ISMSIT 2017
1st International Symposium on Multidisciplinary Studies and Innovative Technologies Proceedings
(November 2-4, 2017, Tokat, Turkey)
Symposium homepage: www.ismsitconf.org

Parallel Image Processing Algorithms on GPU Environment

Zafer Güler[1], Ahmet Çınar [2] and Erdal Özbay[2*]

[1] *Faculty of Technology, Software Engineering Department, Fırat University, Elazığ/Turkey*
[2] *Faculty of Engineering, Computer Engineering Department, Fırat University, Elazığ/Turkey*
[*] *erdalozbay@firat.edu.tr*

Abstract – The use of GPUs for general purpose applications is not a recent approach, but it was quickly becoming widespread with NVIDIA's CUDA (Compute Unified Device Architecture) architecture based on C programming language in 2007. Algorithms suitable for parallel operation, such as image processing applications, can be implemented much more quickly with the GPU. First of all, this paper summarizes the GPU and CUDA. Furthermore, we implement several conventional image processing algorithms on GPU hardware. The CPU and GPU versions of the implemented algorithms will be compared in terms of application speed. For testing purposes, four basic image processing algorithms are implemented using both CPU and GPU. We have chosen image convolution, histogram equalization, color conversion and median filter. As a result, the GPU version executes much faster than CPU version, especially when the image size is bigger.

Keywords – Image Processing, GPU, CUDA, Parallel Computing

I. Introduction

The limited computational power is not sufficient for solving problems requiring high computation. Particularly for the last 10 years, the problem has been solved by using the graphics processor unit (GPU) instead of the central processor unit (CPU). Parallel computing has become widespread with the use of graphics processors for general programming purposes. Thus, operations that require high computation are accelerated with the use of GPU [1].

Many image processing algorithms suitable for operation in parallel due to their structure. However, in some cases the algorithms implemented by the CPU can be faster [2]. In image processing algorithms for proper use of the GPU, it is important to identity the process of parallelizing and the use of GPU capacity.

With the development of Compute Unified Device Architecture (CUDA) technology, the use of GPU for general purpose applications has become widespread. CUDA provides some enhancements to the C language so that programs can be developed without the need for graphical API knowledge [3].

Nowadays, with the development of camera system, high resolution images began to be obtained. Traditionally, applications using CPUs cannot cope with these high-resolution images. CUDA offers a solution for this type of high-throughput applications [4]. The CUDA C language combined with the GPU's parallel processing capability, reveals miraculous results.

In this publication, four basic image processing algorithms have been implemented using CPU and GPU hardware. These four algorithms are image convolution, histogram equalization, color conversion and median filtering. The remaining of paper is organized as follow: section 2 brief description of CUDA architecture, image convolution, histogram equalization and median filtering. Section 3 shows the execution time of image processing algorithms on CPU and GPU environment. The last section is conclusion and further work.

II. Materials and Method

CUDA architecture is an architecture developed by NVIDIA that enables high increases in computing performance by taking advantage of the GPU. Notice that, the GPU architecture has been developed for parallel operations, the cause of the difference arises. The GPUs are designed for the SIMD (Single Instruction Multiple Data) architecture to run multiple threads on the dataset at the same time. For this reason, successful results can only be obtained on parallel data. It is also not necessary to run all parts of the problem in parallel. When large-size data suitable for parallel operation is run on the GPU, operations that are not suitable for parallel operation such as input / output can be executed on the CPU [5].

CUDA architecture supports access to many memory types with different characteristics. These memory types are global, local, shared, texture and registers. As can be seen from many GPU-based research, GPU programs are performing much performance than CPU programs. However, program settings must be carefully selected and careful attention should be given to memory usage in order to achieve sufficient speed increments [1].

A. CUDA Architecture

GPU architecture is quite different from CPU architecture. Basically, the GPU architecture consists of the following hardware blocks [6].

- Memory (global, constant, shared)
- Streaming multiprocessors (SMs)
- Streaming processors (SPs)

Basic structure of GPU is actually SMs. Each SM has N CUDA cores, and each GPU contains one or more SMs. For example, the NVIDIA GTX960Ti display card has a total of 7 SMs and each SM has 192 CUDA cores. More SMs within the GPU mean more CUDA core, so that the GPU can run more operations in parallel at the same time.

Each SM can access a memory region called register. SMs can access this memory location with zero wait time. The size of the register memory area varies from generation to generation. The shared memory region is a region shared by the SM. This region can be thought of as a cache managed by the programmer [6].

Each SM also has texture memory, constant memory and global memory access. Texture memory provides a special access of the global memory area. For example, it can be very useful when accessing 2D or 3D data. Constant memory is only used for read-only values and it is cashed [6]. The global memory area is supported by Graphic Double Data Rate (GDDR) and has cache support along with compute capability 2.0. As you can see, each memory area has different characteristics and scopes.

In Figure 1, the hierarchy of memory locations is given. Each thread has its own private local memory. The thread block has its own private shared memory, which can access all threds in the block. All threads can access the global memory and also access constant and texture memory, which are read-only memory area. Data exchange between the CPU and the GPU is performed with fixed, texture and global memory areas and the lifetimes of these memory areas continue until the application is terminated [4,7].

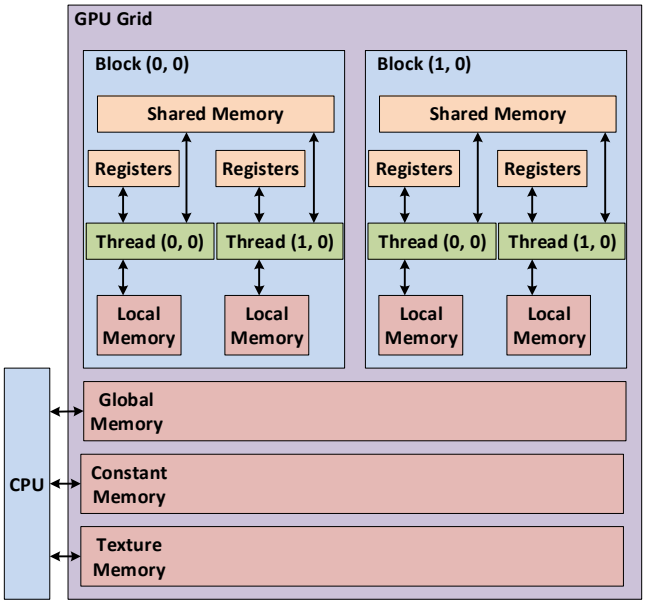

Fig. 1 GPU hardware architecture and memory types [4].

B. Image Convolution

Convolution is used in many scientific fields. Many algorithms like smoothing filter and edge detection use convolutions. Mathematically, convolution is the overlap between two functions. The convolution operation is considered as a blending operation and it is computed by the point-based multiplication of two datasets. The discrete equation of the convolution process in 2-dimensional plane is given below [8].

$$r(i) = (s * k)(i,j) = \sum_{n}\sum_{m} s(i-n, j-m)k(n,m) \qquad (1)$$

In term of image processing, convolution filter is calculated by multiplying the filter weights with the input pixels in the window. This multiplication is suitable for parallel computing devices such as GPU. Thus, results can be obtained much faster than the CPU. Figure 2 shows an example of image convolution process. [9].

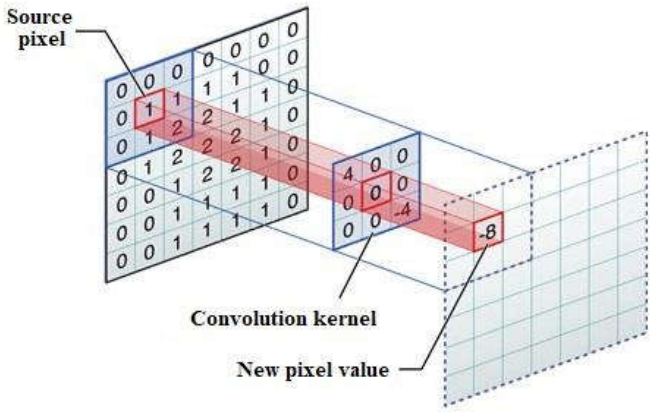

Fig. 2 Image convolution example [9].

A two-dimensional convolution filter requires n * m multiplications (n: width and m: height). It is possible to accomplish convolution filter by separating the two-dimensional filter into rows and columns. This process is called separable filters [10]. Thus, the two-dimensional convolution process can be transformed into two one-dimensional arrays, which can be achieved by n + m multiplication. The separable filter not only provides flexibility in implementation but also reduces algorithmic complexity.

C. Histogram Equalization

Histogram equalization is applied when the color distributions in an image are collected in a certain region and the resulting image details are not very clean. Histogram equalization consists of 3 steps [11]. The first process is the process of calculating the histogram of the image. The second process is to find the cumulative histogram. The cumulative histogram is obtained by summing each color value with its previous color values. Finally, the normalization process is required. In this step, each value of the cumulative histogram is converted into the desired range [12]. Figure 3 shows an example of histogram equalization example.

a) b)

Fig. 3 Histogram equalization example; a) source image; b) Histogram equalization result image

D. Color Conversion

The aim of color models is to transform forms that are accepted as some standard of colors. Every industry uses the most appropriate color model. For example, the RGB color model is used in computer graphics while the YUV color model is used in video systems [13].

YUV model is used for analog color TV broadcasts. The YUV color model is actually derived from the RGB color model. The YUV model consists of luminance (Y) and two color differences (U, V). Luminance is obtained by weighted average of the red green and blue components. The color difference component is calculated from the difference between the luminance component and the blue component [13, 14]. The main advantage of the YUV model is the separation of brightness and color information. Thus, brightness information can be used in image processing algorithms without being influenced by the color component. The following formula is used for conversion from RGB color model in the 0-255 color range to YUV color model [15].

$$Y = 0.299R + 0.587G + 0.114B \quad (2)$$

$$U = -0.169R - 0.331G + 0.5B + 128 \quad (3)$$

$$V = 0.5R - 0.419G - 0.081B + 128 \quad (4)$$

In this publication, conversion from the RGB color model to the YUV color model is performed on the CPU and the GPU, and the execution times are compared. Therefore, only RGB and YUV color models are covered in this section.

E. Median Filter

The median filter is an example of nonlinear noise reduction methods. The median can successfully reduce the noises while preserving the edges. Especially it is very effective at eliminating salt and pepper noise. In the median filter, the neighboring pixels are sorted according to the density value, and the median value becomes the new value of the center pixel [16]. Like the convolution process, the median filter operation is also suitable for parallel operation. Therefore, the median filter application implemented with the GPU is accelerated considerably. Figure 4 shows an example of median filter example.

III. RESULTS

In this section, we present experimental result of image processing algorithms on CPU and GPU environment. We have used a machine equipped with an Intel Core i7-3770 CPU with 8GB RAM. NVIDIA GeForce GTX 660 Ti is used as the graphics card. This GPU includes 7 streaming multiprocessor (SM) with 192 CUDA cores. Each SM has a fixed number of registers and shared memory space. The total memory space of GPU is 2GB and it is supported double precision floating-point arithmetic.

Four image processing applications are tested on CPU and GPU for execution time. First, the image convolution process is tested. Image convolution process has been tested with three different applications. These applications are given below.

- Convolution with texture memory on GPU
- Convolution with separable method on GPU
- Convolution on CPU

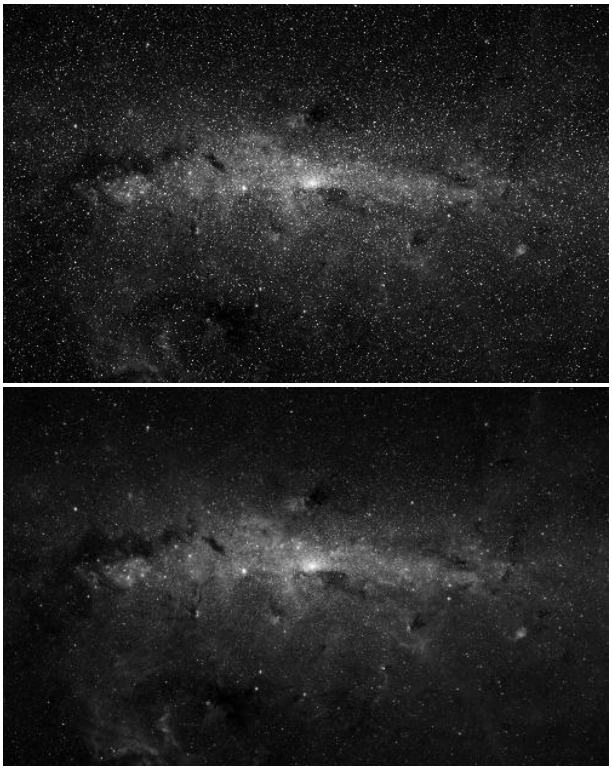

Fig. 4 Median filter example; Original Core Milky Way image (up); Result of median filter (down)

As shown in the Table-1, the GPU version is about 45-190 times faster than the CPU version. As the image size grows, the advantage of the GPU application arises. It is also seen that; texture memory can be access 2D data more quickly.

Table 1. Time comparison of image convolution.

Input Image	CPU(ms)	GPU Texture (ms)	GPU Separable (ms)	Ratio
256x256	1,82	0,04	0,04	45,50
512x512	8,10	0,09	0,11	90,00
1024x1024	34,54	0,24	0,37	143,92
2048x2048	145,76	0,80	1,45	182,20
4096x4096	587,54	3,10	5,72	189,53

Second, the histogram equalization procedure is tested. Here, separate tests are performed on gray level and color images. As you can see from the Table-2, only 2x speedup can be achieved. CPU and GPU runtime are similar in low resolution images. As a result, too much acceleration cannot be achieved with histogram equalization on the GPU. It is thought that more speed increase can be achieved with the optimization operations

Table 2. Time comparison of histogram equalization.

Input Image	CPU Gray*	GPU Gray*	CPU Color*	GPU Color*	Ratio Gray	Ratio Color
256x256	0,19	0,21	0,43	0,41	0,89	1,04
512x512	0,74	0,59	2,09	1,91	1,25	1,09
1024x1024	1,90	1,03	10,25	5,79	1,86	1,77
2048x2048	7,74	3,18	43,77	21,24	2,43	2,06
4096x4096	32,02	17,26	103,13	50,19	1,86	2,05

* (milliseconds)

Third, color conversion has been tested using CPU and GPU. The process of conversion is the conversion from the RGB color model to the YUV color model. As seen in Table-3, as the source image size increases, the achieved acceleration with the GPU increases. Note that the GPU calculation time does not change at 256x256, 512x512 and 1024x1024 image sizes. The reason is that, GPU capacity is not fully used in low-resolution images.

Table 3. Time comparison of color conversion.

Input Image	CPU (ms)	GPU (ms)	Ratio
256x256	0,432	0,838	0,516
512x512	1,432	0,865	1,656
1024x1024	6,135	0,892	6,879
2048x2048	23,822	1,865	12,774
4096x4096	93,351	5,838	15,991

Finally, the Median filter has been tested using CPU and GPU. The Median filter solution has been implemented with two different versions on GPU. These are implemented using shared memory and global memory. As seen in the results in Table-4, the GPU version is about 8-15 times faster than the CPU version. However, as seen from the results obtained, the speed increase cannot be achieved by using shared memory. Theoretically, with the use of shared memory, we can get better results, because we can access the memory area faster. It is thought that this result can be achieved with optimization processes.

Table 4. Time comparison of median filter.

Input Image	CPU (ms)	GPU Shared (ms)	GPU Global (ms)	Ratio
256x256	14,824	2,176	1,824	8,129
512x512	55,941	5,176	5,000	11,188
1024x1024	246,191	15,824	17,647	13,951
2048x2048	816,769	60,385	61,462	13,289
4096x4096	3094,380	205,923	196,615	15,738

IV. CONCLUSION

The aim of this publication is to show that image processing applications are performed much faster using the GPU. Image processing applications can generally be implemented in pieces, so we can run image processing applications in parallel. Four different image processing algorithms have been chosen to show that image processing algorithms on the GPU work faster than the CPU.

The first image processing algorithm is the image convolution algorithm. Largest speed increases are obtained in this application. if we do not take into account the data transfer between the host memory and the device memory, image convolution algorithms could be achieved 190x speedup. Color Conversion and median filter algorithms could be achieved 15x speedup with memory transfer. Only the histogram equalization algorithm cannot achieve a sufficient speed increase. As a future work, the authors are planning to optimize the histogram equalization algorithm.

REFERENCES

[1] J. Cheng, M. Grossman and T. McKercher, Professional CUDA C Programming, Elsevier, John Wiley & Sons, Inc., 2014.

[2] I. Tsmots, O. Berezkyi, I. Ihnatiev and I. Gumovska, “Implementation of image processing algorithms based on GPU,” Scientific and Technical Conference, 2016, p. 27-29.

[3] Z. Guler and A. Cinar, "GPU-based image segmentation using level set method with scaling approach," Computer Science & Information Technology (CS & IT), vol. 3, no. 8, pp. 81-92, 2013. doi:10.5121/csit.2013.3808.

[4] J. Sanders and E. Kandrot, CUDA by example: an introduction to general-purpose GPU programming, Addison-Wesley, 2011.

[5] Z. Güler, A. Çinar and E. Özbay, “Investigation of SIFT, SURF, and GPU-SURF Algorithm for Feature Detection,” ICENS International Conference on Engineering and Natural Science, 2016, p. 578-584.

[6] S. Cook, CUDA Programming: A Developer’s Guide to Parallel Computing with GPUs, Elsevier, Morgan Kaufmann, 2012.

[7] NVIDIA, (2017), CUDA C programming guide. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[8] V. Podlozhnyuk, (2007), Image Convolution with CUDA. [Online]. Available: http://developer.download.nvidia.com/assets/cuda/files/convolutionSeparable.pdf

[9] (2017) Image programming guide: Performing Convolution Operation, [Online]: https://developer.apple.com/library/content/documentation/Performance/Conceptual/vImage/ConvolutionOperations/ConvolutionOperations.html.

[10] J. Shajeena and K. Ramar, “A novel way of tracking moving objects in video scenes,” International Conference on Emerging Trends in Electrical and Computer Technology (ICETECT), 2011, p. 805-810.

[11] Z.Yang, Y. Zhu and Y. Pu, “Parallel image processing based on CUDA,” International Conference on Computer Science and Software Engineering, 2008, p.198-201.

[12] A. Deepa and T. Sasipraba, “Age estimation in images using histogram equalization,” 8. International Conference on Advanced Computing (ICoAC), 2017, p.186-190.

[13] G. Rakesh and T. S. Reddy, “YCoCg color image edge detection,” International Journal of Engineering Research and Applications (IJERA), Vol. 2, No. 2, pp. 152-156.

[14] W. Miled and B. Pesquet-Popescu, “The Use of Color Information in Stereo Vision Processing,”. In: High-Quality Visual Experience. Signals and Communication Technology. Springer, 2010.

[15] Y. Said, T. Saidani, and M. Atri, “High-level design for image processing on FPGA using Xilinx AccelDSP,” World Congress on Computer Applications and Information Systems (WCCAIS), 2014.

[16] R. R. Bulyaculov, K. P. Schogoleva, I. N. Yakovlev, and R. A. Roskostov, “Modelling and analysis of the median filter algorithm of suppression of impulse noise,” IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus), 2017, p.649-654.