Implementación modular en GPU de un algoritmo de reconstrucción basado en FDK para tomografía de rayos X

E. Liria[1], J. García[2], M. Abella[1], F. Isaila[2], J. Carretero[2], M. Desco[1,3,4]

[1] Departamento de Bioingeniería e Ingeniería Aeroespacial, Universidad Carlos III de Madrid, España

[2] Área de Arquitectura y Tecnología de Computadores, Universidad Carlos III de Madrid, España

[3] Laboratorio de Imagen Médica, Unidad de Medicina y Cirugía Experimental, Hospital General Universitario Gregorio Marañón, Madrid, España

[4] Centro de investigación en red en salud mental (CIBERSAM), Madrid, España

Resumen

La mayoría de los tomógrafos para pequeño animal están basados en geometría cone-beam con un detector plano orbitando en trayectoria circular. La reconstrucción en estos sistemas se suele hacer con un método basado en el algoritmo propuesto por Feldkamp, Davis y Kress (FDK). El aumento de velocidad en la reconstrucción para tomografía rayos X (TAC) es un requisito fundamental 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 provista en CUDA al usar memoria de texturas que ofrecen las unidades de procesamiento gráfico (GPU). El algoritmo implementado, probado en un micro-TAC de alta resolución, presenta una mejora de velocidad de ejecución de la etapa de retroproyección de un factor 40x respecto a una implementación secuencial de referencia escrita en C, manteniéndose en todo momento la calidad de la reconstrucción.

1. Introducción

La tomografía axial computarizada de rayos X (TAC) es uno de los procedimientos de diagnóstico y evaluación por imagen médica más utilizados [1]. 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 [1]. A este aumento del volumen de datos se le añade el requisito de reconstrucciones más rápidas impuesto por los usos actuales del TAC. 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 [2]. 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 [2]. Esta problemática motiva la necesidad de buscar procedimientos de aceleración que se adecuen a la creciente complejidad y exigencias de la reconstrucción.

Este trabajo presenta una solución al problema citado en el marco del TAC de geometría de haz cónico y trayectoria circular por aplicación del algoritmo de Feldkamp, Davis y Kress (FDK) [3]. 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 filtrado y retroproyección. Por su mayor complejidad algorítmica, es la segunda fase la que consume la mayor parte del tiempo total de procesamiento [1].

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 y tratando todas las proyecciones simultáneamente; el tercer grupo utiliza un enfoque de divide y vencerás, y divide la imagen reconstruida en imágenes menores, en algún caso utilizando para la división la transformada de Fourier de la imagen [4]. Según sus autores, algunos de estos algoritmos llegan a multiplicar por 40 la velocidad de reconstrucción [5, 6], aunque se debate sobre la calidad de las reconstrucciones [4] y su falta de generalidad al depender de las propiedades de la imagen [7].

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 [8].

Actualmente, una alternativa que está usándose con éxito, es la programación sobre unidades de procesamiento gráfico (GPU), la cual permite explotar la clase de paralelismo que se da en la retroproyección [13-15]. Con un coste relativamente bajo y una potencia creciente, resultan ser herramientas casi perfectas para este tipo de tareas [1]. 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.

Este trabajo presenta una implementación basada en multi-GPU del algoritmo FDK, partiendo de la implementación en C, Mangoose [16]. 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.

2. Trabajos relacionados

Xu y Mu eller [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. [1] 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 [17] incluyen corrección de los artefactos de anillo y cupping. Riabkov et al. [18] comparan dos implementaciones disintas de retroproyección. Knaup et al. [13] 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 vóxel por retroproyección en paralelo; Okitsu et al. [14] 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. [15] se centran en reducir el número de instrucciones y uso de registros.

3. Implementación sobre GPU

En esta sección se describirán los detalles de diseño e implementación de los módulos necesarios para versiones en GPU basadas en CUDA [21]. Es importante remarcar que actualmente se cuentan con versiones basadas en cómputo CPU en caso de no contar con dispositivos GPU en el sistema.

La implementación propuesta busca acelerar las etapas computacionalmente más costosas de Mangoose: el filtrado y la retroproyección. Para facilitar el mantenimiento y la mejora de las etapas de forma independiente, se implementan como módulos genéricos, permitiendo la adaptación de la solución a distintas tecnologías (CUDA, OpenCL, multithreads, etc).

En la **¡Error! No se encuentra el origen de la referencia.** se muestra el diagrama de flujo de los procesos de la aplicación de reconstrucción, indicando los componentes comunes y específicos del diseño modular de la solución. Por otro lado, en la **¡Error! No se encuentra el origen de la referencia.** se enumeran los componentes principales del núcleo de reconstrucción basado en GPU para uno o varios dispositivos.

En esta sección se describirán los detalles de diseño e implementación de los módulos necesarios para versiones en GPU basadas en CUDA [21]. Es importante remarcar que actualmente se cuentan con versiones basadas en cómputo CPU en caso de no contar con dispositivos GPU en el sistema.

Para la etapa de filtrado se ha usado CUFFT, la librería de transformadas rápidas de Fourier (FFT), que se suministra de forma gratuita junto con las herramientas de desarrollo

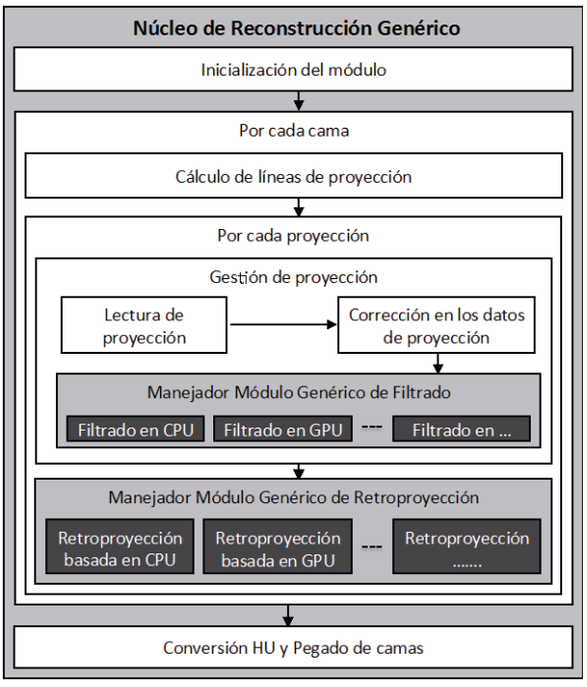

Figura 1. Detalle de la implementación del núcleo de reconstrucción genérico para distintas tecnologías.

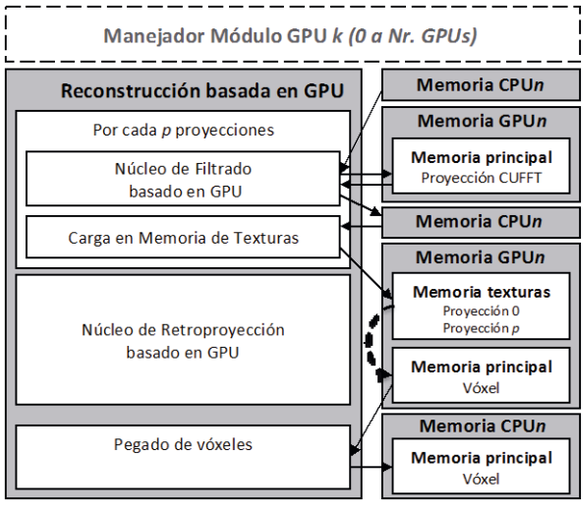

Figura 2. Manejador del módulo genérico basado en múltiples dispositivos GPU. La línea punteada indica un acceso a memoria dentro de la misma GPU.

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 r eales. Es habitual que esta implementación permita mayor rendimiento en velocidad y uso de memoria que el caso más general de transformar de complejos a co mplejos. El filtrado se aplica individualmente sobre cada proyección antes de llamar a la retroproyección.

En lo que respecta a l a retroproyección, se utiliza u n 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.

El volumen entero queda cargado en la memoria del dispositivo de forma que se va actualizando con cada k las sucesivas llamadas a l a 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* y *Z*. 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.

4. Entorno de evaluación

Los datos se han adquirido con el subsistema TAC de un equipo eXplore PET/CT de General Electric Healthcare [23]. Este sistema está basado en una geometría de haz cónico y trayectoria circular, adquiriéndose proyecciones sobre 360° a intervalos de 1°. Las adquisiciones se hicieron con ocho disparos por posición, con un tamaño de proyección de 512×512 y 2048x2048, y tamaño de pixel de 0.2 mm y 0.05 mm respectivamente.

El sistema usado para la evaluación del método está dotado de un Intel Xeon E5640 (dos procesadores de cuatro núcleos a 2.67 GHz), de 64 GiB de RAM, y de dos tarjetas NVIDIA Tesla C2050 y dos tarjetas ASUS GTX 470. Todas las tarjetas evaluadas están conectadas a un bus PCI Express 16x.

La calidad de las reconstrucciones se ha evaluado comparando con una implementación de referencia en C, el reconstructor de Mangoose [16]. Para la comparativa con otros trabajos se presentan solo los tiempos de la parte de retroproyección ya que es éste el dato que da de forma consistente. Para normalizar las medidas, usamos los GUPS (número de actualizaciones de vóxeles necesarias por segundo dividido por 2^{30}) [13].

5. Resultados

Por inspección visual y trazando perfiles no se encuentran diferencias entre la imagen reconstruida por el procedimiento de referencia y la obtenida por la implementación propuesta. La Tabla 1 compara distintas implementaciones de trabajos anteriores.

La Figura 3 muestra los tiempos obtenidos para las etapas de filtrado y reconstrucción, junto al tiempo total de procesamiento. Se observa que para proyecciones pequeñas el mejor resultado obtenido es alcanzado con 2 GPUS. Esto es debido principalmente al coste adicional de transferir datos entre los distintos dispositivos. Sin embargo, para proyecciones de gran tamaño, aumentar el grado de paralelismo ayuda a reducir significativamente el tiempo de ejecución.

Versión	CPU (s)	GPU (s)	Max GUPS
Este trabajo (sec)	131,5	3,0	15,0
Este trabajo (paral)	17,2	0,8	56,2
Riabkov, D., et al [16]	-	12,8	5,0
Yan, G.R., et al [18]	-	5,2	8,6
Knaup,, M., el at [13]	4,0	5,0	13,2

Tabla 1. *Comparación de distintos sistemas de reconstrucción. En este trabajo se muestran los tiempos de reconstrucción secuencial (sec) y paralelo (paral).*

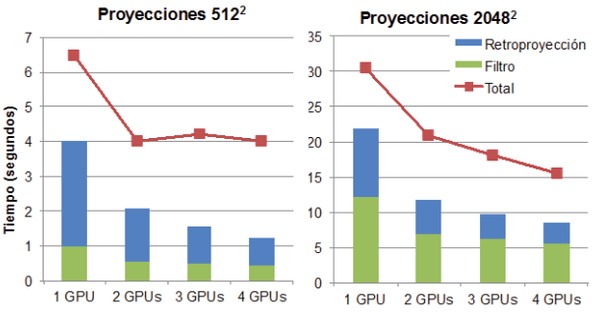

Figura 3. *Resultados de tiempos para las implementaciones de referencia (una CPU) y los casos de 1, 2, 3 y 4 GPUs para 2 tamaños de proyección (512 y 2048).*

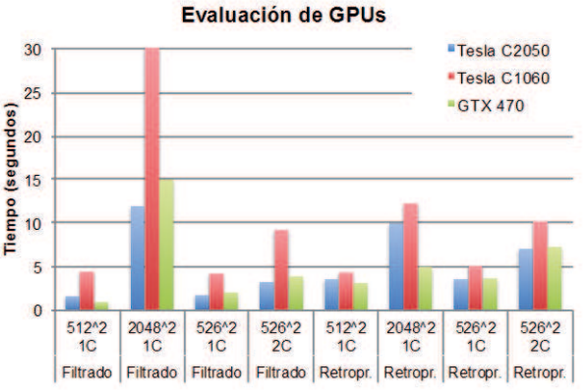

Figura 4. *Resultados de tiempos para las implementaciones sobre distintas GPUs. Comparadas cuatro tamaños de proyección sobre una cama (1C) y dos camas (2C).*

La Figura 4 muestra los resultados de la evaluación de cuatro tamaños de proyecciones sobre tres tarjetas GPU distintas. Como se observa en la figura, la elección de la

tarjeta es un factor clave para reducir tiempos de cómputo.

6. Discusión y conclusiones

Este trabajo presenta una solución al problema de la aceleración de la reconstrucción para FDK. Ofrece una implementación modular de las etapas de filtrado y de retroproyección, que mejora los tiempos en un 25x y 162x respectivamente. Aunque es difícil llevar a ca bo comparaciones directas entre distintas implementaciones debido principalmente a las diferencias entre el hardware [19], podemos decir que la implementación propuesta supone una mejora respecto a t rabajos publicados recientemente de un 4x [13,16,18].

La principal desventaja de hacer filtrado y retroproyección con módulos diferenciados es el no poder aprovechar sinergias entre las distintas etapas, dando lugar, a un aumento significativo de transferencias entre CPU y GPU. Sin embargo, la modularidad permite la sustitución eficiente de la algoritmia implementada, facilitando la adaptabilidad de la solución propuesta a nuevas arquitecturas y dispositivos.

La estrategia escogida consiste en almacenar todo el volumen en memoria principal y cargar sucesivamente las proyecciones. Esta técnica permite reducir significativamente el tiempo de procesado y aumentar la localidad de los datos alojados en memoria.

La necesidad de interpolar sobre los valores de la proyección hace que las proyecciones sean candidatas ideales a ser almacenadas en memoria de texturas. De esta manera se obtienen dos beneficios. Por un lado, el coste de su obtención es prácticamente equivalente a una simple lectura, ahorrándose así, el cálculo de la interpolación. Por otro lado, la caché de acceso a la memoria de texturas acelera el acceso continuado a datos físicamente próximos.

Agradecimientos

Este trabajo ha sido financiado parcialmente por los proyectos AMIT Project del programa 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] Zhao, X., J.-J. Hu, and P. Zhang, GPU-Based 3D Cone-Beam CT Image Reconstruction for Large Data Volume. Int J Biomed Imaging, 2009. 2009: pp 149079.

[2] Xu, F. and K. Mueller, Real-time 3D computed tomographic reconstruction using commodity graphics hardware. Phys Med Biol, 2007. 52(12): pp 3405-3419.

[3] Feldkamp, L.A., L.C. Davis, and J.W. Kress, Practical cone-beam algorithm. J. Opt. Soc. Amer., 1984. 10(6): pp 612–619.

[4] Rodet, T., F . Noo, and M. Defrise, The cone-beam algorithm of Feldkamp, Davis, and Kress preserves oblique line integrals. Med. Phys., 2004. 31(7): pp 1972-1975.

[5] Xiao, S., Y. Bresler, and D.C. Munson, Fast Feldkamp algorithm for cone-beam computer tomography. Proceedings of the 2003 IEEE International Conference on Image Processing, 2003. 2: pp 819-22.

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

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

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

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] Abella, M., et al., Software Architecture for Multi-Bed FDK-based Reconstruction in X-ray CT Scanners. Computer methods and programs in biomedicine, 2011. (in press).

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

[18] 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.

[19] Noël, P.B., et al., GPU-based cone beam computed tomography. Computer methods and programs in biomedicine, 2010. 98: pp 271-277.

[20] Yan, G.R., et al., Fast cone-beam CT image reconstruction using GPU hardware. Journal of X-Ray Science and Technology, 2008. 16(4): pp 225-234.

[21] CUDA CUFFT Library PG-00000-003_V3.1. 2010.

[22] Turbell, H., Cone-beam Reconstruction using Filtered backprojection, in Department of Electrical Engineering. 2001, Linköpings universitet: Linköping, Sweden.

[23] Vaquero, J.J., et al., Assessment of a New Hig h-Performance Small- Animal X-ray Tomograph. IEEE Trans. Nucl. Sci., 2008. 55(3): pp 898-905.