Randa Khemiri*, Fatma Ezahra Sayadi, Haythem Bahri, Marwa Chouchene, Mohamed Atri

J. Electrical Systems 12-3 (2016): 490-499

Regular paper

Implementation and Comparison of the Lifting 5/3 and 9/7 Algorithms in MatLab on GPU

Journal of Electrical Systems

In order to accelerate the Discrete Wavelet Transform DWT, we have implemented and compared the lifting "Le Gall5/3" and "Cohen-Daubechies-Feauveau9/7" (CDF9/7) algorithms on a low cost NVIDIA's GPU. The suggested implementation is realized in MatLab using the in-house parallel computation toolbox (PCT). Our experimental results indicate, that the speedup is proportional to the image size until it attains a maximum at 2048^2 pixels, beyond these values the curve decreases. The performance with GPU enhances above a factor of 2~3 compared with CPU.

Keywords: Discrete Wavelet Transform (DWT); Le Gall5/3; Cohen-Daubechies-Feauveau9/7; GPU; CPU; MatLab.

Article history: Received 12 May 2016, Accepted 3 August 2016

1. Introduction

JPEG2000 is the latest image compression standard from Joint Photographic Experts Group (JPEG). The important and computationally demanding part of this standard is the Discrete wavelet Transform (DWT) [1, 2]. The discrete wavelet transform has fully demonstrated its efficiency when utilized as decorrelator for the compression of the images and the signal [3]. It has meaning benefits over other processes based on linear transformations such as discrete cosine transform (DCT) carried by the standard JPEG [4]. The discrete wavelet transform, was supported by the standards for image coding JPEG2000 for its flexibility [5]. The DWT performances are also higher than those of the DCT taking into account the rate compression / distortion in compression lossless chain. It was designed to satisfy high requirement on image processing which are increasing in the last few years [6]. It is very interesting to reduce DWT computation time to develop fast real-time image encoders. To do so, we will take advantage of the existent hardware resources in current computers such as GPU co-processing units.

In recent years, parallel architecture of current GPUs has become increasingly powerful and more programmable, allowing GPUs to be the main computation device and invade all the fields such as neural networks, physics simulations and even image compression analysis [7, 8]. Wong et al. [5] presented an accelerated DWT implementation on GPU. They evaluated their OpenGL and Cg-based implementation with regard to that of JasPer1 [6]. Tenllado et al. [9] proposed an OpenGL and Cg, DWT implementations. On an NVIDIA 7800 GTX, they obtained speedups varying from 1.2 to 3.4 over an Intel P4 Prescott 3.4 GHz processor. Franco et al. [10] presented an implementation based on

* Corresponding author: R. Khemiri, Laboratory of Electronics and Microelectronics, University of Monastir, Environment Street, 5019 Monastir, Tunisia, E-mail: randa.khemiri@gmail.com
Laboratory of Electronics and Microelectronics, University of Monastir, Environment Street, 5019 Monastir, Tunisia

Compute Unified Device Architecture (CUDA). Using NVIDIA Tesla C870 they achieved speedups varying from 20 to 40.6 in the execution time over a C version on an Intel Core 2 Quad Q6700 (2.66 GHz). Results of Chen et al. [11] show that the implementation of the ADL-based wavelet transforms on GPU attains a speedup 10 times greater than that obtained by the optimized CPU implementation with an AMD ATHLON II X2 240 CPU and a NVIDIA GeForce 8800 GTX 768MB. Our contribution is to implement the DWT "Le Gall 5/3" and "Cohen-Daubechies-Feauveau 9/7" filters on GPU using MatLab in order to be tested and simulated on MatLab/Simulink/Xilinx System Generator tool.

The first section of this paper provides a brief introduction of the whole work. In section 2, an overview of DWT used by JPEG2000 is given placing accent on lifting "Le Gall 5/3"and "Cohen-Daubechies-Feauveau 9/7" schemes. Section 3 briefly describes the GPU computing and the MatLab methods used for the parallel architecture computation. In section 4, the DWT computation using the MatLab parallel architecture is elaborated and discussed using a variety of test images (Cameraman, Barbara, Lena, Peppers).Finally, a conclusion is presented in Section 5.

2. Discrete Wavelet Transform Overview

JPEG2000 is an image compression standard. Compared to other standards of compression image, JPEG2000 brought greater flexibility. It offers two compression types: lossy and lossless compression. To encode and decode an image in JPEG2000 format four main steps illustrated in Figure 1are performed.

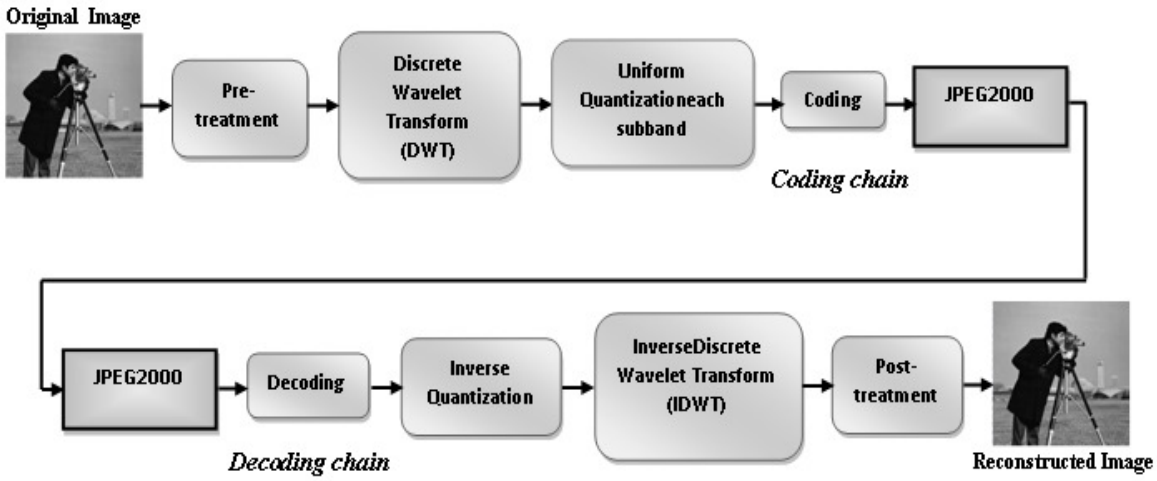
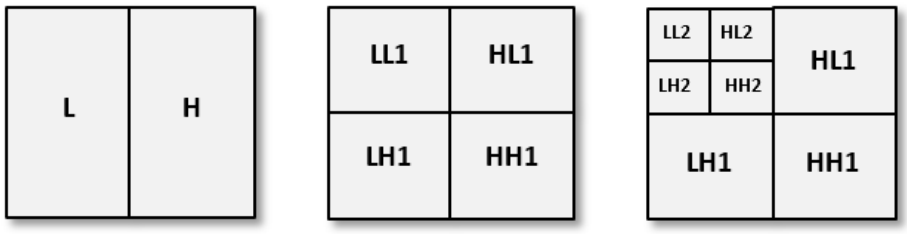


Figure 1. JPEG2000 block diagram [12].

The JPEG2000 is founded on the discrete wavelet transform (DWT), providing higher compression ratio and higher quality. DWT broke down the image at different levels of resolution, into low and high subbands. DWT transform is first applied to all rows then to all columns resulting in four subbands LL, HL, LH, and HH. The LL subbands called low subbands for both row and column filter in gave an approximation of the original signal and can be further transformed recursively as shown in Figure 2.



LL: low subbands for row and column filtering.
HL: high subbands for row filtering and low subbands for column filtering.
HH: high subbands for row and column filtering.
LH: low subbands for row filtering and high subbands for column filtering.

Figure 2. First and second levels 2D of DWT decompositions.

Indeed, four subbands LL1, HL1, LH1 and HH1 are created, in the first level of decomposition. The low-pass subband (LL1) represents the original component subsampled in both horizontal and vertical directions. It is an original component low-resolution version. The other subbands HL1, LH1 and HH1 perform down sampled residual versions of the original image necessary for the original image reconstruction. DWT can be applied on LL1 subband repeatedly to produce four other subbands LL2, HL2, LH2 and HH2 with the same process followed in the first step. This mechanism leads to a pyramidal decomposition.

DWT gave many new ideas to image processing. One of them is the ability to process with or without data loss. The only difference is the use of different wavelet filters. The JPEG2000 standard endorse the use of the symmetrical bi-orthogonal banks filters: Le Gall (5/3) is reversible, it is used for lossless compression, but Cohen-Daubechies-Feauveau (9/7) is irreversible, it is used only in the case of lossy compression. In the filter naming convention (m/n), m refers to the analysis low-pass length and n to the analysis high-pass filters. The wavelet transform can be implemented by the lifting scheme [13, 14] as shown in the following Figure 3.

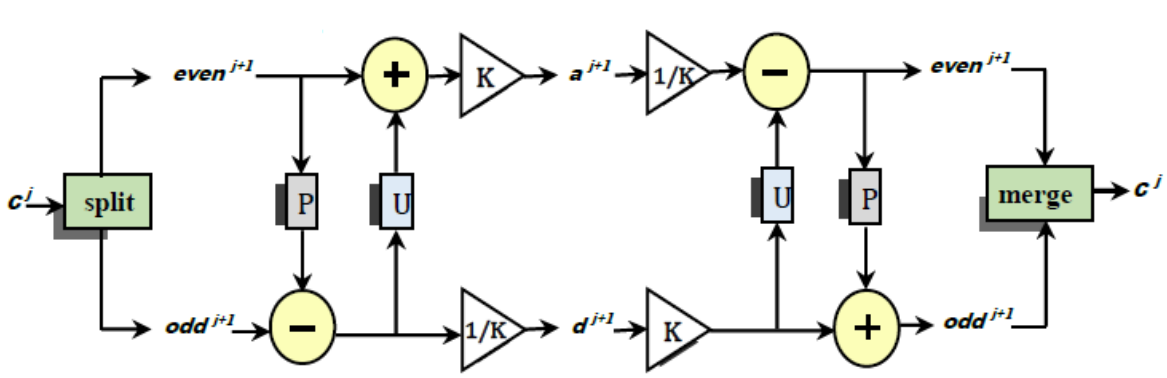


Figure 3. Lifting scheme: Left part: forward lifting. Right part: inverse lifting.

"split" : The trivial wavelet transform,
"merge" : The opposite operation,
P : The prediction step,
U : The update step,
K : The scaling factor.

The Lifting scheme analysis process is as follows. An input signal is divided into even and odd sub-sequences d_n^0 and a_n^0 respectively. These values are further modified using alternating prediction (*P*) and update (*U*) steps. At the end of the prediction step, a prediction error d_n^1 is formed:

$$d_n^1 = d_n^0 - P(a_n^0 + a_{n+1}^0) \tag{1}$$

In the update step, a linear combination of the already modified adjacent odd samples is added to each even sample and updated. The even sequence a_n^1 is formed:

$$a_n^1 = a_n^0 + U(d_{n-1}^0 + d_n^0) \tag{2}$$

The output of the last update stage, a_n^j, is actually a low-pass output of DWT filter and similarly the output of the last prediction stage, d_n^j, is the filter high-pass output. Thus, the wavelet transformation result is a signal split up into low-pass and high-pass subbands.

3. GPU Accelerated Applications

CPU with multi-cores and hyper threading technology has allowed a computationally intensive applications acceleration in different research areas. Latterly another hardware type is gaining attention with its massively parallel computing architecture, it is Graphic Processing Unit, which is gradually applied to a large diversity of communication patterns.

GPU is a highly parallel, multi-threaded, many-core processor with tremendous computing power and large memory bandwidth. The stream GPU processing enables applications and data manipulation to be easily exploited and considered in the form of parallel acceleration for a critical computation. In fact, GPU computing uses a GPU as a co-processor to accelerate CPUs for general-purpose scientific and engineering computing. A CPU consists of several cores, whereas the GPU consists of hundreds of ones. This parallel architecture gives GPU even higher computing performance, and it has obvious advantages over CPU on processing ability. Even if the gain over GPU is not always significant its usage, permits to discharge the CPU resources. Thus, the microprocessor can be used for other tasks.

GPU computing using MatLab: How to perform parallel computing with MatLab: to use computer's graphics processing unit. Several options are available for using GPU in MatLab [15-17]. The gpuArray () function is used to transfer data (a vector, an array …), from the MatLab workspace to the GPU, where the computation is performed. Once the execution is completed, the results are removed from the GPU to the MatLab workspace by the gather () function. This makes the GPU data available as regular variable in the MatLab workspace.

4. Results

DWT offers high compression rate, it is also an important and computationally demanding of the JPEG2000 algorithm. For these reasons we propose a fast DWT implementation using GPU MatLab. We have implemented the lifting "Le Gall 5/3" and "Cohen-Daubechies-Feauveau 9/7" algorithms on CPU and on a low cost NVIDIA GPU with MatLab to achieve speedup in computation. We have compared the performance of both implantations on some test image (Cameraman, Barbara, Lena, Peppers).

Our proposed implementations are performed on representative commercial products from the GPU markets. The GPU is an NVIDIA GeForce GT 640M LE (384 CUDA cores clocked at 499 MHz) with 2048 MB of GPU device memory and a 49 kb per-block shared memory per SM) with NVIDIA driver version 306.97 and CUDA 5.0. We also used a multicore CPU based system with an Intel® core™ i5-3210M CPU (2.5GHz, 2.5GHz, 4.00Go main memory). We have elaborated our GPU code using MatLab version R2013b, "Transferring data between the MatLab workspace and the GPU" to quicken our algorithm.

4.1. CPU and GPU execution time

We proposed the implementation of the "Le Gall 5/3" and "Daubechies 9/7"filters on GPU and CPU computing via MatLab. Before discussing the results, we post the MatLab code (code 1) which was used for the implementation of the two filters on a GPU. The actual changes that have to be made to carry out the simulations on the GPU are in bold format. For using MatLab's PCT an equivalent adjustment with slightly changed syntax is obligatory. The following snippet includes graphical output.

Code 1: An extract of the MatLab code for the "Le Gall 5/3" and "Daubechies 9/7" filters on a GPU using PCT. The differences to a CPU implementation are accentuated by the bold format of the GPU commands.

```
%The code is applied to 5/3 (waveletlegall53)and 9/7 (waveletcdf97)filters
% for double-precision, and tested for different image
ImageFile = 'cameraman.jpg';    % Load the demo image, a photograph or
% a barbara.jpg or lena.jpg or cameraman.jpg or peppers.jpg or baboon.jpg
X = double(imread(ImageFile))/255;  % reads the indexed image in Image File into X
%X=X(:,:,3);
N = size(X);
figure; colormap(gray(256));    % Let's take a quick look at the colormap
subplot(2,2,1);
imshow(X);   % Let's start with a grayscale image from the  Image Processing
%Toolbox and display it using imshow.
axis image; axis off;
title(sprintf('Original image (%dx%d)',N(2),N(1)));
drawnow;
X=imresize(X,10);    % Image Resized
subplot(2,2,2);
import parallel.gpu.GPUArray
X=gpuArray(X);  % Push to GPU memory
tic
Y = waveletlegall53 (X(:,:,1),1);   % 1-stage transform of the intensity channel
```

```
temp_img1=toc;   %Time of the first transformation
imagesc(abs(Y).^0.5);
axis image; axis off;
title('1-stage transform');   %Title of reconstructed image
drawnow;
Y=gather(Y);   % Bring back into MatLab
subplot(2,2,3);
tic
Y = waveletlegall53 (X(:,:,1),2);    % 2-stage transform of the intensity channel
temp_img2=toc
imagesc(abs(Y).^0.5);
axis image; axis off;
title('2-stage transform');
drawnow;
Y=gather(Y);
subplot(2,2,4);
tic
Y = waveletlegall53(X(:,:,1),3);    % 3-stage transform of the intensity channel
temp_img3=toc
imagesc(abs(Y).^0.5);
axis image; axis off;
title('3-stage transform');
drawnow;
Y=gather(Y);
```

The following figures are presented as a result of the proposed code.

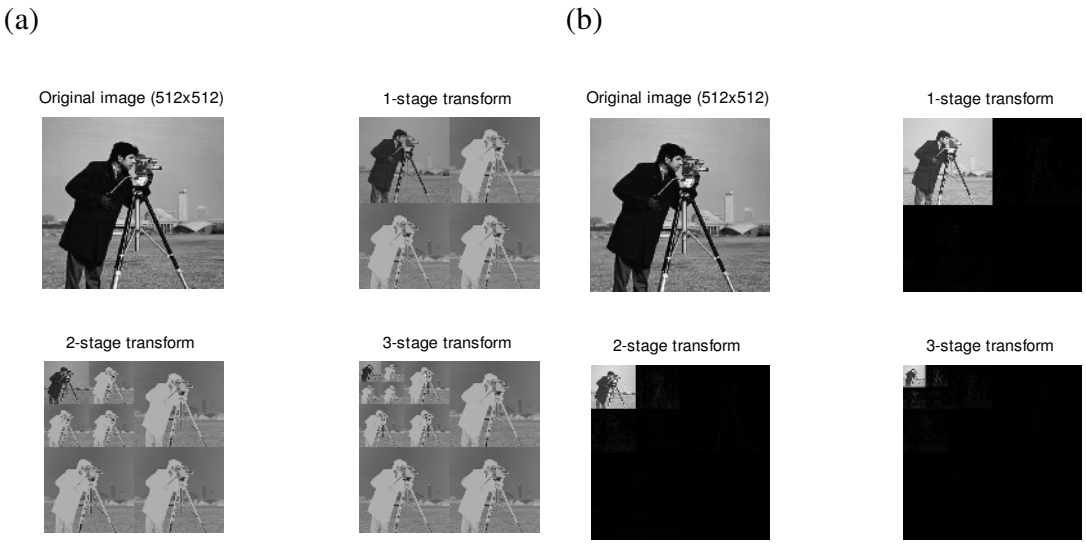


Figure 4. Results of the proposed codes: First, second and third levels 2D wavelet transform of the Cameraman image. a) 5/3 for lossless compression, b) 9/7 for lossy compression.

Table 1.Time cost comparison for the 3-levels wavelet transform between CPU and GPU with "Le Gall 5/3" and "Daubechies 9/7" for four test images (s).

Image size	Filters	Platform	Test images			
			Cameraman	Lena	Barbara	Peppers
256^2	5/3	GPU	0.0138	0.0144	0.0194	0.0167
		CPU	0.0052	0.0049	0.0064	0.0060
		Speedup	0.3768	0.3402	0.3299	0.3590
	9/7	GPU	0.0246	0.0240	0.0227	0.0256
		CPU	0.0049	0.0049	0.0052	0.0051
		Speedup	0.1991	0.2041	0.2290	0.1992
512^2	5/3	GPU	0.0287	0.0271	0.0280	0.0287
		CPU	0.0293	0.0288	0.0285	0.0311
		Speedup	1.0420	1.0627	1.0170	1.0830
	9/7	GPU	0.0476	0.0517	0.0478	0.4700
		CPU	0.0326	0.0329	0.0334	0.0352
		Speedup	0.6840	0.6363	0.6987	0.0748
1024^2	5/3	GPU	0.0665	0.0654	0.0662	0.0693
		CPU	0.1213	0.1166	0.1199	0.1213
		Speedup	2.4139	1.7828	1.8111	1.7500
	9/7	GPU	0.1135	0.1051	0.1236	0.1118
		CPU	0.1330	0.1418	0.1329	0.1459
		Speedup	1.1710	1.3491	1.0750	1.3050
1536^2	5/3	GPU	0.1314	0.1261	0.1275	0.1257
		CPU	0.2634	0.2596	0.2621	0.2729
		Speedup	2.0045	2.0586	2.0556	2.1710
	9/7	GPU	0.2175	0.2108	0.2157	0.2312
		CPU	0.3642	0.3142	0.3037	0.3110
		Speedup	1.6740	1.4905	1.4079	1.3451
2048^2	5/3	GPU	0.2184	0.2103	0.2100	0.2184
		CPU	0.5272	0.5076	0.5138	0.5212
		Speedup	2.4139	2.4136	2.3610	2.3800
	9/7	GPU	0.3570	0.3623	0.3572	0.3791
		CPU	0.6412	0.6605	0.6290	0.6393
		Speedup	1.7960	1.8230	1.7609	1.6863
2560^2	5/3	GPU	0.3371	0.3258	0.3481	0.3340
		CPU	0.7061	0.7541	0.8231	0.7226
		Speedup	2.0946	2.3146	2.3645	2.1634
	9/7	GPU	0.5606	0.5528	0.5590	0.5789
		CPU	0.9010	0.8510	0.9336	0.8954
		Speedup	1.6000	1.7296	1.5394	1.5467
3072^2	5/3	GPU	0.4697	0.4685	0.4567	0.4634
		CPU	1.1137	1.0875	1.0863	1.0789
		Speedup	2.3710	2.3212	2.3785	2.3282
	9/7	GPU	0.7760	0.7999	0.7829	0.8065
		CPU	1.2585	1.3079	1.2686	1.3302
		Speedup	1.6200	1.7138	1.6350	1.6493
3584^2	5/3	GPU	0.7345	1.0267	1.0160	1.0252
		CPU	1.5031	1.4935	1.5123	1.5758
		Speedup	2.0460	1.4546	1.4884	1.5370

9/7	GPU	1.1613	1.5775	1.4089	1.3722
	CPU	1.7655	1.7362	1.6963	1.7277
	Speedup	1.5200	1.1382	0.8378	1.2590

The process of recursive decomposition of the Cameraman image is shown in Figures 4.a and 4.b for 3-levels of resolution for lossless and lossy compression, respectively.

We have evaluated computing time for each platform, and these evaluations are given for different images. One of the reasons why four different images have been used, is whether the image structure has some influence on these measures.

4.1.1. Execution time versus square image size

Table 1 summarizes the execution time of different images versus square image size for 3-levels 2D-forword wavelet decomposition of "Le Gall 5/3" and "Daubechies 9/7" filters. We find that for images of small size, less than 512^2 pixels, the computing with the CPU is faster than the GPU, for all test images. For example, for a Cameraman image of size 256^2 pixels and "Daubechies 9/7" filter, the CPU time, which is equal to 0.0049s, is lower that on GPU, which is equal to 0.0246. This is due to the time spent to copy data from the host's memory to GPU's global memory, which requires a large fraction of total execution time for a small picture sizes. By increasing the size of images, it is clear that the computation with GPU becomes faster than the CPU for all test images.

4.1.2. Performance of lifting filters

On the other hand, Table 1 compare also the execution time by the two lifting "Le Gall 5/3" and "Daubechies 9/7" for the four test images. We notice that the execution time of the filter "Le Gall 5/3" is faster than the "Daubechies 9/7". For example, for 2048^2 pixels Cameraman image size's, "Le Gall 5/3" time on GPU, which is equal to 0.2184s, is lower than that of "Daubechies 9/7", which is equal to 0.357s. And this is because the "Le Gall 5/3" filter requires less computations only half of the coefficients have to be used compared to the "Daubechies9/7". Similar observations were obtained by Chen et al. [11]. Moreover, we note, the same behavior for different test images, "Le Gall 5/3" is faster than the "Daubechies 9/7".

4.2. Speedup factors for 3-level 2D wavelet

The fourth interesting feature of Table 1 is the speedup of "Le Gall 5/3" and "Daubechies 9/7" filter, for our test images versus square image sizes. We note that, for a lower image size to 512^2 pixels, the acceleration factor CPU/GPU is less than one, but for other test image formats, the acceleration factor is greater than one, independently of the numerical precision and the filter type. The speedup is proportional to the image size until it reaches a maximum at the size 2048^2 pixels beyond these values the curve decreases, for each filter used. The speedup versus square image sizes for four testing images, of two filter types are given in Figure 5.

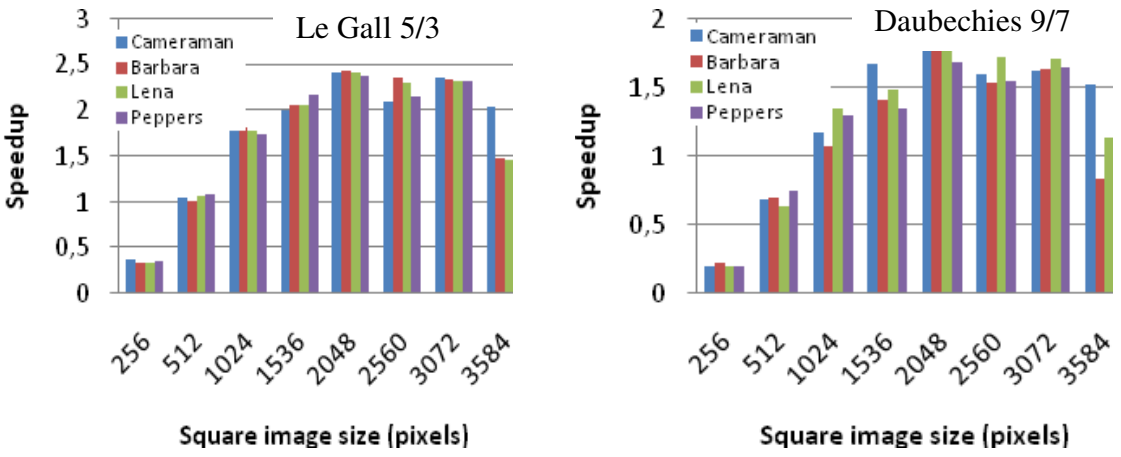


Figure 5. Speedup factor for 3-level 2D.

All test images have similar behaviour conditioned by the image size. For "Daubechies 9/7", the speedup of the tested smallest image (cameraman 256^2) was 0.1991. By increasing the image size, the speedup increases and attains a maximum at 2048^2 pixels equal to 1.7960, after the speedup decreases at 1.52. For "Le Gall 5/3", the speedup of the smallest image we tested (cameraman 256^2) was 0.3768. By increasing the size of image, the speedup increases and reaches a maximum at 2048^2 pixels equal to 2.4139, and then the speedup is decreases at 2.04. The decline in acceleration after 2048^2 was likely due to memory limitations [18, 19].

5. Conclusions

In this paper, we proposed a novel fast wavelet lifting implementation on graphics hardware using MatLab, which is a very important component of JPEG2000 compression algorithm. The efficiency of our GPU based implementation is measured and compared to CPU based algorithms. So, five notable features are evident in our implementation on GPU via MatLab. First, the calculation with GPU becomes faster than the CPU for increased images sizes, regardless of filter used. Second, the execution time of "Le Gall 5/3" is more rapidly than "Daubechies 9/7", for all implementation. Fourth, the speedup is proportional to the image size until it reaches a maximum at the size 2048^2, and beyond these values, the curve decreases for each filter used. The speedup reached a maximum at image size of 2048^2pixels, almost around the 2.39 for "Le Gall 5/3" and around the 1.76 for "Daubechies 9/7". The idea proposed in this work could be also extended to provide video coding high efficiency (HEVC).

References

[1] H. Wendt, S. G. Roux, S. Jaffard, P. Abry, Wavelet leaders and bootstrap for multifractal analysis of images, *J. of Sig. Process*., 89, 1100–1114, 2009.
[2] O. Kao, On parallel image retrieval with dynamically extracted features, *Parall. Comp*., 34, 700–709, 2008.
[3] Information technology JPEG 2000 image coding system part 1: Core coding system. *Int. Organization for Standardization*, ISO/IEC 15444-1, 2004.
[4] D. Taubman, High performance scalable image compression with EBCOT, *IEEE Trans. on Imag. Process.*, 9, 1158-1170, 2000.
[5] T. T. Wong, C. S. Leung, P.A. Heng, J. Wang, Discrete wavelet transform on consumer-level graphics hardware, *IEEE Trans. Multimedia.*, 9, 668–673, 2007

[6] M. Adams, F. Kossentini, Jasper: A software-based JPEG-2000 codec implementation, *International Conference on Image Processing,* 2, 53-56, 2000.
[7] M. Chouchene, F. E. Sayadi, H. Bahri, J. Dubois, J. Miteran, M. Atri, Optimized parallel implementation of face detection based on GPU component*, Journal of Microprocessors and Microsystems,* 39, 393-404, 2015.
[8] R. Khemiri, F. E. Sayadi, M. Atri, R. Tourki, MatLab acceleration for DWT "Daubechies 9/7" for JPEG2000 Standard on GPU*, IEEE Global Summit on Computer & Information Technology GSCIT2014*, 14-16 January Sousse, 1-4, 2014.
[9] C. Tenllado, J. Setoain, M. Prieto, L. Pinuel, F. Tirado, Parallel implementation of the 2D discrete wavelet transform on graphics processing units: Filter bank versus lifting, *IEEE Transactions on Parallel and Distributed Systems,* 19, 299-310, 2008.
[10] J. Franco, G. Bernab, J. Fernndez, M. E. Acacio, A parallel implementation of the 2D wavelet transform using cuda, *16th Euromicro Conference on Parallel, Distributed and Network-Based Processing*, Weimar, Germany, 111-118, 2009.
[11] J. Chen, Z. Ju, C. Hua, B. Ma, C. Chen, L. Qin, R. Li, Accelerated implementation of adaptive directional lifting-based discrete wavelet transform on GPU, *Signal Processing: Image Communication*, 28, 1202–1211, 2013.
[12] P. Bas, Compression d'Images Fixes et de Séquences Vidéo, Cours ENSERG/INPG, Laboratoire des Images et des Signaux de Grenoble France, 2014.
[13] W. Sweldens, The lifting scheme: A construction of second generation wavelets, *SIAM Journal on Mathematical Analysis*, 29, 511-546, 1998.
[14] M. Rabbani, R. Joshi, An overview of the jpeg 2000 still image compression standard, *Signal Processing: Image Communication*, 17, 3-48, 2002.
[15] J. Aceituno, P. Albert, J. Jegard, J. Virey, Programmation sur périphérique GPGPU, Université de Bourgogne. 2010. http://barbuk.org/rapports/rapport_m1_sys3.pdf, Accessed 5 Septembre 2015.
[16] Mathworks Parallel Computing Toolbox, *User's Guide*, R2012a, 2013.
[17] www.mathworks.com/products/parallel-computing, Accessed 5 Septembre 2015.
[18] R. Khemiri, F. E. Sayadi and M. Atri, MatLab-GPU-based 2D-DWT Acceleration for JPEG2000 with Single and Double-Precision, *Indian Journal of Science and Technology*, 9, 1-7, 2016.
[19] D. S. Smith, J. C. Gore, T. E. Yankeelov, E. Brian Welch, Real-Time Compressive Sensing MRI Reconstruction Using GPU Computing and Split Bregman Methods, *Int. Journal of Biomedical Imaging,* Article ID 864827, 1-6, 2012.