PROYECTO FIN DE CARRERA

INGENIERÍA TÉCNICA EN INFORMATICA DE GESTIÓN

Paralelización de un algoritmo de reconstrucción tomográfica de rayos X para plataformas híbridas basadas en multi-GPU y multi-core

Autor: Ernesto Liria Muñoz

Tutores: Francisco Javier García Blas

Mónica Abella García

Leganés, Marzo de 2012

Título: Paralelización de un algoritmo de reconstrucción tomográficade rayos X para plataformas híbridas basadas en multi-GPU y multi-core

Autor: Liria Muñoz, Ernesto.

Tutores: García Blas, Francisco Javier.

Abella García, Mónica.

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día __ de _____ de 20__ en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

En primer lugar tengo que dar las gracias a mi familia, sobre todo a mis padres, Juan Carlos y María Ángeles, que me han dado todo cuanto he necesitado y que me han apoyado día tras día, ayudándome a alcanzar mis metas.

Mi hermano Carlos, con la cantidad de tonterías y discusiones que hemos tenido sobre lo mejor que era la informática frente a la carrera que él estudió, aunque ahora me esclavice para que repare siempre su ordenador.

Agradecer también a Javier, mi tutor de este proyecto, por toda la paciencia que ha tenido conmigo desde el inicio hasta el final, por los buenos ratos y las risas junto al resto del departamento de ARCOS. Por las inmejorables clases de arquitectura de computadores y las curiosidades que despertaste en mi, tienes mi más profundo respeto y admiración como alumno y como compañero tuyo.

A Mónica, codirectora del proyecto y representante del laboratorio, porque gracias a ella y el resto de su equipo he podido acceder a este proyecto y esta oportunidad de aumentar mis conocimientos, con todas las posibilidades que ello me ha reportado y que día a día siguen apareciendo.

Andrés, Laura y Katherin, apoyo incondicional ycompañeros de fatigas de principio a fin, por todas las horas de sufrimiento haciendo prácticas y preparando los exámenes. Gracias por todas las horas de risas y los ratos amenos en la cafetería, sin ellos no habría sido lo mismo y quizá hubiera llegado a abandonar la carrera.

Tampoco puedo olvidarme de todos mis amigos fuera de la universidad, ellos son los verdaderos sufridores de mis malos ratos durante la carrera, mis quejas, mis protestas y mis desesperaciones por una práctica que no me sale, una asignatura que me aburre o un examen que no quiero estudiar. Esos ratos para desconectar, tomando algo en cualquier parte creyendo que somos capaces de resolver el mundo, hablando de todo y nada.

Gracias a todos y cada uno de vosotros que sois importantes en mi vida y que a vuestra manera habéis hecho que hoy sea lo que soy.

Por último quiero dedicar unas palabras a todos aquellos que pensaban que no sería capaz de lograrlo, que me rendiría, todos cuantos dudaron de mi determinación por seguir hasta el final, gracias porque eso me hizo tirar de orgullo y no bajar los brazos para que ahora os pueda responder a cada uno de vosotros con un enorme “Te lo dije”.

RESUMEN

En el presente Proyecto de Fin de Carrera se aborda la necesidad de paralelización y optimización tanto de la memoria como de los recursos de procesamiento disponibles para reducir al mínimo posible el tiempo de procesamiento en la ejecución de una aplicación de imagen médica.

La aplicación bajo estudio es la reconstrucción de imagen de tomografía de rayos X (TAC) basada en geometría de haz cónico. La TAC es una modalidad de imagen médica basada en el uso de rayos X para obtener imágenes de cortes del cuerpo a estudiar. En vez de obtener una única imagen plana (proyección), como es el caso de la radiografía convencional, en la TAC se obtiene un conjunto de proyecciones en distintos ángulos alrededor del cuerpo.

Posteriormente, la computadora recoge todos estos datos de proyección y los combina en un volumen final que representa la reconstrucción digital 3D del cuerpo, permitiendo obtener cortes del mismo en cualquier dirección.

Para lograr el máximo rendimiento se han tenido presentes las distintas etapas por las que transcurre el proceso de reconstrucción y, por tanto, la necesidad de buscar una infraestructura óptima para cada una de ellas, dividiendo así el trabajo entre las tareas que deben ser realizadas por la CPU y las que deben ser realizadas dentro de la GPU.Además, para lograr los resultados más satisfactorios se han utilizado otro tipo de técnicas como la paralelización, mecanismos de entrada y salida asíncronos y alineamiento de memoria.

ABSTRACT

This work addresses the need of parallelization and optimization of both memory and the resources available to minimize the processing time needed in a medical imaging application.

This application is the tomographic image reconstruction of data obtained with X-ray computed tomography (CT) systems based on cone-beam geometry. CT is a medical imaging modality based on the use of X-rays to provide images of slices of the body under study. Instead of acquiring a single image (projection), as in the case of conventional radiography, in CT we acquire a set of projections at different angles.

Subsequently, the computer combines these projection data into a final 3D reconstructed volume, visualizing planes inside the body in any direction.

For maximum performance, the different stages of the reconstruction process have been taken into account in order to find the optimal infrastructure for each of them. As a result, the work is divided among the tasks to be performed by the CPU and those to be performed within the GPU.

Furthermore, in order to achieve more satisfactory results, we have used other techniques such as parallelization, I/O mechanisms, and alignment of asynchronous memory.

Contenido

1. INTRODUCCIÓN 13
   1.1 Motivación 13
   1.2 Objetivos 15
   1.3 Estructura del documento 15
   1.4 Glosario de términos 16
2. ESTADO DEL ARTE 18
   2.1 Trabajos relacionados 19
   2.2 CUDA y GPU 19
   2.3 OpenCL 23
   2.4 OpenGL como alternativa a CUDA 23
   2.5 Computación paralela y OpenMP 25
   2.6 *Pinned memory* (memoria paginada vs. memoria no paginada) 27
   2.7 AIO (Entrada/salida asíncrona) 29
3. DISEÑO DETALLADO DEL SISTEMA 32
   3.1 Análisis de requisitos 36
4. CATÁLOGO DE OPTIMIZACIONES 44
   4.1 Puesta a punto inicial 44
      4.1.1 Primera fase: división de arquitecturas 45
      4.1.2 Segunda fase: estructura del código fuente 45
      4.1.3 Tercera fase: definición del número de hilos 46
   4.2 Entrada / salida asíncrona 47
   4.3 Localidad en memoria 48
   4.4 Uso de memoria compartida en GPU 48
   4.5 Paralelización basada en OpenMP 49
   4.6 Hilos según tamaño del bloque 50
   4.7 Implementación modular sobre GPU 51
5. VERIFICACIÓN DEL SISTEMA 53
6. EVALUACIÓN DE RENDIMIENTO 58
   6.1 Especificaciones hardware 58
   6.2 Casos de estudio 60
   6.3 Lectura y escritura asíncronas 62
   6.4 Lectura adelantada 63

6.5 Selección del planificador de hilos de OpenMP 65
6.6 Paralelismo con multi-GPU 67
6.6.1 Proyecciones de 512x512 puntos 67
6.6.2 Proyecciones de 2048x2048 puntos 70
6.6.3 Proyecciones 526x526 – 1 cama 71
6.6.4 Proyecciones 526x526 – 2 camas 73
6.7 Comparativa de tiempos en paralelismo GPU 76
6.8 Heterogeneidad de hardware 78
6.8.1 GTX 480 + TESLA c2050 (512x512) 78
6.8.2 GTX 480 + TESLA c2050 (2048x2048) 79
6.8.3 GTX 480 + GTX 470 + TESLA c2050 (512x512) 80
6.8.4 GTX 480 + GTX 470 + TESLA c2050 (2048x2048) 81
6.8.5 GPUS SEGÚN EL TAMAÑO DE LA PROYECCIÓN 82
6.9 CPU vs GPU 84
7. CONCLUSIONES Y PRESUPUESTO 89
7.1 Presupuesto 91
BIBLIOGRAFÍA 94
APÉNDICE I 96
APENDICE II 101

Tabla de ilustraciones

Figura 1: Arquitectura GPU 21
Figura 2: Grid GPU 22
Figura 3: Pipeline del procesamiento gráfico de OpenGL 24
Figura 4: eXplore PET/CT 33
Figura 5: Esquema de componentes 33
Figura 6: Perspectiva de la arquitectura de haz cónico en CT. SO es la distancia entre el emisor y el origen; OD es la distancia entre el detector y el origen 33
Figura 8: Implementación propuesta para Mangoose basada en múltiples GPU y multi-core 36
Figura 9: Núcleo reconstrucción de Mangoose Plus 46
Figura 10: Manejador genérico modular para sistemas multi-GPU 52
Figura 11: Perspectiva ImageJ 54
Figura 12: Datos punto basados en coordenadas en ImageJ 54
Figura 13: CUDA Profiler 55
Figura 18: Escápula cocodrilo 60
Figura 19: Maxilar cerdo 61
Figura 20: Jeringuilla 61
Figura 21: Lectura adelantada sobre proyecciones de 512x512 puntos. 63
Figura 22: Lectura adelantada sobre proyecciones de 2048x2048 puntos. 64
Figura 23: Lectura adelantada sobre proyecciones de 564x564 puntos. 65
Figura 24: Schedule dynamic vs static 66
Figura 25: Paralelismo basado en multi-GPU sobre proyecciones de 512x512 puntos. 68
Figura 26: Paralelismo GPU 512x512 (RF y BP) 69
Figura 27: Paralelismo multi-GPU para proyecciones de 2048x2048 puntos y volumen generado de 512x512x512 puntos. 70
Figura 28: Paralelismo GPU 2048x2048 puntos (RF y BP) 71
Figura 29: Paralelismo GPU 526x526 - 1 cama 72
Figura 30: Paralelismo GPU 526x526 - 1 cama (RF y BP) 73
Figura 31: Paralelismo GPU 526x526 - 2 camas. 74
Figura 32: Paralelismo GPU 526x526 - 2 camas (RF y BP) 74
Figura 33: Paralelismo GPU 526x526 - 2 camas (Solape y Desviación) 75
Figura 34: Comparación de tiempos paralelismo GPU (512x512) 76
Figura 35: Comparación de tiempos paralelismo GPU (2048x2048) 77
Figura 44: Tiempos paralelos CPU primer nivel 85
Figura 45: Tiempos paralelos CPU segundo nivel 86
Figura 46: Ratios de aceleración en paralelismo CPU 86
Figura 47: CPU vs GPU 512x512 87
Figura 48: CPU vs GPU 2048x2048 88

Tablas

Tabla 1: Glosario de términos 17
Tabla 2: Requisitos– 1 37
Tabla 3: Requisitos –2 37
Tabla 4: Requisitos – 3 38
Tabla 5: Requisitos – 4 38
Tabla 6: Requisitos –5 38
Tabla 7: Requisitos–6 38
Tabla 8: Requisitos – 7 39
Tabla 9: Requisitos– 8 39
Tabla 10: Requisitos– 9 39
Tabla 11: Requisitos – 10 40
Tabla 12: Requisitos – 11 40
Tabla 13: Requisitos–12 40
Tabla 14: Requisitos – 13 41
Tabla 15: Requisitos– 14 41
Tabla 16: Requisitos - 15 41
Tabla 17: Requisitos – 16, ... 41
Tabla 18: Requisitos - 17 42
Tabla 19: Requisitos - 18 42
Tabla 20: Requisitos– 19 42
Tabla 21: Requisitos - 20 43
Tabla 22: Checklist 57
Tabla 23: Especificaciones GTX 470 59
Tabla 24: Especificaciones Tesla c2050 59
Tabla 25: Especificaciones GTX 480 59
Tabla 26: Especificaciones Tesla c1060 60
Tabla 27: Lecturas asíncronas a través de AIO Read 62
Tabla 28: Escrituras asícronas a través de AIO Write 62
Tabla 29: Schedule - dynamic vs static 66
Tabla 30: Diferenciatiempos 2048x2048 83
Tabla 31: Diferenciatiempos512x512 84
Tabla 32: Presupuesto Hardware 92
Tabla 33: Presupuesto Software 92
Tabla 34: Presupuesto amortizado 92
Tabla 35: Presupuesto personal 93
Tabla 36: Presupuesto final 93
Tabla 37: Hilos / Bloque (1 Hilo) 101
Tabla 38: Hilos / Bloque (2 Hilos) 102
Tabla 39: Hilos / Bloque (4 Hilos) 103
Tabla 40: Hilos / Bloque (8 Hilos) 103
Tabla 41: Hilos / Bloque (16 Hilos) 104

1.INTRODUCCIÓN

En estecapítuloinicial se expone la visión general de este Proyecto Fin de Carrera, cuál ha sido la motivación para realizarlo, qué necesidad tiene, por qué puede resultar útil y por último qué objetivos pretenden alcanzarse.

1.1 Motivación

La tomografía axial computarizada de rayos X(TAC) es uno de los procedimientos de diagnóstico y evaluación por imagen médica más utilizados[9]. Conforme ha ido evolucionando la tecnología, los tiempos de adquisición han ido disminuyendo. Por otra parte, la evolución de los paneles detectores ha supuesto un aumento de la densidad de elementos detectores, resultando en una mayor cantidad de datos a procesar.A este aumento del volumen de datos se le añade el requisito de reconstrucciones más rápidas impuesto por los usos actuales delTAC. En este sentido, la planificación y monitorización en radioterapia, la cirugía asistida por imagen y otras modalidades clínicas requieren una respuesta lo más rápida posible.Por contra, los desarrollos en la algorítmica de reconstrucción no han traído avances equivalentes en lo que a tiempos se refiere, convirtiéndose en una barrera para la

ampliación del uso de esta tecnología.Esta problemática motiva el intereés de buscar procedimientos de aceleración que se adecuen a la creciente complejidad y exigencias de la reconstrucción.

Este trabajo se basa en una solución al problema citado en el marco del TAC con geometría de haz cónico y trayectoria circular por medio de la aplicación del algoritmo deFeldkamp, Davis y Kress (FDK) [4]. Este algoritmo es la extensión de la retroproyección filtrada para geometría de haz cónico incorporando unos factores de corrección de la longitud de los rayos. Así, los dos componentes principales del algoritmo son las fases de filtrado y retroproyección. Por su mayor complejidad algorítmica, es la segunda fase la que consume la mayor parte del tiempo total de procesamiento.

Una posibilidad para acelerar la reconstrucción consiste en emplear algoritmos alternativos. Éstos se pueden clasificar en tres grupos: el primer grupo interpola una retícula polar en otra rectangular en el espacio de Fourier para poder hacer uso de la familia de transformadas rápidas de Fourier (FFT); el segundo grupo acelera la retroproyección mediante procesos recursivos de sumas parciales, tratando todas las proyecciones simultáneamente; el tercer grupo utiliza un enfoque de divide y vencerás, dividiendo la imagen reconstruida en partes, en algún caso utilizando para la división la transformada de Fourier de la imagen. Según sus autores, algunos de estos algoritmos llegan a multiplicar por 40 la velocidad de reconstrucción, aunque se debate sobre la calidad de las reconstruccionesy su falta de generalidad al depender de las propiedades de la imagen[11].

Otro enfoque es la aplicación de técnicas de computación paralela. Para ello existen multitud de enfoques. Algunos de ellos son rígidos y de coste elevado, como el uso de circuitos integrados específicos de la aplicación (ASIC) o de dispositivos FPGA [12,13,14].

Actualmente, una alternativa que está usándose con éxito, es la programación sobre unidades de procesamiento gráfico (GPU), las cuales permiten explotar precisamente la clase de paralelismo que se da en la retroproyección. Con un coste relativamente bajo y una potencia de cómputo creciente, resultan ser herramientas casi perfectas para este tipo de tareas. Para facilitar su manejo, en lugar de programar directamente con las librerías para tratamiento de gráficos, hay lenguajes específicos para aprovechar su potencia, como es el caso de CUDA.

En este Proyecto Fin de Carrera se presenta unaimplementación hibrida basada en multi-GPU y multi-core del algoritmo FDK, partiendo de la implementación base en el lenguaje C conocida como Mangoose. El algoritmo permite aprovechar características de multiprocesador y, en caso de detector una GPU disponible compatible con CUDA, acelera las dos etapas principales del algoritmo: para el filtrado se utiliza la función CUFFT, disponible en el *toolbox* de CUDA; para la etapa de retroproyección las proyecciones filtradas se cargan secuencialmente en memoria de textura para utilizar la eficiente interpolación proporcionada por la GPU.

1.2 Objetivos

El objetivo principal de este Proyecto Fin de Carrera es diseñar e implementar una versión del algoritmo FDK partiendo de la implementación en C, llamada Mangoose, que permita realizar la misma reconstrucción que en su versión inicial pero en menor tiempo, gracias a la optimización en la utilización de recursos. A su vez este proyecto tiene los siguientes objetivos secundarios:

- Desarrollar un modelo programado que acepte arquitecturas heterogéneas para una completa adaptación a cualquier tipo de sistemas.
- Diseñar una implementación viable para estructuras tanto en CPU como GPU, aplicando programación híbrida basada en OpenMP y CUDA.
- Diseñar una versión que utilice únicamente los recursos de CPU para aquellos sistemas en los que no estén presentes dispositivos GPU.
- Emplear técnicas de lectura y escritura asíncrona para ocultar el tiempo invertido en operaciones de entrada y salida.
- Maximizar el rendimiento de cálculo mediante el uso de APIs especialmente diseñadas para el tratamiento de imágenes, en este caso CUDA.
- Lograr el ratio de aceleración más elevado posible manteniendo siempre la integridad de los datos y garantizando que el volumen obtenido sea idéntico al de su versión inicial, tanto en versiones multi-core como en multi-GPU.

1.3 Estructura del documento

En este apartado se detallan los diferentes capítulos que contiene este documento y en qué consisten:

- Capítulo 1, Introducción: Aporta la idea general del proyecto, con la motivación para realizarlo y los objetivos fundamentales que pretenden alcanzarse
- Capítulo 2, Estado del Arte: Se explica una visión general de la tecnología utilizada para la realización del proyecto.
- Capítulo 3, Diseño detallado: Concepto inicial del reconstructor tomográfico a optimizar, las etapas que lo componen y su finalidad junto con los requisitos necesarios para desempeñar correctamente las funciones.
- Capítulo 4, Catálogo de Optimizaciones: Muestra los pasos realizados y las mejoras efectuadas sobre la versión inicial de Mangoose para lograr un mejor rendimiento.

- Capítulo 4, Entorno de pruebas: Detalla el software utilizado para realizar las pruebas y comprobar el resultado correcto de las reconstrucciones digitales junto con los casos de estudio probados.
- Capítulo 5, Verificación del sistema: Software y soporte utilizado para poder desempeñar la evaluación de rendimiento y corroborar su buen funcionamiento.
- Capítulo 6, Evaluación de rendimiento: Detalla el Hardware utilizado y los rendimientos obtenidos con la aplicación de las optimizaciones y el paralelismo.
- Capítulo 7, Entorno de pruebas: Lista de las pruebas a las que han sido sometidas las reconstrucciones en Mangoose para garantizar el correcto funcionamiento.
- Capítulo 8, Conclusiones, líneas futuras y presupuesto: Breve resolución del proyecto, las ideas extraídas de él y posibles líneas de trabajo futuro. Coste del desempeño y ejecución del proyecto.
- Apéndice: Información adicional complementaria a Mangoose y a sus evaluaciones.

1.4 Glosario de términos

Término	Descripción
CUDA	Herramientas para programar en la GPU utilizadas para el desarrollo del proyecto
FDK	Algoritmo de reconstrucción digital que obtiene su nombre de Feldkamp, Davis y Kress
API	Interfaz de Programación de Aplicaciones, funciones y procesos de un servicio.
Vóxel	Porción del volumen.
AIO	Entrada/Salida Asíncrona.
Ratio de aceleración	Número de veces más rápido frente al más lento, obtenido de dividir el tiempo inicial entre el actual.
GPU	Unidad de Procesamiento Gráfico.
TILE	Sub-matriz en la memoria compartida de la GPU para agilizar los cálculos.
Hounsfield	Unidad de medida de densidad ósea.
OpenMP	Herramientas de programación en paralelo con memoria compartida basada en hilos.
Kernel	Proceso ejecutado por múltiples hilos dentro de una GPGPU.

Mangoose	Nombre del reconstructor tomográfico implementado en el lenguaje C que se toma como referencia en este proyecto.
SDK	Software Developement Kit
Pixel	Menor unidad homogénea en color que forma parte de una imagen digital, también definida como punto.
TAC	Tomografía Axial Computerizada
FOV	Field of view

Tabla 1: Glosario de términos

2. ESTADO DEL ARTE

En este capítulose aportará una visión global de las diversas tecnologías usadas para el desarrollo de este Proyecto Fin de Carrera, así como posibles alternativas. Para ello se describirán detalladamente las tecnologías actuales disponibles que permitan extraer información y utilidad para lograr la eficiencia del reconstructor.

Inicialmente se detalla el modelo de programación CUDA y la relevancia que tiene esta API en los entornos gráficos por su generación masiva de hilos y su idoneidad para los cálculos de matrices. Posteriormente se analizará OpenCL como alternativa viable y plausible para una solución alternativa.

En sistemas de paralelización destaca OpenMP por su versatilidad y la estabilidad para cualquier tipo de arquitectura y plataforma,.

Se aborda también la importancia de la memoria paginada y un análisis sobre su utilidad y para qué casos sería conveniente utilizarla y para qué casos no conviene memoria no paginada.

Por último quedarán detalladas las entradas y salidas asíncronas como recurso para reducir tiempos en un alto porcentaje debido a la gran cantidad que es leída de fichero para, una vez procesada, ser escrita de nuevo.

2.1 Trabajos relacionados

Xu y Mueller [2], abordan la aceleración de la retroproyección en la GPU, utilizando tanto las componentes gráficas aceleradas (AG-GPU), como la configuración de multiprocesador (MP-GPU). Zhao *et al.* [4] emplean esquemas eficientes para manejar conjuntos de datos demasiado grandes para la memoria de la GPU y aprovechan las simetrías de rotación usando los cuatro canales de color de la textura. Schiewietz *et al* [20] incluyen corrección de los artefactos de anillo y *cupping*. Riabkov *et al.* [21] comparan dos implementaciones distintas de retroproyección. Knaup *et al.* [17] dividen el volumen total en partes que caben en la memoria compartida para evitar tiempos de latencia largos al acceder a memoria global y optimizar el uso de la caché de texturas (cada parte necesita un trozo de proyección más pequeño.

Noël *et al.* [19] aprovechan la disponibilidad de memoria compartida, cargando todas las imágenes proyectadas en la memoria de la GPU y computando la intensidad de cada vó*xel* por retroproyección en paralelo; Okitsu *et al.* [16] se centran en la reducción del tiempo de acceso a la memoria externa del dispositivo y en la ocultación de la latencia de memoria. Yan *et al.* [20] aplican una combinación de estrategias de aceleración para ahorrar tiempo de procesamiento de copias y reducir el coste computacional en el mapeo entre rodajas a reconstruir y sus proyecciones. Sherl *et al.* [19] se centran en reducir el número de instrucciones y uso de registros.

2.2 CUDA y GPU

Desde hace unos años, se recurre a la programación directa en GPU *(GraphicsProcessingUnit)* en aplicaciones que requieren un gran coste aritmético gracias a su alta capacidad para el trabajo simultáneo con la generación masiva de hilos y su ancho de banda, reduciendo mucho la latencia y el coste por los múltiples accesos a memoria principal, evitando así los llamados cuellos de botella.

La idea fundamental es explotar el paralelismo ofrecido por los dispositivosGPU en base a sus múltiples núcleos y todos los hilos que estos generan, gracias a su estructura y los distintos niveles de memoria.

En concreto, CUDA *(ComputedUnifiedDevideArchitecture)*diseñado por la compañía NVidia, es una arquitectura de cálculo paralelo que aprovecha la GPU para proporcionar un incremento de rendimiento al sistema mediante un conjunto de herramientas que permite programar los distintos algoritmos y funciones dentro de las GPU, utilizando como base el lenguaje de programación C, aunque con algunas modificaciones. Para su óptima implementación se apoya en:

- La generación de conjuntos de hilos.
- El uso de memoria compartida.
- Sincronización de los hilos que se están ejecutando.

Puesto que se buscaba un soporte para computación de propósito general, las unidades aritmético lógicas fueron diseñadas según el estándar IEEE para las operaciones simples y en coma flotante. Así pudo ampliarse el uso de las GPUs a un abanico para el que no estaban inicialmente diseñadas.

Una GPU, en términos generales, de lo que se encarga es de dar un color a cada punto de una imagen y una de sus limitaciones inicialmente era la necesidad de tener que utilizar librerías como OpenGL para poder introducir el valor del punto. Debido a esto CUDA cobró tanta importancia por incluir el lenguaje de programación C con una serie de palabras reservadas.

Con ello, en el año 2006 Nvidia se convirtió en la primera empresa de desarrollo de tarjetas gráficas que lanzó un lenguaje específicamente diseñado para ellas con el modelo GeForce 8800GTX y se eliminaron las restricciones y barreras frente a los programadores, dado que CUDA también provee de los controladores necesarios para su correcto funcionamiento, evitando el uso de librerías ajenas.

Una GPU tiene la siguiente arquitectura hardware:

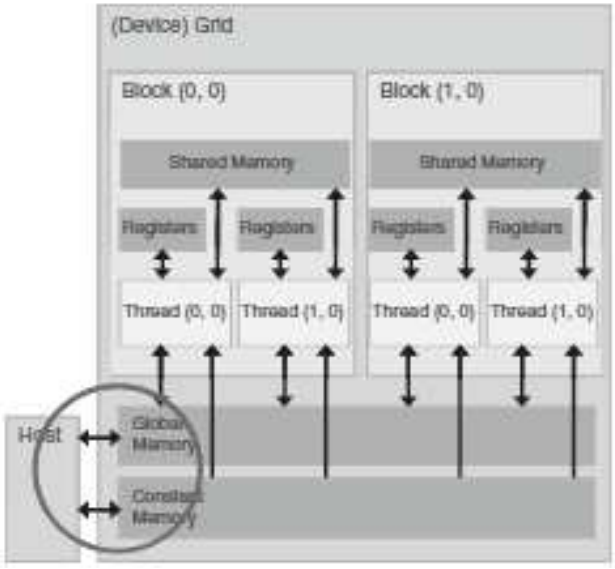

Figura 1: Arquitectura GPU

Siendo el *Host* la plataforma donde está alojada físicamente la GPU, tiene acceso directo a ella mediante las memorias global, texturas yconstantes. Además éstas son comunes para todos los multiprocesos mientras que la memoria compartida es privada para cada uno de ellos y el banco de registros para cada proceso en particular.

En sus inicios, la programación en GPU se realizaba con llamadas a servicios de interrupción de la BIOS y posteriormente mediante un lenguaje ensamblador específico a cada modelo, hasta que los avances permitieron la creación de APIs y así lograr un lenguaje homogéneo.

A continuación se describe en detalles los tipos de memoriaqueincorporanestosdispositivos:

- Banco de registros: memoria privada correspondiente a cada hilo a la que únicamente dicho hilo puede acceder.
- Memoria compartida: memoria con un alto ancho de banda utilizada de manera común por los distintos hilos pertenecientes a un mismo bloque.
- Memoria global: es la única memoria que permite tratamientos de lectura y escritura común a todos los bloques.
- Memoria de texturas: dado que el acceso a memoria global es aleatorio, se pueden insertar los datos en esta memoria, que además dispone de una caché y por tanto la hace más rápida que la memoria global.
- Memoria constante: puede cargarse dentro de la caché de cada multiproceso para acelerar las transferencias entre una y otra.

Dentro de la GPU, existe un espacio reservado a los hilos denominado *grid*. El grid se divide en bloques de hilos, estos bloques pueden ser de distintos tamaños y distintas dimensiones, pero alcanzando únicamente un máximo de 512 hilos por bloque actualmente.

En los bloques existe otra subdivisión denominada *warp*, conjuntos de 32 hilos, que es el tamaño mínimo de datos procesados en cada uno de los multiprocesos. Un *warp* es la unidad mínima de hilos que se planificará para su ejecución dentro de la GPU.

Cada uno de estos elementos se distingue por los parámetros *BlockIdx.α* o *ThreadIdx.α*siendo *α* uno de los valores [x, y, z] de acuerdo al eje de coordenadas correspondiente y haciéndolos unívocos. Además dentro de los bloques también se dispone del parámetro *BlockDim*, que refleja cuántas dimensiones tiene.

Las ejecuciones que realizan los hilos dentro de la GPU se llevan a cabo mediante *kernel*, un *kernel* es una función que será ejecutada por N hilos de manera simultánea en lugar de realizar una ejecución secuencial.

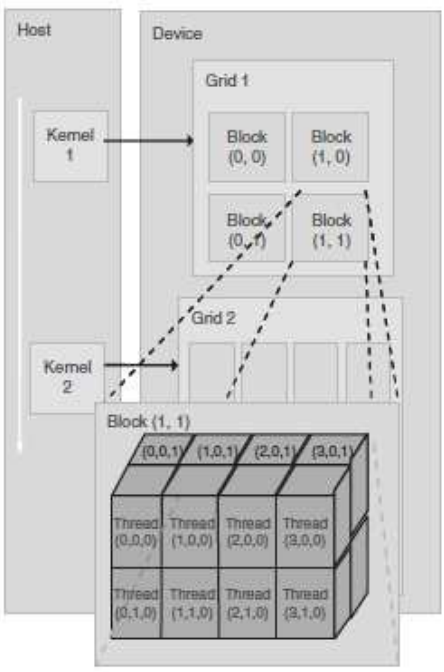

Figura 2: Grid GPU

Para definir un *kernel* la declaración de la función deberá ir precedida por *__global__*, en el momento de la invocación, deberán ser pasados tanto el tamaño del *grid* como el tamaño de los bloques, junto con los parámetros requeridos por la función en el caso de ser necesitados.

En ese momento los hilos comenzarán su ejecución paralela, pudiendo compartir datos entre ellos o necesitar posteriormente la información recogida durante las ejecuciones, por ello se necesitan

medidas de sincronización, garantizando que todos los hilos que han sido ejecutados han llegado a un determinado punto mediante el uso de *__syncthreads()*.

Utilizando esta herramienta se pueden agilizar en gran medida procesos que requieran muchos cálculos aritméticos, especialmente útiles en el tratamiento de imágenes, cobrando así gran interés para su utilización en la tomografía computarizada.

2.3 OpenCL

OpenCL*(Open Computing Language)*fue desarrollado inicialmente por Apple en su versión inicial que fue propuesta al grupo Khronos y posteriormente refinada con equipos de AMD, Intel, IBM y Nvidia. El grupo Khronos es un consorcio industrial financiado para la creación de APIs estándares y abiertas que permiten la creación y reproducción multimedia en múltiples plataformas y dispositivos.

Es una interfaz para escribir programas que se ejecutan sobre plataformas heterogéneas, como es el caso de CPU y GPU.Incluye un lenguaje de programación que se basa también en la utilización de *kernels* usando un código basado en C99 aunque con algunas limitaciones, pertenece a un estándar de software libre para potenciar el uso de las GPU.

Ha sido integrado dentro de los driver de las tarjetas gráficas tanto de AMD/ATI como de Nvidia, que lo expone como una alternativa a su lenguaje CUDA y soporta ejecuciones paralelas tanto homogéneas como híbridas.

OpenCL propicia el paralelismo tanto para los programas basados en la paralelización de tareas como aquellos basados en la paralelización de datos y está diseñado para trabajar con APIs de tratamiento gráfico como es OpenGL.

2.4 OpenGLcomo alternativa a CUDA

Open GL *(Open Graphics Library)*, desarrollado por SiliconGraphics Inc.,es una interfaz que dispone de diferentes funciones para el tratamiento de imágenes complejas a partir de otras más simples y primitivas. Está destinado para diseñar aplicaciones que producen elementos gráficos tanto en 2D como 3D, permitiendo una mayor explotación de los recursos ofrecidos por la tarjeta gráfica.

El estándar de Open GL es un documento donde se describen las funciones de las que dispone y cómo deben funcionar, de esta manera los fabricantes de hardware crean las bibliotecas ajustándose a los requisitos y crean la aceleración del hardware en base a ello. Estas implementaciones de librerías deben pasar unos test de conformidad para lograr su acreditación de adecuación al estándar Open GL.

La finalidad es ocultar el funcionamiento y la complejidad de los dispositivos hardware, englobándolos en una caja negra y mostrando al usuario una API uniforme y más intuitiva.

El funcionamiento consiste en aceptar formas geométricas simples como son puntos o triángulos y transformarlos en píxeles, el proceso se realiza mediante una pipeline gráfica denominada Máquina de estados de Open GL. Los comandos se encargan de emitir dichas primitivas a la pipeline gráfica o de configurar la pipeline en el modo del tratamiento de las primitivas.

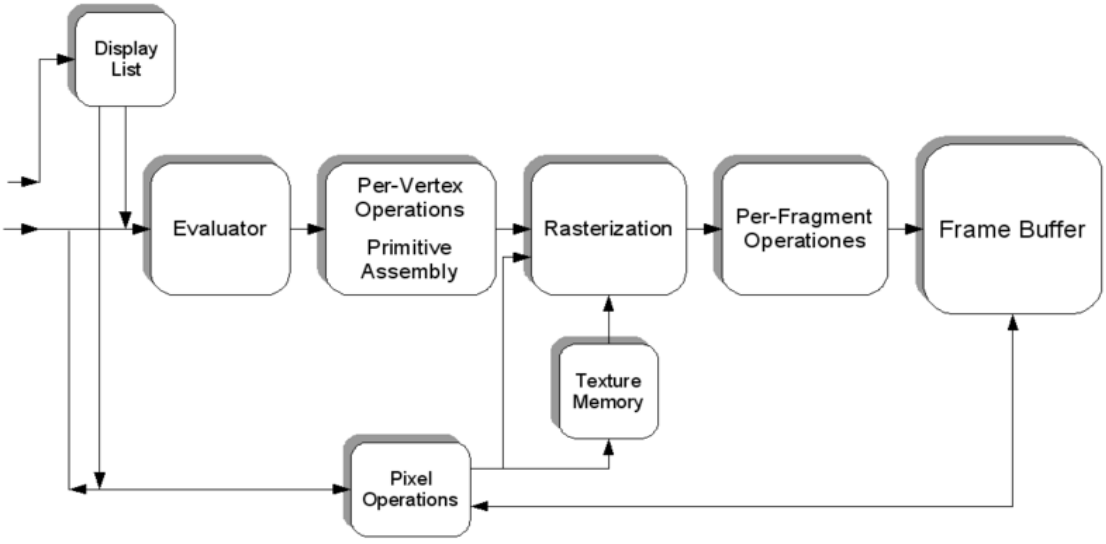

Figura 3: Pipeline del procesamientográficode OpenGL

Los procedimientos se realizan a bajo nivel, obligando al programador a especificar los pasos para *renderizar* la imagen, en contra de otros sistemas donde las bibliotecas se encargan de estos detalles, aunque debido a esto se logra una mayor libertad y versatilidad en la implementación.

En un funcionamiento básico, las diferentes etapas se encargan de:

- Etapa de evaluación: etapa opcional, a través de las funciones poligonales se aproximan las curvas y la geometría de la superficie mediante unos puntos estructurales.
- Operación por vértices y ensamblado de primitivas: sobre los vértices de la Figura se aplican iluminaciones de acuerdo al material y se recortan las partes no visibles generando un volumen de visión.

- Rasterización: convierte la información en píxeles y les aplica el color mediante algoritmos de interpolación.
- Operaciones por fragmentos: actualizaciones de valores y combinaciones entre ellos.
- Frame Buffer: una vez ha sido procesada la imagen entera, es volcada sobre este buffer.

2.5 Computación paralela y OpenMP

La computación paralela es una técnica por la que muchas instrucciones se ejecutan de manera simultánea, utilizando una filosofía de "divide y vencerás", se descompone el problema grande en tareas más pequeñas que pueden tratarse de manera paralela. Existen diversos tipos de paralelismo: paralelismo a nivel de bit, de instrucción o de tarea.

Inicialmente la computación paralela estaba ideada a la computación de altas prestaciones (HPC), pero debido a restricciones físicas en velocidad de procesador por calentamientos, costes y tamaños de los dispositivos, el paralelismo se ha convertido en el punto fuerte de las arquitecturas informáticas apostando por los procesadores con varios núcleos.

Aunque se logran amplias mejoras, las grandes barreras que representan las tareas paralelas se basan en los costes de comunicación y sincronización y es mediante la ley de Amdahl como se calcula el índice de mejora y el ratio de aceleración propiciado por el paralelismo.

El paralelismo desempeña un papel muy importante en este Proyecto Fin de Carrera y es la base de las múltiples optimizaciones que se aplican en Mangoose. Para este caso concreto se ha utilizado el estándar OpenMP.

OpenMP es un API soportado por diferentes plataformas que permite la programación en paralelo en arquitecturas basadas enmemoria compartida. Es un modelo escalable que aporta a los programadores una interfaz sencilla para el desarrollo de aplicaciones paralelas tanto para programas simples, que pueden ser realizados en un ordenador casero, como en complejidades altas que requieran la utilización de un supercomputador.

Soporta diferentes lenguajes de programación como C, C++ y Fortran, aunque para ese Proyecto Fin de Carreraúnicamente es necesaria la compatibilidad con C.

Contiene una serie de directivas para la compilación y variables de entorno que son las que marcan las directrices para la paralelización, afectando así el tiempo de ejecución.

Está basado en el modelo *fork-join*, es decir, dividir una tarea muy pesada en muchos hilos, creando tareas ligeras con un reducido tiempo de procesamiento para finalmente recoger todos los resultados y juntarlos en un único resultado final.

Para realizar una llamada a una sentencia de OpenMP en C debemos usar una cabecera como la siguiente:

```
# pragmaomp<directiva> [cláusula [ , ...] ...]
```

Utilizar una sentencia de OpenMP implica una sincronización en el bloque, un bloque que ha sido definido con ella y en el que se ejecutarán diversos hilos de acuerdo a las especificaciones de la directiva contendrá un una barrera al final para garantizar la sincronización de los mismos, exceptuando aquellas que definan explícitamente la condición *nowait*, situación en la que no se esperará por la terminación de todos los hilos simultáneos.

Entre las directivas que podemos asignar a una sentencia de OpenMP se encuentran:

- Parallel: indica la parte del código que puede ser ejecutada por varios hilos a la vez.
- For: análogo al funcionamiento de *parallel* pero optimizado para los bucles for, para la utilización de esta directiva se debe escribir *parallelfor*.
- Section: secciones que pueden ejecutarse en paralelo pero únicamente por un único hilo cada sección.
- Single: parte del código que sólo se ejecuta por un único hilo de todos los generados, aunque éste no tiene que ser necesariamente el hilo padre.
- Master: parte del código que únicamente puede ser ejecutada por el hilo padre.

Condiciones de sincronización de OpenMP:

- Critical: un único hilo puede ejecutar el código definido en esta sección de manera simultánea.
- Atomic: operaciones que deben ser realizadas en bloque como si de una única sentencia se tratara, emplea una única posición de memoria.
- Ordered: el bloque es ejecutado en el orden en que las iteraciones serían ejecutadas siguiendo el orden secuencial.
- Barrier: cada hilo espera hasta que el resto de hilos que se están ejecutando de manera paralela alcancen el mismo punto donde se encuentra él.
- Nowait: especifica la no necesidad de esperar por la terminación del resto de hilos, en caso de no especificarlo, por defecto se realiza una barrera al final de la construcción paralela.

- Flush: exporta a los hilos un valor modificado por otro hilo durante la ejecución del procesamiento paralelo.

Condiciones para compartir datos en las directivas OpenMP:

- Shared: la variable es compartida por todos los hilos
- Private: la variable tiene una copia privada para cada uno de los hilos
- Firstprivate: las copias de las variables privadas se inicializan con el valor original
- Lastprivate: al salir de la ejecución paralela, la variable contiene el valor que debería tener si se hubiera realizado una ejecución secuencial.
- Variables *reduction*: variables que se realizan sobre elementos de un array.

Condiciones para la planificación en las directivas OpenMP:

- Static: todas las iteraciones del *loop* se reparten equitativamente entre todos los hilos generados.
- Dynamic: no se reparten todas las iteraciones, asignando un número menor a cada uno de los hilos, una vez que uno de ellos ha finalizado su ejecución, toma alguna de las iteraciones pendientes de ejecución.
- Guided: un alto número de iteraciones se asigna a cada hilo de manera dinámica y dicho número va decreciendo exponencialmente a medida que se realiza cada asignación del citado número de iteraciones.
- Numthreads: especifica el número de hilos que serán generados para la construcción paralela.

FuncionesaportadasporOpenMP:

- Omp_set_num_threads: define el número de hilos que se ejecutan simultáneamente.
- Omp_get_num_threads: devuelve el número de hilos en ejecución.
- Omp_get_max_threads: devuelve el máximo de hilos lanzados en la zona paralela.
- Omp_get_thread_num: devuelve el número del hilo.
- Omp_get_num_procs: devuelve el número de procesadores del ordenador.
- Omp_set_dinamic: valor booleano para definir crecimiento o decrecimiento de los hilos de manera dinámica.

2.6 *Pinnedmemory* (memoria paginada vs. memoria no paginada)

En los sistemas operativos, durante la ejecución de un programa, la memoria que éste ocupa es dividida en partes, denominadas páginas. Igualmente la memoria está dividida en marcos de página, que actúa como índice de las distintas páginas que pueden encontrarse.

En un instante de tiempo la memoria puede estar ocupada con páginas de diferentes procesos en ejecución, con sus correspondientes marcos. El sistema operativo mantiene una lista de estos marcos, donde Figura en qué marco se encuentra cada página de un proceso. Así se evita el problema de la necesidad de memoria contigua y estática para un proceso, permitiendo un mayor aprovechamiento de la memoria al intercalarse con otros procesos.

Además cada proceso dispone de una Tabla de páginas conteniendo en ella las ubicaciones de los marcos que contienen cada una de sus páginas de memoria, con esta información se componen las direcciones lógicas, usando como guía la página del proceso y el desplazamiento en la misma, se halla el marco de página correspondiente con su dirección asociada y utilizando el desplazamiento se alcanza la dirección real.

Cuando un proceso se ejecuta y se carga en memoria, todas sus páginas son cargadas en los marcos libres restantes y se rellena la Tabla de páginas del proceso con los marcos que hayan sido ocupados durante dicha carga.

Sin embargo, aunque existen algoritmos de planificación para optimizar la gestión de la memoria, no siempre tiene la capacidad de almacenar todas las páginas en sus marcos, provocando los llamados fallos de página durante la ejecución de un programa.

En estas situaciones, la página debe ser cargada en memoria desde el disco, actualizando la lista de marcos y la Tabla de páginas, conllevando un retraso en el tiempo de ejecución del proceso.

Para solventar esto, los sistemas operativos disponen de espacios de memoria que se encuentran fijos, sin ser llevados nunca a un segundo plano. De esta forma, una página puede ser anclada durante un período de tiempo.

Fijar las páginas de memoria provoca el inconveniente de limitar los marcos de página disponibles para otros procesos u otras páginas requeridas durante la ejecución, pudiendo ocasionar más fallos de página de los que soluciona.

Por este motivo debe ser considerado el número de accesos a memoria, las páginas requeridas y las penalizaciones y aportaciones que aporta emplear el uso de memoria fija contigua.

2.7 AIO (Entrada/salidaasíncrona)

La entrada y salida asíncrona es un método de entrada y salida que permite a otros procesos continuar la ejecución mientras la petición realizada aún no ha sido terminada.

En algunos programas las operaciones de entrada y salida de manera secuencial pueden ocasionar el desperdicio de una cantidad de tiempo considerable, realizando un procesamiento de datos muy lento cuando, además, esta cantidad de datos posee un tamaño grande. Un mecanismo simple de entrada y salida secuencial permite un tratamiento seguro, ya que no requiere un coste de sincronización y verificación de que los datos han sido completamente leídos, de manera correcta, y además son los mismos que los necesitados.

Particularmente para este caso, en el que se procesa un alto número de proyecciones, se van leyendo mientras se van procesando, de manera análoga a su escritura.

Para utilizar las directrices asíncronas, se dispone de una librería específica <aio.h> y unas estructuras propias *aiocb*. Son requeridas dos de ellas para realizar el proceso:

```
Structaiocbmy_aio;

Structaiocb *my_aio_list[1]={&my_aio};
```

La primera de ellas, my_aio_list, es una lista de punteros a las estructuras my_aio, encargadas de guardar la información para realizar las llamadas asíncronas.

My_aioalmacena los campos necesarios para realizar los procesos de lectura y/o escritura:

Aio_fildes: Es el descriptor del fichero que es usado como productor de los datos a tratar.

Aio_buf: Localización del buffer.

Aio_nbytes: Número de bytes que serán transferidos desde el origen.

Aio_offset: Desplazamiento para indicar la posición inicial de la operación desde donde empezarán a ser contabilizados los bytes.

Aio_sigevent: Se trata de un campo opcional, que indica cómo debe tratarse la llamada cuando el proceso ha terminado, pueden darse tres situaciones:

- SIGEV_NONE: en cuyo caso no se envía ningún tipo de señalización.
- SIGEV_SIGNAL: al finalizar la ejecución se envía una señal que viene definida por su valor en sigev_signo.
- SIGEV_THREAD: provoca la ejecución de un hilo indicado mediante el campo sigev_notify_function.

Aio_lio_opcode: Dado que la utilización de elementos asíncronos permite varios tipos de funcionalidades, este campo se utiliza como indicador de la operación que debe ser realizada, siendo definida como:

- LIO_READ: En caso de ser operaciones de lectura, lee del fichero desde la posición definida por el offset los bytes indicados y los almacena en el lugar apuntado por el buffer.
- LIO_WRITE: Para operaciones de escritura, escribe el número de bytes indicado desde el buffer al fichero comenzando en la posición definida por el offset.
- LIO_NOP: No realiza ninguna operación, es utilizado cuando una ejecución desea ser ignorada, ejemplos de situaciones así son porque los valores no quieran ser manejados o bien porque existan huecos vacíos que deben ser omitidos.

Las llamadas que pueden ser realizadas mediante la cabecera de <aio.h> son:

```
int     aio_cancel(int, structaiocb *);
```

Cancela todas las llamadas de entrada/salida asíncrona para el descriptor del fichero definido, aunque las notificaciones especificadas por *sigevent* ocurren de manera normal.

La función devuelve AIO_CANCELED si todas las llamadas fueron canceladas correctamente y AIO_NOTCANCELED si alguna de ellas no fue cancelada porque ya estaba en progreso.

Existe también la posibilidad de recibir AIO_ALLDONE si todas las peticiones fueron realizadas antes de llamar a esta función.

```
Intaio_error(conststructaiocb *);
```

Devuelve el motivo del error de una petición de entrada o salida.

```
Intaio_fsync(int, structaiocb *);
```

Pone en cola una llamada de sincronización para las operaciones de entrada o salida en el descriptor del fichero.

```
intaio_suspend(conststructaiocb *const[], int,conststructtimespec *);
```

Detiene la ejecución hasta que al menos una de las peticiones asíncronas ha sido completada.

```
Intlio_listio(int, structaiocb *restrict const[restrict], int,
structsigevent *restrict);
```

Pone en la cola varias peticiones asíncronas utilizando una única llamada a la función.

3. DISEÑO DETALLADO DEL SISTEMA

La tomografía axial computarizada de rayos X(TAC) es uno de los procedimientos de diagnóstico y evaluación por imagen médica más utilizados. Las proyecciones son obtenidas a través del scanner eXplore PET/CT,diseñado por el Departamento de Ingeniería Biomédica de la UC3M/Hospital Gregorio Marañón y fabricado por SEDECAL. La Figura 4 y Figura 5muestran en detalle una ilustración del aparato y de cómo procede este para obtener proyecciones.

Figura 4: eXplore PET/CT

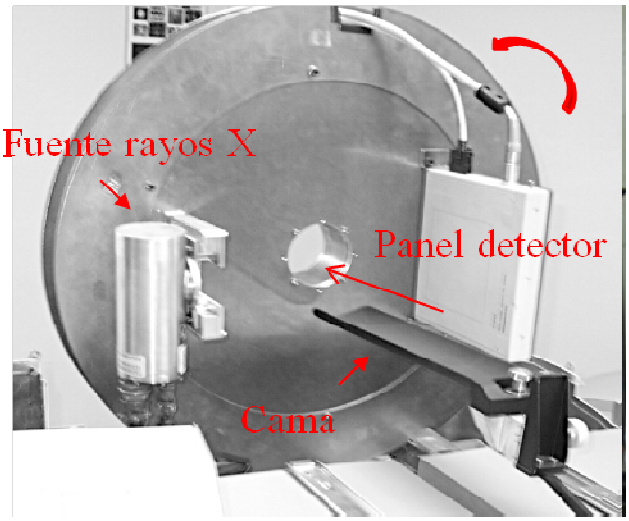

Figura 5: Esquema de componentes

Mangoose [2] es el nombre que recibe el reconstructor tomográfico que se desea ser optimizado. Se trata de una implementación multi-cama de un algoritmo de reconstrucción digital basado en el desarrollado por Feldkamp, Davis y Kress (FDK) para tomógrafos de rayos X de pequeño animal que utilizan la geometría basada en haz cónico.

El funcionamiento del sistema se muestra en la siguiente figura. Para facilitar los cálculos geométricos, se hace uso de las coordenadas *u,v,z* comunes tanto al detector virtual situado en el centro del FOV como al volumen reconstruido. El origen O está localizado en el centro del FOV, que se corresponde también con el punto central del detector virtual y el centro de rotación.

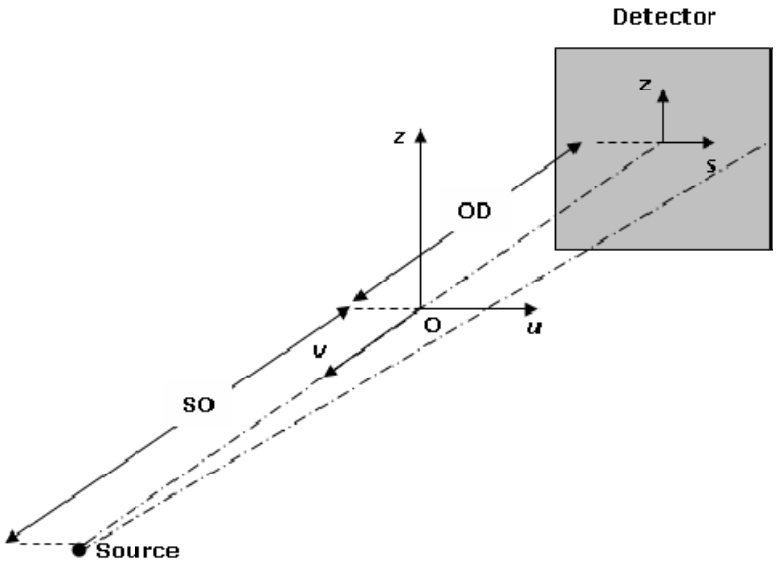

Figura 6: Perspectivade la arquitectura de haz cónico en CT. SO es la distancia entre el emisory el origen; OD es la distancia entre el detector y el origen

El algoritmo FDK parte de una serie de proyecciones obtenidas rotando circularmente la fuente y el detector alrededor del volumen de interés. A partir de las proyecciones, consta esencialmente de dos etapas:

Su función se divide en varias etapas, siendo las más costosas en computación y en las que se ha puesto especial dedicación: el filtrado de las proyecciones y su retroproyección sobre el volumen a reconstruir.

Para tomar proyecciones de un objeto que excede en dimensiones al campo de visión (FOV) de la máquina, se realiza la adquisición en varias secuencias avanzando la cama en la que se apoya el objeto en una dirección paralela al eje de rotación del conjunto fuente-detectores, a cada una de estas posiciones la llamamos “cama”; así, un sujeto habitual de estudio, una rata adulta, puede ocupar tres camas.

En laFigura 7se ilustra esta idea: en gris aparece la posición de la cama, entre la fuente y el panel de detectores, en la cual se adquiere; en azul aparece esa zona adquirida en la siguiente posición, que deja paso a la nueva zona que se desea adquirir.

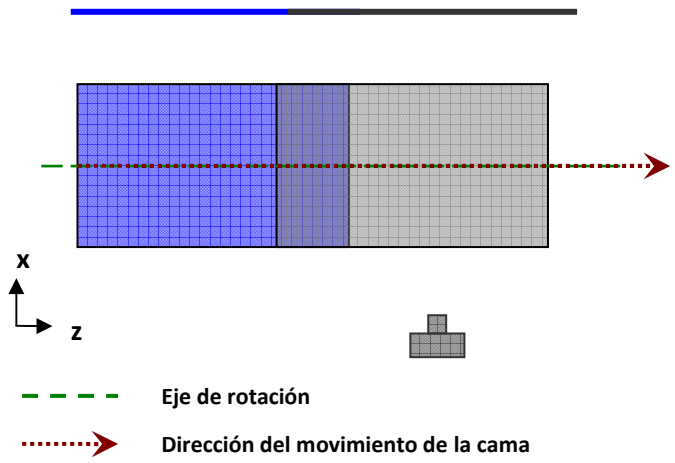

Figura 7: Vista coronal de dos posiciones de la cama, según el movimiento correcto de la misma, paralela al eje de rotación del anillo.

Idealmente las dos posiciones estarán alineadas y existirá una zona en la que se solapan. Tomando esto en consideración, al reconstruir se podrá alinear cada posición de la cama a la anterior para obtener una imagen del objeto completo.En el caso de que haya una desviación en la dirección que

sigue la cama al moverse, aparecerán errores que pasarán desapercibidos de adquirir y reconstruir una sola posición, pero que serán notables cuando el objeto requiere varias posiciones para ser adquirido por completo.

El reconstructor se compone de las siguientes fases:

- Etapa inicial: Período de tiempo destinado a la creación de contextos, reservas de memorias e inicialización de variables. Es un tiempo parcialmente fijo que puede ser reducido si existe un trabajo permanente que mantenga los dispositivos iniciados.

- Etapa de lectura: Los ficheros de origen con las proyecciones son leídos y almacenados en su correspondiente espacio de memoria reservada.

- Etapa de filtrado: Primera de las etapas de alto coste por su alto cómputo que consiste en aplicar el filtro de *Shepp-Logan* a cada línea de la proyección (lo que es propiamente el filtrado) así como en realizar un ajuste según las líneas se alejan del plano de rotación de la adquisición.El resultado es un conjunto de proyecciones con los nuevos valores.

- Etapa de retroproyección: Etapa más costosa del algoritmo y por tanto la principal candidata a ser acelerada. La retroproyección sobre todos los *vóxeles*del volumen a reconstruir se hace para cada una de las proyecciones filtradas en la etapa anterior y consiste en copiar cada valor de la proyección en los píxeles que contribuyeron a dicho valor, lo que implica interpolación. El resultado de esta etapa será un volumen reconstruido, final o parcial dependiendo del caso.

- Etapa de combinación de volúmenes: La generación del volumen reconstruido se hace mediante volúmenes parciales, cada uno a partir de un subconjunto de proyecciones, paralelizando el trabajo y reduciendo así el tiempo. En esta etapa los volúmenes parciales se combinan en un único volumen.

- Etapa de pegado de camas: Afecta únicamente cuando el objeto de la reconstrucción es mayor que el tamaño del campo de visión del equipo y debe ser adquiridoporpartes.En esta etapa, los volúmenes reconstruidos provenientes de cada adquisición (cama) se combinan en un único volumen final más grande. Para ello hay que tener en cuenta dos cosas. Por un lado, hay que corregir la desviación debida el movimiento no ideal de la cama, mencionado anteriormente, para que la unión de los volúmenes sea correcta. Por otro lado, hay que

gestionar la zona de solape, lo que se hace mediante unos pesos que definen que parte de cada cama tiene más contribución en la imagen final.

- Etapa de escritura: Una vez se han reconstruido los volúmenes de cada cama y se ha generado el volumen final, se escribe en el fichero designado.

En una visión general, podría considerarse la Figura 8como un diseño base del sistema propuesto en este Proyecto Fin de Carrera.

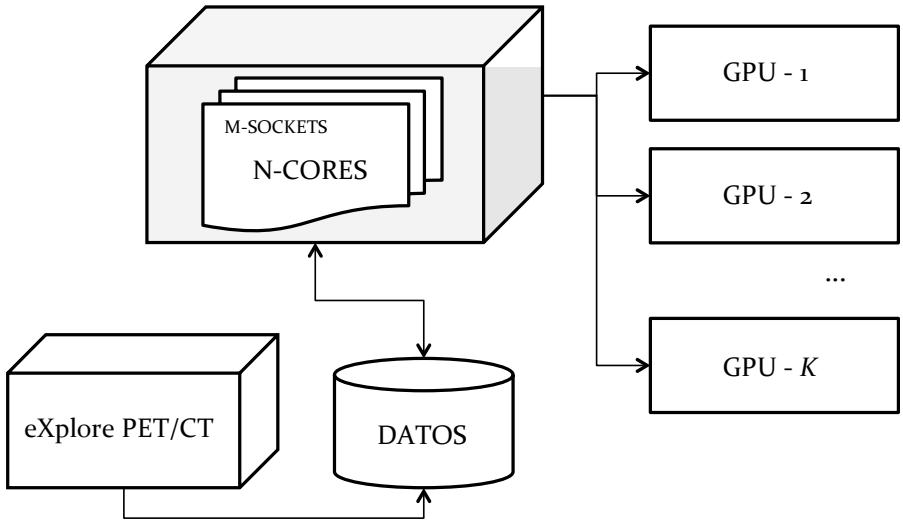

Figura 8: Implementación propuesta para Mangoose basada en múltiples GPU y multi-core

En el que se muestra como el equipo de captura obtiene el conjunto de proyecciones, las cuales son almacenadas en un soporte rígido. A continuación la aplicación Mangoose procesa la reconstrucción del volumen usando conjuntamente los dispositivos disponibles en el sistema, como son, un número variante de núcleos organizados en *sockets*, y las GPU instaladas en el sistema. Finalmente el volumen procesado es almacenado en el disco duro, listo para si visualización en la aplicación de control, implementada en IDL.

3.1 Análisis de requisitos

Para el correcto funcionamiento de Mangoose y para permitir ser optimizado eficazmente, deben existir una serie de características indispensables que propicien el buen funcionamiento de las nuevas implementaciones, denominados requisitos.

La Tabla explicativa se compone de diversos campos:

- ID Requisito: identificador unívoco para diferenciar unos de otros.
- Tipo: funcional o de usabilidad, según corresponda de acuerdo a su cometido.
- Evento: situación que debe ocurrir como disparador.
- Descripción: breve definición del cometido.
- Justificación: por qué es importante el requisito y qué motivación lo ha propiciado.
- Criterio de cumplimiento: comprobante de una implementación del requisito satisfactoria.
- Prioridad: Relevancia del requisito.
- Conflictos: requisitos que no pueden ser implementados si ése requisito es implementado.

Los requisitos que deben cumplirse son los siguientes:

ID Requisito	RNF-01	**Tipo:**	No Funcional
Evento / Caso de Uso	Ejecución de Mangoose		
Descripción	Reconstructor genérico independiente el SO.		
Justificación	Debe funcionar tanto en Windows como en Linux		
Criterio de cumplimiento	Lanzar una ejecución en diferentes arquitecturas y obtener el resultado esperado		
Prioridad	Alta	**Conflictos**	#

Tabla 2: Requisitos– 1

ID Requisito	RNF-02	**Tipo:**	No Funcional
Evento / Caso de Uso	Arquitectura heterogénea con dispositivos paralelos diferentes		
Descripción	Utilización de diferentes GPUs.		
Justificación	Permitir la ejecución con CUDA utilizando todos los recursos disponibles aunque no sean homogéneos.		
Criterio de cumplimiento	Lanzar una ejecución con GPUs heterogéneas y obtener un resultado satisfactorio.		
Prioridad	Alta	**Conflictos**	#

Tabla 3: Requisitos –2

ID Requisito	RNF-03	**Tipo:**	No Funcional
Evento / Caso de Uso	Múltiple GPU		
Descripción	Uso paralelo de dispositivos GPU.		
Justificación	Debe tolerar un número variable deGPUs y funcionar de forma sincronizada en paralelo		

Criterio de cumplimiento	Lanzar una ejecución con múltiples dispositivos y obtener el volumen adecuado.		
Prioridad	Alta	**Conflictos**	#

Tabla 4: Requisitos – 3

ID Requisito	RNF-04	**Tipo:**	No Funcional
Evento / Caso de Uso	Múltiples hilos en CPU		
Descripción	Paralelismo con hilos		
Justificación	Permitir la ejecución paralela en su versión CPU en un entorno multi-core.		
Criterio de cumplimiento	Lanzar una ejecución con multitud de hilos y obtener el resultado esperado		
Prioridad	Alta	**Conflictos**	#

Tabla 5: Requisitos – 4

ID Requisito	RF-01	**Tipo:**	Funcional
Evento	Ejecución de Mangoose		
Descripción	Mangoose reconstruye un volumen		
Justificación	El reconstructor debe generar los volúmenes a partir de las proyecciones del objeto proporcionadas.		
Criterio de cumplimiento	Lanzar una ejecución y obtener el resultado esperado		
Prioridad	Alta	**Conflictos**	#

Tabla 6: Requisitos –5

ID Requisito	RF-02	**Tipo:**	Funcional
Evento / Caso de Uso	Existencia de dispositivos GPU		
Descripción	Mangoose se ejecuta en su versión GPU.		
Justificación	Utilización del API de CUDA, dada la arquitectura sobre la que se ejecutará la aplicación..		
Criterio de cumplimiento	Fijar tantos hilos como dispositivos haya y mostrar cuáles son por pantalla.		
Prioridad	Alta	**Conflictos**	#

Tabla 7: Requisitos–6

ID Requisito	RF-03	Tipo:	Funcional
Evento / Caso de Uso	Reconstrucción sin dispositivo GPU		
Descripción	Mangoose se ejecuta en su versión CPU		
Justificación	Utilización alternativa.		
Criterio de cumplimiento	Lanzar una ejecución sin dispositivos GPU		
Prioridad	Alta	Conflictos	#

Tabla 8: Requisitos – 7

ID Requisito	RF-04	Tipo:	Funcional
Evento / Caso de Uso	Ejecución de Mangoose usando OpenMP		
Descripción	El sistema debe ser compatible con el standardOpenMP.		
Justificación	Aporta un alto grado de optimización por el uso del paralelismo anidado.		
Criterio de cumplimiento	Lanzar una ejecución con la sentencia paralela y que funcione correctamente.		
Prioridad	Alta	Conflictos	#

Tabla 9: Requisitos– 8

ID Requisito	RF-05	Tipo:	Funcional
Evento / Caso de Uso	Ejecución de Mangoose usando AIO		
Descripción	El sistema debe ser compatible con las llamadas asíncronas de Entrada/Salida.		
Justificación	Aporta un alto grado de optimización permitiendo lecturas y escrituras asíncronas.		
Criterio de cumplimiento	Lanzar una ejecución con llamadas asíncronas y que funcione correctamente.		
Prioridad	Alta	Conflictos	#

Tabla 10: Requisitos– 9

ID Requisito	RF-6	Tipo:	Funcional
Evento / Caso de Uso	Ejecución asimétrica.		

Descripción	Las reconstrucciones siempre deberán hacerse con la implementación asimétrica.		
Justificación	Utilizar la simetría conlleva riesgos de falseamiento en los datos.		
Criterio de cumplimiento	Lanzar una ejecución y obtener el resultado esperado		
Prioridad	Alta	**Conflictos**	#

Tabla 11: Requisitos – 10

ID Requisito	RF-7	**Tipo:**	Funcional
Evento / Caso de Uso	Cálculo en flotantes/enteros.		
Descripción	Las reconstrucciones pueden hacerse utilizando números enteros o en coma flotante.		
Justificación	Niveles de precisión.		
Criterio de cumplimiento	Lanzar una ejecución y obtener el resultado esperado		
Prioridad	Alta	**Conflictos**	#

Tabla 12: Requisitos – 11

ID Requisito	RF-8	**Tipo:**	Funcional
Evento / Caso de Uso	Utilización de GPU		
Descripción	El sistema debe ser compatible con la utilización de CUDA		
Justificación	Usar la versión diseñada y destinada para dicha API.		
Criterio de cumplimiento	Lanzar una ejecución y obtener el resultado esperado		
Prioridad	Alta	**Conflictos**	#

Tabla 13: Requisitos–12

ID Requisito	RF-9	**Tipo:**	Funcional
Evento / Caso de Uso	Reconstrucción multi-cama.		
Descripción	Solape con cama múltiple.		
Justificación	En las reconstrucciones con más de una cama se produce un solape de proyecciones comunes eligiendo las idóneas mediante la aplicación de pesos.		
Criterio de	Reconstrucción correcta.		

cumplimiento			
Prioridad	Alta	**Conflictos**	#

Tabla 14: Requisitos – 13

ID Requisito	RF-10	**Tipo:**	Funcional
Evento / Caso de Uso	Reconstrucción multi-cama.		
Descripción	Cálculo de la desviación.		
Justificación	Corregir el índice de la desviación estructural existente en el reconstructor físico.		
Criterio de cumplimiento	Reconstrucción correcta.		
Prioridad	Alta	**Conflictos**	#

Tabla 15: Requisitos– 14

ID Requisito	RF-11	**Tipo:**	Funcional
Evento / Caso de Uso	Ejecución de Mangoose		
Descripción	Parámetros por argumento		
Justificación	Los parámetros serán pasados por argumento y deberán ser analizados sintácticamente de manera correcta..		
Criterio de cumplimiento	Reconstrucción correcta.		
Prioridad	Alta	**Conflictos**	#

Tabla 16: Requisitos - 15

ID Requisito	RF-12	**Tipo:**	Funcional
Evento / Caso de Uso	Ejecución de Mangoose		
Descripción	Reconstrucción independiente del volumen.		
Justificación	La reconstrucciones digitales podrán ser parciales en cualquier volumen, especificado el tamaño del mismo como argumento.		
Criterio de cumplimiento	Reconstrucción correcta con un tamaño anómalo.		
Prioridad	Alta	**Conflictos**	#

Tabla 17: Requisitos - 16

ID Requisito	RF-13	Tipo:	Funcional
Evento / Caso de Uso	Ejecución de Mangoose usando prefetch.		
Descripción	La reconstrucción utiliza lectura adelantada.		
Justificación	La lectura adelantada de las proyecciones produce un mayor rendimiento y menos detenciones esperando datos para procesamiento.		
Criterio de cumplimiento	Reconstrucción correcta utilizando esta técnica.		
Prioridad	Alta	Conflictos	#

Tabla 18: Requisitos - 17

ID Requisito	RU-01	Tipo:	Usabilidad
Evento / Caso de Uso	Ejecución de Mangoose		
Descripción	Tabla resumen de los argumentos y los parámetros aplicados en la reconstrucción.		
Justificación	Mostrar un resumen de los índices utilizados en la reconstrucción como comprobante de los valores de entrada introducidos.		
Criterio de cumplimiento	Mostrar los parámetros por pantalla al iniciar la reconstrucción.		
Prioridad	Alta	Conflictos	#

Tabla 19: Requisitos - 18

ID Requisito	RU-02	Tipo:	Usabilidad
Evento / Caso de Uso	Tiempo de reconstrucción.		
Descripción	Mostrará el tiempo de proceso invertido en cada etapa.		
Justificación	Dividir el tiempo total por etapas para observar el grado de mejora y aquellas fases donde el coste debe ser reducido.		
Criterio de cumplimiento	Tiempos mostrados correctamente al ejecutar Mangoose.		
Prioridad	Alta	Conflictos	#

Tabla 20: Requisitos– 19

ID Requisito	RU-03	**Tipo:**	Usabilidad
Evento / Caso de Uso	Etapa en curso		
Descripción	Mostrará por pantalla y en log los avances en ciertas etapas durante el proceso de reconstrucción.		
Justificación	Que el usuario sepa que el proceso está en curso y progresa adecuadamente..		
Criterio de cumplimiento	Que indique los avances durante la ejecución de Mangoose.		
Prioridad	Alta	**Conflictos**	#

Tabla 21: Requisitos - 20

4. CATÁLOGO DE OPTIMIZACIONES

A continuación pasan a describirse las optimizaciones llevadas a cabo sobre la implementación del algoritmo FDK, Mangosse, en su versión inicial. En este capítulo veremos que optimizaciones han permitido obtener una mejora sustancial en laaceleración del proceso.

4.1 Puesta a puntoinicial

En las siguientes optimizaciones, lo que se busca es facilitar y dejar preparado el código inicial para aportar facilidad en el manejo y en la comprensión, dejándolo listo para proceder a sus modificaciones para la optimización de procesamiento

4.1.1 Primera fase: división de arquitecturas

A partir de la versión inicial, encontrándose en modo secuencial y con redundancia en el código por encontrarse replicado un gran número de veces, se realizó una limpieza del mismo para una mayor simplicidad en su lectura, manipulación y mantenimiento.

El código replicado es eliminado, se encontraba dividido en función de si las proyecciones eran simétricas o asimétricas, dejando únicamente el código correspondiente al asimétrico que permite cualquier caso de estudio en ejecución. Además de la eliminación de una de las grandes funciones de Mangoose, se elimina código inservible y no utilizado en las ejecuciones.

Por último, este la implementación del algoritmo FDK se divide en dosversiones:

- FDK_CPU: código inicial como referente tanto para el modelo que debe seguir la reconstrucción gráfica tanto para la necesidad de generar una copia de respaldo,propiciada por un cambio erróneo.

- FDK_CUDA: versión dispuesta para ser lanzada con los *kernel* de CUDA en una arquitectura compuesta por dispositivos GPU y multi-core.

Con esta división se alcanza un nivel de manejabilidad sostenible y comienzan las optimizaciones que para mejorar el ratio de aceleración.

4.1.2 Segunda fase: estructura del código fuente..

A partir de las versiones creadas, el código se divide en funciones. La motivación para dividir el código en segmentos separados es la manejabilidad del mismo, pudiendo aislar y focalizar más fácilmente las complicaciones que puedan surgir.

Además, aportar un grado de modularización entre las diferentes secciones que componen el reconstructor, permite un mayor nivel de abstracción a la hora de resolver las interacciones entre unas etapas y otras, propiciando la generación del grado de paralelismo óptimo y facilitando también la adaptación a las distintas tecnologías.

Para mantener la cohesión y el encapsulamiento de las funciones, las funciones comunes a las dos arquitecturas fueron extraídas y generadas en un nuevo fichero, *Common,* entre otras, las más llamativas que pueden ser encontradas en él son las siguientes funciones:

- Corrección de desviación.
- Pegado de camas.
- Pegado de camas múltiple.
- Hounsfield.

La Figura 9muestra el diagrama de flujo de los procesos de la aplicación indicando los componentes comunes y específicos del diseño modular de la solución.

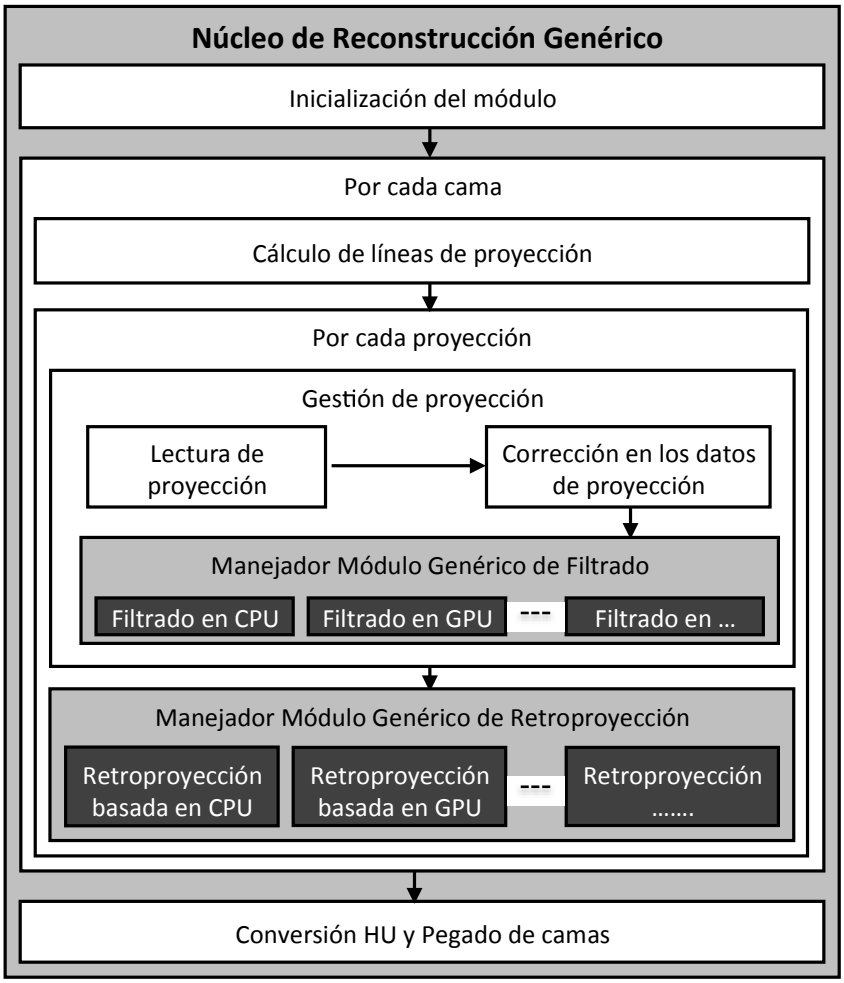

Figura 9: Núcleoreconstrucción de Mangoose Plus

4.1.3 Tercera fase: definición del número de hilos

Para determinar un número manejable de hilos y facilitar la focalización de un error que pudiera surgir, se fijan tantos hilos como dispositivos GPU estén disponibles para la ejecución del

proceso.De esta manera se reparten las cargas equitativamente y se permite un único hilo en cada vía, pudiendo comprobar el buen funcionamiento y sincronización de las secciones paralelas.

4.2 Entrada / salida asíncrona

En su etapa inicial, tanto la lectura como la escritura se realizaban de manera secuencial dentro un único hilo, otorgando una linealidad y un retraso en el tiempo de ejecución que se acrecentaba conforme al número de proyecciones que debían ser leídas y las dimensiones que éstas poseían.

Para un volumen en el que las proyecciones tienen un tamaño de 512x512 puntos, el tiempo de lectura se sitúa inicialmente en 8 segundos. Sin embargo,para proyecciones de 2048 x 2048 puntos la solución inicial puede llegar a consumir hasta1480 segundos aproximadamente. Esta es la principal motivación para solapar en la medida de os posible, el tiempo invertido en accesos de entrada y salida

Para solventar este problema, reduciendo el tiempo invertido,se utilizan las estructuras *AIO* mencionadas anteriormente para proveer a la solución acceso asíncrono a los ficheros. Disponiendo de tantos hilos como dispositivos disponibles, se utiliza la función *aio_read* para realizar una lectura asíncrona de los ficheros de proyecciones, junto con un *aio_suspend* que mantiene la ejecución en espera hasta que, al menos, una proyección ha sido leída completamente.

Tras disponer de al menos una proyección, ésta es procesada con sus correspondientes cálculos que permiten la generación del volumen mientras que la lectura se sigue realizando con las proyecciones restantes en segundo plano.

Durante la lectura asíncrona, se genera una pausa en la ejecución del proyecto mediante la sentencia *aio_suspend*, esto provoca la paralización hasta que al menos una de las proyecciones ha sido leída completamente, momento en el que es transferida al proceso de filtrado. Para evitar que esta parada se ejecutara repetidamente a la espera de una proyección, se realiza una lectura adelantada de varias proyecciones, De esta manera, la detención no existe y deja un margen de proyecciones para asegurar que siempre habrá al menos una lista para ser procesadasmientras se prosigue con la lectura de las proyecciones restantes de forma asíncrona. Uno de los parámetros que podrán se definidos será el número de proyecciones que deseamos leer de forma adelantada.

4.3 Localidad en memoria

En la versión inicial de Mangoose los cálculos eran una traducción de los realizados con una aplicación específica para cálculo numérico como es Matlab. Sin embargo, en el tratamiento de matrices Matlab realiza los cálculos recorriéndolas por columnas en lugar de por filas.Hacerlo de ésta manera provoca continuos fallos en la caché del sistema, aumentando en gran medida el tiempo de computación necesario en las operaciones con matrices.

Para solventarlo, el código fue editado rotando el volumen y modificando el eje por donde debían iniciarse las operaciones sobre las matrices.Así se logra un alto grado de localidad en la memoria, puesto que las matrices pasan a recorrerse por filas en lugar de por columnas, minimizando el número de fallos en la caché y reduciendo los costosos accesos a memoria principal. En la etapa de escritura la rotación vuelve a invertirse para dejar el volumen reconstruido en su posición inicial.

El concepto de caja negra se mantiene ya que el usuario obtiene la misma salida a partir de la misma entrada que en su versión inicial. Sin embargo, el tiempo total se ve reducido, en mayor o menor medida dependiendo de las prestaciones hardware por el coste que genere un fallo en memoria caché.

4.4 Uso de memoria compartida en GPU

En las primeras versiones de las optimizaciones de Mangoose usando el API de CUDA, se utilizaba la memoria global para gestiones de lectura y escritura por cualquier bloque De esta manera los bloques accedían y tomaban las proyecciones, no era óptimo por el coste de latencia que implica el acceso constante a memoria global de la GPU, siendo ésta la más lenta.

Por ello se modificó para explotar los recursos mediante la memoria compartida. Por definición la memoria compartida es una región de memoria que puede ser accedida simultáneamente por múltiples procesos mejorando la comunicación y la replicación de información innecesaria.

Las GPUS disponen de una región específicamente diseñada para realizar esta función compartiendo la información entre todos los hilos de un mismo bloque a una velocidad muy superior frente a la alcanzada mediante el repetido acceso a memoria global, entre 100~150 veces más rápida.

Utilizando una estructura denominada *TILE* se puede crear una submatriz de *Tamaño_Bloque x Tamaño_Bloque*:

```
__shared__ float tile[BLOCK_SIZE_BP][BLOCK_SIZE_BP];
```

Trabajando con estas matrices en memoria compartida, cada bloque de hilos realiza un único acceso de lectura y otro de escritura a memoria global, mientras que las gestiones intermedias se realizan sobre el *TILE*.

4.5 Paralelización basada en OpenMP

Durante el desarrollo de toda la aplicación, se realizan muchas iteraciones definidas por las instrucciones de control *for*, como es en procesos de lectura, filtrado o retroproyección.Para alcanzar un mayor nivel de paralelismo, en los bucles de mayor nivel se han dividido mediante la sentencia:

```
#pragmaompparallelfor
```

El uso más representativo de la utilización del API se produce en el momento de dividir las proyecciones por fichero entre el número de hilos definido, la cabecera empleada es:

```
#pragma omp parallel for private(result,rad_angle)  firstprivate(slots)
     shared(aux_projection,projections) num_threads(countDevices)
     schedule(dynamic, SLOT)
```

Aspectosdestacables:

- **Private(result,rad_angle):** Hay dos parámetros que necesitan tener una copia privada para que no se pueda producir un solape y un hilo de la ejecución pueda modificar su valor, generando un error en el procesamiento de otro hilo.Estos parámetros son *result* y *rad_angle*.

  Result: Es el encargado de almacenar el éxito o fracaso del filtrado, dejando una copia para cada uno de los hilos que permita activarse en caso de que uno de ellos no haya tenido éxito.

  Rad_angle: Guarda el ángulo en radianes de la proyección que va a ser gestionada, debe ser privado para evitar que varias proyecciones accedieran sobre el mismo ángulo, reescribiendo la información y deformando el volumen generado.

- **Firstprivate(slots) :** Representa el número de bloque de proyecciones, el primero debe ser privado para que cada división acceda al bloque inicial.

- **Shared(aux_projection,projections):** Las estructuras donde se almacenan las proyecciones se comparten para que todos los hilos puedan tener acceso a ellas sin crear cuellos de botella ni bloqueos.

- **Num_threads(countDevices):** Se definen los hilos dependiendo del valor almacenado en countDevices, definido con anterioridad en el código siendo éste el número de dispositivos disponibles.

- **Schedule(dynamic, SLOT):** Establece el tipo de planificador en el primer parámetro estatic/dynamic y en SLOT el tamaño de la porción que se le asigna a cada hilo.

4.6 Hilos según tamaño del bloque

Como ya se mencionó al inicio de esta memoria, la estructura de CUDA contiene un elemento denominado *grid*, divido en bloques de hilos, una buena gestión en el tamaño favorece la ocupación dentro de la GPU, permitiendo una mayor optimización de los recursos que éstas aportan.

Para encontrar el reparto más óptimo, se realizó una comparativa del rendimiento del tamaño del bloque en función del número de hilos.

Contrastando los resultados obtenidos usados en 4 casos de estudio diferentes, con unos valores de bloque comprendidos en {4, 8, 16, 32} y los hilos en {1, 2, 4, 8, 16} se obtuvieron los tiempos para los diferentes rendimientos(ver APENDICE II).

En las diferentes Tablas, queda resaltado el tiempo más óptimo para cada uno de los casos de estudio así como el peor de los tiempos posibles, con estos resultados es posible generar un configurador que fije el valor más óptimo para cada caso de estudio.

Además, se aprecia la gran diferencia de tiempo generada entre ambos casos, apoyando la hipótesis de la variedad en el tamaño de bloque y número de hilos conforme al tamaño de las proyecciones para la ocupación.

4.7 Implementación modular sobre GPU

A la hora de diseñar e implementar un sistemas modular, se ha separados aquellas partes comunes para los distintos sistemas, de aquellas específicas para cada dispositivo de cómputo. Los módulos especializados serán aquellos que consumen más tiempo de procesamiento, como se indicó anteriormente, el filtrado y la retroproyección. En este apartado se describen los detalles para las implementaciones para multi-core y multi-GPU, aunque existe la posibilidad extender Mangoose para otros tipos de paradigmas y tecnologías.

Los módulos destinados a multi-core se basan en una paralelización anidada gracias al paradigma de programación soportado por OpenMP. La implementación propuesta se basa en dos niveles de paralelismo: paralelismo de grano grueso dedicado a distribuir distintas proyecciones sobre los núcleos del sistema, y paralelización de grano fino destinado específicamente a los módulos de filtrado y retroproyección.

A continuación se describen los detalles de implementación para los módulos destinados para sistemas que cuentan de una a varias GPUS. La etapa de filtrado se sustenta en la librería CUFFT, la librería de transformadas rápidas de Fourier (FFT) que se suministra de forma gratuita junto con las herramientas de desarrollo de CUDA, en su versión 4.0. Dado que tanto los datos de partida como el resultado del filtrado son números reales, se usan la transformada directa de reales a complejos y la inversa de complejos a reales. Es habitual que esta implementación permita mayor rendimiento en velocidad y uso de memoria que el caso más general de transformar de complejos a complejos. El filtrado se aplica individualmente sobre cada proyección antes de llamar a la retroproyección.

En lo que respecta a la retroproyección, se utiliza un esquema centrado en el vóxel[22]. Las proyecciones se tratan individualmente y se cargan en la memoria de textura según la capacidad de la GPU.

En la siguiente Figura se enumeran los componentes principales del núcleo de reconstrucción basado en GPU para uno o varios dispositivos.

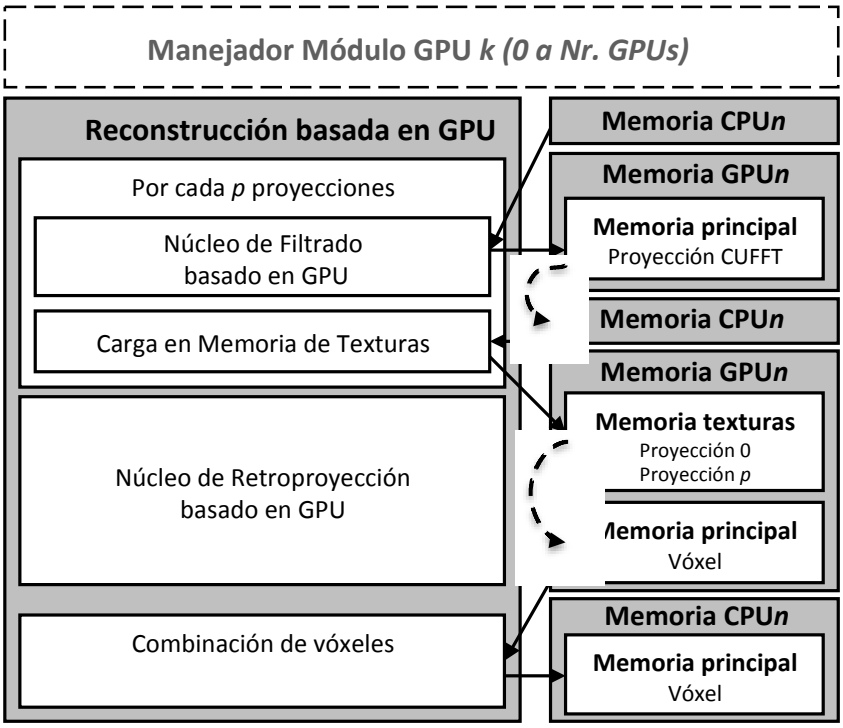

Figura 10: Manejador genérico modular para sistemasmulti-GPU

El volumen entero queda cargado en la memoria del dispositivo de forma que se va actualizando con cada k las sucesivas llamadas a la función de retroproyección (donde k es el número de proyecciones que se tratan a la vez y es configurable automáticamente dependiendo de la memoria de textura disponible). Se almacena en memoria como una sucesión de valores de las dimensiones *X*, *Y* yZ. Para sacar partido de la localidad en memoria, y así explotar al máximo su ancho de banda, la lectura para el cálculo de la retroproyección de cada *vóxel* se hace secuencialmente en ese mismo orden.

Para tamaños arbitrarios no es válido suponer que todo el volumen se puede cargar en la memoria de la GPU. El algoritmo está diseñado para trabajar con tamaños arbitrarios, fragmentando las proyecciones o el volumen según la memoria disponible en la GPU.

5. VERIFICACIÓN DEL SISTEMA

Para la resolución del catálogo de optimizaciones y analizar la implementación de las optimizaciones, se dispusieron de las siguientes herramientas, para satisfacer las necesidades requeridas para la correcta resolución del problema:

- **ImageJ:** Es una aplicación de procesamiento de imagen digital con licencia gratuita, implementado en Java y desarrollado por el *NationalInstitutes of Health*.Da un gran soporte como visor de imágenes por sus altas prestaciones, funcionalidades y la gran cantidad de formatos de imagen soportados, especialmente *RAW* (imágenes en crudo sin tratamiento digital).

  En concreto se ha utilizado este soporte porque permite voltear el volumen, pudiendo analizarlo desde varias perspectivas como se muestra en las siguientes figuras.

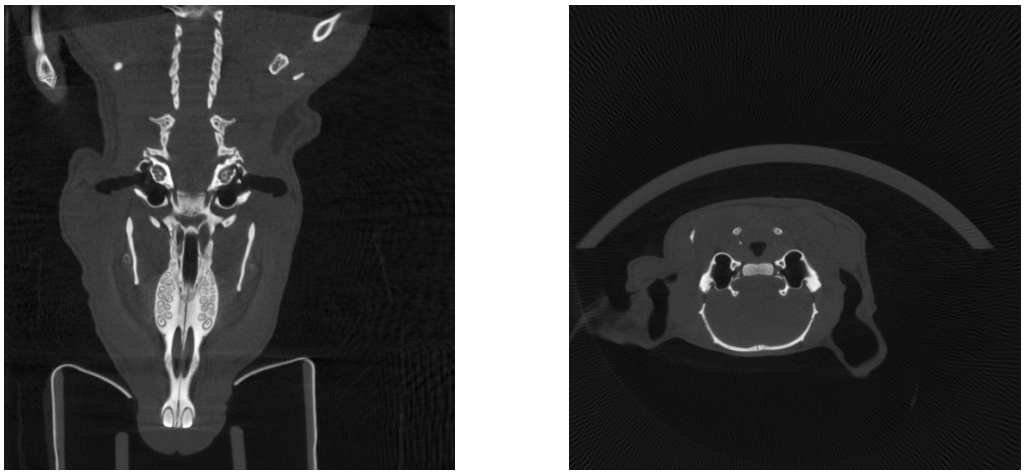

Figura 11: Perspectiva ImageJ

Junto con la parte visual, donde inicialmente se aprecia la Figura aparentemente correcta. En la Figura 11 anteriores puede apreciarse el caso de estudio de una rata donde, tomando la misma imagen, se muestra la reconstrucción vista desde el frente y desde arriba.

Además otorga la posibilidad de mostrar el valor que tiene un punto, marcando sus coordenadas X e Y, como se aprecia en la Figura 12.

x=149, y=88, value=0.01027734

Figura 12: Datos punto basados en coordenadas en ImageJ

Para comprobar la buena reconstrucción del volumen, una de las pruebas realizadas es la comparación del valor sobre distintos puntos elegidos aleatoriamente entre la versión inicial y la nueva compilada.

Otra funcionalidad por la que ImageJ resultaba una aplicación necesaria, era por su opción de restar dos imágenes. Esto consiste en restar los valores directos de cada punto, de esa manera si los puntos poseen el mismo valor o un valor muy próximo, el resultado es cercano a 0, por lo que el píxel toma el color negro. Así, si en la resta de imágenes se obtenía como resultado un lienzo negro, era indicador de una buena solución.

Con el amplio abanico de formatos soportados, permite comparar imágenes en 16 bit y 32 bit, pudiendo analizar resultados intermedios que restaran las imágenes generadas con números en coma flotante y tras la etapa del filtro. Dividiendo por etapas la reconstrucción gráfica para aislar posibles problemas.

- **CUDA Profiler:** Con el API de CUDA, en el *SDK* se incluye un *profiler* que monitoriza los dispositivos GPU, con el rendimiento, el uso, las ejecuciones de cada núcleo o los fallos en memoria.

  Pero el uso más importante es el soporte gráfico que muestra la utilización de las GPUs en paralelo, con los costes de tiempo en cada fase y sincronizaciones, Figura 13.

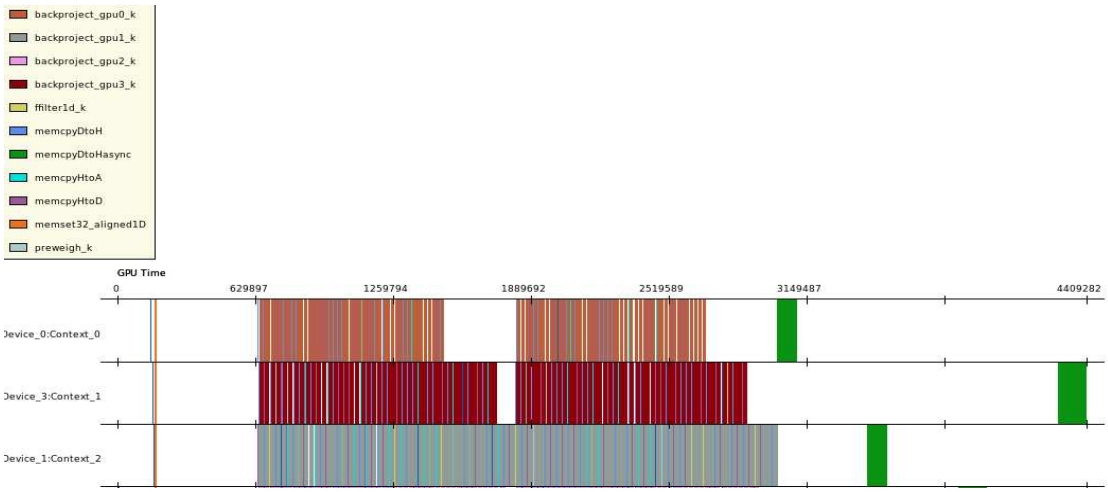

Figura 13: CUDA Profiler

Como se aprecia en la Figura 13, quedan referenciados y etiquetados los distintos dispositivos, pudiendo analizar y comprar a simple vista el impacto de las modificaciones en el rendimiento en las GPUs

Fue especialmente útil, en la decisión del planificador y el uso de elementos asíncronos, junto con el reparto de trabajo e interacciones en arquitecturas heterogéneas.

- **Ztimer:** es una librería de Linux utilizada para cronometrar las etapas y observar los ratios de aceleración en lugar de utilizar cronómetros manuales. Pueden ser insertados en cualquier parte del código, con sentencias para iniciar y parar, mostrando por pantalla el tiempo invertido en cada una de las etapas.Esos mismos temporizadores eran traspasados y escritos en un fichero para ser importados posteriormente y realizar las comparativas entre los distintos casos de estudio.
- Los casos de estudio anteriormente detallados, fueron los dispuestos como objetivos de las pruebas y elementos a satisfacer para garantizar la calidad de las implementaciones de

Mangoose. Con estos tamaños se abarcan los tamaños que serán utilizados con mayor frecuencia en el reconstructor con la escápula de cocodrilo y el maxilar, un valor irregular con la jeringuilla y comprobante del pegado y la desviación.

- Sobre las reconstrucciones realizadas para estos casos de estudio se han definido y aplicado las pruebas representadas en la Tabla 22.

Prueba requerida	Descripción	Estado
Lectura asíncrona	Comprobante de la correcta integración de AIO en operaciones de lectura.	√
Lectura adelantada	Comprobante de la funcionalidad de la estructura definida para establecer las proyecciones adelantadas.	√
Integración con CUDA	Comprobante de la correcta integración de CUDA y las GPU en el sistema.	√
Filtro	Etapa intermedia en la reconstrucción para garantizar el correcto funcionamiento hasta este punto.	√
Reconstrucción de volumen (básico)	Correcta reconstrucción del volumen con CUDA de manera secuencial.	√
Memoria compartida GPU	Explotación de recursos con memoria compartida y TILE	√
Paralelismo con OpenMP	Integración de OpenMP para la paralelización mediante hilos.	√
Integración multi-GPU	Integración de varias GPU para funcionamiento en paralelo.	√
Reconstrucción de volumen (paralelizado)	Correcta reconstrucción del volumen utilizando OpenMP y más de una GPU.	√
Escritura asíncrona	Comprobante de la correcta integración de AIO en operaciones de lectura.	√
Pegado de multi-camas	Comprobante del pegado de volúmenes aplicando correctamente los solapamientos para reconstrucciones compuestas por múltiples camas	√
Pegado con desviación	Análogo al caso anterior pero aplicando parámetros de desviación	√
Reconstrucción multi-cama	Reconstrucción de volumen con varias camas con pegado y desviación.	√

Reconstrucción de volumen 16 bit (final)	Reconstrucción definitiva del volumen utilizando números entero y comprobando su identidad con la versión inicial.	√
Reconstrucción de volumen 32 bit (final)	Reconstrucción definitiva del volumen utilizando números flotantes y comprobando su identidad con la versión inicial.	√

Tabla 22: Checklist

6. EVALUACIÓN DE RENDIMIENTO

En el siguiente capítulo se aborda la efectividad de las mejoras realizadas, se realizan diferentes ejecuciones para analizar características que puedan ayudar en líneas futuras de investigación y posibles optimizaciones adicionales.

6.1 Especificaciones hardware

Nodo de cómputo principal: Compuesto de 2 socket Intel(R) Xeon(R) CPU E5640 con una velocidad de reloj de 2.67GHz, cada socket está compuesto por 4 núcleos, que aumentados con *hyperthreading* otorgan un total de 16 hilos disponibles. Además consta de una memoria RAM de 64GB.

Unidades de procesamiento gráfico: Han sido utilizadas 4 GPU con las especificaciones detalladas a continuación:

Asus GTX 470

Especificaciones del motor GPU:	
CUDA Cores	448
GraphicsClock	607 MHz
ProcessorClock	1215 MHz
Texture Fill Rate (billion/sec)	34.0
Especificaciones de memoria:	
MemoryClock	1674 MHz
Standard MemoryConfig	1280 MB GDDR5
Memory Interface Width	320-bit
MemoryBandwidth (GB/sec)	133.9

Tabla 23: Especificaciones GTX 470

Figura 14: GTX 470

- Nvidia Tesla C2050

Especificaciones del motor GPU:	
CUDA Cores	448
GraphicsClock	575 MHz
ProcessorClock	1150 MHz
Texture Fill Rate (billion/sec)	34.0
Especificaciones de memoria:	
MemoryClock	1500 MHz
Standard MemoryConfig	3 GB GDDR5
Memory Interface Width	384-bit
MemoryBandwidth (GB/sec)	144

Tabla 24: Especificaciones Tesla c2050

Figura 15: Tesla c2050

- Asus GTX 480

Especificaciones del motor GPU:	
CUDA Cores	480
GraphicsClock	700 MHz
ProcessorClock	1401 MHz
Texture Fill Rate (billion/sec)	42.0
Especificaciones de memoria:	
MemoryClock	1848 MHz
Standard MemoryConfig	1536 MB GDDR5
Memory Interface Width	384-bit
MemoryBandwidth (GB/sec)	177.4

Tabla 25: Especificaciones GTX 480

Figura 16: GTX 480

- Nvidia Tesla c1060

Especificaciones del motor GPU:	
CUDA Cores	240
GraphicsClock	607 MHz
ProcessorClock	1296 MHz
Texture Fill Rate (billion/sec)	34.0
Especificaciones de memoria:	
MemoryClock	800 MHz
Standard MemoryConfig	4096 MB GDDR3
Memory Interface Width	512-bit GDDR3
MemoryBandwidth (GB/sec)	102.0

Tabla 26: Especificaciones Tesla c1060

Figura 17: Tesla c1060

6.2 Casos de estudio

Para las pruebas de los diferentes tamaños de proyección, se han utilizado tres casos de estudio reales:

- Proyecciones pequeñas, 512x512 puntos: Escápula de cocodrilo. Usados 2 ficheros de proyecciones con 180 proyecciones por fichero.
  Tamaño de los ficheros: 2 * 90 MB = 180MB
  Tamaño del volumen: 256 MB

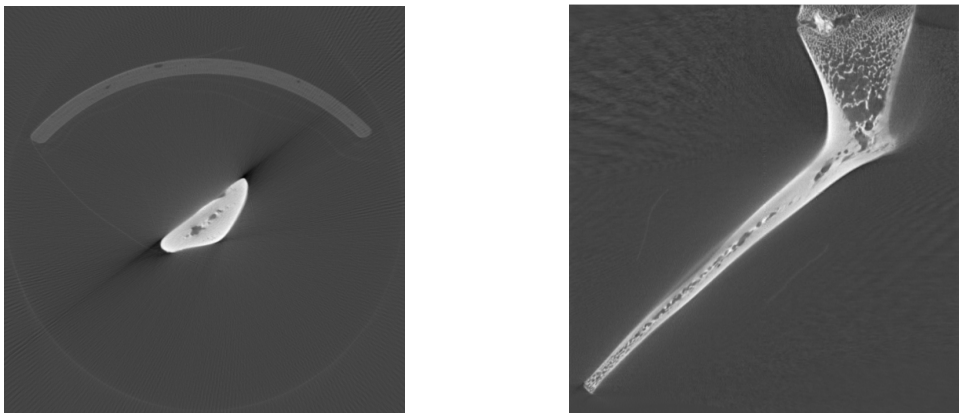

Figura 18: Escápula cocodrilo

- Proyecciones grandes, 2048x2048 puntos: Maxilar de cerdo. Usados 12 ficheros de proyecciones con 30 proyecciones por fichero.

Tamaño de los ficheros: 12 * 240 MB = 2880 MB

Tamaño del volumen: 256 MB

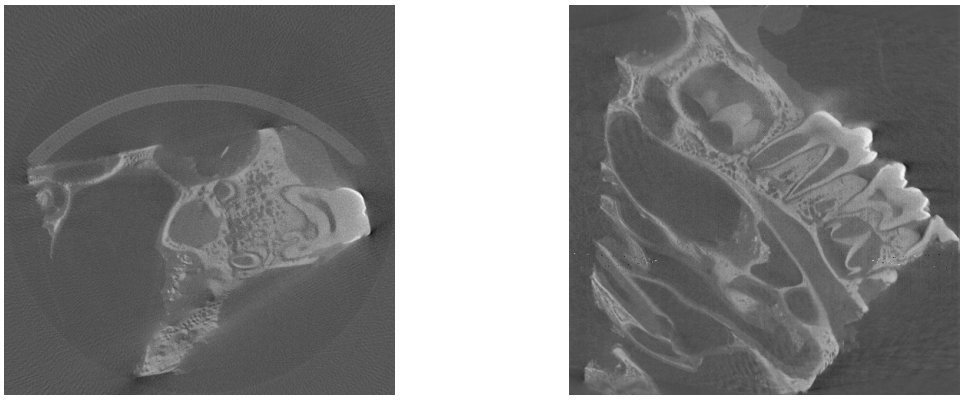

Figura 19: Maxilar cerdo

- Proyecciones intermedias y cama múltiple, 526x526 puntos: Jeringuilla. Usados 4 ficheros de proyecciones con 180 proyecciones por fichero (2 ficheros de 180 proyecciones por cama).

  Tamaño de los ficheros: 4 * 101,85 MB = 407,4 MB

  Tamaño del volumen: 297,63 MB / 510,83 MB (1 cama y 2 camas respectivamente)

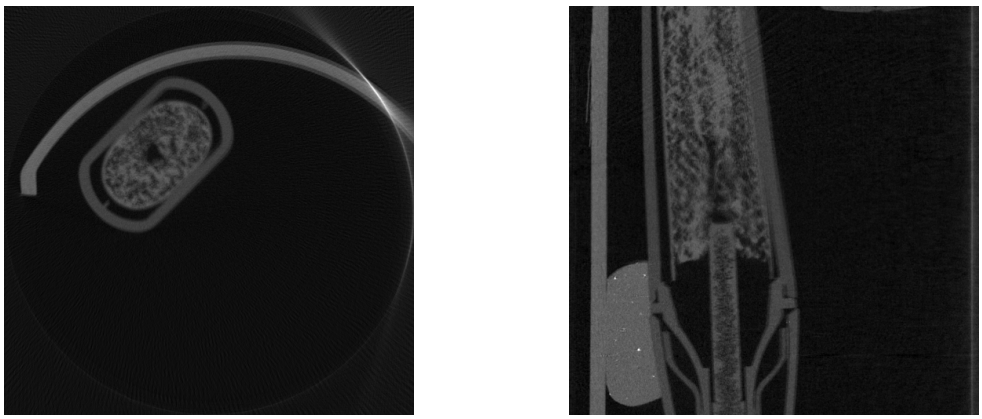

Figura 20: Jeringuilla

6.3 Lectura y escritura asíncronas

Una vez realizada la lectura asíncrona, el tiempo de lectura para las proyecciones pequeñas de 512x512 puntos se reduce a menos de 4 segundos (3,48 segundos exactamente), mientras que para las proyecciones de 2048x2048 puntos el tiempo de lectura queda en un poco menos de 37 segundos (36,98).

Como se puede apreciar, la mejoría en tiempo que se logra utilizando E/S asíncrona es muy grande, x2 en el caso de proyecciones pequeñas y x40 en proyecciones grandes. Si bien la diferencia entre el ratio de mejora de las proyecciones pequeñas a las proyecciones más voluminosas es muy grande, es interesante y muy beneficiosa para ambos casos de estudio.

Desglosado en detalle, la Tabla 27 las mejoras obtenidas al realizar acceso de asíncronos.

Tamaño de proyección	**Tiempo de LecturaSíncrono (segundos)**	**Tiempo de LecturaAsíncrono (segundos)**	**Ratio de mejora**
512x512 puntos	8	3,48	2,30
2048x2048 puntos	1480	36,98	40,02

Tabla 27: Lecturas asíncronas a través de AIO Read

Análogo a la lectura, se emplea la función *aio_write* para escribir en el fichero las proyecciones tras su proceso de filtrado y retroproyección Cuando el volumen va a ser reconstruido, es necesario esperar con *aio_suspend*de la misma forma que en la lectura, para proceder a su escritura en el fichero resultante una vez que ha sido leída entera del buffer.En la Tabla 28 se muestran los resultados obtenidos.

Tamaño de proyección	**Tiempo de EscrituraSíncrono (segundos)**	**Tiempo de EscrituraAsíncrono (segundos)**	**Ratio de mejora**
512x512 puntos	4,63	1,93	2,40
2048x2048 puntos	3,2	1,67	2

Tabla 28: Escrituras asíncronas a través de AIO Write

6.4 Lectura adelantada

Los siguientes gráficos muestran los resultados a la hora de aplicar lectura adelantada de un número dado de proyecciones, mostrando el tiempo de mejora y su valor óptimo en función del tamaño de la proyección.

Se ha probado el grado de mejora utilizando esta técnica con los casos de estudio más relevantes. En el primero de los casos, se realizó una evaluación con proyecciones de 512x512 puntos, con un número de [1..10]proyecciones adelantadas, dado que éste será el caso de estudio más utilizado en el reconstructor.

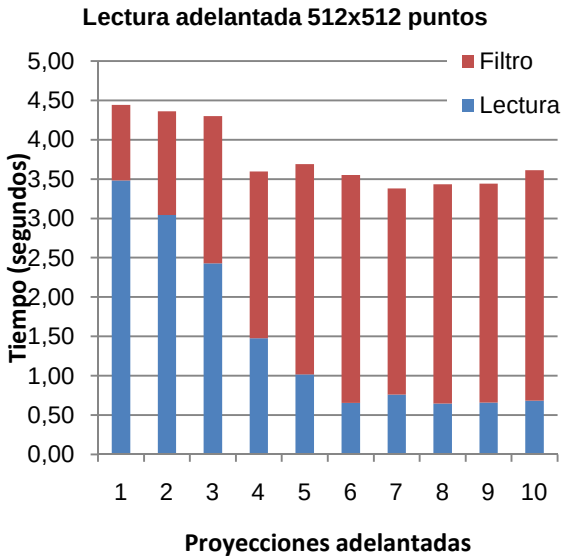

Figura 21: Lectura adelantada sobre proyecciones de 512x512 puntos.

Al aumentar el número de proyecciones adelantadas se puede observar la gran reducción en el tiempo de lectura, disminuyendo desde los 3,5 segundos iniciales a 0,75 al adelantar las 10 primeras proyecciones.

En contraposición, se puede apreciar que el tiempo de filtrado aumenta minimizando el rendimiento alcanzado utilizando esta técnica, la interferencia se produce por los costes fijos generados por inicialización y creación de contextos necesarios para el procesamiento.

Sin embargo, considerando el tiempo final obtenido, que es donde reside la mayor importancia, se aprecia una clara mejoría sobre la versión inicial dada, dando por positiva la utilización del adelantamiento en memoria.

En este segundo caso, seevaluaránproyecciones de gran tamaño, con un tamaño de 2048x2048 puntos, necesario para ver el grado de mejora en casos de estudio con un mayor peso y costes de

procesamiento.

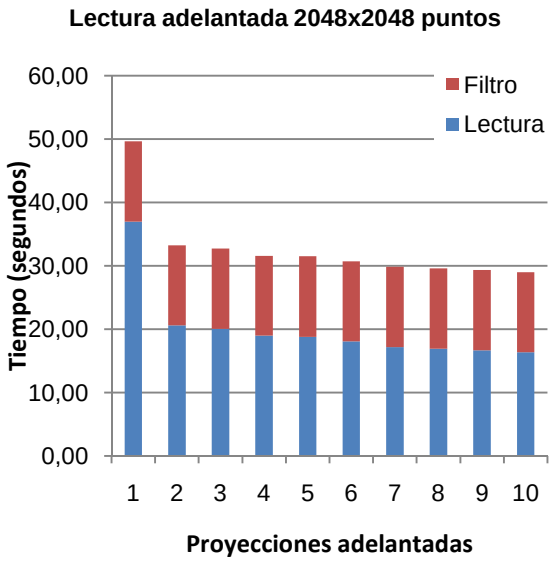

Figura 22: Lectura adelantada sobre proyecciones de 2048x2048 puntos.

Para esta situación donde requiere más tiempo el leer una única proyección, se aprecia en mayor medida los beneficios que aporta dejar un margen de proyecciones preparadas para minimizar tiempos. Las detenciones generadas por el uso de la primitiva *aio_suspend,* se ven muy reducidas compensando el tiempo ganado con las proyecciones listas mientras el resto de ellas siguen siendo leídas y procesadas.

Por último, se realiza otro caso de estudio sobre proyecciones de 564x564 puntos, con la particularidad de que, además, el volumen reconstruido está formado por más de 1 cama, buscando obtener los resultados sobre proyecciones de un tamaño no múltiplo de 2 y por la iteración en la operación de lectura.Como ya sucedía con el caso más pequeño 512 x 512, se obtiene un gran ratio de mejora aunque se empeore el tiempo de filtrado, compensándolo tal y como se muestra en laFigura 23.

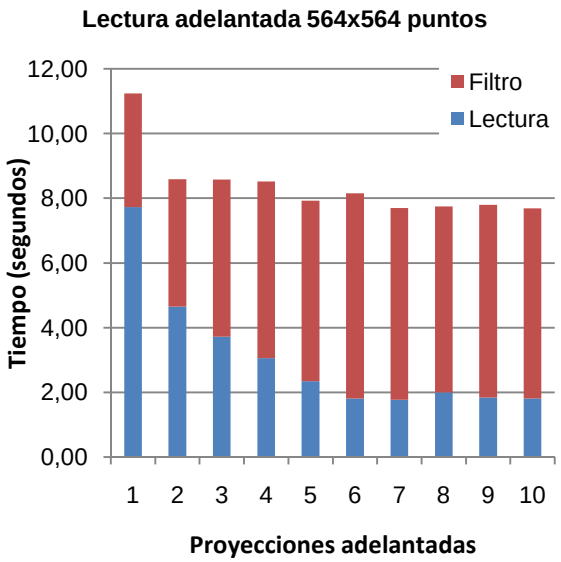

Figura 23: Lectura adelantada sobre proyecciones de 564x564 puntos.

El tiempo de filtrado vuelve a empeorar como en sucesos anteriores, mientras que en las proyecciones de 2048x2048 puntos el impacto es mucho menor y menos apreciado frente a 512x512 puntos por el retardo que suponen las proyecciones con un alto coste de lectura ahorrando 30 segundos frente a su versión inicial, en las proyecciones de 564x564 puntos vuelve a disminuir considerablemente el rendimiento obtenido en la lectura, pero en todos los casos de estudio se logra un índice de aceleración positivo, quedando demostrada y requerida como optimización para Mangoose. Como se puede observar en la figura, la lectura concurrente de proyecciones interfiere sobre el tiempo de ejecución en la fase de filtrado, debido principalmente a los posibles fallos de caché durante le proceso. Sin embargo está optimización puede alcanzar una mejora del 22% a partir de 7 proyecciones adelantadas.

6.5 Selección del planificador de hilos de OpenMP

Dentro de la sentencia de OpenMP, uno de los parámetros configurables es *Schedule*, encargado de establecer la planificación y división del trabajo entre los distintos hilos donde se genera el paralelismo.

Inicialmente la disposición del planificador se encontraba estática, distribuyendo de manera equitativa el número de proyecciones por hilo, sin embargo tras el mejor aprovechamiento de la memoria y las lecturas adelantadas, se modificó por una planificación dinámica para que las

proyecciones se repartieran en los hilos que iban quedando disponibles conforme terminaban su ejecución.

Utilizando como caso de prueba las proyecciones de mayor tamaño (2048 x 2048 puntos) debido a su mayor carga de trabajo y caso más representativo por el alto coste de procesamiento frente a las proyecciones más pequeñas.

Tras haber analizado los resultados de enfrentar una planificación dinámica frente a una planificación estática, se obtuvieron los siguientes resultados.

Tipo Planificación	Lectura (segundos)	Filtrado (segundos)	Retroproyección (segundos)	Pegado (segundos)	Total (segundos)
Dynamic	21,315	17,411	6,659	1,627	47,012
Static	29,544	12,911	6,347	1,245	50,047

Tabla 29: Schedule - dynamic vs static

Con estos datos, tal y como se muestra en la Figura 24 se aprecia la mejora al utilizar una planificación dinámica.

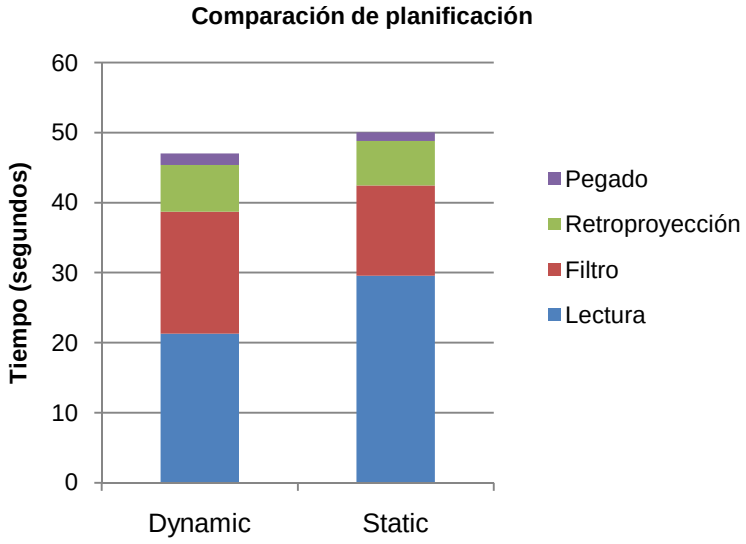

Figura 24: Schedule dynamic vs static

Por ello, aunque el tiempo de filtrado es donde más se acusa el cambio de planificación empeorando considerablemente con la cláusula de *dynamic*, en el cómputo total se aprecia que es óptimo utilizar esta planificación, desechando la hipótesis inicial de una planificación estática.

El empeoramiento es comprensible, puesto que tal y como ya se ha mencionado, existen los costes fijos en la creación de contextos e inicialización que se compensa con la variación en el tiempo de lectura, reducida casi en un 33% repartiendo las proyecciones de manera dinámica entre todos los hilos disponibles.

6.6 Paralelismo con multi-GPU

Las siguientes Tablas muestran las comparativas entre la utilización de un único dispositivo y su grado de mejora dependiendo del número de GPUs utilizadas. Están divididas en 5 columnas:

- Initial: Representa el tiempo que tardaba el reconstructor Mangoose en su etapa 0, antes de someterlo a la primera fase de optimización.
- 1 GPU: Tiempo de procesamiento requerido utilizando una única tarjeta modelo Tesla C2050.
- 2 GPU: Tiempo obtenido ejecutando paralelamente dos tarjetas modelo Tesla c2050.
- 3 GPU: Combinando las dos tarjetas modelo Tesla c2050 y una tarjeta modelo Asus GTX 470.
- 4 GPU: Utilizando las dos tarjetas modelo Tesla c2050 y las dos tarjetas modeloAsus GTX 470.

6.6.1 Proyecciones de 512x512 puntos

En este primer caso de estudio se han generado volúmenes de 512x512x512 puntos.

Con estos tiempos, se comparan en la Figura 25,aparecen las etapas de *Solape* y *Desviación*, pero dado que en este caso de estudio se emplea únicamente 1 cama, ambos valores eran 0. La etapa de *Solape* es la encargada de quedarse con las proyecciones válidas durante las reconstrucciones con múltiples camas mediante pesos aplicados y *Desviación* corrige la desviación estructural en los raíles del reconstructor cuando se introducen camas sucesivas en una reconstrucción.

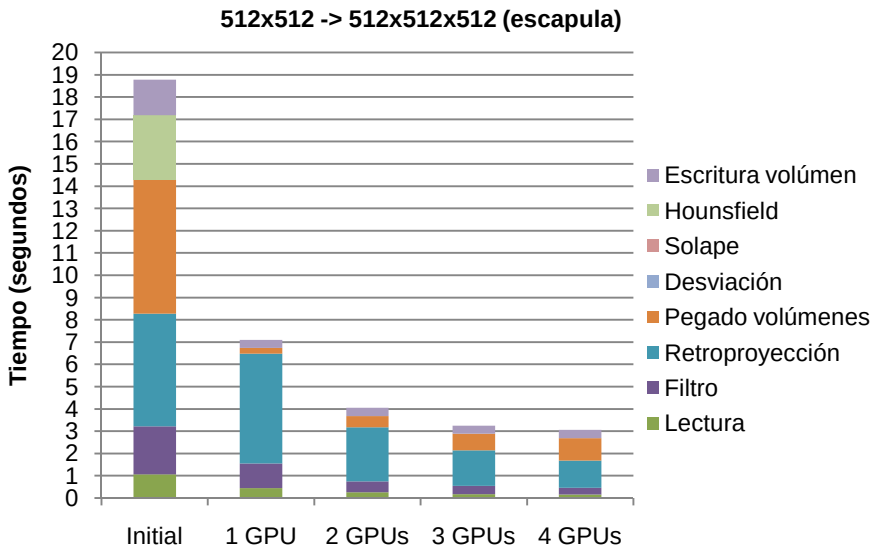

Figura 25: Paralelismo basado en multi-GPU sobre proyecciones de 512x512 puntos.

Como se puede apreciar en la Figura 25, el grado de mejora frente a la versión inicial es muy grande, reduciendo de los 19 segundos iniciales a los 7 segundos con una única GPU y alcanzando el máximo ratio de aceleración (481%) al utilizar las 4 GPUs.

El tiempo de lectura es reducido por la implementación de la lectura asíncrona, además de repartir la carga entre todos los dispositivos usando la cabecera del OpenMP mencionada anteriormente con una planificación dinámica.

Todos los tiempos en cada fase se han visto reducidos y se van reduciendo incrementalmente conforme aumenta el número de GPUs por el aumento en el grado de paralelismo, disponiendo de más hilos y más registros a los que asignar las proyecciones.

La excepción se encuentra en la etapa*Pegado volúmenes*, que dispone de un comportamiento inverso, ese contador de tiempo refleja la unión de los volúmenes parciales en un único volumen de gran tamaño y aunque disminuye frente a la versión inicial, se va deteriorando.

Por este motivo a medida que se disponen más dispositivos, se generan más volúmenes parciales que deben ser pegados y sincronizados, generando una latencia y un retardo que se ve aumentado en el coste en tiempo de procesamiento que debe ser asumido y analizado frente al impacto que genera en el sistema, aunque en este caso se aprecia el margen entre ganancia y pérdida sigue siendo muy favorable.

Existe un contador en la versión inicial, *Hounsfield*, que ha desaparecido, esto se produce porque ahora el cálculo de unidades *hounsfield* se realiza inmediatamente antes a su escritura en fichero, unificando esta etapa en el tiempo de escritura para la utilización de los dispositivos GPU.

Como se muestra en Figura 26, se centra el tiempo en las etapas más costosas de la aplicación y principal objetivo de las optimizaciones.

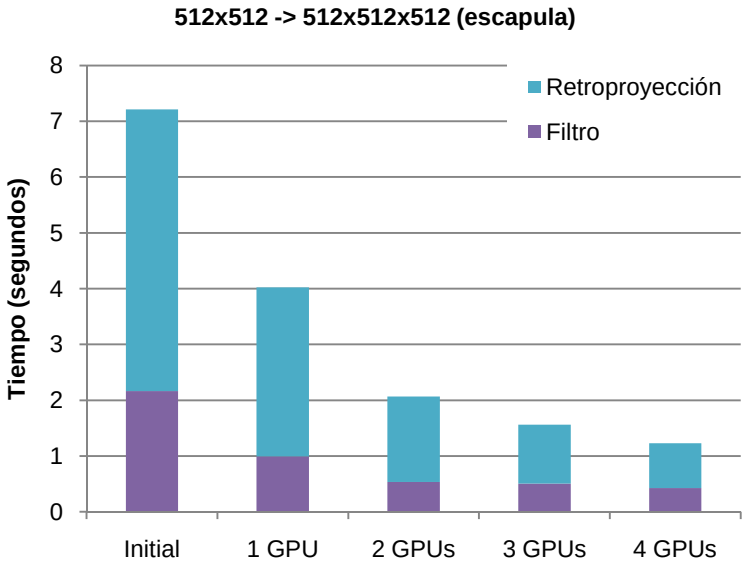

Figura 26: Paralelismo GPU 512x512 (RF y BP)

A partir de esta gráfica se observa el amplio ratio de mejora existente al utilizar un mayor número de dispositivos paralelos en las etapas más importantes de Mangoose, reduciendo a partir de un tiempo inicial superior a los 7 segundos hasta alcanzar un tiempo ligeramente superior a un segundo. La aceleración alcanzada comparando estas dos etapas es de 587%.

La mayor ganancia se produce en la retroproyección, etapa más costosa y donde se realizan más cálculos dentro de la GPU, reduciendo el coste de 5 segundos a menos de 1 segundo, el filtro se ve disminuido también pero dado que son proyecciones con un tamaño pequeño, el impacto temporal es mucho más pequeño, aunque numéricamente el ratio de aceleración sea un 400%, posteriormente se analizará el efecto en proyecciones con más puntos

En base a estos resultados obtenidos y apreciando una clara escalabilidad con el número de unidades de procesamiento gráfico, se obtiene como satisfactoria la utilización de una arquitectura paralela de varios dispositivos GPU que permitan repartir la carga de reconstrucción y optimicen el tiempo de procesamiento al menos en los casos de estudio más comunes, propiciando la generación de nuevos casos de estudio para proyecciones con otros tamaños.

6.6.2 Proyecciones de 2048x2048 puntos

Tras la prueba con las proyecciones pequeñas, que requieren un menor tiempo de procesamiento, se repite el proceso anterior utilizando proyecciones con un mayor tamaño, 2048x2048puntos esperando unos índices de mejora superiores frente a los obtenidos en el caso anterior.

Los tiempos de *Solape* y *Desviación* han vuelto a ser ignorados igual que en el supuesto anterior por ser 0, dado que son reconstrucciones de una única cama y no requieren de pesos en proyecciones solapadas ni una desviación en los raíles.

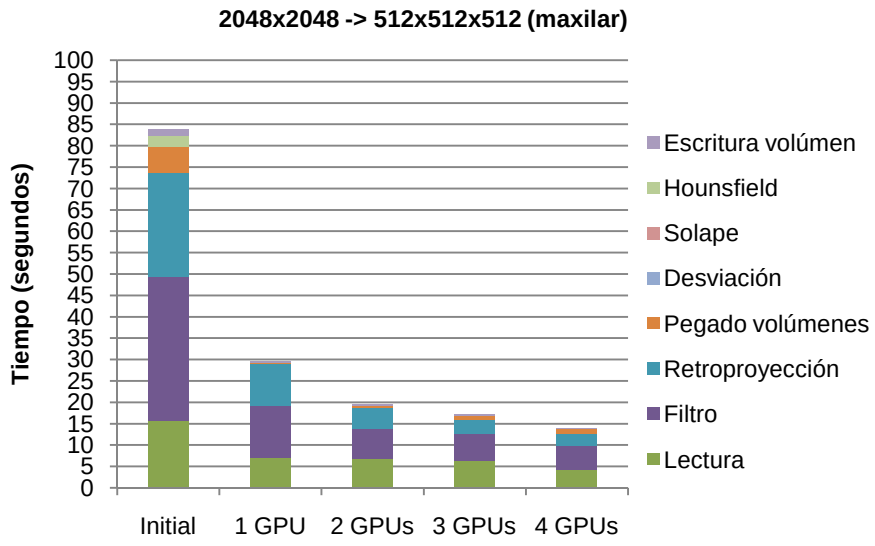

Figura 27: Paralelismo multi-GPU para proyecciones de 2048x2048 puntos y volumen generado de 512x512x512 puntos.

En este caso el grado de mejora en la reducción del tiempo se aprecia mucho más, alcanzando casi los 85 segundos en su etapa inicial y llegando a estar por debajo de los 20 al utilizar las 4 GPU, lo que supone alcanzar un ratio de aceleración delx5,4.

Dado que hay unos costes fijos en la creación de contextos, inicializaciones y reservas de memoria, el margen de optimización y reducción que permiten las proyecciones pequeñas es inferior a aquellas con tiempo de generación mayor donde en proporción la inicialización se puede considerar despreciable.

Los tiempos en las etapas de lectura gracias a la lectura adelantada y asíncrona, el filtrado y la retroproyección vuelven a ser las grandes favorecidas donde el conjunto se reduce de 72 segundos iniciales a 13 en su máxima paralelización. El pegado de los volúmenes parciales vuelve a empeorar

pero el coste es muy pequeño frente al total de la reconstrucción y el uso de escritura asíncrona aporta claros beneficios.

Análogo al caso de estudio anterior, centramos las optimizaciones en las etapas de filtrado y reconstrucción, comparando los resultados obtenidosen la Figura 28.

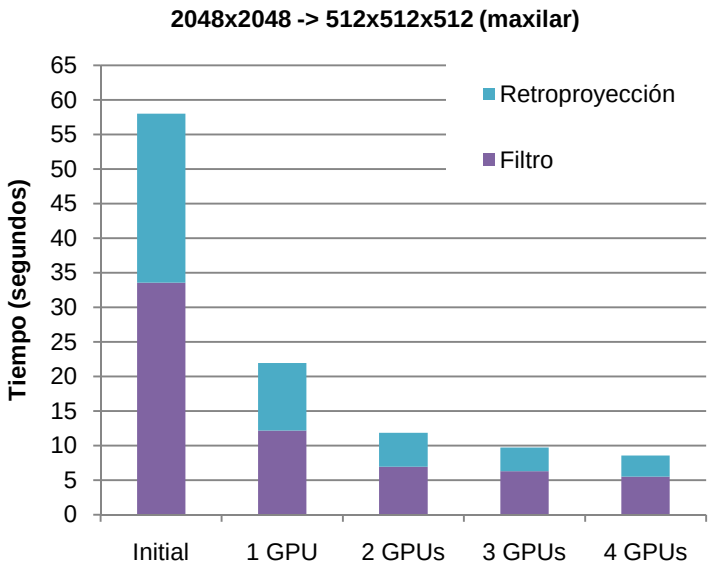

Figura 28: Paralelismo GPU 2048x2048 puntos (RF y BP)

Con la obtención de esta gráfica y en base al tiempo fijo, se observa que el ratio de aceleración alcanzado con proyecciones grandes es superior al de proyecciones pequeñas, en este caso es de 6,76.

Al tener un coste superior con un mayor porcentaje del tiempo total requerido destinado al procesamiento y los cálculos, son más susceptibles a la reducción de tiempo, especialmente la etapa de retroproyección, donde el uso de la memoria compartida y el paralelismo permite una reducción en el tiempo del 800%.

Hasta ahora, el uso de dispositivos paralelos favorece en las reconstrucciones con proyecciones independientes del tamaño con valores en potencias de 2, dando lugar al siguiente caso de estudio.

6.6.3 Proyecciones 526x526 – 1 cama

Después de utilizar proyecciones que facilitan los cálculos por sus dimensiones con números que pueden ser descompuestos en potencias de 2, se emplean dimensiones menos óptimas para la memoria caché disponible.

Además, este experimento consta de 2 camas, para tener en consideración los tiempos de *Solape* y *Desviación*. En primer lugar se obtienen los resultados para una única cama.

En base a los tiempos obtenidos, se genera la Figura 29para comparar los resultados utilizando varias GPU en paralelo.

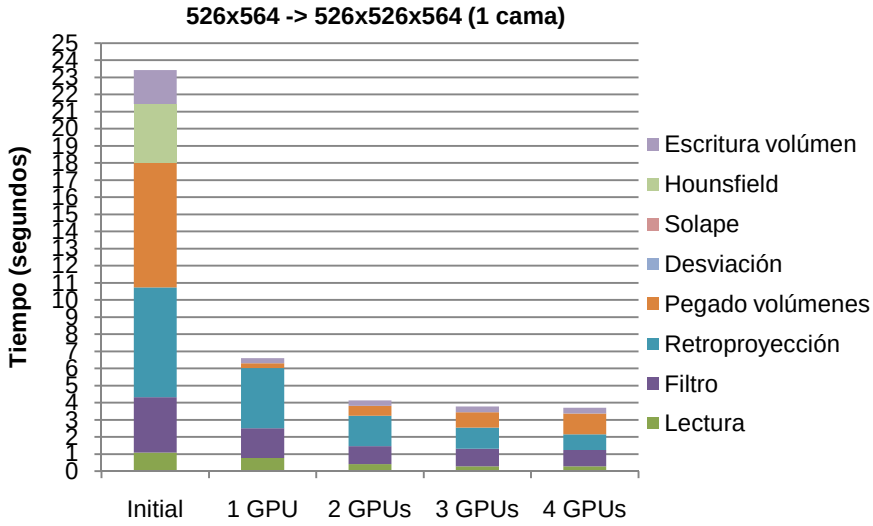

Figura 29: Paralelismo GPU 526x526 - 1 cama

Con un tiempo superior a los 23 segundos en su versión inicial, se logra reducir a un tiempo inferior a 4 segundos en su versión paralela con las cuatro GPU, eliminando la hipótesis de un empeoramiento o una mejora menos importante por el tamaño del bloque.

Suceden los mismos problemas que en casos anteriores y esperados en el pegado de volúmenes, pero no pierde en la contraposición ganancia – mejora aunque ésta sea menor y se pierda en escalabilidad.

El ratio obtenido usando todas las GPU frente a su versión inicial es de 375%. El peor de todos, aunque se mantiene de acuerdo a los márgenes esperados. Al no disponer de un tamaño de bloque óptimo, se producen más fallos de página y más accesos a memoria, empeorando el tiempo y por tanto el rendimiento, pero manteniendo su gran reducción en el tiempo de procesamiento.

Como en todos los estudios anteriores, se observan en detalle los resultados de las etapas más costosas y más importantes en el proceso de reconstrucción, el filtrado y la retroproyección obteniendo la Figura 30.

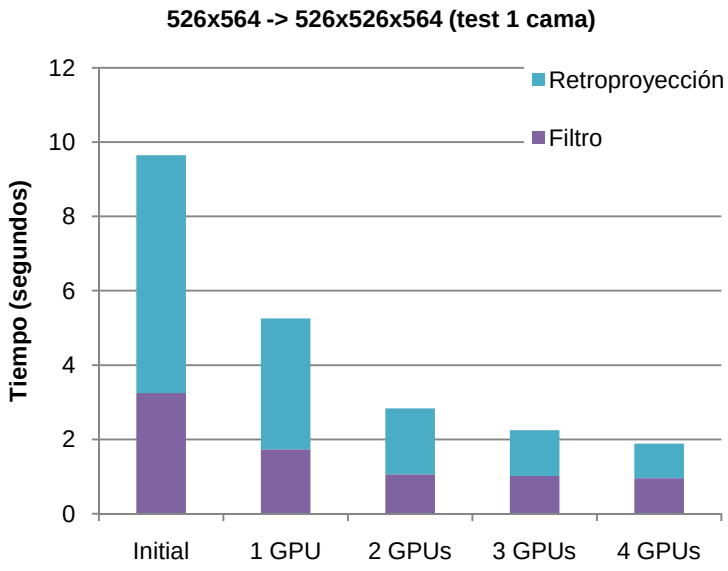

Figura 30: Paralelismo GPU 526x526 - 1 cama (RF y BP)

Con un ratio de aceleración alcanzado de x5 se toman por satisfactorias las optimizaciones y pruebas realizadas con mejoras representativas en la retroproyección. Aunque en el filtrado sean más sutiles, no otorgan ningún grado de empeoramiento y se pasa al caso de estudio de proyecciones de 526x526 puntos utilizando pegado de camas.

6.6.4 Proyecciones 526x526 – 2 camas

Disponiendo de un caso de ensayo con 2 camas, se procede a la toma de tiempos para tener en cuenta las mejoras en las etapas de *Solape* y *Desviación*, aunque estas etapas no han sido objetivo de grandes mejoras, se realizaron en ellas paralelizaciones simples en hilos y localidades en memoria, es necesario analizar un caso donde su valor no sea 0 para ver el impacto.

Al duplicar el número de camas, se duplica el tiempo requerido para la reconstrucción, manteniendo un valor aproximado de 2 veces el obtenido en el experimento.

Con los nuevos datos, se combinan y se contrastan para analizar la aceleración obtenida en la Figura 31.

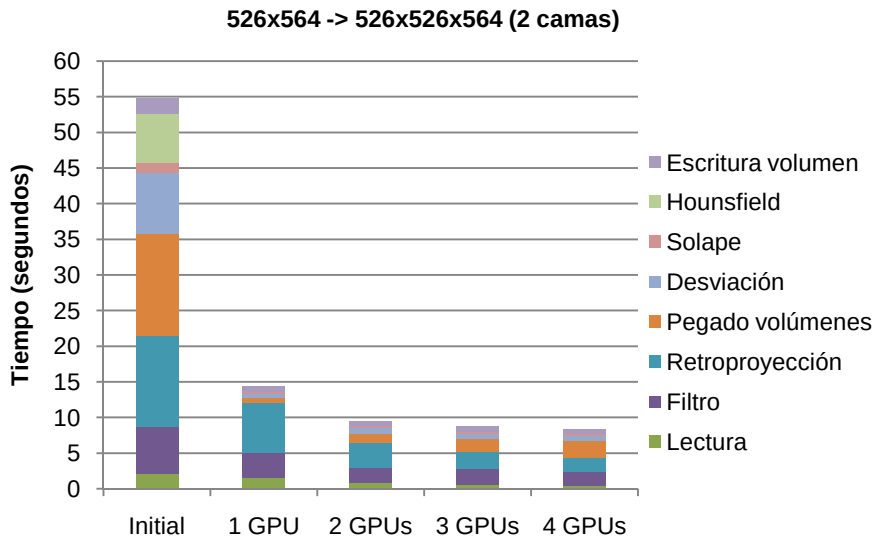

Figura 31: Paralelismo GPU 526x526 - 2 camas.

Como se ha explicado anteriormente, los resultados son parejos a realizar el estudio de 526x526 con única cama dos veces y sumar los tiempos obtenidos, alcanzando en este caso un ratio de x5,5 en el cómputo total de la reconstrucción.

En laFigura 32 se muestran los resultados considerando únicamente las fases de retroproyección y filtrado.

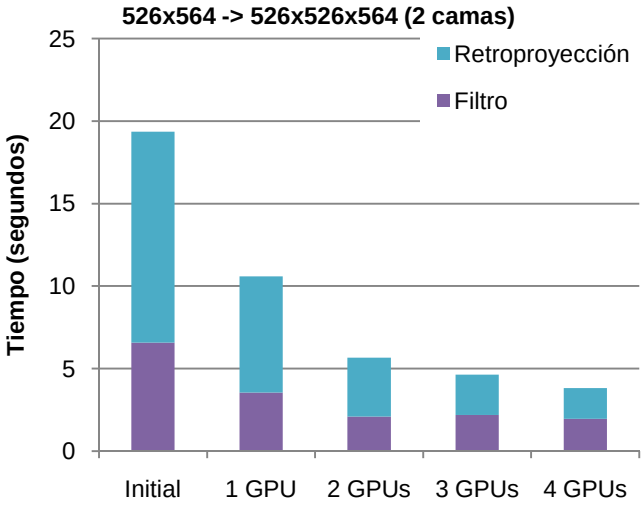

Figura 32: Paralelismo GPU 526x526 - 2 camas (RF y BP)

El ratio de aceleración alcanzado al utilizar las 4 GPUs es de x5. Se mantiene constante frente al obtenido al usar una única cama como estaba previsto y las mejoras al aumentar los dispositivos.

Por ese motivo, lo importante y necesario de esta evaluación son los tiempos invertidos en las etapas para cama múltiple, solape y desviación, como se muestra en la Figura 33.

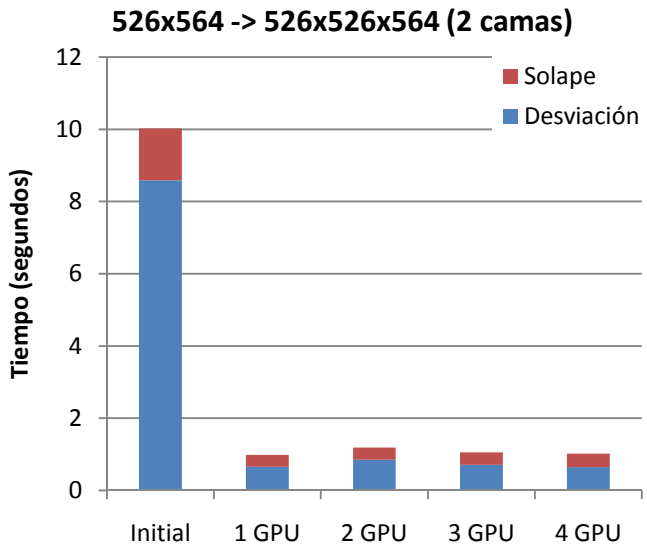

Figura 33: Paralelismo GPU 526x526 - 2 camas(Solape y Desviación)

Se toman en cuenta únicamente los tiempos de las etapas de *Solape*, donde se produce el solapamiento de las proyecciones comunes a ambas camas y mediante la aplicación de pesos una de ellas es desechada y la etapa de *Desviación*, donde se corrige una imperfección estructural que genera desvíos en las reconstrucciones pegadas.

Inicialmente, el tiempo requerido para realizar estas funciones era de 10,02 segundos, una vez paralelizado y optimizado, es reducido hasta 1,01 segundos, siendo la aceleración de x10.

Habiendo obtenido resultados satisfactorios en todos los casos de estudio propuestos y cubierto el abanico de "reconstrucciones habituales" que serán realizadas en su aplicación para la vida real en Mangoose, queda demostrada la efectividad paralela de más de una tarjeta gráfica, procediendo a un balance global en el uso de las mismas.

6.7 Comparativa de tiempos en paralelismo GPU

Haciendo un balance global de la utilización de las GPUs para los casos de proyecciones con un tamaño de 512x512 puntos y las proyecciones de 2048x2048 puntos, se comparan los tiempos invertidos en las secciones de paralelismo con diferente número de dispositivos.

Los resultados obtenidos para las proyecciones de 512x512 puntos se muestran en la Figura 34.

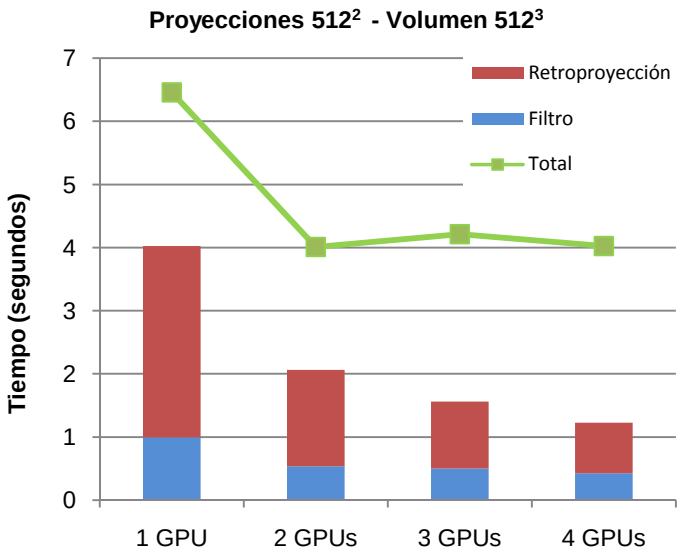

Figura 34: Comparación de tiempos paralelismo GPU (512x512)

Para el caso de estudio con proyecciones pequeñas se aprecia un empeoramiento al utilizar 3 GPUs siendo preferible utilizar dos de ellas, dos Nvidia Tesla C2050, o bien usar las cuatro disponibles en paralelo. Esto es debido principalmente a la sobrecarga del bus PCI. Accesos a memoria asíncronos podría mejorar el ancho de banda obtenido, permitiendo ejecutar concurrentemente transferencias de memoria y la ejecución de los *kernels* de CUDA.

Este efecto se produce por el efecto generado al incluir hardware heterogéneo, con unas capacidades técnicas diferentes, provocando interferencias entre ellos que generan el retraso por latencias, costes de sincronización y variedad en los tiempos de reloj.

Dependiendo del tamaño de las proyecciones, el coste de procesamiento y el grado de ocupación de las GPUs, la interacción puede aportar rendimiento o provocar su pérdida y es necesario valorar el beneficio obtenido frente a la pérdida que ocasiona paralelizar con recursos dispares.

Aunque ya se aprecia una pérdida en el rendimiento para este tamaño de proyección, posteriormente se verá con más detalle el impacto que produce la utilización de este hardware heterogéneo en los diferentes casos de estudio.

Para las proyecciones de mayor tamaño (2048x2048 puntos), se repite el mismo proceso que en el caso anterior, Figura 35, que muestra el comportamiento de la interacción en el paralelismo de las GPUs:

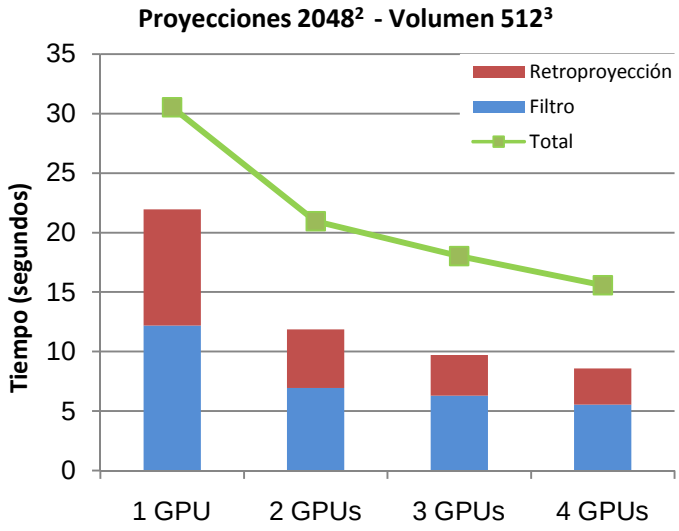

Figura 35: Comparación de tiempos paralelismo GPU (2048x2048)

En esta ocasión los resultados son beneficiosos conforme aumenta el número de GPUs, esto se debe al alto coste de generación del volumen, haciendo más despreciable las diferencias entre los dispositivos, en parte también a la planificación paralela, a favor del beneficio de tiempo ganado al poder paralelizar en mayor medida de simultaneidad las proyecciones en las fases de filtrado y retroproyección.

Los resultados difieren del caso de estudio con proyecciones pequeñas, donde utilizar una arquitectura heterogénea no era beneficioso para la reconstrucción realizada por Mangoose. Sin embargo, para proyecciones grandes, favorece el número de dispositivos sin tener tanto impacto las capacidades técnicas de las mismas.

En el siguiente apartado, se incluirán más GPUs a las pruebas para poder acotar la influencia que generan en la interacción del paralelismo utilizar dispositivos con capacidades técnicas muy diferentes.

6.8 Heterogeneidad de hardware

Hasta el momento, los casos de estudio siempre se han realizado con dispositivos idénticos, o en su defecto, dispositivos que se asemejan bastante en características técnicas. Este caso idílico no se puede lograr siempre y mediante los siguientes estudios se buscaba analizar el impacto que tenía utilizar dispositivos heterogéneos con una mayor diferencia en capacidad.

Para las distintas combinaciones se utilizaron modelos con especificaciones técnicas muy diferentes entre ellas, introduciendo la Asus GTX 480 y la Tesla c1060.

Con estos dispositivos, se realizaron combinaciones obteniendo los resultados con las proyecciones de 512x512 puntos y 2048x2048 puntos, respectivamente. Además se introdujo también el impacto del *prefetching* por la reducción de la capacidad en la memoria.

Los tiempos tomados se centran en las etapas de lectura (donde tiene impacto el número de proyecciones adelantadas), el filtrado y la retroproyección, por su alto coste.Por el alto número de combinaciones posibles, se muestran las más importantes y representativas.

6.8.1 GTX 480 + TESLA c2050 (512x512)

La primera prueba se realiza paralelizando con una tarjeta Asus GTX 480 y una tarjeta Tesla c2050, ambas GPUs forman la dupla heterogénea más potente entre todas las disponibles con las mejores especificaciones técnicas, los resultados se muestran en Figura 36yFigura 37.

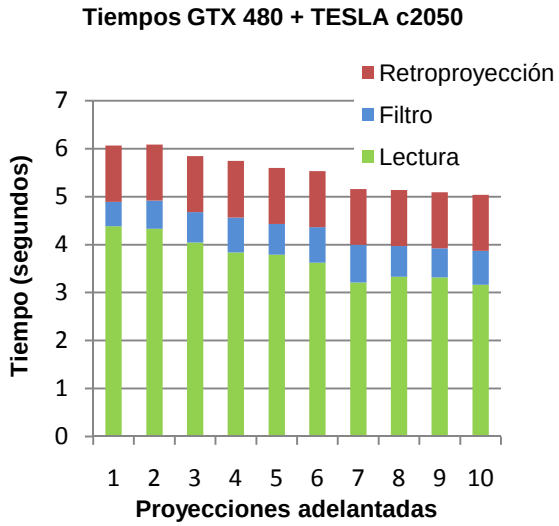

Figura 36: Tiempos GTX 480 + TESLA c2050 [512x512]

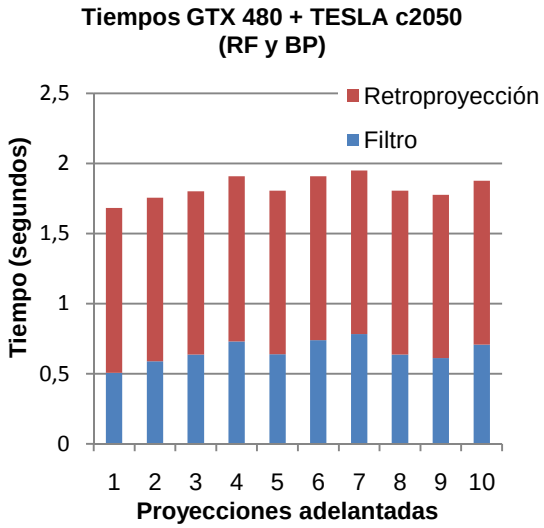

Figura 37: Tiempos GTX 480 + TESLA c2050 (RF y BP) [512x512]

Debido al alto rendimiento proporcionado por esta combinación, superior al obtenido en los estudios anteriores con la Tesla c2060 y la GTX 470, estos tiempos servirán como referencia para la comparativa posterior con otra arquitectura que servirá para decidir entre utilizar más GPUs aun siendo menos potentes, o es preferible utilizar dispositivos potentes pero en menos número.

Es llamativo el empeoramiento que se produce en la Figura 37 al aumentar el número de proyecciones adelantadas donde su valor óptimo reside en una única proyección, pero para los tiempos globales considerando la lectura usar 10 proyecciones sigue invirtiendo un tiempo menor.

6.8.2 GTX 480 + TESLA c2050 (2048x2048)

Utilizando las tarjetas Asus GTX 480 y Nvidia Tesla c2050, igual que en el caso anterior, se realiza el mismo experimento para proyecciones de tamaño grande, Figura 38 yFigura 39.

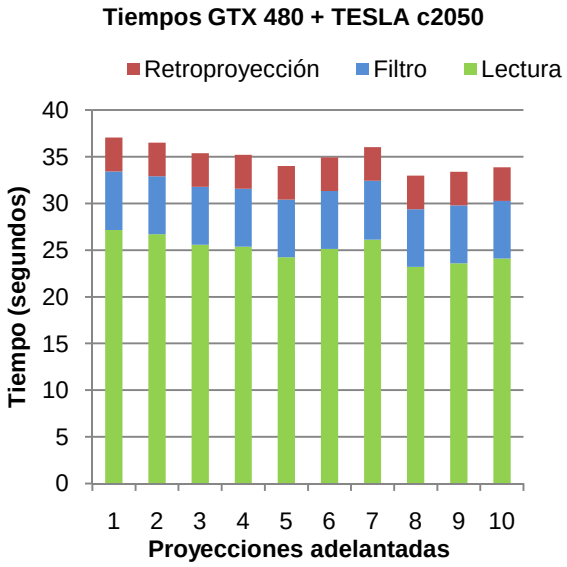

Figura 39: Tiempos GTX 480 + TESLA c2050 [2048x2048]

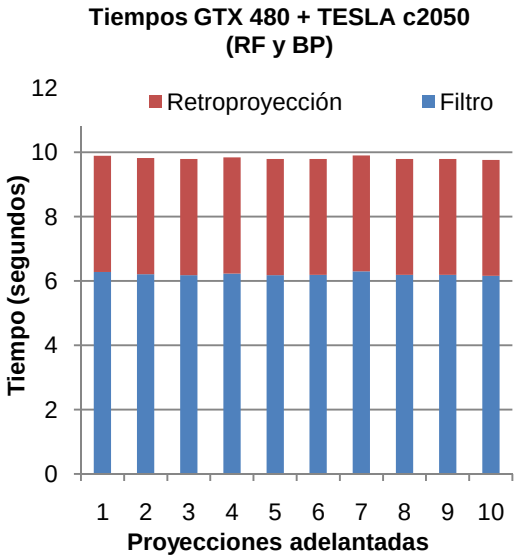

Figura 38: Tiempos GTX 480 + TESLA c2050 (RF y BP) [2048x2048]

Estos tiempos servirán como referencia para contrastarlos con los que se obtendrán al utilizar la arquitectura con una GPU más para estudiar el comportamiento de la heterogeneidad de las GPU y su impacto en la interacción entre ellas en la generación de volúmenes con proyecciones de gran tamaño.

Los tiempos en filtro y retroproyección no se ven tan afectados como en el suceso anterior manteniéndose constantes por el tiempo superior que requieren pero afectando igualmente al tiempo necesario para la lectura de las proyecciones.

6.8.3 GTX 480 + GTX 470 + TESLA c2050 (512x512)

Tras el experimento con la dupla más potente, comienzan a integrarse más dispositivos para probar los resultados de si es más rentable la calidad de las GPU que la cantidad de éstas. A la dupla utilizada antes, se le integra una Asus GTX 470 y se repite el caso de estudio con las proyecciones de menor tamaño, Figura 41 yFigura 40.

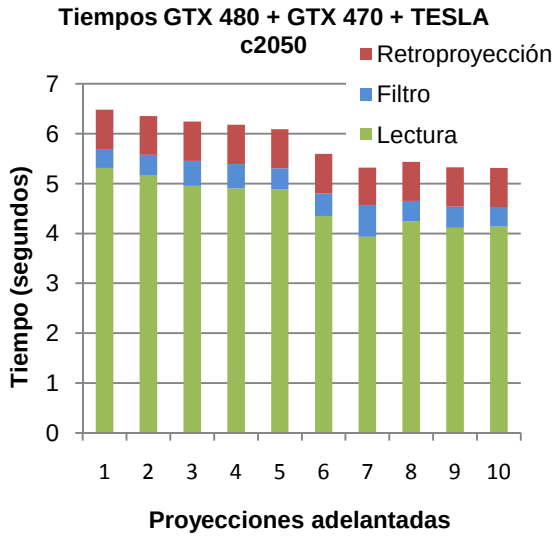

Figura 41: Tiempos GTX 480 + GTX 470 + TESLA c2050 [512x512]

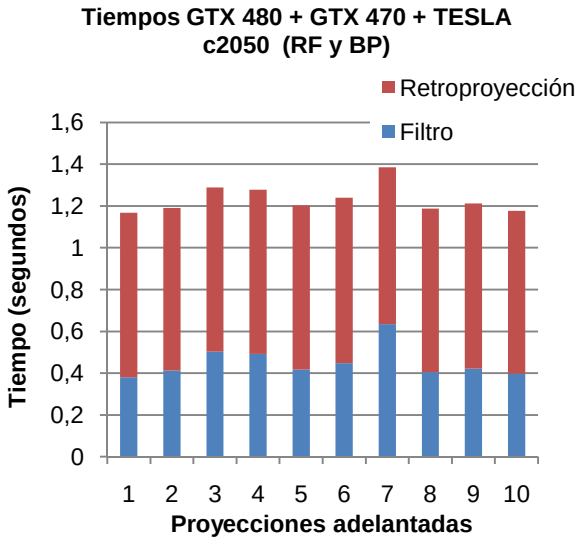

Figura 40: Tiempos GTX 480 + GTX 470 + TESLA c2050 (RF y BP) [512x512]

Como empieza a apreciarse en el primer caso de estudio, los tiempos de retroproyección y filtrado son mejores que al utilizar únicamente la GTX 480 y la Tesla c2050, pero por el contrario se pierde mucho tiempo en la lectura, provocando que, en el cómputo global, utilizar las 3 GPU tenga un peor rendimiento de un 10% en el mejor de los casos y no ganando en ninguno frente a la referencia considerada anteriormente.

Si se puede disponer de un estado en el que las proyecciones se puedan almacenar leídas permanentemente, es adecuado utilizar este tipo de arquitecturas. Analizando las gráficas por separado y contrastando las Figura 38 y Figura 40, el tiempo entre filtrado y retroproyección es superior al utilizar una mayor cantidad de GPUs con un margen superior al 33%, pero en caso negativo y el tiempo global, que es lo prioritario, es preciso prestar especial atención a las especificaciones técnicas.

Una GPU técnicamente inferior, puede suponer un retraso al repartir las cargas de trabajo en determinadas por necesitar mayor tiempo de procesamiento y latencia en la comunicación generando la detención del resto de dispositivos y retener el avance de Mangoose.

6.8.4 GTX 480 + GTX 470 + TESLA c2050 (2048x2048)

Tras haber empeorado los tiempos en el estudio anterior, se repite con proyecciones grandes usando lasAsus GTX 480, 470 y la Nvidia Tesla c2050, obteniendo lasFigura 42yFigura 43

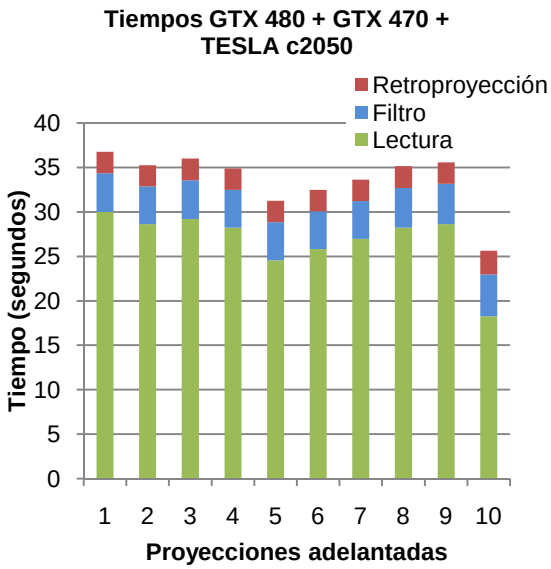

Figura 43: Tiempos GTX 480 + GTX 470 + TESLA c2050 [2048x2048]

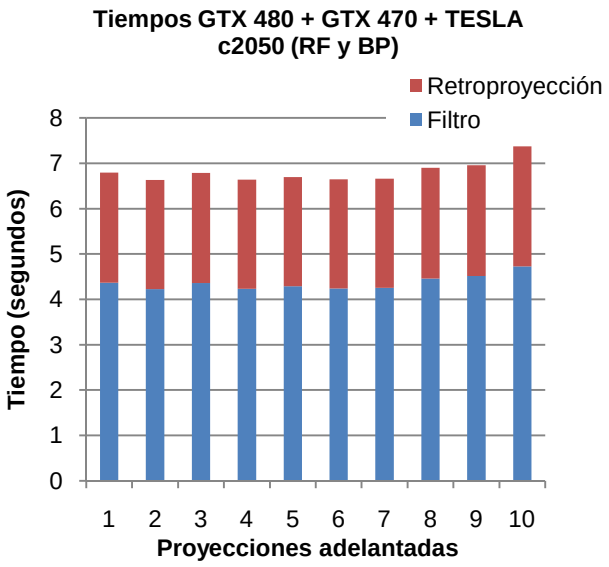

Figura 42: Tiempos GTX 480 + GTX 470 + TESLA c2050 (RF y BP) [2048x2048]

Una vez realizadas las pruebas, se observa que tanto filtrado como retroproyección han mejorado, de la misma forma que en las proyecciones pequeñas con una mejoría del 25%.

Sin embargo, en contraste a lo sucedido con las proyecciones de 512x512 puntos, cuando se introduce la nueva GPU, para proyecciones grandes los tiempos mejoran globales mejoran aunque el tiempo invertido en la lectura sea superior al obtenido en el primer caso de estudio del orden de un 10%.

Cobra gran importancia la lectura adelantada, donde puede llegar a reducirse el tiempo de lectura a 18 segundos utilizando un adelantamiento de 10 proyecciones, que sumado a la mejora en de retroproyección y filtrado, producen una gran mejora del 25%, siendo preferible en esta ocasión un mayor número de GPU frente a la potencia de las mismas.

6.8.5 GPUS SEGÚN EL TAMAÑO DE LA PROYECCIÓN

De acuerdo a las pruebas hechas enfrentando el tamaño de las proyecciones con las GPU utilizadas, se deducen las siguientes conclusiones:

- Para proyecciones de tamaño pequeño, donde la reconstrucción se realiza en una cantidad de tiempo pequeña, no favorece introducir más GPUs si éstas son de menor capacidad.
  Esto se debe a que se divide el trabajo entre todos los dispositivos y no se está explotando el índice de ocupación, siendo tiempos pequeños, aquellas GPUs más veloces terminan y deben esperar a la finalización de las más lentas, generando una demora en el tiempo inferior a la ganancia.

- Para proyecciones de tamaño grande, la reconstrucción requiere una carga muy superior en coste computacional, para este tipo de volúmenes, es preferible la existencia de un mayor número de GPUs aunque sean de menor potencia.

  Los recursos tienen un grado de explotación mucho mayor y aunque la carga que realiza la GPU más limitada técnicamente es inferior a la del resto, el tiempo que ahorra por ésta vía es superior al empeoramiento que conlleva la demora en transferencias y sincronizaciones.

Las siguientes tablas muestran las diferencias de los tiempos totales obtenidos para cada uno de los casos de estudio, en la columna *DIFERENCIA*, está representada la resta del tiempo entre la dupla de *GTX 480 + TESLA c2050* y la terna *GTX 480 + GTX 470 + TESLA c2050*.

Esta diferencia está representada en negrita, destacando los valores más positivo y negativotanto en la terna como en la dupla.

TIEMPOS TOTALES DE 2048x2048 (en segundos)

GTX 480 + TESLA c2050	GTX 480 + GTX 470 + + TESLA c2050	DIFERENCIA
37,0431247	36,7682463	0,2748784
36,5089395	35,2646668	1,24427267
35,3671975	35,9895577	-0,6223602
35,1893651	34,8728321	0,316533
34,01107	31,23518	2,77589
34,9354801	32,4758684	2,4596117
36,0353503	33,6099775	2,4253728
32,9823145	35,12494	-2,1426255
33,3842612	35,5867678	**-2,2025066**
33,8724535	25,6302499	**8,24220357**

Tabla 30: Diferenciatiempos en proyecciones de 2048x2048 puntos

TIEMPOS TOTALES DE 512x512 (en segundos)

GTX 480 + TESLA c2050	GTX 480 + GTX 470 + + TESLA c2050	DIFERENCIA
6,067	6,480	-0,413
6,084	6,352	-0,268
5,847	6,241	-0,394
5,745	6,179	-0,434
5,599	6,089	**-0,490**
5,534	5,593	-0,059
5,16	5,318	-0,158
5,138	5,433	-0,295
5,092	5,324	-0,232
5,036	5,313	-0,277

Tabla 31: Diferenciatiempos en proyecciones de 512x512 puntos

Como se aprecia en la Tabla 30 y la Tabla 31los resultdos apoyan los conceptos explicados con anterioridad donde para proyecciones de gran tamaño es preferible, por norma general, las tres tarjetas, mientras que para las proyecciones pequeñas, utilizar esta arquitectura empeora el resultado en todos los casos.

6.9 CPU vs GPU

En paralelo a la versión GPU generada para el reconstructor y que ha servido como foco para el desarrollo del proyecto de investigación, también fue creada una versión análoga para CPU con la implementación de las mismas optimizaciones que su versión en GPU pero si utilizar el API de CUDA.

La finalidad en este apartado es contrastar el mejor de los tiempos obtenidos con la versión paralela en CPU frente al mejor de los tiempos en la versión que utiliza GPU para así observar el grado de mejora que aporta al ratio de aceleración utilizar el API de CUDA en procesos de reconstrucción digital y demostrar la necesidad y la importancia de usar estas tecnologías para explotar el rendimiento y maximizar la velocidad.

Para la versión de CPU se dispuso de una máquina con 4 socket, cada socket compuesto por 6 núcleosa 1,8GHz. En total supone 24 hilos disponibles para ser paralelizados, que se duplican y convierten en 48 por el uso del *hyperthreading*.

Durante el proceso de reconstrucción se generan dos grandes niveles de paralelismo, el primero se produce durante la lectura de las proyecciones y el segundo durante las etapas de filtrado y retroproyección. Diferentes pruebas concluyeron en que el punto óptimo de esta arquitectura se alcanzaba utilizando 8 hilos principales en el paralelismo de primer nivel y dejando 6 hilos para el segundo nivel de paralelismo a cada uno de los principales, obteniendo así 8x6 = 48 hilos disponibles.

Para alcanzar esta distribución se realizaron ejecuciones con diferente número de hilos, inicialmente con un único nivel de paralelismo, como se muestra en laFigura 44.

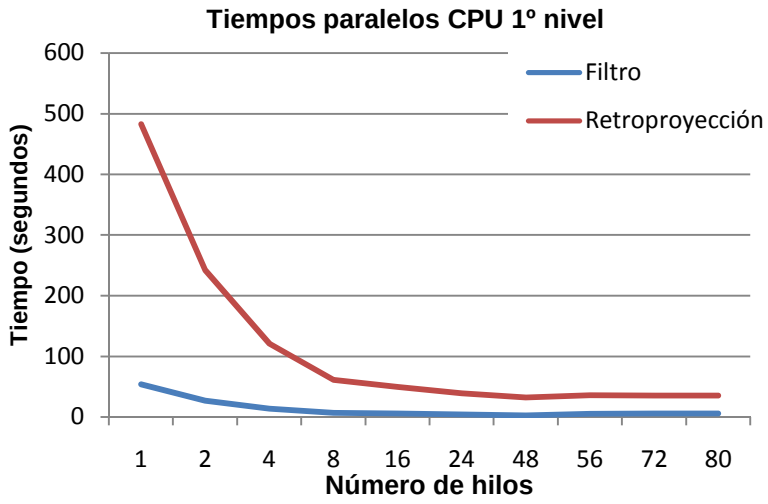

Figura 44: Tiempos paralelos CPU primer nivel

La escalabilidad e mantiene proporcional al número de hilos hasta alcanzar 8 hilos en primer nivel, demostrando que es el punto de inflexión donde aumentar el número de hilos y el coste que esto supone no es rentable con el beneficio obtenido y fijando el valor óptimo de hilos en 8 reservando los restantes para un segundo nivel de paralelización.

Tras anclar el uso de hilos en el primer nivel de paralelismo, se introduce el segundo nivel y se repite el cálculo de tiempos empleando el resto de recursos disponibles, Figura 45.

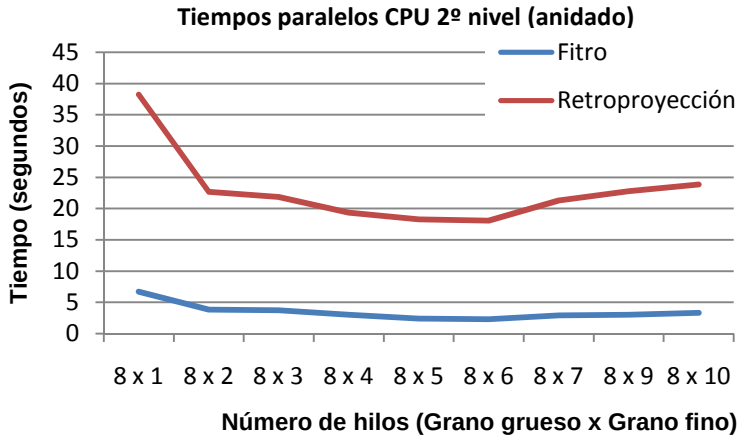

Figura 45: Tiempos paralelos CPU segundo nivel

Los resultados muestran que el mejor tiempo aplicando el segundo nivel de paralelismo se alcanza con 6 hilos, explotando todos los recursos de hardware disponibles (8x6 = 48 hilos) y empeorando de nuevo si se generan más, dejándolo como distribución final.

En laFigura 46 se muestra el ratio de mejora obtenida frente al tiempo de la versión inicial contraponiendo los dos niveles de paralelismo, aplicando paralelismo al reparto de proyecciones entre los núcleos (1º nivel) y aplicando dos niveles de paralelismo anidado de distinto tamaño de grano (2º nivel).

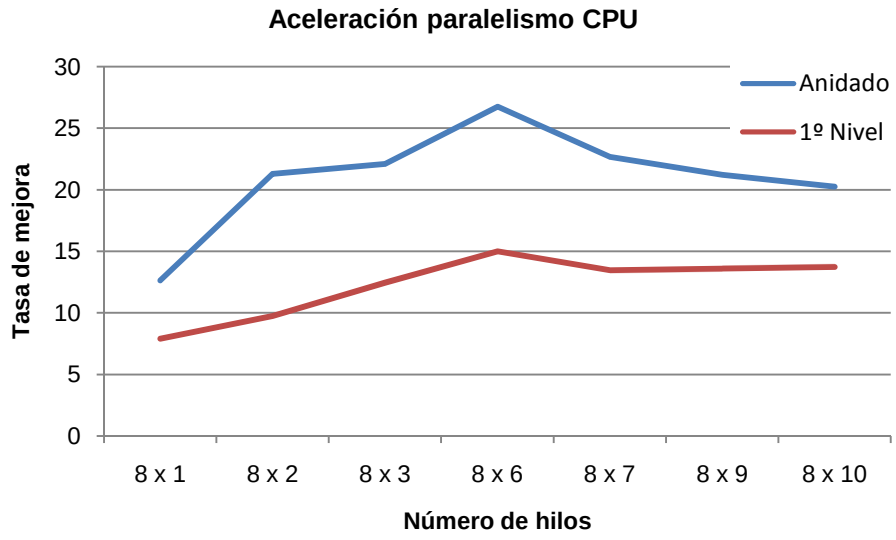

Figura 46: Ratios de aceleración en paralelismo CPU

Como se aprecia, utilizar 2 niveles de paralelismo proporciona un rendimiento muy superior frente a un único nivel, mientras que con el paralelismo de primer nivel exclusivamente podría obtenerse un rendimiento de 14 veces más rápido, al aplicar la doble paralelización se alcanza un pico máximo de x27. Este último dato es particularmente relevante porque se logra un resultado super-escalar, o lo que lo mismo, obtener un rendimiento equivalente a disponer de 26 núcleos en lugar de los 24 existentes, sinónimo de no sólo un buen reparto paralelo, sino también de una buena gestión en el paralelismo, ampliando nuevas posibilidades para trabajos futuros.

Una vez obtenidos los tiempos más adecuados para la versión en CPU, se procede al contraste con su versión homóloga utilizando la arquitectura proporcionada por las GPUs.

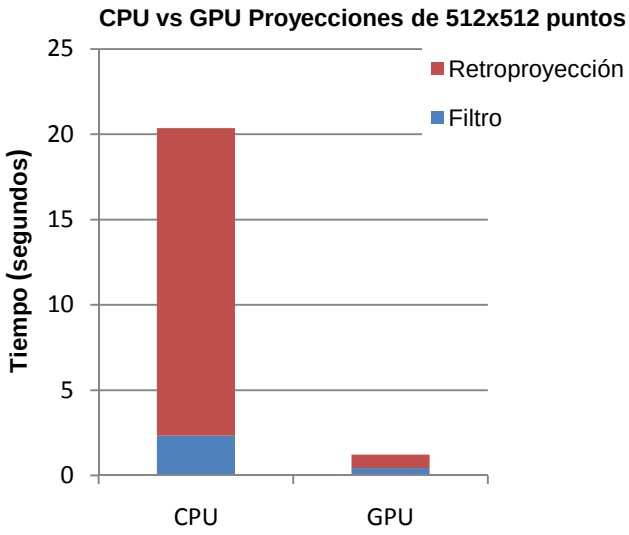

Figura 47: CPU vs GPU 512x512

Para la primera comparativa se han utilizado los tiempos de las proyecciones pequeñas usando dos Nvidia Tesla c2050 y dosAsus GTX 470, calculando laaceleración alcanzada por la versión GPU frente a la CPU se obtiene un ratio de mejora total de x16,57.

Particularmente llamativa la mejora del cálculo en la etapa de retroproyección con un 1800% en mejora de rendimiento, viéndose disminuido por un menor ratio en el filtrado que se ve reducido igualmente a un tiempo muy pequeño incluyendo los costes de creación de contextos, desechando .más que por casos de necesidad e indisposición de hardware de procesamiento gráfico explícito, la utilización de una versión en CPU.

Repitiendo el proceso para las proyecciones de gran tamaño, se obtiene la siguiente gráfica de comparar los tiempos de la versión en CPU con los tiempos de la versión GPU:

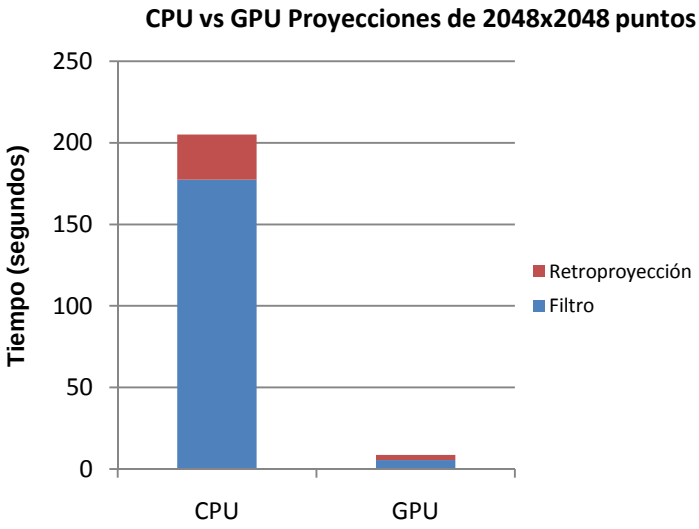

Figura 48: CPU vs GPU 2048x2048

En el caso de las proyecciones grandes la diferencia de tiempos es muy superior al producido con las proyecciones de menor tamaño, en esta ocasión el ratio de aceleración alcanzado se sitúa en 23,92.

En contraposición a las proyecciones de 512x512 puntos, para proyecciones grandes la CPU se demora mucho más tiempo en la ejecución del filtrado de las imágenes que en concluir la retroproyección, teniendo un 3300% en mejora de rendimiento.

Con estos dos resultados queda manifiesta la mejora utilizando CUDA y las GPUs como opción prioritaria en caso de disponer del hardware necesario para poder ejecutar correctamente esta versión, no sólo se gana mucho en tiempo de ejecución, sino en calidad y prestaciones, ampliar la arquitectura basándose en CPU requiere de un equipo computacionalmente muy preparado, con extensos consumos de energía.

En inversión económica inicialmente puede parecer superior la utilización de hardware GPU y aunque este Proyecto Fin de Carrera está destinado a un entorno académico y de investigación, en caso de comercializarlo, la manera de ser competitivo es reduciendo el tiempo y otorgando unas prestaciones que permitan competir con lo que existe en la actualidad y enfrentado a otros estudios similares, reduciendo la creación de la alternativa CPU como comparativa que permita reforzar los resultados obtenidos para la amplia superioridad de las plataformas destinadas explícitamente a desempeñar esa tarea..

7. CONCLUSIONES Y PRESUPUESTO

Este Proyecto Fin de Carrera ha abordado la necesidad de desarrollar un entorno que optimizara los tiempos de reconstrucción de imagen en tomografía por rayos X basada en geometría cone-beam. En concreto se ha optimizado Mangoose, una aplicación en C basada en el algoritmo propuesto por Feldkamp, Davis y Kress. La reducción en los tiempos de reconstrucción favorecería la inclusión de nuevas aplicaciones clínicas que requieran tiempo real como la monitorización en radioterapia o la cirugía asistida por imagen [1]. Por otra parte, la adaptación a métodos de reconstrucción iterativos es inmediata, lo que permitirá su extensión en la práctica clínica en las tareas de reconstrucción de imagen médica digital, en concreto para aplicarlo a la tomografía computarizada mediante el uso de haz cónico.

Para conseguir cumplir los objetivos planteados al inicio del proyecto, se han alcanzado diferentes hitos que han propiciado el satisfactorio desarrollo del proyecto:

- Utilizar herramientas que permitieran implementar las unidades de procesamiento gráfico (GPU), para unos cálculos efectivos explotando su potencial. En este caso se utilizó el API

de CUDA desarrollado por Nvidia por su facilidad y sencillez en comparación a las librerías propias para este tipo de funciones.

- Proporcionar soporte para el paradigma de programación paralela OpenMP mediante la generación de hilos y el aprovechamiento de la tecnología*hyperthreading*, dividiendo en un gran número de tareas simultáneas la ejecución y así reducir el lastre de tiempo generado por un comportamiento secuencial.

- Uso de las llamadas asíncronas con la interfaz de POSIX AIO tanto para lectura como para escritura. Esta técnica permite eliminar los tiempos de detención hasta la finalización de alguna de estas dos tareas, pudiendo hacer una implementación basada en productor – consumidor. Este modelo de procesamientoha supuesto una gran mejora para las proyecciones de gran tamaño.

Además de estos puntos, se han tenido en cuenta consideraciones como la localidad de la memoria y las lecturas adelantadas.En conjunto, los tiempos alcanzados para los diferentes casos de estudio en la nueva versión del programa, Mangoose Plus, rondan las 7 veces inferiores a los de la versión inicial (Mangoose), demostrando la importancia de estos nuevos entornos de desarrollo y las nuevas posibilidades que generan para seguir profundizando y ampliando su potencial.

Además de los objetivos presentados anteriormente, personalmente el proyecto a sido una oportunidad de participar en un proyecto real. Este Proyecto Fin de Carrera me ha permitido aprender nuevos paradigmas de programación como los proporcionados por CUDA y OpenMP.

Como líneas futuras que permitan expandir el proyecto:

- Se propone realizar una evaluación más concisa sobre las optimizaciones sobre los accesos de memoria de la aplicación. Para ello se propone medir contadores hardware, donde es posible obtener parámetros como la tasa de fallos tanto en memoria como en caché del sistema.

- Como se ha visto, en función del caso de estudio y del tamaño de proyecciones que vaya a ser utilizado hay diferentes configuraciones que favorecen un caso u otro como puede ser el número de proyecciones leídas o el número de hilos por bloque o las GPUs a utilizar.

Por ello sería adecuado el desarrollo de una utilidad de configuración automáticas, que de acuerdo a las capacidades técnicas del hardware disponible y el caso de estudio a reconstruir, escogiera y fijara los valores óptimos para la ejecución de Mangoose de manera automática.

- Una gran parte del tiempo invertido se debe a los costes fijos de la creación de contextos, inicialización de variables, reservas de memoria y la lectura de las proyecciones.Una línea futura que permitiría reducir en una gran medida los tiempos, en especial las proyecciones de menor tamaño, sería la generación de un entorno que pudiera mantener estas necesidades de manera permanente, donde únicamente Mangoosese encargara del procesamiento desde la etapa de filtrado y reconstrucción.

- Se ha investigado sobre la viabilidad de desarrollar una versión análoga a la utilizada para este proyecto pero sin utilizar GPU, ya que estos dispositivos hardware pueden tener un alto coste de adquisición, en mayor medida si se precisa una mayor eficiencia. Por ello, como abaratamiento de costes es posible desarrollar una versión de CPU basada puramente en arquitecturas multi-core, como es el caso de la futura generación de dispositivos Intel MIC[1].

- Recientemente, la compañía Intel está desarrollando ArBB*(ArrayBuildingBlocks)*, es una librería diseñada para proporcionar paralelismo vectorial y una alternativa a considerar frente a CUDA en el desarrollo de una versión CPU. El objetivo de ArBB es lograr escalabilidad en aplicaciones diseñadas en el lenguaje C++. Para llevar a cabo este trabajo futuro sería necesario una conversión del código utilizado para este proyecto al lenguaje C++ y con esa base comenzar el desarrollo aprovechando las ventajas deArBB.

Aunando estas líneas futuras, podría alcanzarse una implementación automática y permanente de una aplicación que paraleliza y optimiza los recursos para la reconstrucción que fuera independiente del hardware, de la disponibilidad o no de unidades de procesamiento gráfico, del sistema operativo y que no necesitara supervisión más allá del mantenimiento, facilitado por la modularización en funciones de las distintas etapas de Mangoose Plus.

7.1 Presupuesto

[1]http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html

Para realizar el cálculo del presupuesto del proyecto se han tomado en consideración diferentes aspectos, en primer lugar los requisitos técnicos y de hardware para montar las GPU, por cuestiones de importancia, se expone el coste total y únicamente desglosadas las tarjetas:

Elemento	Coste unitario	Nº Unidades	Total
Equipo de desarrllo y evaluación	2.876,00 €	1	2.876,00 €
Nvidia Tesla c2050	2.350,00 €	2	4.700,00 €
Asus GTX 470	340,00 €	2	680,00 €
Ordenador personal para desarrollo	950,00 €	1	950,00 €
Coste total			**9.206,00 €**

Tabla 32: Presupuesto Hardware

Tras los costes hardware, se pasa a tener en cuenta los recursos de software utilizados como se detallan a continuación:

Elemento	Coste unitario	Nº Unidades	Total
Licencia Windows 7	164,00 €	1	164,00 €
Licencia Microsoft Office 2010	199,00 €	1	199,00 €
SDK CUDA	#	1	#
Software libre	#	1	#
Coste total			**363,00 €**

Tabla 33: Presupuesto Software

El tiempo de vida estimado de los recursos hardware, a donde los recursos software están adscritos, es de 30 meses de duración, por lo que se ajustan los costes amortizados, en base a la duración del proyecto, en este caso de 8 meses:

Elemento	Coste completo	Coste mensual	Coste amortizado
Amortización HW	9.206,00 €	306,87 €	2454,94 €
Amortización SW	363,00 €	12, 10 €	96,80 €
Coste total			**2.551,74 €**

Tabla 34: Presupuestoamortizado

Por último, se consideran los gastos de personal con el desglose de horas invertidas y el coste por hora:

Elemento	Coste hora	Coste mes	Horas totales	Coste proyecto
Becario	5 €	400,00 €	640	3.200,00 €
Personal investigador	11,25 €	1.800,00 €	400	4.500,00 €
Coste total				**7.700,00 €**

Tabla 35: Presupuesto personal

Unificando todos los costes desglosado anteriormente, obtenemos como resultado final que el coste de desarrollo del proyecto ha sido:

Elemento	Total
Hardware	2.454,94 €
Software	96,80 €
Personal	7.700,00 €
Coste total	**10.251,74 €**

Tabla 36: Presupuesto final

Dado que el proyecto ha sido desarrollado por la Universidad Carlos III de Madrid con fines de investigación y sin carácter comercial, no se aplican porcentajes extra por riesgos y beneficios, junto con los costes computados de facturación de personal. Quedando como resultado el coste real de las horas y los recursos.

BIBLIOGRAFÍA

[1] Ernesto Liria, Javier Garcia Blas, FlorinIsaila, Mónica Abella and Manuel Desco. *Exploiting parallelism of a multi GPU-based reconstruction algorithm for X-ray tomography. ISPA 2012*

[2] M. Abella, J. Vaquero, A. Sisniega, J. Pascau, A. Udias, V. Garcia,I. Vidal, and M. Desco. *Software Architecture for Multi-Bed FDK-based Reconstruction in X-ray CT Scanners. Computer methods and programs in biomedicine*, in press, 2011.

[3] C. T. Badea, M. Drangova, D. W. Holdsworth, and G. A. Johnson. *In vivo small-animal imaging using micro-CT and digital subtraction angiography.* Physics in Medicine and Biology, 53(19):R319, 2008.

[4] L. A. Feldkamp, L. C. Davis, and J. W. Kress. *Practical cone-beam algorithm*. J. Opt. Soc. Am. A, 1(6):612–619, Jun 1984.

[5] T. Rodet, F. Noo, and M. Defrise. *The cone-beam algorithm of Feldkamp, Davis, and Kress preserves oblique line integrals*. MedPhys, 31:1972–1975, July 2004.

[6] J. Vaquero, S. Redondo, E. Lage, M. Abella, A. Sisniega, G. Tapias,M. Montenegro, and M. Desco. *Assessment of a New High-Performance Small-Animal X-Ray Tomograph. IEEE Transactions on Nuclear Science*, 55(3):898 –905, june 2008.

[7] B. Wang, L. Zhu, K. Jia, and J. Zheng. *Accelerated cone beam CT reconstruction based on OpenCL*. In 2010 International Conference on Image Analysis and Signal Processing (IASP), pages 291 –295, april 2010.

[8] F. Xu and K. Mueller. *Real-time 3D computed tomographic reconstruction using commodity graphics hardware.* Physics in Medicine and Biology, 52(12):3405, 2007.

[9] X. Zhao, J.-J. Hu, and P. Zhang. *GPU-based 3D cone-beam CT image reconstruction for large data volume*. Journal of Biomedical Imaging, 2009:8:1–8:8, January 2009.

[10] Basu, S., *O(N2 log2N) Filtered Backprojection Reconstruction Algorithm for Tomography.* IEEE Trans. ima. proc., 2000. 9(10).

[11] Pipatsrisawat, T., et al., *Performance analysis of the filtered backprojection image reconstruction algorithms.* IEEE International Conference on Acoustics, Speech, and Signal Processing Proceedings, 2005. 5: pp 153-6.

[12] Aggarwal, P. and R. Mehra, *High Speed CT Image Reconstruction using FPGA.* Int J Computer Applications, 2011. 22(4): pp 7-10.

[13] Stsepankou, D., K. Kornmesser, and J. Hesser, *FPGA acceleration of cone-beam reconstruction for the X-ray CT.* Proceedings of the IEEE International Conference on Field-Programmable Technology, 2004: pp 327-330.

[14] Que, Z., et al., *Implementing Medical CT algorithms on Stand-alone FPGA based systems using an efficient workflow with Sysgen and Simulink.* Proceedings of the IEEE International Conference on Computer and Information Technology, 2010: pp 2391-2396.

[15] Brasse, D., et al., *Towards an inline reconstruction architecture for micro-CT systems.* Phys Med Biol, 2005. 50: pp 5799-811.

[16] Li, J.C., C. Papachristou, and R. Shekhar, *An FPGA-based computing platform for real-time 3D medical imaging and its application to cone-beam CT reconstruction.* Imag Sci Tech, 2005. 49(3): pp 237-245.

[17] Knaup, M., S. Steckmann, and M. Kachelriess, *GPU-Based Parallel-Beam and Cone-Beam Forward- and Backprojection using CUDA*, in *IEEE Nuclear science Symposium Conference Record*. 2008. p. 5153-7.

[18] Okitsu, Y., F. Ino, and K. Hagihara, *Accelerating cone beam reconstruction using the CUDA-enabled GPU*, in *Proceedings of the 15th international conference on High performance computing*. 2008, Springer-Verlag: Bangalore, India. p. 108-119.

[19] Scherl, H., et al., *Fast GPU-based CT reconstruction using the common unified device architecture (CUDA)*, in *IEEE Nucl. Sci. Symp. Conf. Rec.* 2007. p. 4464-4466.

[20] Schiwietz, T., et al., A Fast And High-Quality Cone Beam Reconstruction Pipeline Using The GPU. Proc. SPIE Medical Imaging, 2007. 6510: pp 65105H.

[21] Riabkov, D., et al., *Accelerated cone-beam backprojection using GPU-CPU hardware,* in *Proc. 9th Int'l Meeting Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine.* 2007. p. 68-71.

APÉNDICE I

Manual de usuario para Mangoose

Requisitosmínimos

Los requisitos mínimos son un procesador de 32 bits y al menos 512MB de memoria RAM.No obstante, la memoria mínima para ciertas reconstrucciones depende del tamaño y la resolución de la imagen,llegando a necesitar 16GB de memoria RAM para volúmenes de 3 camas a la máxima resolución (tamaño de píxel de 50 microns). Para los estudios realizados normalmente, un equipo con 4GB de RAM es suficiente.

Ficheros de entrada

El programa necesita 2 ficheros para funcionar:

1. Fichero de calibración de unidades *Hounsfield*

El fichero de calibración de unidades *Hounsfield* contiene los datos para realizar la conversión de las unidades internas a los números utilizados por la tomografía computerizada en cada una de las calibraciones realizadas en los haces espectrales. Como resultado, los números obtenidos por la tomografía son una conversión lineal de los valores originales.

Cada línea en el fichero contiene 3 campos ASCII diferentes, la energía del haz (float), el valor del desplazamiento (float) y la pendiente del ajuste lineal (float).Los diferentes campos dentro de la línea van separados por comas.

Ejemplo: Para una energía en el haz de: 45 kV, desplazamiento: -956.151 y pendiente: 120128.09 los datos dentro del fichero serían:

45.0:-956.151:120128.09

2. Uno o más ficheros de proyecciones que contengan las imágenes obtenidas cubriendo los 360 ángulos, satisfaciendo los siguientes requisitos:

- Los nombres de estos ficheros seguirán este esquema:

  base_filemame_N_O.ctf

  Donde*base_filename*es el nombre del estudio, *N* es la posición de la cama actual dentro del rango {0..bed_positions-1}, y O es el fichero actual dentro del rango {0..(num_files/num_beds)-1}.

- Los valores de los píxeles son almacenados consecutivamente y ordenados por filas, los datos son codificados con unsignedint (16 bits).

- El número de proyecciones de cada fichero debe ser múltiplo de 5.

Archivos del programa

- Mangoose.h
- InputHandle.c
- FDK.c
- FDK_CUDA.cu
- Common.c
- Main.c

Funciones

- ***intparse_args (intargc, char *argv[], structcLineParams*parameters)***: Introduce los parámetros de lectura

- ***voidwrite_progress (intprogress)***: Actualización del progreso.

- ***voidwrite_error (int error)***: Escribe una descripción del error en el fichero de salida.

- ***voiddeviation_func(structdevdeviation)***: Arregla los índices de la desviación con los parámetros introducidos.

- ***voidstitch_func (structstitch *)***: Función de pegado para reconstrucciones de 2 camas.

- ***voidstitch_func_alt (structstitch *)***: Función de pegado para reconstrucciones con más de 2 camas para solapes posteriores al realizado la primera vez.

- ***intROI_index_gen ( )***: Creación de los índices VOI.

- ***intread_hounsfield_data( )***: Lee los datos necesarios para la conversion a Hounsfield.

- ***intmangoose( )***: Núcleo de Mangoosepara casos no simétricos

- ***intmangoose_sym( )***: Núcleo de Mangoose para casos simétricos

- ***voidcrearTexturas(intdevice,constsize_t x, constsize_t y)***: Genera las reservas de memoria en la zona de texturas dentro de la GPU

- ***voidactualizar_textDatos(intdevice,float * datos, constsize_t x, constsize_t y, constsize_t slot)***: Actualiza la memoria de texturas con los nuevos datos.

- ***voiddestruirTexturas(intdevice)***: Libera la memoria de texturas anteriormente reservada

- ***extern intrampfilter_gpu_fft(float *projection, structrampf_parparam, FILE * id_log_file)***: Realiza la etapa de filtrado de Mangoose.

- ***void backproject_gpu(int thread, int slot, float * volume, constsize_t x, constsize_t y, constsize_t z, constfloatangle, structbackparparam)***: Realiza la etapa de retroproyección de Mangoose.

Parámetros de entrada

"-f" Ruta de los ficheros de proyecciones (sin el nombre del estudio, sólo la ruta terminada con "\")
"-m" Nombre del caso de estudio
"-w" Factor de la magnificación
"-q" Distancia desde el origen al centro de FOV (SO)
"-h" Fichero con las unidades Hounsfield
"-v" Voltaje inicial (keV)
"-n" número de ficheros ctf
"-p" número de proyecciones por fichero
"-b" número de posiciones de camas
"-g" espaciode interpolación
"-d" número de píxeles en cada proyección {proj_dim_sproj_dim_z}
"-a" Número total de proyecciones (posiciones angulares)
"-e" solape en mm entre 2 camas consecutivas
"-j" Espacio de reconstrucción
"-r" dimensión del ROI en los tres ejes
"-o" desplazamiento del ROI en los tres ejes
"-z" second processor
"-i" ánguloinicial
"-t" sentido de la rotación {-1: sentido antihorario, 1: sentido horario}
"-k" generar resultado en flotantes (0: no, 1: si)
"-x" parámetros de corrección por el desalineamiento entre las camasalpha y betadesv_z

Línea de llamadaparaejecución

mangoose64.exe **-f** path_projection_files**-m** study_name**–w**magnification_factor**-h** housnfield_file**-v** source_voltage**-n** number_ctf_files**–p**projections_per_file**-b** number_beds**-g** projection_binning**-d** proj_dim_sproj_dim_z**-a** num_angles**-e** bed_overlap**-j** image_uimage_binning_vimage_binning_z**-r** roi_size_uroi_size_vroi_size_z**-o** roi_offset_uroi_offset_vroi_offset_z**–I** init_angle**-t** rotation_dir**-x** alpha beta desv_z

APENDICE II

NÚMERO DE HILOS – 1 (tiempos en segundos)

TAM. BLOCK	TAM. PROJ	READ	RAMPFILTER	BACKPROJECT	STAGING	DEVIATION	OVERLOAD	WRITE
32	512	0,422	0,849	3,152	0,410	-	-	0,417
4	512	0,422	1,036	11,444	0,410	-	-	0,641
8	2048	7,991	12,284	9,966	0,410	-	-	0,639
4	2048	8,029	14,958	16,507	0,410	-	-	0,638
16	526-1 cama	0,682	1,622	3,520	0,477	-	-	0,639
4	526-1 cama	0,695	2,053	13,486	0,477	-	-	0,640
16	526-2 camas	1,364	3,241	7,037	0,951	0,571	0,177	1,279
4	526-2 camas	1,383	4,104	26,986	0,954	0,572	0,178	1,277

Tabla 37: Hilos / Bloque (1 Hilo)

NÚMERO DE HILOS – 2 (tiempos en segundos)

TAM. BLOCK	TAM. PROJ	READ	RAMPFILTER	BACKPROJECT	STAGING	DEVIATION	OVERLOAD	WRITE
16	512	0,344	0,823	3,028	0,261	-	-	0,434
4	512	0,351	1,018	11,443	0,261	-		0,435
8	2048	6,416	11,867	9,991396	0,251687	-	-	0,359
4	2048	6,470	14,627	16,473921	0,252061	-	-	0,360
16	526-1 cama	0,603	1,587	3,518537	0,299251	-	-	0,458
4	526-1 cama	0,561	2,014	13,48519	0,291824	-	-	0,364
32	526-2 camas	1,126	3,210	7,560	0,584	0,318	0,178	0,729
4	526-2 camas	1,124	4,278	26,981	0,584	0,307	0,177	0,730

Tabla 38: Hilos / Bloque (2 Hilos)

NÚMERO DE HILOS - 4

TAM. BLOCK	TAM. PROJ	READ	RAMPFILTER	BACKPROJECT	STAGING	DEVIATION	OVERLOAD	WRITE
16	512	0,288	0,815	3,028	0,222	-	-	0,331
4	512	0,298	1,019	11,445	0,241	-		0,261
8	2048	6,187	11,796	9,965	0,236	-	-	0,262
4	2048	6,693	14,551	16,484	0,240	-	-	0,278
16	526-1 cama	0,531	1,583	3,518	0,257	-	-	0,266
4	526-1 cama	0,670	2,030	13,503	0,257	-	-	0,248
16	526-2 camas	1,055	3,175	7,037	0,516	0,257	0,178	0,539
4	526-2	1,102	4,043	26,977	0,559	0,276	0,171	0,538

	camas							

Tabla 39: Hilos / Bloque (4 Hilos)

NÚMERO DE HILOS - 8

TAM. BLOCK	TAM. PROJ	READ	RAMPFILTER	BACKPROJECT	STAGING	DEVIATION	OVERLOAD	WRITE
16	512	0,283	0,841	3,033	0,240	-	-	0,220
4	512	0,293	1,018	11,451	0,242	-		0,270
8	2048	6,245	11,923	9,955	0,249	-	-	0,227
4	2048	6,437	14,614	16,543	0,236	-	-	0,221
16	526-1 cama	0,529	1,603	3,522	0,275	-	-	0,285
4	526-1 cama	0,544	2,028	13,480	0,282	-	-	0,261
32	526-2 camas	1,280	3,310	7,566	0,597	0,495	0,197	0,553
4	526-2 camas	1,445	4,130	27,019	0,549	0,260	0,185	0,599

Tabla 40: Hilos / Bloque (8 Hilos)

NÚMERO DE HILOS - 16

TAM. BLOCK	TAM. PROJ	READ	RAMPFILTER	BACKPROJECT	STAGING	DEVIATION	OVERLOAD	WRITE
16	512	0,470	0,980	3,031	0,235	-	-	0,303
4	512	0,607	1,280	11,449	0,270	-		0,323
8	2048	6,624	12,152	9,959	0,279	-	-	0,313
4	2048	11,61	15,018	16,506	0,244	-	-	0,244
16	526-1 cama	0,707	1,758	3,522	0,282	-	-	0,602
4	526-1 cama	1,001	2,444	13,484	0,278	-	-	0,608

16	526-2 camas	1,506	3,405	7,046	0,572	0,836	0,224	0,691
4	526-2 camas	1,957	4,530	26,995	0,542	0,532	0,183	0,689

Tabla 41: Hilos / Bloque (16 Hilos)