Mathematical Problems of Computer Science 41, 122---130, 2014.

Image Denoising Using Wavelet Transform and CUDA

Hovhannes M. Bantikyan

State Engineering University of Armenia
e-mail: bantikyan@gmail.com

Abstract

The discrete wavelet transform has a huge number of applications in science, engineering, mathematics and computer science. Most notably, it is used for signal coding to represent a discrete signal in a more redundant form, often as a preconditioning for data compression. Beginning in the 1990s, wavelets have been found to be a powerful tool for removing noise from a variety of signals (denoising). In this paper the implementation of DWT (Discrete Wavelet Transform)-based denoising algorithm in parallel manner on Graphics Processing Unit is presented, using the CUDA technology.

Keywords: Image denoising, Discrete Wavelet Transform, Haar wavelet, Daubechies wavelet, Parallel computing, GPGPU, CUDA programming.

1. Introduction

The wavelet transform, originally developed as a tool for the analysis of seismic data, has been applied in areas as diverse as signal processing, video and image coding, compression, data mining and seismic analysis. The theory of wavelets bears a large similarity to Fourier analysis, where a signal is approximated by superposition of sinusoidal functions. A problem, however, is that the sinusoids have an infinite support, which makes Fourier analysis less suitable to approximate sharp transitions in a function or signal. Wavelet analysis overcomes this problem by using small waves, called wavelets, which have a compact support. One starts with a wavelet prototype function, called a basic wavelet or mother wavelet. Then a wavelet basis is constructed by translated and dilated (i.e., rescaled) versions of the basic wavelet. The fundamental idea is to decompose a signal into components with respect to this wavelet basis, and to reconstruct the original signal as a superposition of wavelet basis functions; therefore we speak of a multiresolution analysis. If the shape of the wavelets resembles that of the data, the wavelet analysis results in a sparse representation of the signal, making wavelets an interesting tool for data compression and noise removal [1].

There is a wide range of applications in which denoising is important. Examples are medical image analysis, data mining, radio astronomy and many more. Each application has its special requirements. For example, noise removal in medical signals requires specific care, since

denoising which involves smoothing of the noisy signal (e.g., using a low-pass filter) may cause the loss of fine details [2].

2. Discrete Wavelet Transform

The main idea of (the first generation) wavelet decomposition for finite 1-D signals is to start from a signal $c^0 = (c_0^0, c_1^0, \dots, c_{N-1}^0)$, with N samples (we assume that N is a power of 2). Then we apply convolution filtering of c^0 by a low pass analysis filter H and downsample the result by a factor of 2 to get an "approximation" signal (or "band") c^1 of length N/2, i.e., half the initial length. Similarly, we apply convolution filtering of c^0 by a high pass analysis filter G, followed by downsampling, to get a detail signal (or "band") d^1. Then we continue with c^1 and repeat the same steps to get further approximation and detail signals c^2 and d^2 of length N/4. This process is continued a number of times, say J. Here J is called the number of levels or stages of the decomposition. The explicit decomposition equations for the individual signal coefficients are:

$$c_k^{j+1} = \sum_n h_{n-2k} c_n^j, \quad d_k^{j+1} = \sum_n g_{n-2k} c_n^j ,$$

where $\{h_n\}$ and $\{g_n\}$ are the coefficients of the filters H and G. Note that only the approximation bands are successively filtered, the detail bands are left "as is".
This process is presented graphically in Fig. 1, where the symbol $\downarrow_2$ (enclosed by a circle) indicates downsampling by a factor of 2. This means that after the decomposition the initial data vector c^0 is represented by one approximation band c^J and J detail bands $d^1, d^2, \dots, d^J$. The total length of these approximation and detail bands is equal to the length of the input signal c^0 [1].

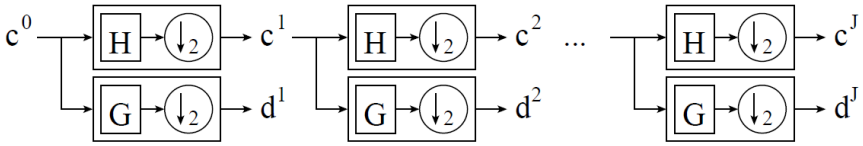


Fig. 1. Structure of the forward wavelet transform with J stages: recursively split a signal c^0 into approximation bands c^j and detail bands d^j.

Signal reconstruction is performed by the inverse wavelet transform: first upsample the approximation and detail bands at the coarsest level J, then apply synthesis filters $\tilde{H}$ and $\tilde{G}$ to them, and add the resulting bands. (In the case of orthonormal filters, such as the Haar basis, the synthesis filters are essentially equal to the analysis filters.) Again this is done recursively. This process is presented graphically in Fig. 2, where the symbol $\uparrow_2$ indicates upsampling by a factor of 2.

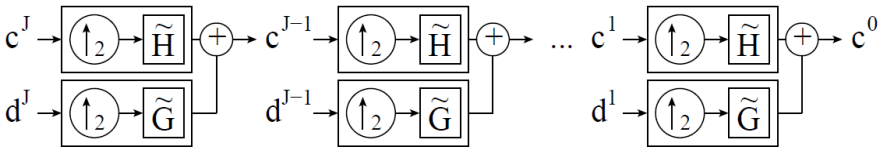


Fig. 2. Structure of the inverse wavelet transform with J stages: recursively upsample, filter and add approximation signals c^j and detail signals d^j.

In case of images we first apply DWT for all rows and then for all columns. In Fig. 3 DWT of Lena image is presented.

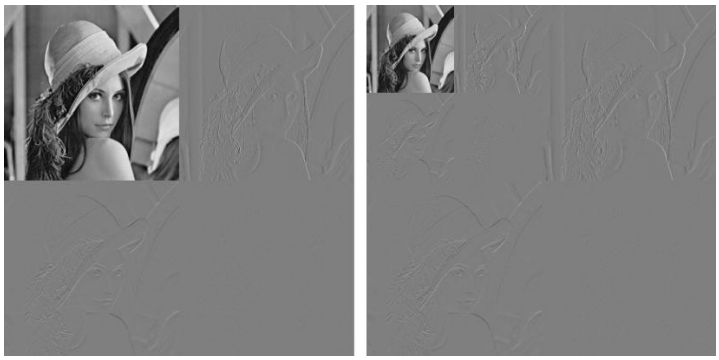


Fig. 3. DWT of Lena image with levels J = 1 (left) and J = 2 (right).

In this paper are implemented Haar and Daubechies 2 (db2) discrete wavelet transforms. Table 1 shows filter coefficients for discrete Haar wavelet transform [9].

Table 1. Filter coefficients of Discrete Haar wavelet

Decomposition		
low-pass filter	$h_0 = 0.7071067812$	$h_1 = 0.7071067812$
high-pass filter	$g_0 = -0.7071067812$	$g_1 = 0.7071067812$
Reconstruction		
low-pass filter	$h_0 = 0.7071067812$	$h_1 = 0.7071067812$
high-pass filter	$g_0 = 0.7071067812$	$g_1 = -0.7071067812$

Table 2 shows filter coefficients for discrete Daubechies 2 wavelet transform [9].

Table 2. Filter coefficients of Discrete Daubechies 2 (db2) wavelet

Decomposition				
low-pass filter	$h_0 = -0.1294095226$	$h_1 = 0.2241438680$	$h_2 = 0.8365163037$	$h_3 = 0.4829629131$
high-pass filter	$g_0 = -0.4829629131$	$g_1 = 0.8365163037$	$g_2 = -0.2241438680$	$g_3 = -0.1294095226$
Reconstruction				
low-pass filter	$h_0 = 0.4829629131$	$h_1 = 0.8365163037$	$h_2 = 0.2241438680$	$h_3 = -0.1294095226$
high-pass filter	$g_0 = -0.1294095226$	$g_1 = -0.2241438680$	$g_2 = 0.8365163037$	$g_3 = -0.4829629131$

3. Wavelet Thresholding

Consider an image $f_{i,j}$ of size N×N. Assume that it is corrupted by independent and identically distributed (i.i.d) zero mean, white Gaussian noise $z_{i,j}$ with standard deviation σ. The corrupted image, denoted by $y_{i,j}$, is given as follows:

$$f'_{i,j} = f_{i,j} + z_{i,j}. \quad (1)$$

The orthogonality of DWT (assuming that orthogonal wavelets are used with periodic boundary conditions) leads to the feature that white noise is transformed into white noise. After applying the two-dimensional orthogonal discrete wavelet transform (DWT) to (1), we have

$$y_{i,j} = w_{i,j} + \varepsilon_{i,j}\,, \quad (2)$$

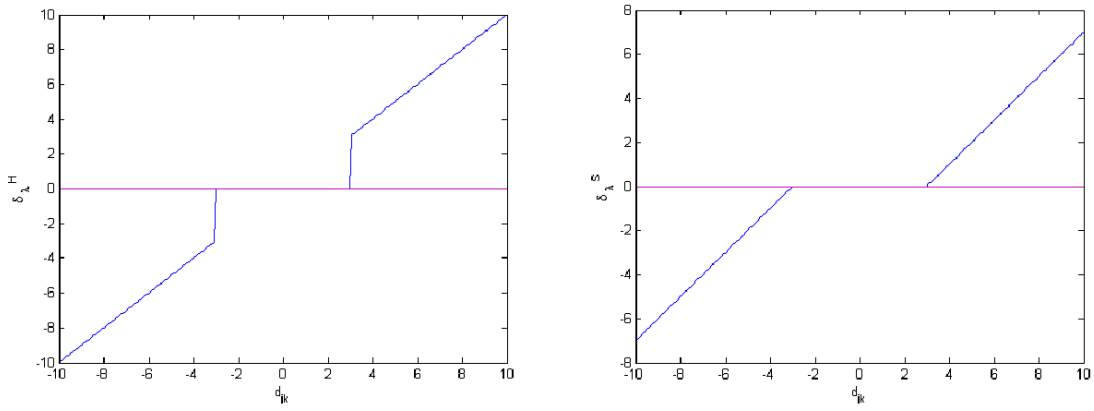
where $y_{i,j} = DWT(f'_{i,j})$, $w_{i,j} = DWT(f_{i,j})$, and $\varepsilon_{i,j} = DWT(z_{i,j})$, i, j = 1, 2, ⋯, N. Since DWT is an orthogonal transform, $\varepsilon_{i,j}$ is also an i.i.d Gaussian random variable, i.e., $\varepsilon_{i,j}$ is N (0, σ^2) [3]. The coefficients of the wavelet transform are usually sparse. That is, most of the coefficients in a noiseless wavelet transform are effectively zero. Therefore, we may reformulate the problem of recovering f as one of recovering the coefficients of f which are relatively "stronger" than the Gaussian white noise background. That is, the coefficients with small magnitude can be considered as pure noise and should be set to zero. The approach, in which each coefficient is compared with a threshold in order to decide whether it constitutes a desirable part of the original signal or not, is called wavelet thresholding. The thresholding of the wavelet coefficients is usually applied only to the detail coefficients $d_{j,k}$ of y rather than to the approximation coefficients $c_{j,k}$, since the latter ones represent 'low-frequency' terms that usually contain important components of the signal, and are less affected by the noise. The thresholding extracts the significant coefficients by setting to zero the coefficients the absolute value of which is below a certain threshold level, which is to be denoted by λ.

The thresholded wavelet coefficients are obtained using either a hard or a soft thresholding rule given respectively by:

$$\delta_\lambda^H(d_{\mathrm{jk}}) = \begin{cases} 0, & if\,|d_{\mathrm{jk}}| \le \lambda \\ d_{\mathrm{jk}}, & if\,|d_{\mathrm{jk}}| > \lambda' \end{cases}$$

$$\delta_\lambda^S(d_{\mathrm{jk}}) = \begin{cases} 0, & if\ |d_{\mathrm{jk}}| \le \lambda \\ d_{\mathrm{jk}} - \lambda, & if\ d_{jk} > \lambda \\ d_{\mathrm{jk}} + \lambda, & if\ d_{jk} < -\lambda \end{cases}.$$

The hard thresholding rule is usually referred to simply as wavelet thresholding, whereas the soft thresholding rule is usually referred to as a wavelet shrinkage, since it "shrinks" the coefficients with high amplitude towards zero. The thresholding rules are depicted in Fig. 4.



(a) Hard-thresholding (b) Soft-thresholding

Fig. 4. Thresholding, λ=3.

Most of the algorithms using the thresholding approach try to estimate the optimal value of λ. However, the first step in these algorithms usually involves the estimation of the noise level σ. Assuming simply that σ is proportional to the standard deviation of the coefficients is clearly not a good estimator, unless f is reasonably flat. A popular estimate of the noise level σ was proposed by Donoho and Johnstone [5]. This is based on the last level of the detail coefficients, according to the median absolute deviation:

$$\hat{\sigma} = \frac{\text{median}(\{|d_{J-1}, k|\} : k = 0,1 \ldots, n-1)}{0.6745}, \tag{3}$$

where n is the number of coefficients in d_{J-1} subband and the factor in the denominator is the scale factor which depends on the distribution of $d_{j,k}$, and is equal to 0.6745 for a normally distributed data. Setting for all levels simply to the universal bound $\lambda_U = \sigma\sqrt{2 \ln s}$, where s is the size of the image equal to N^2, provides good results. This method was proposed by Donoho and Johnstone in [5] and is known as VisuShrink.

4. Parallel Implementation and Results

GPU can process large volume data in parallel when working in single instruction multiple data (SIMD) mode. In November 2006, the Compute Unified Device Architecture (CUDA) which is specialized in compute intensive highly parallel computation is unveiled by NVIDIA. A CUDA-capable GPU is referred to as a device and the CPU - as a host. Thread is the finest grain unit of parallelism in CUDA [8].

Thousands of threads are able to run concurrently on the device. Threads are grouped into the warps. Size of a warp is usually 32 threads. Warp is the smallest unit that can be processed by multiprocessors. Warps scheduled across processors of one multiprocessor are coupled into the thread blocks. Block is a unit of the resource assignment. Typical size of a thread block depends on the particular application what the optimal size of a thread block is to ensure the best utilization of the device. Thread blocks form a grid. Grid can be viewed as a 1-dimensional, 2-dimensional or 3-dimensional array. Fig. 5 is depicting the described hierarchy.

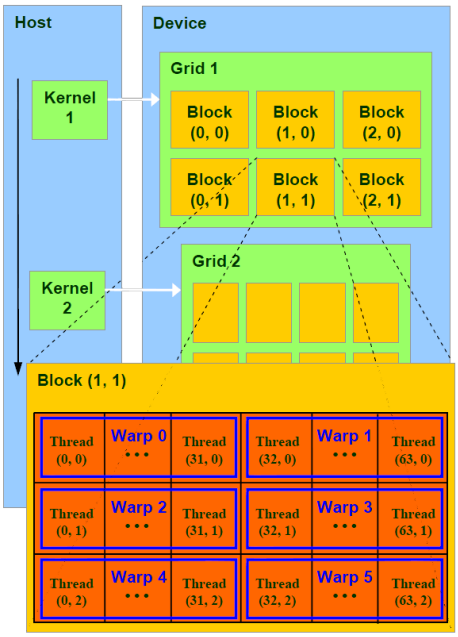


Fig.5. NVIDIA CUDA programming model [8]

First we briefly describe the process to obtain denoised image using 2D-DWT thresholding. Each image is processed as follows:

1. Perform multiscale decomposition on the image corrupted by Gaussian noise. The 2-D orthogonal wavelet transform DWT on a noisy image y is performed up to J^{th} level to generate several subbands.
2. Compute threshold λ at HH_{J-1} subband (using equation 3).
3. For each subband (except the low pass residual), apply λ and then use the soft or hard thresholding method to the noisy coefficients to get the noiseless wavelet coefficients.
4. Finally take the inverse wavelet transform of the resultant image obtained in step 3 in order to obtain a denoised image.

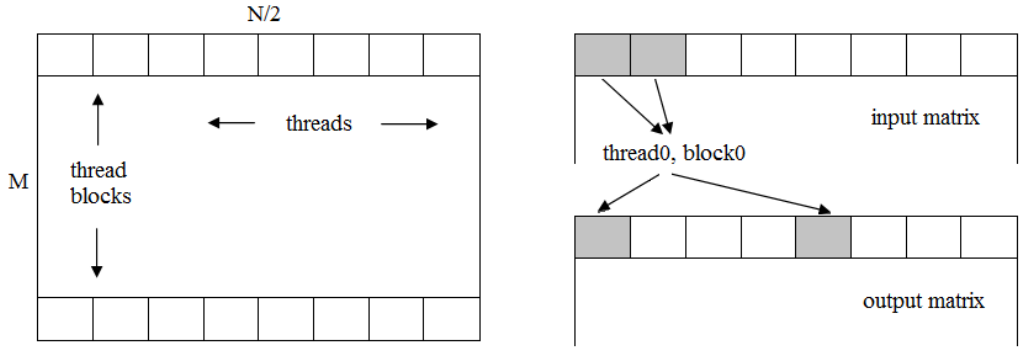


Fig.6. 2D DWT parallel implementation.

Fig. 6 shows how we launch a kernel for DWT. Let's consider an image with size MxN. Number of blocks will be M and number on threads in each block will be N/2. We will have MxN/2 threads working parallel. Each thread will perform k^{th}G and H filter operation. Threads of each block calculate DWT of one row. This single kernel launch will result DWT of all rows of matrix. Then we transpose the matrix, and now by launching the kernel we will get DWT of all columns of matrix. At last we again transpose the matrix.

If we compare the proposed CUDA version of the DWT with a sequential CPU implementation, we can see a large speedup for large images. Table 3 shows the execution time of DWT on CPU and GPU. Experiments are performed on CPU: Intel(R) Core(TM) i3-2100 3,10GHz, and GPU: GeForce GT 630, max threads per block: 1024, max blocks in kernel launch: 65,535. We take into account the time needed to copy data and results to and from the GPU.

Table 3. Haar DWT J=1 execution time on CPU and GPU

Image size	*512x512*	*1024x1024*	*2048x2048*
Sequential on CPU	47ms	218ms	1020ms
Parallel on GPU	16ms	46ms	210ms

To compute the threshold λ we need to estimate σ in HH_J subband. σ estimation is required to calculate median on HH_J subband, and for that task we implement the parallel merge sort algorithm (Fig. 7).

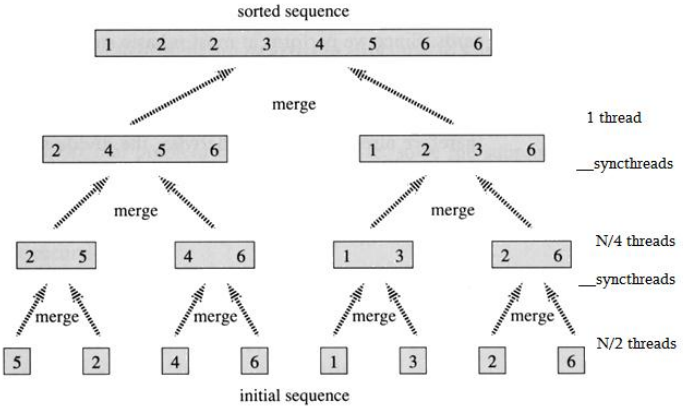


Fig.7. Merge sort parallel implementation

This algorithm performs in $log_2 N$ steps. In each step CUDA kernel runs $N/2^{J-1}$ threads to perform merge operation. To go to the next step we have to wait until the calculation in current step is finished (we call CUDA function __syncthreads).
After computing threshold we perform thresholding operation to all high pass subbands. We choose either hard or soft thresholding method. At last we perform inverse DWT to thresholded. Fig. 8 shows an example of a noisy and denoised image.

Fig.8. Lena original image (left), noisy with variance 0.01 (middle), denoised with Haar DWT J=2 (right).

In Table 4 some experimental results are presented for DWT thresholding for different levels. Here we show PSNR of original and denoised images. Haar and Daubechies 2 (db2) wavelets are used, and tested on noisy images with variance 0.01 (PSNR = 18) and 0.04 (PSNR = 14). Second row in this table corresponds to Fig. 8.

Table 4. PSNR of original and denoised images

Wavelet	*Level*	*Variance = 0.01*	*Variance = 0.04*
Haar	1	23.1161	20.2064
Haar	**2**	**22.9204**	**23.7507**
Haar	3	21.8677	22.8585
Daubechies 2	1	22.5815	19.7278
Daubechies 2	2	23.3683	23.8697
Daubechies 2	3	21.5698	23.4152

5. Conclusion

In this paper we have presented and evaluated an implementation of the 2D discrete wavelet transform and wavelet thresholding for CUDA-enabled devices. A brief introduction to the wavelet transform and thresholding has been provided prior to explaining our parallelization strategy. We have compared the proposed CUDA version of the DWT with a sequential implementation. We have computed PSNRs of denoised images for different transformation levels based on VisuShrink algorithm. As we can see, we gain in performance using parallel GPU algorithm. Parallelizing the sequential and simple algorithms will be beneficial to control code complexity and minimize execution time of the process.

References

[1] W. J. van der Laan, C. J. Andrei and J. B. T. M. Roerdink, "Accelerating wavelet lifting on graphics hardware using CUDA", *Parallel and Distributed Systems, IEEE Transactions*, vol. 22, pp. 132--146, 2011.
[2] (2012) R. Cohen, "Signal denoising using wavelets", [Online]. Available: http://tx.technion.ac.il/~rc/
[3] H. Om and M. Biswas, "A new image denoising scheme using soft-thresholding", *Journal of Signal and Information Processing*, vol. 3, pp. 360--363, 2012.
[4] P. Porwik and A. Lisowska, "The Haar–wavelet transform in digital image processing: its status and achievements", *Machine Graphics and Vision*, vol. 13, pp.79--98, 2004.
[5] D. L. Donoho and J. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage", *Biometrika*, vol. 81, pp. 425--455, 1994.
[6] I. W. Selesnick, *Wavelet Transforms - A Quick Study*, Physics Today magazine, September 27, 2007.
[7] (2012) *NVIDIA CUDA C Programming Guide*, NVIDIA Corp, [Online]. Available: www.nvidia.com
[8] J. Sanders and E. Kandrot, *CUDA by Example*, Addison-Wesley, 2010.
[9] "Wavelet Properties Browser", [Online]. Available: http://wavelets.pybytes.com/

Submitted 20.12.2013, accepted 05.03.2014.

Պատկերում աղմուկի հեռացում վեյվլեթ ձևափոխության և CUDA տեխնոլոգիայի կիրառմամբ

Հ. Բանտիկյան

Ամփոփում

Դիսկրետ վեյվլեթ ձևափոխությունն ունի մեծ թվով կիրառություններ ճարտարագիտության, մաթեմատիկայի և համակարգչային գիտությունների բնագավառներում: Հաճախ այն օգտագործվում է թվային ազդանշանը ավելցուկային տեսքով ներկայացնելու և տվյալների սեղմմանը նախապատրաստելու համար: 1990-ականներից ի վեր վեյվլեթները սկսեցին օգտագործվել որպես տարատեսակ ազդանշաններում աղմուկի հեռացման գործիք: Հոդվածում ներկայացված է աղմուկի հեռացման ալգորիթմ՝ հիմնված դիսկրետ վեյվլեթ ձևափոխության վրա, և զուգահեռացված CUDA տեխնոլոգիայի միջոցով:

Шумоподавление изображения с использованием вейвлет преобразования и CUDA

О. Бантикян

Аннотация

Дискретное вейвлет преобразование имеет огромное количество применений в науке, инженерии, математике и информатике. В частности, он используется для кодирования сигнала, для представления дискретного сигнала в более избыточной форме, в качестве предварительной подготовки для сжатия данных. Начиная с 1990-х годов, вейвлеты стали применяться как мощный инструмент для удаления шума из различных сигналов. В данной статье представлена реализация алгоритма шумоподавления на основе дискретного вейвлет преобразования, параллельно использующая технологию CUDA.