Image Processing Application on Graphics processors

Chouchene Marwa *ch.marwa.84@gmail.com*
Laboratory of Electronics and Microelectronics (EμE)
Faculty of Sciences Monastir
Monastir, 5000, Tunisia

Bahri Haythem *bahri.haythem@hotmail.com*
Laboratory of Electronics and Microelectronics (EμE)
Faculty of Sciences Monastir
Monastir, 5000, Tunisia

Sayadi Fatma Ezahra *sayadi_fatma@yahoo.fr*
Laboratory of Electronics and Microelectronics (EμE)
Faculty of Sciences Monastir
Monastir, 5000, Tunisia

Atri Mohamed *mohamed.atri@fsm.rnu.tn*
Laboratory of Electronics and Microelectronics (EμE)
Faculty of Sciences Monastir
Monastir, 5000, Tunisia

Abstract

In this work, we introduce real time image processing techniques using modern programmable Graphic Processing Units GPU. GPU are SIMD (Single Instruction, Multiple Data) device that is inherently data-parallel. By utilizing NVIDIA new GPU programming framework, “Compute Unified Device Architecture” CUDA as a computational resource, we realize significant acceleration in image processing algorithm computations. We show that a range of computer vision algorithms map readily to CUDA with significant performance gains. Specifically, we demonstrate the efficiency of our approach by a parallelization and optimization of image processing, Morphology applications and image integral.

Keywords: Image Processing, GPU, CUDA.

1. INTRODUCTION

The graphics processing unit (GPU) has become an integral part of today’s mainstream computing systems. The modern GPU is not only a powerful graphics engine but also a highly parallel programmable processor featuring peak arithmetic and memory bandwidth that substantially outpaces its CPU counterpart.

The speed of GPU increases the programming capacity so the resolution and processing complex problems. This effort in computing has positioned the GPU as an alternative to traditional microprocessors in high-performance computing system.

CUDA (Compute Unified Device Architecture) is a general architecture for parallel computing introduced by NVidia in November 2007[1]. It includes a new programming model, new architecture and an different set instruction.

In other words, generally the image processing algorithms treat all pixels in any way. But with new technology it is possible to treat all pixels in parallel. For this reason, we can use the

graphics cards because they are capable of performing a large number of times the same function in parallel. In this paper, we will perform some image processing algorithms on the GPU. The paper is organized as follows: In the first part we present a GPU overview. Subsequently a second part deals with the image processing on the GPU. Finally, we conclude the paper.

2. GPU OVERVIEW

There are about ten years of the birth of a hardware dedicated to graphics processing offloading the CPU, called GPU, Graphics Processing Unit. GPUs are supported by a map annex, taking charge of their image processing and 3D data, and display. With its architecture is designed for a massively parallel processing of data, they should become a co-processor used by any application.

Graphics processors are now used to accelerate graphics and some general applications with high data parallelism (GPGPU). For good performance, the parallel data is processed by a large number of processing units that are integrated within the GPU. These are organized according to a single instruction, multiple data (SIMD) or single program, multiple data (SPMD). There are arithmetic units, loading units and interpolation, units of data pre-processing and finally evaluation units of the basic functions.

Indeed, these architectures have evolved to move towards systems more programmable: first through the shaders, but remains very specific 2D or 3D graphics, and more recently through the kernels total. In addition, it is possible that a cluster of GPUs is both more efficient and less energy than a cluster of CPU [2], which makes GPU clusters particularly attractive in the field of high performance computing.

3. NVIDIA CUDA PROGRAMMING

Traditionally, general-purpose GPU programming was accomplished by using a shader-based framework [3]. This framework has a steep learning curve that requires in-depth knowledge of specific rendering pipelines and graphics programming. Algorithms have to be mapped into vertex transformations or pixel illuminations. Data have to be cast into texture maps and operated on like they are texture data. Because shader-based programming was originally intended for graphics processing, there is little programming support for control over data flow; and, unlike a CPU program, a shader-based program cannot have random memory access for writing data. There are limitations on the number of branches and loops a program can have. All of these limitations hindered the use of the GPU for general-purpose computing. NVIDIA released CUDA, a new GPU programming model, to assist developers in general-purpose computing in 2007 [4]. In the CUDA programming framework, the GPU is viewed as a compute device that is a co-processor to the CPU. The GPU has its own DRAM, referred to as device memory, and execute a very high number of threads in parallel. More precisely, data-parallel portions of an application are executed on the device as kernels which run in parallel on many threads. In order to organize threads running in parallel on the GPU, CUDA organizes them into logical blocks. Each block is mapped onto a multiprocessor in the GPU. All the threads in one block can be synchronized together and communicate with each other. Because there is a limited number of threads that a block can contain, these blocks are further organized into grids allowing for a larger number of threads to run concurrently as illustrated in Figure 1. Threads in different blocks cannot be synchronized, nor can they communicate even if they are in the same grid. All the threads in the same grid run the same GPU code.

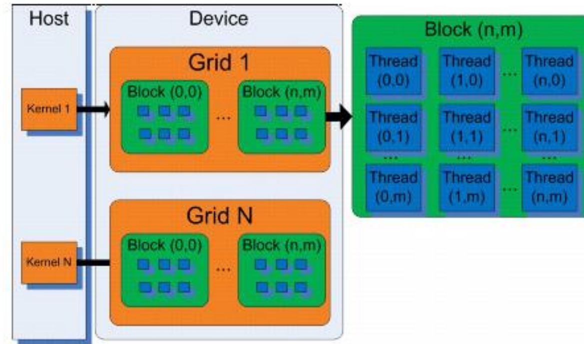

FIGURE 1: Thread and Block Structure of CUDA.

CUDA has several advantages over the shader-based model. Because CUDA is an extension of C, there is no longer a need to understand shader-based graphics APIs. This reduces the learning curve for most of C/C++ programmers. CUDA also supports the use of memory pointers, which enables random memory-read and write-access ability. In addition, the CUDA framework provides a controllable memory hierarchy which allows the program to access the cache (shared memory) between GPU processing cores and GPU global memory. As an example, the architecture of the GeForce 8 Series, the eighth generation of NVIDIA's graphics cards, based on CUDA is shown in Figure 2.

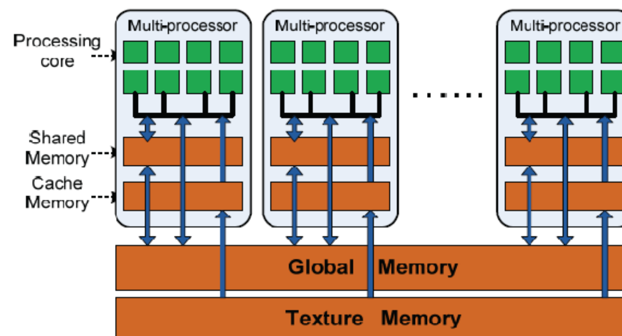

FIGURE 2: GeForce 8 Series GPU Architecture.

The GeForce 8 GPU is a collection of multiprocessors, each of which has 16 SIMD (Single Instruction, Multiple Data) processing cores. The SIMD processor architecture allows each processor in a multiprocessor to run the same instruction on different data, making it ideal for data-parallel computing. Each multiprocessor has a set of 32-bit registers per processors, 16KB of shared memory, 8KB of read-only constant cache, and 8KB of read-only texture cache. As depicted in Figure 2, shared memory and cache memory are on-chip. The global memory and texture memory that can be read from or written to by the CPU are also in the regions of device memory. The global and texture memory spaces are persistent across all the multiprocessors.

4. GPU COMPUTATION IN IMAGE PROCESSING

In recent years, manufacturers of graphics cards highlight the capacity of their GPU to do anything but the game use include implementation we have the image processing applications.

In this part we have implemented many application of image processing using GPU.

4.1 Transformation RGB to Gray

Our application is to display an image, tests are performed on an RGB color image, and we will load this image and send it directly as a texture in memory and then process it for image grayscale first time on CPU and GPU on the second time.

In the remainder of our testing of graphics processors, we ran our program by changing the used image size. We measured the time taken to carry out this treatment using the CPU and GPU, the results are as follows:

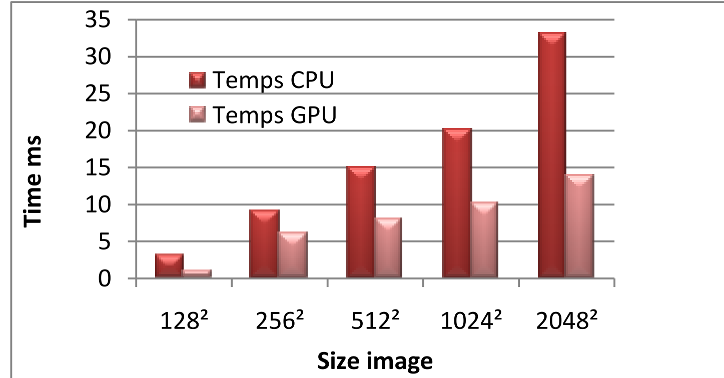

FIGURE 3: Image Processing of Various Sizes.

It should be noted that the execution time on GPU still lower than on the CPU. By increasing the size of the image The GPU time becomes slower and it is due to the data transfer time.

4.2 Morphology Applications

In the following, we will present results of applying morphological (Sobel, Erode) on different images, performed using library GPUCV, and compared with that of the OpenCV library.

We compared the execution time of native CPU operators with their GPU counterpart. GPUCV is up to times faster than native CPU, as shown in figure 4.

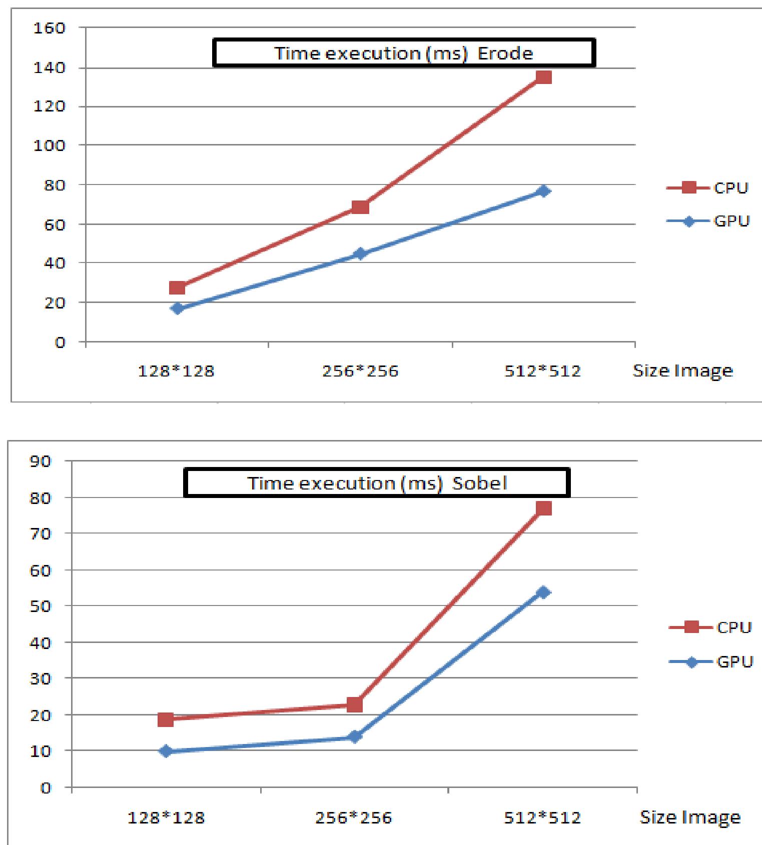

FIGURE 4: Sobel and Erode Transformation of Various Sizes.

4.3 Calculation For Integral Image

The integral image was proposed by Paul Viola and Michael Jones in 2001 and has been used in the real-time object detection [5]. The integral image used is constructed from an original image grayscale.

In [6], the integral image has been extended; it is then used to compute the Haar-like features, and characteristics of centersurround. In the SURF algorithm [7] [8], it accelerates the calculation first and second order of Gaussian derivative that uses the integral image.

In the algorithm CenSurE [9], and its improved version SUSur [10], it uses two levels of filter to approximate the Laplace operator using the integral image.

We present our algorithm to calculate the integral image. The implementation steps are shown in figure 5:

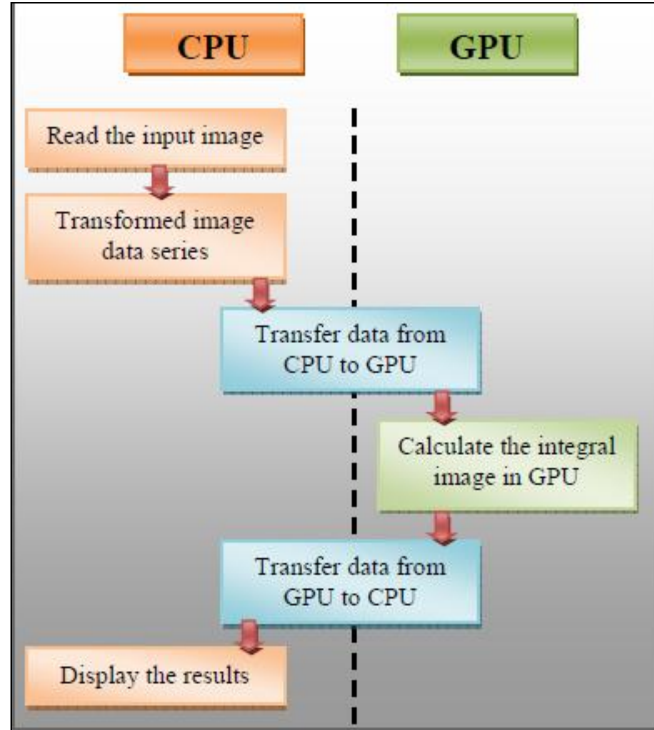

FIGURE 5: Algorithm as Implemented on Hardware.

In order to evaluate our parallel integral image implementation, we executed our algorithm for different image sizes in both CPU and GPU (figure 6).

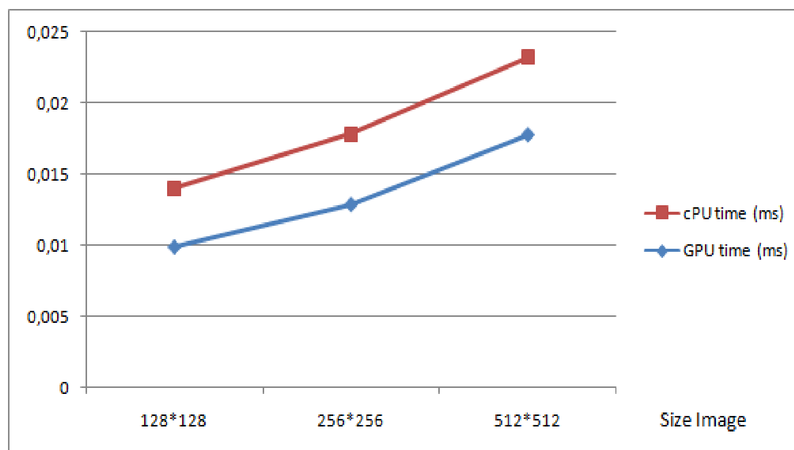

FIGURE 6: Integral Image For Various Sizes.

As can be seen from the table 1, compared to the corresponding CPU-based serial algorithm, our algorithm has a relatively reduction in time-consuming. With the increasing of the image size used in the experiment, speedup increases. However, due to the memory limitations in the video chip, the processed image size at one time cannot be increased unlimitedly.

5. CONCLUSIONS

In this work, we focused on the use of generic graphics cards (GPU) as a parallel computing machine, by explaining their structure and characteristics of their programming. Their increasing use in this context was exposed through a wide range of diverse application areas.

With the GPU, we responded to the needs generated by different computational applications. First, we have accelerated the simulation of a simple image processing. In a second step, we proposed an adaptation GPU, a morphological algorithm (Sobel, Erode). We got time gains evidence from a CPU implementation.

Finally, a parallel integral image algorithm, is presented and implemented on GPU, and compared with the sequential implementations based on CPU. Performance results indicate that significant speedup can be achieved.

6. REFERENCES

[1] NVIDIA Corporation, "NVIDIA CUDA Compute Unified Device Architecture Programming Guide", Version 3, 2013.

[2] L. Abbas-Turki, S. Vialle, B. Lapeyre, and P. Mercier. "High Dimensional Pricing of Exotic European Contracts on a GPU Cluster, and Comparison to a CPU Cluster". In Second InternationalWorkshop on Parallel and Distributed Computing in Finance, May 2009.

[3] Allard, J. and Raffin, B., "A shader-based parallel rendering framework. in Visualization", 2005,VIS 05. IEEE, pp 127-134.

[4] NVIDIA, CUDA Programming Guide Version 1.1. 2007, NVIDIA Corporation: Santa Clara, California.

[5] P Viola, M Jones, "Rapid object detection using a boosted cascade of simple features", Proceedings of IEEE Conference on Computer Vision and Pattern Recognition.2001.

[6] R Lienhart, A Kuranov, V Pisarevsky, "Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection", Pattern Recognition. 2003, vol. 2781, pp. 297-304.

[7] H Bay, A Ess, T Tuytelaars, L Van Gool, "Speededup robust features (SURF) ", International Journal on Computer Vision and Image Understanding, 2008, vol. 110, no. 3, pp. 346–359.

[8] H Bay, T Tuytelaars, L Van Gool, "SURF: Speeded up robust features", Proceedings of the European Conference on Computer Vision, 2006, Springer LNCS volume 3951, part 1, pp 404–417.

[9] M Agrawal, K Konolige, M R Blas "Censure: Center surround extremas for realtime feature detection and matching", 2008, ECCV (4), volume 5305 of Lecture Notes in Computer Science, Springer. pp 102– 115.

[10] M Ebrahimi, W W Mayol-Cuevas, "SUSurE: Speeded Up Surround Extrema feature detector and descriptor for realtime applications", IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, 2009, CVPRW, pp.9-14.