Transformada paralela 3D-DWT para multicores

V. Galiano, O. López, M.P. Malumbres y H. Migallón [1]

Resumen— **La transformada 3D wavelet ha centrado la atención de la comunidad científica en varias áreas como el marcado de agua en vídeo, la compresión de datos médicos volumétricos, la codificación multiespectral de imagen, la codificación de modelos 3D y la codificación de vídeo. En este trabajo presentamos diversas estrategias para optimizar el cómputo de dicha transformada 3D en arquitecturas de memoria compartida. Presentamos resultados que, dependiendo del multiprocesador utilizado y del tamaño de GOP, ofrecen eficiencias entre el 95% con cuatro procesadores (o cores), y por encima del 83% usando hasta doce cores (o procesos). Por otra parte, la memoria extra necesaria en los algoritmos presentados está por debajo del 0.12% para vídeos de baja resolución, y por debajo del 0.017% para videos de alta resolución.**

Palabras clave— **transformada wavelet, codificación de vídeo, algoritmos paralelos, OpenMP.**

I. Introducción

La codificación 3D por subbandas hace uso de la transformada wavelet discreta (DWT), la cual proporciona una mayor compactación de la energía que la proporcionada por la DCT.

En [1] Podilchuk, et al., utilizan una descomposición por subbandas espacio-temporal en tres dimensiones y una cuantización vectorial geométrica. Por otro lado, en [2] Taubman y Zakhor presentan un codificador de vídeo a color basado en la codificación por subbandas 3-D, con compensación de movimiento. Tanto el algoritmo EZW (embedded zerotree wavelet) desarrollado por Shapiro [3], el codificador basado en partición en conjuntos de árboles jerárquicos de Said y Pearlman [4] y el codificador LTW desarrollado por Oliver y Malumbres [5], explotan las similitudes entre las diferentes subbandas wavelet debido a la estructura en árbol de la transformada wavelet. Estas propuestas obtienen muy buen rendimiento en imágenes, es decir en dos dimensiones, además de presentar una baja complejidad computacional. De esta forma, haciendo uso de un codificador de imagen y considerando la nueva dimensión temporal, Chen y Pearlman [6] desarrollan el codificador de vídeo 3D IEZW basado en el método bidimensional IEZW, que ofrece un sistema de codificación de vídeo computacionalmente sencillo sin compensación de movimiento pero eficaz, que obtiene excelentes resultados a nivel computacional y visual. En el campo de la compresión de imágenes volumétricas (ver [7]), se ha desarrollado con buenos resultados un codificador basado en el EZW. En [8] y [9], en lugar de utilizar la codificación típica basada en árboles cuadráticos se ha hecho uso de un árbol con ocho descendientes para utilizar tanto el codificador de imagen LTW como el codificador SPIHT.

Varios trabajos realizados se han centrado en la aceleración de la DWT, especialmente de la 2D DWT, tanto en arquitecturas de memoria compartida como en GPUs (graphic processing units). En [10] se alcanza un buen speed-up en el cálculo de la 2D-DWT, explotando la arquitectura paralela haciendo uso tanto de OpenGL como de Cg, los resultados mostrados han sido obtenidos en la GPU GeForce 7800 GTX. Un trabajo similar, haciendo uso de la misma GPU, presenta speed-ups entre 1.2 y 3.4 respecto al cómputo en CPU. En [11] se presenta una implementación de la 2D-FWT en CUDA con un speed-up superior a 20 respecto a la versión secuencial en un único procesador, y un speed-up de 2 respecto a la versión optimizada para multicores utilizando OpenMP y PThreads. En un trabajo previo (ver [12]), presentamos implementaciones de la 2D-DWT tanto para multicores como para GPUs, en las cuales obtenemos speed-ups de 7.1 y 8.9 haciendo uso de ocho y diez procesadores respectivamente, comparado con los resultados secuenciales.

El trabajo que presentamos extiende nuestros análisis a la 3D-DWT, analizando tanto las estrategias de paralelización como el impacto del compilador y de los flags de optimización. Contrastaremos nuestros resultados con los presentados en [13].

El resto del artículo está organizado de la siguiente forma: en la sección II presentamos las bases de la transformada 3D-DWT; en la sección III describimos la implementación realizada para multicores; en la sección IV analizamos el rendimiento de nuestra propuesta; y, finalmente, en la sección V presentamos las conclusiones obtenidas.

II. Transformada 3D Wavelet

La trasformada DWT es una descomposición con un esquema multirresolución para señales digitales, una descripción detallada puede verse en [14]. La señal de entrada se descompone inicialmente en dos subbandas de frecuencia: banda de baja frecuencia y banda de alta frecuencia. En la transformada clásica DWT se utilizan dos filtros, uno paso bajo H y otro paso alto G. Ambos filtros provienen de una función de escalado $\Phi(t)$ y de las funciones wavelet correspondientes a diferentes frecuencias $\Psi(t)$. Este sistema disminuye a la mitad la resolución de los resultados obtenidos en el proceso de descomposición. Si consideramos un filtro FIR no recursivo de cuatro niveles, las funciones de transferencia pueden representarse de la siguiente forma:

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

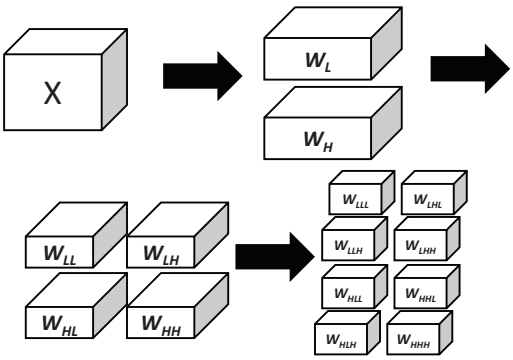


Fig. 1. Reducción de la resolución de una señal 3D en cada dimensión.

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

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

Se debe implementar una versión 3D de los bancos de filtros de análisis y de síntesis para procesar tanto vídeo como imágenes volumétricas haciendo uso de la transformada wavelet. Para el caso de la transformada 3D el banco de filtros de análisis 1D es utilizado en cada una de las tres dimensiones. Considerando un tamaño de datos N_1 por N_2 por N_3, tras aplicar el banco de filtros 1D a la primera dimensión se obtienen dos subbandas, cada una de tamaño $\frac{N_1}{2}$ por N_2 por N_3. Tras aplicar el banco de filtros a la segunda dimensión obtenemos cuatro subbandas, cada una de tamaño $\frac{N_1}{2}$ por $\frac{N_2}{2}$ por N_3. Y tras aplicar el banco de filtros a la tercera dimensión se obtienen ocho subbandas de tamaño $\frac{N_1}{2}$ por $\frac{N_2}{2}$ por $\frac{N_3}{2}$ (ver figura 1).

Tras aplicar la transformada 3D-DWT a un grupo de imágenes (GOP: group of pictures), es decir la transformada 2D-DWT en el dominio espacial y la transformada 1D-DWT en el dominio temporal, se obtienen las ocho subbandas del primer nivel de descomposición (denominadas LLL_1, LHL_1, LLH_1, LHH_1, HLL_1, HHL_1, HLH_1, HHH_1). Si realizamos descomposiciones adicionales sobre la banda de baja frecuencia (LLL_1), se obtiene el segundo nivel de descomposición, y así sucesivamente (ver el ejemplo de la figura 2).

En este trabajo se ha usado el filtro Daubechies 9/7 tanto para la descomposición espacial como para la descomposición temporal. Además, para la transformada wavelet tridimensional se ha utilizado la convolución basada en banco de filtros, basándonos en los resultados obtenidos en [12] para el caso de la transformada bidimensional.

III. Transformada 3D wavelet para multicores

En [12] proponemos el algoritmo paralelo para el cómputo de la transformada 2D-DWT mediante convolución basado en el uso de una memoria adicional para realizar los cálculos, evitando de este modo la necesidad de duplicar el espacio de almacenamiento.

Tamaño de frame	Número de procesos	Memoria adicional	Incremento (%) GOP: 32
352 x 288	1	360	0.0110
	4	1440	0.0443
	10	3600	**0.1109**
1280 x 640	1	1288	0.0024
	4	5152	0.0099
	10	12880	0.0247
1920 x 1024	1	1928	0.0016
	4	7712	0.0065
	10	**19280**	0.0164

TABLA I
Tamaño de memoria adicional en píxeles y porcentaje.

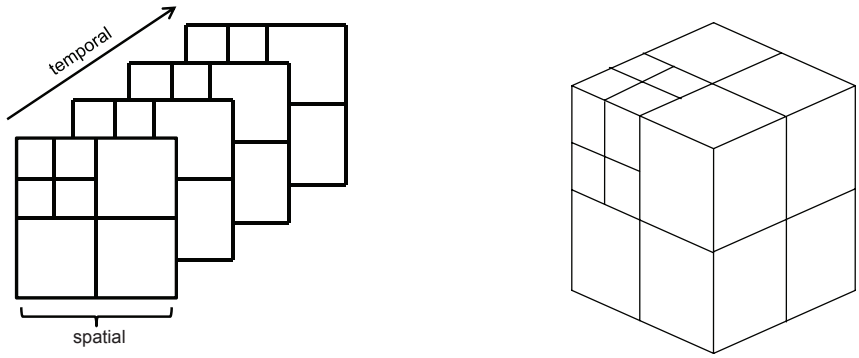
Esta estrategia ha sido seguida de nuevo para el desarrollo del algoritmo 3D-DWT.

Hay que remarcar que utilizamos cuatro niveles de descomposición para calcular la 3D-DWT, y que, como se ha visto en la sección II, cada nivel de descomposición se divide en dos etapas. En la primera etapa aplicamos la transformada 2D-DWT a cada frame del GOP que se se está computando. En la segunda etapa se calcula la transformada 1D-DWT para considerar el eje temporal. Se ha hecho uso de la extensión simétrica para minimizar el efecto de los bordes, tanto los bordes de cada imagen como en los bordes de cada GOP.

Si consideramos la primera etapa (es decir la transformada 2D-DWT aplicada a cada frame), la memoria adicional necesaria depende del tamaño de columna o del tamaño de fila, aquella dimensión que sea mayor, pero también depende del número de procesos utilizados en el cálculo paralelo. La memoria adicional es utilizada para almacenar la fila o columna del frame con la que se está trabajando además de los píxeles necesarios para realizar la extensión simétrica. En el caso del filtro utilizado, Daubechies 9/7, debemos extender cuatro elementos en cada borde.

La tabla I muestra el tamaño de la memoria adicional necesaria en píxeles, y el porcentaje que implica dicho tamaño sobre el tamaño total del GOP considerado, tanto para diferentes tamaños de frame como diferente número de procesos. Hay que remarcar que cada proceso almacena los píxeles con los que trabaja, los cuales no son compartidos con el resto de procesos. El peor caso de la tabla I, respecto al porcentaje de aumento de memoria requerido es un valor de tan solo el 0.1109%. Por otro lado, atendiendo al tamaño total de memoria adicional necesaria, el peor caso incrementa tan solo un 0.0164% el tamaño de memoria necesario. En el caso de que el tamaño del GOP fuera mayor que el tamaño de fila y de columna, el tamaño de la memoria adicional dependería de dicho tamaño de GOP y no del tamaño de frame. En la tabla I los datos mostrados han sido calculados considerando un tamaño de GOP igual a 32.

En la segunda etapa del cálculo de la transformada 3D-DWT (es decir el cálculo de la transformada 1D-DWT sobre el eje temporal), se realiza la extensión simétrica para evitar el efecto producido por los bor-



(a) (b)

Fig. 2. Esquema de cómputo de la 3D-DWT con dos niveles de descomposición usando el algoritmo 3D-DWT.

des en el dominio temporal. En todos los experimentos realizados se ha considerado un tamaño máximo de GOP igual a 128, por tanto la memoria adicional utilizada en la primera etapa es suficiente para ser reutilizada en la segunda etapa.

Hemos hecho uso de OpenMP [15] en el desarrollo del algoritmo paralelo para el cálculo de la transformada 3D-DWT. Los multiprocesadores utilizados son:

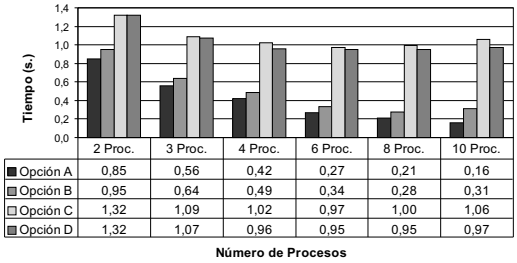
- Intel Core 2 Quad Q6600 2.4 GHz, con 4 cores.
- HP Proliant SL390 G7 con dos Intel Xeon X5660, cada CPU con 6 cores a 2.8 GHz.

Hemos analizado diversas estrategias en la implementación del algoritmo paralelo 3D-DWT en ambas etapas de dicho algoritmo. Estas estrategias analizadas son:

- Bucles paralelos automáticos de OpenMP.
- Secciones paralelas.
- Balanceo de carga entre procesos.

En todos los experimentos realizados la técnica de secciones paralelas no ha obtenido buenos resultados. Por ello presentamos diversas opciones de paralelización del algoritmo 3D-DWT pero únicamente basado en las opciones de bucles automáticos y balanceo de carga. Se han analizado las siguientes cuatro opciones:

- Opción A: La transformada inicial 2D-DWT aplicada a cada frame es computada por todos los procesos, los cuales cooperan en esta tarea, computando cada proceso un bloque de filas o columnas contiguas. La transformada en el eje temporal 1D-DWT se computa asignando un bloque de columnas contiguas a cada proceso en función de su rango en el entorno paralelo.
- Opción B: En la transformada inicial cada proceso calcula la transformada 2D-DWT de uno o varios frames completos. La siguiente transformada 1D-DWT se calcula asignando un bloque de columnas contiguas a cada proceso en función de su rango.
- Opción C: La transformada inicial 2D-DWT se calcula como en el caso de la opción A. La trasformada 1D-DWT temporal se computa mediante el uso de bucles paralelos automáticos de OpenMP.
- Opción D: La transformada inicial 2D-DWT se calcula como en el caso de la opción A. La trasformada 1D-DWT temporal se computa paralelizando el cómputo por fila en lugar de paralelizar el proceso por columna, es decir asignando un bloque de filas contiguas a cada proceso en función de su rango paralelo.



	2 Proc.	3 Proc.	4 Proc.	6 Proc.	8 Proc.	10 Proc.
Opción A	0,85	0,56	0,42	0,27	0,21	0,16
Opción B	0,95	0,64	0,49	0,34	0,28	0,31
Opción C	1,32	1,09	1,02	0,97	1,00	1,06
Opción D	1,32	1,07	0,96	0,95	0,95	0,97

Fig. 3. Tiempo computacional del algoritmo paralelo 3D-DWT. Compilador: ICC. Flags de compilación: -fast -openmp. Tamaño de frame: 1280 × 640. Multicore HP Proliant SL390.

En la figura 3 presentamos un análisis de las estrategias paralelas descritas. En dicha figura puede observarse que el mejor rendimiento se ha obtenido balanceando la carga computacional de los procesos, este comportamiento se ha observado en todos los experimentos realizados. Por tanto la elección óptima de estrategia, la opción A, ha sido utilizada en el resto de resultados mostrados. Hay que remarcar que en todos los experimentos mostrados cada procesador (o core) ejecuta un único proceso.

IV. Análisis del rendimiento

En esta sección analizaremos el rendimiento del algoritmo paralelo descrito en las secciones anteriores, realizando, además, una comparación con una reciente propuesta presentada en [13]. La figura 4 muestra los tiempos computacionales y sus respectivas eficiencias para un vídeo con tamaño de frame igual a 1280 × 640 variando el tamaño de GOP y el número de procesos. La figura 4(a) muestra el buen rendimiento paralelo del algoritmo desarrollado. En el algoritmo de la transformada 3D-DWT hay un evi-

dente uso intensivo de la memoria, esto hace que la mejora en el uso de la caché y la localidad de los datos justifique suficientemente el hecho de obtener eficiencias superiores a 1, como se puede observar en la figura 4(b). Los valores de eficiencia mostrados en dicha figura corresponden a ejecuciones en el multiprocesador Q6600.

Es evidente que el tamaño de GOP es un parámetro importante, tanto a nivel de computación como en el hecho de que la calidad media del vídeo mejora, cuando se aplica a codificación de vídeo, al incrementar el tamaño de GOP. Esto se debe a que se minimizan los efectos de bordes en el eje temporal. Sin embargo, la carga computacional y los requisitos de memoria aumentan al incrementar el tamaño de GOP. Idealmente el tamaño de GOP debería ser igual al número total de frames del vídeo considerado, pero, dado que no es posible debido a las restricciones de memoria, debe seleccionarse un tamaño de GOP atendiendo tanto a la calidad de la codificación como al coste computacional. Como se puede observar en la figura 4(a) el tiempo computacional crece al aumentar el tamaño de GOP. El mínimo tamaño de GOP en nuestro algoritmo es igual a 16 debido a los cuatro niveles de descomposición que se realizan en la transformada 3D-DWT. En la figura 5 se muestra el porcentaje de incremento de tiempo computacional respecto al tiempo computacional correspondiente a un GOP de tamaño igual a 16.

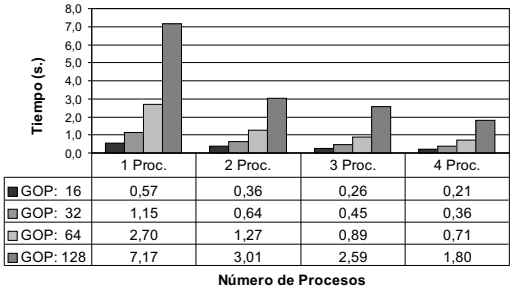
Hay que remarcar que el número de frames computados, es decir de imágenes, es igual al tamaño de GOP. Por lo tanto si se incrementa el tamaño de GOP se disminuye el efecto de los bordes en el eje temporal, obviamente el tiempo computacional debe crecer debido a que se están computando mayor número de imágenes. En la figura 6 se muestran datos de tiempo computacional por frame, correspondientes a los datos mostrados en la figura 5. Puede observarse que el algoritmo paralelo presenta mejores resultados cuando se incrementa tanto el tamaño de GOP como el número de procesos. Hay que remarcar que si se fija un tamaño de GOP igual a 256, considerando imágenes de media y alta resolución, los resultados que se obtienen nunca son buenos debido a los altos requisitos de memoria necesarios. El valor óptimo de tamaño de GOP es bien 64 o bien 128, la elección depende del hecho conocido de que un GOP de tamaño igual a 64 reduce los requisitos de memoria y un GOP de tamaño igual a 128 reduce los efectos de los bordes en el eje temporal. Con ambos valores se obtienen los mejores resultados en términos de tiempo computacional por frame, tal y como puede observarse en la figura 6.

Se ha presentado un exhaustivo análisis de nuestro algoritmo paralelo, mostrando resultados en función de los diversos parámetros que le afectan. Tal y como se ha podido observar, nuestro algoritmo paralelo presenta muy buen rendimiento utilizando hasta 12 procesos en el multiprocesador HP Proliant SL390 y hasta 4 procesos en el multiprocesador Q6600, siempre que se escojan correctamente los parámetros de ejecución. Se realizará una comparativa respecto a un reciente algoritmo presentado en [13], el cual presenta algunas técnicas de optimización interesantes. Ambos algoritmos que van a ser comparados tienen el mismo objetivo pero no lo alcanzan a través de los mismos medios, en particular el algoritmo de referencia utiliza el filtro Daubechies W_4, mientras que en nuestro caso hacemos uso del filtro Daubechies 9/7. Hay que remarcar que nuestro algoritmo realiza cuatro niveles de descomposición para el cálculo de la transformada 3D-DWT, mientras que el algoritmo presentado en [13] realiza un único nivel de descomposición. Por lo tanto, las técnicas de optimización usadas en ambos algoritmos son diferentes. Además, en nuestro algoritmo se hace uso de la extensión simétrica para evitar los efectos de los bordes, mientras que el algoritmo de referencia no aplica ninguna técnica específica con este objetivo.

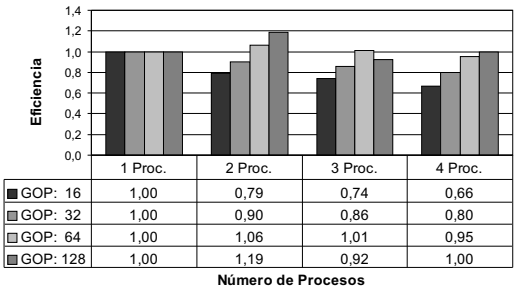
Los multiprocesadores utilizados en ambos casos son muy similares. Nuestro algoritmo muestra resultados en un multiprocesador Q6600, mientras que los resultados presentados con el algoritmo de referencia han sido obtenidos en un multiprocesador Q6700. La figura 7 muestra el tiempo computacional para el cálculo de la 3D-DWT con 4 procesadores, para diferentes tamaños de frame, y con tamaño de GOP igual a 64, para ambos algoritmos. Los resultados mostrados en la figura 7(a) han sido obtenidos con el compilador GCC, dado que no se dispone del compilador ICC en el multiprocesador Q6600. Los datos mostrados en la figura 7 han sido obtenidos con diferentes tamaños de frame, debido a que en nuestro caso trabajamos con tamaños de frame estándar, mientras que el algoritmo de referencia trabaja con tamaños de frame cuadrados.

En la figura 8 se analiza el número de megapíxeles por segundo computados en ambos algoritmos. Dados los resultados mostrados en dicha figura se puede concluir que nuestro algoritmo presenta mayor degradación del rendimiento cuando el tamaño de frame aumenta. Esto es debido a las diferencias comentadas anteriormente, por un lado el número de descomposiciones wavelet realizadas y por otro la extensión simétrica realizada en nuestro algoritmo. Las técnicas usadas en el algoritmo de referencia para mejorar el rendimiento no funcionan en nuestro algoritmo. Hay que remarcar que nuestro algoritmo hace uso de un pequeño porcentaje de memoria adicional para la realización de los cálculos, tal y como se muestra en la tabla I, evitando así el uso del doble de la memoria necesaria por GOP para almacenar los coeficientes calculados.

Como se ha mencionado, ambos algoritmos utilizan diferentes filtros en el cálculo de la 3D-DWT, lo cual implica que la carga computacional por píxel difiera entre ambos algoritmos. Por ello en la figura 9 mostramos resultados en términos de GFLOPS. Se han calculado los GFLOPS de los resultados mostrados en [13] considerando el tamaño de frame, el tamaño de GOP y el filtro Daubechies W_4 uti-

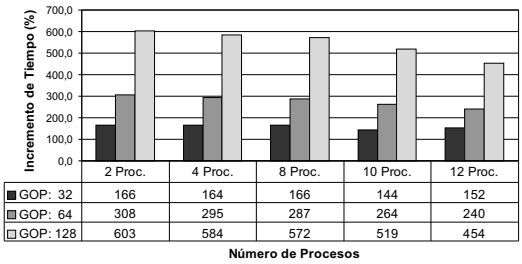


(a) Tiempo (s.).

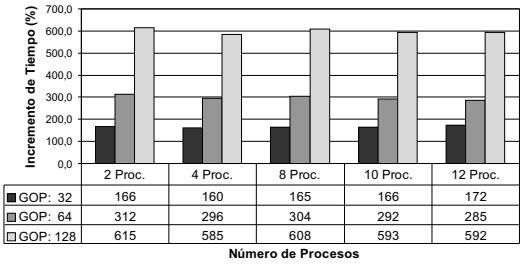


(b) Eficiencia.

Fig. 4. Algoritmo 3D-DWT. Compilador: GCC. Flags de compilación: -O3 -openmp. Tamaño de frame: 1280×640. Multicore Q6600.

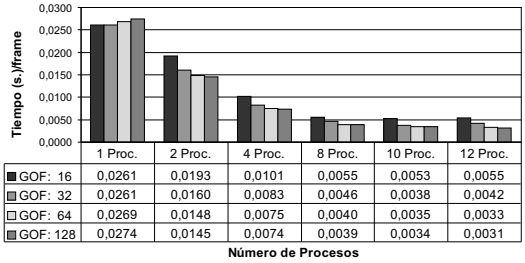


(a) Tamaño de frame: 1280×640.

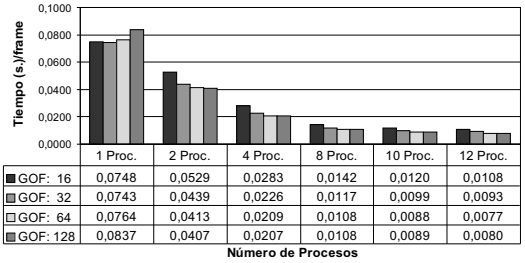


(b) Tamaño de frame: 1920×1024.

Fig. 5. Incremento de tiempo (%) respecto a tiempos con GOP igual a 16. Compilador: ICC. Flags de compilación: -fast -openmp. Multicore HP Proliant SL390.

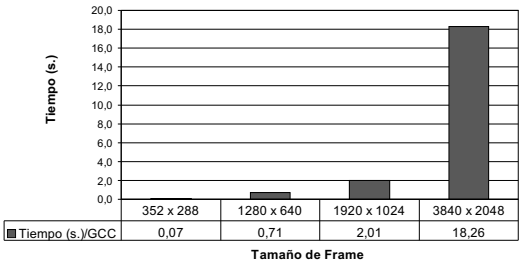


(a) Tamaño de frame: 1280×640.

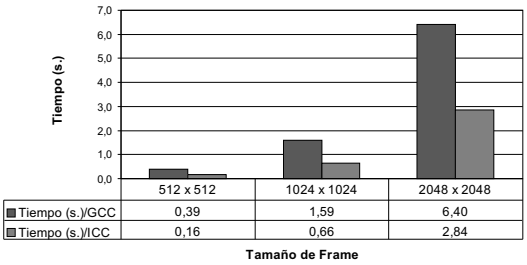


(b) Tamaño de frame: 1920×1024.

Fig. 6. Tiempo computacional por frame. Compilador: ICC. Flags de compilación: -fast -openmp. Multicore HP Proliant SL390.



(a) Algoritmo desarrollado. Multicore Q6600.



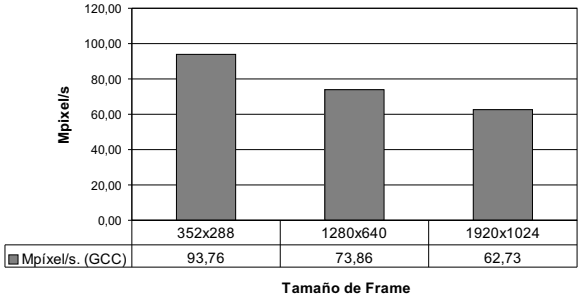
(b) Algoritmo de referencia. Multicore Q6700.

Fig. 7. Tiempo computacional del algoritmo paralelo 3D-DWT. Tamaño de GOP: 64. Número de procesos: 4.

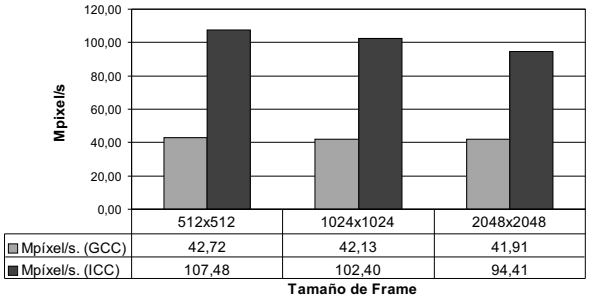
lizado. Considerando los resultados utilizando el mismo compilador, nuestro algoritmo computa hasta 4 veces más GFLOPS que el algoritmo de referencia.

V. Conclusiones

Se ha presentado un algoritmo para arquitecturas de memoria compartida o multicores, que ha sido desarrollado haciendo uso de OpenMP, para el cálculo de la transformada 3D-DWT. Se ha analizado el comportamiento del algoritmo desarrollado en dos plataformas diferentes. Además, hemos comparado nuestro algoritmo respecto a un reciente algoritmo propuesto en [13]. El algoritmo desarrollado presenta speed-ups, en muchos casos, casi ideales en función

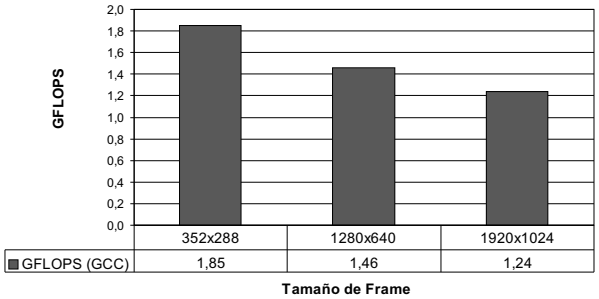


(a) Algoritmo desarrollado. Multicore Q6600.

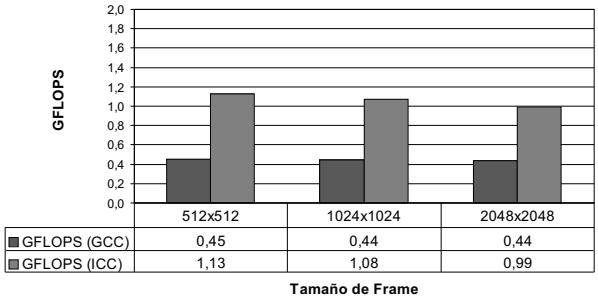


(b) Algoritmo de referencia. Multicore Q6700.

Fig. 8. Megapíxeles por segundo en el cálculo de la transformada 3D-DWT. Tamaño de GOP: 64. Número de procesos: 4.



(a) Algoritmo desarrollado. Multicore Q6600.



(b) Algoritmo de referencia. Multicore Q6700.

Fig. 9. GFLOPS en el cálculo de la transformada 3D-DWT. Tamaño de GOP: 64. Número de procesos: 4.

del tamaño de frame y de GOP considerado, cuando el multiprocesador utilizado es una plataforma de relativa baja potencia de cálculo como es el multiprocesador Q6600. Considerando los resultados en el multiprocesador HP Proliant SL390 G7, el algoritmo desarrollado presenta buenas eficiencias incluso utilizando el número máximo de cores disponible, en función, de nuevo, del tamaño de frame y del tamaño de GOP. En este último caso las eficiencias obtenidas son superiores al 83%. Como se ha mencionado, nuestro algoritmo no necesita espacios de memoria diferentes para almacenar el vídeo origen y los coeficientes calculados, con un incremento de memoria despreciable, incluso usando 12 procesos.

Acknowledgment

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

Referencias

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

[2] 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.

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

[4] 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.

[5] 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.

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

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

[8] B.J. Kim, Z. Xiong, and W.A. 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.

[9] O. Lopez, M. Martinez-Rach, P. Piñol, M.P. 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.

[10] Tien-Tsin Wong, Chi-Sing Leung, Pheng-Ann Heng, and Jianqing Wang, "Discrete wavelet transform on consumer-level graphics hardware," *Multimedia, IEEE Transactions on*, vol. 9, no. 3, pp. 668 –673, april 2007.

[11] J. Franco, G. Bernabé, J. Fernández, M.E. 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.

[12] V. Galiano, O. López, M.P. 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.

[13] Joaquín Franco, Gregorio Bernabé, Juan Fernández, and Manuel Ujaldón, "Parallel 3d fast wavelet transform on manycore gpus and multicore cpus," *Procedia Computer Science*, vol. 1, no. 1, pp. 1101 – 1110, 2010.

[14] 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.

[15] "Openmp application program interface, version 3.1," *OpenMP Architecture Review Board.* `http://www.openmp.org`, 2011.