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

Ernesto Liria Javier García Blas Florin Isaila
Dept. de Informática
Univ. Carlos III de Madrid
28911 Madrid
{eliria,fjblas,florin}@arcos.inf.uc3m.es

Mónica Abella Manuel Desco
Unidad de Medicina y Cirugía Experimental
Hospital General Universitario Gregorio Marañón
28007 Madrid
{mabella,mdesco}@mce.hggm.es

Resumen— **La mayoría de los tomógrafos para pequeño animal están basados en geometría *cone-beam* con un detector plano que orbita en trayectoria circular. La reconstrucción en estos sistemas se suele hacer con el método basado en el algoritmo propuesto por Feldkamp, Davis y Kress (FDK). El aumento de velocidad en la reconstrucción para tomografía de rayos X (TAC) es un requisito para la extensión de su aplicación clínica. En este artículo se presenta una implementación eficiente de un algoritmo de reconstrucción modular basado en FDK, que aprovecha las posibilidades de cómputo paralelo y la eficiente interpolación bilineal que ofrecen las unidades de procesamiento gráfico (GPU). La implementación propuesta del algoritmo es evaluada por un micro-TAC de alta resolución alcanzando un ratio de aceleración de x46 frente a una implementación inicial basada en CPU, manteniendo en todo momento la calidad de la reconstrucción.**

I. Introducción

Muchos escáneres tomográficos de rayos X para pequeño animal están basados en una geometría *cone-beam* con un detector plano orbitando en trayectoria circular [1]. Esta configuración presenta ventajas sobre otras alternativas utilizadas en aplicaciones clínicas y preclínicas: reducción del tiempo de adquisición, amplio campo de visión axial sin distorsiones geométricas y optimización en la dosis de radiación. A pesar de la existencia de progresos destacables en los algoritmos de reconstrucción estadísticos, los métodos de aproximación basados en el algoritmo propuesto por Feldkamp, Davis y Kress [2] siguen siendo extensamente utilizados para realizar las tareas de reconstrucción 3D debido a su sencilla implementación y eficiencia computacional [1]. Este algoritmo es la extensión de la retroproyección filtrada por los factores de corrección de la geometría *cone-beam*, que incorporan la longitud de los rayos.

Con la evolución de la tecnología, el tiempo de adquisición de proyecciones ha sido reducido. Por otro lado, la evolución de los paneles detectores ha conllevado el incremento de la densidad de elementos detectores, y por lo tanto, una mayor cantidad de datos a procesar [3]. Junto con el incremento de datos, existe la necesidad de realizar reconstrucciones en un menor tiempo para tratar los nuevos usos de la tomografía computerizada: planificación y monitorización en radioterapia, imágenes de asistencia quirúrgica y otras modalidades clínicas que requieren el uso de imágenes en tiempo real [4]. Sin embargo, los recientes avances en los algoritmos aún no han explotado todo el potencial en implementaciones de alto rendimiento, lo que representa una barrera para extender el uso de esta tecnología [4].

Una posibilidad para acelerar el proceso de reconstrucción es usar algoritmos alternativos. Estos algoritmos podrían ser clasificados en tres grupos. El primero de ellos está basado en transformar los datos de las proyecciones del dominio de Fourier a una red cartesiana para ser capaz de usar FFT directamente. El segundo grupo está basado en acelerar la etapa de retroproyección mediante un proceso recursivo de sumas parciales, tratando todas las proyecciones de manera simultánea. Finalmente, el tercer grupo está basado en dividir la imagen en partes más pequeñas (en el espacio o el dominio de Fourier). Los algoritmos de este último grupo alcanzan una aceleración de 40 veces en la etapa de reconstrucción, aunque la calidad de la imagen se ve degradada y existe falta de generalidad debido a su dependencia con las propiedades de la imagen.

Otra estrategia es la aceleración del algoritmo FDK utilizando técnicas de computación y arquitecturas paralelas. Existen múltiples enfoques, algunos de los cuales son rígidos y costosos como el uso de de *ASIC* (Application-Specific Integrated Circuit) o dispositivos *FPGA* (Field Programmable Gate Array). Actualmente, una de las alternativas más utilizada consiste en explotar el paralelismo inherente a las unidades de procesamiento gráfico de propósi-

to general (GPGPU).

Este artículo presenta una implementación modular, híbrida y optimizada para arquitecturas multi-core/multi-GPU del algoritmo FDK basado en una implementación en C de Mangoose [5].

En este trabajo presentamos una optimización de esta aplicacción, a la que llamaremos Mangoose++. La implementación toma ventaja de las capacidades de computación paralela y la eficiente interpolación bilineal, proporcionada por las GPU, para acelerar las dos fases principales de la implementación algorítmica. Además, la implementación propuesta, gracias a su diseño modular y parametrizable, permite adaptarse a distintas configuraciones y dispositivos.

El resto de este artículo tiene la siguiente estructura. En la Sección II presentamos la implementación inicial (carente de una implemtación modular) de Mangoose. La Sección III aborda la dificultad de aplicar técnicas de paralelización en arquitecturas híbridas basadas en multi-GPU y multi-core. En la Sección IV presentamos el diseño modular y parametrizable de Mangoose++. Los resultados de la evaluación realizada se presenta en la Sección V. Finalmente en la Sección VI presentamos otras soluciones relacionadas con el trabajo propuesto en este artículo.

II. Mangoose

Mangoose es una estrategia propuesta para una reconstrucción multi-cama basada en FDK, con un kernel programado en lenguaje C y una interfaz gráfica de usuario implementada en IDL, que incluye módulos para los procesos de calibración y diferentes algoritmos de corrección [5]. Estos módulos de reconstrucción están incluídos en la arquitectura software TAC. El sistema permite obtener resultados como los que se muestran en la Figura 1. En el nivel más alto la aplicación principal del TAC, programada en IDL, incluye la interfaz de usuario y controla los módulos de adquisición, calibración y reconstrucción. La adquisición de datos es realizada por un procesador basado en Linux que controla el hardware del sistema TAC. La adquisición de datos es lanzada por la aplicación principal del TAC despúes de que el usuario seleccione los parámetros de adquisición, como la energía de los rayos X o el flujo, la posición del caso de estudio o el tamaño de punto de la proyección.

El proceso de reconstrucción 3D se realiza a través de Mangoose. Mangoose puede ser vista como una aplicación compuesta de seis etapas: lectura (R), reparto de las proyecciones 2D entre las unidades de procesamiento paralelas (S), filtrado de las proyecciones 2D (F), retroproyección de volúmenes parciales 3D (B), unión de volúmenes parciales en un volumen final (G) y escritura (W). Las fases F y B son altamente paralelizables debido a la falta de dependencias y el alto número de operaciones paralelas con los datos. Aunque el reto principal es en mayor medida el explotar el máximo paralelismo disponible tanto en niveles de GPU como CPU. Nuestro enfoque está basado en una descomposición jerárquica, una planificación dinámica y el solape en la computación con la comunicación.

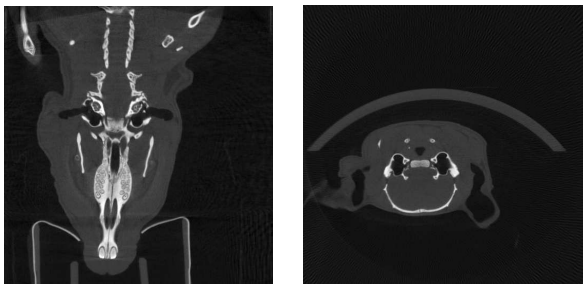

Fig. 1. Dos rodajas (vistas sagital y axial) que corresponden a un experimento con ratones.

III. Paralelización en sistemas híbridos multi-GPU

Uno de los principales desafíos a la hora de utilizar múltiples dispositivos GPU heterogéneos es usar de manera eficiente los diversos tipos de paralelismo disponibles. Estos tipos incluyen el paralelismo interno en GPU, solapando las transferencias CPU-GPU con computación de GPU y CPU, y paralelismo externo entre los diferentes tipos de GPU.

Primero, el paralelismo interno de GPU incluye paralelismo a nivel de hilo, a nivel de instrucción y a nivel de memoria. Optimizar uno de los tipos de paralelismo no optimiza necesariamente todo el rendimiento del programa. Por ejemplo, el medidor de ocupación refleja solo el grado en el nivel de paralelismo de hilo, pero no el de memoria o de instrucción. Incrementar el nivel de ocupación no mejora el rendimiento global si los otros niveles de paralelismo son afectados negativamente.

Usar de manera eficiente una GPU es un reto debido al hecho de que el modelo de programación de CUDA [6] requiere que el desarrollador tenga en consideración las características del hardware con el fin de mejorar tanto los niveles de paralelismo como la localidad en memoria. Para optimizar el paralelismo a nivel de hilo, el desarrollador tiene que elegir cuidadosamente parámetros como el número de hilos o el tamaño de bloque de hilos. Para mejorar el paralelismo a nivel de instrucción se necesita utilizar un número lo suficientemente grande, aunque limitado, de *warps*. Para alcanzar un alto grado de paralelismo a nivel de memoria, los accesos a memoria tanto *inter-warp* como *intra-warp* deben estar considerados. Además, cuando se está mejorando la localidad en memoria, el desarrollador tiene que tener en cuenta las restricciones en el número de registros, memo-

ria compartida y constante que utilizará cada bloque de hilos.

Segundo, solapar transferencias CPU-GPU con cómputo CPU y GPU requiere centrarse tanto en las estructuras dinámicas de cómputo de CPU y GPU como en el uso eficiente de los mecanismos de transferencia asíncrona ofrecidos por las actuales GPU.

Tercero, respecto al paralelismo externo, un sistema compuesto de varias GPUs heterogéneas requiere fraccionar el problema y utilizar una planificación dinámica para maximizar la utilización de todas las GPUs y minimizar el tiempo de inactividad de cada dispositivo. En una configuración típica de un nodo muli-GPU, varias GPUs comparten el bus PCI-e, siendo las transferencias un factor crítico para muchas de las aplicaciones actuales. Una utilización eficiente de esta arquitectura requiere planificar las transferencias CPU-GPU e inter-GPU para lograr dos objetivos: a) reducir la contención del bus PCI-e, b) incrementar en la media de lo posible el solape entre comunicación y computación.

Alcanzar la máxima eficiencia posible utilizando todos estos tipos de paralelismo resulta complicado por la dificultad de alcanzar el balance de carga y la alta utilización de recursos.

Finalmente, uno de los principales retos viene del hecho de que la mayoría de las optimizaciones en GPU no son aditivas, es decir, hay efectos no lineales que complican la inferencia de las mejores combinaciones de optimizaciones para ser utilizadas en la aplicación.

IV. Diseño e implementación

En esta sección presentamos una vista general de Mangoose y del diseño y la implementación de la versión mejorada Mangoose++. El objetivo principal es mejorar la flexibilidad en la configuración de realización y escalabilidad de la aplicación, mientras que la aplicación alcance un alto grado de utilización de los recursos hardware disponibles.

A. Vista general del diseño modular

El objetivo principal en la elaboración del diseño consiste en lograr una arquitectura modular parametrizada que permita un uso óptimo del espacio de optimizaciones. Esto es debido a que existen gran cantidad de parámetros configurables, los cuales influyen enormemente en el rendimiento de la aplicación. Explorar el espacio de optimizaciones es un requisito fundamental para adaptar la aplicación para los actuales y futuros dispositivos hardware y aceleradores.

La arquitectura de Mangoose consiste principalmente en cuatro fases: lectura (R), filtrado (F), retroproyección (B) y escritura (W). La Figura 2 muestra la implementación de Mangoose++ junto con el flujo de datos a través de los diferentes canales de memoria. Estas fases pueden ser parcialmente solapadas (paralelismo funcional) debido a la alta simultaneidad disponible en cada una de las fases.

El desafío principal consiste en desplegar la aplicación de una manera eficiente sobre una arquitectura, proporcionando a su vez un alto nivel de paralelismo. En la siguiente subsección debatiremos la estrategia de paralelización empleada.

B. Estrategia de paralelización

Una de las metodologías de paralelización más populares consiste en seguir cuatro etapas: descomposición, asignación, orquestación y mapeo.

Mientras que esta metodología puede ser aplicada directamente para un problema funcional dado, Mangoose requiere un enfoque más complejo motivado por cuatro razones. Primero, Mangoose consiste en cuatro módulos funcionales principales, que pueden requerir diferentes granularidades en la paralelización. Segundo, el desarrollo de dependencias entre módulos muestra efectos no lineales como mostramos en la Sección V. Tercero, el objetivo principal es mapear la aplicación sobre una arquitectura híbrida con una compleja jerarquía de memoria. Cuarto, hay un número de optimizaciones que pueden ser aplicadas en la interfaz entre distintas unidades funcionales.

La implementación eleva el grado de paralelismo tanto a nivel de aplicación como a nivel de hardware. La fase de lectura toma proyecciones en 2D de un fichero en memoria principal. En nuestra implementación, la granularidad de la fase de lectura puede variar desde una proyección 2D hasta el total del número de proyecciones. La fase de lectura del fichero está implementada utilizando operaciones asíncronas (*aio_read*), solapando la lectura con la fase de filtrado. La cantidad máxima de datos que pueden ser leídos de una vez viene limitada por el tamaño de ventana de la lectura adelantada, siendo éste un parámetro configurable. Dos de las cuestiones en el proceso de optimización son cúando es preferible utilizar operaciones síncronas o asíncronas y cómo elegir el valor óptimo para la granularidad y el tamaño de ventana en la fase de lectura. Como se verá más adelante, esta decisión no es trivial debido a los efectos no lineales que tienen las optimizaciones en las distintas fases de la aplicación.

En la fase de reparto (S) se transfieren los datos desde memoria principal a la memoria de los dispositivos. La transferencia puede ser solapada con la ejecución del kernel utilizando memoria no paginable. Es un enfoque recomendado por Nvidia para mejorar la transferencia entre el host y los dispositivos. Nuestra implementación proporciona como una opción la posibilidad de usar memoria no paginada en la fase S.

La fase de filtrado (F) procesa de manera iterativa cada una de las de proyecciones 2D. La computación se descompone en bloques de proyecciones, que son

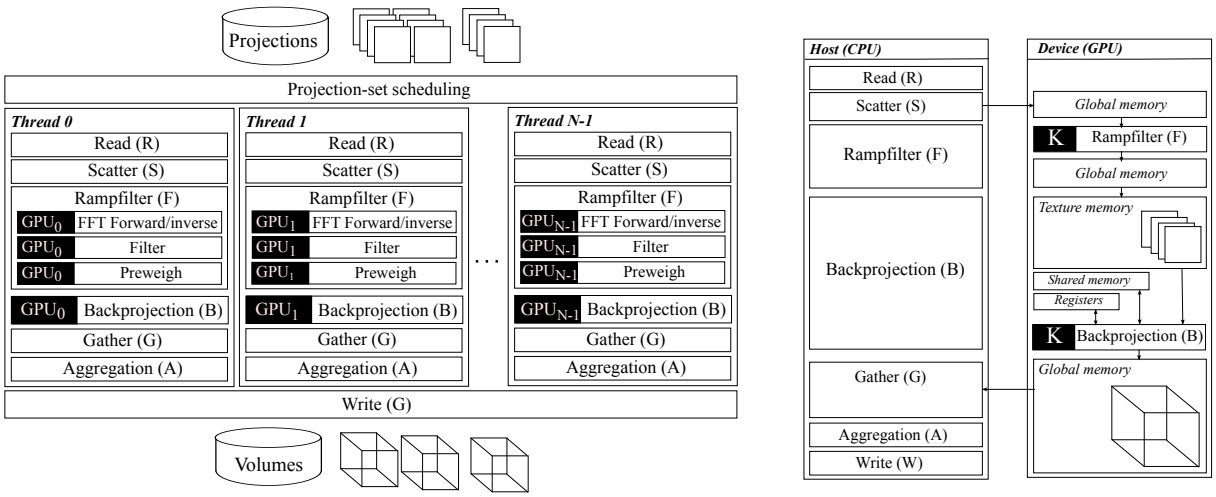

Fig. 2. Diseño de una implementación multi-GPU de Mangoose++. En la parte izquierda de la figura se muestran las fases descompuestas de Mangoose++. Cada GPU es vinculada a un hilo específico de OpenMP, encargado de dirigir los contextos GPU. En la parte derecha se muestran las transferencias de memoria entre el host y los dispositivos GPU.

procesadas en un bucle, cuyas iteraciones son planificadas con OpenMP. El tipo de planificación (estática o dinámica) y la granularidad en la descomposición (que puede variar entre una proyección o el máximo número de proyecciones disponibles), son parámetros configurables. Cada iteración consiste en cuatro pasos sucesivos implementados como kernels de CUDA: 1) FFT directas basadas en CUFFT (biblioteca de las Transformadas Rápidas de Fourier provista por Nvidia [7]), 2) filtrado, 3) FFT inversa basada en CUFFT y 4) cálculo de pesado.

El resultado producido por la fase de filtrado es utilizado como entrada para la fase de retroproyección (B). La entrada de la fase B es llevada a cabo en memoria GPU, es decir, no implica transferencia alguna entre el host y los dispositivos. En la misma tendencia que en la fase F, la fase B es descompuesta en bloques de proyecciones, que son procesados en bucles y cuyas iteraciones son planificadas estáticamente o dinámicamente por OpenMP. Cada iteración consiste en dos pasos sucesivos: la interpolación y la propia retroproyección. La interpolación está basada en un kernel de CUDA aprovechando el hardware de texturas de la GPU para mejorar el rendimiento. Las proyecciones 2D son interpoladas rápidamente, pasándolas en arrays de CUDA vinculados a unidades de textura. La propia retroproyección es un kernel de CUDA, que junta las proyecciones 2D interpoladas en el cálculo de un volumen parcial. Opcionalmente, se optimiza la localidad temporal en los patrones de acceso calculando los volúmenes parciales en submatrices de tamaño configurable almacenadas en memoria compartida en GPU.

La fase de unión (G), transfiere los volúmenes parciales de la memoria del dispositivo a la memoria del host. Como en la fase S, la transferencia puede ser solapada con la ejecución del kernel utilizando memoria no paginada. Además la implementación permite asociar la memoria no paginada del host en memoria del dispositivo, eliminando la necesidad de implementar una operación explícita de copia de memoria.

Finalmente, en la fase de escritura (W) se almacenan los valores resultantes en un fichero. La fase de escritura en fichero está implementada usando operaciones asíncronas (*aio_write*).

La Tabla 1 resume las optimizaciones aportadas en este trabajo. La primera columna muestra si la optimización fue aplicada en un escenario con una o con múltiples GPU. La segunda columna añade una descripción de cada optimización. Las fases optimizadas de la aplicación son mostradas en la tercera columna, el nombre de la optimización viene dado por la cuarta columna. La quinta columna muestra los posibles valores para cada una de las optimizaciones.

V. Evaluación

En esta sección presentaremos la evaluación realizada sobre Mangoose++, la versión optimizada de Mangoose. Los datos capturados han sido adquirido con un subsistema TAC de un equipo ARGUS PET/CT, basado en la geometría cone-beam con una trayectoria circular. A fin de proporcionar tiempos estandarizados, se ha utilizado un estudio de alta resolución (tamaño de punto de 0.05 mm), con 360 proyecciones de 2048x2048 puntos con el objetivo de evaluar el efecto en el rendimiento al manejar grandes volúmenes, y otro estudio compuesto por 360 proyecciones de 512x512 puntos (tamaño de punto de 0.2 mm) para volúmenes más pequeños. Todos los volúmenes reconstruidos tienen una resolución de 512x512x512 puntos, resultando en un fichero de 268 MBytes. El sistema usado para la evaluación

Notación	Descripción	Fases	Número de GPUs	Variables
O_1	E/S asíncrona	Lectura (R), Escritura (W)	Multi-GPU	w_R,g_R
O_2	Memoria fija vs paginada	Reparto (S), Unión (G)	Multi-GPU	activado / desactivado
O_3	Memoria compartida GPU	Retroproyección (B)	1 GPU	tamaño
O_4	Write-combined	Reparto (S), Unión (G)	1 GPU	activado / desactivado
O_5	Tamaño de bloque	Filtrado (F), Retroproyección (B)	1 GPU	$bloque_F$, $bloque_B$
O_6	Desenrollado de bucle	Retroproyección (B)	Una GPU	activado / desactivado
O_7	Transf. de memoria Zero-copy	Unión (G)	Multi-GPU	activado / desactivado
O_8	Planificador OpenMP	Filtrado (F), Retroproyección (B)	Multi-GPU	$grano_F$, $grano_B$

TABLA I
TABLA DE OPTIMIZACIONES DE MANGOOSE++.

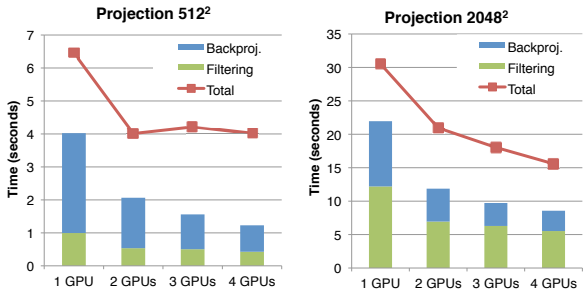

Fig. 3. Desglose de tiempos de Mangoose++ desde 1 hasta 4 GPUs con tamaños de proyección de 512 y 2048 puntos.

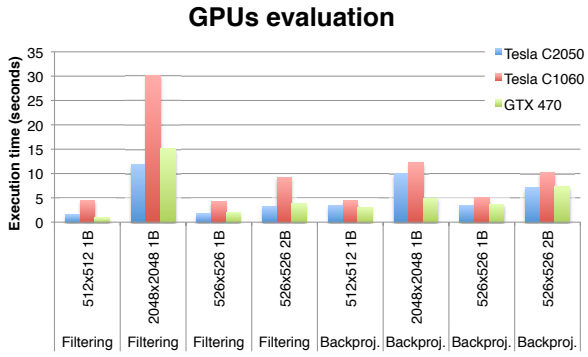

Fig. 4. Tiempos de ejecución para diferentes modelos de GPU de NVIDIA y cuatro tamaños de proyección con una cama (1B) y dos camas (2B).

está equipado con dos Intel Xeon E5640 (procesadores quad-core a 2,67 GHz), 64 GBytes de RAM, dos GPUs NVIDIA Tesla C2050 y dos GPUs ASUS GTX 470.

La Figura 3 compara los tiempos obtenidos para las fases de filtrado y retroproyección junto con el tiempo total de procesamiento de la aplicación. Los resultados sugieren que para volúmenes pequeños el mejor resultado es alcanzado con 2 GPUs. Esto se debe principalmente a coste adicional de trasferencia de datos entre los dispositivos y la CPU. Sin embargo, aumentar el grado de paralelismo para volúmenes de gran tamaño reduce significativamente el tiempo de ejecución.

La Figura 4 muestra los resultados de cuatro tamaños distintos de proyección sobre tres dispositivos GPU distintos. Como se muestra en la figura, la elección del dispositivo es un factor clave en la reducción de tiempos de cómputo.

Finalmente, la Figura 5 muestra los tiempos obtenidos para Mangoose++ de acuerdo a las optimizaciones que han sido aplicadas, mostrando la mejoría desde su etapa inicial (Baseline) al resultado final mediante la utilización de todas las optimizaciones (Full). Como se muestra en la figura, no existe linealidad a la hora de aplicar distintas optimizaciones sobre el código.

VI. TRABAJOS RELACIONADOS

Xu y Mueller [4] 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. 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 [3]. Schiewietz et al. incluyen corrección de los artefactos de anillo y cupping [8]. Riabkov et al. comparan dos implementaciones disintas de retroproyección [9]. Knaup et al. 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 [10]. Noël

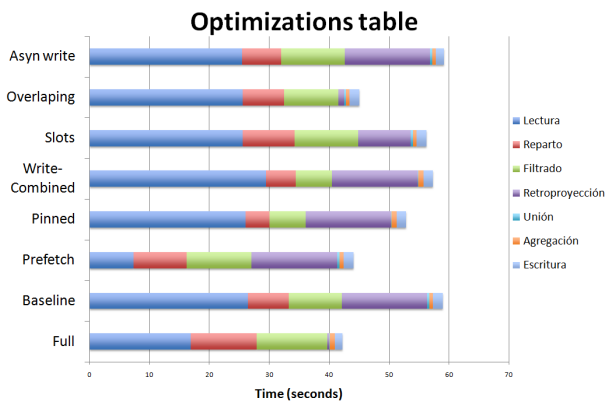

Fig. 5. Tiempos de desglosados de ejecución para distintas optimizaciones sobre un estudio de proyecciones de alta resolución (2048).

et al. aprovechan la disponibilidad de memoria compartida, cargando todas las imágenes proyectadas en la memoria de la GPU y computando la intensidad de cada vóxel por retroproyección en paralelo [11]. Okitsu et al. se centran en la reducción del tiempo de acceso a la memoria externa del dispositivo y en la ocultación de la latencia de memoria [12]. Yan et al. 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 [13]. Sherl et al. se centran en reducir el número de instrucciones y uso de registros [14].

VII. Discusión y conclusiones

En este trabajo hemos presentado una implementación modular, eficiente y heterogénea basada en el algoritmo FDK para TAC de geometría cone-beam. La implementación toma ventaja de las arquitecturas heterogéneas basadas en multi-core y multi-GPU. La principal desventaja de tener diferentes módulos específicos para filtrado y retroproyección es la dificultad de explotar las sinergias existentes entre las distintas fases, ocasionando un incremento significativo en la transferencia de datos entre CPU y GPU. Sin embargo, el enfoque modular permite una implementación eficiente en el reemplazo de algoritmos, facilitando la adaptabilidad de la solución propuesta a las nuevas arquitecturas y futuros dispositivos. Explorar todo el abanico de optimizaciones en cada uno de los módulos es fundamental para alcanzar un tiempo de ejecución óptimo.

Aunque una comparación directa entre las diferentes implementaciones publicadas es difícil debido principalmente a las diferencias en el hardware empleado, hemos encontrado que la implementación propuesta alcanza un ratio de mejora de 4 veces superior frente a otros trabajos publicados recientemente [5], [15].

Agradecimientos

Este trabajo ha sido parcialmente financiado por AMIT Proyecto CDTI CENIT, TEC2007-64731, TEC2008-06715-C02-01, RD07/0014/2009, TRA2009 0175, RECAVA-RETIC, RD09/0077/00087 (Ministerio de Ciencia e Innovación), ARTEMIS S2009/DPI-1802 (Comunidad de Madrid) y TIN2010-16497 (Ministerio de Ciencia e Innovación).

Referencias

[1] 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*, vol. 53, no. 19, pp. R319, 2008.

[2] L. A. Feldkamp, L. C. Davis, and J. W. Kress, "Practical cone-beam algorithm," *J. Opt. Soc. Am. A*, vol. 1, no. 6, pp. 612–619, Jun 1984.

[3] Xing Zhao, Jing-Jing Hu, and Peng Zhang, "GPU-based 3D cone-beam CT image reconstruction for large data volume," *Journal of Biomedical Imaging*, vol. 2009, pp. 8:1–8:8, January 2009.

[4] Fang Xu and Klaus Mueller, "Real-time 3D computed tomographic reconstruction using commodity graphics hardware," *Physics in Medicine and Biology*, vol. 52, no. 12, pp. 3405, 2007.

[5] M. Abella, JJ. 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.

[6] NVIDIA Corporation, *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*, NVIDIA Corporation, 2007.

[7] Nvidia, "CUDA CUFFT Library," 2007.

[8] Thomas Schiwietz, Supratik Bose, Jonathan Maltz, and Rüdiger Westermann, "A fast and high-quality cone beam reconstruction pipeline using the GPU," *Proc. SPIE Medical Imaging*, vol. 6510, pp. 65105H, 2007.

[9] Dmitri Riabkov, Xinwei Xue, Dave Tubbs, and Arvi Cheryauka, "Accelerated cone-beam backprojection using GPU-CPU hardware," *9th Int'l Meeting Fully Three-Dimensional Image Reconstruction in Radiology and Nuclear Medicine*, pp. 68–71, 2007.

[10] Michael Knaup, Sven Steckmann, and Marc Kachelriess, "GPU-based parallel-beam and cone-beam forward- and backprojection using CUDA," in *2008. NSS '08. IEEE Nuclear Science Symposium Conference Record*, oct. 2008, pp. 5153 –5157.

[11] Peter B. Noël, Alan M. Walczak, Jinhui Xu, Jason J. Corso, Kenneth R. Hoffmann, and Sebastian Schafer, "GPU-based cone beam computed tomography," *Computer Methods and Programs in Biomedicine*, vol. 98, no. 3, pp. 271 – 277, 2010, ¡ce:title¿HP-MICCAI 2008¡/ce:title¿.

[12] Yusuke Okitsu, Fumihiko Ino, and Kenichi Hagihara, "Accelerating cone beam reconstruction using the CUDA-enabled GPU," in *Proceedings of the 15th international conference on High performance computing*, Berlin, Heidelberg, 2008, HiPC 08, pp. 108–119, Springer-Verlag.

[13] Yan, Guorui and Tian, Jie and Zhu, Shouping and Dai, Yakang and Qin, Chenghu, "Fast cone-beam CT image reconstruction using GPU hardware," *Journal of Xray Science and Technology*, vol. 16, no. 4, pp. 225–234, 2008.

[14] H. Scherl, B. Keck, M. Kowarschik, and J. Hornegger, "Fast gpu-based ct reconstruction using the common unified device architecture (cuda)," in *Nuclear Science Symposium Conference Record, 2007. NSS '07. IEEE*, 26 2007-nov. 3 2007, vol. 6, pp. 4464–4466.

[15] Bo Wang, Lei Zhu, Kebin Jia, and Jie Zheng, "Accelerated cone beam CT reconstruction based on OpenCL," in *2010 International Conference on Image Analysis and Signal Processing (IASP)*, april 2010, pp. 291 –295.