M. Mohrekesh, S. Azizi, S. Samavi., "Accelerating GPU implementation of contourlet transform," *Machine Vision and Image Processing*, September 2013.

Accelerating GPU Implementation of Contourlet Transform

Majid Mohrekesh, Shekoofeh Azizi, Shadrokh Samavi
Department of Electrical and Computer Engineering,
[1]Isfahan University of Technology, Iran

***Abstract*—The widespread usage of the contourlet-transform (CT) and today's real-time needs demand faster execution of CT. Solutions are available, but due to lack of portability or computational intensity, they are disadvantageous in real-time applications. In this paper we take advantage of modern GPUs for the acceleration purpose. GPU is well-suited to address data-parallel computation applications such as CT. The convolution part of CT, which is the most computational intensive step, is reshaped for parallel processing. Then the whole transform is transported into GPU to avoid multiple time consuming migrations between the host and device. Experimental results show that with existing GPUs, CT execution achieves more than 19x speedup as compared to its non-parallel CPU-based method. It takes approximately 40ms to compute the transform of a 512×512 image, which should be sufficient for real-time applications.**

Keywords—contourlet transform; real-time; convolution; GPU; CUDA.

I. INTRODUCTION

The contourlet transform (CT), which is discrete in nature, is a two dimension transform. CT can efficiently capture image edge information in all directions. Contourlet transform is widely used in image processing applications such as image compression, image retrieval, watermarking, image enhancement and video encoding. Do and Vetterli [1] proposed their new multi-scale and directional representation of images to overcome deficiencies of previous proposed transforms such as wavelet and curvelet. Many applications, which used other transforms, are now using CT to take advantage of its higher performance. Some of these applications need to be processed in real-time [2][3][4][5] and due to the nature of contourlet transform, its computational complexity for large image sizes is a major problem [1]. Resource limitations in wireless networks have caused image processing in wireless transmission to become a challenging task. For example, a wireless face recognition system requires an efficient transmission protocol to maintain high data fidelity. Image compression and denoising should be considered in the design of such a system. There, communication efficiency could be achieved by better image compression and denoising techniques [2]. Edge detection has a prominent role in face recognition algorithms and CT's performance is superior to wavelet when edges are not horizontal or vertical. Hence, image compression in these systems is implemented by CT [2][3]. Moreover, there are other real-time applications for image compression, video compression and video surveillance using contourlet transform such as fingerprint and medical image compression [6][7][8].

Recently, GPUs (Graphics Processing Units) have evolved into extremely powerful computation resources. They have high memory bandwidth, hundreds of floating point processing units, multiprocessors and high pipeline depth. But by loading an algorithm into a GPU one does not necessarily achieve higher performance. It all depends on how well one can extract parallelism out of the application. Although many image processing algorithms are well suited for GPU parallel programming models, there are some algorithms which do not have proper inner mechanisms to be efficiently implemented on a GPU. Any application could be considered as having a parallel portion while the rest of it can be considered as having serial characteristics. Converting the algorithm to have higher parallel part would result in higher GPU efficiency. Contourlet transform as an important image processing algorithm has capability of parallelization at various levels.

Driven by CT's popularity in real-time image processing applications, some works have already been done on efficient implementation and acceleration of this transform [4][5]. Wei et al. presented an implementation of CT used in medical image fusion on a heterogeneous platform (consisting of an ARM processor core and a DSP core) [5]. The ARM core controls the fusion procedure and the DSP core is responsible for the more time-consuming parts of CT. As in any application, hardware implementation is affected by budget and power consumption considerations which could reduce the algorithm's efficiency and accuracy. Furthermore portability of a DSP based implementation of an algorithm between different systems could make this solution unattractive. Advent of General Purpose GPU (GPGPU) cards which are portable and accessible is an appropriate alternative. In [4], the authors proposed a GPU-based implementation of contourlet transform for real time video encoding. In this implementation, the 2D convolutions required are calculated by means of the FFT. FFT-based implementation of 2D convolution is much more efficient than direct time-domain convolution but has two major problems. First, due to the required length adjustments when image size is much larger than the kernel size, it will lose its efficiency. Second, the overhead of FFT and the inverse FFT calculations is time consuming and has negative effect on the speedup.

We propose a new method of exploiting parallelism in CT for better use of GPU architecture. CUDA C was used as a software development tool and NVIDIA GTX570 as the hardware platform. Host part of this implementation is an Intel Core2 Duo CPU. The most computational part of CT, the convolution step, is the first candidate for parallelization and GPU implementation. Due to GPU and CPU data transfer constraints and serial parts of the algorithm, the execution speed was initially suppressed. To overcome this limitation we managed to transport the whole computational tasks to GPU. Furthermore, we parallelized the convolution part of the algorithm. The experimental results showed achievement of speedups in the range of 3 to 19.14x. Moreover, accuracy was maintained within acceptable limits and execution times were proper for real-time applications.

The remainder of this paper is organized as follows. Our implementation and mapping of the algorithm on a GPU is described in Section 2. Experimental results comparing the performance of two platforms are shown in Section 3. Finally, Section 4 concludes the paper.

II. Use Proposed Method

In this section, we describe our proposed parallel algorithm for GPU-based implementation. To extract parallelism, we start with complexity analysis of CT in the following section and then continue with GPU-based implementation.

A. *Contourlet Potential for Parallelism*

First CT consists of two major parts, the Laplacian Pyramid (LP) and Directional Filter Bank (DFB). The LP part is composed of a pair of filters, called analysis and synthesis filters. Also, there are a downsampling and an upsampling blocks in the LP part. The mentioned filters are implemented with a 2D convolution which needs extension. DFB has two building blocks. A 2-channel quincunx filter bank is the first building block, which decomposes a 2-D image into horizontal and vertical directions. The second building block is a shearing operator, which reorders the image samples. In each building block of CT, image is extended and an appropriate filter kernel is convolved with this extended image. The building block of the DFB part is replicated in full binary tree structure. The tree depth is equal to levels of DFB [1]. Filters have higher computational requirements than the sampling and reordering sections. We concentrate on parallelizing these time-consuming filtering steps.

The maximum speedup, for parallel computations, is calculated by Amdahl's law. Suppose a sequential task were to be accelerated by parallelizing a part of it and executing that part with a speedup of p. If f is the fraction of time spent for the non-parallel part, then the maximum speedup as compared with the original sequential program is:

$$maximum\ speedup \leq \frac{p}{1+f.(p-1)} \qquad (1)$$

In our experiments, execution time of the filtering parts is approximately 0.4 of the total time and the maximum speedup will not exceed 2.5 which follows (1). We observed that due to distribution of data between the CPU (host) and GPU (device), there are time consuming transfers of information between them.

In contourlet transform with J levels of Laplacian pyramid decomposition, outputs consist of a lowpass image a_J and J bandpass images b_j, $j = 1,2,\dots,J$. Let l_j be the directional filter bank decomposition level at the j^{th} level of the Laplacian pyramid decomposition. Then each bandpass image b_j is decomposed by l_j level directional filter bank into 2^{l_j} bandpass directional images [1]. The set of l_j values constitute a vector l which defines the number of directions in each level of DFB. For a typical LP and DFB chain, the number of transfers between host and device is twice the number of convolutions needed. Each LP part needs a pair of convolutions and each level j DFB needs a set of 2×2^i convolutions for $1 < i < l_j$. Hence, the number of convolutions needed in one pair of LP and DFB at level j will be $N_{C,j}$ which is equal to $2 + \sum_{i=1}^{l_j} 2^i$.

The total number of convolutions for a full contourlet transform will be

$$\begin{aligned} N_C &= \sum_{j=1}^{J}\left(2+\sum_{i=1}^{l_j} 2^i\right) \\ &= 2J + \sum_{j=1}^{J}\left(2^{l_j+1}-1\right) \qquad (2) \\ &= J + 2\sum_{j=1}^{J} 2^{l_j} \ \in \theta\left(2^{max(l_j)}\right) \ \ni j \in \mathbb{N}, j \leq J \end{aligned}$$

Based on Equation (2) the total number of convolutions is an exponential order of the maximum l_j in the l. This is also the order of number of transfers between the host and device. For a large image, l_j may become a large number if the contourlet transform is to perform a precise decomposition at different levels. Hence, decreasing the number of transfers would be a milestone in accelerating the algorithm.

To reduce the number of transfers the parts which are executed on the host should migrate to GPU to prepare for the parallel execution. Most of the functions that are to be migrated to GPU are for upsampling, downsampling and reordering. Due to different memory configuration of GPU and special memory allocation of threads, mapping data from the host memory to the device memory have a great role in accelerating this process which are elaborated in the implementation part.

Avoiding the time consuming transfer of data between the host and GPU, causes an improvement in the complexity order of our implementation. Now the most time consuming parts of the algorithm take place in the arithmetic operations like summation and multiplication. Since multiplication is more time consuming we use the number of their occurrence as a unit for the computational complexity. The number of products in jth level of convolution depends on the size of the input image and the size of the filter's kernel. For an image of n pixels, the size decreases by one fourth in each level. Suppose the kernel has n_f taps. Then the number of multiplications for each image at level j is obtained as:

$$N_{I,j} = 2n_f.n.\left(\frac{1}{4}\right)^{j-1} + \sum_{i=1}^{2^{l_j}} \frac{2n_f.n.\left(\frac{1}{4}\right)^{j-1}}{2^{l_j}}$$

$$= 4n_f.n.\left(\frac{1}{4}\right)^{j-1} = n_f.n.\left(\frac{1}{4}\right)^{j-2}$$

Hence, for a full set of J levels of transform we have:

$$N_I = \sum_{j=1}^{J} \left(n_f.n.\left(\frac{1}{4}\right)^{j-2}\right) < \frac{16n_f.n}{3}$$

where $n_f.n$ in the above inequality refers to the product of image pixels and filter kernel taps. This product is proportional to the number of multiplication operations for a full image filtering. Hence, the order of the number of multiplications is $N_I \in O(n)$.

It should be noticed that if the filter kernel were separable then the $n_f.n$ factor changes to $2n\sqrt{n_f}$ which decreases the execution times for $n_f > 4$ values, while maintaining the complexity order. Comparing this order of execution with the order calculated in Equation (2), the exponential order is reduced to linear order.

B. GPU Implementation of Contourlet Transform

GPU has different memory types, such as "global", "constant" and "shared" memories. Proper use of these memory spaces for storage of variables, arrays and constants could lead to better exploitation of GPU resources. Based on previous discussion for convolution part, we implement separable filter kernels. At the first step the needed data is copied into the GPU memory space. The input image is placed into the global memory and the filter kernel is placed into the constant memory. Global memory has enough space for a large image and the final results are written back into it. Constant memory has high speed for constant values and is a suitable place for the kernel values. When talking about GPUs, we refer to blocks which form a grid of blocks. Each block would consist of threads. It is expected that blocks could be executed in parallel. Proper partitioning of a task into blocks is the key to proper parallelization.

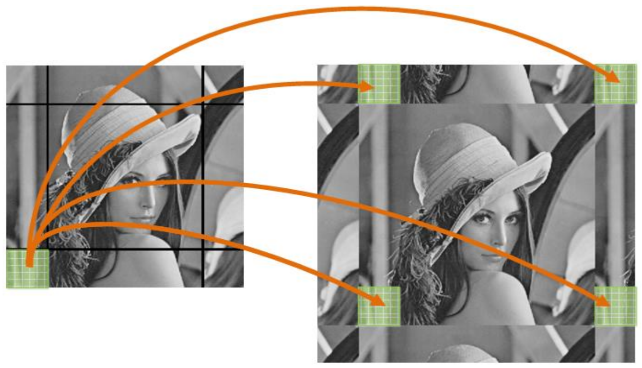
Fig. 1. Periodic extended image formation inside GPU global memory using block threads. Extended image on the right.

In the convolution part, the image in the global memory is partitioned into subsets called tiles. Each tile is sent to the shared memory of a block where convolution is performed on it. Inside each block, pixels are worked on concurrently by threads. Results are written back into the global memory. The manner by which the global memory is accessed has direct effect on the performance of GPU. In an inappropriate design, global memory can act as a limiting factor for speedup. For efficient utilization of the global memory bandwidth, we need to use memory coalescing techniques in conjunction with image tiling. According to coalescence, the best performance is achieved when a warp (32 threads of a block) accesses 128 consecutive words of the global memory [9].

In our GPU implementation, a 16×16 block of threads is used for coalescence considerations. Moreover, other concepts such as memory alignment, tiling boundary cases, and loop unrolling were considered. Had we not used the GPU resources optimally, we could not manage to process such large image sizes. Due to space constraints we do not elaborate on these minor efforts in this paper.

Elimination of data transfers between the host and device and parallel implementation of samplings, reordering and image extension, improved the time complexity of the method. But the extended image, as shown in Fig. 1, is not suitable in terms of coalescence and hence produces many idle threads and lowers the performance of the system. We solved this problem by using the source (original image) rather than destination (extended image) for mapping the threads.

III. Results

In this section we evaluate performance of the proposed parallel implementation of CT on GPU. For thorough assessment of our method, several experiments have been

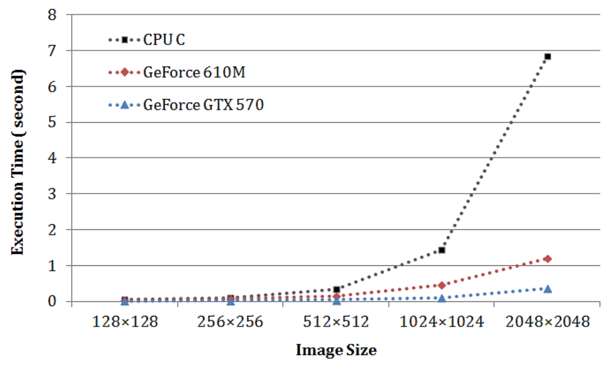

Fig. 2. Execution times for contourlet transform decomposition.

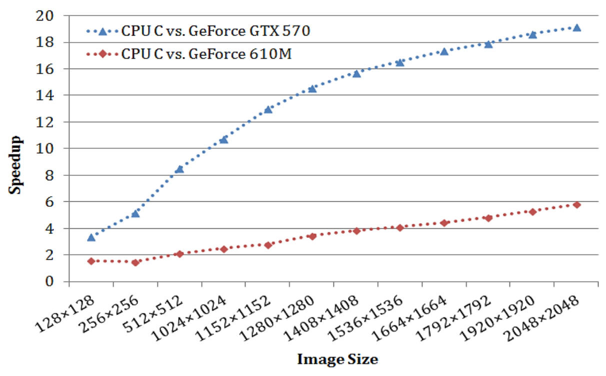

Fig. 3. Speedup ratios for two different GPUs.

conducted using C, CUDA C and MATLAB CT toolbox [10]. Comparisons were made between CT implementation on a CPU and parallelized CUDA on different GPUs. Our experimental platform is an Intel Core2 Duo CPU, as the host, with 4GB memory. The system was equipped with an NVIDIA GeForce GTX570 with 480 cores. An alternative GPU (GeForce 610M) was also used for extension of results' comparisons.

All of the execution times were calculated for one level of LP decomposition with $l_j = 2$. LP decomposition was done using the “9-7”, “5-3” and “Burt” filters. Also for DFB, the “pkva” filters was used. For example, processing of a 512×512 image using these filters takes approximately 40ms, which should meet the requirements of most real-time applications.

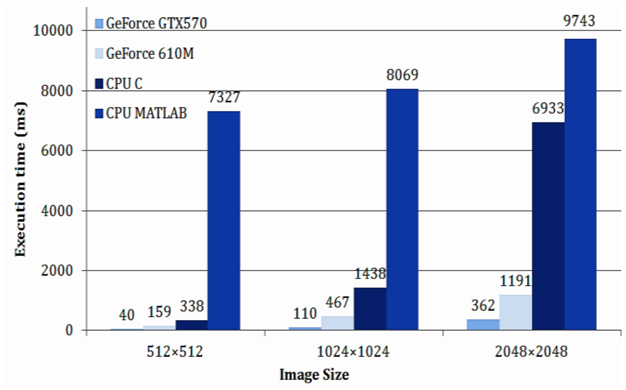

Fig. 4. Execution time comparison between GPU, CPU C and MATLAB implementation of contourlet transform on a CPU.

A more precise study was done on “9-7” and “pkva” filters. Figures 2 and 3 illustrate the execution time and speedup of CT on CPU and GPUs. Figure 2 shows the execution times of CT decomposition for different image sizes. It is shown that execution times of GPU implementations outperform those of CPU, as the image resolution increases. Due to much higher recourses, GTX570 shows better execution times than GeForce 610M. As seen in Fig. 3, speedup improves as the image resolution increases. Comparison of speedups between the two GPUs with different number of cores demonstrated that our method possessed parallelism which had not been fully exploited by the lower resources of GeForce 610M. Graphs of Fig. 3 confirm the hypothesis of Equation (1).

In Fig. 4 execution times of CT decomposition on GPU is compared with C and MATLAB [10] implementation on a CPU. Average relative error of our implementation as compared to what MATLAB CT toolbox produces is in the range of 10-6, which for image processing applications is considered as highly accurate. Figure 4 shows that the proposed approach provides significant speedup of up to 19.14, when compared with CPU C CT. MATLAB CPU implementation is a widely used CT decomposition. Also our CUDA implementation had 443x speedup for 128×128 images as compared with MATLAB CPU implementation.

Due to low memory availability, the GTX570 graphics card does not have the capability to compute the contourlet transform for standard size images larger than 2048×2048.

IV. CONCLUSION

In this paper, we proposed an efficient parallel implementation of the contourlet transform on GPU. Regular GPU implementation of the transform could achieve up to 2.5x speedup. This speedup is not sufficient for most real-time applications. Hence, the transform was carefully investigated. The major steps in this effort were parallelization of filters, reduction of transfers, and proper use of global memory bandwidth by coalescing. Other contributions for CT acceleration were involved with proper memory alignment, tiling boundary cases, and loop unrolling. Our investigations revealed much more potential for parallelization. We managed to convert 95% of the transform procedure to behave parallel. By doing so we were able to obtain speedups of up to 19.14x. This speedup is for our CUDA GPU implementation as compared with an efficient C implementation of the transform on a CPU. Otherwise, we achieved speedups of up to 443x when comparing our results with those obtained from MATLAB toolbox implementation on a CPU.

M. Mohrekesh, S. Azizi, S. Samavi., "Accelerating GPU implementation of contourlet transform," *Machine Vision and Image Processing*, September 2013.

References

[1] M. N. Do and M. Vetterli, "The contourlet transform: An efficient directional multiresolution image representation," IEEE Transaction on Image Processing, vol. 14, pp. 2091–2106, Dec. 2005.

[2] Y. Yan, R. Muraleedharan, X. Ye, and L. Osadciw, "Contourlet Based Image Compression for Wireless Communication in Face Recognition System," IEEE International Conference on Communication, Beijing, pp. 505–509, 19-23 May, 2008..

[3] Y. Yan, and L. Osadciw, "Contourlet Based Image Recovery and De-noising Through Wireless Fading Channels," Conference on Information Science and Systems, The Johns Hopkins University, 16-18 March, 2005.

[4] S. Katsigiannis, G. Papaioannou, and D. Maroulis, "A Contourlet Transform based algorithm for real-time video encoding," SPIE Photonics Europe, Real-Time Image and Video processing Conference, Brussels, Belgium, 16-19 April, 2012.

[5] Y. Wei, Y. Zhu, F. Zhao, Y. Shi, and T. Mo, "Implementing Contourlet Transform for Medical Image Fusion on a Heterogeneous Platform," International Conference on Scalable Computing and Communications, Dalian, pp.115-120, 25-27 September. 2009

[6] S. Esakkirajan, T. Veerakumar, V. Senthil Murugan, R. Sudhakar,"Fingerprint Compression Using Contourlet Transform and Multistage Vector Quantization", International Journal of Biological and Medical Sciences, Vol. 1, No. 2, pp: 140 -147, 2006.

[7] R. Venkatesan and A. B. Ganesh, "Real time implementation on moving object tracking and recognisation using Matlab," International Conference on Computing, Communication and Application, Dindigul, Tamilnadu (India), Februray2012.

[8] N. Karimi, S. Samavi, S. Shirani, H. Talebi, S. M. A. Zaynolabedin, "Contourlet Based Image Compression using Controlled Modification of Coefficients," proceedings of the IEEE CCECE, Canada, pp. 991-994, May 2009.

[9] NVIDIA Corporation, "NVIDIA CUDA Programming Guide 2.3," 2009.

[10] M. N. Do, "Contourlet toolbox," Available at: http://www.ifp.uiuc.edu/minhdo/software/contourlettoolbox.tar

M. Mohrekesh, S. Azizi, S. Samavi., “Accelerating GPU implementation of contourlet transform,” *Machine Vision and Image Processing*, September 2013.