Aceleración de la estimación de movimiento en la codificación H.264/AVC mediante GPUs

R. Rodríguez[1], J. M. Claver[2], G. Fernández-Escribano[1], A. J. Peña[3], J. L. Sánchez[1]

[1]Dpto. de Sistemas Informáticos, Universidad de Castilla-La Mancha, Albacete, España,
{rafael.rodriguez, gerardo.fernandez, jose.sgarcia}@uclm.es

[2]Dpto. de Informática, Universidad de Valencia, Burjassot, España, jose.claver@uv.es

[3]Dpto. Ing. y Ciencia de los Computadores, Univ. Jaume I, Castellón, España, tpenya@gmail.es

Resumen

En la actualidad el uso de Internet para la transmisión de contenido multimedia ha propiciado la aparición de nuevas propuestas para la codificación de vídeo, con el fin de proporcionar cada vez mejor calidad en las imágenes y reducir las necesidades de ancho de banda y almacenamiento en memoria. El estándar de codificación H.264/AVC es un ejemplo de ello, pero lleva asociado un incremento del coste computacional. Esto es especialmente crítico tanto en la codificación en tiempo real, como en el caso de la adquisición de imágenes y su inmediata transmisión. Por otro lado, el cambio de alguna de las características de una secuencia de vídeo previamente codificada en tiempo real (transcodificación), requiere el uso de hardware específico así como de técnicas que aceleren dicho proceso.

Las actuales arquitecturas de las GPU proporcionan una alternativa interesante y de bajo coste para acelerar alguna de las partes del proceso de codificación de vídeo. En este trabajo en curso se presentan diversas aproximaciones que hemos seguido los últimos meses para la aceleración de la estimación de movimiento en la codificación H264/AVC sobre las GPU de NVIDIA G8800 GTX y Tesla C870, utilizando para programarlas la arquitectura CUDA.

1. Motivación

En los últimos años hemos venido asistiendo a un particularmente rápido avance en las telecomunicaciones, debido a la veloz implantación de Internet y las redes de telefonía móvil. Esto da lugar a nuevas oportunidades de mercado a empresas que, utilizando la red de redes, suministran a clientes distribuidos a lo largo y ancho del planeta todo tipo de información. El tráfico multimedia en general, y el vídeo digital en particular, ha crecido de forma espectacular en los últimos años. Este tráfico de vídeo, tanto en Internet como en las redes de telefonía móvil, aumenta diariamente; como prueba de ello están las redes móviles de comunicación de tercera generación, siendo la videoconferencia el principal reclamo publicitario que emplean. Por otra parte, la televisión digital, ya sea terrestre o aérea, está imponiéndose en el mercado de la televisión doméstica. Los usuarios tienen ahora la capacidad de demandar el tipo de contenidos que desean visualizar en determinados momentos y, en breve, serán capaces de solicitar dichos contenidos a cualquier hora y en cualquier momento.

Por otro lado, la salida al mercado de la Televisión en Alta Definición (HDTV del inglés *High Definition TeleVision*), promete ser una revolución en el mundo de las aplicaciones multimedia domésticas, ofreciendo calidades próximas a las que se pueden conseguir en la mejor sala de cine. Esta calidad implica aumentar la resolución de los televisores domésticos. Mientras que la imagen de televisión estándar se transmite con una resolución de 720 x 576 píxeles, la imagen de alta definición tiene un tamaño de hasta 1920 x 1080 píxeles, es decir, 5 veces mayor. Si tenemos en cuenta que como mínimo un píxel se representa con 3 bytes, una sola imagen en alta definición implica más de 6 Mbytes por imagen. Sabiendo que se necesitan 25 imágenes por segundo, una película de una hora requeriría un espacio cercano a los 540 Gbytes, o de 150 Mbytes por segundo. A esta medida también se la conoce como caudal o bit-rate (medida en Mbytes por segundo), y es la cantidad de información que un sistema debe ser capaz de procesar por unidad de tiempo para la correcta visualización de la secuencia. Hay que decir además que, en el supuesto de que el sistema de recepción fuera capaz de procesar esa cantidad de información por segundo, el sistema de red debería tener también la capacidad para servir, por unidad de tiempo, esa cantidad de datos. Sin embargo, tal cantidad de información es imposible de transmitir actualmente por las redes de telecomunicaciones existentes. Es en este punto donde entran en escena los estándares de compresión de vídeo. Las empresas los utilizan para almacenar la información de vídeo de una forma más compacta, reduciendo de forma considerable la información y minimizando la cantidad de la misma que debe enviarse al cliente; manteniendo el nivel de calidad lo más alto posible. En este sentido, y dado que hasta ahora el estándar de vídeo MPEG-2 [3] cumplía con el requisito de ofrecer muy buena calidad de imagen con un consumo de recursos de red aceptable, fue el estándar elegido por las empresas para almacenar sus contenidos multimedia.

La demanda de un estándar de vídeo que mantuviera la calidad de MPEG-2 y que redujera aún más el consumo de recursos (principalmente ancho de banda de la red) manteniendo la calidad, dio origen a un nuevo estándar de codificación de vídeo denominado H.264/AVC o MPEG-4 parte 10 (mayo de 2003) [1][2]. Este nuevo estándar es

capaz, entre otras cosas, de mantener la misma calidad de una película codificada en MPEG-2 pero reduciendo el espacio necesario para almacenarla en aproximadamente un tercio como mínimo; pudiendo en algunos casos disminuir el espacio necesario a la mitad. Naturalmente, también el ancho de banda necesario para transmitirla es reducido.

El principal problema que presentan las técnicas de codificación de vídeo en general, y de su transcodificación, en particular, si su diseño no es el adecuado, es su elevado coste computacional para aplicaciones que demandan requisitos de tiempo-real, muy acentuado en el caso del H.264; debido a que el aumento de las prestaciones de la codificación, se realiza a costa de incrementar exponencialmente las necesidades de cálculo. Este alto coste computacional puede ser eficientemente reducido adaptando el código secuencial de los codificadores/decodificadores a la arquitectura de las nuevas tarjetas gráficas que a día de hoy están surgiendo en el mercado. Conocidas por el nombre de *Graphic Processor Units* (GPUs), constituyen una alternativa interesante y de bajo coste para acelerar algunos de los procesos que forman parte de la codificación/decodificación de vídeo.

Existen recientes propuestas para acelerar la estimación de movimiento de H.264/AVC sobre GPUs [8-10]. Sin embargo, no están adaptadas a las nuevas arquitecturas como CUDA, como en [8] y [9] o proponen algoritmos que se desvían de la metodología estándar [10]. En este trabajo pretendemos acelerar la estimación de movimiento siguiendo, en todo lo posible, la metodología y resultados obtenidos en el software de referencia secuencial de H.264 JM [7]

El resto de este trabajo se organiza como sigue. En la sección 2 presentamos un breve resumen del estándar H.264/AVC. En la sección 3 se describen las características de las GPU y sus posibilidades de programación actuales. En la sección 4 se abordan los intentos llevados hasta la fecha para la paralelización de este codificador sobre una GPU y los resultados obtenidos. En la última sección comentamos las conclusiones a las que hemos llegado, y las direcciones que pesamos tomar para continuar con el trabajo iniciado.

2. Estimación de Movimiento en H.264

El estándar de vídeo MPEG-4 parte 10 [1] o H.264 [2] es un estándar de compresión de vídeo desarrollado conjuntamente entre la ITU-T *Video Coding Experts Group* (ITU-T VCEG) y el ISO/IEC *Moving Picture Experts Group* (MPEG). Debido a esto, el estándar ITU-T H.264 y el ISO/IEC MPEG-4 parte 10 son técnicamente idénticos. Implementado con la tecnología *Advanced Video Coding* (AVC), desarrollada por el grupo MPEG, su versión preliminar se completó en Mayo de 2003.

H.264 surgió con el objetivo de crear un estándar que fuera capaz de proporcionar una buena calidad reduciendo notablemente el flujo de bits de salida de la secuencia de vídeo codificada; en comparación con otros estándares previos tales como MPEG-2 [3]. En segundo lugar, se pretendía conseguir esto sin incrementar la complejidad del codificador, evitando convertir el estándar en imposible de implementar. Como objetivo adicional se fijó que el nuevo estándar pudiera ser aplicado a una gran variedad de aplicaciones, tales como el almacenamiento en DVD, videoconferencia, televisión por cable, aplicaciones de bajo caudal, media y alta definición en televisión, video-streaming a través de Internet, etc. H.264 promete un avance significativo, si lo comparamos con dos de los estándares más comercialmente más empleados que existían hasta ahora (MPEG-2 y MPEG-4 [4]). En términos de eficiencia a la hora de codificar, se espera que con H.264 se consiga aumentar la compresión hasta un 200% en relación con los estándares existentes hasta ahora, a la vez que se aumenta de forma sustancial la calidad y definición de imagen en las creaciones que se realicen con él.

Recientemente, han sido propuestas algunas modificaciones al estándar conocidas como *Fidelity Range Extensions (FRExt)* [5], entre las que destacan: permitir la codificación de las muestras a 10 y 12 bits para resolución en color de alta definición en los formatos de muestreo YUV 4:2:2 y 4:4:4, permitir la predicción intra-cuadro en bloques 8x8 o soportar espacios de color adicionales. FRExt fue completado en septiembre de 2004.

Debido a que H.264 contiene una gran cantidad de novedades y mecanismos de compresión (Figura 1), permite comprimir mucho más las secuencias de vídeo y proporcionar una mayor flexibilidad a la hora de implementar el codificador, por ejemplo, destinado a satisfacer el mayor número de aplicaciones posibles.

La estimación y compensación de movimiento es un proceso en el cual se eliminan las redundancias temporales existentes entre imágenes independientes, mediante la comparación de la imagen tratada en cada momento con otras imágenes anteriores o posteriores (imágenes de referencia), buscando un patrón que indique cómo se produce entre ellas el movimiento de las diferentes partes de la imagen. Para reducir la complejidad del proceso de búsqueda, la imagen que se procesa en cada momento se descompone en elementos de 16x16 píxeles (macrobloques o MBs), los cuales serán comparados dentro de las imágenes de referencia en un área de búsqueda delimitada.

Para mejorar la eficacia en la codificación, el estándar H.264 permite, no sólo tomar como referencia de comparación cada uno de los MB en los que se descompone una imagen, sino que éstos a su vez pueden ser subdivididos en bloques todavía menores, capaces de contener y aislar el movimiento por sí mismos, tomando estas subdivisiones como unidades de comparación. Por ello el estándar H.264 introduce un cálculo con un tamaño de bloque menor al usado hasta ahora; mayor flexibilidad en las formas del bloque y una mayor precisión en los vectores de movimiento, consiguiendo así una mayor fidelidad a la hora de reproducir fielmente los datos de la fuente original. Es lo que se conoce como múltiples tamaños para realizar la compensación de movimiento: 16x16 (MB), 16x8, 8x16, 8x8, 8x4, 4x8 ó 4x4.

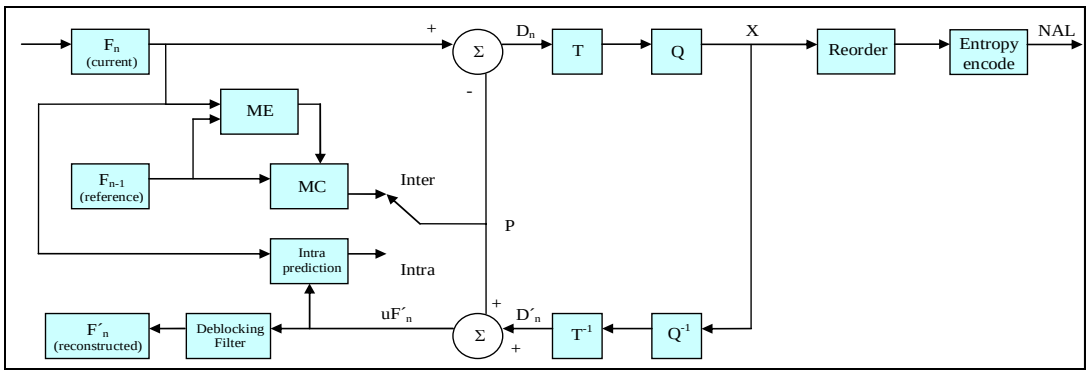

Figura 1. *Diagrama de bloques de un codificador H.264.*

Además, el estándar H.264 incorpora un nuevo concepto de tratamiento de imágenes llamado referencia múltiple de imágenes; el número de imágenes que pueden emplearse como imágenes de referencia se aumenta considerablemente con respecto a otros estándares previos. Esta característica suele ser bastante útil cuando se aplica en las siguientes situaciones:

1. Movimientos que se repiten periódicamente.
2. Interpretación de movimientos y obstrucciones.
3. Alternancia entre distintos ángulos de una cámara que van y vienen entre dos escenas.

Con esto se consigue reducir enormemente el tamaño necesario para almacenar la información. En los estándares anteriores, la referencia a una única imagen necesitaba mucho espacio cuando se producía una clara variación en el movimiento. Ahora, al permitir realizar búsquedas es múltiples imágenes adyacentes, aumenta la probabilidad de que se encuentre coincidencia; con lo que se necesita mucha menos información, puesto que la variación de movimiento puede ser prácticamente nula.

3. GPU y su programación con CUDA

La creciente necesidad de cómputo en los actuales sistemas gráficos por ordenador ha propiciado la aparición de dispositivos que pueden ser utilizados para acelerar otras aplicaciones más generales y que tienen como característica común su gran paralelismo e intensa carga computacional.

Las características principales de este tipo de dispositivos son una gran cantidad de elementos de proceso, o cores, integrados en un único circuito integrado a costa de una importante reducción de su memoria cache. Por lo tanto su arquitectura está altamente segmentada y se requiere un modelo de programación SIMD para obtener el máximo partido de sus características.

Las características arquitecturales de las GPU vienen determinadas por las aplicaciones gráficas, que son diferentes a las necesarias en las CPU del computador. En particular, en los procesos gráficos existe un alto paralelismo en las operaciones y en el acceso a los datos, tanto local como temporal, por lo que no necesita ni la complejidad de las actuales CPU, aunque sí su potencia aritmética, ni sus complejos sistemas de memoria cache.

Las empresas que lideran actualmente el mercado de GPUs son NVIDIA y ATI/AMD. De ellas, NVIDIA es la líder actual en el campo de la GPGPU debido a su bajo precio y el uso de CUDA (Compute Unified Device Architecture), que permite la programación de su gama más potente de tarjetas (a partir de la serie 8, incluyendo GeForce, Quadro y la línea Tesla) de forma similar a la programación en C, manteniendo la compatibilidad binaria entre ellas y acercando su modelo de programación a la de los programadores no expertos en lenguajes específicos para GPU como OpenGL y Cg.

4.1. CUDA

Es habitual emplear CUDA [6] para utilizar las GPU de NVIDIA como coprocesador para acelerar ciertas partes de un programa, generalmente aquellas con una elevada carga computacional por dato, ya que es en este tipo de cálculos donde más rendimiento se suele obtener.

En CUDA, los cálculos se distribuyen en una malla o grid de bloques de *threads*, donde todos los bloques tienen el mismo tamaño (número de threads). Estos threads son los que ejecutan el código de la GPU, conocido como *kernel*. Las dimensiones de la malla y de los bloques que contiene deben ser cuidadosamente escogidas para obtener el máximo rendimiento posible, basándose habitualmente en el problema específico a tratar.

En general, se puede ver un grid como una representación lógica de la propia GPU, un bloque como un procesador multinúcleo y un thread como uno de estos núcleos de proceso. Cada multiprocesador ejecuta una serie de bloques en segmentos de tiempo y un bloque siempre se ejecuta en un mismo multiprocesador. Los threads de un bloque se agrupan en *warps*, de modo que todos los threads de un *warp* en todo momento ejecutan la misma instrucción, habitualmente sobre distintos datos. En dispositivos de *'compute capability'* 1.0, como lo es la GeForce 8800 GTX, un *warp* está computesto por 32 threads.

Los threads de un mismo bloque pueden cooperar a través de 16KB de memoria compartida de rápido acceso y se dispone de mecanismos de sincronización a modo de “barrera” que permiten la cooperación de forma segura.

No ocurre lo mismo entre distintos bloques, donde no se dispone de mecanismo de sincronización a través de la memoria global (la DRAM del dispositivo).

4. Uso de GPU para la estimación y compensación de movimiento

En esta sección comentaremos inicialmente las dependencias que implica la paralelización de la estimación del movimiento para los diferentes modos de un MB, según el software de referencia secuencial de H.264 JM (versión 13.2) [7], para pasar a describir el trabajo llevado a cabo estos últimos meses para la implementación del codificador H264/AVC sobre la GPU de NVIDIA, utilizando CUDA. En concreto, hemos utilizado un computador con un procesador Intel Core 2 Duo E6550 a 2.33GHZ y 2GB de RAM, con una tarjeta GeForce G8800 GTX; y otro con un procesador Intel Core 2 Duo E8400 a 3.00GHz, con una tarjeta Tesla C870.

4.2. Uso de Predictores

Para el cálculo de la estimación y compensación de movimiento, es necesario el establecimiento de una zona de referencia o búsqueda. En el codificador de H.264 empleado, para poder situar la zona de referencia de manera análoga como se realiza en el codificador secuencial original, es necesario obtener un conjunto de predictores. Los predictores sirven para colocar la zona de referencia en la posición idónea para realizar la estimación y compensación de movimiento. Si no se utilizaran estos predictores la zona de referencia para un rango de búsqueda de 32 posiciones consistiría en ampliar 32 píxeles el MB en cada una de las 4 direcciones cartesianas, resultando un área de búsqueda de 80x80 píxeles. Así, para un MB de la imagen I_n con coordenadas origen (x,y), I_n $[x{:}x{+}16, y{:}y{+}16]$, la zona de referencia en la imagen anterior, I_{n-1}, vendrá dada por:

$$I_{n-1}\,[x{-}32{:}x{+}48, y{-}32{:}y{+}48].$$

En el caso de que se utilicen los predictores, ya no se establece la zona de referencia haciéndola coincidir con el entorno más cercano al MB, sino desplazada de acuerdo a la información proporcionada por éstos. Así, para el mismo MB de la imagen I_n con coordenadas origen (x,y), la zona de referencia en la imagen anterior, I_{n-1}, vendrá dada por:

$$I_{n-1}\,[(x{+}p_x){-}32{:}(x{+}p_x){+}48, (y{+}p_y){-}32{:}(y{+}p_y){+}48],$$

donde, p_x y p_y son los desplazamientos obtenidos por el predictor en la coordenadas X e Y para centrar la zona de búsqueda.

Los predictores se obtienen teniendo en cuenta el movimiento de la secuencia, esto es, se calculan teniendo en cuenta los vectores de movimiento obtenidos en los MBs adyacentes en el marco actual que ya han sido codificados (Figura 2), ya que suelen estar altamente correlacionados. Si se está en una codificación con varios modos de codificación activados, también se tienen en cuenta los de los modos previos.

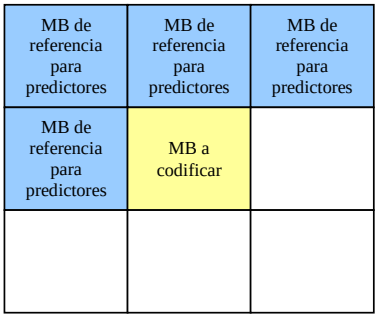

Figura 2. *Macrobloques (MB) de referencia para obtener los predictores.*

De esta manera se coloca el área de referencia de la manera más óptima, para que tanto los vectores de movimiento como los costes sean los mínimos posibles. Para ello se obtiene una función a minimizar que considera la suma de las diferencias absolutas (SAD) para el bloque $B_{ij}(n)$ con los bloques dentro del área de referencia en la imagen anterior [2]. El coste de cada una de estas diferencias tiene un coste computacional de orden $O(2R{+}1)^2$, donde R es el rango de búsqueda.

4.3. Adaptación de la estimación y compensación de movimiento a las GPU

La versión secuencial dispone de todos los datos para realizar la estimación y compensación de movimiento, con lo que simplemente realiza para cada modo y sub-modo un bucle con 4225 iteraciones (para un rango o ventana de búsqueda de 32 píxeles), obteniéndose un coste asociado para cada posición. A partir de estos, se selecciona como mejor candidato del modo o sub-modo aquel con el mínimo coste.

En las distintas implementaciones paralelas que se llevaron a cabo en la GPU, ya no se dispone de los datos tal y como se disponen en la versión secuencial, con lo que es necesario proporcionárselos antes de realizar la estimación y compensación de movimiento.

En la primera versión completa que se ha desarrollado utilizando la GPU, se ha sustituido el bucle que realiza la estimación y compensación de movimiento, por una llamada al *kernel* de la GPU. En dicha llamada, se incluyen los parámetros adecuados para situar la estimación de movimiento: es necesario identificar tanto el bloque actual que se está codificando, como el modo, y los predictores para cada modo. Toda la información necesaria para la codificación de una imagen, y que no se modifica durante la codificación de la misma, se transmite a la GPU al inicio de cada imagen (coste en bits de los vectores, espirales de búsqueda, imagen de referencia e imagen actual, variables diversas, etc.); la imagen actual y la de referencia se copian a la memoria de texturas. Cada modo o sub-modo de codificación del H.264 provoca una llamada al *kernel*, el MB sobre el cual se va a realizar la estimación y compensación de movimiento, es extraído de la memoria de texturas y copiado en la memoria compartida de cada multiprocesador, con el fin de optimizar los accesos. Esta forma de proceder es muy lenta porque el número de llamadas a la GPU crece a medida que se introducen más modos o sub-modos en la

codificación. El resultado, en términos calidad/caudal, es completamente igual al obtenido por el codificador secuencial, puesto que se dispone de todos los datos totalmente actualizados. Las prestaciones obtenidas, tanto de calidad/caudal como el fichero codificado en H.264, son idénticas a las ofrecidas por el codificador secuencial, excepto por los tiempos de codificación, que son significativamente mayores en la tarjeta gráfica. Las Tablas 1 y 2 muestran los tiempos totales, dependiendo del número de modos o sub-modos activos, obtenidos en la codificación de 20 imágenes en formato CIF de las secuencias Akiyo (más estática) y Foreman (con mayor movimiento), respectivamente. El modo o sub-modo listado en la fila n, incluye todos los anteriores, salvo el 16x16, que no tiene modo superior.

Modo	CPU	GPU GTX
16x16	3.563	5.828
16x8	6.220	13.734
8x16	8.749	21.595
8x8	11.049	36.376
8x4	13.049	64.766
4x8	15.126	92.500
4x4	18.467	147.373

Tabla 1. *Tiempos obtenidos (en segundos) de la primera versión sobre la GPU y la CPU para 20 imágenes de la secuencia Akiyo (CIF)*

Modo	CPU	GPU GTX
16x16	4.389	5.832
16x8	8.435	13.748
8x16	12.782	21.625
8x8	17.690	36.345
8x4	21.812	64.364
4x8	25.811	91.923
4x4	30.046	146.546

Tabla 2*. Tiempos obtenidos (en segundos) de la primera versión sobre la GPU y la CPU para 20 imágenes de la secuencia Foreman (CIF)*

En la segunda versión desarrollada, hasta el momento sólo están incorporados los modos 16x16, 16x8 y 8x16. En esta versión se lanza un *kernel* por cada MB, sea cual sea el número de modos activados para la codificación. La llamada a la GPU devuelve un array de costes por cada uno de los modos y sub-modos que hayan sido activados. Estos arrays están compuestos por todos los costes obtenidos por la GPU para todas las posibles posiciones. Por ejemplo, en las simulaciones realizadas, activando los 3 modos implementados, se devuelven 5 arrays (1 correspondiente al modo 16x16, 2 al modo 16x8 y 2 al modo 8x16) con 4225 entradas, que constituyen todas las posiciones de la espiral de búsqueda con un área de búsqueda de 32 píxeles, fijada en el fichero de configuración incluido con el software de referencia H.264 JM. Al devolver el control a la CPU, ésta calcula cuál es el mejor coste para el modo 16x16 y para cada uno de los cuatro sub-modos restantes (dos 16x8 y dos 8x16), quedándose con 5 posiciones y con sus 5 costes asociados. La zona de referencia se aproxima lo más posible a la empleada por el codificador secuencial, para minimizar las diferencias en cuanto a costes mínimos de los modos y sub-modos de codificación.

Esta aproximación no genera resultados idénticos al codificador secuencial, pues al ejecutar paralelamente el código, los predictores no están actualizados, es decir, no pueden utilizarse los vectores de movimiento obtenidos de los modos de codificación anteriores dentro del MB actual. Esto se debe a que dentro de la GPU no se resuelve cuál es el menor coste, ya que el procesado de los costes lo realiza posteriormente la CPU cuando se le devuelve el control. Por ello es necesario, a parte de la información transferida al inicio de cada imagen a la GPU (como se hizo en la versión anterior), la transferencia de los predictores de movimiento correspondientes a los modos que se van a usar al inicio de cada MB.

Para el modo 16x16, los resultados de la CPU y la GPU son idénticos en cuanto a calidad, caudal, etc., sólo se diferencian en el tiempo (4.389 seg. para la CPU, mientras que 5.627 seg. para la GPU); al haber sólo un modo los predictores estarán actualizados en cada invocación al *kernel* de la GPU.

Cuando se incluye el sub-modo 16x8, los predictores ya no estarán totalmente actualizados, no se realiza la actualización entre modos, con lo que se introducen diferencias entre la codificación secuencial y la que realiza la GPU. En la Tabla 3 se muestran las diferencias en la codificación de 20 imágenes de la secuencia Foreman, que es la escena con mayor movimiento.

	CPU	GPU
PSNR(Y)	36.19 dB	36.19 dB
PSNR(U)	38.65 dB	38.64 dB
PSNR(V)	42.22 dB	42.22 dB
Bits totales	356816 bits	356432 bits
Bit rate	535.22 bps	534.65 bps
Tiempo (GTX)	8.435 seg.	7.015 seg.
Tiempo (Tesla)	5.983 seg.	4.743 seg.
Velocidad (GTX)	2.39 fps	2.85 fps
Velocidad (Tesla)	3.34 fps	4.22 fps

Tabla 3. *Resultados obtenidos de la segunda versión con los modos 16x16 y 16x8 sobre la GPU y la CPU para 20 imágenes de la secuencia Foreman CIF*

Finalmente, en la Tabla 4 se muestran los resultados de la codificación de 20 imágenes de la secuencia Foreman (CIF) con los modos 16x16, 16x8 y 8x16. Con esta propuesta, se reduce el tiempo necesario para la codificación con respecto a la versión secuencial, con valores de calidad (medidos en términos de PSNR en las tablas) muy similares. Además, conforme se añadan nuevos modos y sub-modos la ganancia será mayor; puesto que la mayor sobrecarga viene debida a la transferencia de información hacia la GPU, y esta se lleva a cabo cuando se computa el primer modo.

4.4. Trabajo Futuro

Existen diversas posibilidades que estamos teniendo en cuenta para continuar con el trabajo que hemos iniciado, y que ha sido desarrollado en este artículo. La primera de ellas consiste en seguir incluyendo el resto de modos y

sub-modos de la codificación, en la segunda versión implementada, que ha sido comentada en el punto anterior.

	CPU	GPU
PSNR(Y)	36.23 dB	36.24 dB
PSNR(U)	38.68 dB	38.67 dB
PSNR(V)	42.23 dB	42.24 dB
Bits totales	340912	342304
Bit rate	511.37 bps	513.46 bps
Tiempo (GTX)	12.686 seg.	8.344 seg.
Tiempo (Tesla)	9.89 seg	6.118 seg.
Velocidad (GTX)	1.58 fps	2.40 fps
Velocidad (Tesla)	2.02 fps	3.27 fps

Tabla 4. *Resultados obtenidos de la segunda versión con los modos 16x16, 16x8 y 8x16 sobre la GPU y la CPU para 20 imágenes de la secuencia Foreman CIF.*

La segunda vía de trabajo consiste en modificar la segunda versión desarrollada, para que entre los modos se obtenga el mejor vector de movimiento sin devolver el control a la CPU. Sería necesario, por tanto, parar todos los hilos excepto uno, que sería el encargado de procesar y obtener el mejor de los predictores para cada modo. Obtendríamos la misma zona de referencia que el codificador secuencial y ya no sería necesario que la CPU procesara cuál ha sido el mejor modo, puesto que la GPU se encargaría de esta tarea. Así, la GPU obtendría los datos de los costes para cada una de la posiciones en la zona de referencia y, además, procesaría dichos costes y devolvería el mejor para cada modo. De este modo dispondríamos de un codificador cuyo resultado sería idéntico al original, pero desconocemos, a día de hoy, cuál es el coste computacional de las sincronizaciones introducidas para obtener los vectores de movimiento después de cada modo. Esta posibilidad implicaciones sincronizaciones que pueden ser problemáticas

En una tercera opción, barajamos la posibilidad de subir la paralelización a más alto nivel, es decir, reestructurar el código para poder lanzar a codificar varios MBs al mismo tiempo. En esta aproximación, el problema sigue siendo la obtención de los predictores de movimiento, puesto que no podríamos obtener los correspondientes a los MBs adyacentes, puesto que se estarían codificando varios MBs contiguos simultáneamente. Esta solución será la más costosa en cuanto a tiempo de implementación, puesto que conllevará una profunda reestructuración del código secuencial para llevarla a cabo.

5. Conclusiones

La primera propuesta implementada, en la que se realizaba una llamada al *kernel* de la GPU por cada modo y sub-modo configurado, para cada MB, no es adecuada para acelerar el codificador de referencia H.264 JM, debido a que arrancar y detener la GPU para cada modo y sub-modo es demasiado costoso en cuanto a tiempo, no pudiendo ser contrarrestada por la codificación en paralelo que se realiza dentro del *kernel*.

La segunda propuesta parece ser más adecuada, ya que toda la sobrecarga que se introduce, debido a la transferencia de información necesaria para realizar la codificación, más la sobrecarga de arrancar y detener la GPU una vez por MB, es contrarrestada por la ejecución en paralelo de la estimación y compensación de movimiento, obteniendo resultados claramente mejores, en cuanto a tiempo de proceso, de los que obtiene el codificador secuencial. No se obtienen los mismos resultados en cuanto a calidad, tamaño en bits de la secuencia codificada, etc., pero éstos son muy similares.

Agradecimientos

Este trabajo ha sido financiado mediante el programa europeo FEDER, el proyecto del MEC “Consolider Ingenio-2010 CSD 2006-00046" y “TIN 2006-15516-C04-02", y los proyectos de la Junta de Comunidades de Castilla-La Mancha PCC08-0078 y PAI06-106.

Referencias

[1] ISO/IEC 14496-10:2005, – Information technology – Coding of audio-visual objects – Part 10: Advanced Video Coding.

[2] ITU-T RECOMMENDATION H.264, Advanced Video Coding for Generic Audiovisual Services, May 2003.

[3] ISO/IEC JTC1 SC29 WG11 13818:1994, Generic Coding of Moving Pictures and Associated Audio.

[4] ISO/IEC 14486-2 PDAM1:1999, Information Technology-Generic Coding of Audio-Visual Objects- Part 2: Visual.

[5] G.J. Sullivan, P. Topiwala, and A. Luthra, The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions, SPIE Conference on Applications of Digital Image Processing XXVII, Special Session on Advances in the New Emerging Standard: H.264/AVC, August 2004.

[6] NVIDIA CUDA Compute Unified Device Architecture Programming Guide, http://www.NVIDIA.com/ object/cuda_develop.html. Última visita, 27 de septiembre de 2008.

[7] Joint Video Team (JVT) of ISO/IEC MPEG and ITU-T VCEG, Reference Software to Committee Draft. JVT-F100 JM14.0. 2008.

[8] C.-W. Ho, O.C Au, S.-H. Gary Chan, S.-K. Yip, H.-M. Wong, Motion Estimation for H.264/AVC using Programmable Graphics Hardware**,** IEEE Int. Conf. on Multimedia and Expo, 9-12 July 2006, pp. 2049 – 2052.

[9] R..X. Chen, J. Fan, Complexity reduction for SOPC-based H.264/AVC coder via sum of absolute difference, IEEE/CIE 7th. Int. Conf. On ASIC, Oct. 2007, pp. 1277-1280.

[10] Wei.Nien Chen, Hsueh-Ming Hang, H.264/AVC Motion Estimation Implementation on Compute Unified Device Architecture (CUDA) IEEE Int. Conf. on Multimedia and Expo, 9-12 July 2008, pp. 697– 700.