Disponible en www.sciencedirect.com

ScienceDirect

Revista Iberoamericana de Automática e Informática industrial 9 (2012) 144–151

www.elsevier.es/RIAI

Paralelización de los procesos de conformación de haz para imagen ultrasónica con técnicas GPGPU

D. Romero-Laorden*, O. Martínez-Graullera, C.J. Martín-Arguedas, A. Ibañez, L.G. Ullate

Centro de Acústica Aplicada y Evaluación no Destructiva - CAEND (CSIC-UPM), Arganda del Rey, Madrid, España.

Resumen

El proceso de generación de imágenes ultrasónicas mediante técnicas de apertura sintética (SAFT) engloba dos grandes etapas: (1) la etapa de excitación y adquisición, donde se almacenan las señales recibidas por cada elemento o grupo de elementos, y (2) la etapa de conformación de haz o *beamforming*, en la que todas las señales se combinan juntas para obtener los pixeles de la imagen. El uso de hardware gráfico programable (GPUs) puede reducir significativamente el tiempo de cómputo de esta última etapa, debido a que se emplean algoritmos que incluyen diversas funciones como focalización dinámica, filtros paso banda, filtros espaciales o detección de envolvente. Este trabajo estudia la paralelización de los procesos de conformación de haz para imagen ultrasónica y presenta su implementación con técnicas GPGPU (*General Purpose Computation on Graphics Processing Units*). Se estudian asimismo los tiempos de ejecución a partir del número de señales involucradas y las dimensiones de la imagen deseada. Los resultados obtenidos muestran que mediante el uso de la GPU es posible acelerar, en más de un orden de magnitud con respecto a implementaciones equivalentes en CPU, los algoritmos de conformación de haz y de post-procesamiento haciendo posible el desarrollo de sistemas de imagen SAFT en tiempo real.

Palabras Clave:
algoritmos paralelos, procesamiento digital de señales, imagen ultrasónica, conformación de haz.

1. Introducción

El término imagen ultrasónica hace referencia a aquellas imágenes generadas por la composición de las señales de eco obtenidas por las diferencias de impedancia acústica que encuentra una onda mecánica de frecuencia ultrasónica cuando atraviesa un material. El proceso para crear una imagen ultrasónica de barrido sectorial requiere de un conjunto de transductores organizados en un array, que junto con la electrónica de excitación de los mismos, la de recepción y una etapa de procesamiento componen el sistema de imagen ultrasónico y determinan las características finales de la imagen (Steinberg, 1976; Kino, 1987).

El uso de técnicas de apertura sintética (*Synthetic Aperture Focusing Techniques* o SAFT) para reducir el volumen y la complejidad de los sistemas de imagen ha sido un tema profundamente estudiado en diferentes áreas de aplicación como radar (Goodman and Stiles, 2001), sónar (Yen and Carey, 1989) o imagen ultrasónica (Jensen et al., 2006; Corl et al., 1978). Estas técnicas se basan en la activación secuencial, uno a uno, de los elementos del array en emisión y recepción. Una vez se han almacenado todas las señales, se combinan juntas mediante técnicas de conformación de haz para generar una imagen (Lockwood et al., 1998). La complejidad de los algoritmos y el gran número de señales y píxeles involucrados, implican que si se pretende obtener imágenes en tiempo real se precise de una muy alta capacidad computacional.

En los últimos años, una de las tendencias dominantes en arquitectura de microprocesadores ha sido el continuo incremento del paralelismo a nivel de chip. Hoy en día, es común trabajar con CPUs de 2 o 4 cores y no hay duda de que la tendencia hacia el incremento del paralelismo vendrá determinado por chips de varios núcleos. Sin embargo, las unidades de procesamiento gráfico o GPUs (*Graphics Processing Units*) han estado en primera línea con el objetivo de incrementar el nivel de paralelismo en torno al chip y hoy día son procesadores formados por multitud de cores. Como su propio nombre sugiere, las GPUs nacieron con el objetivo de acelerar la repre-

*Autor en correspondencia.
Correos electrónicos: dromerol@caend.upm-csic.es (D. Romero-Laorden), oscarm@caend.upm-csic.es (O. Martínez-Graullera), cjmartin@caend.upm-csic.es (C.J. Martín-Arguedas), aibanez@caend.upm-csic.es (A. Ibañez), luisg@caend.upm-csic.es (L.G. Ullate)

doi:10.1016/j.riai.2012.02.002

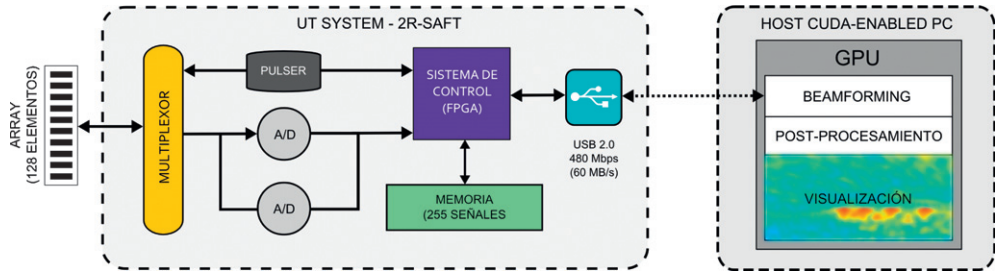

Figura 1: Solución hardware/software propuesta: (izquierda) Sistema 2R-SAFT, (derecha) GPU enabled PC

sentación de gráficos, principalmente en aquellas interfaces de programación como OpenGL o DirectX. Debido al gran paralelismo inherente en los gráficos, las GPUs son máquinas paralelas masivas, y aunque originalmente eran dispositivos diseñados con una función específica, la gran demanda existente para aplicaciones en tiempo real y gráficos tridimensionales les han hecho evolucionar hacia procesadores multicore muy flexibles, altamente paralelos, multihilo, con un alto poder computacional y un gran ancho de banda a memoria. De hecho, las actuales NVIDIA GPUs pueden contener hasta 240 cores por chip (Lindholm et al., 2008) y ser programadas directamente en lenguaje C usando un lenguaje de programación como CUDA o recientemente OpenCL (Nvidia, 2010; Nickolls et al., 2008).

El objetivo principal de este trabajo es presentar un método de conformación de haz para sistemas SAFT basado en técnicas GPGPU con el objetivo de reducir el alto coste asociado a la etapa de post-procesamiento y facilitar su implementación en sistemas en tiempo real. Por tanto, se propone el uso de GPUs como una herramienta fundamental en el proceso y se presenta una solución que mantiene un buen compromiso entre precio y rendimiento.

2. Descripción del Sistema

La solución hardware/software propuesta para el sistema de imagen SAFT en tiempo real consiste principalmente en dos subsistemas, como se muestra en la figura 1:

A) El primer subsistema, representado esquemáticamente a la izquierda de la figura 1, se centra en la excitación y la adquisición de los datos. Esencialmente, el hardware está formado por una serie de componentes como son el multiplexor, pulsers, conversores analógico-digital, FPGA y memorias, necesarios para realizar tanto la excitación de los pulsers y el control del multiplexor, así como la adquisición de la señales y el pre-procesamiento de los datos. En este trabajo se usa el método 2R-SAFT (Martin et al., 2008) como estrategia para la adquisición de los datos, el cuál requiere únicamente un canal en emisión y dos en recepción. Esta técnica utiliza el concepto del coarray para realizar la conformación de haz, cómo se explica en detalle en la sección 3. Todas las señales recibidas son almacenadas para ser posteriormente transferidas a la GPU por medio de una conexión USB 2.0.

B) El segundo subsistema, se ocupa de la etapa de generación de la imagen a partir de los datos adquiridos (derecha de la figura 1). Requiere del uso de una GPU para poder llevar a cabo el pre-procesamiento, la conformación de haz, el post-procesamiento y la posterior visualización de las imágenes. Como se ha comentado con anterioridad, este tipo de algoritmos son computacionalmente intensivos y requieren de una alta capacidad de procesamiento.

3. Concepto del Coarray. Adquisición

El coarray es una herramienta matemática utilizada para estudiar el patrón del haz generado en sistemas pulso-eco. Básicamente se trata de la apertura virtual de un sistema que produce, en un sentido, el mismo patrón del haz que el sistema original trabajando en emisión y recepción.

Supongamos un array lineal con N elementos, siendo a_n los pesos de los transductores. En campo lejano, y asumiendo señales de banda estrecha, el patrón de radiación se puede escribir como

$$f(u) = \sum_{n=0}^{N-1} a_n e^{jkx_n u} = \sum_{n=0}^{N-1} a_n e^{jkndu} = \sum_{n=0}^{N-1} a_n (e^{jkdu})^n \qquad (1)$$

donde $u = sin(\theta)$, y θ es el ángulo medido desde la perpendicular del array. Sustituyendo e^{jkdu} por la variable compleja z, el patrón de radiación se puede expresar como un polinomio, que corresponde con la Transformada-Z de la secuencia a_n. Así, considerando un sistema pulso-eco, el patrón de radiación complejo será el producto de dos polinomios de grado $N-1$

$$f_{total}(z) = Z\{c_n\} = \sum_{n=0}^{2N-2} c_n z^n = \sum_{n=0}^{N-1} a_n z^n \cdot \sum_{n=0}^{N-1} b_n z^n \qquad (2)$$

donde a_n y b_n son las ganancias aplicadas a los transductores en emisión y recepción (actúan como coeficientes de los filtros espaciales), y c_n es el coarray ($Z\{c_n\}$ representa la Tranformada-Z de la secuencia c_n). Así, el patrón de radiación del sistema en onda continua es directamente la DFT (*Discrete Fourier Transform*) del coarray (Steinberg, 1976).

En sistemas de apertura sintética, cada imagen escaneada se obtiene después de una secuencia de disparos de los elementos del array (figura 2). De acuerdo con esto, el coarray puede

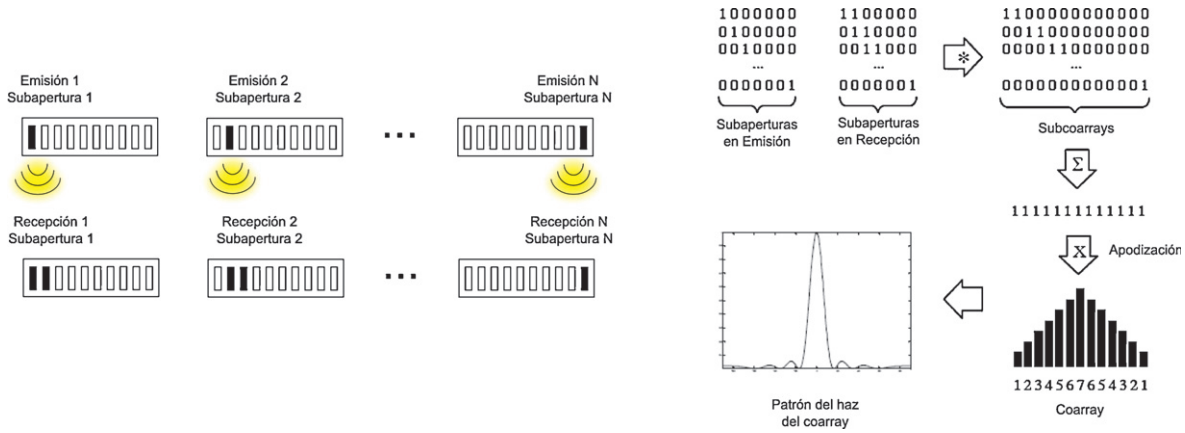

Figura 2: Método 2R-SAFT: (izquierda) secuencias de emisión y recepción de los elementos y (derecha) coarray sintetizado

expresarse como la suma de varios sub-coarrays, cada uno obtenido como la convolución de dos sub-aperturas que representan los pesos de los elementos activos usados para emitir y recibir las señales cada vez.

El método 2R-SAFT (Martin et al., 2008) tiene una serie de ventajas frente a las técnicas SAFT convencionales. El uso de dos transductores en recepción permite eliminar los lóbulos de rejilla causantes de los artefactos característicos de los métodos SAFT con un bajo coste hardware, generando un coarray completo de $2N - 1$ señales y permitiendo obtener imágenes con buen rendimiento, alta resolución lateral y sin lóbulos de rejilla (Martin et al., 2008).

4. Computación paralela

Aunque no es necesario conocer la arquitectura hardware de una GPU para programar sobre ella, si que es apropiado comprender sus principales características para obtener el mayor beneficio posible. Una comparación entre ambas arquitecturas se muestra en la figura 3.

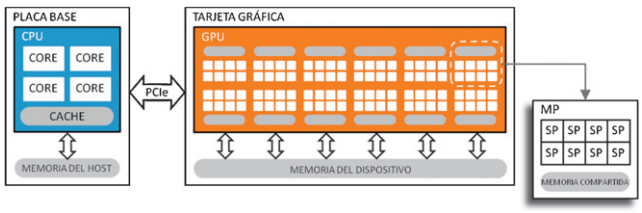

Figura 3: CPU de cuatro-cores frente a GeForce GTX 295 GPU con 240 SPs organizados en 30 MPs

Como se observa, una GPU se compone de una serie de multiprocesadores (MPs), que van desde 1 a 30 en la arquitectura actual de NVIDIA, cada uno de los cuales está formado por 8 procesadores escalares (SPs). Además, cada MP contiene una memoria on-chip (memoria compartida) con muy baja latencia y alto ancho de banda, similar a una cache de primer nivel de una CPU. Tanto la GPU como la CPU gestionan su propia memoria. La primera, llamada memoria del *host* es directamente la memoria RAM del PC mientras que la segunda, llamada memoria del dispositivo o *device*, es la memoria de la tarjeta gráfica compartida por todos los multiprocesadores de la GPU.

El concepto fundamental del modelo de programación CUDA (*Compute Unified Device Arquitecture*) es trabajar con cientos de hilos que ejecutan la misma función pero con diferentes datos. Por tanto, una aplicación se organiza como un programa secuencial en la CPU que incluye funciones especiales, llamadas *kernels*, que son ejecutadas varias veces sobre diferentes datos. El programador organiza los datos a procesar por un *kernel* en estructuras lógicas que consisten en bloques de hilos y rejillas de bloques. Así, una rejilla o *grid* es una estructura 1D, 2D o 3D de bloques y un bloque es una estructura 1D, 2D o 3D de hilos. Cada grupo de *threads* dentro de un bloque pueden cooperar entre ellos por medio de distintos mecanismos de sincronización. En ese sentido, CUDA proporciona a los desarrolladores los medios para ejecutar programas paralelos en la GPU. Una completa descripción sobre la arquitectura y el modelo de programación puede consultarse en la extensa documentación disponible (Nvidia, 2010).

Existen diferentes tipos de memoria disponibles en CUDA definidos para diferentes usos en los *kernels*. Desafortunadamente, una implementación directa de los algoritmos diseñados para CPU a GPU no es normalmente la mejor opción. Es importante enfatizar que las transacciones entre CPU y GPU son lentas, por lo que deben ser minimizadas lo máximo posible para mejorar el rendimiento global. Además, la memoria del dispositivo es limitada por lo que en muchos casos puede ser necesario tener que reducir el volumen de datos que se debe procesar en paralelo. Asimismo, el tiempo de lectura desde memoria global es lento y por tanto se recomienda usar algunos otros mecanismos para acelerar las lecturas, como la memoria de textura (que está cacheada) o la memoria compartida siempre que sea posible. Finalmente, simplificar las operaciones por hilo, minimizar las escrituras a memoria de la GPU y homogeneizar el tiempo de ejecución de todos los hilos puede resultar muy beneficioso en la programación sobre GPUs.

Figura 4: Algoritmo computacional en pseudocódigo. Implementación CPU (izquierda), GPU (derecha) y descripción general de los *kernels* diseñados para cubrir las distintas etapas del algoritmo (abajo)

5. Conformación de haz basada en GPU

La conformación de haz se basa en el retraso y suma de las señales obtenidas desde diversos sensores con el fin de compensar las diferencias en los tiempos de vuelo respecto a un punto dado del espacio. Para un sistema de imagen SAFT esto supone compensar tanto los tiempos de emisión como los de recepción.

Si el proceso se aplica sobre todos los puntos de una región en el interior de un objeto es posible componer una imagen donde se destaque la presencia de fuentes de eco que hacen referencia a discontinuidades en la impedancia acústica del material. Estas discontinuidades se identifican con interfases o defectos en el objeto.

El conformador del haz, junto con el transductor, determina el contraste, la resolución espacial, sensibilidad y precisión del sistema. Los requisitos que se deben cumplir en la formación del haz son: preservar el ancho de banda que contiene toda la información acústica y preservar sin distorsión la información del objeto en la alineación de las ondas.

De acuerdo a la forma en que se actualicen los retardos de focalización estaremos hablando de foco fijo, focalización dinámica o compuesta. La focalización dinámica se consigue adecuando los retardos de focalización a todas las profundidades, obteniéndose la máxima resolución lateral. En el caso de sistemas SAFT ésta puede aplicarse tanto para emisión como recepción.

Las características que hacen ser al proceso de conformación de haz un candidato perfecto para la computación paralela son en primer lugar, el gran volumen de datos (señales adquiridas) así como el número de pixeles involucrados en las imágenes. En segundo lugar, calcular el valor para un pixel determinado de la imagen necesita únicamente realizar una serie de tareas (filtros, focalización dinámica en emisión y recepción, cálculo de envolvente, etc) que afectan a ese pixel independientemente de todos los demás. En ese sentido es factible transformar un algoritmo originalmente diseñado para CPU en uno capaz de ejecutarse en paralelo sobre GPU.

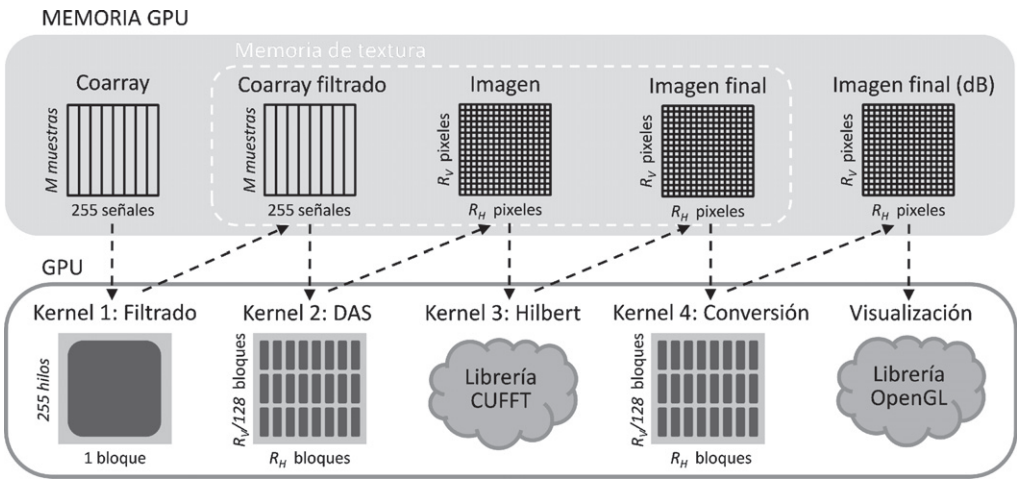

Figura 5: Esquemas de la estrategia seguida para la implementación del conformador de haz en GPU

5.1. *Algoritmo computacional*

La figura 4 muestra el algoritmo computacional utilizado para realizar el proceso de conformación de haz tanto en CPU y GPU (izquierda y derecha respectivamente). A continuación se describe brevemente las distintas etapas que engloba el proceso de conformación de haz:

1. Una vez realizado el proceso de adquisición de las señales, se procede en primer lugar a aplicar un filtro paso banda a cada una de las señales para eliminar los diferentes *off-sets* introducidos durante la etapa de captura y para reducir el ruido, lo que resulta muy beneficioso para las etapas posteriores.
2. A continuación, la segunda etapa se centra en el algoritmo de conformación de haz para generar la imagen, la tarea con más carga computacional y que se compone de varios pasos:
   a) Cálculo de los tiempos de focalización. Aquí, se aplica focalización dinámica en emisión y recepción que pretende aumentar la profundidad del foco sin disminuir la cadencia de imágenes por unidad de tiempo. Si se trata de una imagen estática éstos pueden ser calculados a priori y almacenados en memoria para su uso posterior.
   b) Composición de la imagen. Para ajustar el valor de la muestra a los valores de retardo obtenidos se utiliza una función de interpolación sobre la señal. A continuación, con objeto de limitar los efectos de la difracción, se aplica un filtro espacial (apodización). Se utiliza para mejorar la forma del haz y reducir los lóbulos laterales. Algunos filtros empleados son *blackman-harris*, *triangular*, *hamming* o *box-car* (Smith, 1998).
3. La tercera etapa consiste en calcular la envolvente de la imagen con objeto de suavizar la misma y mejorar la detectabilidad del defecto. Ésta etapa actúa por señal y en esta implementación hace uso de la Transformada de Hilbert.
4. Finalmente, la imagen generada se visualiza directamente por pantalla usando OpenGL.

5.2. *Paralelización*

La implementación de éstas cinco etapas en GPU ha precisado del desarrollo de varios *kernels* paralelizando sobre señales, pixeles o líneas de imagen según sea necesario. En este trabajo se ha diseñado una solución específica para cada etapa del algoritmo involucrado en el proceso de conformación de la imagen, y de ese modo, maximizar la eficiencia en GPU. Lo primero que se hace es copiar el conjunto completo de señales desde la CPU a la GPU. Debido a que esta transacción es lenta, es recomendable copiar todas las señales al mismo tiempo en lugar de ir una por una. Para comprender las estrategias de paralelización empleadas a continuación se describen en profundidad las etapas y los aspectos prácticos de las mismas (figura 5). Asimismo, en la tabla 1 se muestra información acerca de los distintos parámetros de configuración de bloques y *grid*, ocupación de los multiprocesadores, número de registros, etc usados por cada *kernel*.

1. **Primera etapa**: pre-procesamiento, actúa sobre las señales adquiridas (*kernel* 1). La solución que se ha aplicado para resolver el filtrado previo de los datos consiste en lanzar un *thread* de ejecución en GPU por señal almacenada, de tal manera que cada hilo se encarga de filtrar una señal (de un total de 255 ya que se tiene una array de 128 elementos en la implementación actual). Previamente, los datos de las señales se han cargado en memoria de textura así como las posiciones de los elementos emisores y receptores. El tamaño de bloque se ha fijado en 256 de manera que se crea un bloque único que es suficiente para aplicar el filtrado a todas las señales.
2. **Segunda etapa**: el paso más importante es la conformación (*kernel* 2). Debido a que las operaciones por pixel son totalmente independientes se optó por paralelizar a nivel de pixel de la imagen. Para una imagen de dimensiones *tamx* y *tamy* esto significa que un hilo calcula el

Tabla 1: Configuraciones de los distintos kernels para cada estrategia, tamaño de bloque t_{bloque}, número de bloques en x e y n_{bx} y n_{by} respectivamente, dimensiones de la imagen *tamx* y *tamy*, número de registros por kernel n_{reg} y ocupación de los multiprocesadores *ocupacion*

Kernels	Descripción y parámetros de configuración
1	$t_{bloque} = 256$ $n_{bx} = 1$ $n_{by} = 1$ $n_{reg} = 3$ $ocupacion = 100\,\%$
2	$t_{bloque} = 128$ $n_{bx} = ceil(tamx/t_{bloque})$ $n_{by} = tamy$ $n_{reg} = 12$ $ocupacion = 100\,\%$
3	$t_{bloque} = 128$ $n_{bx} = ceil(tamx/t_{bloque})$ $n_{by} = tamy$ $n_{reg} = 5$ $ocupacion = 100\,\%$
4	$t_{bloque} = 128$ $n_{bx} = ceil(tamx/t_{bloque})$ $n_{by} = tamy$ $n_{reg} = 4$ $ocupacion = 100\,\%$

valor correspondiente en ese punto de la imagen lanzando un total de ($tamx \times tamy$) hilos organizados en un *grid* de ($\frac{tamx}{t_{bloque}} \times tamy$) bloques y fijando el tamaño de bloque $t_{bloque} = 128$ hilos. En la figura 6 puede apreciarse un ejemplo acerca de la distribución de bloques e hilos en la rejilla para este caso, donde $(B_0, B_1, \ldots, B_X)$ es el número de bloques creados para tener un hilo por cada pixel en horizontal (*tamx*) y $(B_0, B_1, \ldots, B_Y)$ es el número de bloques creados para tener un hilo por cada pixel en vertical (*tamy*) respectivamente. El cálculo de las lentes se computa dinámicamente para cada pixel cada vez evitando las transacciones de memoria CPU-GPU lo que permite incrementar el rendimiento y adecuar la imagen rápidamente a operaciones de zoom y/o desplazamiento. Posteriormente, el valor de la muestra de señal se obtiene de las señales mediante una función de interpolación y a continuación se aplica un filtro espacial almacenando el resultado en la memoria global de la GPU.

3. **Tercera etapa**: cálculo de la envolvente (*kernel* 3). El tercer paso realiza la transformada de Fourier para cada línea de la imagen operando en el dominio de la frecuencia para multiplicar por el filtro de Hilbert y se realiza la inversa de la transformada de Fourier. Se usan algoritmos de cálculo de transformadas de Fourier optimizados para GPUs (librería CUFFT de NVIDIA). En este caso la paralelización es a nivel de línea.
4. **Cuarta etapa**: conversión (*kernel* 4). Por último se convierte la imagen a decibelios para facilitar su análisis. La parelelización se hace a nivel de pixel siguiendo la misma configuración descrita en la segunda etapa (figura 6).
5. **Quinta etapa**: visualización. Finalmente, la imagen generada se muestra directamente en pantalla usando la librería OpenGL que proporciona funcionalidades específicas para la representación de gráficos integradas con CUDA.

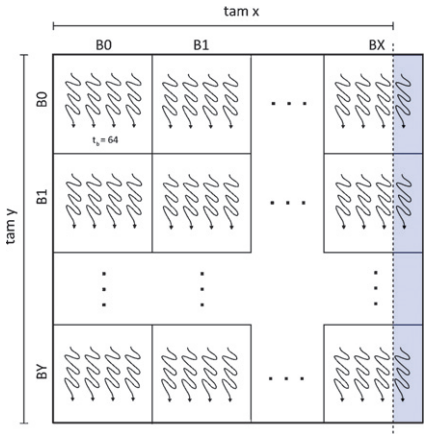

Figura 6: Configuración de bloques elegida para la paralelización por pixel (utilizada en los *kernels* 2, 3 y 4)

6. Resultados

Para testear los algoritmos paralelos desarrollados, se ha utilizado una tarjeta NVIDIA GeForce GTX 295. Esta tarjeta dispone de dos GPUs en su interior, aunque por el momento sólo se ha utilizado una de ellas. Está formada por 240 cores con 1GB de memoria global. Se ha instalado en un PC de 4-cores y procesador Intel Q9450 2.66 GHz con 4GB de RAM. Una evaluación sobre los tiempos de la etapa de reconstrucción de la imagen se ha llevado a cabo usando precisión simple.

La tablas 2 y 3 muestran el desglose de tiempos obtenidos usando la CPU y la GPU respectivamente. Ambas tablas reflejan los tiempos de cómputo medios evaluados tras 10 ejecuciones.

Respecto a los resultados en CPU, se ha realizado una implementación del algoritmo computacional operando con un solo core. Los resultados muestran una progresión lineal a medida que aumenta el número de puntos. Por otro lado, en la tabla de tiempos en GPU las diferencias entre los distintos tamaños de imagen son evidentes a causa de un mayor uso de los recursos del hardware gráfico. Los valores incluyen los tiempos de transferencia de datos. La primera fila muestra el tiempo de adquisición usando una frecuencia de repetición de pulsos de 5000 pulsos/s y de transmisión de 255 señales con 1000 muestras a través de un bus de 20MB/s. En la segunda fila, el filtro paso banda presenta tiempos homogéneos en todos los casos debido a que se aplica siempre sobre el mismo número de señales. Por contra, en la etapa de conformación de haz, que es el núcleo de la etapa de reconstrucción de la imagen (tercera fila), se aprecia un incremento del tiempo de cómputo a medida que la dimensión de la imagen crece. La detección de la envolvente hace uso de FFTs optimizadas para GPUs y normaliza la longitud de la

Tabla 2: Tiempos obtenidos en CPU (1 core) por el sistema de imagen 2R-SAFT

Acciones	Tamaño de la imagen (pixeles)			
	100x100	200x200	300x300	500x500
Adquisición	20 *ms*			
Filtro Paso Banda	6,18 *ms*	6,25 *ms*	6,20 *ms*	6,15 *ms*
Conformación de haz	54,7 *ms*	218,8 *ms*	492,3 *ms*	1367,2 *ms*
Detección de la envolvente	11,25 *ms*	45,92 *ms*	118,86 *ms*	302,5 *ms*
Visualización	0,17 *ms*	0,23 *ms*	0,34 *ms*	0,47 *ms*
Tiempo global conformación de haz	72,3 *ms*	271,2 *ms*	620,7 *ms*	1676,3 *ms*
Frecuencia de trama	10,8 *img/s*	3,43 *img/s*	1,56 *img/s*	0,59 *img/s*

Tabla 3: Tiempos obtenidos en GPU (240 cores) por el sistema de imagen 2R-SAFT

Acciones	Tamaño de la imagen (pixeles)			
	100x100	200x200	300x300	500x500
Adquisición y copia al host	20 *ms*			
Filtro Paso Banda	1,05 *ms*	1,03 *ms*	1,07 *ms*	1,06 *ms*
Conformación de haz	0,69 *ms*	2,52 *ms*	4,47 *ms*	10,92 *ms*
Detección de la envolvente	0,45 *ms*	0,58 *ms*	0,84 *ms*	2,05 *ms*
Visualización	0,07 *ms*	0,10 *ms*	0,13 *ms*	0,20 *ms*
Tiempo global conformación de haz	2,70 *ms*	4,53 *ms*	6,51 *ms*	14,23 *ms*
Frecuencia de trama	44,1 *img/s*	40,7 *img/s*	37,8 *img/s*	29,2 *img/s*

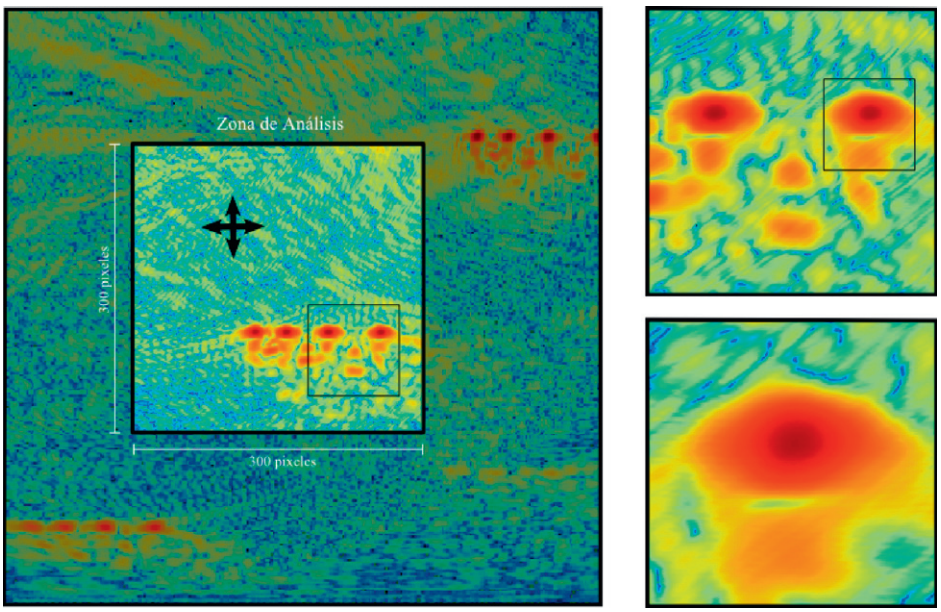

Figura 7: Imágenes obtenidas con el sistema 2R-SAFT. Tamaño de las imágenes: 300x300 pixeles. Cambio dinámico del rango de representación: funciones de zoom y desplazamientos permiten obtener imágenes con más detalle sin pérdida de resolución y sin incrementar el coste computacional. Se aplica focalización dinámica en emisión y recepción

líneas al valor más cercano potencia de dos para incrementar el rendimiento final de este tipo de funciones y por esta razón, los tiempos obtenidos son bastante similares. Finalmente, el tiempo dedicado a visualizar los resultados es muy breve, no siendo necesario copiar la imagen desde la GPU a CPU, porque las librerías OpenGL proporcionan mecanismos para usar directamente la tarjeta gráfica para la visualización de la imagen.

La frecuencia de trama obtenida, suponiendo que la adquisición y la conformación de haz son etapas ejecutadas de manera secuencial, se muestra en la última fila. De otra manera, considerando una ejecución paralela de ambas etapas la frecuencia de trama vendría condicionada por la tarea más lenta (usualmente el tiempo de adquisición) siendo en torno a 50 imágenes por segundo.

Los resultados obtenidos muestran por tanto una mejora considerable en tiempo cuando operamos sobre la GPU. Si se

tienen en cuenta los tiempos globales obtenidos en ambas tablas y se comparan entre sí, es fácilmente apreciable una mejora muy significativa a medida que el tamaño de la imagen crece. En este sentido si las imágenes son grandes las diferencias pueden llegar a ser de varios órdenes de magnitud.

La figura 7 muestra varias capturas de imágenes calculadas en GPU en diferentes instantes de tiempo. Como se puede observar en la parte superior de la figura, la zona activa de análisis es una región de 300x300 pixeles, que es calculada dinámicamente cada vez. Debido a que la lente se calcula de forma dinámica dentro del mismo *kernel* y que se hace uso de la interpolación para obtener el valor de la señal, las operaciones de zoom y desplazamiento en la imagen tienen un coste nulo, presentando las imágenes un aspecto suavizado y preservando las características de la imagen original.

7. Conclusiones

Una de las características de la generación de imagen ultrasónica es que requiere de un alto paralelismo, pues se necesita aplicar un conjunto de algoritmos de procesamiento a una gran cantidad de señales digitalizadas y a un denso volumen de puntos espaciales. En este trabajo se ha explorado el paralelismo de las GPUs, para reducir el tiempo de procesamiento durante la etapa de conformación de la imagen. Utilizando una tarjeta gráfica sencilla equipada con tecnología NVIDIA CUDA, los resultados experimentales muestran que el tiempo de cómputo por pixel (focalización dinámica, interpolación, apodización, envolvente) es 0,53 μs. Esto supone 14 ms para una imagen de 500x500 pixeles. Así, la frecuencia de trama de la etapa de conformación de haz es en este caso de 70 imágenes por segundo. Por tanto, la paralelización en GPUs constituye un excelente método para acelerar la formación de la imagen a alta velocidad y a bajo precio y complejidad, que puede aplicarse a otros muchos casos de estudio. Se ha demostrado que desplazar la tarea de conformación de la imagen ultrasónica desde la CPU a un dispositivo gráfico programable GPU es una muy buena solución para acelerar la conformación de haz en sistemas de imagen SAFT.

English Summary

Beamforming parallelization for ultrasonic imaging using GPGPU techniques.

Abstract

Ultrasonic image generation based on Synthetic Aperture Focusing Techniques (SAFT) can be divided into two stages: (1) the excitation and acquisition stage, where the signals received by each element or group of elements are stored; and (2) the beamforming stage, where the signals are combined together to obtain the image pixels. The use of Graphics Processing Units (GPUs) can significantly reduce the computing time of this last stage, that usually includes different functions such as dynamic focusing, band-pass filtering, spatial filtering or envelope detection. This work studies the parallelization of the beamforming process for ultrasonic imaging and presents its implementation using GPGPU techniques (General Purpose Computation on Graphics Processing Units). We also consider the execution times from the number of signals involved and the desired image dimensions. Experimental results show that using GPU can accelerate, in more than one order of magnitude with respect to CPU implementations, the beamforming and post-processing algorithms making possible the development of real time SAFT imaging systems.

Keywords:

parallel algorithms, digital signal processing, ultrasonic imaging, beamforming.

Agradecimientos

Este trabajo está apoyado por el Ministerio de Ciencia e Innovación de España a través de los proyectos DPI-2007-65408-C02-01, DPI2010-19376 y la beca BES-2008-008675.

Referencias

Corl, P. D., Grant, P. M., Kino, G. S., Septiembre 1978. A digital synthetic focus acoustic imaging system for nde. IEEE Ultrasonic Symposium, 263–268.

Goodman, N. A., Stiles, J. M., Julio 2001. The information content of multiple receive aperture sar systems. Proceedings of IEEE Geoscience and Remote Sensing Symposium 4, 1614–1616.

Jensen, J. A., Nikolov, S. I., Gammelmark, K. L., Pedersen, M. H., Diciembre 2006. Synthetic aperture ultrasound imaging. Ultrasonics 44, e6–e16.

Kino, G. S., Enero 1987. Acoustic Waves: Devices, Imaging, and Analog Signal Processing. Prentice Hall.

Lindholm, E., Nickolls, J., Oberman, S., Montrym, J., Marzo 2008. Nvidia tesla: A unified graphics and computing architecture. IEEE Micro 28 (2), 39–55.

Lockwood, G. R., Talman, J. R., Brunke, S. S., Julio 1998. Real-time 3-d ultrasound imaging using sparse synthetic aperture beamforming. IEEE Transactions on Ultrasonics, Ferroelectrics and Frequency Control 45, 980–988.

Martin, C. J., Martinez, O., Ullate, L. G., Noviembre 2008. Reduction of grating lobes in saft images. IEEE International Ultrasonics Symposium, 721–724.

Nickolls, J., Buck, I., Garland, M., Skadron, K., Abril 2008. Scalable parallel programming with cuda. Queue 6 (2), 40–53.

Nvidia, C., Marzo 2010. NVIDIA CUDA 3.0 Guía de programación.

Smith, S. W., Febrero 1998. Digital Signal Processing. Analog Devices.

Steinberg, B. D., Enero 1976. Principles of Aperture and Array System Design. Wiley-Interscience.

Yen, N. C., Carey, W., Agosto 1989. Application of synthetic-aperture processing to towed-array data. The Journal of the Acoustical Society of America, 158–171.