UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

Trabajo Fin de Master en Computación Paralela y Distribuida

Algoritmos Paralelos de Reconstrucción de Imágenes TAC

Presentado por Liubov Alexandrovna Flores
Dirigido por: Dr. Vicente Vidal Gimeno
Dr. Gumersindo Verdú Martín

Junio 2013

Agradecimientos

Quisiera agradecer al Dr. Gumersindo Verdú Martín y a mi tutor Dr. Vicent Vidal Gimeno por la oportunidad brindada y por la selección de un tema de investigación tan interesante.

A Patricia Mayo Nogueira y Francisco Rodenas Escriba agradezco infinitamente por su continuo apoyo y motivación durante el desarrollo de este trabajo.

Abstract

In medicine, the diagnosis based on computer tomography (CT) imaging is fundamental for 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 analyze and make comparison between two methods of reconstruction: analytical and algebraic methods of reconstruction of images.

Today, in practice, the reconstruction process is based on analytical methods and one of the most widely used algorithm 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 multinuclear processor systems, that provide the possibility to reduce the reconstruction time.

Among the analytical algorithms, the methods based on Fourier transform are well known. In this work we analyze the algorithm of reconstruccion of images from parallel projections based on the inverse Radon transform and its relation to the Fourier transform, with the aim to achive 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.

Algebraic 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 reducction of the execution time is achieved by using the PETSc library in parallel reconstruction of images.

We, also, compare the quality of the images reconstructed by analytical and algebraic methods from simulated and real projections. This comparison allows to make some conclusions on analytical and algebraic methods used in the reconstruction of the images.

Resumen

En medicina, el diagnóstico basado en imágenes de tomografía axial computerizada (TAC) es fundamental para la determinación de abnormalidades a través de diferentes valores de atenuación de la energía de rayos -X, lo que es, frecuentamente, dificil de ser distinguido por radiólogos. Se han desarrollado diferentes técnicas de reconstrucción de imagen. En este trabajo nosotros analizamos y comparamos dos métodos de reconstrucción: método analítico y método algebraico.

Hoy, en la práctica, el proceso de reconstrucción de imagen se basa en algoritmos analíticos entre los cuales, el algoritmo de retroproyección filtrada "filtered back-projection - 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 de TAC.

Desde el comienzo del desarrollo de escánneres 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.

Los métodos basados en la utilización de la transformada de Fourier, son los más estudiados. En este trabajo se analiza el algoritmo de reconstrucción de imagen mediante proyecciones paralelas, 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 de un sistema. Este algoritmo se basa en proyecciones paralelas y se destaca por su simplicidad y robustez, y permite extender los resultados a variedad de situaciones.

Entre los algoritmos analíticos, el algoritmo de retroproyección filtrada (FBP) se considera un estandar en la reconstrucción de imágenes TAC. El algoritmo es de bajo coste computacional, reconstruye imágenes de buena calidad, pero requiere una completidud de datos. En muchas aplicaciones el conjunto de proyecciones necesarias para la reconstrucción puede ser incompleto por razones físicas. Entonces, la unica posibilidad 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.

Actualmente, debido al aumento del poder computacional, los métodos algebraicos han y son un foco de investigación en la reconstrucción de imágenes TAC. Los métodos algebraicos son más adecuados para la reconstrucción de imágenes cuando se dispone de un menor número de proyecciones y 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 reducimos el tiempo de reconstrucción mediante el uso de la librería PETSc en la reconstrucción paralela de imagen.

Se analizan ambos métodos mediante la reconstrucción de imágenes por proyecciones simuladas y reales, comparando la calidad de imagen reconstruida con el objetivo de

obtener conclusiones respecto a los métodos usados.

Resum

En medicina, el diagnòstic basat en imatges de tomografía axial computeritzada (TAC), és fonamental per a la determinació d'anorrmalitats mitjançant diferents valors d'atenuació de l'energia dels rajos X que freqüentment són dificils de diferenciar pels radiòlegs. S'han desenvolupat diferents tècniques de reconstrucció d'imatges. En aquest treball analitzem i comparem dues mètodes de reconstrucció: anal.lítics i algebraics.

En l'actualitat el procés de reconstrucció d'imatges està basat en algoritmes anal.lítics entre els quals destaca l'algoritme de retroprojecció filtrada "filtered backprojection - FBP" que és el més estés en aquest àmbit. Aquest algoritme s'utilitza per a implementar la transformada de Radon inversa que és matemàticament un recurs molt utilitzat a l'enginyeria biomèdica en reconstrucció d'imatges de TAC.

Des del principi del desenvolupament dels escànners ha sigut molt important reduir el temps d'escanneig, millorar la qualitat d'imatge i reduir el temps de reconstrucció. La tecnología d'avui ofereix sistemes molt robustos amb diversos processadors i nuclis que fan possible reduir el temps de reconstrucció de la imatge.

Entre els algoritmes anal.lítics, els mètodes basats en l'ús de la transformada de Fourier són els més estudiats. En aquest treball s'analitza l'algoritme de reconstrucció d'imatge mitjancant projeccions paral.leles, 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 algoritme està basat en projeccions paral.leles i destaca la seua simplicitat i robustesa, perquè permet fer extensiu l'ús en varietat de situacions.

Els mètodes algebràics són més adequats per a la reconstrucció d'imatges pel seu menor nombre de projeccions i el seu ús en condicions en què la imatge té més soroll. El seu ús potser important per a el funcionament en escànners portàtils en condicions d'urgència en qualsevol lloc. No obstant això, en la pràctica aquests mètodes són menys utilitzats pel seu cost computacional. Però en aquest treball reduïm el temps de reconstrucció mitjancant l'ús de la llibreria PETSc en la reconstrucció paral.lela de la imatge.

Els dos mètodes es comparen mitjancant la reconstrucció d'imatges per projeccions simulades i reals, comparant la qualitat de la imatge final reconstruïda per tal d'obtindre'n conclusions de ambdues mètodes.

Índice

Capítulo 1

1. Introducción

1.1. Motivación

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, electrónes, protónes, ondas de sonido y señales de resonancia magnética fueron usados para estudiar objetos cuyo tamaño varía desde moléculas complejas estudiadas con los microscopios electrónicos hasta distantes fuentes de radioseñales estudiadas por radioastrónomos. Numerosas aplicaciones donde el problema de reconstrucción juega un papel principal están descritos por Stanley R. Deans en [1].

La tomografía axial computerizada (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].

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, 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 como optimizar su ejecución para conseguir tiempos menores.

Nos motiva la investigación de los algoritmos de reconstrucción de imagen existentes, el diseño de nuevos algoritmos y su implementación paralela que distribuye procesos de cálculo de forma eficiente.

1.2. Objetivos

En este trabajo se tiene como objetivo diseñar, implementar y evaluar algoritmos paralelos para resolver en forma eficiente el problema de reconstrucción de imágenes en Tomografía Axial Computerizada (TAC) sobre arquitecturas actuales como son procesadores multicores y clusters. 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 Trasformada Rápida de Fourier.
- Estudio de métodos iterativos de reconstrucción de imagen.
- Método de Siddon de generación de la matriz de un sistema de ecuaciones.
- Implementación paralela de la resolución de un sistema de ecuaciones lineales por el método LSQR utilizando la librería PETSC.
- Análisis de prestaciones y escalabilidad de los algoritmos.
- Comparación de la calidad de imágenes reconstruidas por ambos métodos, analìtico y algebráico .

1.3. Estado del arte

El problema de reconstrucción por proyecciones fue resuelto por Johan Radon en 1917. En su trabajo [1] Radon desarrolla un método analítico, basado en la transformada inversa de Radon, de reconstrucción de 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.
Existen varios métodos de reconstrucción de imagen. Estos métodos se pueden clasificar en: - algoritmos analíticos, - métodos algebraicos, - métodos estadísticos de reconstrucción. Una de las áreas más amplias de aplicación de la transformada Inversa de Radón es la Tomografía Axial Computerizada (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 y 2 ilustran la idea de como se generan las proyecciones paralelas y proyecciones fanbeam. En la Figura 1 los rayos -X se emitien de varias fuentes y se registran por los detectores después de atravesar un objeto generando proyecciones paralelas. En la Figura 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.

Actualmente, en la Tomografía Axial Computerizada, la reconstrucción 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]. En este trabajo se analiza el

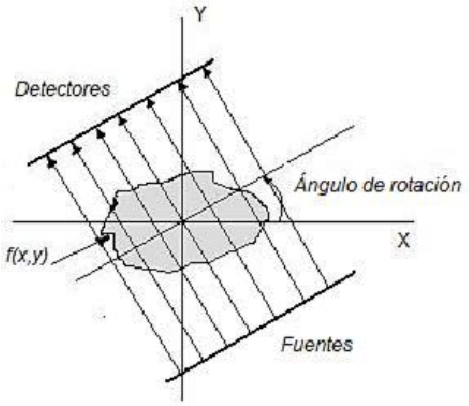

Figura 1: Proyecciones paralelas

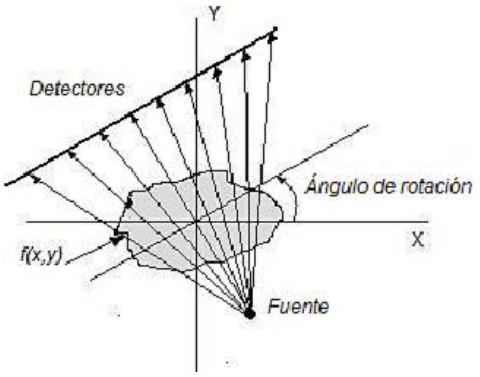

Figura 2: Proyecciones fanbeam

algoritmo que implementa la transformada de Radon inversa 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.

El algoritmo es de bajo coste computacional, reconstruye imágenes de buena calidad, pero requiere una completitud de datos. En muchas aplicaciones el conjunto de proyecciones necesarias para la reconstrucción puede ser incompleto por razones físicas. Entonces, la unica posibilidad 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.

Actualmente, debido al crecimiento del poder computacional, los métodos algebraicos son un foco de investigación en la reconstrucción de imágenes. Los algoritmos basados en métodos algebraicos son capaces de proporcionar imágenes de mayor contraste y precisión en condiciones adversas y con un menor número de proyecciones [5], [6], [7]. En los exámenes de TAC es comun escontrarse con proyecciones incompletas, en estos casos los métodos algebráicos son más adecuados para la recostrucción de imagenes [8], [9], [10].

Sin embargo, en la práctica debido a su alto coste computacional, estos algoritmos no son usados de forma amplia y siguen siendo objeto de estudio. Su uso puede ser importante en escáneres portátiles en condiciones de urgencia [11] y en estudios 3D.

En este trabajo nosotros analizamos y proponemos el uso de la librería PETSc para la resolución eficiente del sistema de ecuaciones en la reconstrucción paralela de imágenes. Después de la implementación de los códigos se realiza la comparativa de la calidad de las imágenes reconstruidas

Aunque se usa en medicina nuclear (gamma - camera, single photon emission computed tomography (SPECT), positron emission tomography (PET)), la reconstruccion iterativa no se difundió bastante en TAC. La principal razón de ello es que el conjunto de datos en CT es mucho mayor que en medicina nuclear y la reconstrucción iterativa se hace muy costosa. La acceleración de la reconstrucción iterativa es un área de investigación activo.

Stone *et al.* describe el algoritmo accelerado en unidades gráficas de procesamiento (GPUs) para imágenes de resonancia magnética (MRI) [25]. Ellos reconstruyen imágenes de 128^3 voxels en cerca de un minuto. Johnson y Sofer en [26] 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 [27]. 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 [28] accelera la reconstrucción de imágenes de 350x350x9 hasta 67 segundos en una tarjeta GPU y hasta 32 segundos en cuatro GPUs.

Parece que el tamaño de imagenes reconstruidas sigue siendo un problema. En este trabajo nosotros pusimos como un objetivo utilizar todo el poder computacional massivo de las GPUs para la reconstrucción de imágenes de mayor resolución y sin perder la calidad. Nosotros presentamos la descripción y validación del algoritmo basado en GPUs.

1.4. Estructura del documento

El resto del estudio desarrollado en este trabajo de investigación está organizado de la siguiente forma:

En el **Capítulo 2** se describen los concepos principales de los sistemas de altas prestaciones actuales sobre los cuáles están orientados los métodos desarrollados en el trabajo que se presenta. Se describen las herramientas de hardware utilizadas en los experimentos, la librería PETSc y los entornos utilizados en la programación.

El **Capítulo 3** 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. Se decribe los aspectos matemáticos del método de reconstrución basado en la transformada inversa de Radon empeado en el proceso de la reconstrucción. También se analiza la implementación paralela de este método.

El **Capítulo 4** se dedica a los métodos algebraicos que representan el objetivo central de los estudios de este trabajo. Se describen las ventajas que presenta este método y también la razón principal por la cual los métodos algebraicos no son de uso comercial actualmente. Se propone: (1) el uso de la librería PETSc para la reconstrucción paralela con el objetivo de reducir el coste computacional; (2) Se presenta el algoritmo iterativo de reconstrucción en la unidad de procesamiento gráfico (GPU).

En el **Capítulo 5** se comparan las imagenes reconstruidas por el método analítico y algebraico. En la comparación se emplean las funciones Error Cuadrático Medio (MSE) y Peak Signal-to-Noise Ratio (PSNR).

En el **Capítulo 6** se presentan algunas conclusiones del trabajo realizado y se plantean ideas para futuro desarrollo.

Capítulo 2

2. Arquitecturas de sistemas de computación de altas prestaciones (CAP)

La estructura de hardware o arquitectura determina que posibilidades e imposibilidades hay en mejorar el rendimiento de un sistema comparado con el rendimiento de 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 [14], se basa en la forma de manipular instrucciones y datos y se divide en cuatro clases de arquitecturas [15]:

- **Máquinas SISD.** Sistemas convencionales con un CPU y, consecutivamente ejecutan instrucciones en forma secuencial. Hoy muchos servidores tienen más de una CPU, pero cada una ejecuta 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 han construido este tipo de máquinas, aunque en forma teórica, en estas máquinas ejecutan diferentes instrucciones sobre único conjunto de datos.
- **Máquinas MIMD.** Estas máquinas 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 3 y 4.

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

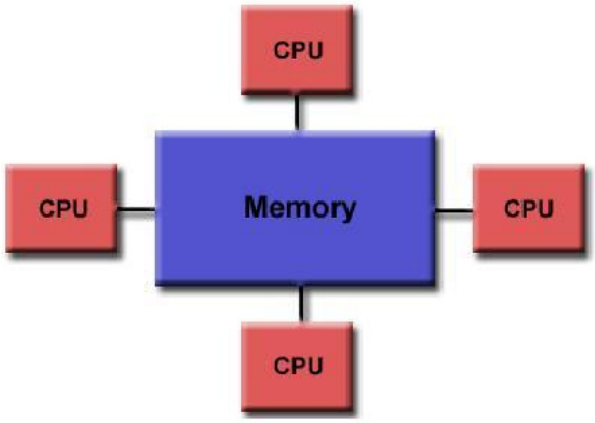

Figura 3: Memoria compartida UMA

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

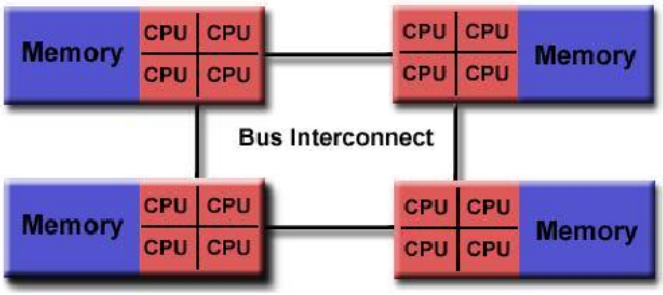

Figura 4: Memoria compartida (NUMA)

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 lento.

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

- Rapido 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. Los CPUs se conectan a través de una red y pueden intercambiar datos entre sus memorias. Estos sistemas también puede ser SIMD o MIMD. MIMD de memoria distribuida usan redes de interconexión de topologías muy variadas. Un sistema con memoria distribuida se presenta en la Figura 5.

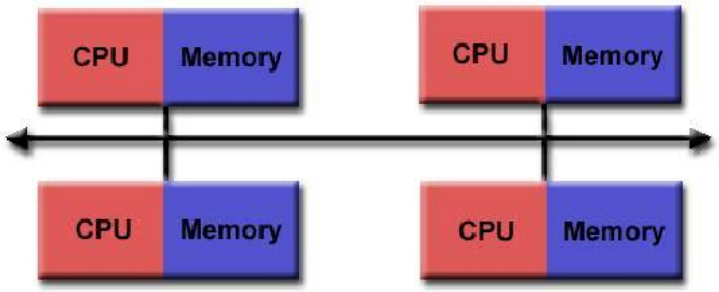

Figura 5: Memoria distribuida

Se destacan por las siguentes características:

- Necesitan una red de comunicación para 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 connectadas 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 multinúcleos

Los procesadores multi-núcleos (**multi-core processor**) son procesadores que contienen dentro de su empaque varios núcleos o unidades de procesamiento de instrucciones. Un procesador multi-núcleo puede repartir los procesos entre sus varios nucleos para su posterior ejecución. Los cores están integrados en un chip llamado multiprocesador.

Un dual-core procesador tiene dos cores (e.g. AMD Phenom II X2, Intel Core Duo), a quad-core procesador tiene cuatro nucleos (e.g. AMD Phenom II X4, la línea core Intel 2010, i3, i5, and i7), y hexa-core procesador que tiene seis cores (e.g. AMD Phenom II X6, Intel Core i7 Extreme Edition 980X).

Un **many-core procesador** es un procesador que tiene un número de cores 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 cores que establece el paso de un multi-core a un many-core procesador se mide en docenas de cores. 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 cores. Las topologías comunes para interconectar cores incluyen bus, anillo, grid dos dimencionales, 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/video, 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 multinúcleo 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) [16], [17] 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. Dos 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 cores en un chip crece rápido, y como resultado, se puede dar una nueva interpretación a la ley de Moore: es el número de

cores se duplica cada 18 meses.

El esquema principal de procesador AMD Athlon se presenta en la Figura 6.

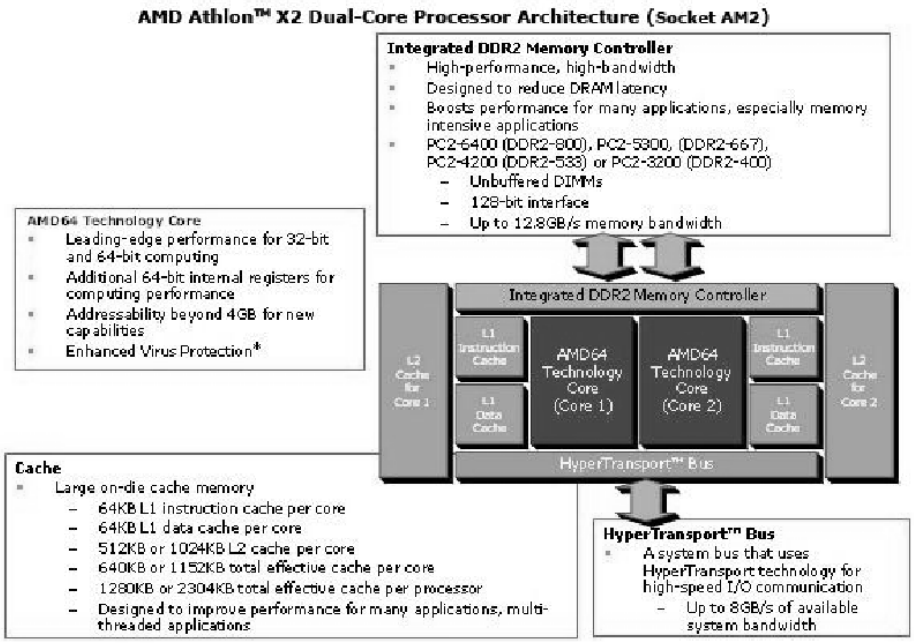

Figura 6: Arquitectura de procesador AMD Athlon

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 7.

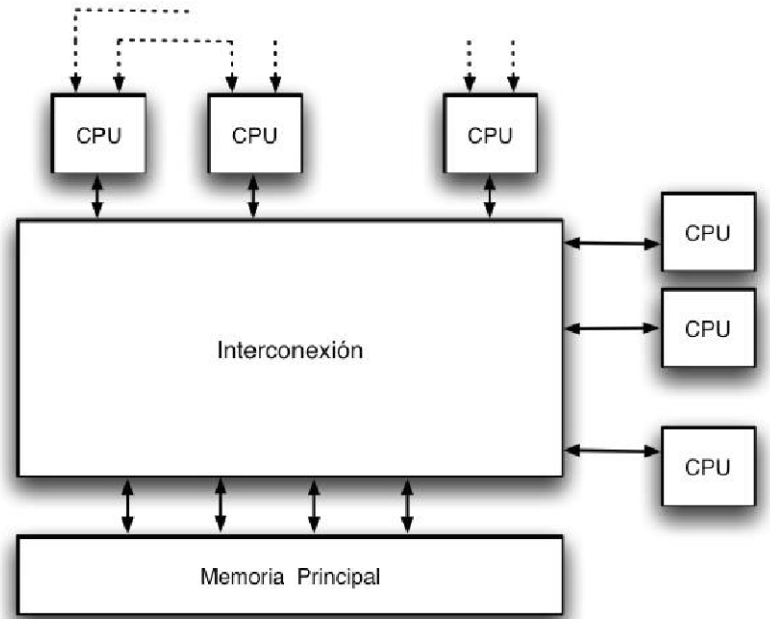

Figura 7: Diagrama de bloques de un sistema multiprocesador

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 una computadora 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 paralelas 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 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 NIVIDIA en 1999, es el procesador paralelo más potente hoy día [22]. Para satisfacer la demanda en los gráficos 3D de alta resolución en tiempo real, la GPU se ha desarrollado como un procesador multicore con multithreading y de alto paralelismo. Es un procesador de tremenda potencia para

los cálculos con simple y doble presición. La GPU es un prosesador de muy alto ancho de banda de memoria como se ilustra en las Figuras 8 y 9.

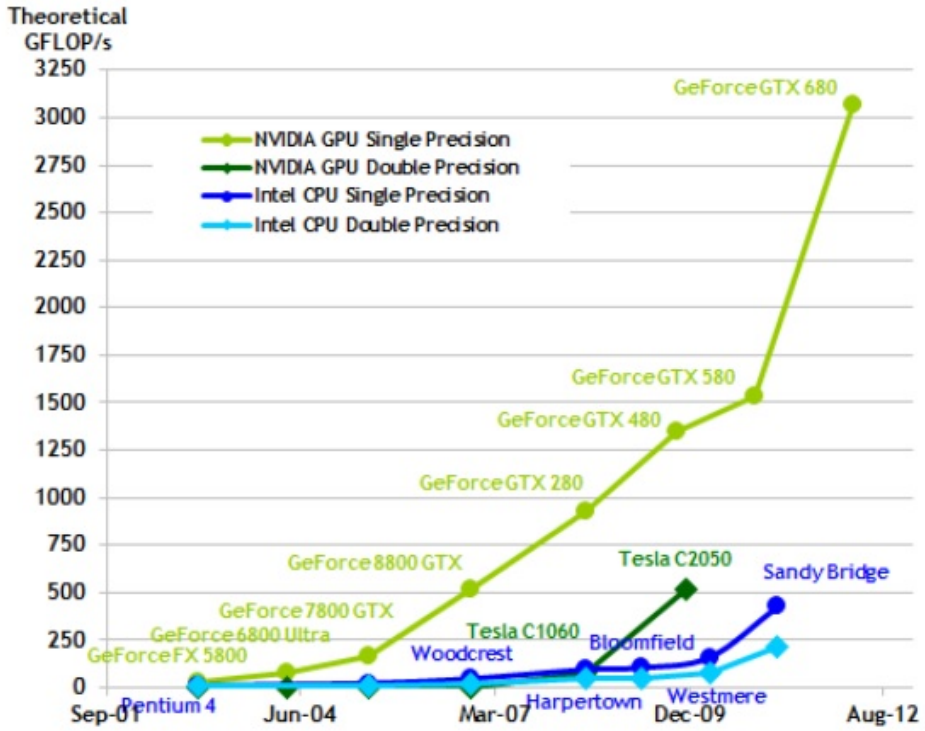

Figura 8: Operaciones por segundo para CPU y GPU. Documentación de NVIDIA

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 10.

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.

NVIDIA introdujo dos tecnologías principales - la arquitectura G80 [24] (primero fue introducida en GPUs GeForce 8800®, Quadro FX 5600® y Tesla C870®) y CUDA [22], 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 extención CUDA dirigidos a explotar el procesador en forma paralela y masiva. Esta forma

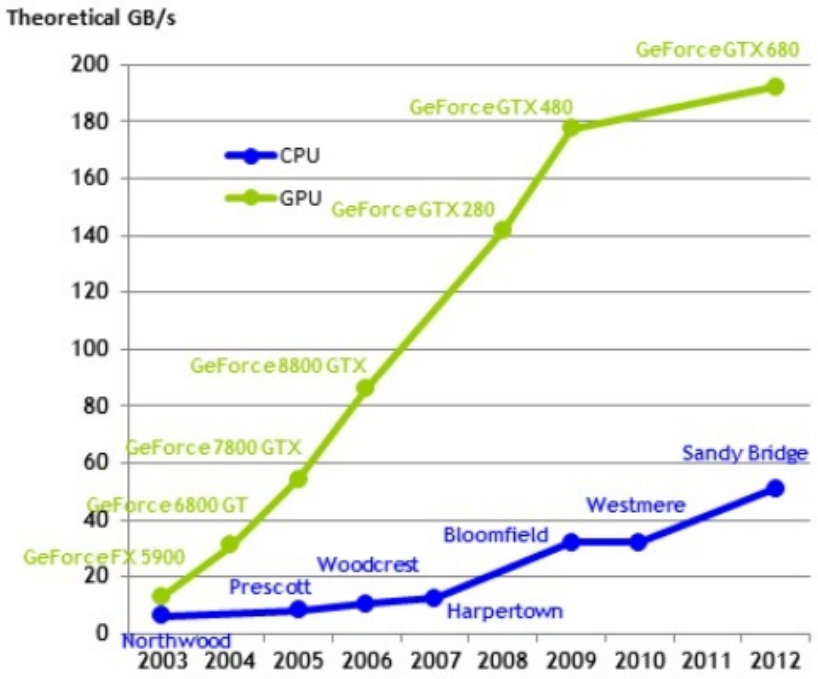

Figura 9: Ancho de banda de memoria para CPU y GPU. Documentación de NVIDIA

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.

La siguiente generación de NVIDIA. GeForce 8800 era el primer producto que puso el inicio del modelo de Computación GPU. Introducido en 2006, permitió a los programadores usar toda la potencia de los GPUs sin necesidad de aprender un nuevo idioma de programación.

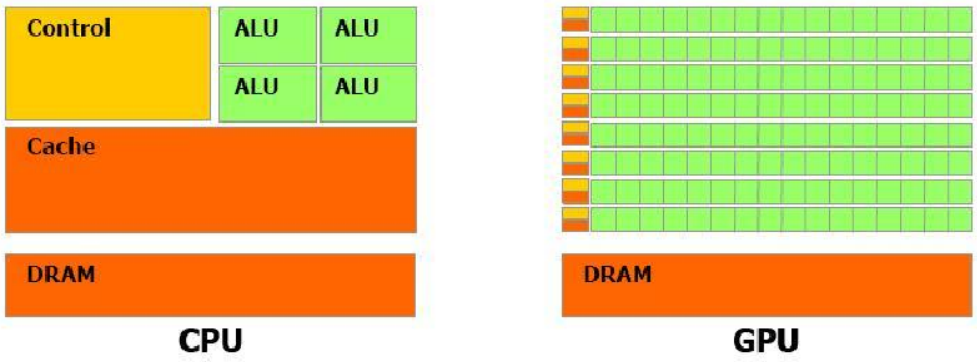

Figura 10: La GPU incluye más transistores para procesamiento de datos

La arquitectura **Fermi** [23] es una arquitectura más significante desde entonces. Las caracteristicas principales mejoradas en Fermi:

- rendimiento mejorado con doble presición - muchas aplicaciones requeren rendimiento alto con doble presición
- 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.

Un sistema es Gpu.dsic.upv.es. El sistema se caracteriza por el procesador de 2.6GHz y dos unidades de prosecamiento gráfico. Las tarjetas GPU tienen las siguientes propiedades:

- son tarjetas TESLA K20c no integradas
- la capacidad de computo: 3.5
- Memoria Global: 5GB
- Memoria Constante: 64 kB
- Número de Multiporocesadores: 13 con 192 Cuda cores / MP
- Número total de cores: 2496 Cuda cores
- Memoria Compartida por multiprocesador: 49 kB

Otro sistema utilizado es KAHAN. El KAHAN es un cluster de computación de pequeño tamaño. 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 formula:
Peak = ncores x nFPU x f = 192 x 4 x 3 = 2304 GFlops.

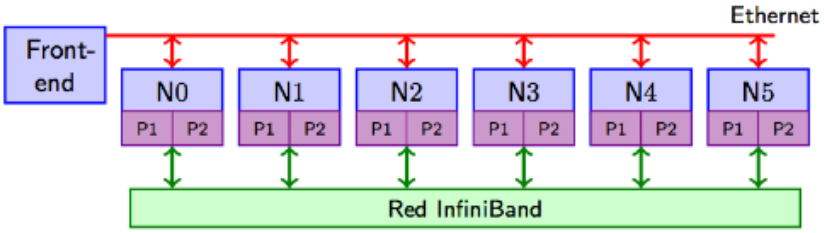

Figura 11: El Cluster Kahan.dsic.upv.es

2.9. Herramientas software

En este apartado se describen modelos de programación paralela y herramientas software usadas 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 asincronamente. 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 Threads.** 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 (threads) que se ejecutan simultaneamente. Cada thread 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, solo el programa (thread) principal se queda presente hasta que la aplicación se completa.

  Generalmente, este modelo de programación esta asosiado con arquitecturas de memoria compartida. Desde el punto de vista de programación, la implementación consta de un conjunto de directivas insertadas en el código serial o paralelo y el

programador es responsable de determinar todo el paralelismo. Un estandar de implementación de threads es OpenMP.

OpenMP. La librería OpenMP [19] permite en forma simple crear threads y controlar concurrencias de estos basandose en directivas del compilador. Es una librería portable. Existe implementaciones en C/C++ y Fortran.

- **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.
  - 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 estandar de la implementación de modelo de programación de Paso de Mensajes en la actualidad [20].

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 [18]. 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 nolineales
- 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)** estandar para las comunicaciones. Las librerías posibilitan la personalización y extención de algoritmos y sus implementaciones. PETSc posibilita el uso de paquetes externos como Matlab y otros, es completamente usable 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 han introducido una nueva subclase de vectores que ejecuta las operaciones sobre los vectores. 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 escala grande.

2.9.3. Librerías CUBLAS y CUSPARSE

La utilización de GPU, con la enorme potencial de computo paralelo, puede elevar considerablemente la eficiencia del algoritmo. El modelo de programación CUDA [30] permite resolver muchos problemas complejos computacionalmente de un modo más eficiente que en CPU. Las librerías CUBLAS [31] y CUSPARSE [32] proporcionan al usuario el acceso a los recursos computacionales de unidades de procesamiento gráfico de NVIDIA (GPUs). La librería CUBLAS es la implementación BLAS(Basic Linear Algebra Subprograms) para GPUs. Para usar la librería, la application tiene que colocar las matrices y vectores necesarios en en la memoría 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 basicas de algebra lineal usadas para las operaciones con matrices dispersas y está diseñado para ser llamada 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 [29]. 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.

Capítulo 3

3. Métodos analíticos

Actualmente, en los escáneres comerciales el proceso de reconstrucción de imagen se basa en algoritmos analíticos entre los cuales, el algoritmo de retroproyección filtrada "filtered backprojection" 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 (Tomografía Axial Computarizada) [1].

En la Tomografía Axial Computarizada, 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, representa la misma imagen [1]. La transformada de Radon es un conjunto de proyecciones tomadas bajo diferentes ángulos que contienen información sobre el objeto (imagen) escaneado y la inversa de la transformada, representa la imagen de dicho objeto.

3.1. Aspectos matemáticos de reconstrucción de imagen

Dada la utilidad de la transformada de Radon y su inversa en la reconstrucción de imagen TAC en este apartado vamos a resumir el algoritmo en el que se basa.

3.1.1. Definición de la transformada de Radon y su inversa

Si los rayos -X pasan a través de un objeto, se observa que la intensidad de los rayos disminuye de acuerdo con la ecuación (1)

$$I = I_0 \exp(-\mu x). \tag{1}$$

En (1) I_0 es la intensidad de rayos inicial, I es la intensidad al pasar la distancia x a través del objeto. El coeficiente de atenuación μ depende de la densidad del material ρ y del número atómico Z

$$\mu = \mu(\rho, Z). \tag{2}$$

Si los rayos -X pasan a través de varios medios, el decrecimiento relacional está dado por la formula (3)

$$\frac{I}{I_0} = \exp\left[-\sum_j \mu_j x_j\right]. \tag{3}$$

Si $\mu = \mu(x)$ es una función contínua de x, el sumatorio en (3) se puede expresar como una integral de línea a lo largo del paso del rayo

$$\frac{I}{I_0} = \exp\left[-\int_L \mu(x)dx\right]. \tag{4}$$

Ahora, consideraremos una sección transversal de un objeto tridimensional. Sea la sección transversal perpendicular al eje z, entonces, el coeficiente de atenuación μ en el plano es una función de dos variables:

$$\mu = \mu(x, y). \tag{5}$$

Y el decrecimiento relacional del rayo -X a través del plano xy a lo largo de la línea L está dado por la formula (6)

$$\frac{I}{I_0} = \exp\left[-\int_L \mu(x, y)ds\right]. \tag{6}$$

Al mover la fuente del rayo -X y el detector, es posible obtener un conjunto de proyecciones.

La proyección $\Re(\phi, s)$ de un objeto inicial se define como el conjunto de integrales de línea de la intensidad de imagen, representada por $f(x, y)$, a lo largo de las líneas l a una distancia s del origen del sistema de coordenadas y formando un ángulo ϕ con el eje x:

$$\Re(\phi, s) = \int_l f(x, y)dl. \tag{7}$$

Una proyección $\Re(\phi, s)$ para un ángulo ϕ determinado, puede verse en la Figura 12.

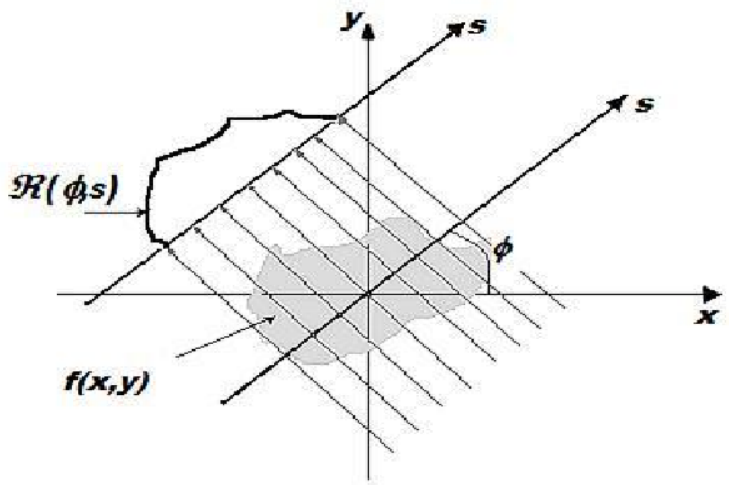

Figura 12: Proyection $\Re(\phi, s)$ para un ángulo ϕ

Un conjunto de proyecciones tomadas para diferentes ángulos, representa la imagen y se conoce como la transformada de Radon. La transformada de Radon en R^2 puede ser expresada por la formula (8)

$$\Re(\phi, s) = \widetilde{f} = \int_{-\infty}^{+\infty} f(x(t), y(t))dt. \tag{8}$$

Donde las coordenadas (Θ, s) están expresadas en forma paramétrica mediante la ecuación (9)

$$(x(t), y(t)) = t(\sin\Theta, -\cos\Theta) + s(\cos\Theta, \sin\Theta). \tag{9}$$

La función $f(x, y)$, como se ha comentado, representa la imagen original, y puede ser calculada mediante la inversa de la transformada de Radon. En forma simbólica:

$$f(x, y) = \Re^{-1}(\widetilde{f}). \tag{10}$$

Uno de los métodos de calcular la transformada de Radon inversa se conoce como el método directo de Fourier.

3.1.2. Relación con la transformada de Fourier

La trasformada de Radon está relacionada con la transformada de Fourier por medio del Teorema de Slices [3], que obtiene la trasformada de Fourier 2D de la imagen por medio de las proyecciones. De esta forma, se conoce la transformada de Fourier 2D de la imagen (al menos en algunos puntos) y tomando su inversa se puede obtener la imagen por la formula (11).

$$f(x, y) = \frac{1}{4\pi^2} \int\int G(\phi, \omega) e^{j\omega(x\sin\phi - y\cos\phi)} |\omega| d\omega d\phi. \tag{11}$$

En (11) $G(\phi, \omega)$ es la transformada de Fourier 1D de las proyecciones, $|\omega|$ es el determinante del Jacobiano al pasar de sistema de coordenadas rectangulares a polares. El producto $G(\Theta, \omega)|\omega|$ se llama retroproyección filtrada y se obtiene multiplicando las proyecciones por $|\omega|$ en el espacio de Fourier.

La formula (11) es la base del cálculo de la transformada inversa de Radon en la reconstrucción de TAC y representa la relación entre las proyecciones y la imagen inicial.

El algoritmo de retroproyección filtrada es uno de los algoritmos más aceptados en la reconstrucción de imagen original en la Tomografía Axial Computarizada.

El proceso de reconstrucción por proyecciones con el algoritmo FBP se puede visualizar en la Figura 13.

3.2. Paralelización

Las proyecciones tomadas por el escáner se dan en forma de una matriz PR_{mxn}. Las columnas PR_{cols} de esta matriz corresponden al número de ángulos para cuales se toman las proyecciones.
Las filas PR_{fils} de la matriz corresponden al número de detectores y los valores de la matriz de proyección representan las intensidades de los rayos -X registradas por los detectores al atravesar el objeto.

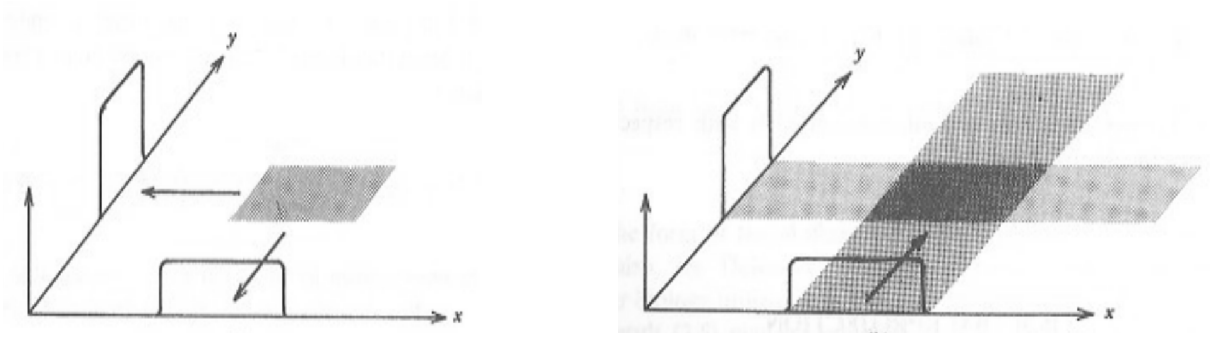

Figura 13: Reconstruccion por dos proyecciones

El algoritmo consta de los siguientes módulos:

- módulo principal: Image Reconstruction
- módulo: Backproject
- módulo: Filter Projections

El diagrama del algoritmo se presenta en la Figura 14.

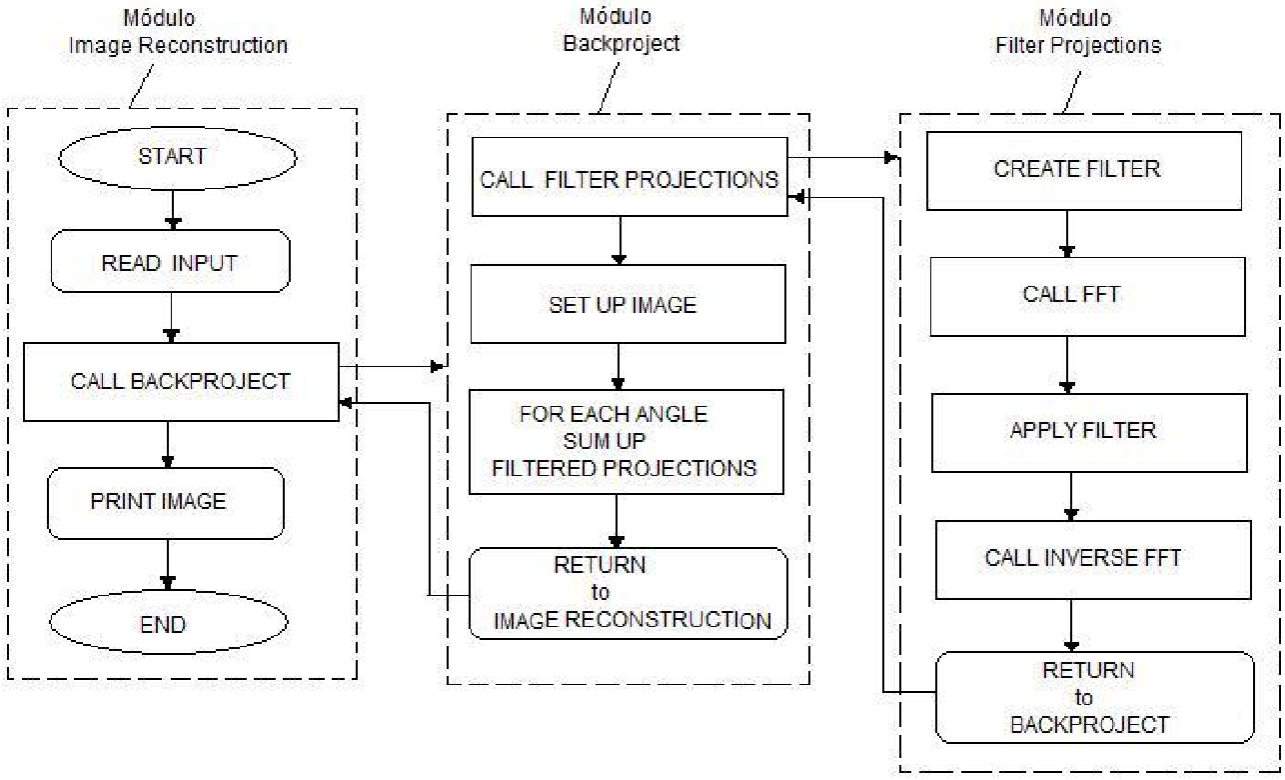

Figura 14: El diagrama del algoritmo

Descripción de los módulos.

- Módulo Image Reconstruction:
  - Leer los datos de entrada
  - Llamada al módulo Backproject
  - Generar el archivo de salida
- Módulo Backproject:
  - Se hace una llamada al Filter projections para filtrar las proyecciones PR (9)
  - Se hace una copia de las proyecciones filtradas (10)
  - Se inicializa la imagen a reconstruir BPR (11)
  - Se define la grid rectangular (13:19)
  - Se inicializa BPIa - la imagen actual que corresponde al ángulo actual (20)
  - Para cada ángulo se determinan los píxeles por los cuales pasa la proyección (22:37)
  - Los píxeles de pasos (22:37) se registran en la imagen actual BPIa (39:43)
  - Se actualiza y se normaliza la imagen final BPR (44:50)
- Módulo Filter projections:
  - Se determina el orden del filtro y se construye el Hamming filter (51:68)
  - Se inicializa la matriz FPR que representa las proyecciones filtradas (69)
  - Las proyecciones PR se registran en FPR en orden de columna mayor(70:74)
  - Se aplica la transformada rápida de Fourier a las proyecciones (75:78)
  - Las proyecciones se filtran en el espacio de Fourier (79:84)
  - Se aplica la transformada inversa de Fourier a las proyecciones filtradas (85:88)
  - Las proyecciones se normalizan y se registran en '*row mayor*' orden (89:94)

Algoritmo

- **Entrada:** Matriz PR_{mxn} - matriz de proyeciones.
- **Salida:** Matriz BPR_{mxm} cuyos valores representan las intensidades de la imagen reconstruida.
- **Módulo principal: Image Reconstruction:**

1: Leer datos de entrada
2: Backproject(PR,theta,theta size,rows,cols,BPR)

/**Escribir la imagen reconstruida en el archivo de output
3: PARA $i = 0$ HASTA $i < Resolucion - 1$
4: PARA $j = 0$ HASTA $j < Resolucion - 1$
5: *write* $pBPR[i][j]$

6: END PARA
7: END PARA
8: End Image Reconstruction

- **Módulo Backproject (PR, theta, theta size, rows, cols, BPR):**

9: Filter projections (PR, PRrows, PRcols, theta size, 1)
10: cblas-dcopy (PRrows*PRcols, PR, 1, filtPR, 1)

/** Set up la imagen
11: cblas-dscal(sideSize*sideSize, 0, BPR, 1)
/** Convertir los angulos a radianes
12: cblas-dscal(theta size, pi/180, theta, 1)
13: $midindex = (PR_{rows} + 1)/2$

/** Set up xpr & ypr matrices
pragma omp parallel for private(j)
14: PARA $i = 0$ HASTA $i < sideSize - 1$
15: PARA $j = 0$ HASTA $j < sideSize - 1$
16: $p_{xpr}[i][j] = j + 1 - (sideSize + 1)/2$
17: $p_{ypr}[i][j] = i + 1 - (sideSize + 1)/2$
18: END PARA
19: END PARA

/** Inicializar: $BPIa = 0$
20: cblas-dscal (theta size*sideSize*sideSize, 0, BPIa, 1)
/** Región paralela
21: # pragma omp parallel for private(z,i,j,i1,i2,k) firstprivate(sideSize)
22: PARA $z = 0$ HASTA $z = z_{Size} - 1$
pragma omp parallel for private (j)
23: PARA $i = 0$ HASTA $i = sideSize - 1$
24: PARA $j = 0$ HASTA $j = sideSize - 1$
25: $pfiltIndex[z][i][j] = floor(0{,}5 + midindex + p_{xpr}[i][j] * sin(theta[z]) -$
26: $-p_{ypr}[i][j] * cos(theta[z])) - 1$
27: END PARA
28: END PARA

/** Determinar los píxeles por los cuales pasa la proyección
29: $k = 0$
30: PARA $i = 0$ HASTA $i = sideSize - 1$
31: PARA $j = 0$ HASTA $j = sideSize - 1$
32: Si $pfiltIndex[z][i][j] >= 0$ && $pfiltIndex[z][i][j] < sideSize$

33: $pspota[z][k] = i * sideSize + j$
34: $pnewfiltIndex[z][k] = pfiltIndex[z][i][j]$
35: $k = k + 1$
36: END PARA
37: END PARA

/** Region paralela
38: # pragma omp parallel for private(i,i1,i2) schedule(dynamic,100)
39: PARA $i = 0$ HASTA $i = k - 1$
40: $i1 = pspota[z][i]$
41: $i2 = pnewfiltIndex[z][i]$
42: $pBPIa[z][i1] = pfiltPR[i2][z]$
43: END PARA

/** Región paralela
44: # pragma omp parallel for schedule(dynamic,100)
45: PARA $i = 0$ HASTA $i = sideSize * sideSize - 1$
46: $BPR[i] = BPR[i] + pBPIa[z][i]$
47: END PARA
48: END PARA /* End of the angle loop */

/** Rescale the image
49: cblas-dscal(sideSize*sideSize, 1.0/(theta size), BPR, 1)
50: END of Backproject

- **Módulo Filter projections(PR, PRrows, PRcols, theta size, 1):**

51: $a = floor(\log(2 * PR_{rows})/\log(2))$
52: SI $(pow(2, a) < (2 * PR_{rows}))$
53: $a = a + 1$
54: END SI
55: order = máx$(64, pow(2, a))$ /* order del filtro

/** Ramp filter
56: PARA $i = 0$ HASTA $i = order/2 - 1$
57: $H1[i] = 2 * i/order$
58: $w[i] = 2 * pi * i/order$
/** Hamming filter
61: $H1[i] = H1[i] * (0{,}54 + 0{,}46 * \cos(w[i]/d))$
62: END PARA
63: cblas-dcopy (order/2+1, H1, 1, H2, 1)
64: k = order/2
65: PARA $i = order/2 - 1$ HASTA $i > 0$

66: $H2[k+1] = H[i]$
67: $k = k + 1$
68: END PARA

/** Región paralela
/** Filtrar las proyecciones en el espacio de Fourier
69: cblas-dscal(order*PRcols, 0, FPR, 1)
70: PARA $i = 0$ HASTA $i = PR_{cols} - 1$
71: PARA $j = 0$ HASTA $j = PR_{rows} - 1$
72: $pFPR[i][j] = pPR[j][i]$
73: END PARA
74: END PARA

/** Call fftw
75: # pragma omp parallel for
76: PARA $i = 0$ HASTA $i = PR_{cols} - 1$
77: call FFT
78: END PARA
79: # pragma omp parallel for private (i,j)
80: PARA $i = 0$ HASTA $i = PR_{cols} - 1$
81: PARA $j = 0$ HASTA $j < order - 1$
82: $pout[i][j] = pout[i][j] * H2[j]$
83: END PARA
84: END PARA

/** A las proyecciones filtradas se aplica la inversa de FFT
/* Región paralela
85: # pragma omp parallel for

86: PARA $i = 0$ HASTA $i = PR_{col} - 1$
87: call inverse FFT
88: END PARA
89: PARA $i = 0$ HASTA $i = PR_{cols} - 1$
90: PARA $j = 0$ HASTA $j < PR_{rows} - 1$
91: $pPR[j][i] = pFPR[i][j]/order$
92: END PARA
93: END PARA
94: END of Filter-projections

3.3. Evaluación de coste

Vamos a suponer que el algoritmo se ejecuta en un sistema con memoria compartida.

Coste del módulo Backproject.
Sea N_z es el número de ángulos, $SideSize$ es el tamaño de la imagen a reconstruir (tamaño del problema), p - número de procesos.

- Tiempo secuencial - el número total de operaciones ejecutadas en forma secuencial por un procesador:
  $T_1 = \sum_{i=0}^{SideSize} \sum_{j=0}^{SideSize} 8 +$
  $+ \sum_{z=0}^{N_z} \left[\sum_{i=0}^{SideSize} \sum_{j=0}^{SideSize} 6 + \sum_{i=0}^{SideSize} \sum_{j=0}^{SideSize} 3 + \sum_{i=0}^{SideSize*SideSize} 1 \right]$

  $T_1 = (8 + 10 * N_z) * SideSize * SideSize$

- Tiempo paralelo - el número de operaciones ejecutadas en forma paralela por cada procesador:
  $T_p = \sum_{i=0}^{\frac{SideSize}{p}} \sum_{j=0}^{SideSize} 8 +$
  $+ \sum_{z=0}^{\frac{N_z}{p}} \left[\sum_{i=0}^{SideSize} \sum_{j=0}^{SideSize} 6 + \sum_{i=0}^{SideSize} \sum_{j=0}^{SideSize} 3 + \sum_{i=0}^{SideSize*SideSize} 1 \right]$

  $T_p = \frac{1}{p} SideSize * SideSize * (8 + 10 * N_z)$

- SpeedUp del sistema - la relación entre el tiempo de ejecución en un solo procesador y el tiempo de ejecución en procesadores múltiples:

  $S_p = \lim_{N \to \text{ínf}} \frac{T_1}{T_p} = p$

- Eficiencia del sistema - grado de utilización del sistema multiprocesador:

  $E_f = \lim_{N \to \text{ínf}} \frac{S_p}{P} = 1$

- Escalabilidad - la capacidad de un 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:

  $E_f = \frac{S_p}{p} = \frac{T_1}{pT_p} = \frac{T_1}{T_1 + pT_p - T_1} = \frac{1}{1 + \frac{pT_p - T_1}{T_1}}$
  si $T_1 = w$, $E_p = \frac{1}{1 + I(w,p)}$,
  donde $I(w,p) = \frac{pT_p - w}{w}$ se conoce como la función de isoeficiencia.

  $I(w,p) = \frac{pT_p - w}{w} = \frac{pT_p}{w} - 1 \approx \frac{O(p)}{O(w)}$

La función de isoeficiencia $I(w, p)$ se mantiene constante si el tamaño del problema w y el número de procesadores p crecen en la misma proporción. Es decir, la eficiencia del sistema E_f se mantiene constante, lo que lleva a la conclusión que el algoritmo es escalable.

Coste del módulo Filter projections.
Sea *order* - es el orden del filtro, PR_{rows} , PR_{cols} - el número de filas y columnas en la matriz de proyecciones.

- Tiempo secuencial: $T_1 = \sum_{i=0}^{order/2} 10 + \sum_{i=0}^{PR_{cols}} callFFTP + \sum_{i=0}^{PR_{cols}} 1 + \sum_{i=0}^{PR_{cols}} inverseFFTP + \sum_{j=0}^{PR_{cols}} \sum_{i=0}^{PR_{rows}} 1$

$$T_1 = 10 * order/2 + 3 * PR_{cols} + PR_{cols} * PR_{rows}$$

- Tiempo paralelo: $T_p = \sum_{i=0}^{\frac{order/2}{p}} 10 + \sum_{i=0}^{\frac{PR_{cols}}{p}} 3 + \sum_{i=0}^{\frac{PR_{cols}}{p}} \sum_{i=0}^{PR_{rows}} 1$

$$T_p = 10 * \frac{order/2}{p} + 3 * \frac{PR_{cols}}{p} + \frac{PR_{cols}}{p} PR_{rows}$$

- SpeedUp: $S_p = \frac{T_1}{T_p} = p$
- Eficiencia: $E_f = \frac{S_p}{p} = 1$

3.4. Resultados experimentales

El algoritmo representado por la formula (11) es paralelizable porque 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 fue testeado en el sistema KAHAN. Se han adquirido imágenes digitales en formato DICOM, que es el formato que se está implantando en el campo de 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 procesos N_p relacionado con los cores y procesadores del sistema, y el tamaño del problema N que está relacionado con la dimensión de la imagen.

Los resultados de los experimentos se resumen en la Tabla 1 donde se presenta el tiempo (en segundos) de reconstrucción de imágenes de diferentes tamaños N con diferente número de procesos N_p en el cluster KAHAN.

$N_p \backslash N$	367x90	367x180	729x180	1453x180	GFLOPS
1	2.2	4.3	14.6	62.5	12
2	1.0	1.9	7.6	27.6	24
4	0.5	1.1	4.1	14.9	48
8	0.3	0.7	2.5	9.1	96
16	0.2	0.4	1.4	5.7	192
32	0.1	0.2	0.9	3.3	384
64	0.1	0.2	0.8	2.9	768

Tabla 1: Tiempo de ejecución en un nodo del cluster KAHAN

El rendimiento máximo teórico (el número de operaciones en coma flotante por segundo - FLOPS) se calcula de de la siguiente forma:
$FLOPS = cores * clock * \frac{FLOPs}{cycle}$, donde $\frac{FLOPs}{cycle} = 4$.

En la Figura 15 se observa la dependencia del tiempo de ejecución en función del tamaño de imagen N y número de procesos Np.

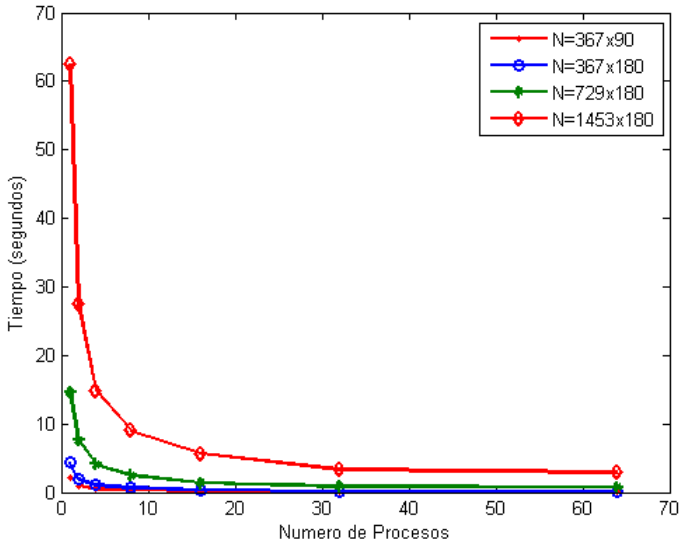

Figura 15: Variación de tiempo de ejecución en un nodo de KAHAN

En las figuras 16 y 17 se presenta la variación de SpeedUp y Eficiencia en función del número de procesos y el tamaño del problema N en KAHAN.

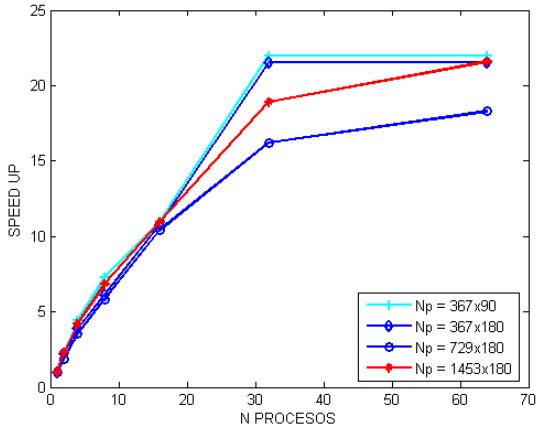

Figura 16: Variación de SpeedUp

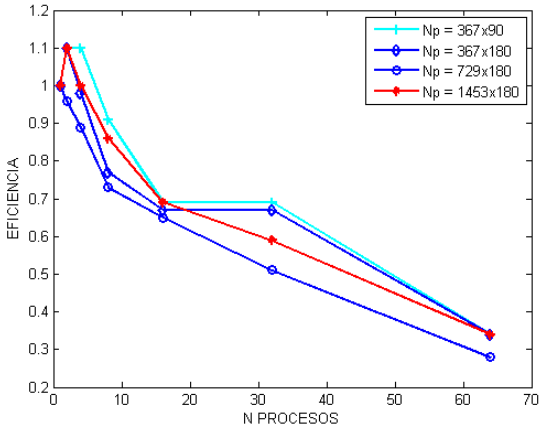

Figura 17: Variación de Eficiencia

Analizando los resultados de la Tabla 1 y Figuras 15-17 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.

A continuación, en las figuras 18, 19, 20, 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.

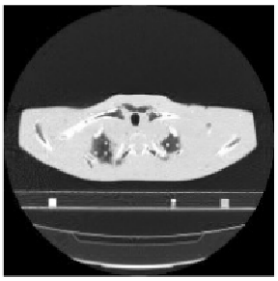

Figura 18: Imagen original

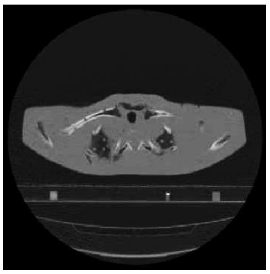

Figura 19: Reconstrucción secuencial

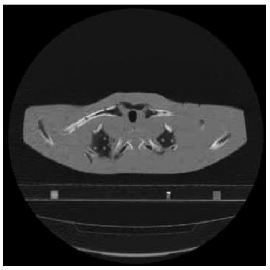

Figura 20: Reconstrucción paralela

3.5. 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 de cores en el procesador del sistema.

Capítulo 4

4. Métodos algebraicos de reconstrucción de imagen

4.1. Métodos algebraicos: ventajas y desventajas

Aunque en la primera reconstrucción tomográfica por proyecciones usaron métodos algebraicos, en la actualidad el proceso de reconstrucción en scanners clínicos está basado en algoritmos analíticos que usan la transformada inversa de Fourier. Uno de los algoritmos más usados se conoce como Retroproyección filtrada (FBP) comentado en el Capítulo 3. Este algoritmo está descrito en literatura bastante bien, e.g. [3]. Por otro lado, los algoritmos algebraicos son menos usados debido a su coste computacional muy elevado.

Sin embargo, los métodos algebraicos 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. [8]. En condiciones de ruido, los métodos algebraicos permiten la reconstrucción de imagenes de más alto contraste y precisión por menor número de proyecciones que los métodos basados en la transformada de Fourier [5], [6].

En Tomografía Axial Computarizada (TAC), es comun encontrar proyecciones incompletas, proyecciones no equiespaciadas. En estos casos los métodos algebraicos reconstruyen imagenes de mejor calidad [9], [10].

Una de las líneas de investigación donde puede ser utilizada esta metodología está relacionada con equipos tomográficos portátiles que usan nanotubos como un componente por su capacidad de producir rayos -X [11]. Este tipo de escáneres pueden ser usados para realizar examenes de urgencia en cualquer lugar. Ellos no producen datos equiespaciados, y, por esta razón, la reconstrucción algebraica es más apropiada en estos equipos.
Sin embargo, la mayor desventaja de los métodos algebraicos es su alto costo computacional. En este capítulo, nosotros analizamos y proponemos (1) - el uso de la librería PETSc para el uso eficiente de un sistema en la reconstrucción algebraica de imágenes y (2) - implementamos el algoritmo de reconstrucción de imágenes basado en GPU.

4.2. Aspectos matemáticos y expresión algebraica

Suponemos que la matriz X es la versión digital de una imagen $f(x, y)$ como se ilustra en la Figura 21.
Esto significa que la imagen $f(x, y)$ puede ser aproximada por los elementos de la matriz X 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 21

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

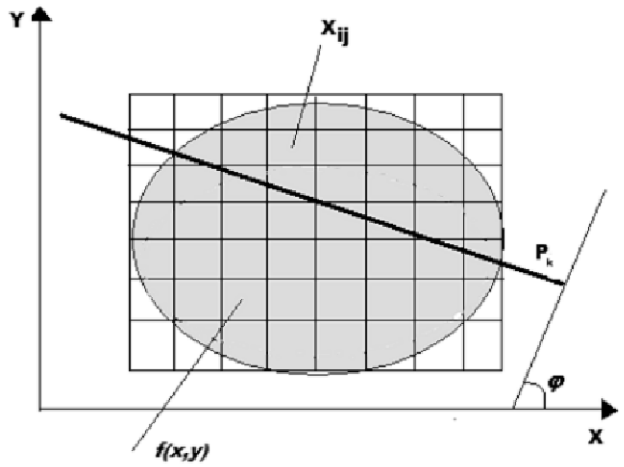

Figura 21: Proyección P_k para el ángulo ϕ

y puede ser calculada por la fórmula (12)

$$\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij}(k,\phi_r)X_{ij} = P_{k,\phi_r}. \tag{12}$$

En (12), los valores $W_{ij}(k,\phi_r)$ representan la contribución de cada pixel de la imagen a cada proyección y dependen de la proyección k y del ángulo ϕ_r ; X_{ij} son intensidades incógnitas de la imagen; P_{k,ϕ_r} son las proyecciones tomadas por el escáner.

En forma matricial la ecuación (12) está dada por la fórmula (13)

$$AX = P, \tag{13}$$

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

Los algoritmos de reconstrucción por proyecciones que se basan en métodos algebráicos se conocen como algoritmos iterativos. Básicamente, estos algoritmos de reconstrucción de imagen son esquemas para resolver sistemas de equaciones de la forma (12).
El caso más simple, es el de una matriz de cuatro píxeles (2x2), como se muestra en la Figura 22.

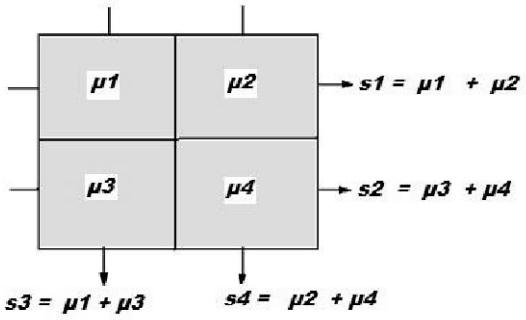

Figura 22: Sistema de 4 ecuaciones

En este caso dos medidas de dos proyecciones van a generar un sistema de cuatro ecuaciones con cuatro incógnitas que se puede resolver.

La extención a una matriz de 3x3 con nueve incógnitas puede ser resuelto con doce valores de medición (Figura 23).

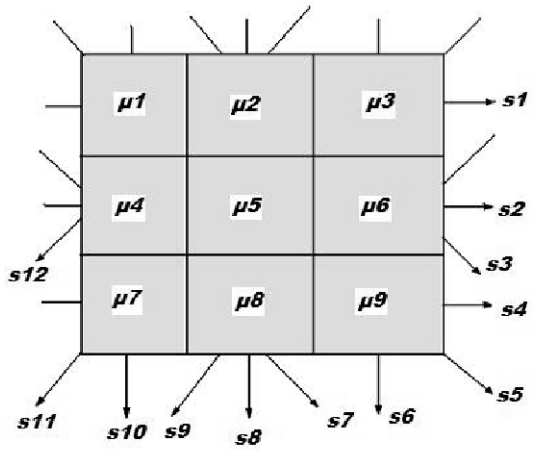

Figura 23: Sistema de 12 ecuaciones

Las dimensiones de SMR crecen proporcionalmente a 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 multicores. Esta tecnología permite paralelizar cálculos asignando cada parte independiente a un proceso lo que posibilita un manejo de recursos de sistema más eficiente.

Como se ha mencionado, en este trabajo nosotros usamos la librería PETSc para reducir el tiempo de cálculos.

Para un ángulo dado, asumimos que el número de proyecciones varia de 1 a m (m es igual al número de detectores). Si tomamos k diferentes ángulos, entonces en (13) P es el vector-columna con mk elementos, X es una matriz-columna con n^2 elementos

$$P = \begin{bmatrix} P_{11} \\ ... \\ P_{m1} \\ ... \\ P_{mk} \end{bmatrix}, \; X = \begin{bmatrix} X_{11} \\ X_{12} \\ ... \\ X_{nn} \end{bmatrix}. \tag{14}$$

Y la matriz A es una matriz rectangular con las dimensiones mkxn^2

$$A = \begin{bmatrix} W_{11}(11) & W_{12}(11) & ... & W_{nn}(11) \\ ... & ... & ... & ... \\ W_{11}(m1) & W_{12}(m1) & ... & W_{nn}(m1) \\ ... & ... & ... & ... \\ W_{11}(mk) & W_{12}(mk) & ... & W_{nn}(mk) \end{bmatrix}. \tag{15}$$

La matriz SMR puede ser calculada de diferentes formas. En este trabajo hemos usado el algoritmo de Siddon [21] para calcular los elementos de la matriz en el grid rectangular. El algoritmo de Siddon se describe a continuación.

4.2.1. Método de Siddon

Para los calculos de los pesos W_{ij} de la matriz SMR 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 24.
En la Figura 24 se presenta un rayo que atraviesa una imagen discretizada entre los puntos P_1 y P_2.

Notaciones.
Los factores de peso del píxel (i, j) se denotan por $L(i, j)$ y son iguales a la longitud de la intersección del rayo con el píxel (i, j). En la Figura 24, N_x y N_y representan el número de planos perpendiculares a los ejes x y y.
El valor de W_{ij} para una proyección se presenta también en la Figura 25.

De esta forma, la integral de línea del punto P_1 al punto P_2 en forma discreta puede ser expresada mediante la formula (16)

$$d_{12} = \sum_{i,j} L(i, j)\rho(i, j). \tag{16}$$

En (16) $\rho(i, j)$ representa el valor de píxel que corresponde a la intensidad de la imagen. Sería muy ineficiente evaluar la ecuación (16) para todos los indices i, j, pues la mayoría de los valores de $L(i, j)$ son ceros. La mejor forma es seguir el rayo al pasar del punto P_1 a P_2.

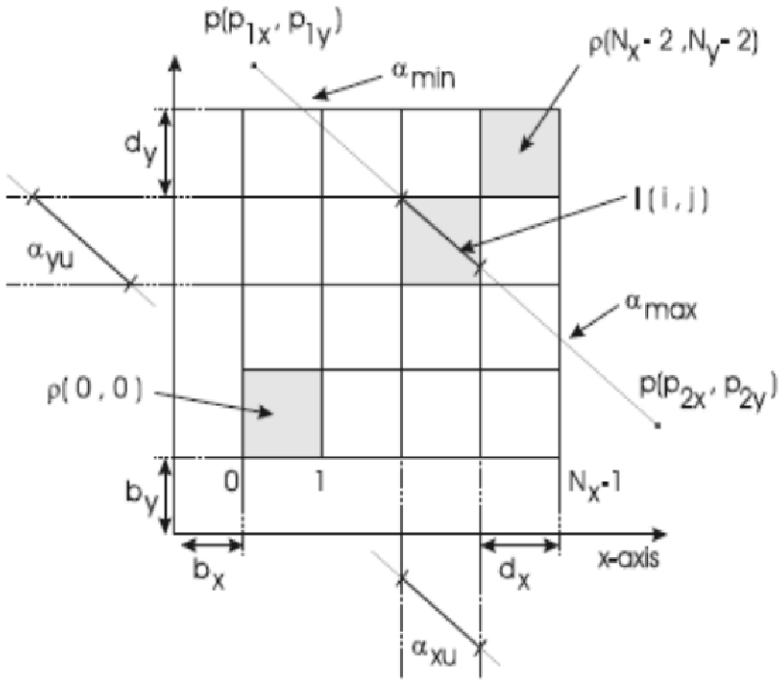

Figura 24: Ilustración del algoritmo de Siddon

Se usa la forma paramétrica del rayo:

$$p_{12} = \left\{ \begin{array}{rcl} p_x(\alpha) & = & p_{1x} + \alpha(p_{2x} - p_{1x}) \\ p_y(\alpha) & = & p_{1y} + \alpha(p_{2y} - p_{1y} \end{array} \right\}. \tag{17}$$

En (17) $\alpha \in [0, 1]$ para puntos entre P_1 y P_2, y $\alpha \notin [0, 1]$ para otros puntos.

Algoritmo de Siddon.
El algoritmo de Siddon está implementado en forma secuencial para generar la matriz del sistema (13) SMR que simula el proceso de escaneo de la imagen. Una vez generada para una imagen de la resolución determinada, la matriz se usa para la resolución iterativa del sistema.

Entrada:
N_x, N_y - resolución de la imagen, $\triangle z$ - incremento angular, D - distancia de la fuente de los rayos -X hasta el centro de la imagen.

Salida:
En la salida el algoritmo genera la matriz del sistema SMR_{Nx3} en el formato coordenado de matrices dispersas (COO):

- En la primera columna se registra el número de fila del elemento no nulo de la matriz SMR.

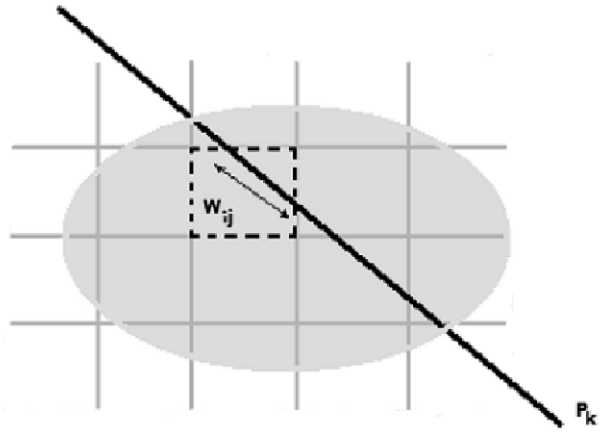

Figura 25: El valor de W_{ij} de la prayeccion P_{k,ϕ_r}

- 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. N indica el número total de elementos no nulos.

Algoritmo:

- **1:** Calcular los parámetros α correspondientes a la intersección de $i-th$ x-plano y $i-th$ y-plane con el rayo que pasa de P_1 a P_2 por las formulas (18)

$$\begin{aligned} \alpha_x(i) &= \frac{(b_x+id_x)-P_{1x}}{P_{2x}-P_{1x}} \\ \alpha_y(i) &= \frac{(b_y+id_y)-P_{1y}}{P_{2y}-P_{1y}}. \end{aligned} \tag{18}$$

- **2:** Determinar los puntos de entrada ($\alpha = \alpha_{min}$) y salida ($\alpha = \alpha_{max}$) por las formulas (19)

$$\begin{aligned} \alpha_{min} &= \text{máx}(\alpha_{xmin}, \alpha_{ymin}) \\ \alpha_{max} &= \text{máx}(\alpha_{xmax}, \alpha_{ymax}) \end{aligned} \tag{19}$$

con

$$\begin{aligned} \alpha_{xmin} &= \text{mín}(\alpha_x(0), \alpha_x(N_x-1)) \\ \alpha_{xmax} &= \text{máx}(\alpha_x(0), \alpha_x(N_x-1)) \\ \alpha_{ymin} &= \text{mín}(\alpha_y(0), \alpha_y(N_y-1)) \\ \alpha_{ymax} &= \text{máx}(\alpha_y(0), \alpha_y(N_y-1)). \end{aligned} \tag{20}$$

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

Para los puntos $P_{1x} < P_{2x}$ usar fórmulas (21), para los puntos $P_{1x} > P_{2x}$ usar (22)

$$\begin{array}{c} \alpha_{min} = \alpha_{xmin} \rightarrow i_{min} = 1 \\ \alpha_{min} \neq \alpha_{xmin} \rightarrow i_{min} = \lceil \phi_x(\alpha_{min}) \rceil \\ \alpha_{max} = \alpha_{xmax} \rightarrow i_{max} = N_x - 1 \\ \alpha_{max} \neg \alpha_{xmax} \rightarrow i_{min} = \lfloor \phi_x(\alpha_{max}) \rfloor \end{array} \tag{21}$$

$$\begin{array}{c} \alpha_{min} = \alpha_{xmin} \rightarrow i_{max} = N_x - 2 \\ \alpha_{min} \neq \alpha_{xmin} \rightarrow i_{max} = \lfloor \phi_x(\alpha_{min}) \rfloor \\ \alpha_{max} = \alpha_{xmax} \rightarrow i_{min} = 0 \\ \alpha_{max} \neg \alpha_{xmax} \rightarrow i_{min} = \lceil \phi_x(\alpha_{max}) \rceil . \end{array} \tag{22}$$

Las definiciones de $\phi_x(\alpha)$ están dadas por la fórmula (23)

$$\phi_x(\alpha) = \frac{P_x(\alpha) - b_x}{d_x}. \tag{23}$$

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

- **4:** Calcular dos conjuntos $\alpha_x[...]$ y $\alpha_y[...]$ con los valores de parámetros de los puntos de intersección del rayo con los planos x y y respectivamente.

  Si $P_{1x} < P_{2x}$ el primer conjunto está dado por la ecuación (24), en caso contrario, por la ecuación (25)

$$\alpha_x[i_{min}...i_{max}] = (\alpha_x(i_{min}), \alpha_x(i_{min}+1), ..., \alpha_x(i_{max})) \tag{24}$$

$$\alpha_x[i_{max}...i_{min}] = (\alpha_x(i_{max}), \alpha_x(i_{max}-1), ..., \alpha_x(i_{min})). \tag{25}$$

- **5:** Ordenar los conjuntos en el orden ascendente y reemplazar los valores dobles por una sola copia. El conjunto resultante $\alpha_{xy}[0, ..., N_v]$ contiene los valores paramétricos de todos los puntos de intersección.

- **6:** Usando el conjunto α_{xy} calculamos las coordenadas de los píxeles intersectados por las fórmulas (26) y las longitudes del rayo, que pasa por este píxel, por las fórmulas (27) para todos los $m \in [1, ..., N_v]$

$$\begin{array}{c} i_m = \left\lfloor \phi_x \left(\frac{\alpha_{xy}[m] + \alpha_{xy}[m-1]}{2} \right) \right\rfloor \\ j_m = \left\lfloor \phi_y \left(\frac{\alpha_{xy}[m] + \alpha_{xy}[m-1]}{2} \right) \right\rfloor \end{array} \tag{26}$$

$$L(i_m, j_m) = (\alpha_{xy}[m] - \alpha_{xy}[m-1])D. \tag{27}$$

En (27) D representa la distancia entre los puntos P_1 y P_2. Los valores $L(i,j)$ representan los factores de peso de un píxel en la matriz del sistema.

El sistema de ecuaciones (13) puede ser sobredeterminado o subdeterminado. Los sistemas sobredeterminados contienen más información sobre la imagen y, consecutivamente, las imágenes reconstruidas son menos ruidosas.

4.2.2. Estudio de la matriz del sistema SMR

La matriz del sistema SMR se define por la formula (15). En la práctica, SMR es una matriz rectangular y de alta dispersidad.

El número de columnas de SMR 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 bajo las cuales se toman las proyecciones. Consecutivamente, las dimensiones de SMR crecen proporcionalmente a la resolución de la imagen y al número de ángulos.

La estructura de la matriz SMR que simula las proyecciones tomadas 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 26.

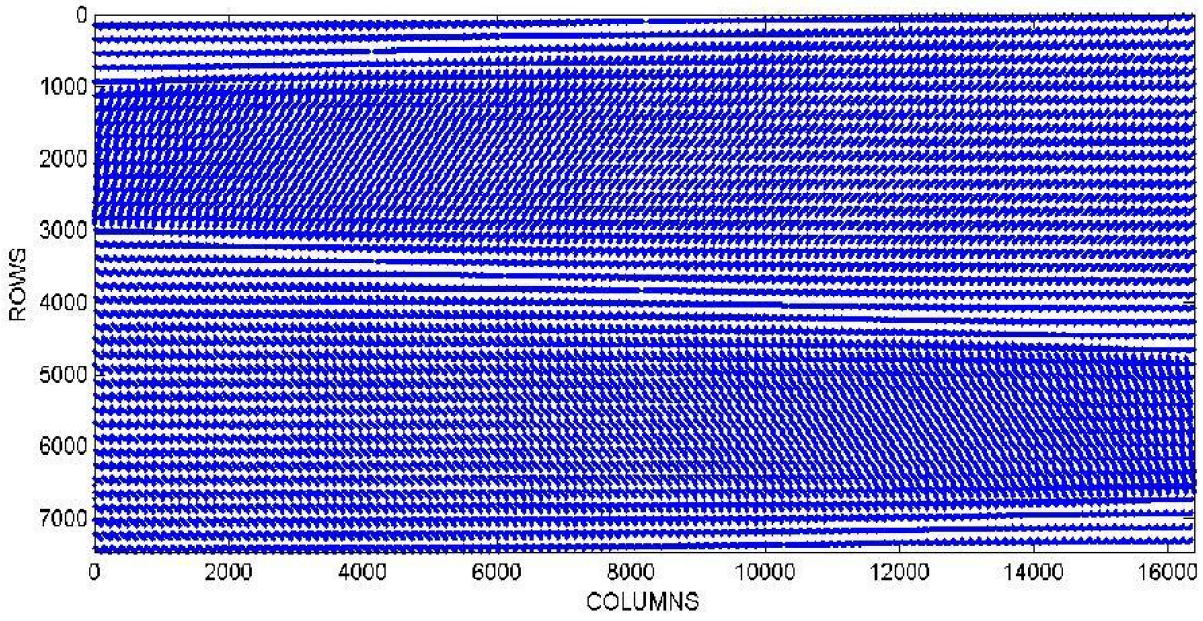

Figura 26: La estructura de SMR

La imagen aumentada de SMR se presenta el la Figura 27.

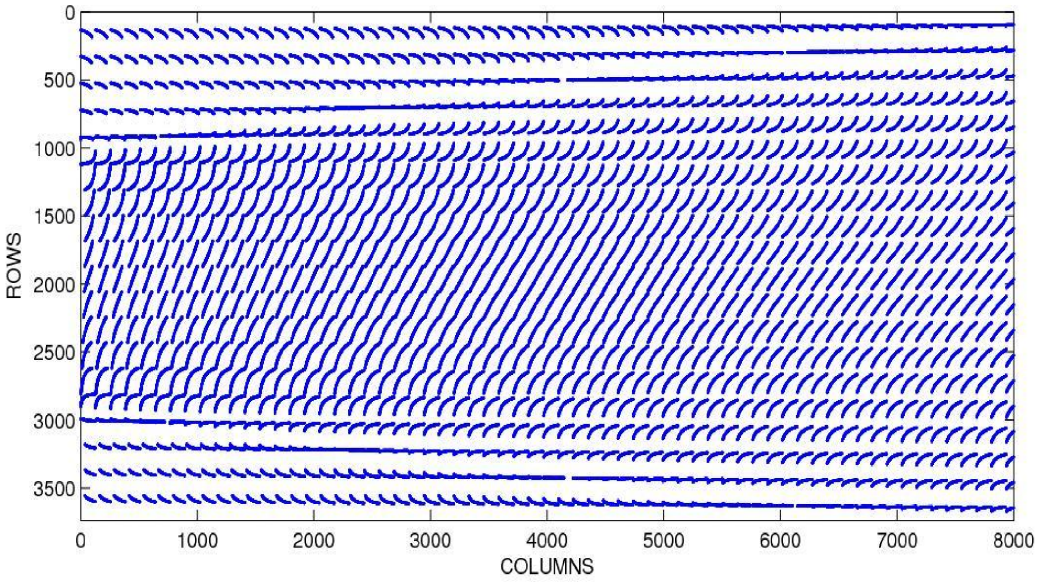

Figura 27: Visión aumentada de SMR

Una parte de la SMR con 1500 filas y 2500 columnas de presenta en la Figura 28.

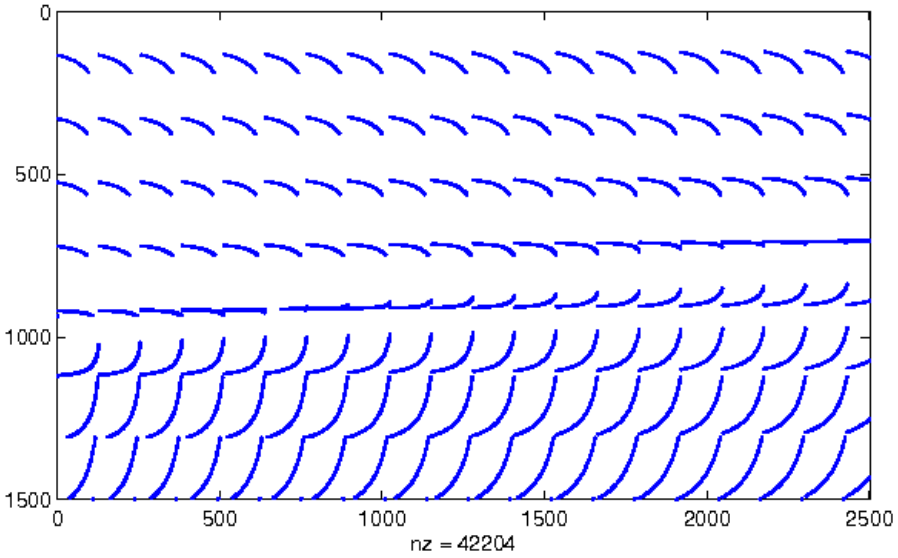

Figura 28: SMR con 1500 filas y 2500 columnas

Se puede concluir que, como se esperaba, la matriz del sistema SMR 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.

4.2.3. Método iterativo LSQR

LSQR [12] es un mètodo iterativo dentro de un conjunto de métodos de Krylov que resuelve un sistema de ecuaciones de la forma:

$$Ax = b \tag{28}$$

y minimiza :

$$min \left\| Ax - b \right\|_2 \tag{29}$$

En (28) A es una matriz real no simétrica de grande dimensión y dispersa con m filas y n columnas, b es un vector real. Es comun que $m > n$ 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 solo se usa para calcular productos del tipo Av y $A^T u$ para varios vectores v y u.

La resolución del sistema de ecuaciones (28) es equivalente a resolver el sistema:

$$A^T Ax = A^T b. \tag{30}$$

En (30) $A^T A$ es simétrica definida positiva. La ecuación (30) recibe el nombre de Ecuación Normal.

El método LSQR está basado en el proceso de bidiagonalización de Golub y Kahan [13]. La idea básica del LSQR es hallar la solución del sistema simétrico:

$$\begin{pmatrix} I & A \\ A^T & -\lambda^2 I \end{pmatrix} \begin{pmatrix} r \\ x \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix} \tag{31}$$

minimizando:

$$\left\| \begin{pmatrix} A \\ \lambda I \end{pmatrix} x - \begin{pmatrix} b \\ 0 \end{pmatrix} \right\|_2, \tag{32}$$

donde λ es un número arbitrario.

El método genera una secuencia de aproximaciones $\{x_k\}$ de tal forma que la norma residual $\left\| r_k \right\|_2$ decrese en forma monótona, donde $r_k = b - Ax_k$. Analíticamente, la secuencia $\{x_k\}$ es identica a la secuencia que se genera con el algoritmo estandar CG. Sin embargo, el método LSQR es numericamente más fiable en la mayoría de los casos.

Pasos generales del algoritmo LSQR.

Sea A es la matriz de mxn, b es el vector que representa el término independiente en la ecuación (28), $\alpha_i \geq 0$, $\beta_i \geq 0$ son escalares que se escogen para normalizar los vectores correspondientes v y u.

1: /** Inicializar
2: $\beta_1 u_1 = b$

3: $\alpha_1 u_1 = A^T u_1$
4: $w_1 = v_1$
5: $x_0 = 0$
6: $\overline{\phi} = \beta_1$
7: $\overline{\rho} = \alpha_1$

/** Proceso de bidiagonalización
8: PARA $i = 1$ HASTA $i = iter_{max}$
9: $\beta_{i+1} u_{i+1} = Av_i - \alpha_i u_i$
10: $\alpha_{i+1} v_{i+1} = A^T u_{i+1} - \beta_{i+1} v_i$

/** Construir y aplicar la siguente transformación ortogonal
11: $\rho_i = (\overline{\rho}_i^2 + \beta_{i+1}^2)^{1/2}$
12: $c_i = \overline{\rho}_i / \rho_i$
13: $s_i = \beta_{i+1} / \rho_i$
14: $\Theta_{i+1} = s_i \alpha_{i+1}$
15: $\overline{\rho}_{i+1} = -c_i \alpha_{i+1}$
16: $\phi_i = c_i \overline{\phi}_i$
17: $\overline{\phi}_{i+1} = s_i \overline{\phi}_i$

/** Update los vectores x y w
18: $x_i = x_{i-1} + (\phi_i / \rho_i) w_i$
19: $w_{i+1} = v_{i+1} - (\Theta_{i+1} / \rho_i) w_i$

/** Aplicar test de convergencia
20: SI
21: Convergencia == True
22: STOP
23: END SI
24: END PARA

Coste de LSQR.
El algoritmo LSQR es un algoritmo iterativo. En cada iteración se realizan las operaciones Av y $A^T u$. Se calcula el coste del algoritmo por una iteración (pasos 8:24).

- $T_1 = \sum_{i=1}^{m} \sum_{j=1}^{n} 1 + \sum_{i=1}^{n} 1 + \sum_{i=1}^{n} \sum_{j=i}^{m} 1 + \sum_{i=1}^{n} 1 + 15$

$$T_1 \approx 2mn + m + n$$

4.3. Algoritmo paralelo

Hemos usado la librería PETSc para resolver el sistema (13). 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 SMR_{mxn}. 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.

- **Entrada.**
  - Matriz S_{Nx3} en el formato COO - salida del algoritmo de Siddon, N - número elementos no nulos.
  - Vector-columna B_{mxk} - proyecciones registradas por el escaner.
- **Salida**
  Matriz X_{nxn} cuyos valores representan intensidades de la imagen reconstruida.

Etapas del algoritmo:

- **Inicialización**
  - Se inicializa el sistema con PetscInitialize()
  - Se leen 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 (1:16).
- **Resolucion del sistema**
  - Se determinan el método de resolución KSP, el precondicionador PC deseado, tolerancia, el numero de iteraciones máximo (17:20). Estos parámentos se puede reescribir a la hora de ejecución (run time).
  - Se resuelve el sistema y se obtiene la información resultante de todo el proceso (21:23).
  - Se genera el archivo de salida de la resolución del sistema (24)
- **Finalización del proceso** (25).

Algoritmo

- **Ensamblaje de la matriz.**

```
1:  MatGetOwnershipRange(A,&Istart,&Iend)
2:  PARA i = Istart HASTA i = Iend − 1
3:      ii = i
4:      PARA k = 0 HASTA k = N − 1
5:          SI ival[k] == ii
6:              jj = jval[k]
7:              aa = aval[k]
8:              MatSetValues(A, 1, &ii, 1, &jj, &aa, INSERT−VALUES)
9:          END SI
10:         SI ival[k] > ii
11:             k = N
12:         END SI
13:     END PARA
14: END PARA
```

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

- **Resolución del sistema**

```
/* Set up el solver y precondicionadores
17: KSPSetOperators(ksp, A, A)
18: KSPGetPC(ksp, &pc)
19: PCSetType(pc, PCJACOBI)
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 %Aiterations %D", norm, its)

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

4.4. Evaluación de coste

La parte más costosa del algoritmo es el ensamblaje de la matriz del sistema SMR, los pasos $(1:16)$.

El coste del ensamblaje de la matriz SMR.
N representa el tamaño del problema y corresponde al número de filas de SMR, M - número de elementos no nulos de la matriz de entrada generada por el método de Siddon, y p - número de procesos.

- Tiempo secuencial de ensamblaje: $T_1 = \sum_{i=0}^{N} \sum_{k=0}^{M} 1 = N * M$
- Tiempo paralelo de ensamblaje: $T_p = \sum_{i=0}^{\frac{N}{p}-1} \sum_{k=0}^{M} 1 = \frac{N}{p} * M$
- SpeedUp teórico: $S_p = \frac{T_1}{T_p} = p$
- SpeedUp límite : $\lim_{N \to \text{ínf}} S_p = p$
- Eficiencia del sistema : $E_f = \frac{S_p}{P} = 1$, $\lim_{N \to \text{ínf}} E_f = 1$

El sistema se resuelve por el algoritmo iterativo LSQR. El coste por una iteración está dado en la parte 4.2.3 de la descripción del algoritmo.

4.5. Metodología

En los experimentos se han usado fantomas de Shepp-Logan de diferentes dimensiones. Para la 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 por menor número de proyecciones.

Las proyecciones que representan el término independiente en el sistema (13), se usan para generar la matriz del sistema SMR por el algoritmo de Siddon. Una vez generado la SMR, se emplea PETSc para obtener la solución del sistema (13).

La solución representa una matriz nxn cuyos valores corresponden a las intensidades de la imagen incógnita. Se ha usado Matlab para visualizar la solución que representa la imagen reconstruida.

Para lograr mejor convergencia del sistema y encontrar la solución con el menor número de iteraciones se analizaron diferentes precondicionadores de la matriz que proporciona la librería PETSc.

4.6. Resultados experimentales.

Para la fantoma de 256x256 píxeles el tamaño resultante de la matiz SMR generada con diferente número de ángulos para los cuales las proyecciones fueron tomadas, está resumido en la Tabla 2. El número de proyecciones se obtiene multiplicando el número de ángulos por el número de detectores.

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

Tabla 2: El tamaño de la matriz del sistema SMR

El tiempo (en segundos) de ensamblaje y de resolución del sistema con diferente número de procesadores N_p y proyecciones N que corresponden a la imagen de 256x256 píxeles se presenta en la Figura 29.

N de	Tiempo de Ensamblaje						Tiempo de Resolución de Sistema					
proyec	Np=1	Np=4	Np=8	Np=16	Np=32	Np=64	Np=1	Np=4	Np=8	Np=16	Np=32	Np=64
120x369	1190	523.1	286.5	200.9	174.8	89.1	0.9	0.4	0.6	0.7	1.0	5.6
90x369	669.6	294.9	157.9	82.9	58.1	49.8	0.7	0.4	0.3	0.5	0.7	5.8
60x369	297.2	131.6	71.1	51.4	43.9	23.1	0.4	0.2	0.3	0.4	0.6	5.4
40x369	132.1	58.1	31.7	22.2	19.1	9.8	0.3	0.2	0.2	0.3	0.5	3.7
36x369	101.2	47.2	25.6	17.0	16.0	8.5	0.3	0.1	0.2	0.3	0.5	3.6

Figura 29: El tiempo de ensamblaje y resolución del sistema

Se observa que el tiempo de resolución del sitema no varía mucho al aumentar el número de procesadores. Al mismo tiempo, el ensamblaje de la matriz del sistema consume mayor tiempo y depende notablemente del número de procesadores. Por lo tanto, se puede concluir que las condiciones óptimas para este caso se logran con 32 procesadores.

4.6.1. Precondicionadores

Para lograr la mejor convergencia de un sistema de ecuaciones, PETSc proporciona diferentes precondicionadores. El objetivo de un precondicionador es mejorar el comportamiento

de la matriz del sistema y resolver el sistema con menor número de iteraciones.

Para analizar el effecto de los precondicionadores en la resolución del nuestro sistema hemos empleado differentes precondicionadores de PETSc al sistema con la SMR del tamaño (64x64)x(97x180) que se obtiene al generar las proyecciones fanbeam del phantoma de 64x64 píxeles en el rango de 0 a 360 grados con 2 grados del paso angular.
En la Tabla 3 se resume el número de iteraciones y el tiempo resultante al emplear los precondicionadores al algoritmo.

Precondicionador	Identificador	Número de iteraciones	Tiempo (en segundos)
None	'none'	27	49.4
Jacobi	'jacobi'	8	73.6
SOR	'sor'	7	77.4
Incomplete Cholesky	'icc'	3	233
Hypre	'hypre'	8	86

Tabla 3: El número de iteraciones y el tiempo resultante con precondicionadores

Las imágenes reconstruidas empleando los precondicionadores de la Tabla 3 se presentan en la Figura 30.

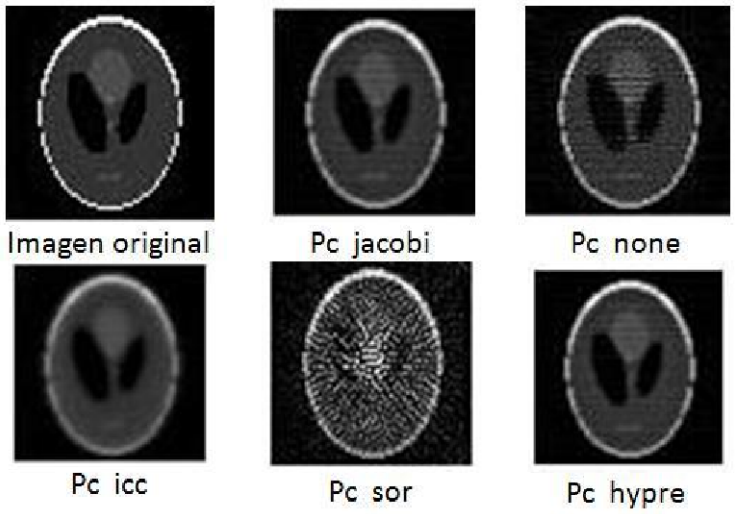

Figura 30: Reconstrucción con precondicionadores

El empleo de los precondicionadores en nuestro problema, con la matriz SMR no cuadrada, permite reducir el número de iteraciones, pero el tiempo de resolución aumenta considerablemente. Los resultados que se presentan en la Figura 29 fueron obtenidos sin precondicionar la matriz del sistema SMR.

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 las Figuras

31 y 32.

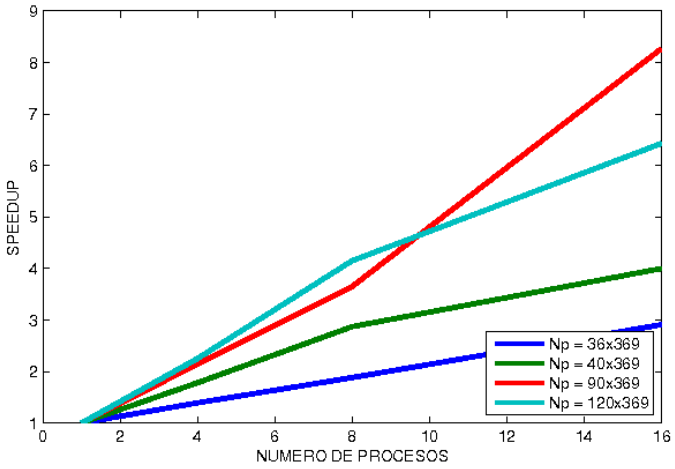

Figura 31: SpeedUp en función del número de proyecciones N_p

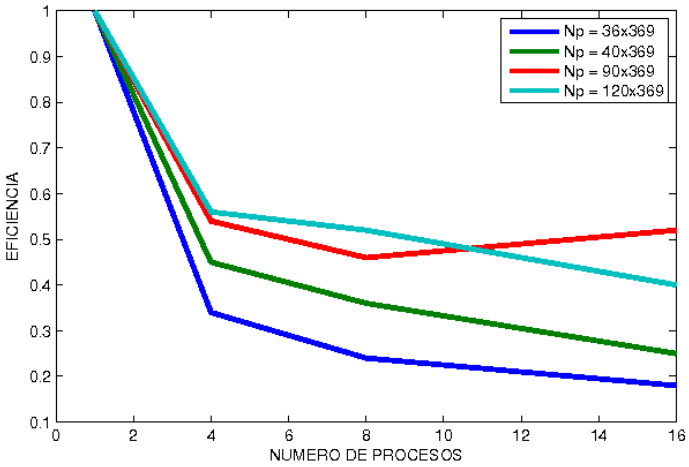

Figura 32: Eficiencia en función del número de proyecciones

4.7. Conclusiones

Analizando los resultados obtenidos en las figuras 31 y 32 se puede concluir:

- La implementación paralela del algoritmo lleva a la reducción notable del tiempo de ejecución.

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

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

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

4.8. Implementación GPU

La utilización de GPU con la enorme potencial de computo paralelo puede elevar considerablemente la eficiencia del algoritmo. La tarjeta GPU dedicada a la computación científica, como la de NVIDIA Tesla K20c se usó en este trabajo para llevar a cabo los experimentos. La tarjeta tiene en total 2496 cuda cores con 5GB de memoria compartida.

Para la reconstrucción de imágenes hemos implementado el mismo algoritmo iterativo LSQR descrito en la subsección 4.2.3. Hemos utilizado el modelo de programación CUDA junto con las librerías CUBLAS y CUSPARSE. El modelo permite resolver muchos problemas complejos computacionalmente de un modo más eficiente que en CPU. Las librerías CUBLAS and CUSPARSE por medio de las funciones proporcionan al usuario el acceso a los recursos computacionales de unidades de procesamiento gráfico de NVIDIA (GPUs) y realización de varias operaciones algebraicas con matrices dispersas. Para usar las librerías, la aplicación tiene que colocar las matrices y vectores necesarias en la memoría de GPU, rellenarlos con datos, hacer llamadas a las funciones CUBLAS y CUSPARSE 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.

Las Figuras 33 y 34 ilustran el esquema del algoritmo y la relación entre las librerías usadas.

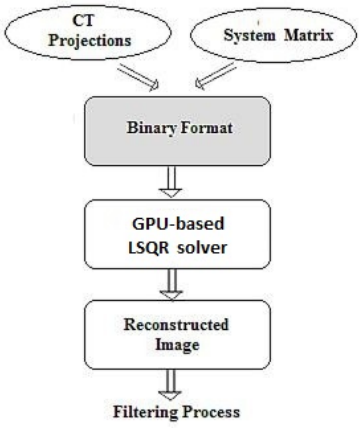

Figura 33: LSQR solver usa los datos de entrada en formato binario para la reconstrucción

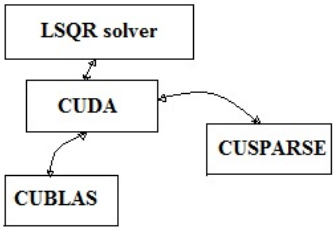

Figura 34: Librerías usadas para la implementación del algoritmo

A continuación, se presenta la parte del código de uso de librerías para calcular la norma de un vector y el producto de matriz por 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 matrix - 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 );
```

Los datos se almacenan en la memoria global de la tarjeta. 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. En nuestra implementacíon, los datos de solo lectura se almacenan en la memoria constante. La memoria más rapida, memoria compartida, se usa para guardar los resultados temporales siempre donde es posible. La figura 35 representa la arquitectura de una tarjeta de NVIDIA GPU.

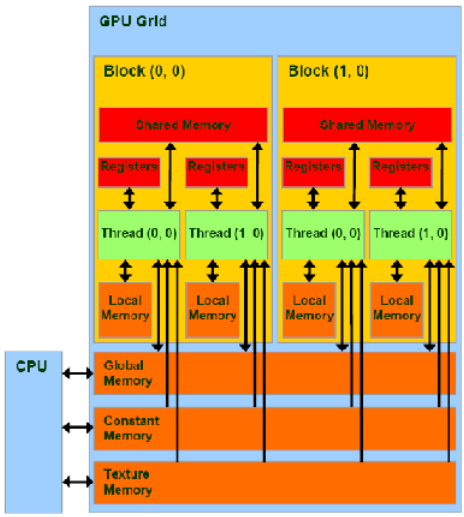

Figura 35: Arquitectura de NVIDIA GPU

4.8.1. Resultados experimentales

Para propósitos experimentales, hemos usado las proyecciones reales y imágenes de referencia adquieridos en el Hospital Clinico Universitario de Valencia. Las proyecciones fan-beam fueron recolectadas por el escáner con 512 detectores en el rango 0 - 180 con espacio angular de 0.9 grados. Para poder reconstruir la imagen por el método iterativo el conjunto dado de proyecciones fué 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 por menor número de proyecciones. Con este fin, del cojunto inicial hemos derivado tres conjuntos de proyecciones equiespaciadas (con el paso angular de 0.9, 1.8, y 3.6 grados).

El experimento se ha llevado acabo en el sistema Gpu.dsic.upv.es que pertenece a la Universidad UPV. Para imágenes de 256x256 y 512x512 píxeles, el tiempo de reconstrucción en CPU con un core y en una unidad GPU se presenta en la Tabla 4. En la GPU, el algoritmo fue ejecutado de dos formas: utilizando solo la memoria global de la tarjeta, y, optimizando el algoritmo, utilizando las memorias constante y compartida. El tiempo de ejecución en CPU fué medido con la función gettimeofday(). En GPU, para medir el tiempo se usó la función cudaEventRecord() que mide tiempo de ejecucion solo en la unidad GPU sin tener en cuenta el tiempo de espera en la cola. La desviación estandar de los resultados después de correr la aplicación 10 veces es 2.9e-004. En la matriz del sistema, el número de filas se obtiene multiplicando el número de detectores por el número de ángulos usados. El número de filas corresponde al número de proyecciones utilizados para la reconstrucción de la imagen. El número de columnas corresponde al tamaño de la imagen reconstruida (256x256 and 512x512 píxeles).

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. Asi mismo, se observa que el algoritmo es escalable ya que se hace más eficiente para problemas de mayor escala. Finalmente, Figura36 muestra las imágenes reconstruidas por diferente número de proyecciones. Usualmente, después de la reconstrucción se aplica postprocesamiento (como filtrado) con el objetivo de mejorar la calidad de la imagen reconstruida. En este trabajo se presentan imágenes immediatamente después de la reconstrucción sin ningún filtrado.

Matriz del sistema (filas x columnas)	CPU (un core) (segundos)	GPU (M. Global) (segundos)	GPU (M. Const. y Compartida) (segundos)
(256x100) x (256x256)	2.7	0.1569	0.10
(256x200) x (256x256)	5.3	0.3056	0.18
(256x400) x (256x256)	10.5	0.6127	0.32
(512x100) x (512x512)	12.3	0.6584	0.33
(512x200) x (512x512)	24.4	1.2741	0.67

Tabla 4: El tiempo de reconstrucción en CPU y GPU en Gpu.dsic.upv.es

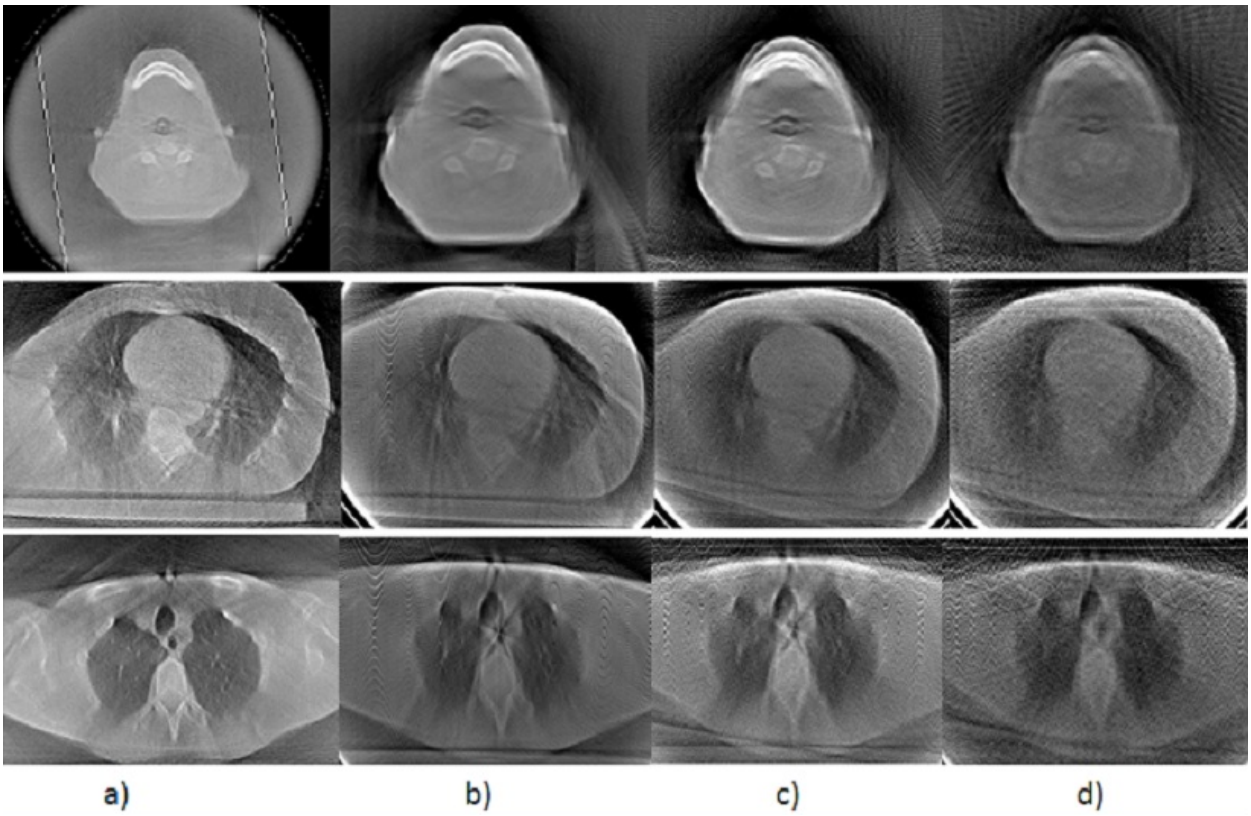

Figura 36: Imágenes reconstruidas: a) imágenes de referencia, b), c), d) reconstrucción iterativa por 400, 200 y 100 ángulos in la iteración 12 cuándo se logra la tolerancia indicada

4.8.2. Conclusiones

El algoritmo de reconstrucción basado en GPU 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 complejas.

Capítulo 5

5. Análisis de la calidad en las imágenes reconstruidas

5.1. Comparación de calidad

Después de la implementación del código, se realizó la comparación de la calidad entre las imágenes original I_1 y reconstruida I_2. Se presenta a modo de ejemplo una imagen sintética adquirida en formato DICOM y reconstruida por los dos métodos, método analítico basado en la transformada inversa de Radon y método iterativo LSQR. Como medidas de comparación de calidad se han usado el Error Medio Cuadratico (MSE) y el Peak Signal-to-Noise Ratio($PSNR$) que se definen por las ecuaciones (33) y (34)

$$MSE = \frac{1}{nxn} \sum_{i=1}^{n} \sum_{j=1}^{n} [I_1(i,j) - I_2(i,j)]^2 \tag{33}$$

$$PSNR = 10 \log_{10} \left(\frac{MAX_I^2}{MSE} \right). \tag{34}$$

En (33) n corresponde a la resolución nxn píxeles de la imagen reconstruida, MAX_I es el máximo valor posible del píxel. Los resultados de comparación de calidad entre las imágenes original y reconstruida por los dos métodos están dados en la Tabla 5.

Número de proyecciones	Método algebraico		Método analítico	
	MSE	PSNR	MSE	PSNR
120x289	0.0067	69.8575	0.0317	63.1266
90x289	0.0080	69.1110	0.0322	63.0545
60x289	0.0098	68.2258	0.0343	62.7746
40x289	0.0127	67.0943	0.0386	62.2701
36x289	0.0125	67.1733	0.0405	62.0568

Tabla 5: Comparación de calidad de imágenes reconstruidas

Las imágenes reconstruidas por los dos métodos, basado en la transformada inversa de Radon y LSQR, por diferente número de proyecciones se presentan en la Figura 37.

5.2. Conclusiones

Los resultados de la Tabla 5 y Figura 37 muestran la capacidad de los métodos algebraicos para reconstruir imágenes de buena calidad con menor número de proyecciones con respecto a métodos analíticos.

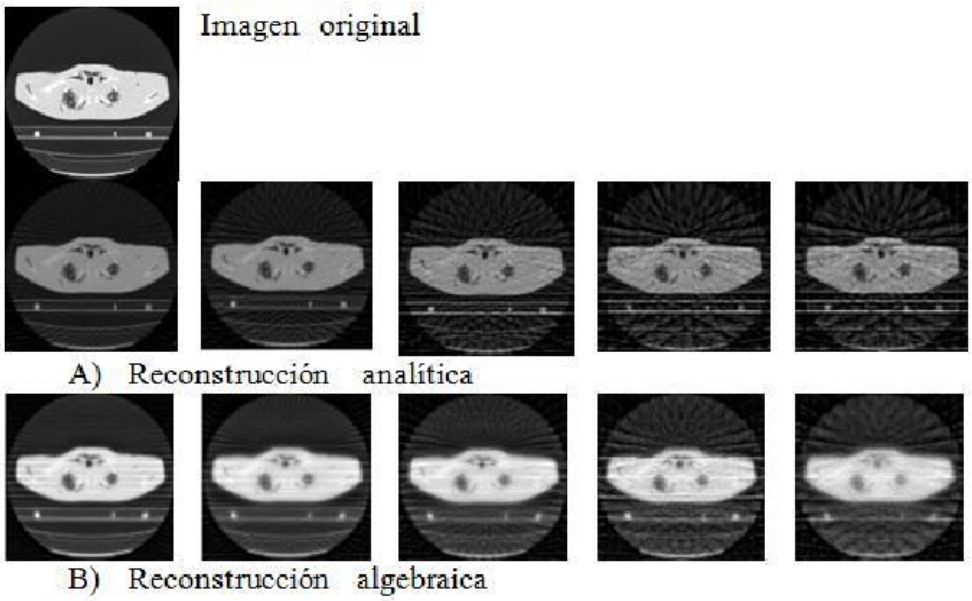

Figura 37: Imágenes reconstruidas por 120, 90, 60, 40 y 36 ángulos.

Los métodos analíticos son superiores cuando se tiene información completa sobre un objeto escaneado o cuando se dispone de un conjunto completo de proyecciones por las cuales se reconstruye la estructura interna del objeto.

Capítulo 6

6. Conclusiones y trabajos futuros

6.1. Conclusiones

En este trabajo hemos analizado el método analítico (algoritmo basado en la inversa de la transformada de Radon), y el método algebraico (algoritmo iterativo LSQR) en la reconstrucción de imagenes TAC. Los resultados obtenidos nos llevan a las siguentes conclusiones:

- Los dos algoritmos son paralelizables, son algoritmos que permiten explotar las características de las arquitecturas modernas de sistemas como sistemas multiprocesadores y multicores.
- La paralelización de los algoritmos lleva a la reducción de tiempo de reconstrucción de imágenes TAC.
- 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.
- El algoritmo basado en la inversa de la transformada de Radon es un algoritmo rápido, pero necesita una completitud de datos para la reconstrucción.
- El empleo de la librería PETSC facilita la implemetación de algoritmos paralelos.
- Los métodos algebraicos reconstruyen la imagen de mejor calidad por menor número de proyecciones y presentan una opción dominante en la actualidad.

6.2. Trabajos futuros

Los avances tecnológicos posibilitan el desarrollo constante de algoritmos existentes y aparicíon de algoritmos nuevos de reconstrución de imágenes. Como muestran los resultados obtenidos (e.g. Tabla 2), el tamaño de la matriz del sistema de ecuaciones aumenta con mayor número de proyecciones. Es decir, el tamaño de la matriz es proporcional a la resolución de la imagen a recostruir, 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, crecen el tiempo de ejecución y 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:

- Comparación con otros métodos de reconstrucción, en particular, con métodos estadísticos y multigrid.
- Investigación de otros métodos de generación de la matriz del sistema SMR con el objetivo de reducir su tamaño y requerimientos de la memoria del sistema.
- Aprovechar la simetría en la estructura de la matriz del sistema SMR y otros formatos más compactos para su almacenamiento.
- Investigación de otras librerías y adaptación de sus módulos en la resolución de sistemas de ecuaciones en paralelo.
- El empleo de GPUs en la reconstrucción paralela de imágenes en 3D con métodos algebraicos.
- Explorar la utilización de las diferentes memorias de GPU.

Los resultados de este trabajo han dado lugar a los artículos donde se describen las técnicas utilizadas:

Fast Parallel Algorithm for CT Image Reconstruction. Liubov A. Flores, Vicent Vidal, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú. Proceedings of 34th Annual International Conference of the IEEE Engieneering in Medicine & Biology Society. August 28-September 1, 2012 San Diego, p. 4374-4377.

Iterative Reconstruction of CT Images with PETSc, Liubov A. Flores, Vicent Vidal, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú. Proceedings of The 4th International Conference on BioMedical Engineering and Informatics 15-17 October 2011, Shanghai, China., Volume 1, p 343-346. IEEE 2011.

Una implementación paralela eficiente de la transformada de Radón Inversa para la reconstrucción de imágenes médicas, L.A. Flores, V. Vidal Gimeno, P. Mayo Nogueira, F. Rodenas Escriba, G. Verdú Martín - aceptada la publicación en la revista universitaria de la Universidad Nacional de Ingenieria, Lima-Perú, 2012 .

Contrast of two methods of reconstruction of CT images using high performance computing, Liubov A. Flores, Vicent Vidal, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú. El artículo fue aceptado para hacer la presentación en la conferencia Mathematical Modelling in Engineering & Human Behaviour, Valencia, September 6th-9th, 2011.

Algoritmo Paralelo de Reconstrucción de Imágenes TAC. Liubov A. Flores, Vicent Vidal, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú - ponencia en la 38 REUNION ANUAL SOCIEDAD NUCLEAR ESPAÑOLA, Cáceres, 17-19 Octubre 2012

Reconstrucción Iterativa de Imágenes de TAC Mediante Computación de Altas Prestaciones, Liubov A. Flores, Vicent Vidal, Patricia Mayo, Francisco Rodenas, Gumersindo Verdú - ponencia en la 37 REUNION ANUAL SOCIEDAD NUCLEAR ESPAÑOLA, Burgos, 28-30 Septiembre 2011.

Referencias

[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 M. Slaney. *Principles of Computerized Tomographic Imaging.* IEEE Press, 1988.

[5] G. Wang, H.Yu, and B. De Man. *An outlook on X-ray CT research and development.* Medical Physics, 35(3):1051-1064, Mar. 2008.

[6] 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., 25:78-94, 2007.

[7] 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.,43:729-737, 1998.

[8] 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, 5(1):58-65, Jul 1999.

[9] A. H. Andersen. *Algebraic reconstruction in CT from limited views.* IEEE Trans. Med. Imaging, 8(1), 1989.

[10] G. Wang, D. Snyder, J. O'Sullivan, and M. Vannier. *Iterative deblurring for CT metal artifact reduction.* IEEE. Trans. Med. Imaging, 15(5):657-663, Oct. 1996.

[11] N. Sinha and J. T. W. Yeow. *Carbon nanotubes for biomedical applications.* IEEE Trans. Nano., 4(2):180-196, 2005.

[12] C.C. Paige and M.A. Saunders. *The Algorithm LSQR:Sparse linear equations and least square problems.* ACM Trans. Math. Soft. 8,2, 1982.

[13] G.H. Golub and W. Kahan. *Calculating the singular values and pseudoinverse of a matrix.* SIAM J. Numer. Anal. 2, 205-224, 1965.

[14] M.J. Flynn. *Some computer organisations and their effectiveness*, IEEE Trans. On Computers, Vol. c-21, 9,948-960, 1972.

[15] Aad J. van der Steen. *Overview of recent supercomputers.* HPC Research, August 2008.

[16] http://www.intel.com/es ES/products/server/processor/index.htm

[17] http://www.amd.com/us/products/desktop/processors/Pages/desktop-processors.aspx

[18] http://acts.nersc.gov/petsc/index.html

[19] L.Dagum, R.Menon. *OpenMP: an industry standard API for shared-memory programming*. Computational Science & Engineering, IEEE.

[20] http://www.mcs.anl.gov/research/projects/mpi/

[21] R. L. Siddon, *Fast calculation of the exact radiological path for a threedimensional CT array*. Med Phys 12 252-255, 1986.

[22] NVIDIA. *NVIDIA CUDA Programming Guide*, Version 2.0, 06/07/2008.

[23] http://www.nvidia.com/content/PDF/fermiwhite papers/NVIDIA Fermi Compute Architecture Whitepaper.pdf

[24] http://www.nvidia.es/object/geforce family es.html

[25] Stone S. S., Haldar J. P., Tsao S.C., Hwu W.-m W., Sutton B. P., Liang Z. P., 2008. Accelerating advanced MRI reconstructions on GPUs. Journal of Parallel and Distributed Computing, vol. 68, issue 10, 1307-1318

[26] Johnson C.A., Sofer. A., 1999. A data-parallel algorithm for iterative tomographic image reconstruction. Frontiers of Massively Parallel Computation, pp. 126-137.

[27] Pratx G., Chinn G., Olcott P.D., Levin C. S., 2009. Fast, Accurate and Shift-Varying Line Projections for Iterative Reconstruction Using the GPU. IEEE Transactions on Medical Imaging, 28(3), pp. 435-445.

[28] Jang B, Kaeli D., Do S., Pien H., 2009. Multi GPU implementation of iterative tomographic reconstruction algorithms. Biomedical Imaging: From Nano to Macro, pp. 185-188.

[29] http://www.netlib.org/blas/

[30] Cuda C Programming Guide. Downloaded in Oct. 2012 from URL http://docs.nvidia.com/cuda/index.html.

[31] CUBLAS Library. Downloaded in Oct. 2012 from URL http://docs.nvidia.com/cuda/index.html.

[32] CUSPARSE Library. Downloaded in Oct. 2012 from URL http://docs.nvidia.com/cuda/index.html