Accelerating Discrete Haar Wavelet Transform on GPU Cluster

Selcuk Aslan, Hasan Badem, Dervis Karaboga, Alper Basturk, Tayyip Ozcan

Erciyes University, Engineering Faculty, Computer Engineering Department
selcukaslan, hbadem, karaboga, ab, tozcan@erciyes.edu.tr

Abstract

The Discrete Haar Wavelet Transform has a wide range of applications from signal processing to video and image processing. Data-intensive structure and easy of implementation make Discrete Haar Wavelet Transform convenient to distribute fundamental operations to multi-CPU and multi-GPU systems. In this paper, the wavelet transform was ported in a compute-efficient way to CPU cluster and programmable GPU cluster by utilizing MPI and CUDA respectively. Experimental studies conducted as part of the parallelization strategies for two-dimensional Discrete Haar Wavelet Transform show that the total running time required to process all rows and columns of an image with different size is significantly decreased on the GPU cluster when compared to the its counterparts on a single CPU, single GPU and CPU cluster. Besides the speedup of the GPU based transform, preliminary analysis also showed that the size of the image is an important parameter on the scalability of the GPU cluster.

1. Introduction

With the increasing growth of technology and driven by the demand for real-time, high resolution graphics, we have to operate a vast amount of information every time which often presents difficulties. The digital information must be stored and retrieved in an efficient and effective manner, in order to make it ready for instant access. Wavelet provide a mathematical way of encoding information in such a way that it is layered according to level of detail [1]. The wavelet transform, originally developed as a tool for the analysis of seismic data, has been applied in areas as diverse signal processing, video and image coding and data mining [1]. 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 [1]. 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. In the theory of wavelet analysis, both continuous and discrete wavelet transforms (DWT) are defied [1]. If discrete and finite data are used such as digital images, it is appropriate to consider the discrete wavelet transform. The discrete wavelet transform is a linear and invertible transform that operates on a data vector whose length is usually an integer power of 2 [1]. The DWT and its inverse can be computed by an efficient filter bank algorithm which includes repeated high and low-pass filter and downsampling for forward transform or upsampling for inverse transform.

Increasing computational complexity with the size of the digital image being processed and the needy of real-time compression or decompression have enhanced the importance of parallelized DWT for multi CPU (Central Processing Unit) and GPU (Graphical Processing Unit) systems [2-5]. Custom hardware and special implementations of the DWT have been developed to meet these computational demands [2-5]. GPUs with programmable, higher floating point computing power and bandwidth when compared to the regular processing units have taken attention and widely used by researchers [2-5]. Wong et al. implemented the DWT on consumer level programmable graphics card hardware with the goal of speeding up JPEG2000 compression [6]. Tenllado et al. investigated the performance of the filter bank and lifting schemas of the 2D DWT on GPUs [7]. Different type of wavelets and size of data used in experimental studies and the performance gain was improved between 10 percent and 140 percent [7]. Franco et al. described a Compute Unified Device Architecture (CUDA) based implementation on the DWT on Nvidia Tesla c870 and gained speedups of approximately 20x over a sequential implementation [8]. Laan et al. proposed a hybrid method between row-column and block based methods for DWT and achieved considerable speedups compared to optimized CPU implementations both for 2-D images and 3-D volume data by utilizing CUDA platform [9]. Galiano et al. analyzed the parallel implementations of the 2-D DWT both on a shared memory multiprocessor and GPU platform [10].

In this study, Haar wavelet which is the oldest wavelet introduced by Hungarian mathematician Alfred Haar has been used to investigate the compatibility of the parallelized implementations of the transformation on a GPU-equipped compute nodes. Images with various resolutions have been chosen to analyze the relationship between the size of the data and number of compute nodes. The rest of the paper is organized as follows. Section 2 summarizes the background to Discrete Haar Wavelet Transform. In Section 3 we provide an introduction to Message Passing Interface (MPI) and CUDA. The details of the CUDA based parallelization approach is given in Section 4. Experimental studies are analyzed in Section 5. Finally, conclusions and future works are given in Section 6.

2. Discrete Haar Wavelet Transform

The basic idea of the wavelet transform is to approximate a complex function as a superposition of simpler functions, which are obtained from one prototype function called basic wavelet by conveniently scaling and translating it [1, 6-10]. Haar wavelet or Haar basis is the simplest and the oldest type of wavelet. Like all wavelet transforms, Haar wavelet transform decomposes the information of a discrete signal into approximation and detail sub-signals whose lengths are half of the transformed signal [6-10]. Given a 1-D discrete signal $c^0 = (c_0^0, c_0^0, \dots, c_0^{N-1})$ with N samples, where N is a power of 2, to obtain a approximation c^1 and detail d^1 bands by utilizing wavelet transform, a low-pass filter and a high-pass filter bank denoted as H and G respectively are applied to the c_0 and

filtered bands are downscaled by a factor of 2 [6-10]. Then we continue with approximation signal c^1 and repeat the same filtering procedures, we get a second approximation c^2 and detail d^2 signals. The recursive process continued J times where J is called the number of levels is presented graphically in Fig. 1.

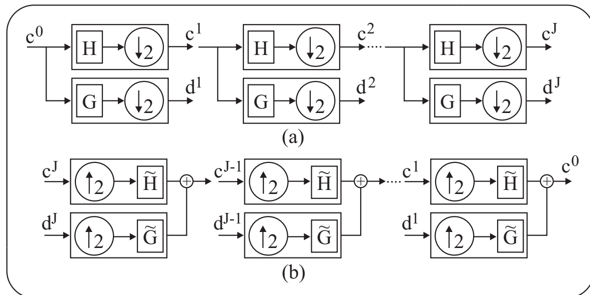

Fig. 1. Forward (a) and inverse (b) wavelet transform

The transformation of an image is a 2-D generalization of the 1-D wavelet [6-10]. It applies the 1-D wavelet transform for each row [6-10]. Next, these transformed rows are treated as if they were creating an image and 1-D transform is applied to every column of the image [6-10]. The resulting values are all detail coefficients and a single overall approximation coefficients. By using these filters in one stage, an image is decomposed into four bands each at half of the original resolution [6-10]. The approximation band shows the general trend of pixel values and detail bands show the vertical-horizontal changes in the images. If these details are very small then they can be set to zero without significantly changing the image. The number of zero valued transformed pixels is an important measure to success of the compression of the original image or signal [6-10].

The lower left band is the LH sub-band that is obtained by applying low-pass filtering to the rows and high-pass filtering to the columns [6-10]. The lower right sub-band is the HH sub-band and obtained by applying high-pass filtering rows and the columns of the image [6-10]. The top right sub-band is the HL sub-band and obtained by applying high-pass filter to rows and low-pass filter to the columns of image [6-10]. The top left sub-band is the LL sub-band which is obtained by applying low pass filter to rows and columns of the image [6-10]. LL sub-band of the image corresponds to the approximate coefficients and if wavelet transform will be employed another time, this approximate coefficients used. In the Fig 2., mentioned approximation sub-band which is used for subsequent wavelet transform and detail sub-bands are illustrated in the image processed by one-level Discrete Haar Wavelet Transform.

3. MPI and CUDA

Numerous programming languages and libraries that differ in their view of the address space available to the programmer have been developed for explicit parallelism. Message-passing programming paradigm that is standardized with the library Message Passing Interface or MPI as it is commonly known is one of the most widely used approach for parallel computers [11, 12]. The key attribute that characterizes message-passing programming paradigm is the partitioned address space associated directly with a particular process [11, 12]. Non-shared address space was accessed by a particular process and if more than one process are needed to perform computations, required part of data must be transferred by a send-receive call between processes [11, 12]. For these send-receive operations, MPI contains a large library of functions that can be called from some other programming languages and macro and type definitions [11, 12].

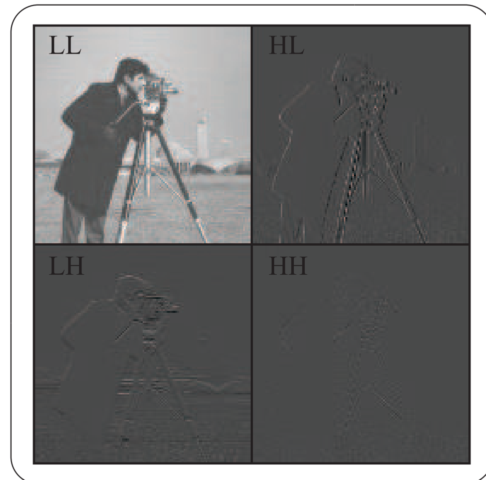

Fig. 2. Approximation and details sub-bands in the image

Nowadays, GPU devices evolved into a highly parallel, multithreaded, many core processor with high memory bandwidth has gained popularity as a main computation device for several applications such as physics simulations, neural network training, image processing and even biological sequence alignment [13, 14]. With their G80 series of graphics processors, NVidia introduced a programming environment called CUDA. CUDA is a general purpose parallel computing platform and allows the GPU to be programmed with a high-level programming language [13, 14]. With the extension of C language of CUDA, programmer defines special functions called kernels which are directly executed N times in parallel by N different CUDA threads [13, 14]. These threads are organized in thread blocks and the collection of blocks of a kernel is called grid. In execution time, each block of the grid is distributed to the Streaming Multiprocessors (SMs) which are arithmetic-logic units of the GPU [13, 14]. This scalable programming model is illustrated for different number of SMs in the Fig. 3.

4. Implementation of the Discrete Haar Wavelet on Distributed Systems

Implementation of the 2-D Discrete Haar Wavelet Transform on CPU cluster is relatively easy when compared to the implementation on GPU cluster. The entire image is equally divided into sub-parts and then these sub-parts directly distributed to the compute nodes in the CPU cluster to process them simultaneously. However, in GPU cluster a second level parallelism should be taken into account to more effectively utilize the compute power of the GPU devices. After related part of the image transferred from host to the global memory of the GPU, suitable kernels should be invoked to perform the desired transform.

In the proposed GPU parallel implementation of the 2-D Discrete Haar Wavelet Transform, two different kernels have been used to transform columns and rows of an image or part of an image. When processing the rows of the stored data, M and N represent the number of the columns and number of rows in the transferred image or part of image and L shows the level of

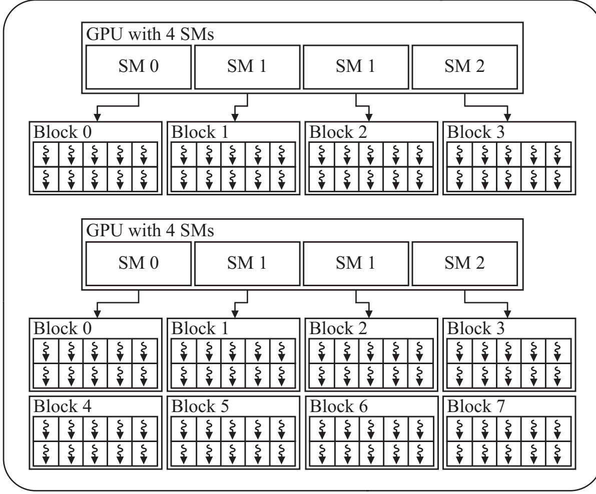

Fig. 3. Executed blocks by streaming multiprocessors

transform, $N/2^L$ blocks with $M/2^{(L+1)}$ threads employed in a synchronization manner that each thread in the block waits after taking half of the sum and difference of two subsequent pixels until all threads in the block complete their tasks. A similar work flow for processing columns of the data is used in the parallelized implementation. All columns of the appropriate part of the image are transformed by utilizing $M/2^L$ blocks with $N/2^{(L+1)}$ threads. The relationship between rows, columns and level of transform to build correct block and thread hierarchy is given in the Fig. 4 below.

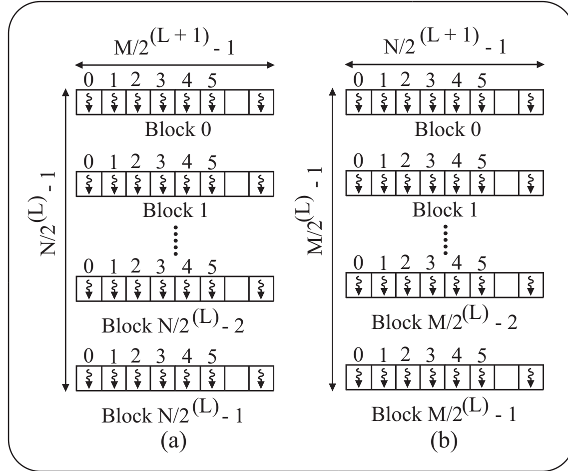

Fig. 4. Thread structure of the row (a) and column (b) kernels

Minimizing data transfer between host and device is one of the most important performance consideration in CUDA programming. For GPU based parallelization of the 2-D Discrete Haar Wavelet, the entire image in the case of using single GPU or equally divided part of the image in the case of using GPU cluster was copied one time from the host to the device memory. After completing transformation, processed image was copied back from the device to host memory. If GPU cluster was used to process image, each GPU node including the master node that organizes the distribution and assembly operations performs a similar copy task as described for the single GPU application. The basic operations dedicated to sending parts of the raw image and receiving them with the integration of MPI and setting up a correct thread block by utilizing level of transform and size of image are summarized in the Algorithm 1.

Algorithm 1 Main steps of the GPU based parallelization

1: MPI_Comm_size(MPI_COMM_WORLD, &size)
2: MPI_Comm_rank(MPI_COMM_WORLD, &rank)
3: **if** $rank\%size == 0$ **then**
4: Read the entire MxN image.
5: Equally divide image to $Mx(N/size)$ sized sub-parts.
6: **Scatter** parts of image to $0...(size-1)$ ranked GPU nodes.
7: **end if**
8: **Copy** $Mx(N/size)$ **image part to GPU global memory.**
9: **for** $L \leftarrow 0...TL$ **do**
10: processRows$\lll M/2^L, (N/size)/2^{(L+1)} \ggg (\ldots)$
11: processColumns$\lll N/2^L, (M/size)/2^{(L+1)} \ggg (\ldots)$
12: **end for**
13: **Copy** $Mx(N/size)$ **image part back to host memory.**
14: **if** $rank\%size == 0$ **then**
15: **Gather** parts of image from $0 \ldots (size-1)$ ranked GPUs.
16: Combine image fragments into MxN image.
17: Write the entire MxN image.
18: **end if**

5. Experimental Studies

We have evaluated serial and parallel implementations of the 2-D Discrete Haar Wavelet Transform in a cluster with 4 compute nodes. Each compute node powered by Asus Geforce GTX 780 Direct CU II Nvidia graphics card which has 2304 CUDA cores and 3 GB of global memory has been equipped with Intel i5 4670 processor with 4 cores running at frequencies between 3.4 GHz and up to 3.8 GHz in turbo mode, 2GB RAM and connected with a standard Gigabit Ethernet. We performed the timing tests using images of different sizes 512x512, 1024x1024 and 2048x2048. For each image with different sizes, three-level 2-D Discrete Haar Wavelet Transform has been applied 10 different times. The elapsed time between start and finish of the transform that includes the data transfer overhead on GPU applications was recorded in terms of miliseconds and average values of 10 different runs was given in Table 1.

Table 1. Average running times of transform

Image Size	Compute Environment			
	CPU	GPU	CPU Clust.	GPU Clust.
512x512	3.9705	0.4392	1.2392	0.2866
1024x1024	23.4592	1.3566	9.4674	0.5162
2048x2048	99.6000	4.9992	34.9194	1.4145

From the results given in the Table 1, its clear that GPU cluster has significantly decreased the running time needed to generate a transformed image which is 8 times smaller than the original image. While the size of the image is increased by a factor of 4, the demanding tasks on sequential pixel values are more efficiently handled by the massively parallel processing power of the GPUs. For a more detailed investigation on the performance gain of the parallelized implementation of the sequential algorithm, speedup and efficiency are two remarkable performance metrics in the literature. Speedup is the ratio of sequential execution time to parallel execution time and efficiency is the ratio of the speedup to number of compute nodes or

Table 2. Speedup compared to other compute environments

Image Size	Compute Environment		
	CPU	GPU	CPU Clust.
512x512	13.8538	1.5324	4.3238
1024x1024	45.4376	2.6281	18.3406
2048x2048	70.4136	3.5343	34.9194

processors. The gained speedup compared to a single threaded CPU, standalone GPU and CPU cluster by using GPU cluster is shown in the Table 2.

Analyzing the speedup and efficiency values gained by utilizing the GPU cluster, an individual comparison being made on a single GPU should be more convenient. In Table 3, this specialized comparison is summarized. While the speedup and efficiency values for 512x512 and 1024x1024 sized images lags behind the optimal values which are 4 for speedup and 1 for efficiency, our distributed 2-D Discrete Haar Wavelet Transform is very close to ideal speedup and efficiency with the values 3.5343 and 0.8836 respectively for 2048x2048 sized image. This type of changing on the mentioned metrics gives important information about the usability of the GPU clusters. Distributing equally divided data chunks to parallel computing nodes is not enough to decrease the running time substantially. If a single GPU is capable of handing the entire data which will be distributed among GPU nodes in the cluster to process its elements simultaneously, using more than one GPU node to transform this data could not improve the running performance as expected. The communication overhead between cluster nodes and necessary data transfers on GPUs deteriorate the speedup and efficiency values for the cluster if a single GPU node is already sufficient for the data being transformed. In addition to this, some adjustments should be done on the GPU which includes the data being transferred between the host and device, accessing patterns of the transferred data and a good balance between multiprocessors of the GPU to maximize the hardware utilization.

Table 3. Speedup and efficiency compared to a single GPU

Image Size	Compute Environment		Speedup	Efficiency
	GPU	GPU Clust.		
512x512	0.4392	0.2866	1.5324	0.3831
1024x1024	1.3566	0.5162	2.6281	0.6570
2048x2048	4.9992	1.4145	3.5343	0.8836

6. Conclusion

In this paper, we analyzed the parallel implementation of the 2-D Discrete Haar Wavelet Transform with different size of images on different types of compute architectures. From the comparison results, it is clear that using a GPU cluster for solving appropriate parts of the problems which have certain inherent parallelism characteristics and require considerable amount of processing power has greatly improved the running time when compared with the implementations on single CPU, single GPU and CPU cluster. Another important conclusion in this study is that if the ratio of arithmetic operations to other operations including data transfers to or from GPU memory, communication between compute nodes and read-write request is low or single GPU is capable of executing the whole data as is done in a GPU cluster, parallelized implementations on both CPU and GPU clusters could not give promising results in terms of running time and speedup. Future work involves a more detailed analysis of the parallelization strategy on different image processing techniques in order to fully exploit all the computing resources provided by the clusters.

7. References

[1] K.H. Talukder, K. Harada, "Haar wavelet based approach for image compression and quality assessment of compressed image", *IJAM*, vol. 36, no. 1, pp: 1:9, 2007.

[2] T. Wong, C. Leung, P. Heng, J. Wang, "Discrete wavelet transform on consumer level graphics hardware", *IEEE T Multimedia*, vol. 9, no. 3, pp: 668-673, 2007.

[3] 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 T Parall Distr*, vol. 19, no. 3, pp: 299-310, 2008.

[4] J. Franco, G. Bernabe, J. Fernandez, M.E. Acacio, "A parallel implementation of the 2D wavelet transform using CUDA", *17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, Weimar, 2009, pp: 111-118.

[5] W.J. van der Laan, A.C. Jalba, J.B.T.M. Roerdink, "Accelerating wavelet lifting on graphics hardware using CUDA", *IEEE T Parall Distr*, vol. 22, no. 1, pp:132-146, 2011.

[6] V. Galiano, O. Lopez, M.P. Malumbres, H. Migallon, "Parallel strategies for 2D discrete wavelet transform in shared memory systems and GPUs", *J Supercomput*, vol. 64, no. 1, pp: 4-16, 2013.

[7] Z. Yang, Y. Zhu, Y. Pu, "Parallel image processing based on CUDA", *International Conference on Computer Science and Software Engineering*, Wuhan, Hubei, 2008, pp: 198-201.

[8] F. Zheng, X. Xu, Y. Yang, S. He, Y. Zhang, "Accelerating biological sequence alignment algorithm on gpu with CUDA", *International Conference on Computational and Information Sciences (ICCIS)*, Chengdu, 2011, pp: 18-21.

[9] J. Barnat, P. Bauch, L. Brim, M. Ceska, "Employing multiple CUDA devices to accelerate LTL model checking", *16th International Conference on Parallel and Distributed Systems*, Shanghai, 2010, pp: 259-266.

[10] J. Zhang, S. You, L. Gruenwald, "Tiny GPU cluster for big spatial data: a preliminary performance evaluation", *35th International Conference on Distributed Computing Systems Workshops*, Columbus, Ohio, 2015, pp: 142-147.

[11] A. Grama, G. Karypis, V. Kumar, A. Gupta, "Introduction to parallel computing", *Addison Wesley*, Harlow, England, 2003.

[12] P. Pacheco, "An Introduction to parallel Programming", *Morgan Kaufmann*, Burlington, USA, 2011.

[13] D.B. Kirk, W.W. Hwu, "Programming massively parallel processors: A Hands-on Approach", *Morgan Kaufmann*, Burlington, USA, 2010.

[14] J. Sanders, E. Kandrot, "CUDA by example: an introduction to general-purpose GPU programming", *Pearson Education*, Boston, USA, 2011.