Using Graphic Processors for Computer Vision Applications

Marwa Chouchene, Fatma Ezahra Sayadi, Mohamed Atri, Rached Tourki
Laboratory of Electronics and Microelectronics (EμE)
Faculty of Sciences Monastir
Monastir, Tunisia
ch.marwa.84@gmail.com, sayadi_fatma@yahoo.fr, {mohamed.atri, rached.tourki}@fsm.rnu.tn

ABSTRACT: *All modern PCs are equipped with an extraordinarily powerful graphic card based on a GPU "Graphic Process Unit". A component that must be present to be able to animate the 3D games. But, nowadays we use this component to relieve the CPU and thus by asking the GPU to perform heavy parallel computation and quite important applications such as image processing and video encoding.*

This use of GPU has opened a new avenue for improving the performance of multimedia applications. This is the idea behind CUDA: a library of C programming language written by NVIDIA to exploit the capabilities of calculating GeForce through their drivers. While traditional parallel programming libraries will share the work on these CPU on a machine or even a cluster of machines, with planning software, CUDA intends to exploit the extraordinary resources of the GPU graphics processing.

The aim of this work is the use of graphics processors to perform general computation usually performed by the central processor CPU.

Keywords: GPU, CUDA, OpenCL, Image Processing, Video Processing

Received: 29 November 2012, Revised 4 January 2013, Accepted 8 January 2013

1. Introduction

The GPU (graphics processing unit), is a microprocessor present on the graphics card in a computer or video game, taking charge of their image processing and 3D data, as well as Display.

Using multiple programming language such CUDA and OpenGL, we can program the GPU for image and video processing.

Our work is divided into three parts: In the first part we present a GPU overview.

Subsequently a second part deals with the image processing on the GPU. Processing of video graphics card is presented later. Finally, we conclude the paper.

2. Graphics Processors

A GPU is a programmable logic chip that performs parallel operations on graphics data. Typically found on a plug-in graphics card (display adapter), it is used to encode and render 2D and 3D graphics as well as process video. The more sophisticated and

faster the GPUs, combined with the graphics card's inherent architecture, the more realistically games and movies are displayed.

Specifically, the GPU is adapted to solve a multi-algorithms parallel computing where the data should be organized in tables and each thread will access these specific elements without interacting with the data of other threads.

This type of calculation is called a vector calculation. As against the power of a modern CPU has its ability to execute sequential instructions very quickly using pipelining, managing controls very effectively and even by running one or more different processes simultaneously.

The main reason for the evolution of GPU from the CPU is the specialization in highly parallel computing. Because of its parallel architecture, the GPU is able to solve complex calculations much faster; resulting in a same operation performed in parallel on multiple data (Figure 1).

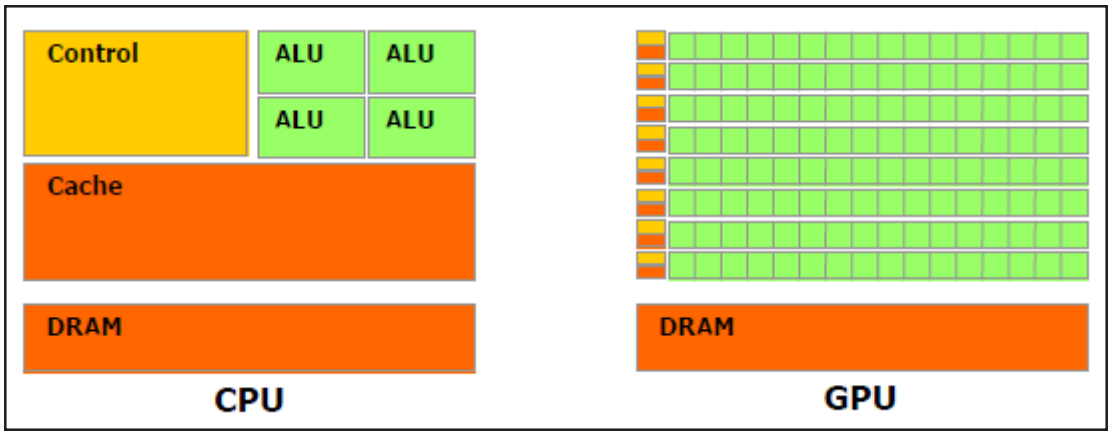

Figure 1. GPU allows more transistors (represented by ALU) to calculate the data that the CPU [1]

2.1 Programming a GPU

The goal of GPU is to generate the high-quality images from three-dimensional models. Their processing pipeline includes the following steps:

- World to screen projection;
- Lighting computation;
- Primitives assembly;
- Rasterization: transforming geometric data into image fragments;
- Color computing: fragments are turned into pixel values. [2]

Two steps in the processing pipeline are programmable:

- The world to screen projection and lighting computation, which are handled by the vertex processor or vertex shader;
- The color computation, which is handled by the fragment processor or fragment shader.

Both shaders have limitations.

- They can only access and modify their own data and have a fixed format output.
- Parameters are passed to shaders in read-only mode.
- Their number of assembly instruction is limited to 1024 or 65536.

The languages available on the GPU are divided into two categories: Shading Languages: whose instructions are primarily intended for graphic calculations and GPGPU languages whose objective is the generic calculation.

2.1.1 Shading Languages

These languages have in common to provide the developer sets of instructions specific to the generation of images, they are

oriented material. They allow you to write shaders. They will be performed by specific hardware devices, Vertex Unit and Fragment Unit, both types of units are unified in the latest generation of cards.

Cg, HLSL, GLSL, these three languages, respectively proposed by NVidia, Microsoft and 3DLabs were developed jointly and are very similar. These are the most commonly used shading languages.

2.1.2 GPGPU languages

All these new languages have a common denominator to be of the highest level shading languages, providing more opportunities and ideas completely separating graphics: texture, shader, vertex or fragment are no more sense.

GPGPU languages the most important are [3]:

• CUDA Compute Unified Device Architecture is the latest NVidia languages. [4]

• OpenCL (Open Computing Language) was announced by Apple in the Compute Working Group

• BrookGPU GPU is an implementation of language Brook, both developed by the Stanford University Graphics Lab. Scout is a language for GPGPU analysis and scientific visualization.

• Accelerator, designed to simplify GPGPU programming in a model of parallel computing accessible only through other languages

• CGIS, developed by the University of Saarlandes, is another parallel language, similar to Brook and Accelerator, also manipulating objects of type stream.

2.2 Previous work

In this section we describe the applied work on GPU:

The OpenVidia library [5] developed by Fung et al. offers a collection of fragment shaders dedicated to computer vision and image processing and a framework to easily apply them on an image or a video.

In [6], Moreland and Angel used a fragment shader to compute a fast Fourier transform on GPU. The fast Fourier transform on GPU is four times faster than on CPU.

Strzodka and Telea presented a generalized bi-dimensional distance transform in [7]

Performances are impressive and this method is up to 34 times depending on the complexity of the image.

Strzodka and Galbe [8] introduced a motion estimation method based on eigen vectors analysis in a spatio-temporal tensor. The sequence is transmitted frame by frame in video memory and eigen vectors, tensor and visualization are computed with fragment shaders. This speeds up the process up to 2.8 times with reference to a Pentium 4 processor.

3. Image processing on GPU

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 a simple application of image processing using GPU shading language "*OpenCL*."
For the implementation, we used the OpenCV library and the GPUCV library.

GPUCV is a library of vision running the calculations on the GPU. It is distinguished from others by its compatibility with OpenCV (instructions and data structures are identical) and ease of access, including for a user who had never handled graphical programming. The goal is to allow easy porting of applications written with OpenCV [9] [10].

Our application is to display an image, tests are performed on an RGB color image of size 256×256 pixels, 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.

We measured the time taken to carry out this treatment using the CPU and GPU. The execution result is shown below, image1 is the input image and image 2 is the image after treatment.

The measurement of time running on the CPU and GPU is illiustrated in Figure 3.

Figure 2. Result of image processing on GPU

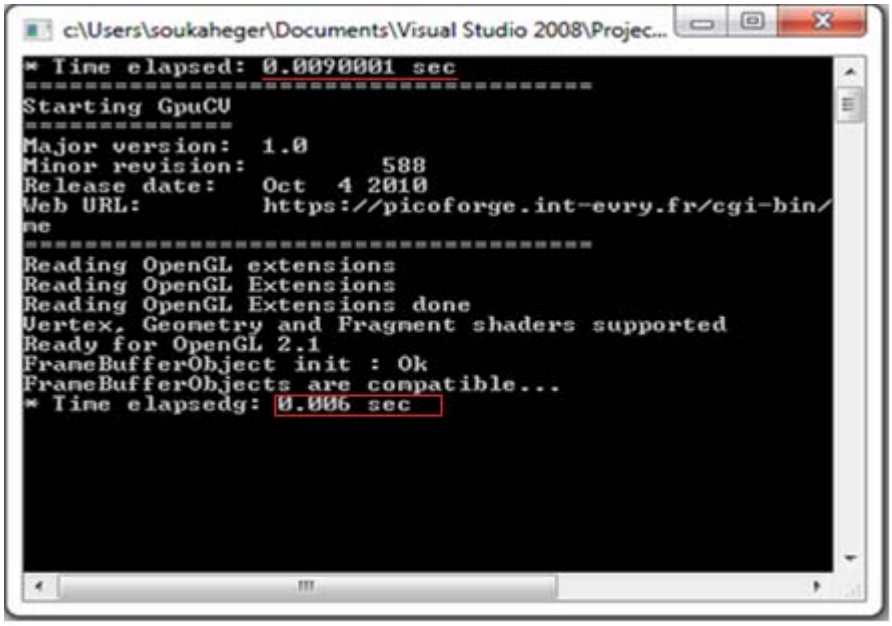

Figure 3. Result of execution time on GPU and CPU

As already illustrated the execution time on CPU is about 0,009 seconds while on the GPU is about 0,006 seconds. This result proves once again that using the GPU can perform images processing faster than a CPU.

In the remainder of our testing of graphics processors, we ran our program by changing the used image size, the results are as follows:

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. Processing video IN graphics processor

In this part we will run the code of the encoder H.264 on GPUs with CUDA language, we will make an assessment of performance in terms of execution time.

The H.263 encoder is a recommendation for video coding standard developed by ITU-TQ.6/SG16.

Initially H.263 was developed for the transmission of video lines at very low speeds, for video telephony applications via the public switched telephone network type H.324. It was then incorporated into protocols for videoconferencing over IP H.323. The same codec has been adopted by the standard H.324M that allows for video calls on circuit-mode third generation mobile networks.

At the beginning of a video communication between two devices equipped with this codec, they change their characteristics through the H.245 protocol and choose modes of H.263 they will use when communicating.

Now we run our program on an Nvidia graphics processor, it already provides an H.264 encoder developed under CUDA

We have a video file as input format "*YUV*" output video frames are encoded in H.264 file.

4.1 Performance Evaluation of encoding between CPU and GPU

Code profiling is an essential tool to optimize a relevant code. Nvidia provides its own profiling tool "*Visual Profiler Nvidia Compute*".

The tool "*Compute Visual Profiler*" allows us to have lots of information, according to the method, it gives us the execution time of each method on the GPU and the execution time of CPU (which is the amount of time the CPU load and GPU to run method)

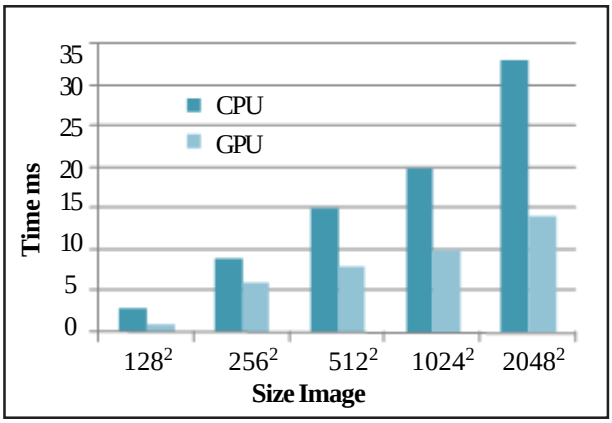

Figure 4. Image processing of various sizes

	Method	Time CPU (µs)	Time GPU (µs)
1	memcpyHtoDasync	216.576	232.359
2	DownSample2To1Kernel	235.488	1883.2
3	IntraPredModeFullSearch 4 × 4	770.88	1181.43
4	IntraPredMode 4 × 4Reduct	843.52	1183.35
5	IntraPredModeFullSearch16 ×16	634.048	1011.28
6	IntraPredModeFullSearchChroma	464.768	876.928
7	memcpyDtoHasync	44.128	47.208
8	memcpyDtoHasync	21.056	21.826
9	memcpyDtoHasync	19.296	19.681
10	memcpyHtoDasync	206.272	211.276
11	memcpyHtoDasync	200.288	210.297
12	memcpyHtoDasync	198.784	203.788

Table 1. Execution time for the encoder functions

H.264 encoder to our code we have a total of 12 functions, the following table presents the results obtained.

The use of graphics cards is always a bonus with the CPU, which is why we have:

CPU time = time GPU + time data transfer

The tool gives us the ability to display also the histogram; we classify methods according to the order with which they were executed according to time GPU

4.2 Comparison of GPU and CPU

After compiling our application (H.246 encoder) respectively on a CPU and a GPU, the results obtained are presented in Table:

	Encoding time (*ms*)
CPU	66342
GPU	45.72

Table 2. Execution time of the H.246 encoder in CPU and GPU

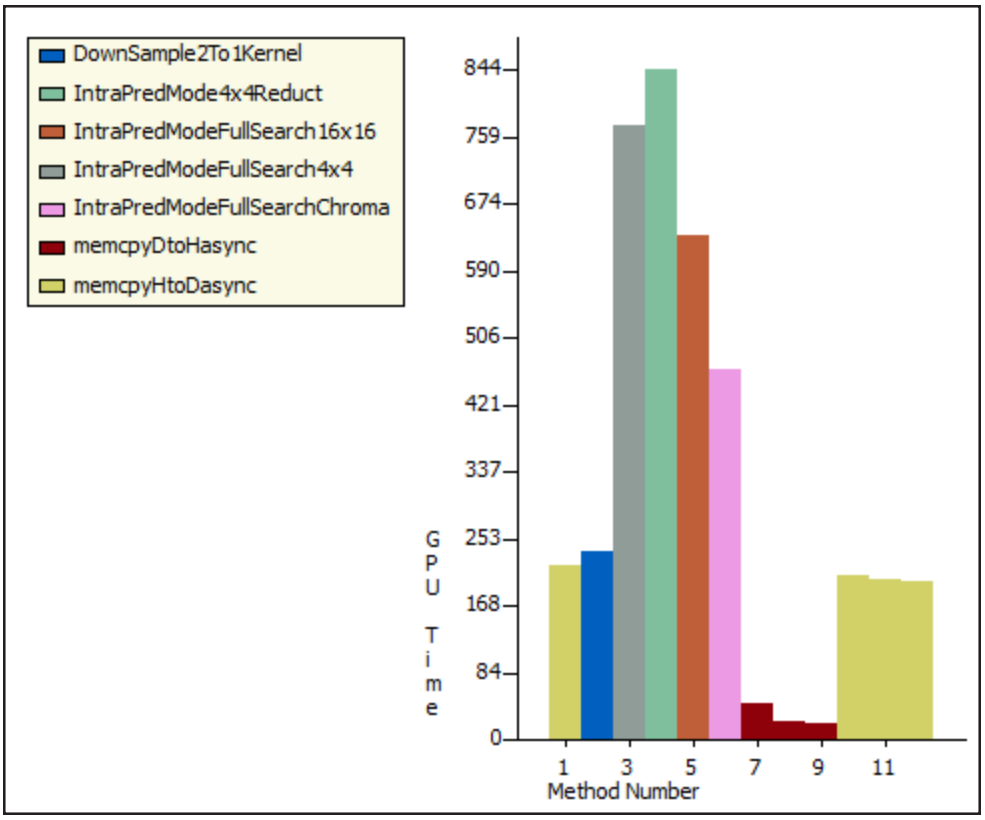

Figure 5. Histogram of execution time of functions

The table shows that the performance of H.246 video encoder on the GPU is much faster compared to the CPU. A good reason for this difference is that incomparable GPUs offer a much higher memory bandwidth than the CPU and engage in equal energy consumption, higher performance computing.

5. Conclusion

In this work we carried out a treatment on a color image output for a gray scale image, we measured the execution time for CPU and GPU, we also measured the time for different image size.

In the second part we executed the program of the H.264 encoder on the CPU and on the GPU. Using the CUDA Visual profiler, we compared the performance in terms of execution time for calculations and for communication between CPU and GPU.

Finally, the computation on the GPU is a growing field that should operate properly in order to increase the efficiency of algorithms especially in the field of bioinformatics.

.

References

[1] CUDA Programming Guide NVIDIA.

[2] Jean-Philippe Farrugia, Patrick Horain, Erwan Guehenneux, Yannick Alusse. (2006). GPUCV: a framework for image processing acceleration with graphics processors, ICM.

[3] Bertrand DANOS. (2009). Présentation d'un GPU, Artiflo Inside October.

[4] NVIDIA. (2006). Cuda (compute unified device architecture). http://www.nvidia.com/object/cuda home.html.

[5] James Fung, Steve Mann, and Chris Aimone. (2005). Openvidia: Parallel gpu computer vision, *In*: Proceedings of the ACM Multimedia, November.

[6] Moreland, K., Angel, E. The FFT on a GPU, in SIGGRAPH/Eurographics Workshop on Graphics Hardware. Eurographics Association.

[7] Strzodka, R., Telea, A. (2004). Generalized Distance Transforms and skeletons in graphics hardware, *In*: Proceedings of EG/IEEE TCVG Symposium on Visualization (VisSym).

[8] Strzodka, R., Garbe, C. (2004). Real-time motion estimation and visualization on graphics cards, *In*: Proceedings IEEE Visualization

[9] Jean-Philippe Farrugia, Patrick Horai. Utilisation de la bibliothèque gpuCV.

[10] Intel. Opencv: Open source computer vision library. http://opencvlibrary.sourceforge.net/.