July, 2-5, 2012.

GPU-based 3D Wavelet Transform

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

Wide amount of applications like volumetric medical data compression, video watermarking and video coding use the three-dimensional wavelet transform (3D-DWT) in their algorithms. In this work, we present GPU algorithms, based on both global and shared memory, to compute the 3D-DWT transform on both the GTX280 and the GMT540 platforms. The results obtained show that speed-ups of 19.7 and 10.65 on average can be obtained for the GTX280 and GMT540 platforms respectively when only the GPU's global memory is used. Moreover, Speed-ups increase considerably to 87 and 25 when the shared memory in the device is used optimizing the memory access to avoid idle threads. Futhermore, we discuss speed-up evolution depending on the group of pictures size (GOP).

Key words: wavelet transform,image coding , video coding, parallel algorithms, CUDA, GPU

1 Introduction

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 video coding [5]) have adopted 3-D subband video coding based on the three-dimensional wavelet transform (3D-DWT) as an alternative to the traditional motion-compensated Discrete Cosine Transform (DCT) coding. The 3D subband coding uses the discrete wavelet transform (DWT) instead of the DCT, achieving a better energy compaction.

In [6], Podilchuk, et al. utilized 3-D spatio-temporal subband decomposition and geometric vector quantization (GVQ). Taubman and Zakhor presented a full color video coder based on 3-D subband coding with camera pan compensation [7]. The embedded zerotree wavelet (EZW) algorithm developed by Shapiro [8], the set partitioning in hierarchical trees (SPIHT) algorithm developed by Said and Pearlman [9] and the lower tree wavelet encoder

(LTW) developed by Oliver and Malumbres [10] exploit the similarity between different wavelet subbands based on the wavelet tree structure. They provide remarkably good performance on 2-D images, with low computational complexity. Adapted versions of an image encoder can be used, taking into account the new dimension (temporal). For instance, the two dimensional (2D) embedded zero-tree (IEZW) method has been extended to 3D IEZW for video coding by Chen and Pearlman [11], 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 [12]. In [5] and [13], 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.

Wide research have been carried out to accelerate the DWT, specially the 2D DWT, exploiting both multicore architectures and graphic processing units (GPU). In [14], a Single Instruction, Multiple Data (SIMD) algorithm runs the 2D-DWT on a GeForce 7800 GTX using Cg and OpenGL, with a remarkable speed-up. A similar effort has been performed in [15] combining Cg and the 7800 GTX to report a 1.2 - 3.4 speed-up versus a CPU counterpart. In [16], authors present a CUDA implementation for the 2D-FWT running more than 20 times as fast as a sequential C version on a CPU, and more than twice as fast as the optimized OpenMP and Pthreads versions implemented on multicore CPUs. In [17], authors present GPU implementations for the 2D-DWT obtaining speed-ups up to 20 when compared to the CPU sequential algorithm.

The rest of the paper is organized as follows. Section 2 presents the foundations of the 3D-DWT. Section 3 focuses on the specifics of the GPU programming with CUDA, and Section 4 analyzes the performance and compares the results obtained. Finally in Section 6 some conclusions are drawn.

2 3D Wavelet Transform

The DWT is a multiresolution decomposition scheme for input digital signals, see detailed description in [18]. 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

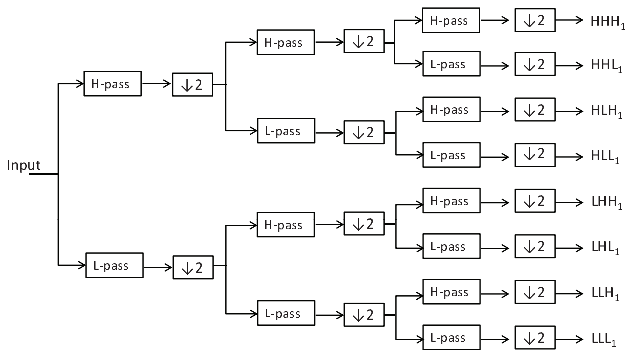


Figure 1: Overview of the 3D-DWT computation in a one-level decomposition using the regular 3D-DWT algorithm.

represented as follows:

$$H(z) = h_0 + h_1 z^{-1} + h_2 z^{-2} + h_3 z^{-3} \tag{1}$$
$$G(z) = g_0 + g_1 z^{-1} + g_2 z^{-2} + g_3 z^{-3} \tag{2}$$

A video sequence is an extension of 2D images along the time axis. In the most common form of wavelet video coding, a three dimensional Discrete Wavelet Transform (DWT) is applied on a group of video pictures (GOP). The 3D-DWT is a combination of a 2D spatial DWT and a 1D temporal DWT, where the temporal DWT absorbs motion in the GOP. The temporal DWT is carried out on the pixel values of the same location along the time axis. Figure 1 shows one level decomposition of the 3D-DWT where H-pass and L-pass represent the high pass filter and the low pass filter respectively. The downsampling of the filtered results is denoted as '$\downarrow$ 2'. After the decomposition, eight first level wavelet subbands are obtained (typically named as LLL_1, LHL_1, LLH_1, LHH_1, HLL_1, HHL_1, HLH_1, HHH_1). Afterwards, the same decomposition 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 bi-orthogonal Daubechies 9/7 filter for both spatial and temporal decompositions. In [17], we worked with both regular filter-bank convolution and lifting transform for the 2D-DWT case. In the GPU analysis, we implemented the 2D-DWT using the convolution filtering process and we obtained an speed-up up to 20 against the sequential CPU implementation. In this next section, we extend this work to the 3D-DWT case trying to obtain similar results than in the 2D-DWT case.

As in [17] we perform the convolution-based 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. The extra memory size depends on both, the row size or column size (the larger one), and it 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.

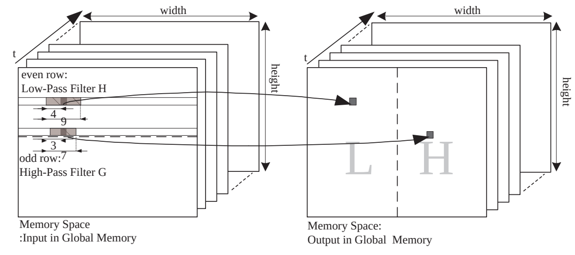
3 GPU 3D Wavelet Transform

In this section, we provide a short overview of GPUs and discuss the steps in our 3D-DWT process where we employ CUDA and decrease the data processing times significantly.

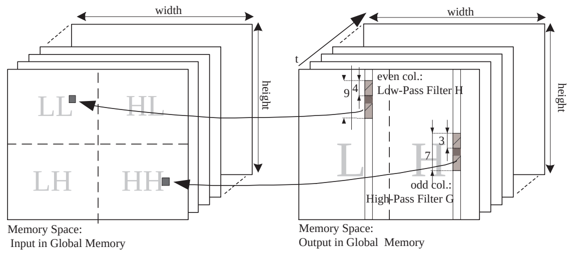
The release of NVIDIAs CUDA API [19, 20] to developers has led to an spectacularly increase of interest in using the GPU capabilities towards faster and more efficient computation in parallel. The GPU serves as a coprocessor to the CPU through the CUDA API and exploits the massive data parallelism on the Single Instruction, Multiple Data (SIMD) architecture of the GPU. Independently operating threads executing CUDA kernels while efficiently sharing high speed memory can be implemented with a set of threads being organized into blocks. It should be noted that to obtain higher bandwidth and overall performance gains, memory sharing between threads must be optimized with very careful programming to ensure the least amount of memory latency between reads/writes. Without effective coordination the entire operation could be compromised leading to little or no performance gains at all.

Two different GPUs are used in this work. The first one is a GTX280 which contains 240 CUDA cores with 1 GB of dedicated video memory. We also show results with a laptop GPU (GMT540) with 96 CUDA cores and 2 GB of video memory. We can appreciate significant differences between both devices that will be reflected in the results showed in Section 4 and 5. In order to compare GPU and CPU performance, we have implemented a sequential version of the 3D-DWT to be executed on a Intel Core 2 Quad Q6600 2.4 GHz.

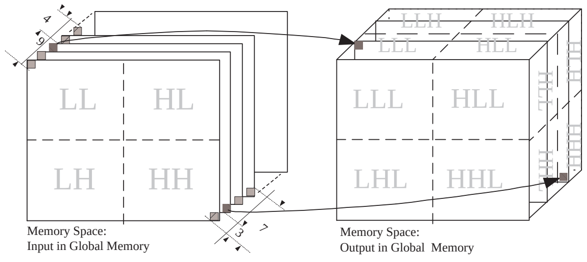
The algorithm used for compute the 3D-DWT in the GPUs is illustrated in Figure 2. Before the first computation step, image data must be transfered from host memory to the global memory of the device. We must transfer the number of frames indicated by the GOP size. As we increase the GOP size, more amount of global memory is needed in the GPU. All frames are stored in adjacents memory positions. In this way, the memory requirements for the GPU is $Width \times \ Height \times frames \times sizeof(float)$ bytes. As showed in Figure 2, in this implementation we are using two memory spaces in the global memory of the GPU: one for the input data and other for the output data after applying the filtering process. In the first step (see Figure 2(a)), each thread computes the row convolution and stores the result in the output memory. For computing the second step, the source data is now the output data obtained in the previous step, so it is not needed a copy of memory data for preparing this step. In the second step (see Figure 2(b)), the column filter is applied and the 2D-DWT is completed for each frame and after that, the output is again in the source space memory. Finally, in the third step, a 1D-DWT is performed to consider the temporal axis. At the end, data must be transfered to the host's memory to proceed with the next



(a) First Step: Row Filter



(b) Second Step: Column Filter



(c) Third Step: Time Filter

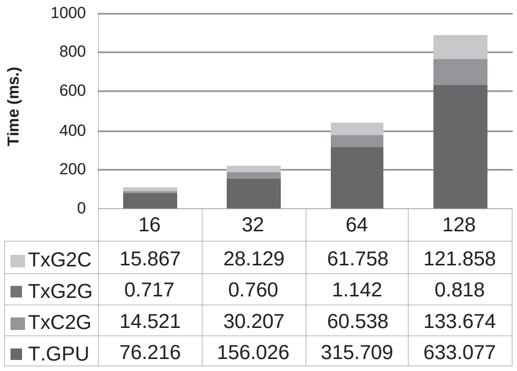
Figure 2: Steps for Computing 3D-DWT in GPUs

GOP.

4 Performance Evaluation

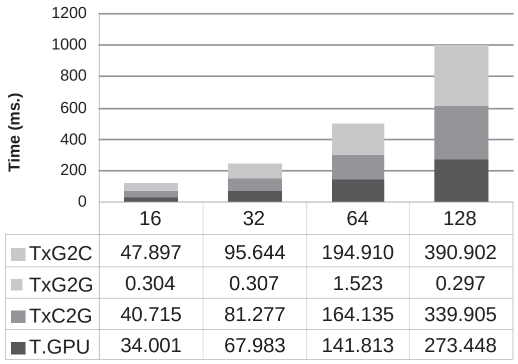
In this section we present the performance evaluation of our GPU-based 3D-DWT algorithm in terms of computational and memory transfer times and the speed-ups obtained when compared to the CPU sequential algorithm. We present results for both previously mentioned GTX280 and GMT540 platforms.

In Figure 3 we present the computational and memory transfer times for both GPU platforms used in this work and for two different video frame resolutions considering GOP sizes varying from 16 to 128. Remark that labels *TxG2C*, *TxG2G*, *TxC2G* and *T.GPU* in Figure 3 refers to memory transfer time from GPU to CPU, internal GPU memory transfer time, CPU to GPU memory transfer time and time to compute the 3D-DWT respectively. As shown in Figures 3 (a) and (b) for the 1280x640 video frame resoluton, the GTX280 is 2.3 times as fast as the GMT540 regarding the GPU computational time. This is mainly due to the greater number of cores available in the GTX280 (2.5 times more cores). However, the global computational time including the memory transfer time is lower in the GMT540 due to the significantly lower memory transfer time, thanks to a second generation of PCI



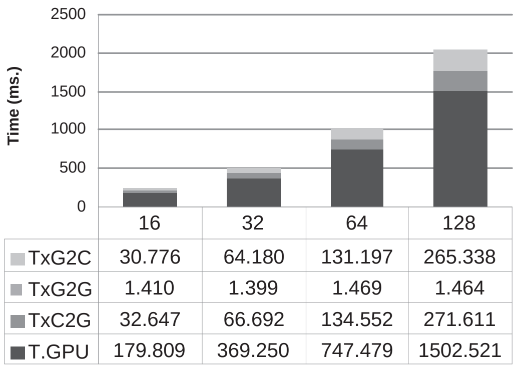
	16	32	64	128
TxG2C	15.867	28.129	61.758	121.858
TxG2G	0.717	0.760	1.142	0.818
TxC2G	14.521	30.207	60.538	133.674
T.GPU	76.216	156.026	315.709	633.077

(a) 1280x640 in GMT540



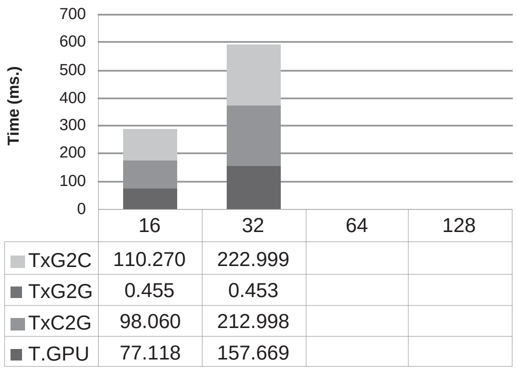
	16	32	64	128
TxG2C	47.897	95.644	194.910	390.902
TxG2G	0.304	0.307	1.523	0.297
TxC2G	40.715	81.277	164.135	339.905
T.GPU	34.001	67.983	141.813	273.448

(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

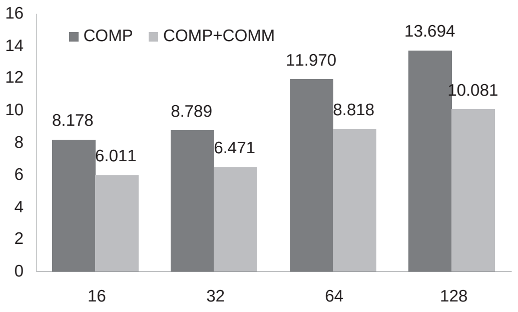
(c) 1920x1024 in GMT540



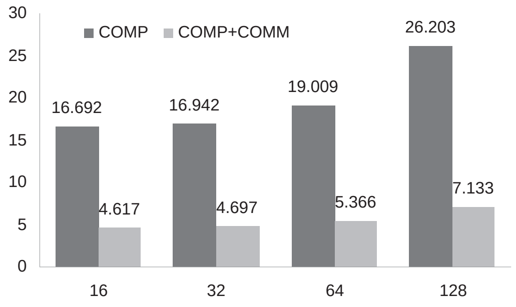
	16	32	64	128
TxG2C	110.270	222.999		
TxG2G	0.455	0.453		
TxC2G	98.060	212.998		
T.GPU	77.118	157.669		

(d) 1920x1024 in GTX280

Figure 3: Transfer times and computational times over GMT540 and GTX280 for different video resolutions



(a) GMT540-1920x1024

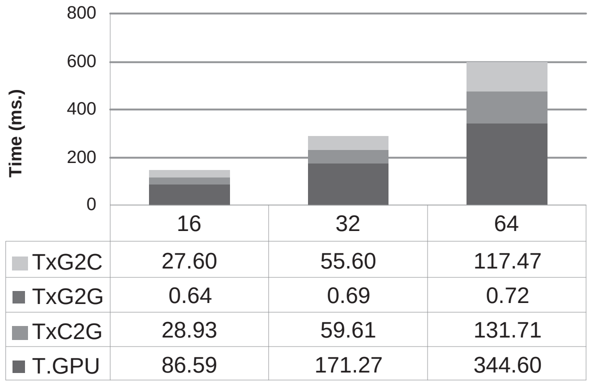


(b) GTX280-1280x640

Figure 4: Speed-up in GPUs with (COMP+COMM) and without(COMP+COMM Communications and COMP respectively)

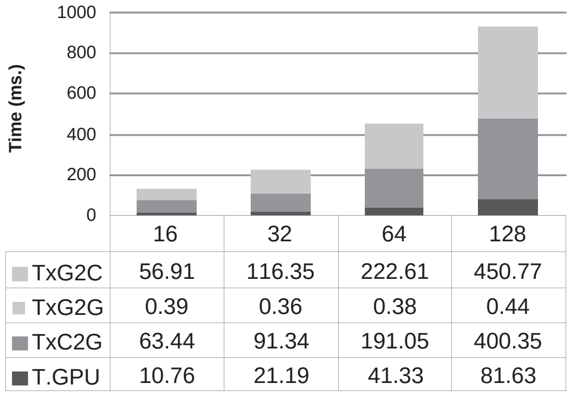
Express bus which improves data transfers. Similar conclusions can be extracted for the 1920x1024 video frame resolution in Figures 3 (c) and (d), where the CPU to GPU and vice versa memory transfer time are 3.3 times greater on average in the GTX280, becoming in this case a botleneck. Computational times for GOP sizes equal to 64 and 128 on the GTX280 are not available because this device has only 1 GByte of video memory, while the GMT540 has 2GBytes. As shown in Figure 3, transfer time inside the GPU, which is required when applying the 2nd and the remaining wavelet decomposition levels, is negligible.

In Figure 4, the GPU-based algorithm speed-ups against the sequential CPU algorithm are presented for both GPU platforms using different GOP sizes for the 1280x640 video frame size in the GTX280 and 1920x1024 video frame resolution in GMT540. As shown, the GTX280 platform obtains better computation speed-ups than the GMT540 platform, being up to 2 times better for the 128 GOP size. But, as previously mentioned, if the CPU to GPU and reverse comunication times are considered, the speed-up drastically drops in the GTX280 platform. Also shown in Figure 4, the speed-up increases as the GOP size increase for both GPUs platforms, even taking into account the communication time. As shown, data transfer between device memory and host memory introduce a significant penalty in using GPUs as general puporse computing. However, this penalty could be avoided if we would overlap communication and computation in the final aplication. Note that the 3D-wavelet transform is only the first step to develop a high level application as the mentioned in Section 1.



	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

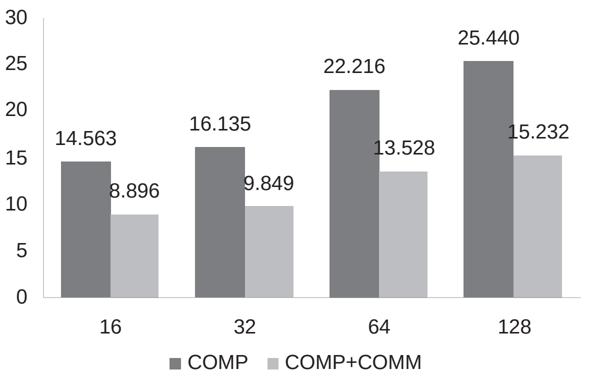
(b) 1280x640 in GTX280

Figure 5: Transfer times and computational times over GMT540 and GTX280 for 1280x640 video resolution using optimized shared memory access

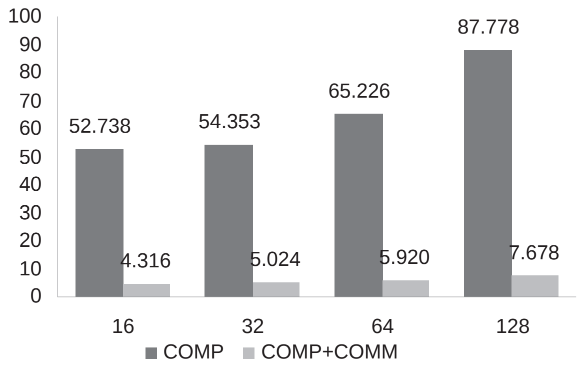
5 Memory access optimization

Algorithm presented in previous section use the global memory to store both source and output in wavelet computation. A reasonable speed-up (x26) has been obtained with high video resolutions. However, we can achieve better performance if we compute the filtering steps from the shared memory. A block of the frame can be loaded into a shared memory array with BLOCKSIZE pixels. The number of thread blocks, NBLOCKS, depends on BLOCKSIZE and video frame resolution. We must note that around the loaded video frame block there is an apron of neighbor pixels that is also required to load in shared memory in order to properly filter the video frame block. We can reduce the number of idle threads by reducing the total number of threads per block and also using each thread to load multiple pixels into shared memory. This ensures that all threads are active during the computation stage. Note that the number of threads in a block must be a multiple of the warp size (32 threads on GTX280 and GMT540) for optimal efficiency. To achieve better efficiency and higher memory throughput, the GPU attempts to coalesce accesses from multiple threads into a single memory transaction. If all threads within a warp (32 threads) simultaneously read consecutive words then single large read of the 32 values can be performed at optimum speed.

In this new algorithm, each row/column/temporal filtering stage is separated into two sub-stages: a) the threads load a block of pixels of one row/column from the global memory into the shared memory, and b) each thread computes the filter over the data stored in the shared memory and stores the results in the global memory. We must not forget about the cases when a row or column processing tile becomes clamped by video frame borders, and



(a) GMT540-1920x1024



(b) GTX280-1280x640

Figure 6: Speed-up in GPUs with and without communications using optimized shared memory access

initialize clamped shared memory array indices with correct values. In this case, threads also must load in shared memory the values of adjacent pixels in order to compute the pixels located in borders.

In Figure 5, we evaluate the new algorithm using the shared memory. As we can see, both GPUs have reduced considerably the execution time. On the other hand, transfer times between CPU and GPU is not affected by the use of shared memory. As an example, for an 1920×1024 video frame resolution in GTX280, we have a performance upper to three times than the time needed when we only use the global memory. Morever, if we compare the new execution times against the CPU sequential algorithm execution time, we obtain better speed-ups than using the global memory, being the improvement 1.83 times on average as fast as the global memory based algorithm for the GMT540 and 3.3 times on average for the GTX280 (see Figure 6).

6 Conclusions

In this paper, we have presented a CUDA-based algorithm that performs the 3D discrete wavelet transform. We have analyzed the behavior of the developed algorithm when running on two different GPU platforms. In a first proposal, we use GPU's global memory and we obtain speed-ups up to 26 for the GTX280 platform and up to 13.6 for the GMT540 platform both for a GOP size of 128 frames. After that, we propose a second implementation using optimized memory access to the GPU's shared memory. This proposal improves the performance in both GPUs and we obtain speed-ups greater than 87x when using a GOP size of 128 frames. In this both proposals, the communication between CPU and GPU and vice versa becomes a bottleneck. In order to avoid this bottleneck we focus our future work on how to overlap communications and computation in the final application to avoid this botleneck. In this way, we could process a GOP (coding, watermarking,etc) while another GOP is transfering from or to the host's memory.

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] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] N. Corporation, “Nvidia cuda c programming guide. version 3.2.”

[21] 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.