Transformada Wavelet 3D en GPUs

Vicente Galiano, Otoniel López, Manuel P. Malumbres y Hector Migallón [1]

Resumen— **Se presentan diversos algoritmos e implementaciones de la transformada Wavelet sobre secuencias de video que denominaremos 3D-DWT (*three-dimensional wavelet transform*). Dicha transformada se utiliza como técnica de compresión en secuencias de video en medicina e inserción de marcas de agua en vídeo. Presentamos unas implementaciones sobre procesadores gráficos (GPU) y las compararemos con los tiempos obtenidos en una CPU. Los dispositivos gráficos utilizados en las pruebas han sido una tarjeta GTX280 y una GMT540. Haciendo uso únicamente de la memoria global del dispositivo, los resutados obtenidos nos muestran un speedup de 19.7 y 10.65 para los dispositivos GTX280 y GMT540. Por otro lado, la utilización de la memoria compartida por los hilos en cada bloque nos permite una mejora considerable del speedup llegando a valores de 87 y 25 respectivamente, con respecto a la simulación secuencial en CPU. Un parámetro influyente en estos tiempos es el número de tramas en la transformada temporal GOP (*Group of Pictures Size*).**

Palabras clave— **wavelet transform, image coding , video coding, parallel algorithms, CUDA, GPU**

I. Introducción

En áreas como la creación de marcas de agua en video [2] y la codificación 3D (por ejemplo en la compresión de datos volumétricos médicos [17] o imágenes multiespectráles[4], codificación de modelos 3D [1], y codificación de vídeo [7]) se ha adoptado la codificación de video en subbandas 3D basada en la transformada wavelet en tres dimensiones (3D-DWT) como una alternativa a la compresión tradicional basada en la transformada discreta del coseno (DCT). La codificación de subbandas 3D utiliza la transformada discreta wavelet en lugar de la transformada DCT, obteniendo una mejor compactación de la energía.

En [15], Podilchuk et al. utilizaron la descomposición en subbandas 3D espacio-temporal y la cuantificación de vector geométrico (GVQ). Taubman y Zajqr presentaron un codificador de vídeo en color completo basado en codificación sub-banda 3-D con compensación de cámara panorámica [19]. El algoritmo wavelet embebido de árbol cero (EZW, *Embedded Zerotree Wavelet*) desarrollado por Shapiro [18], el algoritmo de conjunto de particiones en árboles jerárquicos (SPIHT) desarrollado por Said y Pearlman [16] y el codificador wavelet de árbol inferior (LTW) desarrollado por Oliver y Malumbres [14] explotan las similitudes entre subbandas wavelet diferentes en función de la estructura del árbol wavelet. Estos algoritmos ofrecen un rendimiento muy bueno en las imágenes 2-D exigiendo baja carga computacional. Versiones adaptadas de un codificador de imagen se pueden utilizar teniendo en cuenta la nueva dimensión temporal. Por ejemplo, el método embebido de árbol cero 2D (IEZW) se ha extendido a IEZW-3D para codificación de vídeo por Chen y Pearlman [3], y muestra un esperanzador un sistema de codificación de vídeo computacionalmente efectivo y simple sin compensación de movimiento, obteniendo excelentes resultados numéricos y visuales. Un codificador 3D de cero árbol a través de un EZW modificado también se utilizó con buenos resultados en la compresión de imágenes volumétricas [13] . En [7] y [12] , en lugar de los típicos árboles de codificación de imagen de cuatro descendientes, se utiliza un árbol con ocho descendientes por coeficiente para extender tanto los codificadores de imagen SPIHT y LTW a video en 3D.

La amplia investigación por parte de la comunidad científica, ha logrado acelerar la DWT, en especial en 2D, tanto la explotación de arquitecturas multinúcleo como en unidades de procesamiento gráfico (GPU). En [21], un algoritmo SIMD (*Single Instruction, Multiple Data*) ejecuta la transformada 2D-DWT en una GeForce 7800 GTX con Cg y OpenGL, con una notable aceleración. Un esfuerzo similar se ha realizado en [20] mediante la combinación de Cg en la tarjeta 7800 GTX obteniendo finalmente de un 1,2 a 3,4 de aceleración frente a una implementación en CPU. En [5], los autores presentan una implementación en CUDA para la obtención de la transformada 2D-FWT más de 20 veces más rápido que la versión C secuencial en CPU, y más del doble de rápida que una ejecución optimizada con OpenMP y Pthreads implementa en CPUs multinúcleo . En [6], los autores presentan implementaciones GPU para la obtención de la transformada 2D-DWT con speedups superiores a 20 cuando se compara con el algoritmo secuencial en CPU.

El resto del trabajo se organiza de la siguiente manera. La sección II presenta los fundamentos de la 3D-DWT. La sección III se centra en los detalles de la programación de la GPU con CUDA, y en la sección IV se analiza el rendimiento y se comparan los resultados obtenidos. Finalmente en la sección VI se presentan las conclusiones.

II. 3D Wavelet Transform

La DWT es un esquema de descomposición multiresolución de señales digitales de entrada, véase la descripción detallada en [9]. En primer lugar, la señal fuente se descompone en dos sub-bandas de frecuencia, de baja frecuencia (*low-pass*) y la subbanda de alta frecuencia (*high-pass*). Para la DWT clásica, la descomposición de una señal se lleva a cabo mediante filtros digitales paso bajo H y paso alto G. Ambos filtros digitales se obtienen a partir de la función de escalado $\Phi(t)$ y las funciones wavelet cor-

[1]Departamento de Física y Arquitectura de Computadores. Universidad Miguel Hernández. 03203 Elche. e-mail: {vgaliano,otoniel,mels,hmigallon}@umh.es

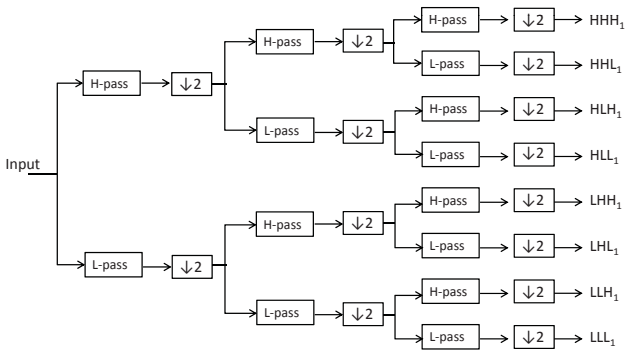


Fig. 1. Visión general de la computación en 3D-DWT en una descomposición de un nivel con el algoritmo de 3D-DWT.

respondientes a escalas de frecuencia diferentes $\Psi(t)$. El sistema disminuye la resolución de la señal a la mitad de los resultados filtrados en el proceso de descomposición. Si se consideran filtros no recursivos y de cuatro etapas, las funciones de transferencia de H y G pueden ser representadas de la siguiente manera:

$$H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} \quad (1)$$

$$G(z) = g_0 + g_1 z^{-1} + g_2 z^{-2} + g_3 z^{-3} \quad (2)$$

Una secuencia de vídeo es una extensión de imágenes 2D a lo largo del eje de tiempo. En la forma más común de codificación de vídeo wavelet, un Transformada Discreta Wavelet (DWT) tridimensional se aplica sobre un grupo de imágenes de vídeo (GOP, *Group Of video Pictures*). El 3D-DWT es una combinación de una DWT 2D espacial y una DWT 1D temporal, donde la DWT temporal absorbe el movimiento en el GOP. La DWT temporal se lleva a cabo en los valores de los píxeles de la misma localidad a lo largo del eje de tiempo. La Figura 1 muestra una descomposición de un nivel de la DWT en 3D donde *H-pass* y *L-pass* representan los filtros de paso alto y paso bajo, respectivamente. La disminución de la resolución de los resultados filtrados se denota como '↓ 2'. Después de la descomposición, se obtienen las ocho primeras subbandas wavelet (normalmente denominadas como LLL_1, LHL_1, LLH_1, LHH_1, HLL_1, HHL_1, HLH_1, HHH_1). Posteriormente, se puede realizar la misma descomposición centrándose en la sub-banda de baja frecuencia (LLL_1), obteniendo de esta manera una descomposición wavelet de segundo nivel, y así sucesivamente.

En este trabajo vamos a utilizar los filtros biortogonales Daubechies 9/7 tanto para las descomposiciones espaciales como temporales. En [6], los autores utilizan los bancos de filtros indicados para convolución y la transformada LTW en el caso 2D-DWT. En el análisis de la GPU de dicho trabajo, se implementa la convolución el 2D-DWT en el proceso de filtrado y se obtienen speedups superiores a 20 con respecto a la aplicación secuencial en CPU. En la siguiente sección, extenderemos este trabajo al caso 3D-DWT tratando de obtener resultados similares que en el caso 2D-DWT.

Como en [6], hemos realizado la convolución basada en 2D-DWT utilizando dos espacios en la memoria global del dispositivo. El tamaño de la memoria adicional dependerá del tamaño de la resolución del vídeo y del GOP. Además deberemos almacenar los píxeles correspondientes a la extensión simétrica que en el caso del filtro Daubechies 9/7 se corresponde con una extensión de cuatro elementos en todas las fronteras.

III. GPU 3D Wavelet Transform

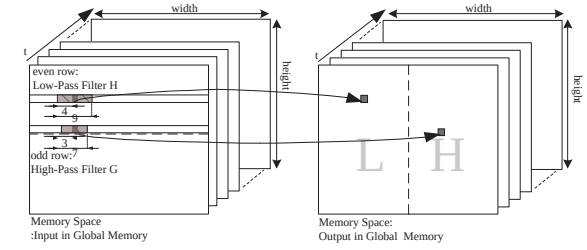
En esta sección, ofrecemos un breve resumen de las GPUs y exponemos el diseño y la implementación de nuestro propuesta de transformada 3D-DWT realizada con CUDA que consigue disminuir los tiempos de ejecución de manera significativa.

IV. Evaluación del Rendimiento

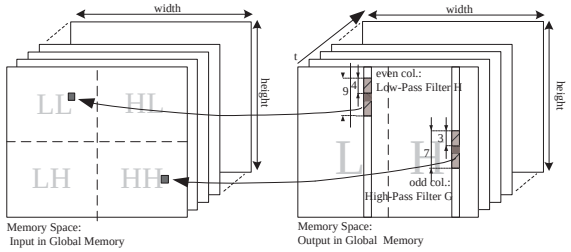
El lanzamiento de la intefaz de programación CUDA de NVIDIA [10][11][8] para los desarrolladores ha llevado a un espectacular aumento del interés en el uso de las capacidades de la GPU para conseguir una computación más rápida y más eficiente. La GPU funciona como un coprocesador de la CPU a través de la API de CUDA y explota el paralelismo masivo de datos SIMD de la arquitectura de la GPU. Cada hilo de ejecución se ejecuta de forma independiente en los núcleos de la GPU. Por otro lado, el intercambio de datos entre hilos del mismo bloque mediante la memoria compartida ofrece un mecanismo de comunicación de alta velocidad. Cabe señalar que para obtener mayor ancho de banda y un aumento del rendimiento en general, el intercambio de memoria entre hilos deben ser optimizado con una programación muy cuidadosa para asegurar la menor latencia de acceso a memoria entre las lecturas/escrituras. Sin una coordinación eficaz , toda la optimización podría verse comprometida impidiendo las mejoras en el rendimiento.

En este trabajo hemos utilizado dos GPUs diferentes. La primera es una GTX280, que contiene 240 núcleos CUDA con 1 GB de memoria de vídeo dedicada. También se muestran los resultados con una GPU de un ordenador portátil (GMT540) con 96 núcleos CUDA y 2 GB de memoria de vídeo. Podemos apreciar diferencias significativas entre los dos dispositivos que se verán reflejadas en los resultados mostrados en la sección IV y V. Con el fin de comparar el rendimiento de la CPU y la GPU, hemos implementado una versión secuencial de 3D-DWT que se ejecuta en un procesador Intel Core 2 Quad Q6600 de 2,4 GHz.

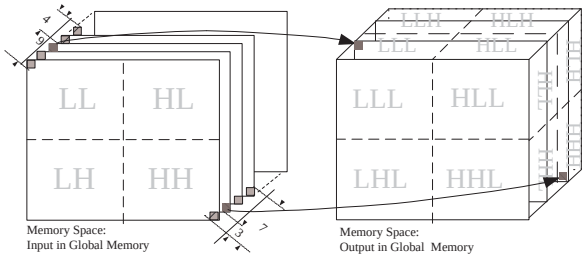
El algoritmo utilizado para calcular la transformada 3D-DWT en la GPU se ilustra en la Figura 2. Antes del cálculo en el primer paso, los datos del grupo de fotogramas de video deben ser transferidos desde la memoria principal a la memoria global del dispositivo. Hay que transferir el número de fotogramas que indica el tamaño del GOP. A medida que aumentamos el tamaño del GOP se requiere una mayor cantidad de memoria global en la GPU. Todas los



(a) Primer paso: Filtrado por filas



(b) Segundo paso: Filtrado por columnas



(c) Tercer paso: Filtrado en el tiempo

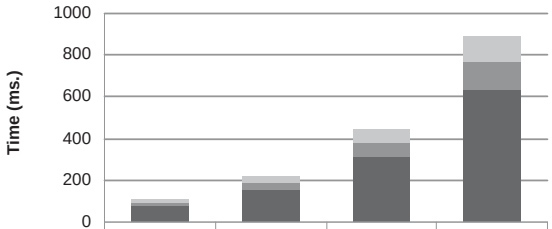
Fig. 2. Pasos para la computación 3D-DWT en GPUs

fotogramas se almacenan en posiciones de memoria adyacentes. De esta manera, los requisitos de memoria de la GPU es de $Width \times Height \times frames \times sizeof(float)$ bytes. Como se muestra en la Figura 2, en esta implementación estamos utilizando dos espacios de memoria en la memoria global de la GPU: una para los datos de entrada y otro para los datos de salida después de aplicar el proceso de filtrado. En el primer paso (véase la Figura 2(a)), cada hilo calcula la convolución de fila de un pixel y almacena el resultado en la memoria de salida. Para computar el segundo paso, el origen de datos es ahora el espacio de memoria de salida obtenido en el paso anterior, por lo que no se necesita una copia de datos de memoria para la preparación de este paso. En el segundo paso (véase la Figura 2(b)), se aplica el filtrado por columna y la transformada 2D-DWT se completa para cada fotograma de la secuencia. El resultado de este segundo paso se almacena en el espacio de memoria de origen del primer paso. Finalmente, en el tercer paso, se realiza una transformada 1D-DWT para filtrar el eje temporal. Al finalizar, los datos deben ser transferidos a la memoria del host para proceder con la siguiente secuencia de imágenes.

En esta sección se presenta el rendmiento obtenido de la transformada 3D-DWT sobre las GPU en términos de tiempos de transferencia y de cálculo y los compararemos con los tiempos obtenidos cuando se ejecuta el algoritmo secuencial sobre una CPU. Los resultados han sido obtenidos sobre las plataformas menciadas anteriormente GTX280 y GMT540.

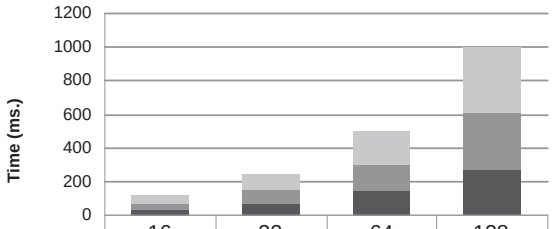
En la Figura 3 se presentan los tiempos de transferencia de memoria y de cómputo para ambas plataformas GPU utilizadas en este trabajo. Además mostramos resultados para dos resoluciones diferentes de secuencia de vídeo considerando una variación del GOP de 16 a 128. Las etiquetas utilizadas *TxG2C*, *TxG2G*, *TxC2G* y *T.GPU* en la Figura 3 se refieren al tiempo de transferencia de datos desde la memoria de la GPU a la de CPU, tiempo de transferencia de memoria entre espacios de memoria de GPU, tiempos de transferencia desde CPU a GPU y tiempo de cómputo en GPU respectivamente. Como se muestra en las Figuras 3(a) y (b) para una resolución de fotograma de 1280x640, la GTX280 es 2,3 veces más rápida que la GMT540 en relación con el tiempo de cómputo en GPU. Ésto se debe principalmente al mayor número de núcleos disponibles en la GTX280 (2,5 veces más núcleos). Sin embargo, el tiempo global de cálculo incluyendo el tiempo de transferencia entre memoria es menor debido a que la transferencia de memoria es significativamente menor en la GMT540, gracias a una segunda generación de bus PCI Express que mejora la transferencia de datos. Similares conclusiones se pueden extraer para la resolución de fotogramas de 1920x1024 en las Figuras 3(c) y (d), donde los tiempos de transferencia entre memorias de la CPU y la GPU son 3,3 veces mayor en promedio en la GTX280 convirtiéndose en este caso en un cuello de botella. Tal y como se observa en la Figura 3(d), el tiempo de computación para los tamaños GOP iguales a 64 y 128 en la GTX280 no están disponibles porque este dispositivo tiene sólo 1 Gbyte de memoria de vídeo. Por otro lado, la GPU GMT540 dispone de 2 GB y sí es capaz de realizar su ejecución. Como se muestra en la Figura 3, el tiempo de transferencia de datos entre regiones de la memoria global de la GPU es insignificante.

En la Figura 4 se representa el speedup obtenido para la implementación basada en GPU con respecto al algoritmo secuencial sobre CPU. Se presenta el speedup para ambas plataformas GPU utilizando diferentes tamaños del GOP para la resolución de vídeo 1280x640 para la GPU GTX280 y 1920x1024 para la GPU GMT540. Tal y como se muestra, la



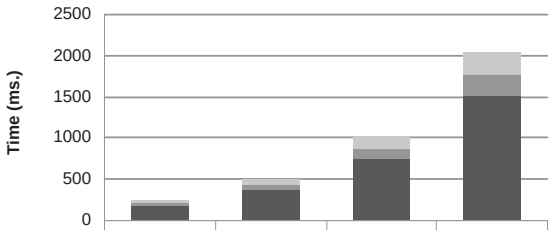
	16	32	64	128
TxG2C	15.867	28.129	61.758	121.858
TxG2G	0.717	0.760	1.142	0.818
TxC2G	14.521	30.207	60.538	133.674
T.GPU	76.216	156.026	315.709	633.077

(a) 1280x640 in GMT540



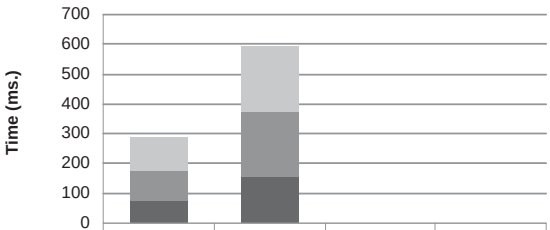
	16	32	64	128
TxG2C	47.897	95.644	194.910	390.902
TxG2G	0.304	0.307	1.523	0.297
TxC2G	40.715	81.277	164.135	339.905
T.GPU	34.001	67.983	141.813	273.448

(b) 1280x640 in GTX280



	16	32	64	128
TxG2C	30.776	64.180	131.197	265.338
TxG2G	1.410	1.399	1.469	1.464
TxC2G	32.647	66.692	134.552	271.611
T.GPU	179.809	369.250	747.479	1502.521

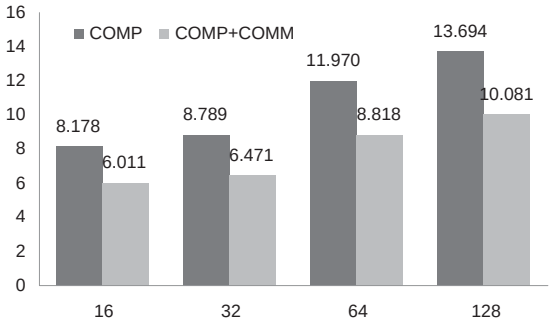
(c) 1280x640 in GMT540



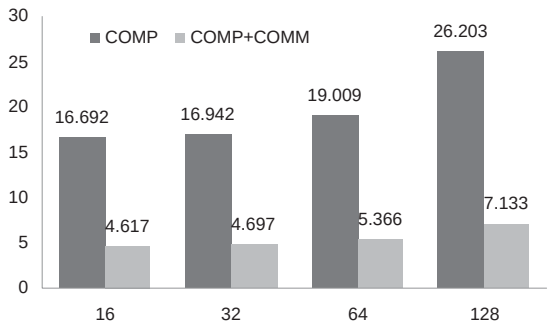
	16	32	64	128
TxG2C	110.270	222.999		
TxG2G	0.455	0.453		
TxC2G	98.060	212.998		
T.GPU	77.118	157.669		

(d) 1280x640 in GTX280

Fig. 3. Tiempos de transferencia y computación en GMT540 y GTX280 para diferentes resoluciones de vídeo



(a) GMT540-1920x1024



(b) GTX280-1280x640

Fig. 4. Speedups obtenidos en las GPUs sin y con comunicaciones (COMP y COMP+COMM respectivamente)

plataforma GTX280 obtiene mejores speedups que la plataforma GMT540, siendo hasta 2 veces mejor para el tamaño de GOP de 128. No obstante, tal y como se mencionó anteriormente, los tiempos de comunicaciones entre memorias de la CPU y la GPU reducen drásticamente el rendimiento obtenido en la plataforma GTX280. También se muestra en la Figura 4, , cómo el speedup aumenta a medida que aumenta el tamaño del GOP para ambas plataformas GPU, incluso teniendo en cuenta el tiempo de comunicación. Como puede observarse, la transferencia de datos entre la memoria del dispositivo y la memoria principal introduce una penalización importante en el uso de la GPU para una computación de propósito geneal. Sin embargo, esta penalización se podría evitar si se superponen las comunicaciones y el cálculo de los filtros de forma concurrente. Debemos tener en cuenta que la transformada de wavelet 3D es sólo el primer paso para desarrollar una aplicación de alto nivel como el que se menciona en la sección I.

V. Optimización del acceso a memoria

El algoritmo presentado en la sección anterior, utiliza la memoria global para almacenar tanto los datos de entrada como los de la salida en cálculo de las filtros divididos en dos espacios de memoria.

Mediante esta implementación hemos obtenido un speedup razonable (x26) en las resoluciones altas de fotogramas. Sin embargo, podemos lograr un mejor rendimiento si utilizamos la memoria compartida de 16KB que dispone cada bloque de hilos en su ejecución. Cada bloque carga en la memoria compartida una porcion de una fila o columna (dependiendo del filtro) y de este modo, todos los hilos (*threads*) leen desde la memoria compartida. El número de bloques de hilos $NBLOCKS$ depende del tamaño de bloque y de la resolución de los fotogramas del vídeo. Hay que señalar que alrededor de cada marco del fotograma de vídeo debemos mantener un margen o borde que tambien se debe almacenar en la memoria compartida con el fin de filtrar adecuadamente el bloque del fotograma. Mediante una adecuada ejecución, se puede reducir el número de hilos en espera, reduciendo el número total de hilos por bloque y también el uso de cada hilo para cargar múltiples pixeles en la memoria compartida. Esto asegura que todos los hilos estén activos durante la etapa de cálculo. Debemos tener en cuenta que el número de hilos en un bloque debe ser un múltiplo del tamaño *warp* (32 hilos en GTX280 y GMT540) para obtener una eficiencia óptima. Para lograr una mejor eficiencia y un mayor rendimiento de memoria, los múltiples accessos a memoria deben unirse en una transacción de la memoria individual. Si todos los hilos dentro de un *warp* (32 hilos) leen al mismo tiempo posiciones consecutivas de memoria, la lectura se puede realizar a una velocidad óptima.

En este nuevo algoritmo, cada etapa de filtrado por fila/columna o temporal se separa en dos sub-etapas: en la primera etapa los hilos cargan un bloque de píxeles de una fila/columna o eje temporal desde la memoria global a la memoria compartida, y en la segunda etapa, cada hilo calcula el filtro sobre los datos almacenados en la memoria compartida y almacena los resultados en la memoria global. No debemos olvidar los casos en los que se está procesando un pixel que se encuentra cercano a uno borde que cada bloque cargado en memoria. En este caso, los hilos deben cargar en la memoria compartida los valores de pixeles adyacentes de otro bloques con el fin de calcular la salida de los los pixeles situados en las fronteras.

En la Figura 5, se evalúa el nuevo algoritmo que utiliza el acceso optimizado a la memoria compartida. Como podemos ver, en ambas GPU se ha reducido considerablemente el tiempo de ejecución. Por otro lado, el tiempo de transferencia entre CPU y GPU no se ve afectado por el uso de la memoria compartida. A modo de ejemplo, para una resolución de fotograma de vídeo de 1920×1024, en la GTX280 se obtiene un rendimiento tres veces superior al obtenido utilizando únicamente la memoria global. Además, si se comparan los nuevos tiempos de ejecución con respecto al algoritmo secuencial en la CPU, se obtiene un notable incremento en el speedup. Por ejemplo, para un tamaño de GOP de 128 se alcanza un speedup de 25 en el dispositivo GMT540 mientras que en la GPU GTX280 llegamos a alcanzar un speedup superior a 87 y 3,3 veces superior al obtenido utilizando únicamente la memoria global (véase la Figura 6).

VI. Conclusiones

En este trabajo, hemos presentado un algoritmo basado en CUDA que realiza la transformación wavelet discreta en 3D. Hemos analizado el comportamiento del algoritmo cuando se ejecuta en dos diferentes plataformas GPU. En una primera propuesta, se utiliza la memoria global de la GPU y se obtiene un incremento del rendimiento de hasta 26 veces superior en la GTX280 y de hasta 13,6 en GMT540 con respecto a una ejecución en una CPU para un tamaño de GOP de 128. Después de esto, se propone una segunda implementación que utilice y optimice el acceso a la memoria compartida de la GPU. Esta propuesta mejora notablemente el rendimiento en la GPU y se consiguen speedups de más de 87x cuando se utiliza un tamaño de GOP de 128. En ambas propuestas, la comunicaciones entre CPU y GPU y viceversa se convierten en un cuello de botella. Para evitar este cuello de botella centramos nuestro trabajo futuro en superponer comunicaciones y computación en la aplicación final para evitar esta penalización. De esta manera, se podría procesar un grupo de imágenes mientras que otro grupo está siendo transferido desde o hacia la memoria de la GPU.

Agradecimientos

El presente trabajo ha sido financiado por el Ministerio de Educación mediante el proyecto TIN2011-27543-C03-03 y por el Ministerio de Ciencia e Innovación mediante el proyecto TIN2011-26254.

Referencias

[1] M. Aviles, F. Moran, and N. Garcia, "Progressive lower trees of wavelet coefficients: Efficient spatial and SNR scalable coding of 3D models," *Lecture Notes in Computer Science*, vol. 3767, pp. 61–72, 2005.

[2] P. Campisi and A. Neri, "Video watermarking in the 3D-DWT domain using perceptual masking," in *IEEE International Conference on Image Processing*, September 2005, pp. 997–1000.

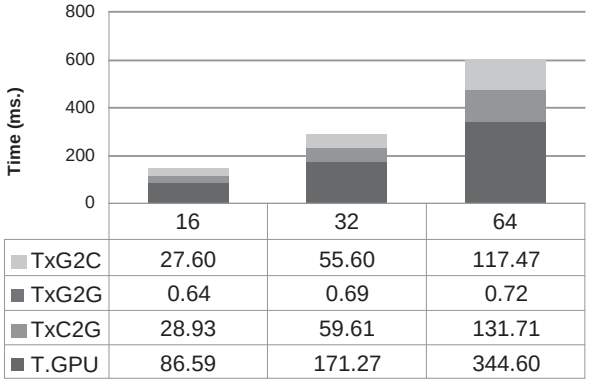
[3] Y. Chen and W. Pearlman, "Three-dimensional subband coding of video using the zero-tree method," in *Visual Communications and Image Processing*, vol. Proc. SPIE 2727, March 1996, pp. 1302–1309.

[4] P. Dragotti and G. Poggi, "Compression of multispectral images by three-dimensional SPITH algorithm," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 38, no. 1, pp. 416–428, January 2000.

[5] J. Franco, G. Bernabé, J. Fernández, M. Acacio, and M. Ujaldón, "The gpu on the 2d wavelet transform. survey and contributions," in *In proceedings of Para 2010: State of the Art in Scientific and Parallel Computing*, 2010.

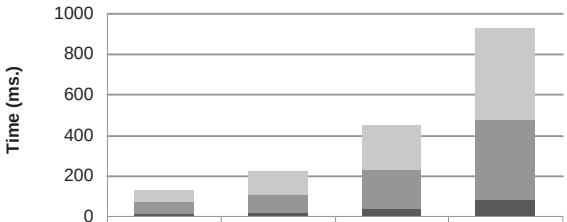
[6] V. Galiano, O. López, M. Malumbres, and H. Migallón, "Improving the discrete wavelet transform computation from multicore to gpu-based algorithms," in *In proceedings of International Conference on Computational and Mathematical Methods in Science and Engineering*, 2011.

[7] B. Kim, Z. Xiong, and W. Pearlman, "Low bit-rate scalable video coding with 3D set partitioning in hierarchical trees (3D SPIHT)," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 10, pp. 1374–1387, December 2000.



	16	32	64
TxG2C	27.60	55.60	117.47
TxG2G	0.64	0.69	0.72
TxC2G	28.93	59.61	131.71
T.GPU	86.59	171.27	344.60

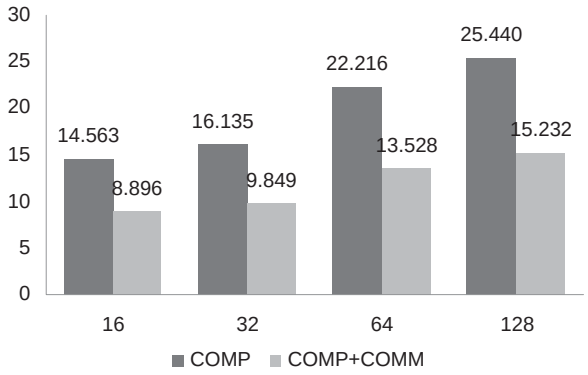
(a) 1280x640 in GMT540



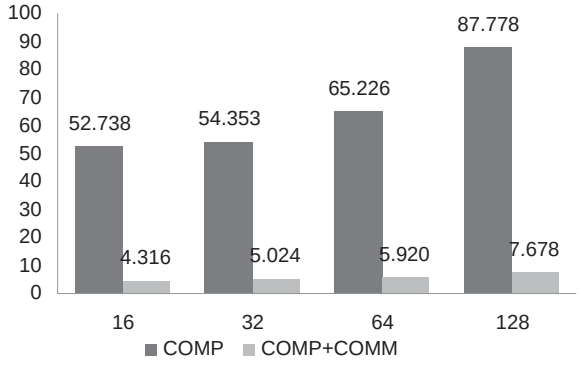
	16	32	64	128
TxG2C	56.91	116.35	222.61	450.77
TxG2G	0.39	0.36	0.38	0.44
TxC2G	63.44	91.34	191.05	400.35
T.GPU	10.76	21.19	41.33	81.63

(b) 1280x640 in GTX280

Fig. 5. Tiempos de transferencia y de Computación en GMT540 y GTX280 para una resolucion 1280x640 utilizando el acceso optmizado a la memoria compartida



(a) GMT540-1920x1024



(b) GTX280-1280x640

Fig. 6. Speed-up obtenido en las GPUs utilizando el acceso optimizado a la memoria compartida

[8] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "Nvidia tesla: A unified graphics and computing architecture," in *IEEE Micro*, vol. 28, no. 2, 2008, pp. 39–55.

[9] S. G. Mallat, "A theory for multi-resolution signal decomposition: The wavelet representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, July 1989.

[10] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," in *Queue*, vol. 6, no. 2, 2008, pp. 40–53.

[11] Nvidia Corporation, "Nvidia cuda c programming guide. version 3.2."

[12] O. Lopez, M. Martinez-Rach, P. Piñol, M. Malumbres, and J.Oliver, "Lower bit-rate video coding with 3D lower trees (3D-LTW)," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, 1998, pp. 3105–3108.

[13] J. Luo, X. Wang, C. Chen, and K. Parker, "Volumetric medical image compression with three-dimensional wavelet transform and octave zerotree coding," in *Visual Communications and Image Processing*, vol. Proc. SPIE 2727, March 1996, pp. 579–590.

[14] J. Oliver and M. P. Malumbres, "Low-complexity multiresolution image compression using wavelet lower trees," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 16, no. 11, pp. 1437–1444, 2006.

[15] C. Podilchuk, N. Jayant, and N. Farvardin, "Three dimensional subband coding of video," *IEEE Tran. on Image Processing*, vol. 4, no. 2, pp. 125–135, February 1995.

[16] A. Said and A. Pearlman, "A new, fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Transactions on Circuits, Systems and Video Technology*, vol. 6, no. 3, pp. 243–250, 1996.

[17] P. Schelkens, A. Munteanu, J. Barbariend, M. Galca, X. Giro-Nieto, and J. Cornelis, "Wavelet coding of volumetric medical datasets," *IEEE Transactions on Medical Imaging*, vol. 22, no. 3, pp. 441–458, March 2003.

[18] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, December 1993.

[19] D. Taubman and A. Zakhor, "Multirate 3-D subband coding of video," *IEEE Tran. on Image Processing*, vol. 3, no. 5, pp. 572–588, September 1994.

[20] C. Tenllado, J. Setoain, M. Prieto, L. Pinuel, and F. Tirado, "Parallel implementation of the 2d discrete wavelet transform on graphics processing units: Filter bank versus lifting," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 19, no. 3, pp. 299 –310, march 2008.

[21] T.-T. Wong, C.-S. Leung, P.-A. Heng, and J. Wang, "Discrete wavelet transform on consumer-level graphics hardware," *Multimedia, IEEE Transactions on*, vol. 9, no. 3, pp. 668 –673, april 2007.