July, 2-5, 2012.

Fast and In-place Computation Parallel 3D Wavelet Transform

Vicente Galiano[1], Otoniel López[1], Manuel P. Malumbres[1] and Héctor Migallón[1]

[1] *Physics and Computer Architecture Department., Miguel Hernández University. Elche, Spain 03202.*

emails: vgaliano@umh.es, otoniel@umh.es, mels@umh.es, hmigallon@umh.es

Abstract

Three-dimensional wavelet transform (3D-DWT) has focused the attention of the research community, most of all in areas such as video watermarking, compression of volumetric medical data, multispectral image coding, 3D model coding and video coding. In this work, we present several strategies to speed-up the 3D-DWT computation through multicore processing. An in depth analysis about the available compiler optimizations is also presented. Depending on both the multicore platform and the GOP size, the developed parallel algorithm obtains efficiencies above 95% using up to four cores (or processes), and above 83% using up to twelve cores. Furthermore, the extra memory requirements are under 0.12% for low resolution video frames, and under 0.017% for high resolution video frames.

Key words: wavelet transform, video coding, parallel algorithms, OpenMP, in-place computing.

1 Introduction

In the last years, the three-dimensional wavelet transform (3D-DWT) has focused the attention of the research community, most of all in areas such as video watermarking [1] and 3D coding (e.g., compression of volumetric medical data [2] or multispectral images [3], 3D model coding [4], and especially, video coding). 3-D subband video coding is an alternative to the traditional motion-compensated Discrete Cosine Transform (DCT) coding. The 3D subband coding uses the discrete wavelet transform (DWT), which achieves better energy compaction, instead of the DCT.

Podilchuk, et al., utilized 3-D spatio-temporal subband decomposition and geometric vector quantization (GVQ) [5]. Taubman and Zakhor presented a full color video coder based on 3-D subband coding with camera pan compensation [6]. Adapted versions of

wavelet image encoders can be also used, taking into account the new dimension. For instance, the two dimensional (2D) embedded zero-tree (IEZW) method has been extended to 3D IEZW for video coding by Chen and Pearlman [7], and showed promise of an effective and computationally simple video coding system without motion compensation, obtaining excellent numerical and visual results. A 3D zero-tree coding through modified EZW has also been used with good results in compression of volumetric images [8]. In [9] and [10], instead of the typical quad-trees of image coding, a tree with eight descendants per coefficient is used to extend both SPIHT and LTW image encoders to 3D video coding.

Several attempts has been made in order to accelerate the DWT, specially the 2D DWT, exploiting both multicore architectures and graphic processing units (GPU). In [11], a SIMD algorithm runs the 2D-DWT on a GeForce 7800 GTX using Cg and OpenGL, with a remarkable speed-up. A similar effort in [12] combined Cg and the 7800 GTX to report a 1.2x-3.4x speed-up versus a CPU counterpart. In [13], a CUDA implementation for the 2D-FWT running more than 20 times as faster the sequential C version on a CPU, and more than twice as faster the optimized OpenMP and Pthreads versions implemented on multicore CPUs. In a previous work [14], we presented both multicore and GPU implementations for the 2D-DWT obtaining speed-ups up to 7.1 and 8.9 on a multicore platform using eight and ten processes, respectively when compared to the CPU sequential algorithm.

This work extends our analysis to the 3D-DWT, analyzing the compiler flags impact as well as the different optimizations applied. We analyze the computational behavior in order to set the optimal performance parameters. Also, we compare our results against [15].

The rest of the paper is organized as follows. Section 2 presents the foundations of the 3D-DWT. Section 3 describes our implementation proposal on multicore CPUs, and Section 4 analyzes its performance. Finally in Section 5 some conclusions are drawn.

2 3D Wavelet Transform

The DWT is a multiresolution decomposition scheme for input digital signals, see detailed description in [16]. The source signal is firstly decomposed into two frequency subbands, low-frequency (low-pass) subband and high-frequency (high-pass) subband. For the classical DWT, the forward decomposition of a signal is implemented by a low-pass digital filter H and a high-pass digital filter G. Both digital filters are derived using the scaling function $\Phi(t)$ and the corresponding wavelet functions at different frequency scales $\Psi(t)$. The system downsamples the signal to half of the filtered results in the decomposition process. If four-tap and non-recursive FIR filters are considered, the transfer functions of H and G can be represented as follows:

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

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

To use the wavelet transform for volume and video processing we must implement a 3D version of the analysis and synthesis filter banks. In the 3D case, the 1D analysis filter bank is applied in turn to each of the three dimensions.

After applying the 3D-DWT on a group of video pictures (GOP), a 2D spatial DWT and a 1D temporal DWT, we obtain eight first level wavelet subbands (named as LLL_1, LHL_1, LLH_1, LHH_1, HLL_1, HHL_1, HLH_1, HHH_1). Further decompositions can be done, focusing on the low-frequency subband (LLL_1), achieving in this way a second-level wavelet decomposition, and so on.

In this work we will use the Daubechies 9/7 filter for both the spatial and temporal decompositions. In addition, the regular filter-bank convolution is considered to develop the three-dimensional wavelet transform, based on the results obtained in [14] for the two-dimensional wavelet transform case. In particular, in [14] we obtain best results, in terms of computational times and in terms of parallel performance, applying the regular filter-bank convolution than applying the lifting scheme.

3 Multicore 3D Wavelet Transform

As we have said, the Daubechies 9/7 filter, proposed in [16], has been used to perform the regular filter-bank convolution in order to develop the parallel 3D-DWT algorithm. In [14] we proposed the convolution-based parallel 2D-DWT using an extra memory space in order to perform a nearly in-place computation, avoiding the requirement of twice the image size to store the computed coefficients. This strategy is also followed to develop the parallel 3D-DWT algorithm.

We want to remark that we use four decomposition levels in order to compute the 3D-DWT, and, as we have said in Section 2, each decomposition level computation is divided into two main steps. In the first step the 2D-DWT is applied to each frame of the current GOP, and in the second step the 1D-DWT is performed to consider the temporal axis. We have used the symmetric extension technique in order to avoid the border effects on both the frame borders and the GOP borders.

If we consider the first step (i.e. the 2D-DWT applied to each video frame), the extra memory size depends on both, the row size or column size (the larger one), and the number of processes in the parallel algorithm. The extra memory stores the current frame row/column pixels plus the pixels of the symmetric extension. For Daubechies 9/7 filter we must extend four elements on all borders.

Table 1 shows the extra memory size in pixels and the percentage in memory increase, for several video frame sizes and number of processes used in the parallel algorithm. Note that each process stores its own working pixels which are not shared with other processes. The worst case in Table 1, attending at memory increase, is a very small value equal to 0.1109%. On the other hand, attending to the amount of extra memory size, the worst

Frame Size	Processes	Extra memory size Pixel size	Increment (%) GOP: 32
352 x 288	1	360	0.0110
	2	720	0.0221
	4	1440	0.0443
	6	2160	0.0665
	10	3600	**0.1109**
1280 x 640	1	1288	0.0024
	2	2576	0.0049
	4	5152	0.0099
	6	7728	0.0148
	10	12880	0.0247
1920 x 1024	1	1928	0.0016
	2	3856	0.0032
	4	7712	0.0065
	6	11568	0.0098
	10	**19280**	0.0164

Table 1: Amount of extra memory size.

case increases the memory size by 0.0164%. If the GOP size is larger than the row size and column size the amount of extra memory is set by the GOP size. In Table 1 the percentage has been obtained considering a GOP size equal to 32.

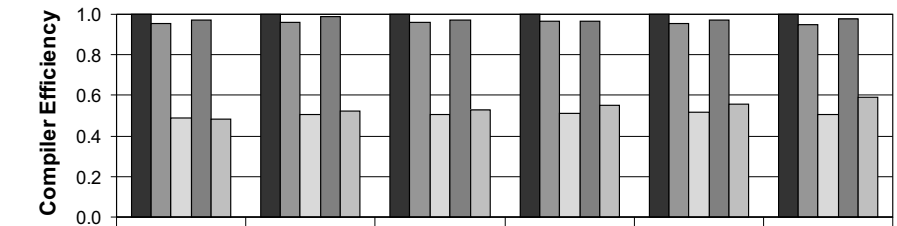
In the second step of the 3D-DWT (i.e. the temporal 1D-DWT), we perform the symmetric extension in order to avoid the border effects in the temporal domain. In all performed experiments the maximum GOP size considered is 128, therefore the extra memory used in the first step is enough to be reused in the second step.

We have used OpenMP [17] paradigm in order to develop the parallel 3D-DWT algorithm. The multicore platforms used are:

- Intel Core 2 Quad Q6600 2.4 GHz, with 4 cores.
- HP Proliant SL390 G7 with two Intel Xeon X5660, each CPU with six cores at 2.8 GHz.

We analyze some OpenMP-based techniques to parallelize the two main steps. The techniques tested to parallelize the 3D-DWT algorithm are:

- Automatic OpenMP parallel loops.
- Parallel sections.
- Load balancing according the thread rank.



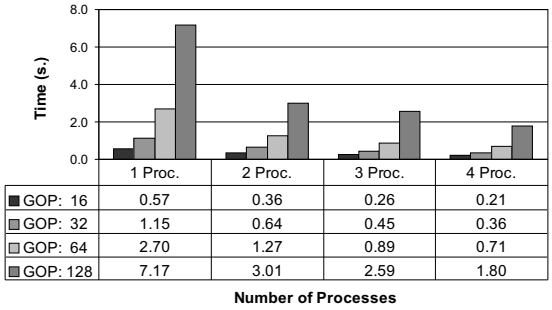
	1 Proc.	2 Proc.	4 Proc.	8 Proc.	10 Proc.	12 Proc.
icc -fast	1.00	1.00	1.00	1.00	1.00	1.00
icc -O3	0.96	0.96	0.96	0.96	0.96	0.95
icc	0.49	0.50	0.50	0.51	0.52	0.51
gcc -O3	0.97	0.99	0.97	0.96	0.97	0.98
gcc	0.48	0.52	0.53	0.55	0.55	0.59

Number of Processes

Figure 1: Compiler efficiency for the multicore 3D-DWT algorithm. Frame size: 1280×640. Multicore HP Proliant SL390.

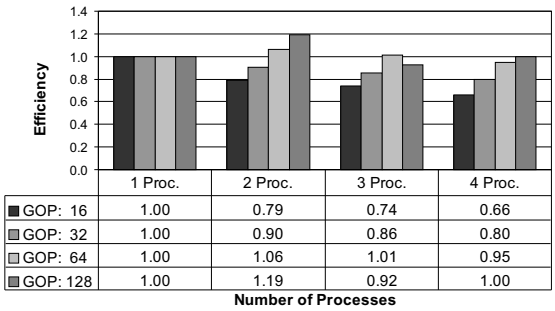
In all performed experiments the parallel sections technique has not obtained good results, while the best results are obtained by balancing the computational load according the thread rank.

We have also analyzed the behavior depending on the compiler and the flags used to build the 3D-DWT algorithm. We have tested the ICC [18] compiler, a corporate tool from Intel, and the GCC [19] compiler, which is a free compiler developed by the GNU project. In the multicore HP Proliant SL390 compilers available are the GCC 4.1.2 and the ICC 12.0.0, however in the multicore Q6600 the only available compiler is the GCC 4.4.3. Figure 1 shows the compiler efficiency, respect to the best option, for a grayscale video frame size of 1280×640 pixels and a GOP size equal to 64 on the multicore HP Proliant SL390. The best option is obtained by using the ICC compiler and the flag *-fast* (note that *-fast* is a shorthand that includes the flags *-O3 -ipo -static -xHOST -no-prec-div*). Note that the efficiencies showed in Figure 1 are computed respect to the computational time obtained using the ICC compiler and the compiler flag *-fast* and the number of processes used in each experiment. Also in Figure 1 we can observe that the performance of both compilers remains unchanged as we increase the number of processes. This conclusion can be applied to the use of the different compiler flags. It is important to remark that both compilers offer the same performance when we use the same optimization flags and the ICC compiler obtains a slight performance increase by the flag *-fast*.



	1 Proc.	2 Proc.	3 Proc.	4 Proc.
GOP: 16	0.57	0.36	0.26	0.21
GOP: 32	1.15	0.64	0.45	0.36
GOP: 64	2.70	1.27	0.89	0.71
GOP: 128	7.17	3.01	2.59	1.80

(a) Time (s.).



	1 Proc.	2 Proc.	3 Proc.	4 Proc.
GOP: 16	1.00	0.79	0.74	0.66
GOP: 32	1.00	0.90	0.86	0.80
GOP: 64	1.00	1.06	1.01	0.95
GOP: 128	1.00	1.19	0.92	1.00

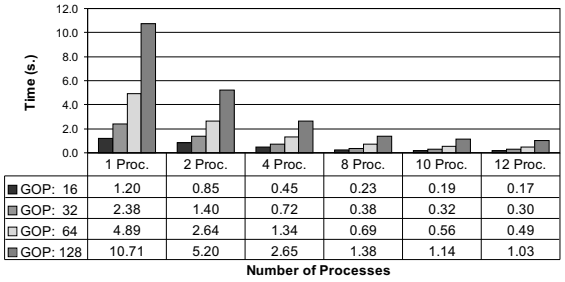
(b) Efficiency.

Figure 2: 3D wavelet algorithm. Compiler: GCC. Compiler flags: -O3 -openmp. Frame size: 1280×640. Multicore Q6600.

4 Performance Evaluation

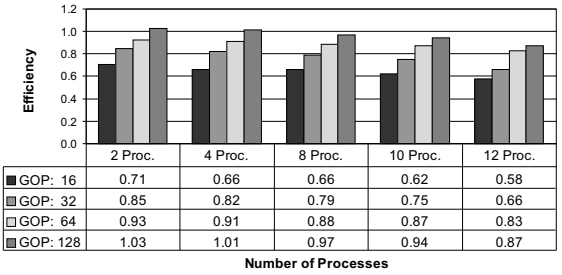
In this section we discuss the behavior of the parallel algorithm described in previous sections and we compare it against a recent optimized multicore proposal presented in [15]. Figure 2 presents the 3D-DWT computational times and their associated efficiencies for a video frame size of 1280×640 varying the GOP size and the number of processes. Figure 2(a) shows the good computation behavior of the parallel algorithm. In the 3D-DWT there is an intensive use of the memory, therefore the improvement in the use of the cache memory and data locality justifies the efficiencies greater than 1 showed in Figure 2(b). Efficiency values showed in this figure correspond to executions on the multicore Q6600. However, in Figure 3(b) this fact is not observed for the multicore HP Proliant SL390 due to the higher memory access performance respect to the multicore Q6600. The HP Proliant SL390 architecture provides a high-bandwidth memory access, through the Intel QPI Speed 64GT/s, therefore, the global performance improvement is less significant than in the Q6600 platform. In Figure 3 we also present the computational times and their associated efficiencies for the multicore HP Proliant SL390. The efficiencies obtained on both platforms are similar, however, comparing data obtained from video frames of different sizes we can conclude that the behavior on the multicore Q6600 becomes worse than on the multicore HP Proliant SL390, as the GOP size increases, i.e. when the global memory size increases. Note that the data presented in figures 2(b) and 3(b) correspond to different video frame sizes.

The GOP size is an important parameter in the 3D-DWT computation, when applied to video coding, because the average video quality increase as we increase the GOP size due to the minor GOP boundary effect. However, the computational load and memory requirements increase. Ideally, the GOP size would be equal to the total number of video



	1 Proc.	2 Proc.	4 Proc.	8 Proc.	10 Proc.	12 Proc.
GOP: 16	1.20	0.85	0.45	0.23	0.19	0.17
GOP: 32	2.38	1.40	0.72	0.38	0.32	0.30
GOP: 64	4.89	2.64	1.34	0.69	0.56	0.49
GOP: 128	10.71	5.20	2.65	1.38	1.14	1.03

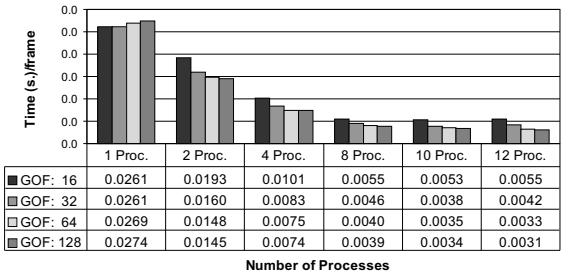
(a) Time (s.).



	2 Proc.	4 Proc.	8 Proc.	10 Proc.	12 Proc.
GOP: 16	0.71	0.66	0.66	0.62	0.58
GOP: 32	0.85	0.82	0.79	0.75	0.66
GOP: 64	0.93	0.91	0.88	0.87	0.83
GOP: 128	1.03	1.01	0.97	0.94	0.87

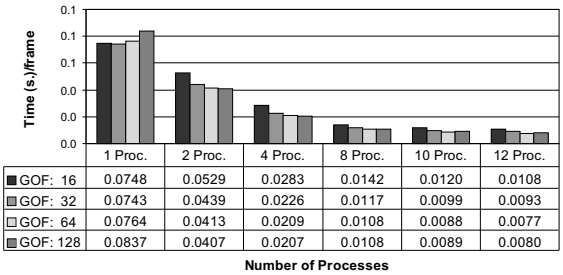
(b) Efficiency.

Figure 3: 3D wavelet algorithm. Compiler: ICC. Compiler flags: -fast -openmp. Frame size: 1920×1024. Multicore HP Proliant SL390.



	1 Proc.	2 Proc.	4 Proc.	8 Proc.	10 Proc.	12 Proc.
GOF: 16	0.0261	0.0193	0.0101	0.0055	0.0053	0.0055
GOF: 32	0.0261	0.0160	0.0083	0.0046	0.0038	0.0042
GOF: 64	0.0269	0.0148	0.0075	0.0040	0.0035	0.0033
GOF: 128	0.0274	0.0145	0.0074	0.0039	0.0034	0.0031

(a) Frame size: 1280×640.



	1 Proc.	2 Proc.	4 Proc.	8 Proc.	10 Proc.	12 Proc.
GOF: 16	0.0748	0.0529	0.0283	0.0142	0.0120	0.0108
GOF: 32	0.0743	0.0439	0.0226	0.0117	0.0099	0.0093
GOF: 64	0.0764	0.0413	0.0209	0.0108	0.0088	0.0077
GOF: 128	0.0837	0.0407	0.0207	0.0108	0.0089	0.0080

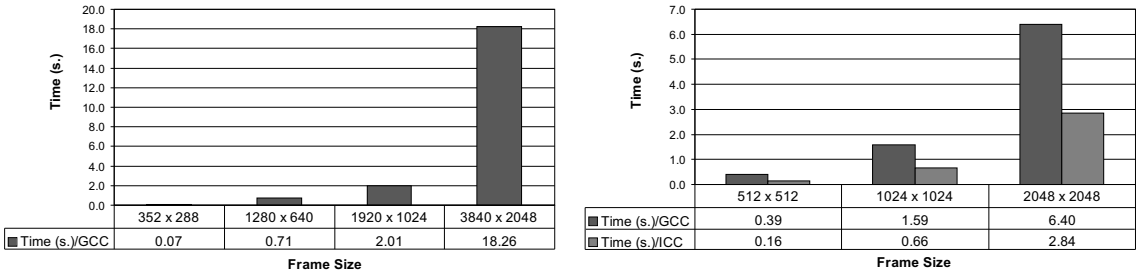
(b) Frame size: 1920×1024.

Figure 4: Computational time per frame. Compiler: ICC. Compiler flags: -fast -openmp. Multicore HP Proliant SL390.

frames, since this is not possible due to the device memory restrictions, we must to select the GOP size attending to both the video quality and the computational time. As we can see in figures 2(a) and 3(a) the computational time increases as the GOP size increases. The minimum GOP size in our algorithm is 16 due to the four wavelet decomposition levels performed in the 3D-DWT. Note the number of frames (or pictures) computed is the value of the GOP size.

The optimal values of the GOP size are 64 and 128, setting the GOP size equal to 128 reduces the border effects and setting the GOP size equal to 64 reduces the memory requirements. Both values obtain the best results, as it can be seen in Figure 4, in terms of computation times per frame.

We have presented an exhaustive analysis of our parallel algorithm, showing its behavior according to the possible modifications of all parameters. As we have seen, the parallel



(a) Developed algorithm. Multicore Q6600.

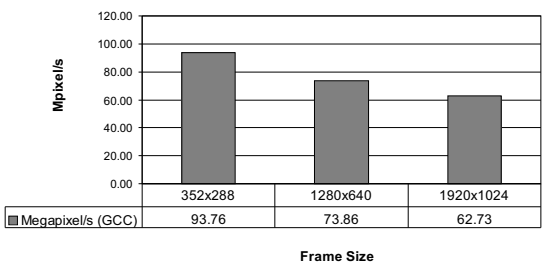
(b) Reference algorithm. Multicore Q6700.

Figure 5: Computational times for 3D wavelet algorithm. GOP size: 64. Number of processes: 4.

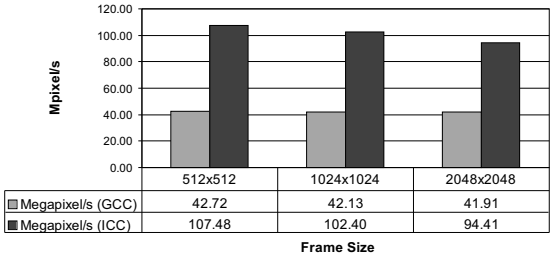
algorithm obtains good efficiency, with the proper parameters setting, using the available cores, up to 12 in the multicore HP Proliant SL390 and up to 4 in the multicore Q6600. At this time we will perform a comparative analysis against the recent algorithm presented in [15], which presents some interesting optimization techniques. Both compared algorithms use different methods to compute the 3D-DWT, in particular the reference algorithm uses the Daubechies W_4 filter instead of the Daubechies 9/7 filter used in our algorithm. It should be noted that our algorithm performs four decomposition levels to compute the 3D-DWT, while the reference algorithm presented in [15] performs only one decomposition level. Therefore, the optimization techniques used in both algorithms can not be the same. Furthermore we use the symmetric extension technique to avoid boundary effects, while the reference algorithm does not apply any specific technique for this purpose.

The platforms used to test both algorithms are very similar. Our platform has an Intel Q6600 quad-core processor and the reference algorithm has been run on an Intel Q6700 quad-core processor, i.e. the reference algorithm has been tested on a platform with a slightly higher performance. Figure 5 shows the computational times to compute the 3D-DWT for several video frame sizes, using 4 processes and a GOP size equal to 64. The results provided of the reference algorithm depend substantially on the compiler, as we can see in Figure 5(b), while our algorithm shows a lower compiler dependency, as it showed in Figure 1. The results shown in Figure 5(a) are obtained with the GCC compiler, because our multicore Q6600 does not provide the ICC compiler. The computational times presented in Figure 5 are obtained for different video frame sizes, because we work with standard video frame sizes and the reference algorithm works with square video frame sizes.

In Figure 6 we analyze the number of megapixels per second in both algorithms. Analyzing the results shown in this figure, we can conclude that our algorithm shows a greater performance degradation when the video frame size is increased. This is due to the pre-



(a) Developed algorithm. Multicore Q6600.



(b) Reference algorithm. Multicore Q6700.

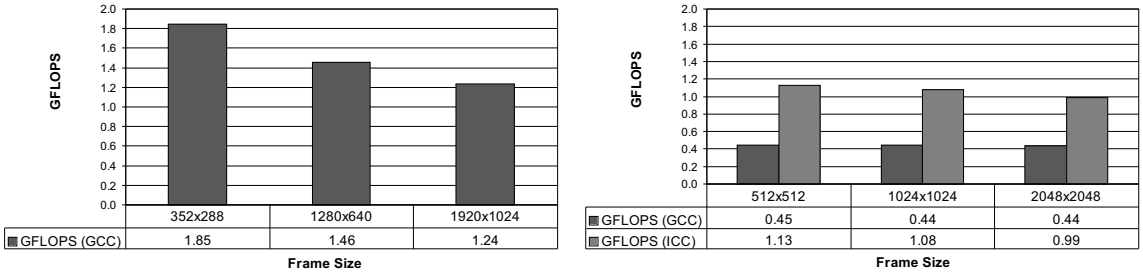
Figure 6: Megapixels per second for 3D wavelet algorithm. GOP size: 64. Number of processes: 4.

viously discussed differences, both the number of wavelet decomposition levels and the symmetric extension performed in our algorithm. Techniques used to improve the reference algorithm does not improve the performance of our algorithm. Note that we use an extra amount of memory to store the working data as showed in Table 1. We want to remark that our algorithm avoids the use of twice the video size to store the computed coefficients through this working memory.

As mentioned both algorithms use different filters in order to compute the 3D-DWT, which means that the computational load per pixel differs on both algorithms. Therefore in Figure 7 we present results in terms of GFLOPS. We have computed the GFLOPS of the experiments reported in [15] considering the video frame size, the GOP size and the filter Daubechies W_4. Attending to the results using the same compiler, our algorithm is able to compute up to 4 times more the GFLOPS than the reference algorithm. Analyzing different compilers our algorithm increases the GFLOPS using the free compiler GCC respect to the corporate ICC compiler.

5 Conclusions

We have presented the multicore-based algorithm, developed using the OpenMP paradigm, that performs the 3D discrete wavelet transform. We have analyzed the behavior of the developed algorithm when running on two different shared-memory platforms. Furthermore, we have compared our algorithm against a recent algorithm proposed in [15]. The multicore-based algorithm obtains a speed-ups closely ideal depending on the video frame size and the GOP size, running on a relatively low computing power platform as the Q6600 multicore platform, when compared to the sequential CPU algorithm. When running on the HP Proliant SL390 G7, our algorithm obtains good efficiencies even using the maximum number



(a) Developed algorithm. Multicore Q6600.

(b) Reference algorithm. Multicore Q6700.

Figure 7: GFLOPS for 3D wavelet algorithm. GOP size: 64. Number of processes: 4.

of available cores, depending on the video frame size and the GOP size. In this case the efficiencies achieved are greater than 83%. Furthermore, we do not require twice the video size to compute the 3D-DWT and the increased memory, even up to 12 processes, is negligible.

Acknowledgment

This research was supported by the Spanish Ministry of Education and Science under grant TIN2011-27543-C03-03 and the Spanish Ministry of Science and Innovation under grant number TIN2011-26254.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] J. Franco, G. Bernabé, J. Fernández, and M. Ujaldón, "Parallel 3d fast wavelet transform on manycore gpus and multicore cpus," *Procedia Computer Science*, vol. 1, no. 1, pp. 1101 – 1110, 2010.

[16] 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.

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

[18] "ICC, intel software network," *http://software.intel.com/en-us/intel-compilers/*, 2009-2011.

[19] "GCC, the gnu compiler collection," *Free Software Foundation, Inc. http://gcc.gnu.org*, 2009-2012.