Speeding-up the discrete wavelet transform computation with multicore and GPU-based algorithms

V. GALIANO [a], O. LÓPEZ [a] M.P. MALUMBRES [a] and H. MIGALLÓN [a],
[a] *Miguel Hernández University*
Physics and Computer Architecture Department
03202 Elche, Spain

Abstract. In this work we propose several parallel algorithms to compute the two-dimensional discrete wavelet transform (2D-DWT), exploiting the available hardware resources. In particular, we will explore OpenMP optimized versions of 2D-DWT over a multicore platform and we will also develop CUDA-based 2D-DWT algorithms which are able to run on GPUs (Graphics Processing Unit). The proposed algorithms are based on several 2D-DWT computation approaches as (1) filter-bank convolution, (2) lifting transform and (3) matrix convolution, so we can determine which of them better adapts to our parallel versions. All proposed algorithms are based on the Daubechies 9/7 filter which is widely used in image/video compression.

Keywords. Wavelet transform, OpenMP, CUDA, image coding, parallel algorithms.

Introduction

The discrete wavelet transform (DWT) is a mathematical tool that has aroused great interest in the field of image processing due to its nice features. Some of these characteristics are: 1) it allows image multi resolution representation in a natural way because more wavelet subbands are used to progressively enlarge the low frequency subbands; 2) it supports wavelet coefficients analysis in both space and frequency domains, thus the interpretation of the coefficients is not constrained to its frequency behavior and we can perform better analysis for image vision and segmentation; and 3) for natural images, the DWT achieves high compactness of energy in the lower frequency subbands, which is extremely useful in applications such as image compression. The introduction of the DWT made it possible to improve some specific applications of image processing by replacing the existing tools with this new mathematical transform. The JPEG 2000 standard [1] proposes a wavelet transform stage since it offers better rate/distortion (R/D) performance than the traditional discrete cosine transform (DCT).

Unfortunately, despite the benefits that the wavelet transform entails, some other problems are introduced. Wavelet-based image processing systems are typically implemented by memory-intensive algorithms with higher execution time than other transforms. In the usual DWT implementation [2], the image decomposition is computed by

means of a convolution filtering process and so its complexity rises as the filter length increases. Moreover, in the regular DWT computation, the image is transformed at every decomposition level first row by row and then column by column, and hence it must be kept entirely in memory.

The lifting scheme [3,4] is probably the best-known algorithm to calculate the wavelet transform in a more efficient way. Since it uses less filter coefficients than the equivalent convolution filter, it provides a faster implementation of the DWT.

Other fast wavelet transform algorithms have been proposed in order to reduce both memory requirements and complexity, like line-based [5] and block-based [6] wavelet transform approaches that performs wavelet transformation at image line or block level. These approaches increase flexibility when applying wavelet transform and significantly reduce the memory requirements. On the other hand, in [7], authors present a novel way of computing the wavelet transform which they call Symmetric Mask-based Discrete Wavelet Transform (SMDWT). This new wavelet transform algorithm is computed as a matrix convolution, using four matrix masks, one for each subband type, that are built in order to reduce the repetitive computations found in the classical lifting approach. In this scheme, the 2D-DWT is performed in only one pass, avoiding multiple-layer transpose decomposition operations. One of the most interesting advantages of this method is that the computation of each wavelet subband is completely independent.

When designing fast wavelet-based image/video encoders, one of the most computational intensive tasks is the 2D-DWT, which in some cases may take up between 30% and 50% of the overall encoding time (depending on the image size and the number of decompositions levels). In this paper, we analyze and develop several optimized parallel algorithms based on some 2D-DWT computation approaches such as (1) filter-bank convolution, (2) lifting transform and (3) matrix convolution, so we can determine which of them better adapts to our parallel versions. The main goals of the proposed optimizations are to obtain low memory requirements similar to the ones of lifting scheme as well as good computational behavior by exploiting multicore architectures, i.e. shared memory platform, and GPUs co-processing units.

1. Discrete Wavelet Transform (DWT)

DWT is a multi resolution decomposition scheme for input signals, see detailed description in [2]. The original signals are firs decomposed into two subspaces, 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 wavelets $\Psi(t)$. The system downsamples the signal to half of the filtered results in the decomposition process. If the four-tap and non-recursive FIR filters with length L are considered, the transfer functions of H and G can be represented as follows:

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

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

1.1. Lifting-based Wavelet Transform (LDWT)

One of the main drawbacks of the DWT is that it doubles the memory requirements because it is implemented as a filter. A proposal that reduces the amount of memory needed for the computation of the 1D DWT is the lifting scheme [3]. Despite this disadvantage, the main benefit of this scheme is the reduction in the number of operations needed to perform the wavelet transform if compared with the usual filtering algorithm (also known as convolution algorithm). The order of this reduction depends on the type of wavelet transform, as shown in [8].

In Figure 1, we present a diagram to illustrate the general lifting process. The whole process consists of a first lazy transform, one or several prediction and update steps, and coefficient normalization. In the lazy transform, the input samples are split into two data sets, one with the even samples and the other with the odd ones. Thus, if we consider $\{X_i\} = \{\Phi_{n,p}\}$ the input samples at a level *n*, we define:

$$\left\{s_i^0\right\} = \{X_{2i}\} \tag{3}$$

$$\left\{d_i^0\right\} = \{X_{2i+1}\} \tag{4}$$

Then, in a prediction step (sometimes called dual lifting), each sample in $\left\{d_i^0\right\}$ is replaced by the error committed in the prediction of that sample from the samples in $\left\{s_i^0\right\}$:

$$d_i^1 = d_i^0 - P\left(\left\{s_i^0\right\}\right) \tag{5}$$

while in an update step (also known as primal lifting), each sample in the set $\left\{s_i^0\right\}$ is updated by $\left\{d_i^1\right\}$ as:

$$s_i^1 = s_i^0 + U\left(\left\{d_i^1\right\}\right) \tag{6}$$

After *m* successive prediction and update steps, the final scaling and wavelet coefficients are achieved as follows:

$$\{\Phi_{n+1,p}\} = K_0 \times \{s_i^m\} \tag{7}$$

$$\{\Psi_{n+1,p}\} = K_1 \times \{d_i^m\} \tag{8}$$

A special case of wavelet filter is the Daubechies 9/7 filter. This filter has been widely used in image compression [9,10], and it has been included in the JPEG2000 standard [1]. In this paper, all the DWT algorithms will be focused on this filter because of its good behavior. The coefficients of the Daubechies 9/7 decomposition filters, $h[n]$ and $g[n]$ are:

$$\begin{aligned} h\,[n] = {} & 0.026749, -0.016864, -0.078223, 0.266864, 0.602949, \\ & 0.266864, -0.078223, -0.016864, 0.026749 \\ g\,[n] = {} & 0.091272, -0.057544, -0.591272, 1.115087, \\ & -0.591272, -0.057544, 0.091272 \end{aligned}$$

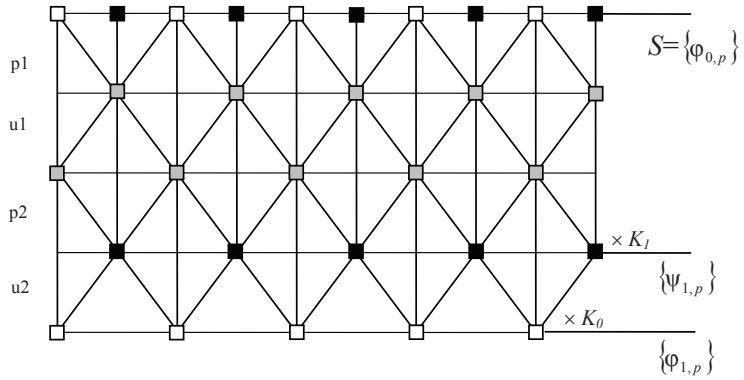


Figure 1. Overview of a wavelet decomposition of an input signal using the lifting scheme for the B9/7 FWT.

while the result of the lifting-based decomposition is:

$$P(z) = \begin{pmatrix} 1 & \alpha\left(1+z^{-1}\right) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \beta\left(1+z\right) & 1 \end{pmatrix} \begin{pmatrix} 1 & \gamma\left(1+z^{-1}\right) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \delta\left(1+z\right) & 1 \end{pmatrix} \begin{pmatrix} \zeta & 0 \\ 0 & 1/\zeta \end{pmatrix} \quad (9)$$

where $\alpha = -1.586134342$, $\beta = -0.052980118$, $\gamma = 0.882911075$, $\delta = 0.443506852$ and $\zeta = 1.230174105$.

1.2. Symmetric Mask-based Wavelet Transform (SMDWT)

In [7], the authors present a novel way of computing the wavelet transform trying to reduce the computational complexity for the wavelet filtering process. The Symmetric Mask-based Discrete Wavelet Transform (SMDWT) is performed as a matrix convolution, using four matrix derived from the 2D DWT of Daubechies 9/7 floating point lifting-based coefficients. The 2D LDWT lifting scheme requires vertical and horizontal 1D LDWT calculations, and each of the 1D LDWT requires four steps: splitting, prediction, updating, and scaling. Conversely, the four subband 2D SMDWT can be yielded using four independent matrices of size 7×7, 7×9, 9×7 and 9×9 for the 9/7 filter.

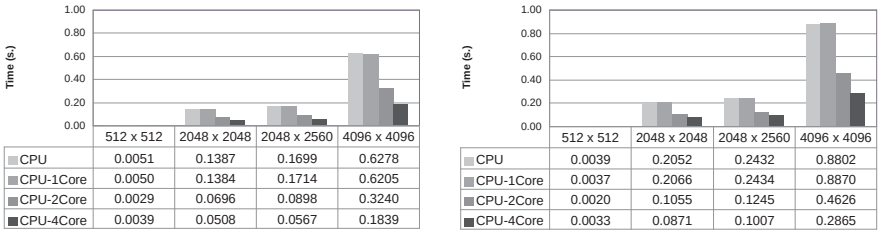
2. Multicore Wavelet Transform Algorithm

We have used the regular filter-bank convolution based on the Daubechies 9/7 filter, in order to develop the optimized parallel 2D discrete wavelet transform (DWT), proposed in [2]. On the other hand, we have used the lifting scheme proposed by Sweldens in [3] in order to develop the optimized parallel 2D lifting wavelet transform (LDWT). We have optimized the proposed sequential algorithms in order to optimize both the computational behavior and the memory requirements. The optimizations performed are based on both the well-known characteristics of the Daubechies 9/7 filter, and a temporary buffer in order to store the computed coefficients in the image memory space. Therefore, the temporary storage buffer should contain either the working row or column.

As we have previously mentioned, we require the image size memory space to store the computed wavelet coefficients. In the wavelet transform based on convolution, an extra memory space to store the current image row/column is required. Moreover, we use the symmetric extension technique to minimize the edge effect in the 2D discrete wavelet transform computation, in which the length of the symmetric extensions depends on the filter lenght. Therefore, for the Daubechies 9/7 filter, a four elements extension and a three elements extension are required for rows and

Image Size	Cores	Extra memory size		Image Size	Cores	Extra memory size	
		Conv.	Lifting			Conv.	Lifting
512 x 512	1	520	1024	2048 x 2560	1	2568	4608
	2	1040	2048		2	5136	9216
	4	2080	4096		4	10272	18432
2048 x 2048	1	2056	4096	4096 x 4096	1	4104	8192
	2	4112	8192		2	8208	16384
	4	8224	16384		4	16416	32768

Table 1. Amount of extra memory size using four-tap filter.



(a) Convolution (b) Lifting

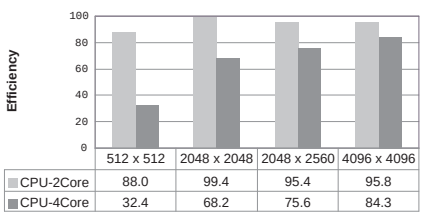
Figure 2. Computational times for multicore fast wavelet transform algorithms.

columns, respectively. On the other hand, in lifting wavelet based transform, we need the extra memory space to store a copy of both one row and one column. Note that, in order to analyze our proposed algorithms, we will compare the computational results obtained by our algorithm against those obtained by recent proposals; one of them is the SMDWT algorithm. It should be noted, regarding the memory requirements, that the SMDWT algorithm requires twice the image size space to perform the four mask filtering.

We have used the OpenMP [11] paradigm in order to develop the parallel algorithms. The multicore platform used is an Intel Core 2 Quad Q6600 2.4 GHz with 4 cores, where a block of rows and a block of columns has been assigned to one process in each core to compute the wavelet transform, therefore each process (or core) requires the aforementioned amount of extra memory. Note that the objective of this buffer is to store the image pixels to compute the wavelet transform, so we could store the final wavelet coefficients in the same memory space occupied by the image, avoiding in this manner to double the memory requirements. Table 1 shows the amount of extra memory in pixels (i.e. floats in grayscale images) used by each algorithm depending on the number of cores used.

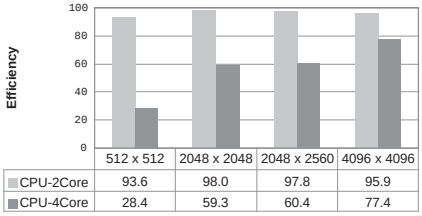
We have tuned the algorithms to obtain the best performance on multicore architectures, taking into account that these algorithms are characterized by an intensive use of memory. In Figure 2 we show the computational times obtained for both convolution-based and lifting-based wavelet transform for different images sizes: 512×512, 2048×2048, 2048×2560 and 4096×4096 pixels. Although the memory access bottleneck is the major obstacle to obtain ideal efficiencies, in Figure 2 we can observe that the computational time decreases, except for small images, as we increase the number of processes. Note that each core executes only one process. Working with small images does not achieve good performance because the relationship between computational load and memory accesses, degrading the inherent parallelism.

Figure 3 shows the efficiency obtained for both convolution and lifting methods as we increase the number of cores. We obtain a closely ideal efficiency using 2 cores, while we obtain a good efficiency using 4 cores. Note that the memory access bottleneck gets worse as the number of cores



	512 x 512	2048 x 2048	2048 x 2560	4096 x 4096
CPU-2Core	88.0	99.4	95.4	95.8
CPU-4Core	32.4	68.2	75.6	84.3

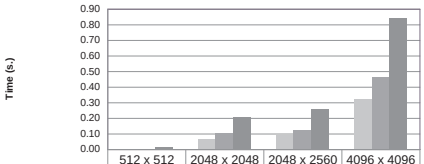
(a) Convolution



	512 x 512	2048 x 2048	2048 x 2560	4096 x 4096
CPU-2Core	93.6	98.0	97.8	95.9
CPU-4Core	28.4	59.3	60.4	77.4

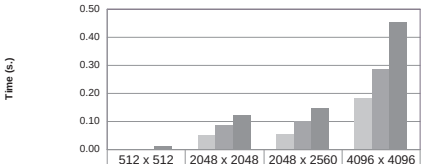
(b) Lifting

Figure 3. Efficiency for parallel tuned wavelet transform algorithms.



	512 x 512	2048 x 2048	2048 x 2560	4096 x 4096
Convolution 9/7	0.0029	0.0696	0.0898	0.3240
Lifting 9/7	0.0020	0.1055	0.1245	0.4626
SMDWT	0.0126	0.2083	0.2582	0.8443

(a) 2 Cores



	512 x 512	2048 x 2048	2048 x 2560	4096 x 4096
Convolution 9/7	0.0039	0.0508	0.0567	0.1839
Lifting 9/7	0.0033	0.0871	0.1007	0.2865
SMDWT	0.0136	0.1227	0.1478	0.4554

(b) 4 Cores

Figure 4. Comparison between Convolution, Lifting and SMDWT algorithms.

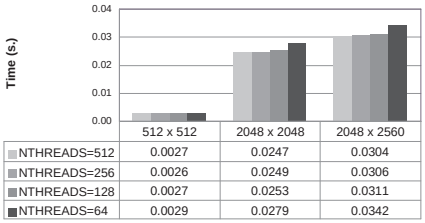
increase because the number of entities that use the memory is greater.

Finally, we have compared our algorithms with a recent and not classical implementation of the DWT called "symmetric mask-based DWT" (SMDWT) [7]. We have developed the method introduced in [7] and also, we have parallelized this reference algorithm. In Figure 4 we present a comparison between convolution, lifting and the SMDWT algorithm using two and four cores, respectively. As it can be seen, our convolution and lifting implementations are 2.5 times as fast as the SMDWT algorithm. Note that the authors in [7] propose the SMDWT algorithm to improve the computational complexity of the lifting scheme and also for the ability of the SMDWT algorithm to compute the four subbands (LL, LH, HL and HH) independently. It is important to remark that the behavior of our algorithms computing the four subbands is similar to the SMDWT behavior only computing the LL subband. On the other hand, in [12], the computational time to compute the DWT for a 4 megapixel size image (2048×2048) is around 9 seconds, while when our convolution algorithm is applied the computational time is 0.07 seconds using 2 cores and 0.05 using 4 cores. Note that the shared memory platform used in [12] is an Intel Core i7 that presents higher computing behavior than the Intel Core 2 Quad Q6600 used in our experiments.

3. CUDA GPU-based Wavelet Transform Algorithm

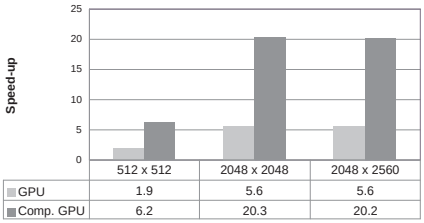
In the previous section, we confirmed that our shared memory parallel algorithm for computing the 2D DWT presents a good behavior. Moreover, we question in this section if the best behaviour can be achieved with Graphical Processor Units (GPUs). We have used the GPU GTX280 that contains 30 multiprocessors with 8 cores in each multiprocessor, and it can work with a maximum of 30K threads. The GTX280 has 1GB of global memory and 16KB of shared memory.

In order to implement a GPU-based algorithm with the same convolution scheme as the one presented in Section 2, the key element is the use of shared memory to store the buffer that contains



	512 x 512	2048 x 2048	2048 x 2560
NTHREADS=512	0.0027	0.0247	0.0304
NTHREADS=256	0.0026	0.0249	0.0306
NTHREADS=128	0.0027	0.0253	0.0311
NTHREADS=64	0.0029	0.0279	0.0342

(a) Computational Times



	512 x 512	2048 x 2048	2048 x 2560
GPU	1.9	5.6	5.6
Comp. GPU	6.2	20.3	20.2

(b) Speed-up

Figure 5. CUDA-based fast wavelet transform.

a copy of the working data row, and the constant memory to store the filter taps $h[n]$ and $g[n]$. As there are no global synchronization mechanisms in the GPU, but only between threads of one block of threads, and, moreover, the shared memory is owned by a block, each block of threads computes a single row or a single column, using the shared memory for the temporary buffer that stores the copy of the working row or the working column. In our proposal we call each CUDA kernel with a one-dimensional number of blocks NBLOCKS and a one-dimensional number of threads NTHREADS. The number of blocks (NBLOCKS) must be equal to or greater than the maximum of the number of rows or the number of columns because each block computes a single row or a single column.

The shared memory of a GPU is visible by the threads of one block, therefore, as we mentioned, each block computes a single row or a single column. The maximum number of threads in one block is 512. So, if the row or the column size is greater than the number of threads, each thread must compute a set of pixels. It is important to remark that each thread copies their working pixels in the shared memory and we perform a synchronization process to start the computation after the data copy is finished. At the end of a block computation we need to perform another synchronization process to start the next block computation.

Figure 5(a) shows the results for the CUDA-based algorithm following a regular filter-bank convolution. In this figure we analyze the behavior when we decrease the number of threads of each block. As it was expected, when the number of threads by block decreases the efficiency decreases, therefore the number of threads to use should be the maximum allowed number of threads or a value close to it. Note that the efficiency loss occurs in the GPU computation step, and not in the GPU-CPU or in the CPU-GPU transfer.

In Figure 5(b) we present the speed-up achieved by the CUDA-based algorithm over the GPU GTX 280 with respect to the sequential convolution algorithm presented in Section 2. Note that we compare the CUDA-based algorithm with respect to an optimized CPU multicore-based algorithm. The speed-up includes both CPU-GPU transfer and GPU-CPU transfer (GPU option), and we present the speed-up only considering the GPU computational time (Comp. GPU). We must remark that the reference algorithm in CPU has quite good performance. For example, in another work related to 2D wavelet transform [13], the CPU reference algorithm is four times slower than ours and, in this case, the speed-up achieved using GPU instead of a GPU for an image of size 2048 is 46 while our speed-up is 20. In conclusion, the speed-up achieved should be analyzed taking into account the reference algorithm performance. Moreover, in [13], the authors use a multicore with similar characteristics, but they use the Intel compiler. However, we can not draw any conclusion because of the different GPUs used. Regarding the results presented in [12], they are obtained using the same GPU, the GTX280. In this work, for a 4 megapixel image size (2048×2048), the computational time to obtain the DWT is around 0.8 seconds, while our algorithm requires only 0.025 seconds, the speed-up being 32.

4. Conclusions

We have presented both multicore-based algorithms (convolution and lifting) and a CUDA-based algorithm (convolution) that performs the two dimensional discrete wavelet transform. We have analyzed the behavior of the proposed algorithms over a shared-memory multiprocessor and over GPU architecture. Furthermore, we have compared our proposals against a recent algorithm called SMDWT. The multicore-based algorithms obtain a speed-up above 1.9 when using two processors and above 2.4 and up to 3.4 when using four processors. Since the best results have been obtained by the convolution algorithm in a multicore platform and also requires a smaller buffer size than the other algorithms, we have developed the corresponding GPU-based algorithm using CUDA and implemented the row/column buffer in the GPU shared memory. The speed-up achieved by the GPU-based algorithm is up to 20. In conclusion, we would like to point out that the use of the multicore platform obtains a good performance, and we obtain a good speed-up in a GPU with respect to the good results obtained in the multicore platform. In future work we will analyze the behavior of the proposed algorithms when using the texture memory of the GPU.

Acknowledgements

This research was supported by the Spanish Ministry of Education and Science under grant DPI2007-66796-C03-03 and the Spanish Ministry of Science and Innovation under grant number TIN2008-06570-C04-04.

References

[1] ISO/IEC 15444-1. JPEG2000 image coding system, 2000.

[2] S. G. Mallat. A theory for multi-resolution signal decomposition: The wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[3] W. Sweldens. The lifting scheme: a custom-design construction of biorthogonal wavelets. *Applied and Computational Harmonic Analysis*, 3(2):186–200, April 1996.

[4] W. Sweldens. The lifting scheme: a construction of second generation wavelets. *SIAM Journal on Mathematical Analysis*, 29(2):511–546, March 1998.

[5] C. Chrysafis and A. Ortega. Line-based, reduced memory, wavelet image compression. *IEEE Transactions on Image Processing*, 9(3):378–389, March 2000.

[6] Y. Bao and C.C. Jay Kuo. Design of wavelet-based image codec in memory-constrined environment. *IEEE Trans. on Circuits and Systems for Video Technology*, 11(5):642–650, May 2001.

[7] Chih-Hsien Hsia, Jing-Ming Guo, Jen-Shiun Chiang, and Chia-Hui Lin. A novel fast algorithm based on smdwt for visual processing applications. In *Circuits and Systems, 2009. ISCAS 2009. IEEE International Symposium on*, pages 762 –765, May 2009.

[8] I. Daubechies and W. Sweldens. Factoring wavelet transforms into lifting steps. *Fourier Analysis and Applications*, 4(3):247–269, 1998.

[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*, 6(3):243–250, 1996.

[10] J.M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12), December 1993.

[11] OpenMP Architecture Review Board. Openmp c and c++ application program interface, version 2.0. March 2002.

[12] D. Wippig and B. Klauer. Gpu-based translation-invariant 2d discrete wavelet transform for image processing. *International Journal of Computers*, 5(2):226–234, 2011.

[13] 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.