Rendimiento y análisis energético en formación de haz con GPGPU-Sim

Performance and Power Analysis of Beamforming with GPGPU-Sim

Rafael Alejandro Vejarano[1], Jeong-Gun Lee[2]
[1]Centro Regional de Coclé, Universidad Tecnológica de Panamá, [2]Departamento de Ingeniería Computacional, Hallym University
[1]rafael.vejarano@utp.ac.pa, [2]jeonggun.lee@hallym.ac.kr

Resumen— *Las unidades de procesamiento gráfico (GPU) se utilizan actualmente en una amplia gama de aplicaciones científicas y comerciales. Estos son de las primeras plataformas de energía eficientes y asequibles para el procesamiento de datos en paralelo. En el campo de las imágenes médicas, las GPU son en algunos casos cruciales para hacer uso práctico de algoritmos computacionalmente exigentes. Por esta razón, en esta investigación se explora el área de consumo y eficiencia energética al utilizar GPU como procesadores de señal e imagen primaria para sistemas médicos portátiles futuros de imágenes de ultrasonido. Como metodología de estudio se utilizó la aplicación GPGPU-Sim, un simulador de nivel de ciclo de cargas de trabajo de computación GPU ejecutando código escrito en CUDA, realizando variadas configuraciones a fin de determinar la arquitectura con óptimo rendimiento para nuestra aplicación de formación de haz (beamforming).*

Palabras claves— *Formación de haz, CUDA, Rendimiento, Consumo energético, GPPGU-Sim, GPUWatch.*

Abstract— *Graphics processing units (GPUs) are used today in a wide range of scientific and business applications. They have emerged as one of the first affordable and energy efficient platforms for data-parallel processing. In the field of medical imaging, GPUs are in some cases crucial for enabling practical use of computationally demanding algorithms. For this reason, this paper explores the suitability of using GPUs as the primary signal and image processors for future portable ultrasound systems. The case study method embraces the full set of procedures needed to test the beamforming application by changing different parameters in the architecture of GPGPU-Sim, a cycle-level simulator running GPU computing workloads written in CUDA.*

Keywords— *Beamforming, CUDA, Performance, Power consumption, GPPGU-Sim, GPUWatch.*

Tipo de artículo: Original
Fecha de recepción: 12 de octubre de 2015
Fecha de aceptación: 7 de abril de 2016

1. Introducción

El cofundador de Intel, Gordon Moore predijo, en la llamada ley de Moore, que el número de transistores por pulgada en los circuitos integrados se duplicaría cada 18 meses y que esa tendencia continuaría al menos durante dos décadas. Para aquel entonces, 1956, el chip más complejo contaba con 64 transistores. Hoy, un procesador i7 tiene 731 millones.

Los avances progresivos de la tecnología y la reducción en costo, hacen posible que los dispositivos computacionales sean accesibles a un creciente número de personas en el mundo.

El rendimiento, consumo energético y la eficiencia son un asunto que preocupa a científicos y consumidores. En este sentido, emergen las unidades de aceleración gráfica como plataforma para computación paralela ya que estas arquitecturas de GPU poseen miles de procesadores de flujo diseñados para proveer un alto rendimiento para flujos de trabajo en paralelo. La programación de propósito general, tal como la Arquitectura de Cómputo Unificada, CUDA, por sus siglas en inglés (Compute Unified Device Architecture), son a menudo utilizadas para facilitar el desarrollo de aplicaciones en GPGPU. El rendimiento en las computadoras modernas es principalmente afectado por el procesador y la memoria, las cuales afectan la velocidad de respuesta, por lo que es importante priorizar su relación con rendimiento y potencia. El problema aumenta cuando nos dirigimos a los dispositivos móviles, ya que estos poseen un pequeño presupuesto de energía que requiere un rendimiento que excede los niveles de las computadoras de escritorio, sin considerar las técnicas de reducción de energía y las arquitecturas de ahorro energético. La vida de las baterías limita las capacidades de los dispositivos.

Por otro lado, los escáneres modernos de ultrasonido son capaces de producir imágenes de las estructuras anatómicas con gran detalle. Estos utilizan ondas sonoras de alta frecuencia para observar los tejidos blandos, así como músculos y órganos internos. Una imagen de *beamforming* (conformación de haces) es una tecnología de procesamiento de señal que se utiliza para dirigir la recepción o transmisión de un haz de una matriz de transductores con una orientación angular elegida de transmisión de ondas sonoras de alta frecuencia, y la generación de imágenes utilizando ecos. El *beamforming* puede aumentar sus unidades a través del rendimiento gráfico de procesamiento (GPU). Sin embargo, hoy en día, el rendimiento por vatio y el consumo de energía se ha planteado como una medida importante para la evaluación de la eficiencia de una GPU.

Actualmente los escáneres de ultrasonido son muy pequeños como para ser portable, incluso un dispositivo de tableta portátil. El uso eficiente de energía en estos dispositivos portátiles es importante, ya que afecta la vida de la batería. Una nueva gama de posibilidades se puede realizar con herramientas como OpenCL en dispositivos móviles basados en Android.

La potencia de las GPU no se limita solo al procesamiento gráfico, esta refleja la intención de procesos de alto rendimiento que se realizan en la GPU. Sin embargo, en los últimos diez años o más, los desarrolladores e ingenieros han logrado adaptar la potencia de procesamiento de las GPU para propósitos de cálculos más generales. Esta mejora está motivada por su máximo rendimiento teórico, que supera con creces la de las CPU. No obstante, las GPU todavía se consideran como altamente consumidoras de energía y han sido cuestionables su liderazgo de diseño en computación de alto rendimiento. Hoy en día, representa solo una pequeña fracción del consumo total de energía, en torno al 2%. Sin embargo, este consumo está creciendo muy rápido, a un ritmo de dos dígitos por año [1]. La batería es una de las principales preocupaciones para los dispositivos móviles en cuanto a la gestión de los costos, energía / enfriamiento, que domina gran parte del costo de los equipos en los centros de datos. En la actualidad, muchas aplicaciones de datos paralelos utilizan unidades gráficas de procesamiento para alcanzar aceleraciones impresionantes, en el rango de 100X para muchas aplicaciones. Las tecnologías de información representan solo una pequeña fracción del consumo total de energía, en torno al 2%. Sin embargo, este consumo está creciendo muy rápido, a un ritmo de dos dígitos por año [1]. Tienen consumos de energía de hasta 300W [2]. El consumo de energía de un GPU típico es también mucho más alta que la de una CPU multinúcleo. Por ejemplo, la GPU NVIDIA Tesla consume más de 250 Watts en carga pico, más del doble del consumo de potencia pico de una CPU multinúcleo. Ren [3] y Huang [4] demuestran que la eficiencia energética de las GPU es alta en comparación con las CPU de núcleos en una operación de multiplicación de matriz.

Existen pocos estudios sobre los efectos especiales de la tecnología, con conciencia en el uso de energía en las GPU. El pionero en la conservación energética en GPU es Hong [5]. Él utiliza el análisis de código PTX para predecir los ciclos de ejecución del núcleo de la GPU y estimar el consumo de energía por núcleo. Él propone utilizar menos SM (Stream Multiprocesor) cuando sea posible para reducir el consumo de energía de la GPU.

Jiao [6] ha estudiado el rendimiento y potencia de tres kernel sobre una GPU ejecutándose en una NVIDIA GeForce GTX 280, con diferentes procesadores y frecuencias de memoria en diferentes núcleos. Él observó que al aumentar la frecuencia de la GPU se mejora el rendimiento, excepto para aplicaciones que implican principalmente acceso a la memoria. Para aquellas aplicaciones que no requieren una gran cantidad de acceso a memoria, la reducción en el consumo de energía es solo del 5%, con una mejora relativa en la eficiencia de energía de solo el 4%.

ABE [7], informó que cambiando el voltaje y la frecuencia de la GPU se logra reducir el consumo de energía en un 28%, en un sistema que realiza una multiplicación de una matriz 64 x 64 con un incremento de frecuencia sobre la memoria en una NVIDIA GeForce GTX 480.

Rong Ge [8], ofrece varias métricas para evaluar el impacto de la GPU, rendimiento, potencia y energía. En este estudio se concluye que un mayor ajuste en el Voltaje Dinámico y Escalado de Frecuencia (DVFS, Dynamic voltage and frequency scaling) de la GPU es óptima en términos de rendimiento y eficiencia energética.

Dos importantes contribuciones en el análisis de potencia y rendimiento de la GPU, usando un simulador, son suministrados por Wilson WL Fung y JinwenLeng

[9]. Wilson WL Fung en el Manual de GPGPU-Sim [10], en el cual proporcionan una amplia información sobre el modelo implementado en la microarquitectura GPGPU-Sim. JinwenLeng integra GPU-Wattch y GPGPU-Sim y demuestra el ahorro de energía mediante la variación de voltaje y frecuencia de escalamiento dinámico (DVFS). Sus aportes, son una referencia básica para nuevas investigaciones, que suministran una buena guía experimental sobre la validación de modelos de energía sobre GPU. También ofrecen un buen ejemplo de cómo validar el modelo de energía que puede ser utilizada para estudios de *software* de eficiencia energética. El modelo es configurable, capaz de realizar cálculos a nivel de ciclo y se puede validar cuidadosamente contra el soporte físico real.

El objetivo de esta investigación es observar cómo la energía (E) varía en función de esas variables. Por ejemplo, para la CPU en general, se cree que con una tensión fija, utilizando frecuencias de procesador se reduciría la energía debido a un menor tiempo de ejecución. Sin embargo, esto no siempre es cierto para las GPU. Se trata de un problema de investigación interesante el encontrar la frecuencia de base óptima que puede ahorrar la mayor cantidad de energía para una aplicación particular, especialmente cuando se requiere alta intensidad operativa.

Este trabajo describe el análisis de rendimiento y consumo de energía de *Beamforming* para un solo frame (un solo cuadro de imagen). El mismo se organiza de la siguiente manera: la arquitectura de la GPU, la arquitectura del modelo de programación CUDA, la arquitectura del simulador GPGPU-Sim, la GPU para uso en imágenes médicas, *beamforming*, la estructura de nuestro código de prueba en *beamforming*, resultados y conclusiones.

2. Arquitectura GPU

En la micro arquitectura GPU, los núcleos están conectados a controladores de memoria usando una red de interconexión en *chip* y fuera de *chip*, mediante una matriz de celdas de memoria que se comunican con los núcleos a través de controladores en *chip*. La arquitectura NVIDIA Fermi contiene hasta 16 multiprocesadores de flujo. En cada multiprocesador de flujo (SM, StreamvMultiprocesor), hay varios núcleos en orden, cada una de los cuales se conocen como procesadores de flujo (SP, StreamvProcessors). Cada SP tiene una unidad aritmética lógica y una unidad de punto flotante que puede ejecutar instrucciones en hilos, por ciclo de reloj. El número de SP por SM varía con las generaciones de GPU. Cada SM posee una caché de instrucciones, una caché de memoria constante, una memoria compartida, registros, instrucción multihilo y unidades funcionales especiales. Cada GPU, actualmente viene con 4 *gigabytes* de memoria gráfica de velocidad doble (GDDR) DRAM, conocido como memoria global. Cada SM cuenta con dieciséis unidades de carga / almacenamiento y cuatro unidades de funciones especiales (SFU). Un conjunto puede contiene 32 hilos paralelos, ejecutando una sola instrucción sobre múltiples datos (SIMD). Dos o más SM se agrupan para compartir una única unidad de procesamiento de textura y caché L1 entre el SM. Una red de interconexión en el *chip* conecta diferentes SM al caché de memoria y a controladores L2. Además de la memoria principal de la CPU, la GPU tiene su propia memoria (fuera de *chip*), que está conectada a los controladores de memoria en el *chip*. La CPU y la GPU están en diferentes *chips*, y se comunican a través de la PCIe o PCI Express tal como se observa en la figura 1.

La memoria de la GPU está organizada en forma jerárquica: memoria global, caché de textura, caché constante, memoria compartida y registros.

Una GPU tiene cuatro variables escalables: la frecuencia del núcleo, voltaje del núcleo, frecuencia DRAM de E/S, y voltaje DRAM [8]. Los equipos modernos son principalmente afectados por el procesador y la memoria.

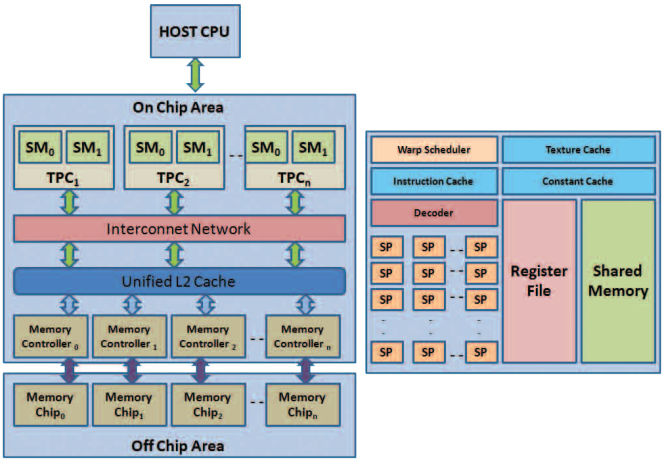

Figura 1. Arquitectura de la GPU.

Estos dos elementos tienen un fuerte efecto en el tiempo de respuesta y los accesos de memoria de la computadora. Por lo tanto, se hace necesario considerar lo más importante, el rendimiento o la potencia. El problema se incrementa si nos movemos al campo de la computación móvil, donde los dispositivos cuentan con poca carga energética y requieren tiempos de respuestas que exceden los niveles de las computadoras de escritorio, limitando las capacidades de esos dispositivos.

3. Arquitectura CUDA

CUDA, Arquitectura Unificada de Dispositivos de Cómputo por sus siglas en inglés (Compute Unified Device) es un modelo y plataforma de programación creado por NVIDIA y ejecutado en las unidades de procesamiento gráfico (GPU) que fabrican. El lenguaje de programación es conocido como CUDA C / C ++ para la computación de propósito general, y es una extensión del lenguaje de programación C / C ++.

Usando CUDA, miles de hilos en paralelos y concurrentes pueden ser lanzados por la CPU y agrupados en una entidad de ejecución denominada bloque. Un programa CUDA consiste en una o más fases que se ejecutan, ya sea en el anfitrión (CPU) o un dispositivo (GPU). La arquitectura CUDA consta de tres partes básicas: el grid, el bloque y los hilos tal como se observa en la figura 2.

Un hilo de CUDA es solo la ejecución de una función llamado kernel o núcleo, con un índice dado. Cada hilo utiliza su índice para acceder a elementos en un arreglo. El chip de la GPU se organiza como una colección de multiprocesador (MP), cada uno responsable del manejo de uno o más bloques en una cuadrícula. Un bloque, es simplemente un grupo de hilos de ejecución simultánea en ningún orden en particular, que es coordinar de alguna manera mediante el uso de funciones de sincronización. Los diferentes bloques se agrupan en una entidad llamada grid con ninguna sincronización entre los bloques. Durante la ejecución, un bloque se asigna a un SM determinado. Los bloques que se ejecutan en diferentes SM no pueden comunicarse entre sí.

Sobre la base de los recursos disponibles en un SM, uno o más bloques se pueden asignar al mismo SM. Todos los registros del SM se asignan a los diferentes hilos de diferentes bloques.

En la GPU, los hilos se ejecutan basados en un modelo denominado SMIT, es decir, una sola instrucción sobre múltiples datos. Las peticiones se agrupan en unidades de carga y almacenamiento, dentro de un paquete de memoria independiente con patrones de acceso. Así, la memoria mejora su rendimiento al ocultar los accesos pequeños de memoria en accesos más grande. Este fenómeno se denomina coalescencia de memoria, donde los hilos se ejecutan de manera sincronizada.

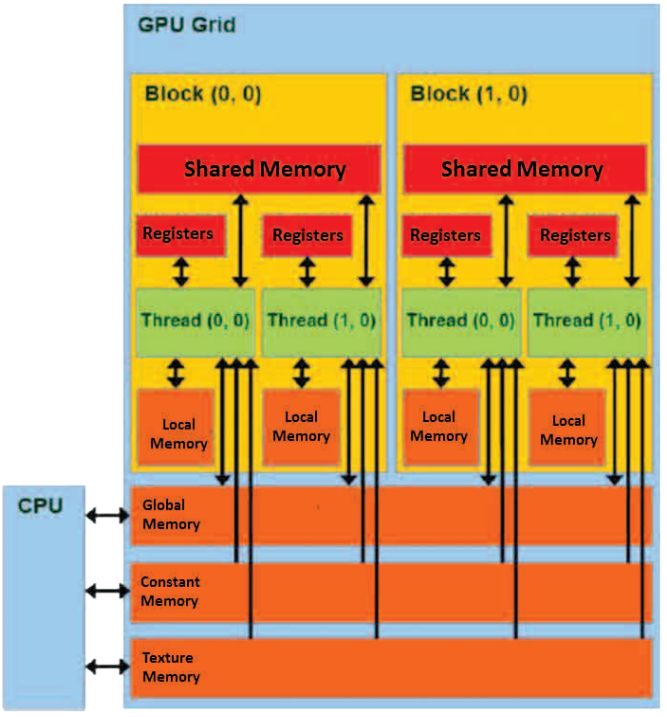

Figura 2. Arquitectura CUDA GPU.

4. Arquitectura de GPGPU-Sim

GPGPU-Sim es un simulador de rendimiento de nivel de ciclo detallado para GPU, integrado con GPUWatch para predicción energética de ciclo a ciclo que puede ejecutar CUDA u OpenCL. GPGPU-Sim se desarrolla en la Universidad de British Columbia y ofrece resultados estadísticos detallados, tales como permitir la más amplia gama de opciones de diseño y simulación. GPGPU-Sim informa sobre los ciclos en la GPU que está ocupando, no modela los tiempos en la CPU o el momento de transferencia sobre la PCI Express (es decir, el tiempo de transferencia de memoria entre la CPU y la GPU) [10]. La arquitectura modelada en GPGPU-Sim es una arquitectura Fermi, que constituye una completa plataforma para la investigación y optimización del rendimiento y la energía.

GPGPU-Sim utiliza el modelo de una sola instrucción sobre múltiples datos (SIMT) que ejecutan lotes de hilo en un mismo paso. Estos núcleos están acoplados a través de una conexión de red en chip a particiones de memoria GDDR DRAM. Una estructura de pipeline (tubería), SIMD altamente multihilo (NVIDIA llaman Corriente multiprocesador o SM), es modelada por la organización GPGPU-Sim. La organización de un núcleo SIMT se describe en la figura 3 y muestra la arquitectura general de GPGPU-Sim.

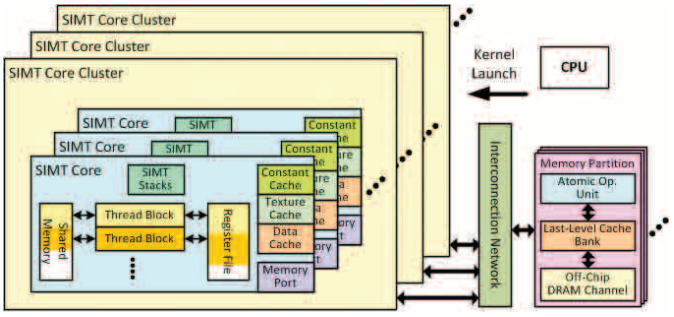

Figura 3. Arquitectura general modelada por GPGPU-Sim.

GPGPU-Sim soporta cuatro dominios de reloj independientes: a nivel de núcleo, interconexión de red, reloj y caché L2. Los valores de ese dominio pueden ser cambiados arbitrariamente (sin dependencia entre ellos). También, GPGPU-Sim logra integrar a GPUWatch, creando una fusión de estructura perfecta de rendimiento y energía para optimizar arquitecturas multinúcleo. La figura 4 muestra la estructura de GPGPU-Sim / GPUWatch integrada en una sola estructura.

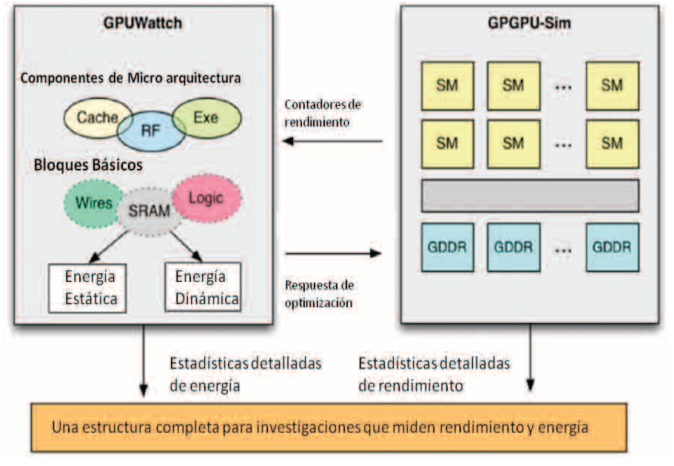

Figura 4. Estructura GPGPU-Sim / GPUWatch.

GPGPU-Sim, ofrece dos tipos de simulación: funcional y de rendimiento. En la simulación funcional, se simula la arquitectura general mientras que en la simulación de rendimiento se simula el tiempo de ejecución de informes por ciclo mediante la ejecución de los núcleos de CUDA.

5. GPU para imágenes médicas

El ultrasonido ha sido una técnica de imagen médica muy popular durante muchos años. Las unidades de procesamiento gráficos (GPU) son hoy en día utilizados en una variedad de tecnologías para imágenes médicas. El uso de la GPU en este campo ha madurado hasta el punto de que hay varias modalidades médicas que soportan los GPU Tesla [11].

Todos estos desarrollos se han orientado hacia el apoyo de gráficos en tiempo real, los cuales toman ventajas de estos procesadores para obtener considerables beneficios de rendimiento en la operación de análisis de imágenes. Las GPU rápidamente se han transformado de un dispositivo especializado a un elemento de cálculo de gran alcance.

El ultrasonido es una de las técnicas de imagen más utilizadas que permiten al médico ver el interior del cuerpo de un paciente para el diagnóstico y la intervención mediante imagen guiada, libre de radiación, no invasiva y en modalidad de tiempo real de imagen. Los ultrasonidos son ondas longitudinales que causan una oscilación hacia atrás y hacia adelante y que producen una serie de compresiones y refracciones [12]. Las imágenes se producen cuando las ondas de sonido se dirigen hacia el cuerpo y se refleja de regreso a un escáner que los mide. Cada línea de exploración es producida por un pulso de ultrasonido. Los impulsos reflejados se registran, y la amplitud y el tiempo de los impulsos devueltos dan una imagen de la estructura del tejido a lo largo de la trayectoria del haz. El Modo de brillo (B-Mode) es el modo básico que se utiliza generalmente y que es utilizado en esta investigación.

6. *Beamforming*

Las ondas de ultrasonido para diagnóstico transportan información sobre el cuerpo de regreso al sistema de imagen [13]. El modo B-Mode, produce una imagen en dos dimensiones, blanco y negro. El cuerpo puede ser dibujado en diferentes planos dependiendo de la posición de la sonda.

Beamforming es una técnica de procesamiento de señal utilizado para controlar la direccionalidad de la recepción o transmisión de una señal en una matriz de transductores [14]. El pulso de ultrasonido se transmite desde un transceptor que tiene una serie de pequeños elementos piezoeléctricos. Cada elemento puede transmitir y recibir impulsos de ultrasonidos de forma independiente a otros elementos. Los elementos transceptores individuales serán afectados por los pulsos que se reflejan de vuelta y se guardan en diferentes momentos en función de su origen.

El *beamforming* que se recibe, retarda las señales de cada canal (elemento), de manera que se sincronizan las señales generadas por los pulsos. Cuando se transmite el pulso, solo es posible centrarse en un punto dado y cuando se recibe, el retraso se puede ajustar de forma dinámica para centrarse en todos los puntos a lo largo del haz. La tecnología GPU ha ido en aumento y ha superado a la CPU tan rápido, que ahora podría ser posible llevar a cabo la formación de haz en *software*.

7. Estructura del código de *beamforming*

El B-Mode convencional consiste en una serie de pasos: adquisición de datos RF, la formación de haz, demodulación en cuadratura, detección envolvente, compresión logarítmica y generación de imagen. Nuestra imagen B-Mode implementa el algoritmo en tres núcleos a los que se han llamdo: delay_intpol, quad_demod, env_detect. Todos los núcleos son limitados en memoria y directamente afectados por la arquitectura de memoria (bus de memoria, tamaño de caché y ancho de banda de memoria) [15].

Pinned memory es utilizada con el fin de evitar el intercambio con la memoria de la CPU. Esto proporciona velocidades de transferencia mejoradas.

El kerneldelay_intpol ejecuta el cálculo por retardo y filtrado de interpolación, el cual es una parte esencial de la formación del *beamforming* que calcula las diferencias en el tiempo de llegada de la onda entre los elementos de la matriz. Mientras que el filtrado de interpolación proporciona una serie de filtros con el fin de construir nuevos puntos de datos dentro de la gama de un conjunto discreto de puntos de datos conocidos. El pulso transmitido desde cada componente transductor independiente, determina la forma del haz donde el cálculo de retardo depende de la posición de un punto focal en el mismo tiempo.

La memoria constante se utiliza debido a que los hilos deben leer varias veces la misma dirección de memoria. La salida del kerneldelay_interpol, es el punto central que contiene el índice de dirección de memoria para los valores pares e impares.

El kernelquad_demod recibe del kerneldelay_interpol los datos como parámetro de entrada, que es la señal de *beamforming* después de realizada la suma de todos los pulsos de señal en todos los canales. El resultado es almacenado en memoria compartida y debido a su ubicación, la latencia es menor que en un *buffer* típico. Durante el proceso de demodulación de cuadratura, la señal es convertida a una banda base y en señales de cuadratura.

En el kernelenv_detectkernel, todos los I/Q son combinados en una imagen B-Mode para obtener la señal de envoltura. El I/Q utiliza un filtro de paso bajo que se combina para producir una data envolvente. Este kernel, también utiliza memoria compartida para la entrada de datos así como memoria constante. La salida de este kernel es enviado de regreso al CPU para mostrar la imagen. Otras tareas como comprensión logarítmica, conversión de escaneo y formación de imagen se realizan del lado del CPU.

8. Experimentación y resultados

Para esta investigación se ha utilizado GPGPU-Sim, un simulador de rendimiento a nivel de ciclo para la computación GPU y para la estimación de energía con GPUWatch, el cual está integrado a GPGPU-Sim. El simulador está configurado para que coincida con una arquitectura GTX470 con frecuencias reducidas a la mitad en los pasos: 600, 800, 1000 y 1200MHz (núcleo) y 900 MHz (memoria) para un total de 9 diferentes SM, variando de 12SM y 20SM. Para configurar correctamente nuestro ambiente, es necesario modificar ciertos parámetros en tres archivos diferentes del simulador: en GPGPU-SIM (gpgpusim.config), en GPUWATTCH (gpuwattch_gtx480.xml) y la Red de interconexión (config_fermi_islip.icnt).

GPGPU-sim ofrece una serie de estadísticas de rendimiento que ponen a disposición una buena comprensión de cómo la aplicación CUDA se ejecuta con la arquitectura de un GPU simulado. El contador de ciclo global,gpu_tot_sim_cycle, es compartida por todos los núcleos SIMT y muestra el número de ciclos simulado para todos los núcleos lanzados hasta el momento.

Después de cada simulación del núcleo, se añade el valor de gpu_sim_cycle a gpu_tot_sim_cycle. Si solo hay un núcleo, será lo mismo al final de la simulación. Si hay varios lanzamientos del kernel, gpu_tot_sim_cycle será la suma de todos gpu_sim_cycle por lanzamientos del kernel.

Aquí se evalúa el rendimiento, consumo de energía, tiempo de ejecución y la eficiencia energética de diferentes situaciones:

En primer lugar, el código CUDA para *Beamforming* está configurado para adaptarse a 224, 112 y 56 líneas de escaneo con el fin de observar su comportamiento con 15SMs, con un reloj fijo de 1.200 MHz y un reloj de memoria fija de 900 MHz.

En segundo lugar, se ha variado la frecuencia de reloj de 400MHz a 1200MHz y desde 12SMs a 20SMs con un reloj de memoria fija a 900MHz para investigar el impacto de agregar más SMS a la GPU, sobre la frecuencia de reloj del núcleo.

En tercer lugar, se ha variado la frecuencia de memoria entre 400 MHz y 1200 MHz a fin de investigar el impacto de la memoria sobre la GPU. En este experimento, se ha configurado un conjunto de 15SMs con GPGPU-Sim. También se ha investigado el impacto cuando se utiliza paginación o no paginación.

La formación de *beamforming* consta de 224 líneas de exploración. Como se muestra en la figura 5, el número de ciclo varía según el número de línea de exploración debido a la carga de trabajo.

Para examinar el impacto de DVFS, cuatro clases de métricas se utilizan en esta investigación:

- Rendimiento: es la cantidad de trabajo ejecutado por un sistema informático en comparación con el tiempo cumplido para ejecutar esa tarea.
- Potencia: describe el consumo de energía por unidad de tiempo y se mide en vatios.
- Energía: se define como la habilidad o la capacidad para hacer el trabajo.
- Eficiencia energética: es un indicador combinado de rendimiento y potencia. En la física, la energía se define como la cantidad de trabajo que puede ser realizado por la fuerza y se mide en Julios.

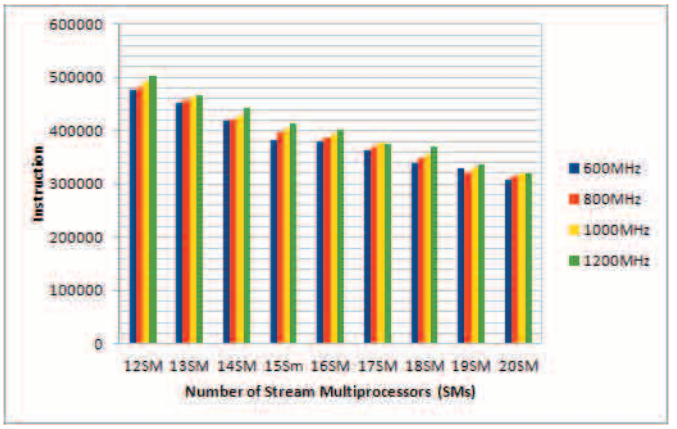

Figura 5. Rendimiento en *beamforming*.

El experimento muestra una escalada del SM 12 al SM 14 en cuatro frecuencias de reloj diferentes. La figura 6 muestra el rendimiento de la conformación del haz y la relación de la variación del número de SMS, así como la frecuencia de reloj. El impacto de este ajuste mejora el tiempo de ejecución, pero en detrimento de la conservación de la energía, como se muestra en la figura 7.

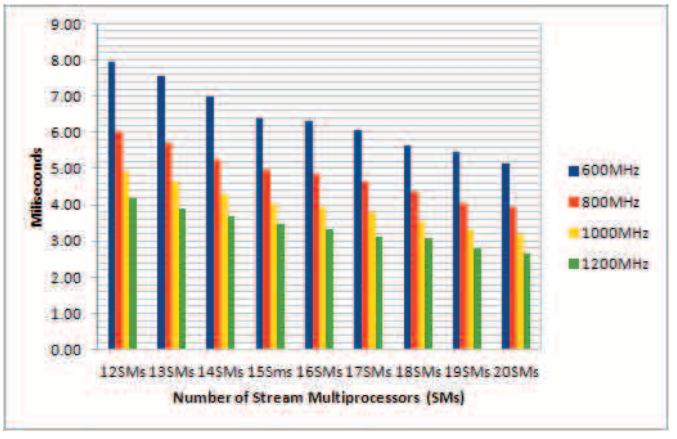

Figura 6. Tiempo de ejecución de *beamforming*.

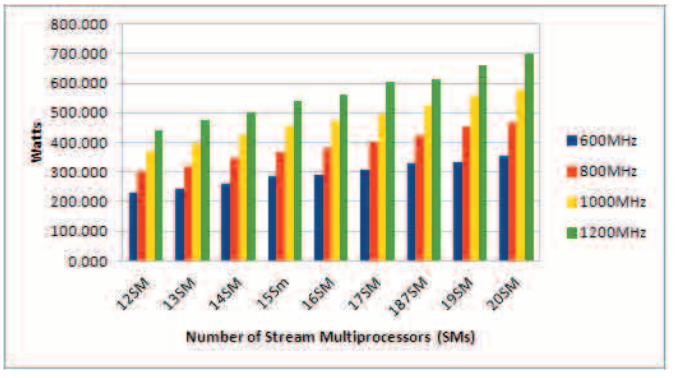

Figura 7. Consumo energético en *beamforming*.

La potencia y rendimiento viene dado por GPGPU-Sim. La eficiencia de energía se calcula según la siguiente ecuación propuesta por Ed Grochowski [16]:

$$\frac{Joules}{Intructions} = \frac{\left(\frac{Joules}{Intructions}\right)}{\left(\frac{Intructions}{Second}\right)} = \frac{Watt}{IPS} \quad (1)$$

y su gráfico correspondiente se muestra en la figura 8.

Cuando se reduce el número de línea de exploración y se mantiene la configuración anterior, se observa que el tamaño del problema disminuye, pero el consumo de energía se mantiene casi igual.

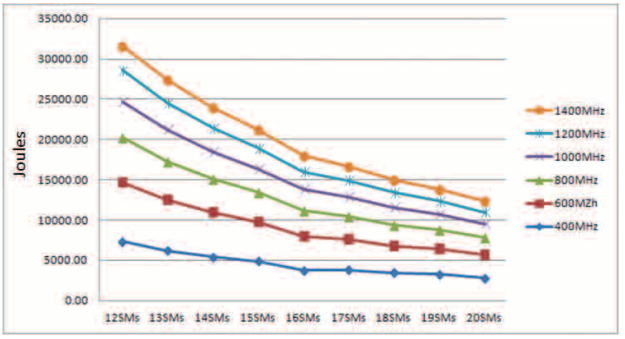

Figura 8. Eficiencia energética en *beamforming*.

Teniendo en cuenta el rendimiento, tiempo de ejecución y el consumo, una buena configuración para nuestra formación de haz es un conjunto de 17SM funcionando a una frecuencia de reloj de 800 MHz. Esta configuración consume un promedio de 401 vatios en un tiempo de 4 ms. El análisis de componentes de energía muestra que los 5 componentes más consumidores de energía son el registro de archivo (REF), FPUP (Unidad de Punto Flotante), SFUP (Unidad Flotante Especial), PIPE y NOCP (Red en Chip) como se muestra en la figura 9. Este dato es proporcionado por gpgpusim_power_trace_report.log.gz

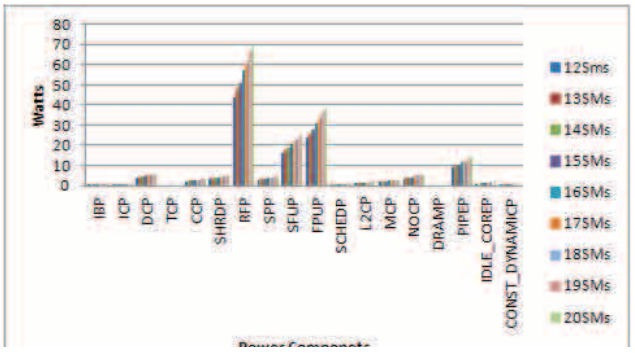

Figura 9. Consumo energético por componentes.

9. Conclusión

Escalar el número de SM y la frecuencia del núcleo no logra una ganancia sobre la energía. La combinación de SM y la frecuencia de reloj producen como resultado un alto consumo de energía para la aplicación de *beamforming*. Sin embargo, podemos observar que aumentar el número de SM a una frecuencia de reloj baja puede lograr conservar mejor la energía y mantener un buen rendimiento. El uso de GPU en la próxima generación de sistemas de ultrasonido es prometedor. Además, la portabilidad tecnológica que posee procesadores de múltiples núcleos permite la paralelización. Las imágenes en sistemas de ultrasonido deben ser de bajo consumo energético, bajo costo, portables, de fácil programación y flexible.

En conclusión, se recomienda un *overclock* de la GPU y la memoria individualmente para aislar cualquier inestabilidad que se puede introducir en ese componente en particular. Una vez que la frecuencia máxima es conocida, la GPU y la memoria se pueden configurar a esas velocidades simultáneamente y luego, se debe realizar una prueba de estabilidad una vez más.

10. Referencias

[1] Michel Bénard: "Energy Efficient HPC in Metropolitan Environments," in International Conference on Energy-Aware High Performance Computing, Hamburg , 2010.

[2] David Defour: Arnaud Tisserand, and Sylvain Collange, "Power Consumption of GPUs from a Software Perspective," in ICCS '09 Proceedings of the 9th International Conference on Computational, Berlin, 2009, pp. 914–923.

[3] Reiji Suda and Da Qi Ren: "Power Efficient Large Matrices Multiplication by Load Scheduling on Multi-core and GPU Platform with CUDA," in IEEE CSE'09, 12th IEEE International Conference on Computational Science and Engineering, Vancouver, 2008, pp. 424-429.

[4] S. Huang, S. Xiao, and W. Feng: "On the Energy Efficiency of Graphics Processing Units," in Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, Rome, 2009, pp. 1-8.

[5] Sunpyo Hong: "Modeling performance and power for energy-efficient GPGPU computing," Georgia Institute of Technology, Tesis Doctoral 2012.

[6] Y. Jiao, H. Lin, P. Balaji, and W. Feng: "Power and Performance Characterization of Computational Kernels on the GPU," in Green

Computing and Communications (GreenCom), 2010 IEEE/ACM Int'l Conference on & Int'l Conference on Cyber, Physical and Social Computing (CPSCom), Hangzhou, 2010, pp. 221 - 228.

[7] Y. Abe; H. Sasaki; S. Kato; K. Inou; M. Edahiro; M. Peres: "Power and Performance Characterization and Modeling of GPU-Accelerated Systems," in HotPower'12 Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems, Berkeley, 2012, p. 10.

[8] Rong Ge, Ryan Vogt, Jahangir Majumder, and Arif Alam: "Effect of Dynamic Voltage and Frequency Scaling on K20 GPU," in Parallel Processing (ICPP), 2013 42nd International Conference on, Lyon, 2013, pp. 826 - 833.

[9] Leng, J.; Hetherington, T.; Tantawy, A E; Gilani, S; Kim, N.; Aamodt, T..; Reddi, V. (2013) GPUWattch Energy Model Manual. http://www.gpgpu-sim.org/gpuwattch/

[10] Wilson W.L. Fung, Tayler H. Hetherington Tor M. Aamodt. (2012) GPGPU-Sim. http://gpgpu-sim.org/manual/index.php/Main_Page

[11] NVIDIA. High Performance Computing. Vertical Industry Solutions. http://www.nvidia.com/object/medical_imaging.htm

[12] James A. Zagzebski: Essentials of Ultrasound Physics, New Edition ed.: Elsevier Health Sciences, 1996.

[13] Thomas L. Szabo: Diagnostic Ultrasound Imaging: Inside Out, 1st ed. London: Elsevier Academic Press, 2004.

[14] Andrew A. Ganse. An Introduction to Beamforming. http://research.ganse.org/physics/beamforming/

[15] PaweáGepner and Michaá F. Kowalik: "Multi-Core Processors: New Way to Achieve High System Performance," in Parallel Computing in Electrical Engineering, 2006. PAR ELEC 2006. International Symposium on, Bialystok, 2006, pp. 9-13.

[16] Ed Grochowski and Murali Annavaram. (2006, Marzo) Energy per Instruction Trends in Intel Microprocessors. http://www.intel.com/pressroom/kits/core2duo/pdf/epi-trends-final2.pdf