_{ij}}, \tag{5.24}$$

donde $x = \{ x_j | j = 1, 2...N \}$ es el vector imagen de N píxeles, $p = \{ p_i | i = 1, 2...M \}$ representa las proyecciones, $A = \{ a_{ij} \}$ es la matriz del sistema de MxN, cuyos elementos dan la proporción del rayo i a través del píxel j.

5.6.1. MLEM: versión secuencial

Algoritmo 5.6.1: calcularDenominador() - producto de fila i-esima de matriz por vector: $A_i * x$ para $i = 1 : M$

- **Entrada: matriz A en COO, vector x**
- **Salida: vector V[]**

calcularDenominador() $\Longrightarrow$ Algoritmo 5.5.4

Algoritmo 5.6.2: calcularNumerador () - cálculo de numerador para j dado

- **Entrada**: valor j, matriz A en COO, vector p, vector V[]
- **Salida**: numerador

01: $numerador = 0$
05: for $k = 1 : NNZ$
06: if $(cols[k] == j)$
07: $ro = rows[k]$
09: $numerador+ = p[ro] * values[k]/V[ro]$
12: endif
13: endfor

15: return *numerador*

Algoritmo 5.6.3: updateX ()

- **Entrada**: valor j
- **Salida**: valor $x(j)$ actualizado

01: x(j) = x(j)* **calcularNumerador(j)**

Algoritmo 5.6.4: MLEM - versión completa

- **Entrada:** Matriz A_{MxN} en COO, vector p_{1xM} - vector de proyecciones.
- **Salida:** Vector x_{1xN} - la imagen reconstruida.

/** Inicialización
01: x $\neq$ 0;
02: for iter = 1:iterMax
03: **calcularDenominador()**
04: for $j = 1 : N$
05: **calcularNumerador(j)**
06: **updateX(j)**
07: endfor
08: endfor

5.6.2. MLEM: versión paralela

La primera versión paralela de MLEM se realiza en base del **Algoritmo 5.6.4**. El bucle más grande ($j = 1 : NNZ$) se reparte entre los hilos OpenMP para acelerar los cálculos.

Algoritmo 5.6.5: MLEM: versión 1
/** Inicialización
01: x $\neq$ 0;

02: for $iter = 1 : iterMax$
03: **calcularDenominador()**
04: # pragma omp parallel for
05: for $j = 1 : N$
06: **calcularNumerador(j)**
07: updateX(j)
08: endfor
09: endfor

La normalización de los coeficientes de la matriz A se presenta en el siguiente algoritmo.

Algoritmo 5.6.5: calcularValues2

calcularValues2() {
01: # pragma omp parallel for
02: for $k = 1 : NNZ$
03: ro = rows[k]
04: values2[k] = values[k] * b[ro] / V[ro]
05: endfor

El cálculo de los coeficientes normalizados de la matriz A fuera del bucle j da lugar a la segunda versión de MLEM.

Algoritmo 5.6.6: MLEM: versión 2

/** Inicialización
01: x $\neq$ 0;
02: for iter = 1:iterMax
03: **calcularDenominador()** $\Rightarrow$ V[]
04: **calcularValues2()**
05: for $j = 1 : N$
06: suma = 0
07: for $i = 1 : NNZ$

```
08:              if ( cols[i] == j )
09:                  suma += values2[i]
10:              endif
11:          endfor
12:          X[j] *= suma
13:      endfor
14:  endfor
```

5.6.3. Evaluación de prestaciones

Para comparar las versiones 1 y 2 del algoritmo MLEM, los algoritmos se ejecutaron utilizando diferente número procesos en el cluster Kahan.
El tiempo de ejecución (en segundos) que corresponde a una matriz /textbfA del sitema 5.2 con 256x133 filas y 256x256 columnas se resume en la tabla 5.3. Se observa

# hilos OpenMP	MLEM versión 1	MLEM versión 2
1	4312	4892
2	2202	2431
4	1137	1258
8	630	680
16	366	398
24	250	281
32	192	215
40	335	338

Tabla 5.3: Tiempo de reconstrucción (en segundos) por una iteración de MLEM: versión 1 y 2.

que el tiempo no varía mucho para las versiones 1 y 2. Al aumentar el número de procesadores se puede reducir significativamente el tiempo de resolución del sistema cuando el algoritmo se ejecuta en un solo nodo. Al ejecutar el algoritmo en varios nodos, se reduce el SpeedUp y disminuye la eficiencia del sistema debido a la gran cantidad de datos y en consecuencia, alto coste de comunicación entre nodos. La

variación de SpeedUp y de la Eficiencia experimentales del algoritmo se presentan en la Figura 5.21. Se puede concluir que la solución óptima para este caso se logra utilizando todos los procesadores (32) en un sólo nodo de cluster .

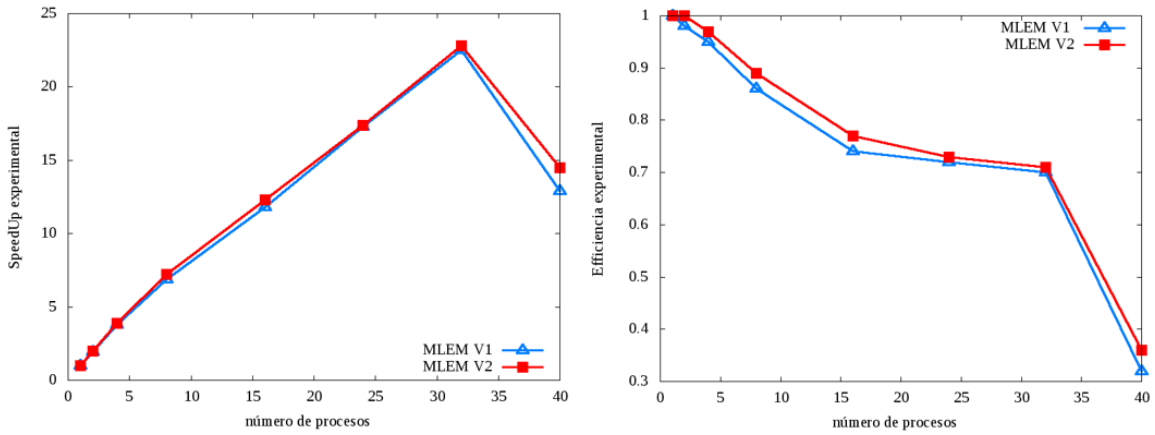

Figura 5.21: SpeedUp (a) y Eficiencia (b) experimentales de MLEM en función de número de hilos OpenMP.

5.6.4. Conclusiones

El algoritmo iterativo MLEM tiene un alto coste computacional. El problema de reconstrucción de imágenes es un problema de gran cantidad de datos y las plataformas más apropiadas para utilizar MLEM en una forma eficiente son plataformas 'many-cores'.

5.7. Método iterativo LSQR

El método para resolver un sistema de ecuaciones lineales disperso (LSQR) [57], es un método iterativo dentro de un conjunto de métodos de Krylov que resuelve un sistema de ecuaciones de la forma:

$$Ax = p \tag{5.25}$$

y minimiza :

$$min \left\| Ax - p \right\|_2 \tag{5.26}$$

En (5.25) A es una matriz real no simétrica de gran dimensión y dispersa con m filas y n columnas, p es un vector real. Es común que $m \gg n$ y el $rank(A) = n$, pero esto no es esencial. El método es similar al bien conocido método del Gradiente Conjugado (CG). La matriz A sólo se usa para calcular productos del tipo Av y $A^T u$ para varios vectores v y u.

La resolución del sistema de ecuaciones (5.25) es equivalente a resolver el sistema:

$$A^T Ax = A^T p. \tag{5.27}$$

En (5.27) $A^T A$ es simétrica definida positiva. La ecuación (5.27) recibe el nombre de Ecuación Normal.

El método LSQR está basado en el proceso de bidiagonalización de Golub y Kahan [58]. La idea básica del LSQR es hallar la solución del sistema simétrico:

$$\begin{pmatrix} I & A \\ A^T & -\lambda^2 I \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} p \\ 0 \end{pmatrix} \tag{5.28}$$

minimizando la expresión:

$$\left\| \begin{pmatrix} A \\ \lambda I \end{pmatrix} x - \begin{pmatrix} p \\ 0 \end{pmatrix} \right\|_2, \tag{5.29}$$

donde λ es un número arbitrario.

El método genera una secuencia de aproximaciones $\{x_k\}$ de tal forma que la norma residual $\|r_k\|_2$ decrece de forma monótona, donde $r_k = p - Ax_k$. Analíticamente, la secuencia $\{x_k\}$ es idéntica a la secuencia que se genera con el algoritmo estándar CG. Sin embargo, el método LSQR es numéricamente más fiable en la mayoría de los casos.

5.7.1. LSQR - versión secuencial

Sea A una matriz de dimensión MxN, p es el vector que representa el término independiente en la ecuación (5.25), $\alpha_i \geq 0$, $\beta_i \geq 0$ son escalares que se escogen

para normalizar los vectores correspondientes v y u.

Algoritmo 5.7.1: LSQR ()

- **Entrada:** Matriz A_{MxN} - matriz del sistema, vector p_M - vector de proyecciones.
- **Salida:** Vector x_N cuyos valores representan las intensidades de la imagen reconstruida.

Inicialización:

01: $\beta_1 u_1 = p$

02: $\alpha_1 u_1 = A^T u_1$

03: $w_1 = v_1$

04: $x_0 = 0$

05: $\overline{\phi} = \beta_1$

06: $\overline{\rho} = \alpha_1$

Proceso de bidiagonalización

07: for $i = 1 : iter_{max}$

08: $\beta_{i+1} u_{i+1} = A v_i - \alpha_i u_i$

09: $\alpha_{i+1} v_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$

Construir y aplicar la siguente transformación ortogonal:

10: $\rho_i = (\overline{\rho}_i^2 + \beta_{i+1}^2)^{1/2}$

11: $c_i = \overline{\rho}_i / \rho_i$

12: $s_i = \beta_{i+1} / \rho_i$

13: $\Theta_{i+1} = s_i \alpha_{i+1}$

14: $\overline{\rho}_{i+1} = -c_i \alpha_{i+1}$

15: $\phi_i = c_i \overline{\phi}_i$

16: $\overline{\phi}_{i+1} = s_i \overline{\phi}_i$

Actualización de los vectores x y w

17: $x_i = x_{i-1} + (\phi_i / \rho_i) w_i$

18: $w_{i+1} = v_{i+1} - (\Theta_{i+1} / \rho_i) w_i$

Aplicar test de convergencia

10: if ($Convergencia == True$)

20: STOP

21: end if

22: end for

Coste de LSQR.

El algoritmo LSQR es un algoritmo iterativo, en cada iteración las operaciones más costosas se corresponden con productos matriz por vector Av y $A^T u$.
El coste por iteración de los productos Av y $A^T u$ puede ser expresado en términos de elementos no nulos NNZ. Para estos productos, el número total de operaciones en coma flotante ejecutadas en forma secuencial por un procesador es:

- $T_1 = 2 * NNZ$.

Entonces, el coste de LSQR puede ser expresado como :

- $T_1 = 4 * NNZ$ operaciones en coma flotante.

5.7.2. LSQR: versión paralela

En este apartado se presenta una implementación paralela del algoritmo LSQR utilizando la librería PETSc. Hemos empleado esta librería para resolver el sistema (5.25). La librería ofrece varios 'solvers' y permite incorporar o modificar códigos para su uso en las distintas aplicaciones. La mayor parte del tiempo de computación se gasta en el ensamblaje de la matriz del sistema **A**. En nuestros experimentos, la matriz se almacena en la forma compacta (formato coordenado COO) para matrices dispersas. Es decir, sólo se almacenan el número de fila, número de columna y el valor del elemento no nulo de la matriz. El proceso de ensamblaje y la resolución del sistema se paraleliza para lograr mejor rendimiento.

La implementación de LSQR en PETSc se resume en el siguiente pseudocódigo:

- **Entrada.**
  - Matriz A_{NNZx3} en el formato COO - salida del algoritmo de Siddon, NNZ - número elementos no nulos.

- Vector-columna p_{Mx1} - proyecciones registradas por el escáner.

- **Salida**

  Matriz X_{Nx1} cuyos valores representan intensidades de la imagen reconstruida.

Etapas del algoritmo:

- **Inicialización**
  - Inicializar el sistema con PetscInitialize()
  - Leer los datos de entrada.
- **Ensamblaje de la matriz.**
  - La matriz del sistema se particiona por filas entre los procesos y se ensambla en forma paralela (**pasos 1:16 del Algoritmo 5.7.1**). Las filas que corresponden a un proceso p se definen por el rango $Istart$-$Iend$.
- **Resolución del sistema**
  - Se indican el método de resolución KSP, la tolerancia, y el número de iteraciones máximo (**pasos 17:20**). Estos parámentos se puede seleccionar en tiempo de ejecución (run time).
  - Se resuelve el sistema y se obtiene la información resultante de todo el proceso (**pasos 21:23**).
  - Se genera el archivo de salida con solución del sistema (**paso 24**)
- **Finalizar el proceso** (**paso 25**).

Los pasos principales de esta implementación se detallan en el siguiente código:

Algoritmo 5.7.2: LSQR-PETSc

- **Ensamblaje de la matriz.**

```
01: MatGetOwnershipRange(A,&Istart,&Iend)
02: for i = Istart : Iend
03:     ii = i
04:     for k = 1 : NNZ
05:         if rows[k] == ii
```

06: $jj = cols[k]$
07: $aa = values[k]$
08: $MatSetValues(A, 1, \&ii, 1, \&jj, \&aa, INSERT-VALUES)$
09: endif
10: if $rows[k] > ii$
11: $k = NNZ$
12: endif
13: endfor
14: endfor

/* Ensamblaje de la matriz en 2 etapas
15: $MatAssemblyBegin(A)$
16: $MatAssemblyEnd(A)$

- **Resolución del sistema**

/* Determinar el solver y tolerancia
17: $KSPSetOperators(ksp, A, A)$
20: $KSPSetTolerances(ksp, 1.e{-}5, PETSC{-}DEFAULT, PETSC{-}DEFAULT)$

/* Resolver el sistema
21: $KSPSolve(ksp, b, x)$
22: $KSPGetConvergedReason(ksp, \&reason)$

/* Imprimir la informacion de convergencia
23: $PetscPrintf("Normoferror, iterations")$

/* Escribir la solucion en un archivo de salida
24: $PetscViewerASCIIOpen(PETSC{-}COMM{-}WORLD, "x{-}values.txt", \&viewer)$
25: PetsFinalize()

Como criterio de parada del algoritmo se define el valor de tolerancia deseada, en

nuestro caso en la sentencia 20 hemos empleado $tol = 1e^{-5}$.

5.7.3. Evaluación de coste

Evaluaremos la parte más costosa del algoritmo - el ensamblaje de la matriz del sistema A, los pasos (1:16) del **Algoritmo 5.7.1**.

Denotamos por M al número de filas de A, por NNZ al número de elementos no nulos de la matriz de entrada generada por el método de Siddon ($NNZ \gg M$) y por p al número de procesos.

- Tiempo secuencial de ensamblaje:
  $T_1 = \sum_{i=1}^{M}(\sum_{k=1}^{NNZ} 1 + 1) = M(NNZ + 1) \approx NNZ$

- Tiempo paralelo de ensamblaje:
  $T_p = \sum_{i=1}^{\frac{M}{p}}(\sum_{k=1}^{NNZ} 1 + 1) = \frac{M}{p}(NNZ + 1) \approx \frac{NNZ}{p}$

- SpeedUp teórico: $S = \frac{T_1}{T_p} = p$

- SpeedUp límite: $\lim_{N \to \text{ínf}} S = p$

- Eficiencia del sistema: $E = \frac{S}{p} = 1, \qquad \lim_{N \to \text{ínf}} E = 1$

El sistema (5.25) se resuelve por el algoritmo iterativo LSQR. El coste por iteración de LSQR está dado en la parte 5.7.1 de la descripción del algoritmo.

5.7.4. Metodología de los experimentos realizados

En los experimentos se han usado fantomas Shepp-Logan de diferentes dimensiones generados en Matlab. Para el fantoma de 256x256 píxeles se usaron 369 sensores para simular proyecciones en Matlab. Se generaron las proyecciones con diferente número de ángulos con el objetivo de analizar la posibilidad de la reconstrucción de imagen con un menor número de proyecciones.

Las proyecciones simuladas representan el término independiente en el sistema (5.25). La matriz del sistema **A** se generó previamente por el algoritmo de Siddon. Una vez generada la matriz **A**, se emplea PETSc para obtener la solución del sistema (5.25). La solución representa una matriz nxn cuyos valores corresponden a las intensidades de la imagen incógnita.

5.7.5. Resultados experimentales.

Para el fantoma de 256x256 píxeles el tamaño resultante de la matiz **A** generada con diferente número de ángulos para los cuales las proyecciones fueron tomadas, está resumido en la Tabla 5.4. El número de proyecciones se obtiene multiplicando el número de ángulos por el número de detectores.

Número de proyecciones	Tamaño de A (MB)
120x369	283
90x369	272
60x369	181
40x369	120
36x369	107

Tabla 5.4: El tamaño de la matriz del sistema **A**.

El tiempo (en segundos) de ensamblaje y de resolución del sistema 5.25 con diferente número de procesadores p y proyecciones que corresponden a la imagen de 256x256 píxeles se resumen en las tablas 5.5 y 5.6.
Se observa que el tiempo de resolución del sistema no varía mucho al aumentar el número de procesadores hasta $p = 32$. Al mismo tiempo, el ensamblaje de la matriz del sistema consume mayor tiempo y se reduce notablemente al aumentar el número de procesadores. Por lo tanto, se puede concluir que la solución óptima para este caso se logra con 32 procesadores.

# Proyecciones	p = 1	p = 4	p = 8	p = 16	p = 32	p = 64
120x369	1190.0	523.1	286.5	200.9	174.8	89.1
90x369	669.6	294.9	157.9	82.9	58.1	49.8
60x369	297.2	131.6	71.1	51.4	43.9	23.1
40x369	132.1	58.1	31.7	22.2	19.1	9.8
36x369	102.2	47.2	25.6	17.0	16.0	8.5

Tabla 5.5: Tiempo de ensamblaje (en segundos) de la matriz **A** en función de número de procesadores.

# Proyecciones	p = 1	p = 4	p = 8	p = 16	p = 32	p = 64
120x369	0.9	0.4	0.6	0.7	1.0	5.6
90x369	0.7	0.4	0.3	0.5	0.7	5.8
60x369	0.4	0.2	0.3	0.4	0.6	5.4
40x369	0.3	0.2	0.2	0.3	0.5	3.7
36x369	0.3	0.1	0.2	0.3	0.5	3.6

Tabla 5.6: Tiempo de resolución del sistema 5.25 (en segundos) en función de número de procesadores.

La variación de la Eficiencia y SpeedUp experimentales en función del número de procesadores y del tamaño del problema (número de proyecciones N_p) se presenta en la Figura 5.22.

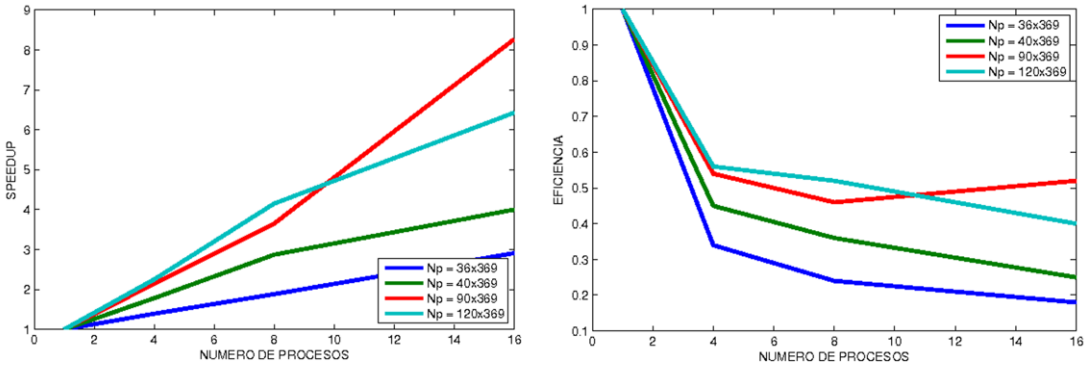

Figura 5.22: SpeedUp (a) y Eficiencia (b) en función de número de procesadores para diferentes tamaños del problema N_p.

5.7.6. Conclusiones

Analizando los resultados presentados en esta sección se puede concluir:

- La implementación paralela del algoritmo lleva a una reducción notable del tiempo de ejecución, sobre todo en la parte del Ensamblaje donde en el mejor de los casos se obtiene una reducción en 13.4 veces (para N_p = 120x369 y $p = 64$).

  Para el número de procesadores $p = 8$ y tamaño del problema N = 120x369

  - SpeedUp = $\frac{T_1}{T_{16}} = 6{,}43$
  - Eficiencia = $\frac{SpeedUp}{p} = 0{,}40$

- SpeedUp y Eficiencia aumentan con el aumento del número de proyecciones. Es decir, el algoritmo es más eficiente para problemas de tamaño grande.

5.8. Optimización de algoritmos iterativos de reconstrucción de imágenes sobre arquitecturas multi-core

Como hemos indicado en los capítulos anteriores, el problema de reconstrucción de imágenes es un problema algebraico de grandes dimensiones y su resolución tiene un alto coste computacional. La matriz del sistema que simula el proceso de escaneo es de gran dimensión, altamente dispersa y se usa principalmente para las operaciones siguientes:

- $\mathbf{A}v$- producto matriz por vector
- $\mathbf{A}^T v$ - producto matriz transpuesta por vector
- $\sum_{i=1}^{M} a_{ij}$ - suma de los elementos de las filas de A
- $\sum_{j=1}^{N} a_{ij}$ - suma de los elemntos de las columnas de A.

En el proceso de reconstrucción de imágenes, estas operaciones se ejecutan en bucles grandes y representan la mayor parte del coste computacional de los algoritmos iterativos. En el siguiente apartado, vamos a ver las formas de optimizar las operaciones principales de los algoritmos iterativos.

5.8.1. Formatos compactos de matrices dispersas

Sea $\mathbf{A}$ una matriz dispersa con M filas , N columnas y NNZ elementos no ceros. Los formatos compactos de $\mathbf{A}$ se presentan en la Figura 5.23.
El formato COO consume más memoria del sistema. Además, las operaciones como producto matriz-vector o suma de elementos de una fila/columna de la matriz $\mathbf{A}$ son operaciones de tipo 'row/column-driven operations' y son más eficientes si se ejecutan en formatos CSR o CSC respectivamente. En las siguientes secciones evaluaremos diferentes versiones de los algoritmos relacionados con las operaciones más costosas de los algoritmos SART y LSQR.

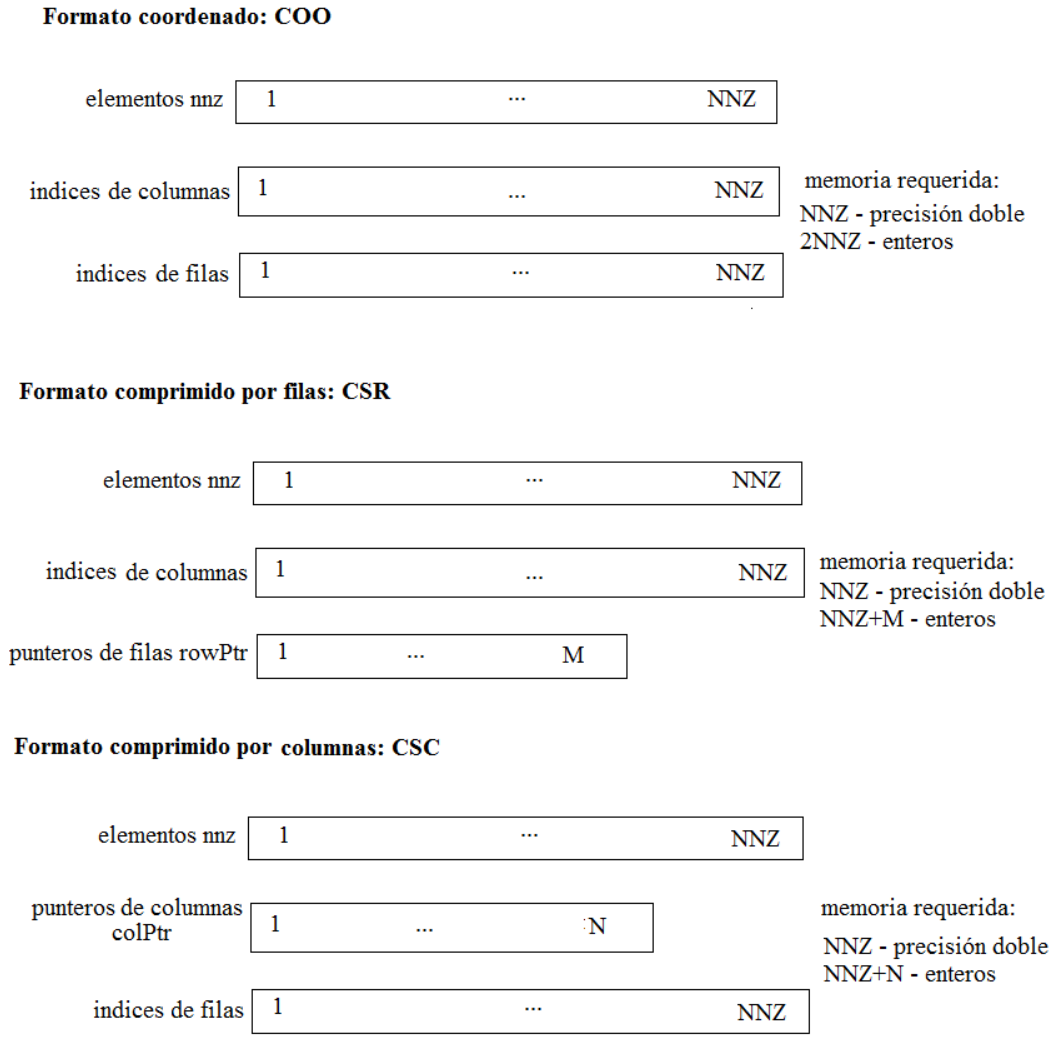

Figura 5.23: Los formatos compactos para una matriz dispersa.

En la siguiente sección, analizaremos los algoritmos de las operaciones mencionadas en el inicio de esta sección para diferentes formatos de la matriz dispersa.

5.8.2. Producto matriz-vector

A continuación, se presenta el algoritmo de producto matriz-vector para la matriz A en formato COO ordenado por filas.

Algoritmo 5.8.1: Producto matriz-vector - versión 1.

- **Entrada**: matriz **A** en COO ordenado por filas , vector x
- **Salida**: vector V- resultado del producto

01: $row = 0$, $V[row] = 0$
02: for $k = 0 : NNZ$
03: if $(rows[k] = row)$
04: $co = cols[k]$
05: $V[row] \Leftarrow V[ro] + values[k] * x[co]$
06: end if
07: else
08: row++
09: $co = cols[k]$
10: $V[row] \Leftarrow V[row] + values[k] * x[co]$
11: end else
12: end for

Para la matriz en formato COO no ordenado por filas el producto matriz-vector se resume en el siguiente código:

Algoritmo 5.8.2: Producto matriz-vector - versión 2.

- **Entrada**: matriz **A** en COO , vector x
- **Salida**: vector V- resultado del producto

01: for $k = 0 : NNZ$
02: $ro = rows[k]$
03: $co = cols[k]$
04: $V[ro] \Leftarrow V[ro] + values[k] * x[co]$
05: end for

La tercera versión del producto matriz-vector se utiliza para la matriz A en formato CSR.

Algoritmo 5.8.3: Producto matriz-vector - versión 3

- **Entrada**: matriz **A** en CSR , vector x
- **Salida**: vector V- resultado del producto

01: for $p = 1 : M$
02: $start = rowPtr[p]$
03: $stop = rowPtr[p + 1]$
04: for $i = start : stop$
05: $co = cols[i]$
06: $V[p] \Leftarrow V[p] + values[i] * x[co]$
07: end for
08: end for

Las versiones paralelas del producto matriz-vector se realizan en base de los algoritmos **Algoritmo 5.8.2** y **Algoritmo 5.8.3** para comparar los formatos COO y CSR.

Algoritmo 5.8.4: Producto matriz-vector -versión paralela 1

- **Entrada**: matriz **A** en COO , vector x
- **Salida**: vector V - resultado del producto

pragma omp parallel for private (ro, co)
01: for $k = 0 : NNZ$
02: $ro = rows[k]$
03: $co = cols[k]$
pragma omp atomic
04: $V[ro] \Leftarrow V[ro] + values[k] * x[co]$
05: end for

Algoritmo 5.8.5: Producto matriz-vector - versión paralela 2

- **Entrada**: matriz **A** en CSR , vector x
- **Salida**: vector V - resultado del producto

```
  # pragma omp parallel for private( start, stop, co, i )
01:   for p = 1 : M
02:     start = rowPtr[p]
03:     stop = rowPtr[p + 1]
04:     for i = start : stop
05:         co = cols[i]
06:         V[p] ⇐ V[p] + values[i] ∗ x[co]
07:     end for
08:   end for
```

Prestaciones teóricas de los algoritmos

Coste del Algoritmo 5.8.1

$$\text{T1} = \sum_{k=1}^{NNZ} 2 => 2NNZ, \qquad \text{Tp} = \sum_{k=1}^{NNZ/p} 2 => 2NNZ/p$$

$$\text{SpeedUp} = \frac{T1}{Tp} = p, \qquad \text{Eficiencia} = \text{lím}_{N\to\text{inf}} \frac{S_p}{p} = 1.$$

Coste del Algoritmo 5.8.2

$$\text{T1} = \sum_{k=1}^{NNZ} 2 => 2NNZ, \qquad \text{Tp} = \sum_{k=1}^{NNZ/p} 2 => 2NNZ/p$$

$$\text{SpeedUp} = \frac{T1}{Tp} = p, \qquad \text{Eficiencia} = \text{lím}_{N\to\text{inf}} \frac{S_p}{p} = 1.$$

Coste del Algoritmo 5.8.3

$$\text{T1} = \sum_{i=1}^{M} \sum_{start}^{stop} 2 = \sum_{i=1}^{M} 2NNZ_i \approx 2NNZ, \qquad \text{Tp} = \sum_{i=1}^{M/p} \sum_{start}^{stop} 2 \approx \frac{2NNZ}{p}.$$

$$\text{SpeedUp} = \frac{T1}{Tp} = p, \qquad \text{Eficiencia} = \text{lím}_{N\to\text{inf}} \frac{S_p}{p} = 1.$$

Resultados experimentales

La evaluación experimental de los algoritmos del producto de matriz-vector en COO y CSR se resume en la tabla 5.7.

ProblemSize	256x100		512x100	
# hilos OpenMP	COO	CSR	COO	CSR
1	0.24	0.23	1.16	0.42
2	0.12	0.12	0.60	0.21
4	0.07	0.06	0.30	0.11
8	0.04	0.03	0.18	0.06
16	0.02	0.02	0.12	0.04
24	0.02	0.02	0.09	0.03
32	0.02	0.01	0.07	0.02

Tabla 5.7: Tiempo de ejecución (en segundos) de producto matriz-vector en formatos COO y CSR en función de tamaño del problema y el número de procesos.

El speedUp y la eficiencia experimental de los algoritmos se presentan en las figuras 5.24 y 5.25.

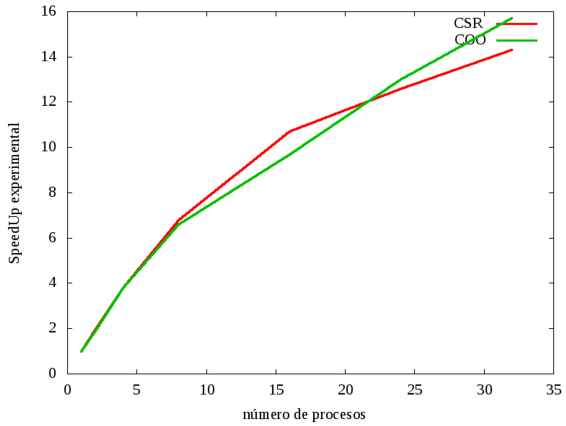

Figura 5.24: SpeedUp experimental del producto matriz-vector (versiones paralelas 1 y 2) para un problema de 512x100 en COO y CSR.

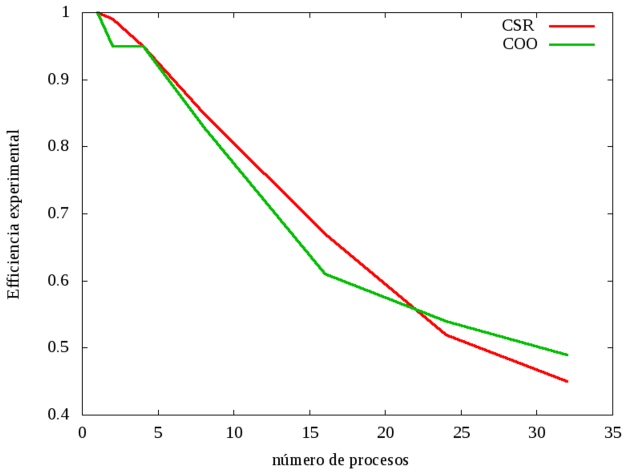

Figura 5.25: Eficiencia experimental del producto matriz-vector (versiones paralelas 1 y 2) en COO y CSR.

Analizando los resultados en la tabla 5.7 y figuras 5.24 y 5.25, se observa que para problemas relativamente pequeños (256x100) el tiempo de cómputo del producto matriz-vector no depende mucho del formato de la matriz. Sin embargo, con el crecimiento de tamaño del problema esta dependencia es notable. Así, para la matriz del tamaño 512x100 empleando 32 procesadores, se logra acelerar en 3.5 veces la operación producto matriz-vector en formato CSR comparado con COO.

5.8.3. Algoritmo Suma por filas

La operación de suma de coeficientes de la matriz A por filas se presenta para los formatos COO y CSC.

Algoritmo 5.8.5: Suma de elementos por filas - versión 1

- **Entrada**: matriz **A** en COO , vector x
- **Salida**: vector $sumaFilas$

01: for $j = 1 : N$
02: $suma = 0$
03: for $k = 1 : NNZ$

04: if $cols[k] == j$
05: $suma \Leftarrow suma + values[k]$
06: endif
07: endfor
08: $sumaFilas[i] \Leftarrow suma$
09: end for

Algoritmo 5.8.6: Suma de elementos por filas - versión 2

- **Entrada**: matriz **A** en CSC , vector x
- **Salida**: vector $sumaFilas$

01: for $p = 1 : N$
02: $start = colPtr[p]$
03: $stop = colPtr[p + 1]$
04: for $i = start : stop$
05: $sumaFilas[p] \Leftarrow sumaFilas[p] + values[i]$
06: end for
06: end for

Las versiones paralelas de los algoritmos **Algoritmo 5.8.5** y **Algoritmo 5.8.6** se resumen a continuación.

Algoritmo 5.8.7: Suma de elementos por filas - versión paralela 1

- **Entrada**: matriz **A** en COO , vector x
- **Salida**: vector $sumaFilas$

pragma omp parallel for private (suma, k)
01: for $j = 1 : N$
02: $suma = 0$
03: for $k = 1 : NNZ$
04: if $cols[k] == j$
05: $suma \Leftarrow suma + values[k]$

06: endif
07: endfor
08: $sumaFilas[i] \Leftarrow suma$
09: end for

Algoritmo 5.8.8: Suma de elementos por filas - versión paralela 2

- **Entrada**: matriz **A** en CSC , vector x
- **Salida**: vector $sumaFilas$

pragma omp parallel for private(start, stop, i)
01: for $p = 1 : N$
02: $start = colPtr[p]$
03: $stop = colPtr[p + 1]$
04: for $i = start : stop$
05: $sumaFilas[p] \Leftarrow sumaFilas[p] + values[i]$
06: end for
06: end for

Evaluaciones teóricas de los algoritmos

Coste del Algoritmo 5.8.5

$$\text{T1} = \sum_{j=1}^{N} \sum_{k=1}^{NNZ} 2 \Longrightarrow 2N * NNZ \approx 2NNZ,$$

$$\text{Tp} = \sum_{j=1}^{N/p} \sum_{k=1}^{NNZ} 2 \Longrightarrow \frac{N}{p} 2 * NNZ \approx \frac{2NNZ}{p}.$$

$$\text{SpeedUp} = \frac{T1}{Tp} = p, \qquad \text{Eficiencia} = \lim_{N \to \text{inf}} \frac{S_p}{p} = 1.$$

Coste del Algoritmo 5.8.6

$$\text{T1} = \sum_{j=1}^{N} \sum_{start}^{stop} 1 \sum_{j=1}^{N} NNZ_j => NNZ,$$

$$\text{Tp} = \sum_{j=1}^{N/p} \sum_{start}^{stop} 1 => \frac{N}{p} NNZ_j \Rightarrow \frac{NNZ}{p}.$$

$$\text{SpeedUp} = \frac{T1}{Tp} = p, \qquad \text{Eficiencia} = \lim_{N \to \text{inf}} \frac{S_p}{p} = 1$$

5.8.4. Algoritmo Suma por columnas

En este apartado, en la forma análoga se presentan los algoritmos secuenciales y paralelas para la operación de suma de elementos por columnas de la matriz A en los formatos COO y CSR.

Algoritmo 5.8.9: Suma de elementos por columnas - versión 1

- **Entrada**: matriz **A** en COO , vector x
- **Salida**: vector $sumaColumnas$

```
01:  for i = 1 : M
02:    suma = 0
03:    for k = 1 : NNZ
04:       if rows[k] == i
05:          suma ⇐ suma + values[k]
06:       endif
07:    endfor
08:    sumaColumnas[i] ⇐ suma
09:  end for
```

Algoritmo 5.8.10: Suma de elementos por columnas - versión 2

- **Entrada**: matriz **A** en CSR , vector x
- **Salida**: vector $sumaColumnas$

```
01:  for p = 1 : M
02:    start = rowPtr[p]
03:    stop = rowPtr[p + 1]
```

04: for $i = start : stop$
05: $sumaColumnas[p] \Leftarrow sumaColumnas[p] + values[i]$
06: end for
07: end for

Algoritmo 5.8.11: Suma de elementos por columnas - versión paralela 1

- **Entrada**: matriz **A** en COO , vector x
- **Salida**: vector $sumaColumnas$

pragma omp parallel for private (suma, k)
01: for $i = 1 : M$
02: $suma = 0$
03: for $k = 1 : NNZ$
04: if $rows[k] == i$
05: $suma \Leftarrow suma + values[k]$
06: endif
07: endfor
08: $sumaColumnas[i] \Leftarrow suma$
09: end for

Algoritmo 5.8.12: Suma de elementos por columnas - versión paralela 2

- **Entrada**: matriz **A** en CSR , vector x
- **Salida**: vector $sumaColumnas$

pragma omp parallel for private(start, stop, i)
01: for $p = 1 : M$
02: $start = rowPtr[p]$
03: $stop = rowPtr[p + 1]$
04: for $i = start : stop$
05: $sumaColumnas[p] \Leftarrow sumaColumnas[p] + values[i]$
06: end for
07: end for

5.8.5. Vectorización

Los procesadores tienen unidades vectoriales que se usan con instrucciones SIMD (Simple Instruction Multiple Data). Utilizando las construcciones SIMD que proporciona la librería OpenMP4.0 se puede lograr el paralelismo en bucles a nivel vectorial.

Por ejemplo, el **Algoritmo 5.8.12** de suma de elementos por columnas de una matriz en forma paralelizada y vectorizada va tener la siguiente forma:

```
# pragma omp parallel for simd private( start, stop, i )
01:   for p = 1 : M
02:     start = rowPtr[p]
03:     stop = rowPtr[p + 1]
04:     for i = start : stop
05:         sumaColumnas[p] ⇐ sumaColumnas[p] + values[i]
06:     end for
07:   end for
```

En la sección 5.8.6 se presentan los resultados del efecto de paralelización y vectorización del método LSQR.

Evaluación de algoritmos

Coste del Algoritmo 5.8.9

$$\text{T1} = \sum_{i=1}^{M} \sum_{k=1}^{NNZ} 2 => 2 * M * NNZ \approx 2NNZ,$$
$$\text{Tp} = \sum_{i=1}^{M/p} \sum_{k=1}^{NNZ} 2 => 2 * \frac{M}{p} * NNZ \approx \frac{2NNZ}{p.}$$

$$\text{SpeedUp} = \frac{T1}{Tp} = p, \qquad \text{Eficiencia} = \lim_{N \to \text{ínf}} \frac{S_p}{p} = 1$$

Coste del Algoritmo 5.8.10

$$\text{T1} = \sum_{i=1}^{M} \sum_{start}^{stop} 1 = \sum_{i=1}^{M} NNZ_i => NNZ,$$

$$\text{Tp} = \sum_{i=1}^{M/p} \sum_{start}^{stop} 1 => \frac{NNZ}{p}.$$

$$\text{SpeedUp} = \frac{T1}{Tp} = p, \qquad \text{Eficiencia} = \lim_{N \to \text{inf}} \frac{S_p}{p} = 1$$

La evaluación experimental de los algoritmos de esta sección se presenta a continuación.

5.8.6. Comparación de SART en formatos compactos

Los algoritmos de las secciones 6.2 - 6.4 permiten realizar las operaciones básicas de algoritmos iterativos de reconstrucción de imágenes. Para su evaluación experimental, los utilizaremos en el algoritmo SART para llevar acabo tales operaciones como producto matriz-vector, suma de los elementos por filas/columnas. Se realizaron pruebas para diferentes tamaños del problema de reconstrucción. Se usaron las implementaciones de los algoritmos en formatos COO y CSC/CSR. Los resultados se resumen en la Tabla 5.8, donde se presentan el tiempo de ejecución (en segundos) para tales operaciones principales de algoritmo SART como producto matriz-vector, suma de elementos de una fila de la matriz del sistema A y el tiempo de una iteración de SART para una matriz del tamaño [256x100]x[256x256].

#hilos	sumaFilas-CSC	sumaFilas-COO	suma Columnas-CSR	sumaColumnas-COO
1	0.052	2283	0.051	910
2	0.026	1156	0.026	456
4	0.014	586	0.014	232
8	0.008	311	0.007	124
16	0.005	196	0.004	98
24	0.004	128	0.004	51
32	0.004	101	0.004	39

Tabla 5.8: Tiempo de ejecución (en segundos) de la operación suma de elementos de filas y de columnas en formatos CSR/CSC y COO como función de número de procesos.

Como se esperaba, el formato COO eleva considerablemente el costo computacional de las operaciones. Esto se debe a la diferencia en la forma de acceso a los elementos

de la matriz en formatos compactos.

Para analizar el efecto de la paralelización de SART en función del tamaño de problema, se ejecuta SART en formato CSR/CSC con las matrices de tamaños [128x100]x[128x128], [256x100]x[256x256] y [512x100]x[512x512]. Los resultados de los tiempos de ejecución (en segundos) por iteración se resumen en la Tabla 5.9. En la Tabla 5.10 y en la Figura 5.26 se presentan las prestaciones experimentales de SART. Analizando los resultados de la Tabla 5.10 se observa lo siguiente: al aumentar el número de hilos la eficiencia decrece; al mantener el número de hilos constante y aumentando el tamaño del problema, la eficiencia del algoritmo crece.

# hilos OpenMP	128x128x100	256x256x100	512x512x100
1	0.0232	0.181	1.45
2	0.0122	0.092	0.73
4	0.0632	0.053	0.38
8	0.0042	0.028	0.20
16	0.0032	0.018	0.15
24	0.0028	0.014	0.11
32	0.0026	0.013	0.10

Tabla 5.9: Tiempo de ejecución (en segundos) por iteración de SART en formato CSR/CSC para diferentes tamaños del problema.

También se puede observar que al aumentar simultáneamente el número de hilos y el tamaño del problema en la misma proporción, la eficiencia del sistema paralelo se mantiene al nivel aproximado de 90 por ciento. De este análisis se concluye que el algoritmo es escalable.

# hilos OpenMP	128x128x100	256x256x100	512x512x100
1	1.00	1.00	1
2	**1.00**	1.00	0.99
4	0.83	**0.90**	0.95
8	0.63	0.81	**0.91**
16	0.42	0.63	0.61
24	0.29	0.54	0.55
32	0.22	0.43	0.46

Tabla 5.10: Eficiencia de SART en formato CSC/CSR: al aumentar el número de hilos y el tamaño de problema en la misma proporción se mantiene la eficiencia aproximadamente de 90 por ciento que indica la escalabilidad del algoritmo.

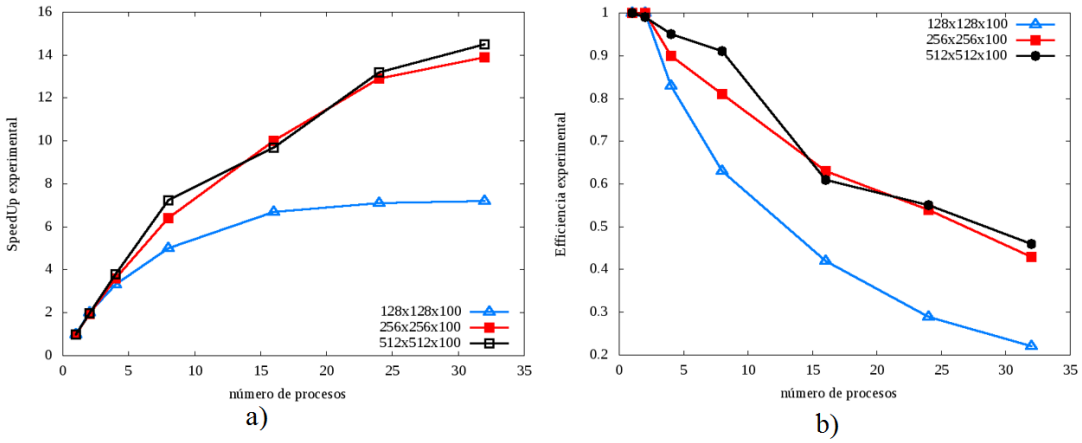

Figura 5.26: Speed Up (a) y Eficiencia (b) de SART como función del tamaño de problema y el número de processos.

Finalmente, en la Figura 5.27, se presentan las reconstrucciones de una imagen con el algoritmo SART en el cluster Kahan.

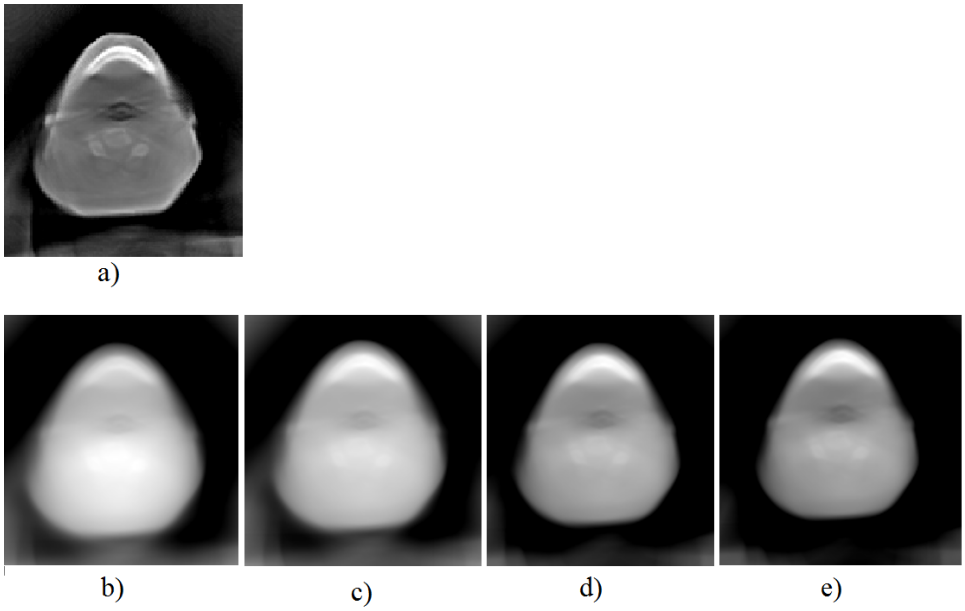

Figura 5.27: (a) - imagen de referencia reconstruida por FBP; (b)-(e) - imágenes reconstruidas por SART después de 2, 10, 50 y 100 iteraciones respectivamente.

Se puede concluir que el algoritmo SART es paralelizable y la implementación paralela es más eficiente para problemas grandes.

5.8.7. Comparación de LSQR en formatos compactos

Como se mencionó en la descripción del método LSQR, las operaciones más costosas de este método son el producto matriz-vector y el producto matriz transpuesta-vector. En este apartado presentamos la evaluación numérica del algoritmo LSQR en los formatos compactos COO, CSR/CSC.

En la tabla 5.11 se resume el tiempo por iteración (en segundos) usado por LSQR en la reconstrucción de una capa de la imagen de 256x256 y 512x512 píxeles. Las imágenes se reconstruyen para 100 proyecciones tomadas en el rango 0-360 grados. En la Tabla 5.12 se resume el tiempo de reconstrucción con LSQR de las imágenes de diferentes tamaños después de 10 iteraciones.

El speedUp y la eficiencia experimentales del algoritmo LSQR en la reconstrucción iterativa de las imágenes de diferentes tamaños se presentan en la Figura 5.28 y Tabla 5.13. Se compara LSQR en formato CSR/CSC para diferentes tamaños del

# hilos	CSR/CSC 256x256	COO 256x256	CSR/CSC 512x512	COO 512x512
1	0.353	0.915	1.646	4.482
2	0.182	0.555	0.845	2.520
4	0.097	0.303	0.455	1.336
8	0.058	0.210	0.261	0.785
16	0.038	0.144	0.169	0.571
24	0.040	0.114	0.158	0.431
32	0.040	0.100	0.149	0.384

Tabla 5.11: Tiempo de ejecución (en segundos) por iteración de LSQR para una imagen de 256x256 y 512x512 píxeles en formatos compactos.

# hilos	128x128x100	256x256x200	512x512x100
1	0.221	2.432	24.603
2	0.150	1.561	13.411
4	0.090	0.850	7.212
8	0.061	0.622	4.304
16	0.042	0.421	3.120
24	0.036	0.320	2.421
32	0.034	0.261	2.052

Tabla 5.12: Tiempo de ejecución (en segundos) por 10 iteración de LSQR para diferentes tamaños del problema en función de número de hilos OpenMP.

problema. En la Tabla 5.13 se observa que al aumentar simultáneamente el número de hilos y el tamaño del problema en la misma proporción, la eficiencia del sistema paralelo se mantiene al nivel aproximado de 71 por ciento. Esto indica la escalabilidad del algoritmo.

# hilos	128x128x100	256x256x200	512x512x100
1	1.00	1.00	1.00
2	**0.73**	0.77	0.91
4	0.61	**0.70**	0.85
8	0.46	0.48	**0.71**
16	0.35	0.38	0.49
24	0.20	0.32	0.42
32	0.25	0.24	0.37

Tabla 5.13: Eficiencia de LSQR para diferentes tamaños del problema: aumentando el número de procesos y el tamaño de problema en la misma proporción se mantiene la eficiencia de 71 por ciento que indica la escalabilidad de LSQR.

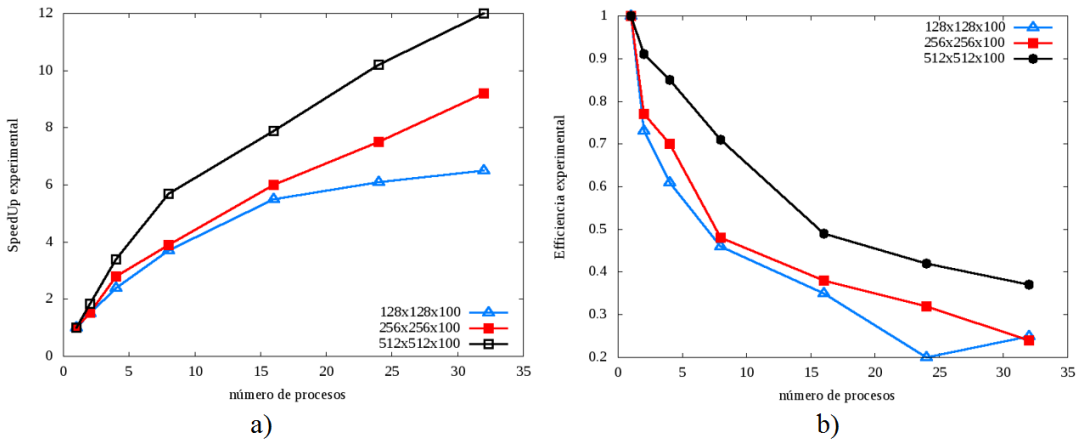

Figura 5.28: El speedUp (a) y la Eficiencia (b) de LSQR en formato CSR/CSC como función de número de procesos para diferentes tamaños de problema.

Para explotar las unidades vectoriales del procesador, el algoritmo LSQR fue ejecutado de diferentes modos: paralelizando bucles, vectorizando bucles y utilizando ambos, la paralelización y la vectorización de bucles. La comparación de los resultados de reconstrucción de imágenes de 256x256 y 512x512 píxeles reconstruidas con 100 proyecciones se presenta en la Figura 5.29. Se observa que mientras crece el tamaño del problema, la combinación de la paralelización y vectorización de bucles lleva al mejor resultado.

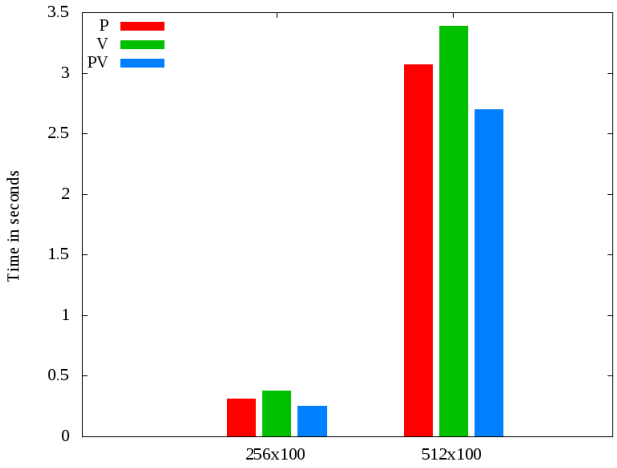

Figura 5.29: Comparación de ejecución de LSQR en la reconstrucción de imágenes de 256x256 y 512x512 píxeles: paralelizando bucles (P), vectorizando bucles (V) y paralelizando y vectorizando bucles (PV).

En la Figura 5.30 se muestran las reconstrucciones obtenidas por LSQR en el cluster Kahan.

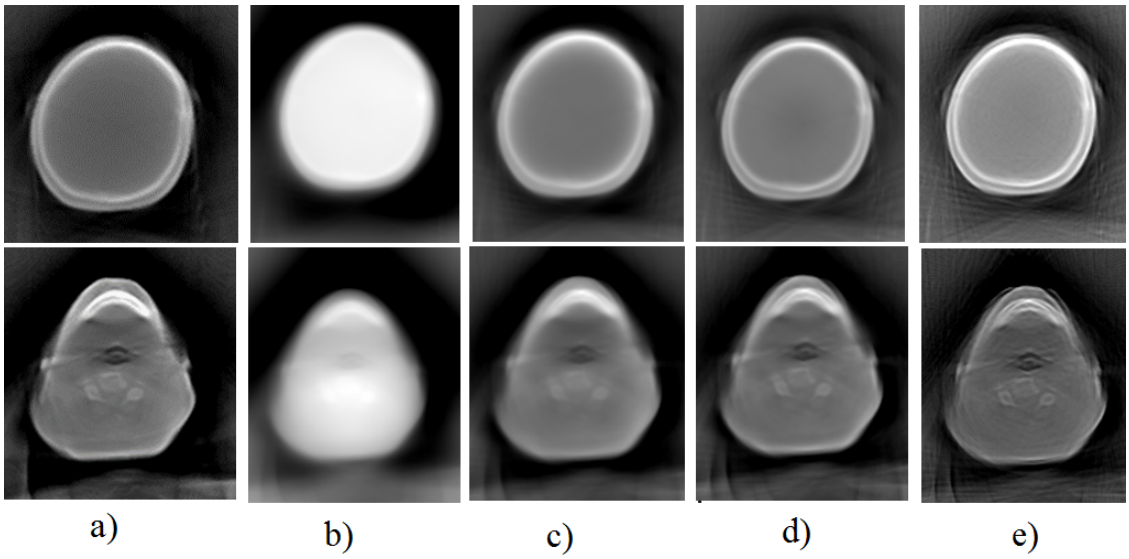

Figura 5.30: Reconstrucción de imágenes de 256x256 píxeles: (a) - imágenes de referencia reconstruidas por FBP; (b)-(e) - reconstrucciones por LSQR después de 2, 6, 10 y 20 iteraciones respectivamente.

Los resultados de esta subsección indican que la utilización de la matriz del sistema en el formato CSR/CSC lleva a la reducción del tiempo de reconstrucción y de la memoria del sistema.
Analizando los resultados de la Tabla 5.13 se observa que al mantener el número de hilos constante y aumentando el tamaño del problema, la eficiencia del algoritmo crece. También se puede observar que al aumentar simultáneamente el número de hilos y el tamaño del problema en la misma proporción se puede mantener la eficiencia del sistema paralelo al nivel aproximado de 70 por ciento. De este análisis se concluye que el algoritmo LSQR es escalable.

5.8.8. Conclusiones

Comparando la utilización de formatos compactos en el proceso de reconstrucción de imágenes por métodos iterativos, como SART y LSQR, se concluye que el formato CSR/CSC es el más adecuado. Este formato permite reducir la memoria utilizada por el sistema y también el tiempo de reconstrucción de la imagen. Para la reconstrucción de una imagen de 512x512 píxeles el tiempo por una iteración se reduce en promedio 2.6 veces. Esto lleva a una reducción de tiempo considerable en el proceso de la reconstrucción de una imagen entera, especialmente en 3D. Por esta razón, en las secciones posteriores, se van a utilizar las matrices almacenadas en el formato CSR/CSC.
Los resultados obtenidos también indican que la paralelización de los algoritmos estudiados sobre las arquitecturas multi-core conduce a los mejores resultados cuando el algoritmo se ejecuta con el número de procesos igual al número de cores en el procesador con la utilización de unidades vectoriales.

6 Arquitecturas altamente paralelas

Los multi-núcleos CPUs y las GPUs representan sistemas paralelos que permiten particionar el problema y resolverlo de forma independiente en paralelo por bloques de 'threads'. Esta descomposición permite la cooperación entre 'threads' de un bloque, y al mismo tiempo, la escalabilidad.
Cada bloque puede ser ejecutado por cualquiera de los multiprocesadores disponibles en GPU, en cualquier orden como se ilustra en la Figura 6.1.

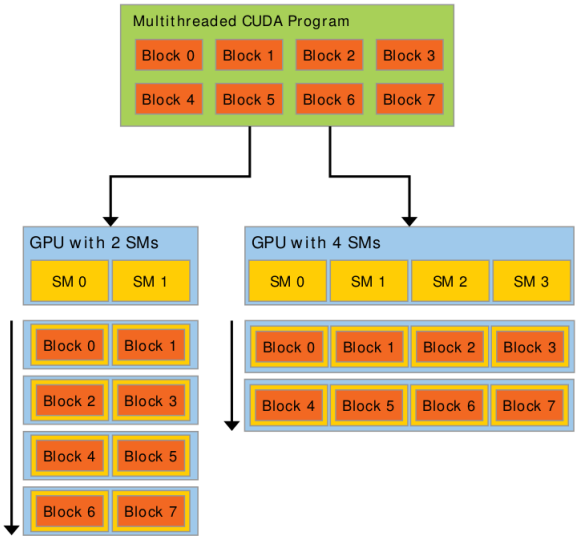

Figura 6.1: Una GPU con un array de multiprocesadores (SMs). Una GPU con más multiprocesadores va a ejecutar el programa en menos tiempo [48].

El modelo de programación CUDA se ilustra en la Figura 6.2. En este modelo, el código secuencial se ejecuta en 'host' y el código paralelo va ser ejecutado en 'device'. El modelo de programación CUDA asume también que el 'host' y 'device' mantienen su propio espacio de memoria, llamados memoria 'host' y memoria 'device'.

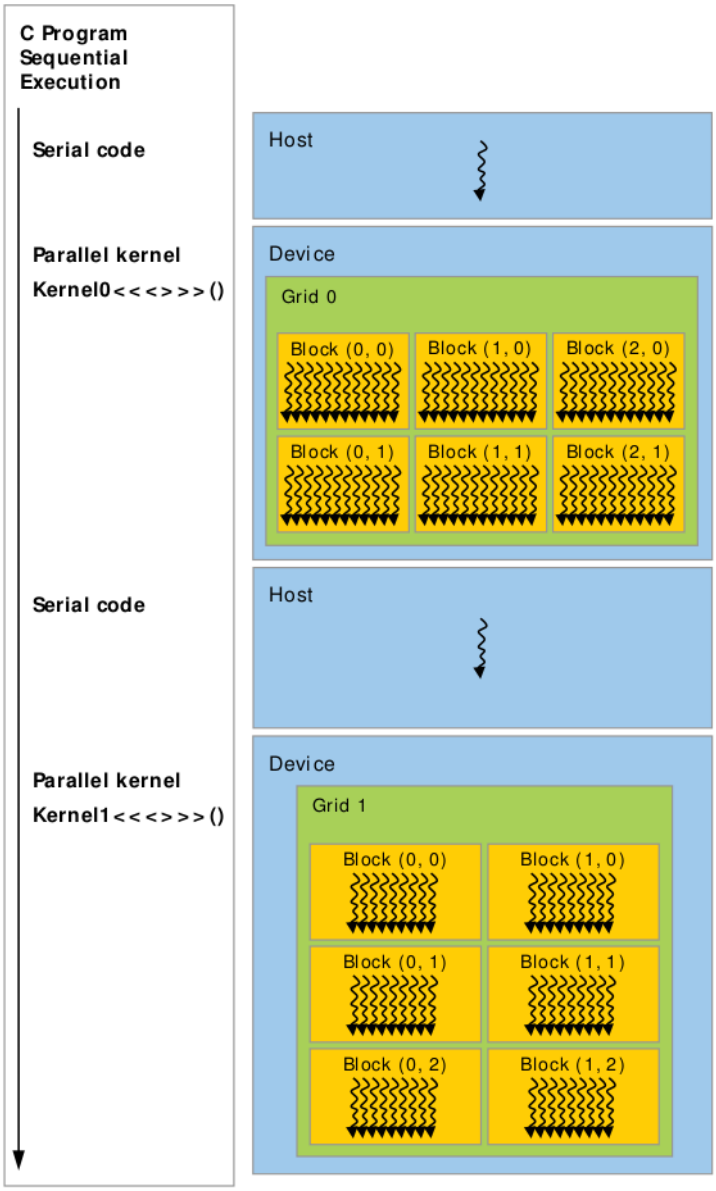

Figura 6.2: Programación heterogénea [48].

El principal factor que limita el rendimiento de las GPUs es el tiempo de acceso a datos. Por este motivo, es importante que la memoria esté bien estructurada y se gestione de forma eficaz y eficiente.

Con un número elevado de procesadores, todos realizando cálculos simultáneamente

sobre datos en diferentes direcciones de memoria, se hace necesario un gran ancho de banda para atender todas las peticiones. Además, cada 'thread', cada 'warp' y cada bloque trabajarán sobre un grupo limitado de direcciones de memoria, por lo que la localidad espacial es fácilmente explotable mediante cachés.

Jerarquía de memoria en CUDA. Como se presenta en la Figura 6.3, se puede distiguir cuatro zonas dentro de la memoria de video:

- memoria de constantes
- memoria de texturas
- memoria global
- memoria local

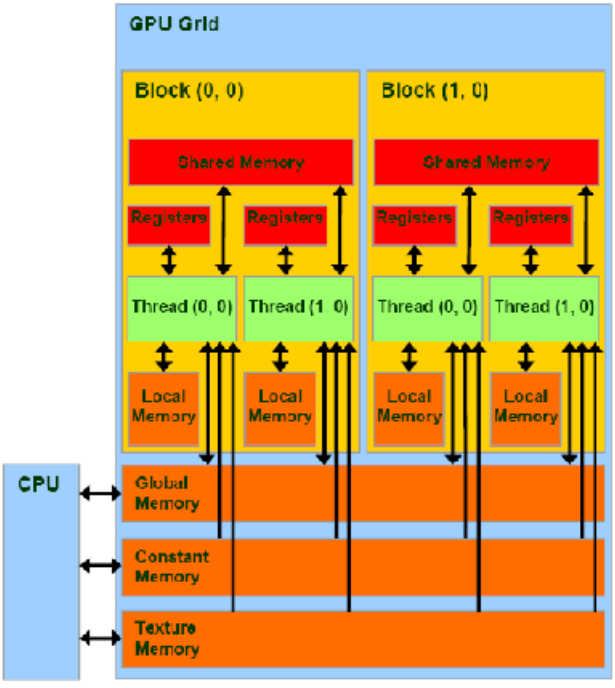

Figura 6.3: Memorias en la GPU [48].

Las memorias de constantes y de texturas se utilizan para datos que no varían a lo largo de la ejecución del subprograma. Son de sólo lectura, y en caso de las texturas, sus patrones de acceso son muy predecibles. Por ello se sitúan aquí unas cachés

especiales, las unidades de acceso a memoria de texturas, que aprovechan localidad espacial.

La memoria global es la memoria donde la GPU almacena los resultados de los subprogramas antes de devolverlos a la memoria principal. Teniendo en cuenta el ancho de banda, es más rápida que la memoria principal, y todos los 'threads' tienen acceso a ella. Pero es tremendamente lenta en comparación con la velocidad a la que los procesadores gráficos llevan a cabo los cálculos. Por eso tenemos un nuevo nivel de memoria, una memoria compartida para cada procesador. Es más rápida porque está más cerca de su procesador, y es más pequeña por razones de coste. Permite comunicación de 'threads' del mismo bloque.

Además, cada núcleo de cada procesador tiene la memoria local, y sólo es accesible para su 'thread' correspondiente, lo que se muestra en la Figura 6.4.

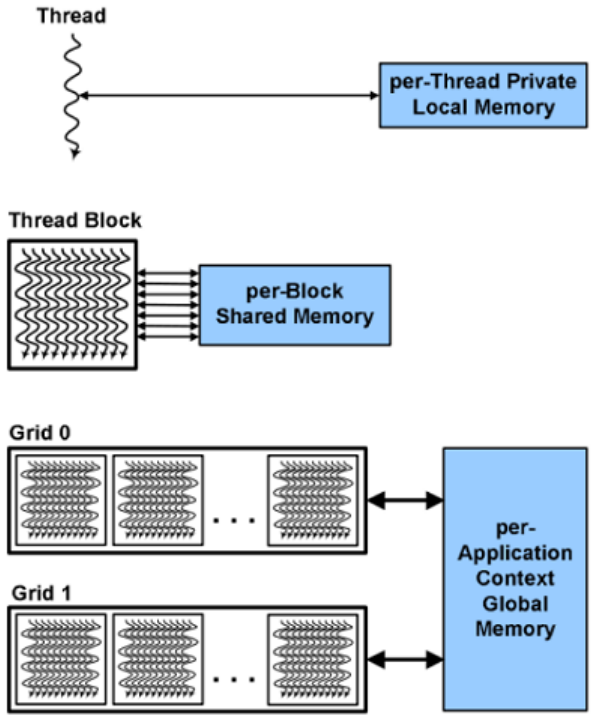

Figura 6.4: Acceso a las memorias en la GPU [48].

Las memorias global, local y de constantes y de texturas se ubican en la memoria

de vídeo de la tarjeta, en chips independientes del chip principal de la GPU. El principal problema de la memoria de vídeo es que se encuentra físicamenete más alejada de la GPU y por tanto tiene una latencia mayor comparada con latencia de memoria compartida y registros (integrada en la GPU).

Para la GPU, una textura plana es una matriz bidimensional. Estas matrices son de grandes dimensiones. Esto requiere una memoria muy grande y muy rápida, lo cual conlleva un coste en consumo muy elevado. Para minimizar este coste, se utilizan memorias más lentas y cachés para agilizar el acceso a éstas. En un acceso típico a una imagen, no se va a leer tan sólo un punto de ésta; lo habitual es leerla entera o, al menos una parte. Se juega con esta particularidad para diseñar una caché eficaz aprovechando la localidad espacial.

Además, hay operaciones muy comunes en texturas, como filtrados que es más rápido y fácil realizar en hardware, que escribir y ejecutar software para llevarlas a cabo.
Existe una desventaja en el uso de la memoria del dispositivo: la memoria global es grande pero lenta, mientras que la memoria compartida es rápida pero pequeña. La zona de memoria destinada a almacenamiento de texturas tiene asociadas memorias caché. Al igual que sucede con la memoria principal de un ordenador, la GPU sólo accederá a memoria global si existe un fallo de caché al ir a buscar un dato.

La memoria de texturas está optimizada para aprovechar la localidad espacial de dos dimensiones. De este modo, se consigue mejor rendimiento a medida que los hilos del mismo 'warp' acceden a direcciones más cercanas de la memoria. La Figura 6.5 ilustra el funcionamiento de la memoria de texturas.

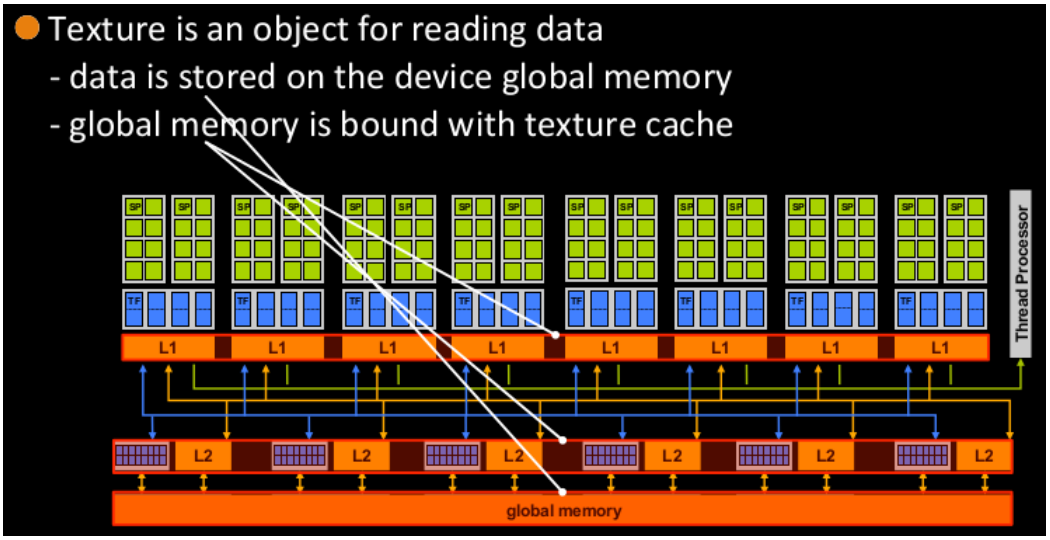

Figura 6.5: Memoria de texturas.

El acceso a la memoria a través de texturas se conoce como 'texture fetch' y se realiza especificando a 'texture object' o 'texture reference'.

Un objeto de textura (texture object) se crea con la función cudaCreateTextureObject() previamente especificando la textura con cudaResourceDescription y describiendo la textura con cudaTextureDescription. Un objeto de textura creado pasa al kernel como un parámetro.
Los pasos para el de uso de la memoria de texturas a traves de referencias son los siguientes:

- Declarar referencia de textura
  texure $\langle float, 1, cudaReadModeElementType \rangle$ textureX;
- Reservar espacio en host y device
  $hX = (float*)malloc(size * sizeof(float))$;
  cudaMalloc ((void**) $dX, sizeof(float) * size)$);
- Copiar array de host a device
  cudaMemcpy $(dX, hX, sizeof(float) * size,$ cudamemcpyHostToDevice);
- Enlazar la memoria de texturas con memoria lineal

cudaBindTexture $(0, textureX, dX, sizeof(float) * size)$;

- Ejecutar kernel
  kernel$\langle\langle\langle ... \rangle\rangle\rangle$ $(dX, size)$;
- Desenlazar la referencia de textura para liberar resurso
  $cudaUnbindTexture(textureX)$;
- En kernel
  global void kernel()
  - calcular índice global de cada thread
    $index = blockIdx.x * blockDim.x + threadIdx.x$;
  - acceder a la memoria global a traves de referencia de textura
    $outArray[idex] = tex1Dfetch(textureX, index)$;

A continuación presentamos las versiones los algoritmos SART y LSQR en GPUs.

6.1. SART: implementación en GPU

El algoritmo SART para la tomografía de transmision fue presentado y analizado en la sección 5.5 del capítulo 5. Hemos dicho que las operaciones de cómputo utilizadas en el proceso de reconstrucción son de tipo píxel-vóxel, tienen pocas dependencias y son ejecutadas en bucles grandes. Las plataformas apropiadas para este tipo de operaciones son vector-procesadores o arquitecturas masivamente paralelas como unidades de procesamiento gráfico. En este capítulo vamos a presentar la implementación SART en GPUs.

El algoritmo SART empieza con una estimación inicial x_0 y proporciona la siguente procedura para obtener la nueva estimación a partir de la anterior:

$$x^{k+1} = x^k + \lambda^k \frac{1}{\sum_{i=1}^{M} a_{ij}} \sum_{i=1}^{M} \frac{a_{ij}}{\sum_{j=1}^{N} a_{ij}} (p_i - A_i x^k), \tag{6.1}$$

donde x es el vector de N píxeles de la imagen, p es el vector de proyecciones con M componentes, $a_{i,j}$ son elementos de la matriz del sistema de dimensiones MxN

que da la proporción de rayo i que pasa a través de un píxel j; λ es un parámetro regularizador que toma un valor entre 0 y 1.

6.1.1. SART - versión 1

El algoritmo SART es un algoritmo iterativo. En la ecuación 6.1 hay operaciones que no se ejecutan en forma iterativa y deben realizarse antes de empezar el proceso de iteraciones. La Figura 6.6 presenta el diagrama de flujos de la primera versión de la implementación de SART en GPU.

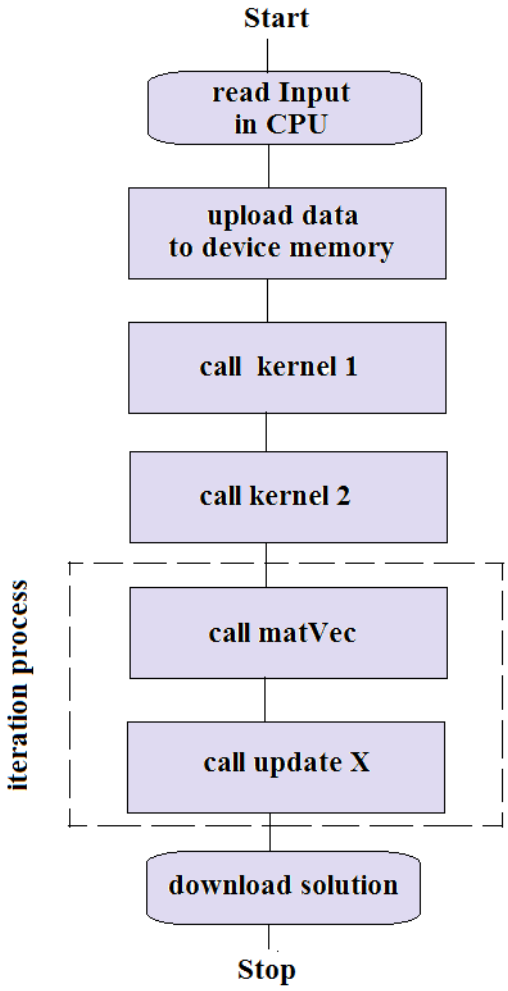

Figura 6.6: El diagrama de flujos de la implementación SART-versión 1 en GPU.

A continuación describimos los kernels utilizados en SART-versión 1 en correspon-

dencia con la fórmula(6.1).

kernel 1: Se calcula la suma de elementos de **A** por filas:

. $\sum_{i=1}^{M} a_{ij} \Longrightarrow$ sumaFilas [Mx1].

kernel 2: Se calcula la suma de elementos de **A** por columnas:

. $\sum_{j=1}^{N} a_{ij} \Longrightarrow$ sumaColumnas [Nx1].

matVec: Se calcula el error en la iteración k:

. $p - Ax \Longrightarrow$ ptemp [Mx1].

updateX: Se actualiza el valor de x en la iteración k:

Se trabaja con la matriz **A** en formato CSR/CSC con M filas y N columnas. Cada hilo del bloque trabaja en un píxel de la imagen.

```
01:  While j < N
02:      numerador = 0
03:      start = cscColPtr[i]
04:      stop = cscColPtr[i+1]
05:      for k = start : stop
06:          ro = cscRowInd[k)
07:          numerador ⇐ cscValues[k]/sumaColumnas[ro] * btemp[ro]
08:      endfor
09:      x[j] += pow(λ, iter) * numerador / sumaFilas[j]
10:  endwhile
```

6.1.2. SART - versión 2

En la versión 2 del algoritmo SART, se excluye del bucle de iteraciones una operación más, la operación de normalización de coeficientes de **A** descrita en el **kernel3**. El diagrama de flujos del algoritmo SART-versión 2 se presenta en la Figura 6.7.

Los kernels : **kernel 1**, **kernel 2** y **matVec** son similares a SART-versión 1. Los kernels diferentes se describen a continuación.

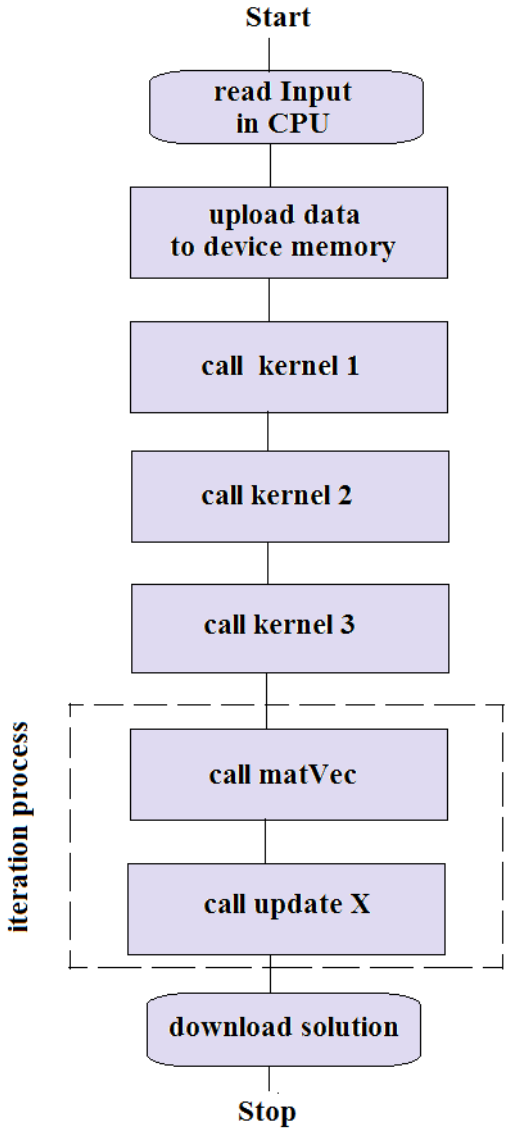

Figura 6.7: El diagrama de flujos de la implementación SART-versión 2 en GPU.

kernel 3: Se normalizan los valores de **A**:

. $\frac{a_{ij}}{\sum_{j=1}^{N} a_{ij}} \Longrightarrow$ **cscValues2[nnzx1]**.

updateX: Se actualiza el valor de x en la iteración k considerando kernel 3:

```
01:  While j < ncols
02:      numerador = 0
03:      start = cscColPtr[i]
04:      stop = cscColPtr[i+1]
05:      for k = start : stop
06:          ro = cscRowInd[k)
07:          numerador ⇐ values2[k] * btemp[ro]
08:      endfor
09:      x[j] += λ^iter * numerador / sumaFilas[j]
10:  endwhile
```

6.1.3. Resultados experimentales

Los resultados de comparación de las versiones SART 1 y 2 se resumen en la Tabla 6.1.

SART	Tamaño de Matriz	
	[256x133]x[256x256]	[256x400]x[256x256]
versión 1	26.6 ms	89.5 ms
versión 2	20.0 ms	66.3 ms

Tabla 6.1: Tiempo (en ms) por una iteración SART para diferentes tamaños de problema.

Como ejemplo, en la Figura 6.8 se presentan imágenes reconstruidas con SART después de 13, 20, 50, 100 y 200 iteraciones.

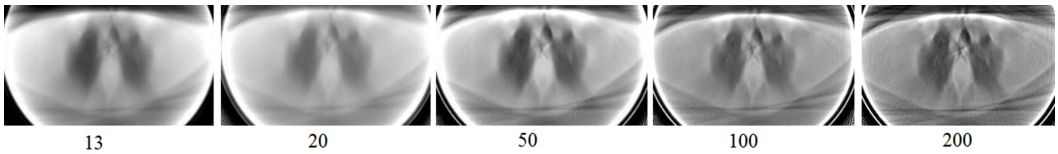

Figura 6.8: Reconstrucción SART después de 13, 20, 50, 100 y 200 iteraciones.

6.1.4. Conclusiones

El proceso de convergencia en SART es muy lento. En consecuencia, la reconstrucción de una imagen requiere numerosas iteraciones, lo que, a su vez, eleva el coste computacional del algoritmo SART y puede presentar dificultades para usos prácticos, especialmente en 3D.

6.2. LSQR: implementación en GPU

Para la reconstrucción de imágenes hemos implementado el mismo algoritmo iterativo LSQR descrito en la sección 5.7. Hemos utilizado el modelo de programación CUDA junto con las librerías CUBLAS y CUSPARSE ([48]-[50]). El modelo permite resolver muchos problemas complejos computacionalmente de un modo más eficiente que con CPU. La Figura 6.9 ilustra el esquema del algoritmo y la relación entre las librerías usadas.

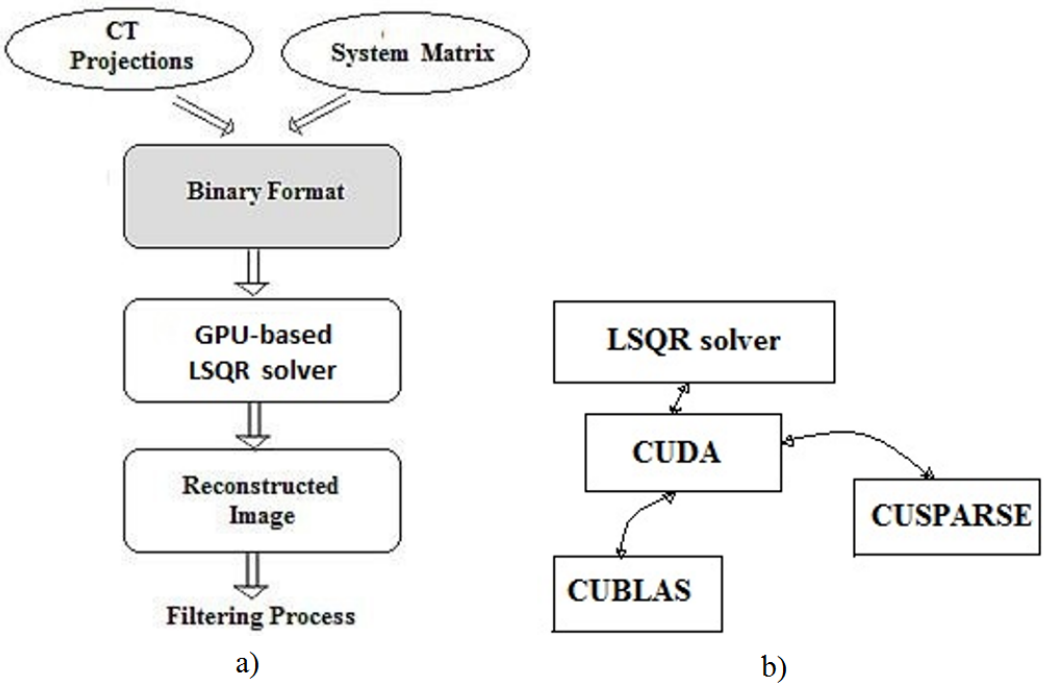

Figura 6.9: (a) El solver LSQR usa los datos de entrada en formato binario para la reconstrucción; (b) Librerías usadas para la implementación de LSQR.

A continuación, se presenta la parte del código de uso de librerías para calcular la norma de un vector y el producto matriz-vector:

```
01: cublasCreate ( &handle b );
02: cublasSetVector ( nrow, sizeof(float), h U, 1, d U, 1 );
03: cublasSnrm2( handle b, nrow, d U, 1, &beta );
04: cublasScal( handle b, nrow, &beta1, d U, 1 );
05: cublasGetVector( nrow, sizeof(float), d U, 1, h U, 1 );
```

producto matriz - vector :

```
06: cusparseCreate(&handle s);
07: cusparseCreateMatDescr(&descra);
08: cusparseSetMatType(descra, CUSPARSE MATRIX TYPE GENERAL);
09: cusparseSetMatIndexBase(descra,CUSPARSE INDEX BAS ZERO);
10: cusparseScsrmv ( handle s, CUSPARSE OPERATION NON TRANSPOSE,
ncol, nrow, 1.0, descra, csc values, cscColPtr, cscRowInd, d U, 0.0, d V );
11: cublasGetVector( ncol, sizeof(float), d V, 1, h V, 1 );
```

6.2.1. Optimización del algoritmo

Una de las técnicas de optimización consideradas en la implemantación de LSQR es la utilización de funciones de librerías CUBLAS y CUSPARSE.
Las oportunidades de optimización más importantes y efectivas se presentan en la exploración del uso efectivo de la memoria de las unidades GPU. En GPU existen diferentes tipos de memoria. Los datos se almacenan en la memoria global de la tarjeta. En nuestra implementación, los datos de sólo lectura se almacenan en la memoria constante. La memoria más rápida, memoria compartida, se usa para guardar los resultados temporales siempre que sea posible.

6.2.2. Resultados experimentales

Para propósitos experimentales, hemos usado las proyecciones reales y imágenes de referencia adquiridos en el Hospital Clínico Universitario de Valencia. Las

proyecciones 'fan-beam' fueron recolectadas por el escáner con 512 detectores en el rango 0 - 180 con espaciado angular de 0.9 grados. Para poder reconstruir la imagen por el método iterativo el conjunto dado de proyecciones fue completado hasta 360 grados usando la estructura simétrica de la matriz del sistema. Hemos propuesto como objetivo analizar la capacidad del metodo iterativo LSQR en la reconstrucción de imágenes con un menor número de proyecciones. Con este fin, del conjunto inicial hemos obtenido tres subconjuntos de proyecciones equiespaciadas (con el paso angular de 0.9, 1.8, y 3.6 grados).

Para las imágenes de 256x256 y 512x512 píxeles, el tiempo de reconstrucción (en segundos) en CPU con varios hilos OpenMP y en una unidad GPU se presenta en la Tabla 6.2.

Matriz del sistema (filas x columnas)	CPU #hilos=1	CPU #hilos=16	GPU Memoria Global	GPU Memoria Const.&Compartida
(256x100) x (256x256)	2.72	0.40	0.16	0.10
(256x200) x (256x256)	5.34	0.74	0.31	0.18
(256x400) x (256x256)	10.53	1.48	0.62	0.32
(512x100) x (512x512)	12.34	2.93	0.66	0.33
(512x200) x (512x512)	24.43	5.67	1.27	0.67

Tabla 6.2: El tiempo (en segundos) de reconstrucción en CPU y GPU.

En la GPU, el algoritmo fue ejecutado de dos formas: utilizando sólo la memoria global de la tarjeta, y, optimizando el algoritmo, utilizando las memorias constante y compartida. El tiempo de ejecución en CPU fue medido con la función gettimeofday(). En GPU, para medir el tiempo se usó la función cudaEventRecord() que mide tiempo de ejecución sólo en la unidad GPU sin tener en cuenta el tiempo de espera en la cola. La desviación estándar de los resultados después de correr la aplicación 10 veces es 2.9e-004.

Los resultados muestran la eficiencia del algoritmo basado en la habilidad de la

computación paralela de tarjetas gráficas. El SpeedUp hasta 36.4 fue logrado para reconstruir una imagen de 512x512 píxeles. Así mismo, se observa que el algoritmo es escalable ya que se hace más eficiente para problemas de mayor escala. Finalmente, la Figura 6.10 muestra las imágenes reconstruidas por diferente número de proyecciones. Usualmente, después de la reconstrucción se aplica postprocesamiento (filtrado) con el objetivo de mejorar la calidad de la imagen reconstruida. Las imágenes que se presentan son las obtenidas directamente del proceso de reconstrucción por LSQR sin ningún tipo de filtrado.

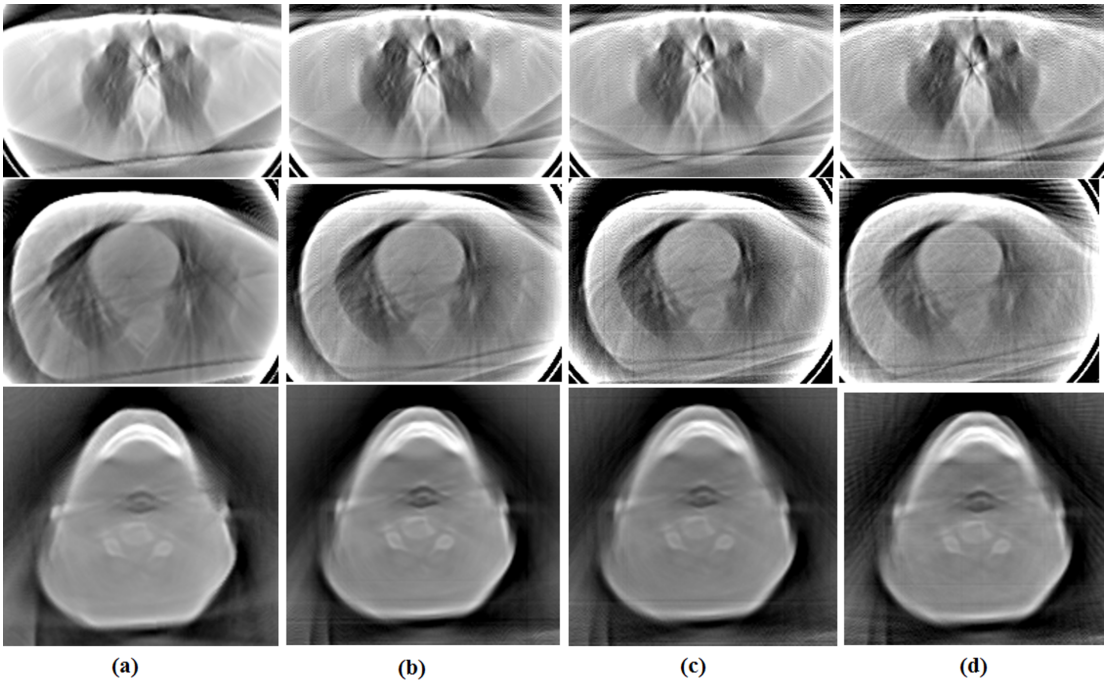

Figura 6.10: Imágenes reconstruidas: a) imágenes de referencia reconstruidas con FBP con 200 proyecciones; b), c), d) imágenes reconstruidas con LSQR con 200, 100 y 67 proyecciones respectivamente en la iteración 12 cuando se logra la tolerancia indicada.

6.2.3. Reconstrucción 3D

En este apartado se describe la aplicación de los algoritmos estudiados en el proceso de reconstrucción de imágenes en 3D. Una imagen en 3D se puede considerar como un conjunto de cortes horizontales y la reconstrucción de cada corte es un

proceso independiente. Por lo tanto, la reconstrucción en 3D puede ser realizada por un conjunto de hilos.
Las arquitecturas multi-core y multi-GPUs son más apropiadas para este tipo de tareas. Asignando un hilo a cada unidad GPU, se reconstruye la imagen capa por capa. El proceso de reconstrucción de una capa es independiente, por lo tanto, un hilo se ocupa por la reconstrucción de una capa de la imagen en la unidad GPU, traslada el resultado al 'host' y se ocupa de la siguiente capa. La Figura 6.11 representa en forma esquemática la utilización de un sistema multi-GPUs en la realización de esta tarea.

La combinación de OpenMP y el modelo de programación CUDA se utiliza para crear hilos en CPU y operar las unidades de procesamiento gráfico. El número de hilos creados es igual al número de las unidades gráficas en un sistema.

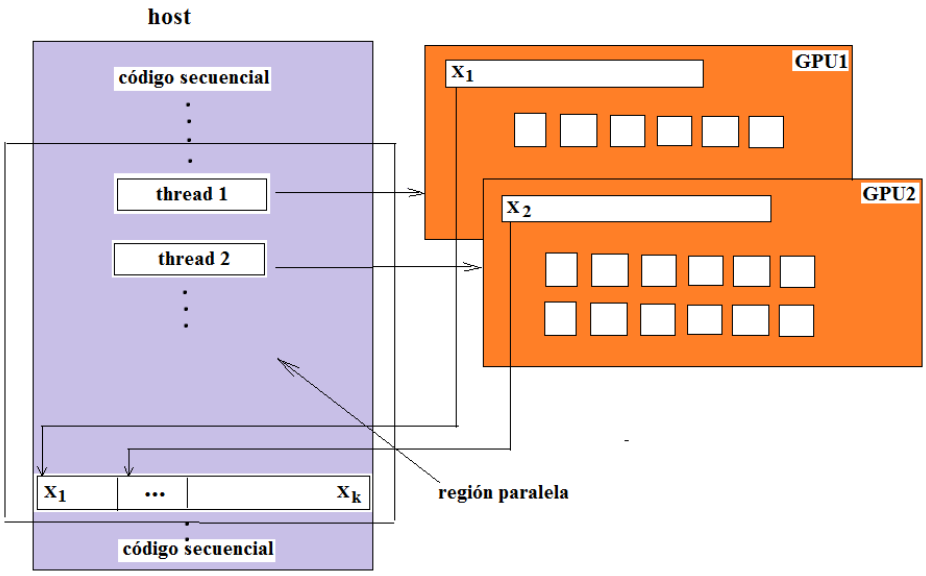

Figura 6.11: El esquema de utilización de GPUs en la reconstrucción de una imagen X_{nxn} de k cortes en 3D: las unidades GPU reconstruyen la imagen en 3D por cortes horizontales.

La comparativa de tiempos de reconstrucción de una imagen de 512x512 píxeles que consta de 50 cortes (512x512x50 vóxeles) por los algoritmos LSQR y SART se resume en tabla 6.3. La imagen se reconstruye en el sistema con 2 GPUs, por lo

tanto se reconstruyen 2 cortes simultáneamente.

algoritmo	512x512x2	512x512x50
LSQR	0.083 s	2 s
SART	0.524 s	13 s

Tabla 6.3: Tiempo de reconstrucción de una imagen de 2 (512x512x2) y 50 (512x512x50) cortes en el sistema con 2 GPUs después de 14 iteraciones.

Como un ejemplo de la utilización de la estrategia descrita, a continuación se presentan algunos resultados de la reconstrucción en 3D.
Las vistas sagital, coronal, axial y volumétrica de las imágenes reconstruidas por las proyecciones reales se presentan en las figuras 6.12 y 6.13 .

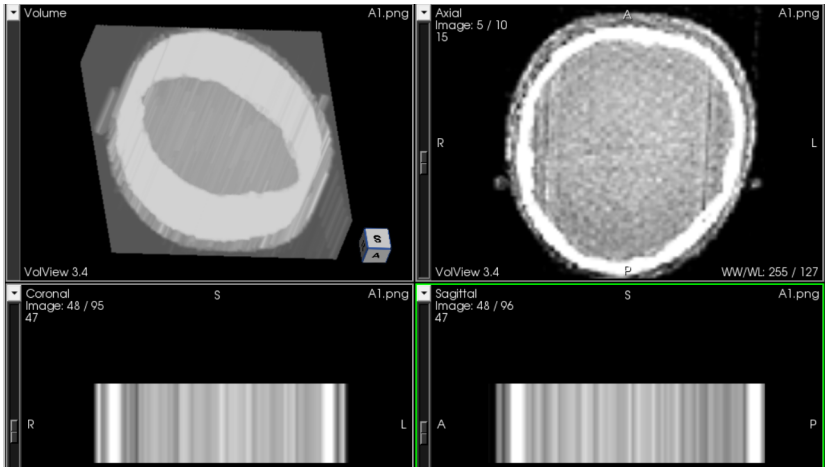

Figura 6.12: Reconstrucción en 3D con LSQR: imagen 1.

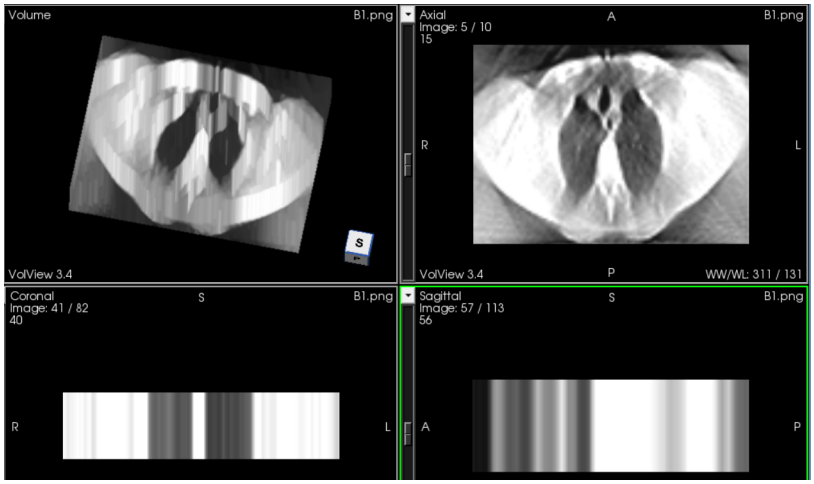

Figura 6.13: Reconstrucción en 3D con LSQR: imagen 2.

6.2.4. Conclusiones

El algoritmo de reconstrucción basado en GPUs muestra la capacidad del método iterativo de reconstruir imágenes con bajo coste computacional.
El modelo de programación CUDA junto con las librerías CUBLAS y CUSPARSE facilita la utilización de recursos computacionales de NVIDIA GPUs y proporciona una técnica eficiente de resolución de problemas computacionalmente complejos.

7 Reconstrucción de imágenes con un conjunto limitado de datos

7.1. Introducción

En medicina, el diagnóstico basado en tomografía axial computerizada es fundamental para la detección de anormalidades, basándose en la diferente atenuación de los rayos-X, estas anormalidades a menudo no se distinguen por los radiólogos. Al mismo tiempo, la excesiva exposición del paciente a la radiación no es deseada.

Los metodos iterativos se han convertido en un tópico de gran interés para muchos vendedores de sistemas de TAC clínicos por su capacidad de resolver el problema de reconstruccción con un número de limitado de proyecciones. Esto proporciona la posibilidad de reducir la dosis radiactiva en los pacientes durante el proceso de adquisición de datos. Por ejemplo, mirar [59]-[61]. Sin embargo, el proceso de reconstrucción sigue siendo computacionalmente muy intenso, especialmente en 3D. El alto costo computacional y la elevada dosis de radiación representan dos problemas principales en TAC.

Para resolver estos problemas en forma efectiva, hemos adaptado el método LSQR con el método de filtrado 'Soft Threshold Filtering' (STF) y el algoritmo de aceleración 'Fast Iterative Shrinkage-thresholding Algorithm' (FISTA) para TAC. La eficiencia y fiabilidad del método nombrado LSQR-STF-FISTA se presenta en este capítulo.

Una forma de reducir la dosis de radiación es disminuir el número de rotaciones durante la adquisición de datos. En consecuencia, en la reconstrucción aparecen

artefactos no deseados. Con la aparición y desarrollo de la teoría de 'compressed sensing' (compressed sensing theory) [63], [64], los algoritmos de reconstrucción basados en esta teoría han atraído mucha atención. Posteriormente, varios algoritmos han sido desarrollados y extendidos al campo de reconstrucción 'few-view CT image reconstruction'. Yu and Wang [65] adaptaron la técnica de soft-threshold filtering (STF) al metodo de minimización de variación total (TV) en la reconstrucción de imágenes. Con el objetivo de eliminar artefactos y preservar la estuctura de bordes, Yu and Zeng [66] desarrollaron un algoritmo iterativo basado en minimización de diferencias totales ponderadas ('weighted total difference (WTD) minimization) para 'few-view' computed tomography. Para resolver el problema de forma efectiva, ellos emplearon el método 'soft-threshold filtering' y 'fast iterative shrinkage thresholding algorithm' para acelerar la convergencia.

En los dos métodos, para actualizar la reconstrucción, se usa el método SART, descrito en las secciones anteriores. Como hemos mencionado, el alto coste computacional, en especial en reconstruction 3D, hace difícil el uso de SART en condiciones prácticas.

Inspirados por estos métodos y con el objetivo de reducir el costo y preservar la calidad de imágenes reconstruidas, nosotros proponemos el algoritmo Least Square QR combinado con las técnicas de 'soft threshold filtering' y aceleración. A continuación presentamos el analisis del método y resultados de la simulación numérica.

7.2. Métodos usados

El enfoque algebraico a la problema de reconstrucción se reduce al sistema de ecuaciones lineales:

$$Ax \cong p \tag{7.1}$$

comentado en el capítulo 5. Para encontrar las estimaciones de x de p, uno puede utilizar el método **LSQR** (sección 5.7) para minimizar la discrepancia $\|p - Ax\|$.
La reconstrucción con un número limitado de proyecciones es considerado como un problema mal condicionado (ill posed problem), la solución a (7.1) no es satis-

factoria y se requieren pasos adicionales para regularizar la solución. Daubechies y sus colaboradores describen un algoritmo general (general threshold algorithm) para resolver el problema inverso [67], [68]. Un enfoque aplicado a reconstrucciones TAC con un número limitado de proyecciones se presenta por Hengyong and Ge [69]. Ellos desarrollaron el algoritmo basado en la transformada de pseudo-inversa donde la idea principal consiste en realizar un filtrado no lineal antes de actualizar la reconstrucción.

Soft threshold filtering

Con el objetivo de eliminar los artefactos y preservar la estructura del contorno, hemos adaptado el método STF para la reconstrucción con un número limitado de proyecciones similar a las referencias [69], [70]. Para un vector-imágen x = { $x_j, j = 1 : N$ } de N píxeles, en la iteración k, el paso de filtrado para un píxel j se construye como sigue:

$$x_j = \frac{1}{4+4\alpha}(q(\omega, x_j, x_{j+N}) + q(\omega, x_j, x_{j+1}) + q(\omega, x_j, x_{j-1}) + q(\omega, x_j, x_{j-N}) + \alpha(q(\omega, x_j, x_{j+N+1}) + q(\omega, x_j, x_{j+N-1}) + q(\omega, x_j, x_{j-N-1}) + q(\omega, x_j, x_{j-N+1}), \tag{7.2}$$

donde:

$$q(\omega, y, z) = \left\{ \begin{array}{lll} (y+z), & if & |y-z| < \omega \\ y - \omega/2, & if & (y-z) \geq \omega \\ y + \omega/2, & if & (y-z) \leq -\omega, \end{array} \right\} \tag{7.3}$$

threshold $\omega = max_i|s_i|, s = A^T r$, $r = (p - Ax)$.

La técnica de aceleración: FISTA

Mientras STF ayuda a preservar la estructura de contorno del objeto, el algoritmo de aceleración FISTA introduce las nuevas direcciones de búsqueda de la solución que, usados por un 'solver' (LSQR o SART), aceleran la convergencia y mejoran la calidad de la imagen. La técnica de aceleración empleada en este trabajo es similar a ([71]) y, en una iteración k, es como sigue:

01: $h = x^k$
02: $t_{k+1} = \frac{1+\sqrt{1+4t_k^2}}{2}$
03: $x^{k+1} = h + \frac{t_k - 1}{t_{k+1}}(h - x_{temp})$
04: $x_{temp} = h,$

donde x^k es una solución obtenida por un proceso, t_k se inicia con el valor inicial igual a 1 y se actualiza por FISTA y x_{temp} se inicializa con la solución inicial.

LSQR-STF-FISTA

Nuestro método representa una combinación de LSQR, STF y FISTA. Como se mencionó antes, esta combinación acelera el proceso de convergencia y elimina artefactos en una imagen reconstruida. Una iteración de LSQR-STF-FISTA se presenta en el siguiente pseudocódigo:

(1) Inicialización:

$k = 1, x^k = 0, \alpha = 1, t_1 = 1, x_{temp} = 0$

(2) Comienza el proceso de iteración:

- Actualizar la reconstrucción x usando t iteraciones de LSQR.
- Realizar un paso de filtrado(STF) utilizando la ecuación (7.2).
- Aplicar u-veces la técnica de aceleración FISTA.
- Regresar al paso (2) hasta satisfaser el criterio de parada.

El algoritmo se detiene cuando las estimaciones $\| A^T r_k \| / \| A \| \| r_k \|$, con $r_k = p - Ax$, son menores que la tolerancia indicada: $atol = 1e - 06$.

El algoritmo permite implementar diferentes estrategias para lograr la eficiencia máxima. Por ejemplo, se puede usar t iteraciones de LSQR con una de STF y u iteraciones de FISTA. Adicionalmente, se puede elaborar un algoritmo que en función del residuo r_k, determine el número óptimo de iteraciones de cada proceso (LSQR y FISTA).

Adquisición de datos

Para determinar los valores óptimos de los parámetros del método LSQR-STF-

FISTA, hemos realizado la simulaciones con el fantoma FORBILD Head (Figura 7.1) que es considerado como un estándar adecuado ([72]).

Figura 7.1: Un corte típico del fantoma FORBILD Head

Las proyecciones sin ruido para el fantoma de 256x256 píxeles fueron generadas con 1025 detectores virtuales para 36 vistas cubriendo 360 grados alrededor del objeto para los siguientes ángulos:

$$\phi = \left\{ \begin{array}{cc} 10i-9, & 1 \leq i \leq 9 \\ 10i-5, & 9 < i \leq 18 \\ 10i-4, & 18 < i \leq 27 \\ 10i, & 27 < i \leq 36 \end{array} \right\} \tag{7.4}$$

7.3. Análisis de parámetros

A) Parámetro α

En el paso de filtrado, se mide la dispersidad vertical y horizontal, así como la continuidad diagonal entre puntos. El parámetro α determina la contribución de los píxeles diagonales al píxel central y juega el papel de balance entre la dispersidad y continuidad. El valor de $\alpha = 1$ significa los pesos iguales de los cuatro píxeles diagonales respecto los píxeles verticales y horizontales en su contribución al píxel central.

Para determinar los valores de α y el intervalo de filtrado óptimos, hemos medi-

do el índice de similitud estructural (SSIM) en las reconstrucciones con diferentes valores de α y diferentes intervalos de aplicación del filtro. Para llevar a cabo este análisis, el número máximo de iteraciones se fijó a 2000. La Figura 7.2 muestra que aplicando STF cada 8 iteraciones de LSQR el valor SSIM is 0.99 (máximo valor es 1).

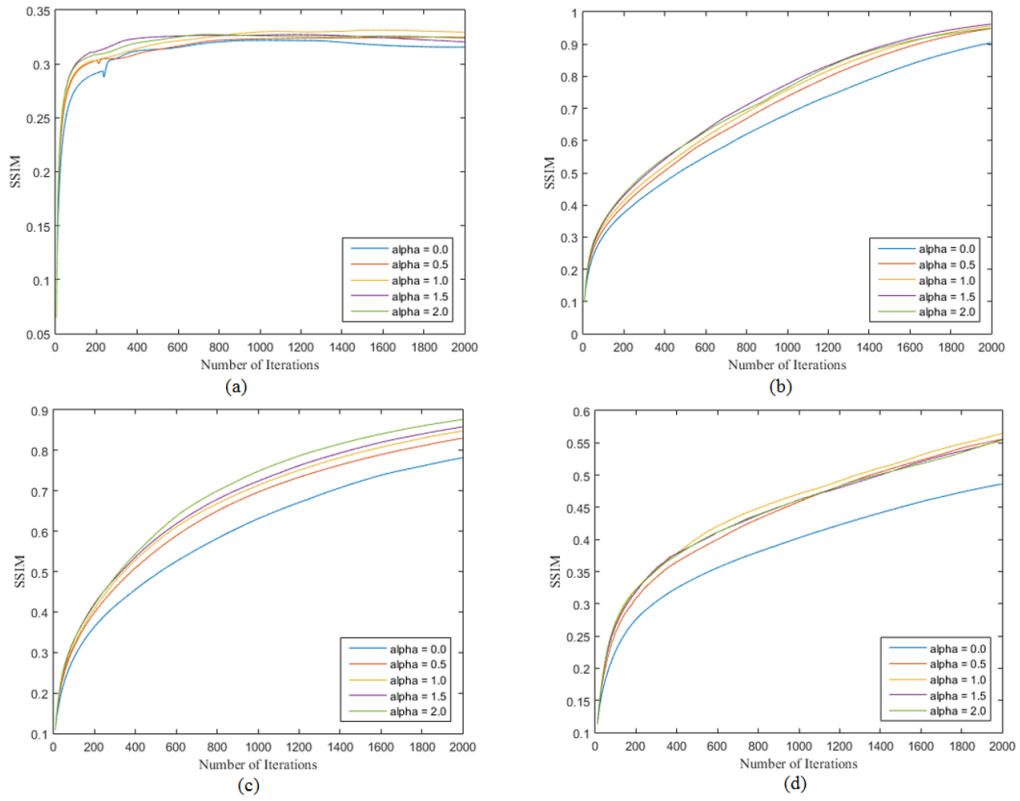

Figura 7.2: El valor del índice SSIM en las reconstrucción para valores de α en rango [0-2]. El filtro se aplica en las iteraciones 4, 6, 8, y 10 de LSQR en a), b), c) y d) respectivamente.

B) La técnica de aceleración FISTA

Para analizar la contribución de la técnica de aceleración FISTA, se selecionó el valor de $\alpha = 1$. STF y FISTA se aplican después de 2, 6, 8, y 10 iteraciones de LSQR. La Figura 7.3 muestra valores de SSIM en función de las iteraciones. Se ve que un intervalo corto (igual a 4) de aplicación de filtrado y aceleración no es recomendable. Aplicando 6 o 8 iteraciones de LSQR seguidos con una aplicación de

STF y FISTA se llegan a altos valores de SSIM con pocas iteraciones.

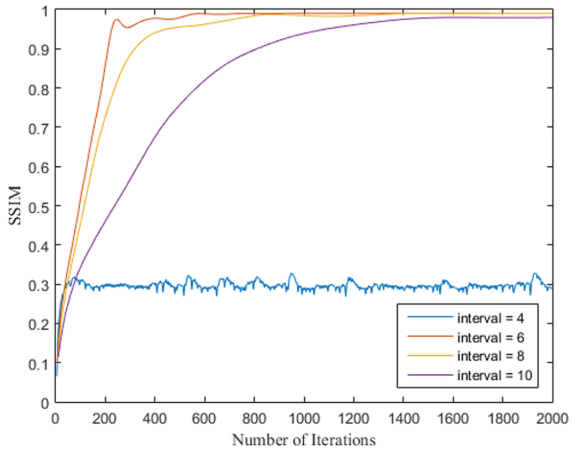

Figura 7.3: El valor del índice SSIM en las reconstrucciones aplicando la aceleración después de 4, 6, 8, y 10 iteraciones de LSQR.

Después de los estudios realizados, hemos concluido que la estrategia óptima del algoritmo LSQR-STF-FISTA es la siguiente: 6 iteraciones de LSQR seguidos por una iteración de STF y una de FISTA. Con esta estrategia, se llega a una solución aceptable después de 200 iteraciones y después de 800 iteraciones se obtiene el valor del índice $SSIM > 0{,}9$.

7.4. Resultados y discusión

Los algoritmos fueron implementados utilizando el modo de programación CUDA y los cálculos llevaron a cabo en el sistema gpu.dsic.upv.es.

Para evaluar los algoritmos LSQR, LSQR-STF y LSQR-STF-FISTA, se define el criterio de parada como límite de tolerancia atol = 1.e-6. La Figura 7.4 muestra las imágenes de 256x256 píxeles reconstruidas con estas técnicas con 36 proyecciones después de diferente número de iteraciones. Con el método LSQR la convergencia no se logra antes de 2000 iteraciones. Añadiendo STF, aumentando las iteraciones, se reducen los artefactos, el proceso converge, y la imagen se reconstruye en 4000 iteraciones. La combinación de 6 iteraciones de LSQR, una de STF y una de FISTA, acelera la convergencia reduciendo el número de iteraciones a 1200 con la calidad

de imagen perfecta.

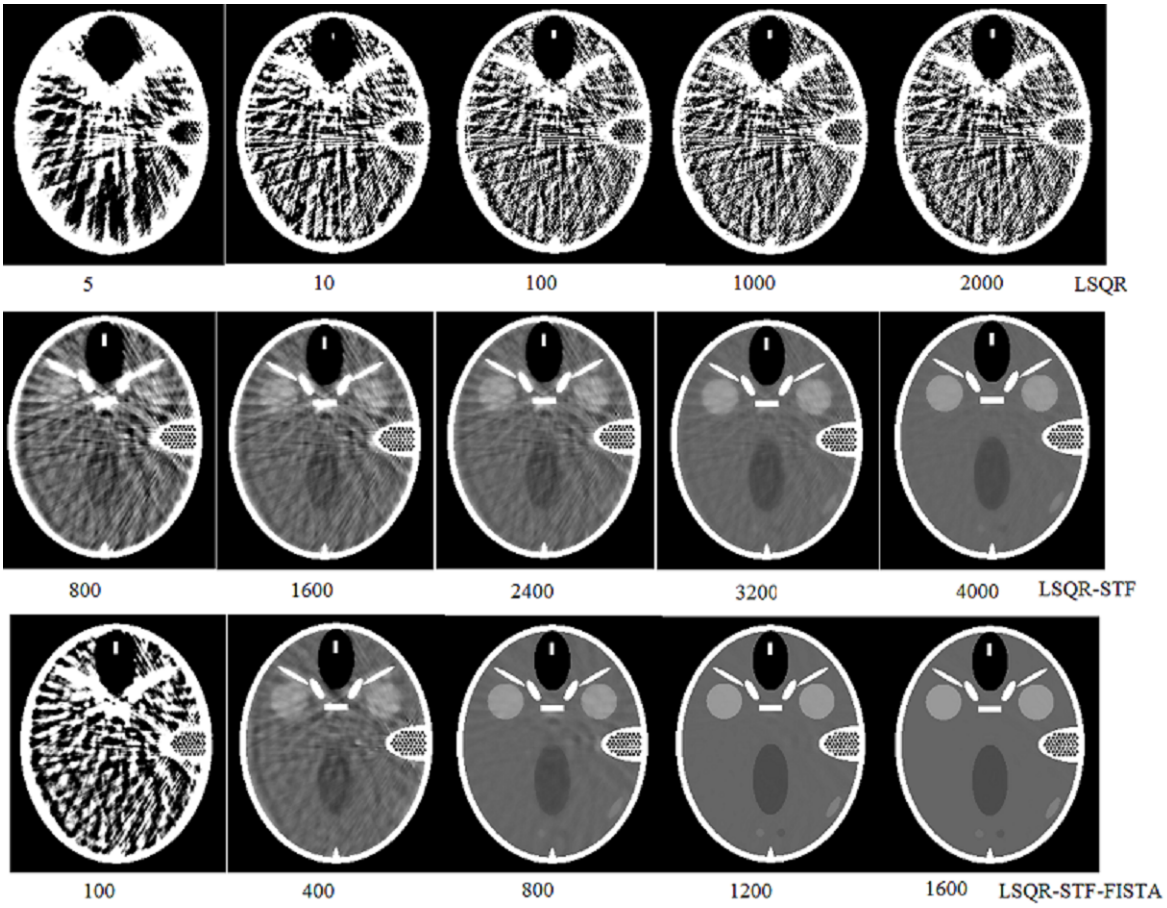

Figura 7.4: Imágenes reconstruidas por (a) LSQR, (b) LSQR-STF, (b) LSQR-STF-FISTA por 36 proyecciones. Debido a la diferencia en la convergencia de los métodos, las imágenes se presentan después de diferentes iteraciones para mejor visualización.

Para una mayor evaluación de los algoritmos, se realizó un análisis cuantitativo de las imágenes reconstruidas utilizando las siguentes medidas: MAE, MSE, PSNR y SSIM. Los resultados se resumen en la tabla 7.1. Se puede observar que el método LSQR-STF-FISTA supera a los otros dos métodos. El índice SSIM es igual 0.99, lo que indica que la similitud entre la imagen reconstruida y de referencia es casi perfecta.

Medidas	LSQR	LSQR-STF	LSQR-STF-FISTA
MAE	0.116859	0.020911	0.000231
MSE	0.029729	0.001484	0.000000
PSNR	21.577696	33.772536	71.943127
SSIM	0.217657	0.749491	0.999791

Tabla 7.1: Comparación cuantitativa de las reconstrucciones con diferentes algoritmos después de 1000 iteraciones.

7.5. Aplicación a un caso real

El método propuesto (LSQR-STF-FISTA) fue aplicado a las proyecciones reales recogidas por el CT escáner del Hospital Clínic Universitari of València. El escáner es CT simulator Metaserto con el sistema tomográfico Kermath.
El conjunto inicial de proyecciones fue recogido con el escáner con 512 detectores en el rango 0-180 grados con 0.9 grados del paso angular. Para evaluar el algoritmo con un menor número de proyecciones, tres subconjuntos de proyecciones fueron obtenidos del conjunto inicial: un conjunto completo de 200 proyecciones y dos subconjuntos de 100 y 67 proyecciones.

Las imágenes reconstruidas con LSQR y LSQR-STF-FISTA fueron comparadas con la imagen reconstruida por el algoritmo analítico clásico FBP. La Figura 7.5 muestra el error en función del número de iteraciones. El error disminuye rápido y después de 12 iteraciones se mantiene casi constante. En realidad, con 12-14 iteraciones se obtiene la imagen con la calidad comparable a la reconstrucción de FBP.

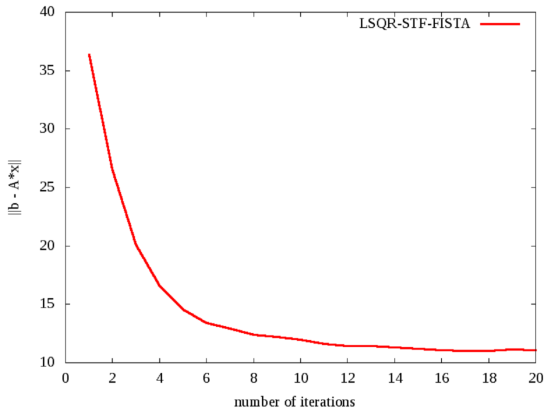

Figura 7.5: Norma de error como función de iteraciones.

La Figura 7.6 muestra las reconstrucciones con diferentes métodos. La reconstrucción con FBP se realizó con el conjunto completo de proyecciones (200). Las reconstrucciones con LSQR y LSQR-STF-FISTA se realizaron con 67, 100 y 200 proyecciones y se presentan después de las 13 iteraciones. En LSQR-STF-FISTA se usó la combinación óptima descrita anteriormente. Se puede observar que la técnica mejora la calidad de la imagen. Adicionalmente, la técnica LSQR-STF-FISTA es capaz de reconstruir la imagen con el conjunto menor de proyecciones (67) con calidad comparable a la reconstrucción con FBP. En el experimento, el número de proyecciones fue reducido a 67, lo que significa una disminución de la dosis de radiación en pacientes de 3 veces.

Comparando LSQR y SART, hemos deducido que el coste por iteración de estos algoritmos es aproximadamente el mismo. Sin embargo, la comparación visual de las reconstrucciones después de 13 iteraciones presentan diferencias significativas. La Figura 7.7 muestra que mientras las reconstrucciones hechas con LSQR-STF-FISTA, después de las 13 iteraciones son comparables con las reconstrucciones FBP, el algoritmo SART necesita al menos 200 iteraciones para obtener resultados semejantes. En consecuencia, el coste computacional de SART para reconstruir una imagen es elevado.

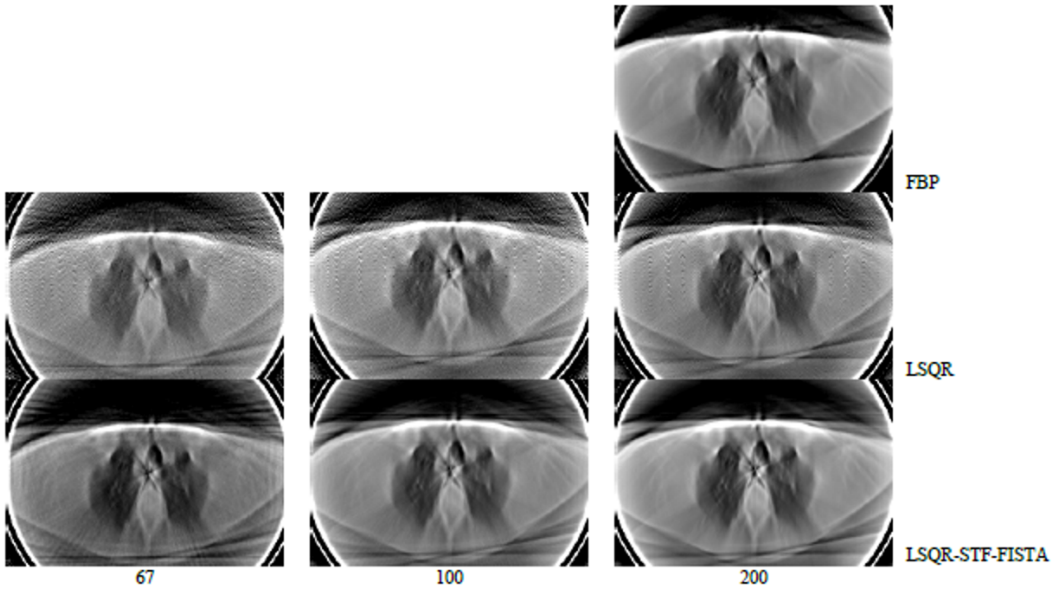

Figura 7.6: Las imágenes reconstruidas por FBP, LSQR y LSQR-STF-FISTA por 67, 100 y 200 proyecciones. Las reconstrucciones de LSQR y LSQR-STF-FISTA se presentan después de 13 iteraciones.

7.6. Optimización del algoritmo

Como hemos mencionado previamente, el mayor coste computacional de LSQR y SART se presenta en las operaciones como el producto matriz-vector, suma de los elementos de filas o columnas de la matriz. Para optimizar estas operaciones, se usaron formatos compactos CSR y CSC de la matriz para realizar 'row and column driven' operations.

Otra forma utilizada de optimizar las operaciones básicas, como las operaciones

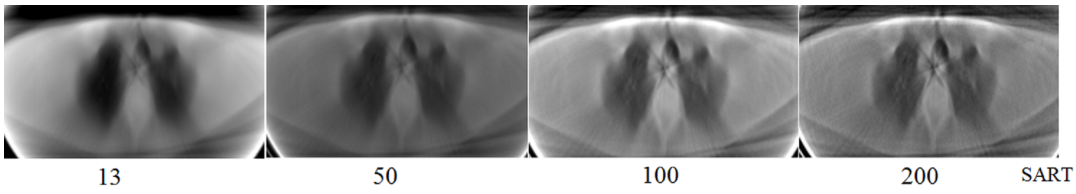

Figura 7.7: Las imágenes reconstruidas por SART después con 67 proyecciones después de 13, 50, 100 y 200 iteraciones.

matriz-vector, es emplear las funciones de las librerías CUSPARSE y CUBLAS.

Y un punto más se presenta en la parte de filtrado de la solución. En este paso, el valor del píxel j se actualiza por la técnica STF (ver la ecuación (7.2)), donde participan 8 píxeles vecinos como se ve en la Figura 7.8).

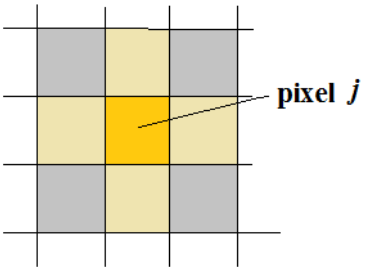

Figura 7.8: En STF ocho píxeles vecinos contribuyen al valor de píxel j.

En este caso, el acceso a la memoria lineal a traves de texturas, presenta otra forma de optimización del algoritmo. Hemos analizado dos formas de acceso a la memoria de texturas: una a través de la referencia a texturas (Texture Reference), y la otra, a través del objeto de texturas creado (Bindless Texture) utilizando diferentes tamaños de bloques. En la tabla 7.2, se resumen el tiempo (en ms) de la operación de filtrado para una imagen de 256x256 píxeles realizando differente accesos a la memoria global: directamente a la memoria global, acceso a través de la referencia a texturas, y por medio del objeto de texturas creado previamente. Se puede observar que utilizando objetos de textura, se logra reducir el tiempo de filtrado utilizando los bloques de tamaño 16x16 hilos por bloque.

Para la evaluación de la efectividad de acceso a la memoria de texturas, en la Figura 7.9 se presenta la comparativa de los cálculos de Ancho de Banda Efectivo (Effective Bandwidth) (en GB/s) para 'Texture Reference' y 'Bindless Texture' durante la operación de filtrado. Se observa que utilizando el acceso a la memoria de texturas a través de objeto de texturas (Bindless Texture) con el tamaño de bloques de 16x16 se consigue el procesamiento de filtrado más efectivo (29.13 GB/s).

BlockSize	Global Memory	Texture Reference	Bindless Texture
4x4	0.432	0.029	0.069
8x8	0.133	0.029	0.025
16x16	0.042	0.030	0.018
16x32	0.028	0.028	0.021
32x32	0.033	0.029	0.030

Tabla 7.2: Tiempo (en ms) de filtrado con diferentes accesos a la memoria para una imagen de 256x256 píxeles.

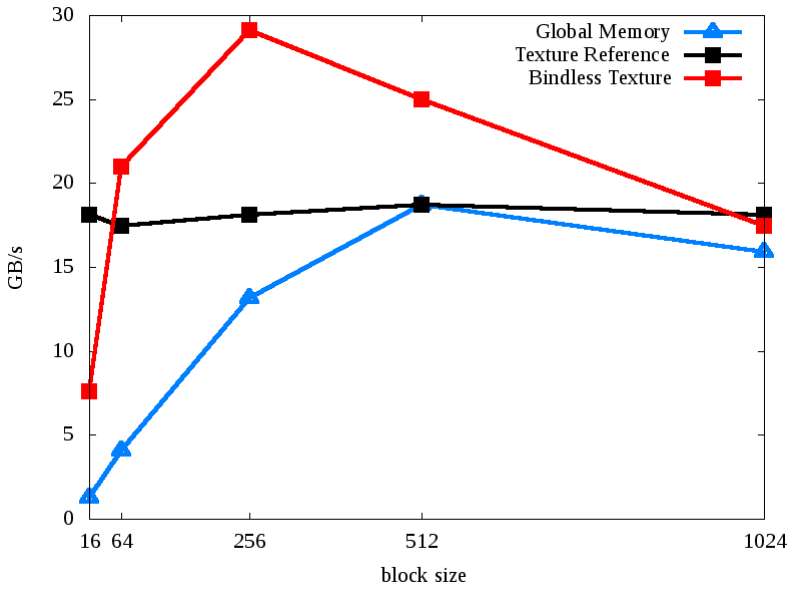

Figura 7.9: Comparación de ancho de banda efectivo durante la operación de filtrado utilizando accesos a la memoria de texturas a través de 'Texture Reference' y 'Bindless Texture' con diferentes tamaños de bloques.

7.7. Conclusiones

Hemos analizado la aplicación del método LSQR en la reconstrucción de imágenes con un menor número de proyecciones. Los resultados muestran que LSQR combinado con las técnicas de filtrado STF y aceleración FISTA reconstruye imágenes

con un conjunto incompleto de proyecciones sin deteriorar la calidad de la imagen.

En esta tesis hemos comprobado que el método LSQR-STF-FISTA reconstruye la imagen del fantoma FORBILD con 36 proyecciones. La posibilidad de combinar LSQR, STF y FISTA permite implementar diferentes estrategias y elegir una óptima.

Se mostró que el método puede ser aplicado a una imagen real logrando un buen resultado con 13 iteraciones utilizando 67 proyecciones. La posibilidad de reconstruir una imagen con un menor número de proyecciones permite reducir el tiempo de adquisición de datos, así como reducir la dosis de radiación en los pacientes.

El método propuesto puede ser implementado en aplicaciones prácticas (por ejemplo, en escáneres portátiles) porque el método no necesita un conjunto completo de datos, tiene un costo computacional bajo y es capaz de reconstruir imágenes de buena calidad a partir de proyecciones con ruido.

Finalmente, el método puede ser extendido a la reconstrucción 3D donde la cantidad de cómputo es enorme.

8 Conclusiones y trabajos futuros

8.1. Conclusiones

En esta tesis hemos analizado métodos de reconstrucción analítica e iterativa de imágenes TAC por proyecciones. El estudio, análisis de la implementación secuencial y paralela de los algoritmos de reconstrucción y los resultados obtenidos nos llevan a las siguentes conclusiones:

- Los algoritmos analíticos basados en la Transformada de Fourier son rápidos y permiten la reconstrucción de imágenes de buena calidad en condiciones óptimas de las proyecciones.
- En condiciones de ruido y en caso de un conjunto de datos incompleto, los métodos iterativos son superiores a los métodos analíticos.
- Los algoritmos de reconstrucción son paralelizables y permiten explotar las características de las arquitecturas modernas de sistemas como sistemas multi-cores y many-cores
- La paralelización de los algoritmos permite la reducción de tiempo de reconstrucción de imágenes TAC.
- En sistemas multi-cores, los tiempos óptimos se consiguen al llevar a cabo el proceso de reconstrucción en forma paralela con el número de procesos iguales al número de núcleos o procesadores del sistema.
- En métodos iterativos, la matriz del sistema contiene la información de la imagen a reconstruir y la forma de generar la matriz afecta a la calidad de la imagen.

- La estructura de bloques simétricos en la matriz de sistema permite generar y almacenar 1/8 parte de toda la matriz y reutilizarla en el proceso de reconstrucción extendiéndola al rango completo de 0-360 grados. De esta forma, se puede reducir la utilización de memoria del sistema y el tiempo de reconstrucción.

- El conjunto de datos en el proceso de reconstrucción es muy grande, especialmente en 3D y 4D. Por esta razón y porque las operaciones básicas de computo son de tipo píxel/vóxel, las arquitecturas modernas masivamente paralelas, basadas en GPUs, son las más apropiadas.

- Comparando los métodos iterativos clásico SART y LSQR, se observa que el número de operaciones en el proceso de reconstrucción de una imagen es del mismo orden. Sin embargo, SART requiere muchas más iteraciones para la reconstrucción comparado con LSQR. En consecuencia, el tiempo de reconstrucción se eleva.

- El método iterativo LSQR es capaz de reconstruir la imagen con pocas iteraciones y tiene un costo muy bajo.

- En esta tesis se propone el método LSQR combinado con la técnica de filtración STF y aceleración FISTA para la reconstrucción de imágenes. Se muestra que la combinación óptima de estos métodos permite preservar la estructura del objeto y acelerar el proceso de reconstrucción.

- El método LSQR-STF-FISTA es eficaz en resolver el problema de reconstrucción con un menor número de proyecciones. La reducción de vistas en el proceso de adquisición de datos es importante en TAC porque permite reducir la dosis de radiación a pacientes.

- La capacidad de los métodos iterativos de reconstruir una imagen con un conjunto incompleto de proyecciones puede ser utilizada en la práctica en escáneres portátiles que pueden ser usadas en condiciones extremas.

8.2. Trabajos futuros

Los avances tecnológicos posibilitan el desarrollo constante de algoritmos existentes y aparición de algoritmos nuevos de reconstrución de imágenes. Como muestran los resultados obtenidos (e.g. Tabla 5.1), el tamaño de la matriz del sistema de ecuaciones aumenta con el mayor número de proyecciones. Es decir, el tamaño de la matriz es proporcional a la resolución de la imagen a reconstruir, o, lo que es equivalente, al número de detectores en el escáner, o al número de ángulos bajo los cuales se toman las proyecciones.

En consecuencia, crece el tiempo de ejecución y también la necesidad de mayores recursos de memoria del sistema. La reconstrucción de imágenes en 3D o 4D es un proceso que consume mucho tiempo y opera con gran cantidad de datos lo que sigue siendo un reto en la actualidad. Para abarcar estos problemas se plantean las siguentes ideas para el futuro desarrollo:

- Comparar los método estudiados con otros métodos de reconstrucción, en particular, con el método multigrid para el postprocesamiento de las imágenes.
- En esta tesis se ha trabajado con el método de Siddon para generar la matriz del sistema. Investigar otros métodos de generación de la matriz con el objetivo de minimizar los recursos del sistema nesesarios y mejorar la calidad en la imagen reconstruida.
- Investigar la posibilidad de la reconstrucción con menor número de proyeciones, en particular, estudiar la aplicación de la teoría de 'compressed sensing theory' en la reconstrucción de imágenes.
- La reconstrucción con menor número de proyecciones da lugar a la aparición de artefactos en la imagen. Estudiar las posibilidades de reducir los artefactos y mejorar la calidad de las imágenes reconstruidas.
- Comprobar el comportamiento del método de LSQR al añadir diferentes tipos de ruido en las proyecciones.
- Explorar las arquitecturas many-cores en la reconstrucción paralela de imágenes en 3D y 4D.

Publicaciones

Los resultados de esta tesis han dado lugar a los artículos donde se describen las técnicas utilizadas.

- **Liubov Flores**, Vicente Vidal, Estíbaliz Parcero, and Gumersindo Verdú. *Application of LSQR and Thresholding for CT Imaging with Few Projections.* Journal of Health & Medical Informatics. 2015. In revision.
- V. Vidal, E. Parcero, **L. Flores**, M.G. Sánchez, G. Verdú. *Impact of View Reduction in CT on Radiation Dose for Patients.* Journal Radiation Physics and Chemistry. 2015. In revision.
- **Liubov Flores**, Vicent Vidal and Gumersindo Verdú. *System Matrix Analysis for Computed Tomography Imaging.* 2015. Journal PLOSOne. In print.
- **Liubov A. Flores**, Vicent Vidal and Gumersindo Verdú. *Parallel CT image reconstruction based on GPUs.* Radiation Physics and Chemistry. Vol. 95(1): 247-250, 2014. doi:10.1016 / j.radphyschem.
- **Liubov A. Flores**, Vicent Vidal and Gumersindo Verdú. *Iterative Reconstruction from Few-View Projections.* INTERNATIONAL CONFERENCE ON COMPUTATIONAL SCIENCE 2015. PROCEDIA CS. Vol. 51: 703-712, 2015.
- Vicent Vidal, **Liubov Flores**, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú. *Comparación de Métodos Iterativos de Reconstrucción de Imágenes TAC.* REVISTA DE LA SOCIEDAD NUCLEAR ESPAÑOLA (ISSN 1575-3204). Vol. 1: 113-118, 2014.
- **Liubov A. Flores**, Vicent Vidal and Gumersindo Verdú. *CT Image reconstruction based on GPUs.* INTERNATIONAL CONFERENCE ON COMPU-

TATIONAL SCIENCE 2013. PROCEDIA CS.(ISSN 1877-0509). Vol. 18: 1412-1420, 2013.

- Vicent Vidal, **Liubov Flores** y Gumersindo Verdú. *Una implementación Eficiente del Proceso de Reconstrucción Iterativa de Imágenes CT basada en GPUs*. REVISTA DE LA SOCIEDAD NUCLEAR ESPAÑOLA (ISSN 1575-3204). Vol. 1: 21-28, 2013.

- **Liubov Flores**, Vicent Vidal, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú. *Iterative Reconstruction of CT Images on GPUs*. Proceedings of the 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2013)(ISSN 9781457702150). Pag: 5143-5146, 2013.

- Vicent Vidal, **Flores Liubov A.**, Verdú Martín Gumersindo Jesús, Mayo Nogueira Patricia, Rodenas Escriba Francisco De Asís. *Algoritmo Paralelo de Reconstrucción de Imágenes TAC*. REVISTA DE LA SOCIEDAD NUCLEAR ESPAÑOLA (ISSN 1137-2885). Vol. 1: 1-5, 2012.

- **Liubov Flores**, Vicent Vidal, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú. *Fast Parallel Algorithm for CT Image Reconstruction*. Proseedings of the 34th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC 2012) (ISSN 978-1-4244-4120-4). Vol. 1: 1502-1505, 2012.

- **L.A. Flores**, V. Vidal Gimeno, P. Mayo Nogueira, F. Rodenas Escriba, G. Verdú Martín. *Una implementación paralela eficiente de la transformada de Radón Inversa para la reconstrucción de imágenes médicas*. Revista Universitaria de la Universidad Nacional de Ingeniería, Lima-Perú, 2012.

- **Liubov Flores**, Vicent Vidal, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú. *Contrast of two methods of reconstruction of CT images using high performance computing*. Mathematical Modelling in Engineering & Human Behaviour 2011 (ISSN978-84-695-2143-4). Pag. 27-30, 2011.

- **Flores Liubov A.** Vicent Vidal, Mayo Nogueira Patricia, Ródenas Escribá Francisco De Asís, Verdú Martín Gumersindo Jesús. *Iterative reconstruction of*

CT images with PETSc. Proceedings of the 4th International Conference on BioMedical Engineering and Informatics and the 4th International Congress on Image and Signal Processing. BMEI '11 - CISP ' 11.(ISSN 978-1-4244-9350-0). Pag: 343-346, 2011.

- Vicent Vidal, **Flores Liubov**, Patricia Mayo, Rodenas Francisco, Verdú Martín Gumersindo. *Reconstrucción Iterativa de Imágenes de TAC Mediante Computación de Altas Prestaciones*. REVISTA DE LA SOCIEDAD NUCLEAR ESPAÑOLA (ISSN 1575-3204). Vol. 1: 106-112, 2011.

Bibliografía

[1] Stanley R. Deans. *The Radon Transform and Some of Its Applications.* Dover Publications, INC. Mineola, New York, 2007.

[2] Will A. Kalender. *Computed Tomography.* Publicis Corporate Publishing, Erlangen, Germany, 2005.

[3] Rafael C. Gonzáles, Richard E. Woods. *Digital Image processing.* Prentice Hall, 3rd edition, 2008.

[4] A. C. Kak and Malcolm Slaney, Principles of Computerized Tomographic Imaging, Society of Industrial and Applied Mathematics. chapter 07, 2001.

[5] Herman, G. T. *Fundamentals of computerized tomography: Image reconstruction from projections.* 2nd ed. Springer, 2009.

[6] Gordon, R; Bender, R; Herman, GT. *Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and x-ray photography.* Journal of Theoretical Biology. Vol. 29(3):471-81, 1970.

[7] Geman, S., McClure, D. E. *Statistical methods for tomographic image reconstruction.* Bull Int Stat Inst. Vol. 52(4):521, 1987.

[8] De Man, B. *Iterative Reconstruction for Reduction of Metal Artifacts in Computed Tomography.* ISBN 90-5682-300-0, 2001.

[9] Langan, D.; Claus, B.; Edic, P.; Vaillant, R.; De Man, B.; Basu, S.; Iatrou, M. *An iterative algorithm for soft tissue reconstruction from truncated flat panel projections.* Medical Imaging: Physics of Medical Imaging. Edited by Flynn, Michael J.; Hsieh, Jiang. Proceedings of the SPIE. Vol. 6142: 749-757, 2006.

[10] K. Lange and R. Carson. EM reconstruction algorihtms for emission and transmission tomography. J. Comput. Assist. Tomogr. Vol. 8:306-316, 1984.

[11] G. Wang, H.Yu, and B. De Man. *An outlook on X-ray CT research and development*. Medical Physics. Vol. 35(3):1051-1064, 2008.

[12] B. M Crawford and G. T Herman. *Low-dose, large-angled cone-beam helical CT data reconstruction using algebraic reconstruction techniques*. Image and Vision Comp. Vol. 25:78-94, 2007.

[13] J. Nuyts, B. De Man, P. Dupont, M. Defrise, P. Suetens, and L. Mortelmans. *Iterative reconstruction for helical CT : A simulation study*. Phys. Med. Biol. Vol. 43:729-737, 1998.

[14] G. Wang, M. W. Vannier, and P. C. Cheng. *Iterative X-ray cone beam tomography for metal artifact reduction and local region reconstruction*. Microscopy and Microanalysis. Vol. 5(1):58-65, 1999.

[15] A. H. Andersen. *Algebraic reconstruction in CT from limited views*. IEEE Trans. Med. Imaging. Vol. 8(1), 1989.

[16] G. Wang, D. Snyder, J. O'Sullivan, and M. Vannier. *Iterative deblurring for CT metal artifact reduction*. IEEE. Trans. Med. Imaging. Vol. 15(5):657-663, 1996.

[17] N. Sinha and J. T. W. Yeow. *Carbon nanotubes for biomedical applications*. IEEE Trans. Nano. Vol. 4(2):180-196, 2005.

[18] J. Ni, X. Li, T. He, and G. Wang. *Review of parallel computing techniques for computed tomography image reconstruction*. Current Med. Imaging Reviews. Vol. 2(4):405-414, 2006.

[19] Stone, S., Haldar, J., Tsao, S., Hwu, W., Sutton, B., & Liang, Z. *Accelerating advanced MRI reconstructions on GPUs*. Journal of Parallel and Distributed Computing. Vol. 68(10): 1307-1318, 2008.

[20] Johnson, C., & Sofer, A. *A data-parallel algorithm for iterative tomographic image reconstruction*. Frontiers of Massively Parallel Computation. pp 126-137, 1999.

[21] Pratx, G., Chinn, G., Olcott, P., & Levin, C. *Fast, Accurate and Shift-Varying Line projections for Iterative Reconstruction Using the GPU.* IEEE Transactions on Medical Imaging. Vol 28(3): 435-445, 2009.

[22] Jang, B., Kaeli, D., Do, S.,& Pien, H.*Multi GPU implementation of iterative tomographic reconstruction algorithms.* Biomedical Imaging: From Nano to Macro. pp 185-188, 2009.

[23] Flores, Liubov A.; Vicent Vidal; Mayo Nogueira, Patricia; Ródenas Escribá,Francisco De Asís; Verdú Martín, Gumersindo Jesús;*Iterative reconstruction of CT images with PETSc.* The 4th International Conference on BioMedical Engineering and Informatics. The 4th International Congress on Image and Signal Processing. (ISSN 978-1-4244-9350-0). pp 343-346, 2011.

[24] D. J. Brenner, E. J. Hall, and D. Phil. Computed tomography: An increasing source of radiation exposure. N. Engl. J. of Med. Vol. 357:2277-84, 2007.

[25] E. Stephen, P. F. Butler, K. E. Applegate, S. B. Birnbaum, L. F. Brateman, J. M. Hevezi, F. A. Mettler, R. L. Morin, M. J. Pentecost, G. G. Smith, K. J. Strauss, and R. K. Zeman. American college of radiology white paper on radiation dose in medicine. J. Am. Coll Radiol . Vol. 4:272-284, 2007.

[26] Jersey Chen, Andrew J. Einstein, Reza Fazel, Harlan M. Krumholz,Yongfei Wang, Joseph S. Ross, Henry H. Ting,Nilay D. Shah, y Brahmajee K. Nallamothu. *Cumulative Exposure to Ionizing Radiation from Diagnostic and Therapeutic Cardiac Imaging Procedures: A Population-Based Analysis.* J Am Coll Cardiol. Vol. 56(9): 702-74, 2010.

[27] A. B. de Gonzalez and S. Darby. *Risk of cancer from diagnostic X-rays: estimates for the UK and 14 other countries.* The Lancet, Vol. 363(9406): 345-351, 2004.

[28] D. J. Brenner and E. J. Hall. *Current concepts:computed tomography-an increasing source of radiation exposure.* The New England Journal of Medicine. Vol. 357(22):2277-2284, 2007.

[29] O. W. Linton and F. A. Mettler Jr. *National conference on dose reduction in CT, with an emphasis on pediatric patients.* American Journal of Roentgenology. Vol. 181, no. 2, pp. 321-329, 2003.

[30] Gitta Kutyniok. *Theory and Applications of Compressed Sensing.* GAMM-Mitteilungen, 10 July 2013.

[31] G.H. Chen, J. Tang, and S. Leng. *Prior image constrained compressed sensing (PICCS): a method to accurately reconstruct dynamic CT images from highly undersampled projection data sets.* Medical Physics. Vol. 35(2): 660-663, 2008.

[32] H. Yu and G. Wang. *Compressed sensing based interior tomography.* Physics in Medicine and Biology. Vol. 54(9): 2791-2805, 2009.

[33] L. I. Rudin, S. Osher, and E. Fatemi. *Nonlinear total variation based noise removal algorithms.* Physica D. Vol. 60(4): 259-268, 1992.

[34] G. T. Herman and R. Davidi. *Image reconstruction from a small number of projections.* Inverse Problems. Vol. 24(4), Article ID 045011, 17 pages, 2008.

[35] G. Wang and M. Jiang. *Ordered-subset simultaneous algebraic reconstruction techniques (OS-SART).* Journal of X-Ray Science and Technology. Vol. 12(3): 169-177, 2004.

[36] Liubov Flores, Vicente Vidal, Estíbaliz Parcero, and Gumersindo Verdú. *Application of LSQR and Thresholding for CT Imaging with Few Projections.* PLOSONE, in revision.

[37] M.J. Flynn. *Some computer organisations and their effectiveness*, IEEE Trans. On Computers, Vol. c-21(9): 948-960, 1972.

[38] Aad J. van der Steen. *Overview of recent supercomputers.* HPC Research, August 2008.

[39] http://www.intel.com/es ES/products/server/processor/index.htm

[40] http://www.amd.com/us/products/desktop/processors/Pages/desktop-processors.aspx

[41] NVIDIA. *NVIDIA CUDA Programming Guide*, Version 2.0, 06/07/2008.

[42] http://www.nvidia.com/content/PDF/fermiwhite papers/NVIDIA Fermi Compute Architecture Whitepaper.pdf

[43] http://www.nvidia.es/object/geforce family es.html

[44] http://users.dsic.upv.es/ jroman/kahan.html

[45] L.Dagum, R.Menon. *OpenMP: an industry standard API for shared-memory programming*. Computational Science & Engineering, IEEE.

[46] http://www.mcs.anl.gov/research/projects/mpi/

[47] http://acts.nersc.gov/petsc/index.html

[48] Cuda C Programming Guide. Downloaded in Oct. 2012 from URL http://docs.nvidia.com/cuda/index.html.

[49] CUBLAS Library. Downloaded in Oct. 2012 from URL http://docs.nvidia.com/cuda/index.html.

[50] CUSPARSE Library. Downloaded in Oct. 2012 from URL http://docs.nvidia.com/cuda/index.html

[51] http://www.netlib.org/blas/

[52] http://fftw.org/

[53] R. L. Siddon, *Fast calculation of the exact radiological path for a threedimensional CT array*. Med Phys. Vol. 12: 252-255, 1986.

[54] Cibeles Mora Mora. *Métodos de reconstrucción volumétrica algebraica de imágenes tomográficas*. Tesis PhD, 2008.

[55] Andersen, A. & Kak, A. *Simultaneous algebraic reconstruction technique (SART): A superior implementation of the ART algorithm*. Ultrasonic Imaging. Vol. 6: p. 81-94, 1986.

[56] *Maximum Likelihood Reconstruction for Emission Tomography*. L.A. Shepp and Y. Vardi. IEEE Transactions on Medical Imaging. Vol. 1(2), 1982.

[57] C.C. Paige and M.A. Saunders. *The Algorithm LSQR:Sparse linear equations and least square problems.* ACM Trans. Math. Soft.Vol. 8(2), 1982.

[58] G.H. Golub and W. Kahan. *Calculating the singular values and pseudoinverse of a matrix.* SIAM J. Numer. Anal.Vol. 2: 205-224, 1965.

[59] Beister, M., Kolditz, & Kalender, W. *Iterative reconstruction methods in X-ray CT.* Physica Medica-European Journal of Medical Physics. Vol. 28: 94-108, 2012.

[60] Zhao, X., Hu, J., & Yang, Y. *GPU based iterative cone-beam CT reconstruction using empty space skipping technique.* Journal of X-Ray science and Technology. Vol. 21: 53-69, 2013.

[61] Flores, L., Vidal, V., Mayo, P., Rodenas, F., & Verdú, G. *CT image reconstruction based on GPUs.* Procedia Computer Science. Vol. 18: 1412-1420, 2013.

[62] Wang G.,Yu H. & De Man B. *An outlook on X-ray CT research and development.* Medical Physics. Vol. 35(3): 1051-1064, 2008.

[63] Donoho, D. *Compressed sensing.* IEEE Transactions on Information Theory. Vol. 52: 1289-1306, 2006.

[64] Candès, E., Romberg, J., & Tao, T. *Stable signal recovery from incomplete and inaccurate measurement*s. Communication on Pure and Applied Mathematics. Vol. 59: 1207-1223, 2006.

[65] Yu, H., & Wang, G. *A soft-threshold filtering approach for reconstruction from a limited number of projections.* Physics in Medicine and Biology. vol. 55: 3905-3916, 2010.

[66] Yu, W., & Zeng, L. *A novel weighted total difference based image reconstruction algorithm for few-view computed tomography.* PLos ONE. Vol. 9(10): 1-10, 2014.

[67] Daubechies, I., Defrise, M., & De Mol, C. *An iterative thresholding algorithm for linear inverse problems with a sparsity constraint.* Commun. Pure Appl. Math. p. 1413-57, 2014.

[68] Daubechies, I., Fornasier, M., & Loris, I. *Accelerated projected gradient method for linear inverse problems with sparsity constraints.* J. Fourier Anal. Appl. p. 764-792, 2008.

[69] Hengyong, Y., & Ge, W. (2010). *A soft-threshold filtering approach for reconstruction from a limited number of projections.* Physics in Medicine and Biology. Vol. 55: 3905-3916, 2010.

[70] Yu, W., & Zeng, L. *A novel weighted total diference based image reconstruction algorithm for few-view computed tomography.* PLos ONE. Vol. 9 10): 1-10, 2014.

[71] Beck, A.,Teboulle, M. *A fast iterative Shrinkage-Thresholding Algorithm for linear inverse problems.* SIAM Journal on Imaging Sciences. Vol. 2:183-202, 2009.

[72] Yu, Z., Noo, F., Dennerlein, F., Wunderlich, A., Lauritsch, G., and Hornegger, J. *Simulation tool for two-dimentional experiments in x-ray computed tomography using the FORBILD head phantom.* Physics in medicine and biology. Vol. 57(13): 237-252, 2012.

[73] Wang, Z., Bovik, A., Sheikh, H., & Simoncelli, E.*Image quality assessment: From error visibility to structural similarity.* IEEE Transactions on Image Processing. Vol. 13(4): 600-612, 2004.