TECNIACUSTICA Cáceres · 2011

42º CONGRESO ESPAÑOL DE ACÚSTICA
ENCUENTRO IBÉRICO DE ACÚSTICA
EUROPEAN SYMPOSIUM ON ENVIRONMENTAL ACOUSTICS
AND ON BUILDINGS ACOUSTICALLY SUSTAINABLE

PARALELIZACIÓN DE LOS PROCESOS SAFT DE CONFORMACIÓN DE HAZ MEDIANTE GPGPU

PACS: 43.35.Zc

D. Romero-Laorden; O. Martínez-Graullera; C.J. Martín-Arguedas; M. Pérez-López;
L. Gómez-Ullate
Grupo de Evaluación por Ultrasonidos
Centro de Acústica Aplicada y Evaluación No Destructiva (CAEND UPM-CSIC)
Carretera de Campo Real, km 0.200. La Poveda.
Arganda del Rey, Madrid, E-28500 (España)
Tel: (+34) 91 871 19 00
Fax: (+34) 91 871 70 50
E-mail: dromerol@caend.upm-csic.es

ABSTRACT

Synthetic aperture techniques (SAFT) are based on the separate acquisition of all the individual signals involved in the process of generating an ultrasonic image and applying them a beamforming process to focus the image dynamically in emission and reception, getting the highest quality. This paper examines how to perform the parallelization process of beamforming with GPUs presenting practical aspects of its implementation. We analyze the image quality and the computation time depending on the number of signals involved and the dimensions of the image, reducing the generation time in several orders of magnitude and reaching real time.

RESUMEN

Las técnicas de apertura sintética (SAFT) se basan en la adquisición separada de todas las señales individuales que intervienen en el proceso de generación de imagen ultrasónica, aplicando sobre ellas una compensación de los tiempos de propagación para focalizar dinámicamente la imagen en emisión y recepción, obteniendo la mayor calidad posible. Este trabajo analiza cómo llevar a cabo la paralelización de los procesos de conformación de haz con GPUs presentando aspectos prácticos de la implementación. Se analiza la calidad de imagen y el tiempo de cálculo en función del número de señales involucradas y las dimensiones de la imagen, consiguiendo reducir en varios órdenes de magnitud el tiempo de generación y acercarnos al tiempo real.

1. INTRODUCCIÓN

La imagen ultrasónica hace referencia a aquellas imágenes creadas 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 determinado. Para generar una imagen de estas características se requiere el uso de varios elementos: (1) un conjunto de transductores organizados en un array; (2) la electrónica de excitación de los mismos y la de

recepción; y (3) una etapa de procesamiento para producir la imagen. Estos componentes forman parte del sistema de imagen ultrasónico y determinan las características finales de la imagen [1,2].

Una alternativa al empleo de sistemas paralelos phased array son las técnicas de apertura sintética. En sus inicios se plantearon como soluciones para la reducción de la complejidad del sistema de imagen y hoy en día es un tema de estudio en distintas áreas de aplicación como radar [3], sonar [4] o imagen ultrasónica [5, 6]. Se basan en la adquisición separada de todas las señales individuales que intervienen en el proceso de generación de la imagen ultrasónica, sobre las que se aplica posteriormente un proceso de conformación de haz que permite focalizar de forma dinámica la imagen tanto en emisión como en recepción. Esta forma de operar proporciona la máxima calidad posible [3-4] pero es causante también de una serie de desventajas. La complejidad de los algoritmos, el gran número de señales y pixeles involucrados ocasionan irremediablemente un coste computacional elevado y un uso prohibitivo para sistemas en tiempo real.

Recientemente los fabricantes de hardware han dotado a las GPUs con nuevas funcionalidades de programación. Las tarjetas gráficas se han convertido no solo en procesadores gráficos, sino también en pequeñas unidades de computación general que ahora están disponibles para cualquier persona con un PC convencional o un portátil. Aumentan la velocidad y la potencia de cálculo en otros campos de trabajo que difieren de las aplicaciones gráficas tradicionales. Este nuevo paradigma se ha denominado Computación de Propósito General en Unidades de Procesamiento Gráfico, o GPGPU. Hoy en día, las GPUs pueden ser programadas directamente usando diferentes interfaces de programación de aplicaciones o APIs, como CUDA (Computed Unified Device Architecture, estándar propuesto por NVIDIA) [6,7]; u OpenCL [8] un nuevo estándar en la industria que permite un cómputo paralelo heterogéneo entre CPUs y GPUs. De este modo ahora es posible explotar el poder computacional de las GPUs simplemente codificando nuestros algoritmos en código C y ejecutándolos en cientos de hilos de cómputo paralelo dentro de la GPU.

El objetivo de este trabajo ha sido estudiar el uso de hardware gráfico como herramienta eficaz en el desarrollo de sistemas ultrasónicos y concretamente en los procesos de conformación de haz con el objetivo de reducir el alto coste asociado al post-procesamiento y facilitar su implementación en sistemas en tiempo real. Asimismo, proponemos una solución que mantiene un buen compromiso entre precio y rendimiento.

2. CONCEPTO DEL COARRAY

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:

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

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 $N - 1$:

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

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 Transformada-Z de la secuencia c_n). Así, el patrón de radiación del sistema en onda continua es directamente la DFT del coarray [1].

En sistemas de apertura sintética, cada imagen escaneada se obtiene después de una secuencia de disparos de los elementos del array (figura 1). De acuerdo con esto, el coarray puede 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.

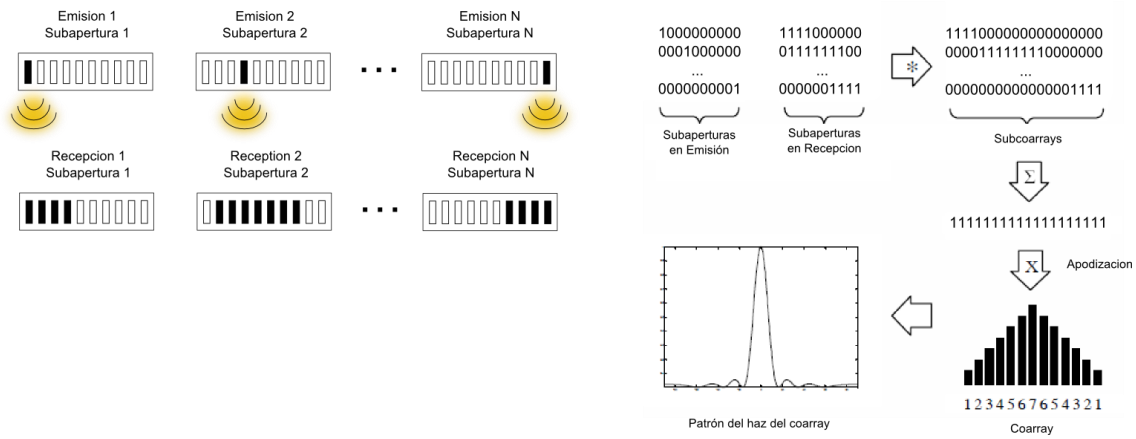

Figura 1. Método SAFT de mínima redundancia: (izquierda) secuencias de emisión y recepción de los elementos y (derecha) coarray sintetizado

En nuestro caso de estudio, se ha elegido el método SAFT de mínima redundancia donde cada elemento del coarray está compuesto por una sola señal. Con un conjunto reducido de canales en paralelo (8 canales integrados en un AD) y una secuencia de disparos donde emite uno de cada 4 elementos (1:4:128), la estrategia de adquisición tiene una serie de ventajas frente a las técnicas SAFT convencionales. El hardware es mínimo acorde a los estándares tecnológicos actuales y el número de disparos necesarios para componer la imagen se ha reducido por cuatro. De esta manera es posible obtener imágenes con buen rendimiento, con alta resolución lateral y carente de lóbulos de rejilla [9].

3. SISTEMA DE IMAGEN

3.1 Adquisición

Un diagrama del sistema de imagen puede verse esquemáticamente representado en la figura 2. El sistema consta de dos partes claramente diferenciadas: el subsistema de adquisición y el subsistema de procesamiento.

Por un lado, el sistema de adquisición se basa en un equipo comercial SITAU (fabricado por Dasel Sistemas, España) que contiene de 128 canales paralelos en emisión y que se comunica con el PC mediante un puerto USB. El sistema SITAU dispone de un procesador que gestiona

el proceso de adquisición y permite programar la adquisición continua de un *Full Matrix Array*. El PC se ocupa de recoger los datos conforme se van produciendo a partir del bus USB 2.0. Se ha empleado un array lineal de 128 elementos con una frecuencia central de $5\,MHz$ (fabricado por Imasonic, Francia) con una separación entre elementos o *pitch* igual a $0.6\,mm$ que aseguran una supresión de los lóbulos de rejilla en aluminio.

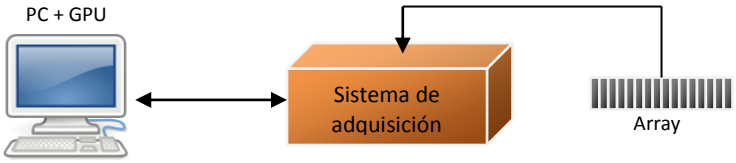

Figura 2. Representación esquemática del sistema de adquisición

3.2 Conformación De Haz Basada En Hardware Gráfico

El resto del sistema se compone fundamentalmente de una GPU que se encarga de copiar las señales adquiridas desde la CPU a la memoria de la tarjeta gráfica y de procesarlas adecuadamente siguiendo el algoritmo encargado de los procesos de conformación de haz. Por último, las imágenes generadas se representan por pantalla. Antes de discutir la implementación especifica de nuestro conformador de haz, es importante comentar brevemente algunos aspectos acerca de la arquitectura de las tarjetas gráficas, su modelo de programación y mostrar algunos detalles prácticos sobre como paralelizar los algoritmos en este tipo de hardware.

3.2.1 Computación paralela en GPU

Aunque no es necesario conocer la arquitectura hardware de una GPU para programar sobre ella, sí que es apropiado comprender sus principales características para obtener el mayor beneficio posible. En esencia, una GPU se compone de una serie de multiprocesadores (MPs) cada uno de los cuales están compuestos por 8 procesadores escalares (SPs). Además, cada MP contiene una pequeña memoria compartida de 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.

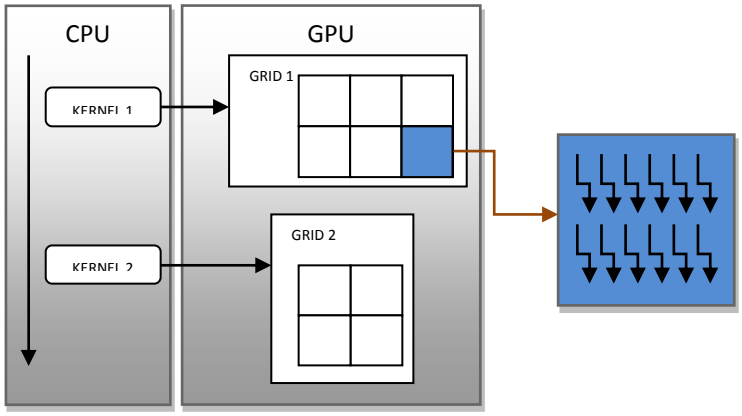

Figura 3. Estructuras lógicas de bloques y rejillas. El código secuencial se ejecuta en la CPU mientras que el código paralelo lo hace sobre la GPU

El concepto fundamental del modelo de programación CUDA 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. Únicamente se puede definir un *grid* dentro de un *kernel*. A modo de ejemplo, la figura 3 muestra la ejecución de un programa escrito en CUDA donde se han programado dos *kernels* (el primero está formado por un *grid* de $[2 \times 3]$ bloques y el segundo de $[2 \times 2]$ bloques de 2x6 hilos respectivamente).

Además, existen diferentes tipos de memoria disponible en CUDA definidas para diferentes usos en las funciones paralelas. 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 esta *cacheada*) o la memoria compartida siempre que sea posible [6]. 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.

De esta manera podemos decir que aquellas funciones que son independientemente ejecutadas muchas veces sobre datos diferentes, como es el caso de los procesos de conformación de haz, son buenos candidatos a ser ejecutados en una GPU. En nuestro caso, las operaciones a realizar son simples: sumas y multiplicaciones con los datos almacenados. El principal problema es encontrar la mejor estrategia para paralelizar los algoritmos y así obtener el máximo beneficio del poder computacional de la GPU. En este contexto, presentamos aquí la solución que hemos encontrado para nuestro sistema basado en la composición de imagen sobre GPU.

3.2.2 Paralelización del conformador

La implementación del conformador en GPU requiere el desarrollo de varios *kernels* paralelizando sobre señales o pixeles según sea necesario. En este trabajo se ha diseñado una solución específica para cada paso del algoritmo para maximizar la eficiencia en GPU. Para comprender las estrategias de paralelización empleadas a continuación se describen en profundidad las etapas y los aspectos prácticos de las mismas (figura 4).

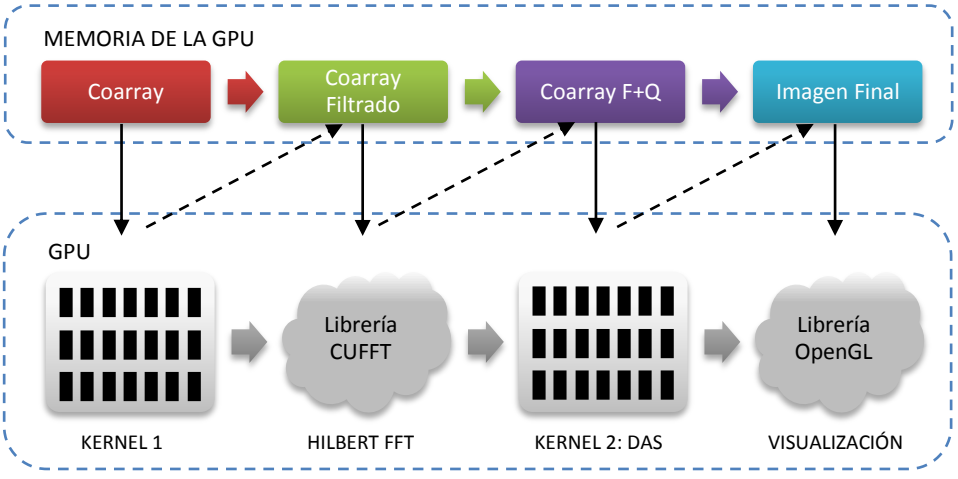

Figura 4. Conformación de haz implementado en GPU

Como se puede ver, lo primero que se hace es copiar el conjunto completo de señales (el coarray) desde la CPU a la GPU. Recordemos que en nuestro ejemplo $\boldsymbol{N\_elementos} = \mathbf{128}$. Como hemos dicho, las transacciones a memoria no son rápidas, por lo que es recomendable copiar todas las señales al mismo tiempo en lugar de ir una por una.

A continuación las señales se filtran para eliminar los diferentes off-sets introducidos durante la etapa de captura y para reducir el ruido que resulta muy beneficioso para las etapas posteriores. Para ello se crea un hilo de ejecución en GPU por señal almacenada (*kernel* 1), de tal manera que cada hilo se encargue de filtrar una señal del coarray (de un total de $\mathbf{2} * \boldsymbol{N\_elementos} - \mathbf{1}$, en el ejemplo que aquí consideramos son 255 señales) organizados en un *grid* de $[\boldsymbol{N\_elementos}/\boldsymbol{T\_bloque} \times \mathbf{1}]$ bloques siendo $\boldsymbol{T\_bloque} = \mathbf{64}$ hilos. Destacar que la estructura de datos que almacena el coarray se ha cargado en memoria de textura para acelerar los accesos a memoria.

Después se aplica la transformada de Hilbert a cada señal filtrada para poder obtener las señales analíticas. En este caso, se usan algoritmos para el cálculo de FFTs optimizados en GPU (librería CUFFT [7]) para calcular la IFFT del producto entre las señales del coarray y el transformador de Hilbert. Con estas librerías no es necesario preocuparse de la estrategia de paralelización puesto que ellas son responsables de adecuar el algoritmo de la mejor manera posible, aunque es recomendable que el tamaño de la señal sea potencia de dos para conseguir la máxima velocidad. Las señales resultantes (F+Q en la figura 4) se almacenan en memoria de textura para la próxima etapa.

El paso más importante es la conformación (*delay and sum*). 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 $\boldsymbol{Dim\_x}$ y D$\boldsymbol{im_z}$ esto significa que un hilo calcula el valor correspondiente en ese punto de la imagen lanzando un total de $\boldsymbol{Dim\_x} \times \boldsymbol{Dim\_z}$ hilos organizados en un *grid* de $[\boldsymbol{Dim\_x}/\boldsymbol{T\_bloque} \times \boldsymbol{Dim\_z}]$ bloques y fijando el tamaño de bloque $\boldsymbol{T\_bloque} = \mathbf{128}$ hilos (*kernel* 2). 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 complejas mediante una función de interpolación (trabajando separadamente con los componentes de fase y cuadratura). Por último, se calcula el modulo del valor complejo resultante obteniendo de ese modo el valor del pixel de la imagen de la envolvente y se almacena en la memoria global de la tarjeta gráfica.

Finalmente, la imagen generada se visualiza por pantalla (puede existir una conversión previa a decibelios si se desea mostrar en ese tipo de escala) usando la librería gráfica OpenGL. Entonces un nuevo conjunto actualizado de señales se copian desde CPU a GPU y el proceso vuelve a empezar.

4. RESULTADOS

Para testear los algoritmos paralelos desarrollados, se ha utilizado una tarjeta NVIDIA GeForce GTX 295 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. Las tablas 1 y 2 muestran respectivamente el desglose de tiempos obtenidos usando la CPU y la GPU para diferentes tamaños de imagen y un conjunto de señales igual a $\mathbf{2} * \mathbf{N\_elementos} - \mathbf{1} = \mathbf{255}$ de longitud $\mathbf{L\_señal} = \mathbf{1024}$ muestras.

Los resultados obtenidos muestran una mejora considerable en tiempo cuando operamos sobre la GPU.

Acciones	**Tamaño de la imagen (pixeles)**			
	100x100	200x200	300x300	500x500
Adquisición	26 ms			
Filtrado del coarray	6,18 ms	6,25 ms	6,20 ms	6,15 ms
Transformada de Hilbert	11,25 ms	45,92 ms	118,86 ms	302,5 ms
Conformación de haz	54,7 ms	218,8 ms	492,3 ms	1367,2 ms
Visualización	0,17 ms	0,23 ms	0,34 ms	0,47 ms
Tiempo global de la generación de imagen	72,3 ms	271,2 ms	620,7 ms	1676,3 ms
Frecuencia de trama	10,2 img/s	3,68 img/s	1,61 img/s	0,59 img/s

Tabla 1. Tiempos de ejecución del conformador en CPU

Acciones	**Tamaño de la imagen (pixeles)**			
	100x100	200x200	300x300	500x500
Adquisición	26 ms			
Filtrado del coarray	1,05 ms	1,03 ms	1,07 ms	1,06 ms
Transformada de Hilbert	0,45 ms	0,58 ms	0,84 ms	2,05 ms
Conformación de haz	0,69 ms	2,52 ms	4,47 ms	10,92 ms
Visualización	0,07 ms	0,10 ms	0,13 ms	0,20 ms
Tiempo global de la generación de imagen	2,70 ms	4,53 ms	6,51 ms	14,23 ms
Tasa de imágenes de la conformación de haz	370 img/s	220 img/s	153 img/s	70 img/s
Frecuencia de trama	35 img/s	32 img/s	30 img/s	25 img/s

Tabla 2. Tiempos de ejecución del conformador en GPU

Como se puede observar, la frecuencia de trama obtenida para una imagen de $\boldsymbol{300 \times 300}$ pixeles es de 30 imágenes por segundo en el caso de la GPU en comparación con las 1,61 imágenes por segundo en el caso de la CPU. En este sentido si las imágenes son grandes las diferencias pueden llegar a ser de varios órdenes de magnitud.

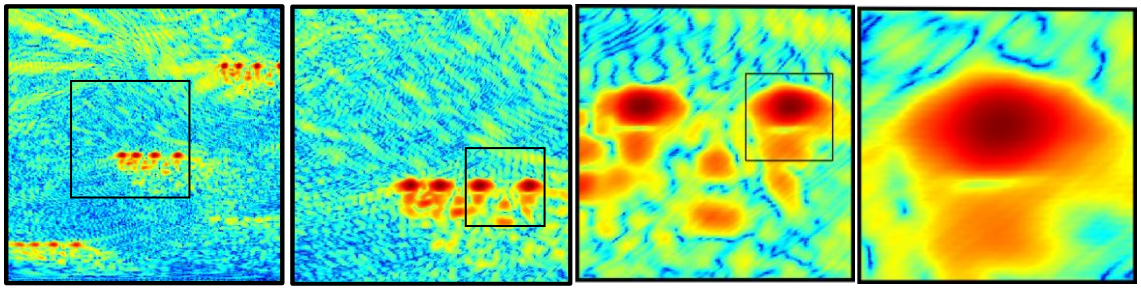

Figura 5. Imágenes obtenidas con 2R-SAFT (300×300 pixeles) en distintos instantes de acercamiento. Rango dinámico $-50\ dB$

Por último, la figura 5 muestra varias capturas de imágenes calculadas en GPU. Las imágenes corresponden a una pieza de aluminio ($\boldsymbol{c = 6.300\ m/s}$) que contiene dos taladros separados $\boldsymbol{2\ mm}$ entre sí repartidos a diferentes profundidades. El array lineal que se ha utilizado es de $\boldsymbol{128}$ elementos con una frecuencia central $\boldsymbol{f_c = 5\ MHz}$ y una separación entre elementos $\boldsymbol{pitch = 0.6\ mm}$. Debido a que la lente se calcula de forma dinámica dentro del mismo *kernel*, 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.

5. CONCLUSIONES

Se ha visto que la generación de imagen ultrasónica tiene como principal característica el requerimiento de alto grado de paralelismo. Los algoritmos de procesamiento encargados de la composición de una imagen (algoritmos de filtrado, de copia, de conformación, interpolación, apodización, focalización dinámica, de visualización, etc) se aplican sobre un gran conjunto de señales digitalizadas y un denso volumen de puntos espaciales en función de las dimensiones de la imagen que se desea obtener. Así pues, en este trabajo se ha estudiado como resolver estos problemas haciendo uso de la potencia de cálculo (y por tanto del paralelismo) presente en las tarjetas gráficas de la actualidad. El objetivo ha sido reducir el tiempo de ejecución de los procesos de conformación de haz y para ello se ha utilizado una tarjeta gráfica sencilla equipada con tecnología NVIDIA CUDA. Los resultados experimentales muestran unas buenas tasas de imagen logrando alcanzar en el caso de una imagen de 500x500 pixeles una tasa de 70 imágenes por segundo. Por ello, se puede concluir que la paralelización en GPU constituye un excelente método para acelerar la formación de la imagen a alta velocidad, a bajo precio y complejidad y que puede aplicarse a muchos otros casos de estudio.

AGRADECIMIENTOS

Este trabajo está apoyado por el Ministerio de Ciencia e Innovación de España a través del proyecto DPI2010-19376 y la beca BES-2008-008675.

BIBLIOGRAFÍA

[1] B. D. Steinberg, Principles of Aperture and Array System Design, Wiley-Interscience, 1976.
[2] G.S. Kino, Acoustic Waves: Device, Imaging and Analog Signal Processing, Prentice Hall, 1987.
[3] P.D. Wilcox, C. Holmes, B.W. Drinkwater, Advanced reflector characterization with ultrasonic phased arrays in NDE applications, IEEE Transactions on Ultrasonics Ferroelectrics and Frequency Control 54 (4) (2007) 1541–1550.
[4] C. Holmes, B.W. Drinkwater, P.D. Wilcox, Advanced post-processing for scanned ultrasonic arrays: Application to defect detection and classification in non-destructive evaluation, Ultrasonics 48 (2008) 636–642.
[5] C. Holmes, B.W. Drinkwater, P.D. Wilcox, Post-processing of the full matrix of ultrasonic transmit–receive array data for non-destructive evaluation, NDT and E International 38 (8) (2005) 701–711.
[6] D. Luebke, “CUDA: Scalable parallel programming for high-performance scientific computing,” in 2008 5th IEEE International Symposium on Biomedical Imaging: From Nano to Macro. Paris, France: IEEE, May 2008, pp. 836–838.
[7] Nvidia CUDA Guia de programación. Abril 2011.
[8] J. Stone, D. Gohara, and G. Shi, “OpenCL: A parallel programming standard for heterogeneous computing systems,” Computing in Science & Engineering, pp. 66–72, 2010.
[9] Martin, C. J., O.Martinez and L. G. Ullate (2008). Reduction of grating lobes in saft images. IEEE International Ultrasonics Symposium pp. 721–724.