I.J. Image, Graphics and Signal Processing, **2018, 8, 1-8**
Published Online August 2018 in MECS (http://www.mecs-press.org/)
DOI: 10.5815/ijigsp.2018.08.01

Efficient 2D Convolution Filters Implementations on Graphics Processing Unit Using NVIDIA CUDA

Mouna Afif, Yahia Said, Mohamed Atri
Laboratory of Electronics and Microelectronics (EμE), Faculty of Sciences of Monastir
University of Monastir, 5000, TUNISIA
Email: mouna.afif@outlook.fr

Received: 07 June 2018; Accepted: 03 July 2018; Published: 08 August 2018

***Abstract*—**Convolution algorithms present a key component and a significant step in image processing field. Despite their high arithmetic complexity, these algorithms are widely used because of their great importance for extracting image properties and features. Convolution algorithms require significant computing time, for that we propose a GPU acceleration of these algorithms by using the programming language CUDA presented by NVIDIA. Since these algorithms consume a lot of computing power, we understand the impact of the implementation of this type of algorithm on the acceleration of processing. GPU implementation present a suitable path to achieve better results than other implementation , for that optimizing time consuming time consuming of applications became an increasingly important task in many research areas. The goal of this work is to try to boost convolution algorithms execution time by adopting GPU implementations to accelerate treatments and to achieve real time constraints.

***Index Terms*—**Convolution algorithms, Sobel filter, Gaussian Blur, CUDA, GPU, CPU.

I. Introduction

Convolution presents a crucial step in many computer vision algorithms. Image convolution is widely used in many fields of research as edge detection [1], image classification [2], image recognition [3], object detection [4], etc.

The convolution consists on applying a convolution mask to the input image pixels along the x and y directions. In other words, the convolution consists of performing a parallel and independent calculation of the pixels of the images, thus validating the relevance of the implementation of the convolution algorithms on parallel architectures. Convolution algorithms present a common process and a primordial step in image processing field, especially used for object detection and recognition, image segmentation and features extraction. This type of algorithms require high level of time consuming because of repeated calculation of the convolution filter application for each pixel of the image. Since these algorithms require a lot of computing time, we understand the impact of the implementation of these types of algorithms on the acceleration of processing. Applying a kernel filter to a given image is an independent operation between an image pixels and its neighbor pixel, for that we can calculate image convolution in a parallel way by adopting GPU computing. With the great necessity of the acceleration of the processing of image processing algorithms. Parallelization of algorithms will be the best choice as well as the best method to achieve more efficient and faster results. For this fact, we will need to use architectures that are more modern and more adapted to our needs. Exploring parallelism by using GPUs implementations is a common strategy adopted by researchers to accelerate process.

GPUs present a very attractive way to harness all its resources provided to perform parallel algorithms and to improve much better results than others types of implementations. The high number of cores present in GPU architecture allows thousands of threads to be running simultaneously, yielding to exploit efficiently GPU resources. Due to its attractive cost- performance ratio, GPU has become a competing platform and it is increasingly used to gain in computing resources as well as algorithms runtime. Developers have exploited on massively parallel architecture of GPU in order to reduce computational resources and time consuming of expensive algorithms. To obtain optimal implementations of applications on GPU in order not only require rendering sequential algorithm into parallel one, but require to pay attention about data transfer between CPU and GPU, also another important thing to understand is the bottlenecks and tradeoffs caused by memory latency.

II. Related Works

Image convolution is one of the important and crucial step in image processing algorithms. Convolution as a basic concept has been used in many fields of research such as: telecommunication, electrical engineering, acoustics, optics image segmentation and tracking and

computer graphics. Image convolution is a parallelizable process, where each input image pixel is affected by a given convolution mask or kernel. Convolution process is an independent task between image pixel for that, exploring parallelism is the better strategy adopted by developers for accelerating convolution process. Most of convolution filters algorithms consist of similar computations for each input image pixels. This fact means that this category of algorithms is suitable for implementations on parallel architectures. This type of algorithms is computationally intensive, which impose significant power computation, especially when we need to satisfy real time constraints. Due to their big need to parallelism, convolution algorithms are better implemented and performed on parallel architecture such as FPGAs (Field Programmable Gate Arrays) [5] and GPUs (Graphic Processing Units) [6].

As a convolution filter, among the most used algorithms for filtering and edge detection we find the Sobel filter. This method used especially for edge detection to facilitate features extraction process from images and videos. Several attempts to accelerate sobel filter algorithm on GPU are present in literature. Chouchene and Al [7] have implemented the sobel filter algorithm on GPU by using CUDA NVIDIA technology [8]. They achieve good speed ups comparing to the same treatment performed on CPU. Other attempting of sobel filter algorithm acceleration by using GPU implementation presented in our previous work [9]. We have achieved good accelerations of our algorithm by using CUDA as a programming language. Accelerations achieved comes up to 15 time comparing to the CPU implementation. Our results achieved outperform those obtained by Chouchene and Al.

Another much known convolution technique in image processing field and specifically for image smoothing and removing visual noise is the Gaussian filter. However the implementation of this type of technique require heavy computational resources, for that many applications of image convolution where implemented on parallel architecture such as GPUs and FPGAs. Cabello and Al [10] have adopted a parallel implementation of the Gaussian filter based on FPGA. During their algorithm implementation authors have used many different kernel sizes where set from [3x3], [5x5], [7x7] to [41x 41]. Buzkurt and Al [11] propose a Gaussian filter implementation using the GPU programming tool CUDA. In their study, the authors apply Gaussian blur technique for different image resolution to test their algorithm efficiency. They compare also their algorithm's GPU implementation performances with traditional CPU implementation. Experimental results implementations showed the big efficiency of the algorithm implementation on GPU comparing to the CPU implementation.

With the increasing development of image processing algorithms, and due convolution algorithms minimal complexity computation, developers have to benefit from the increasing GPUs capacities. As shown in [12] Perrot and al attempt to benefit from the GPU NVIDIA by implementing their 2D convolution filter PCRF (Parallel Register- only Convolution Filter) on an NVIDIA K40, this work have shown the efficiency of the convolution filter implementation by their well use of the GPU resources.

This paper will be organized as follows:

In section II, we will give an overview of our proposed methods: the Sobel filter and the Gaussian Blur. Section III is devoted to differentiate between CPU and GPU architectures. Experiments and results are detailed and discussed in section IV. Section V ends the paper by some remarks and conclusions.

III. Convolution Algorithms

A convolution presents a scalar product between the convolution mask and the image pixels within a window. Actually, convolution algorithms are parallel operations that require massive parallel computation, this present a very suitable case for a parallel implementation based on CUDA NVIDIA to achieve best results.

A convolution operation requires N x M multiplications, where N and M present the width and the height of the convolution filter (filter kernel). The kernel size of the convolution is a choice, but 3 x 3 kernel size is the most used. These different kernels contain different patterns, for that we obtain different results after image convolution.

Convolution operations present essential tool in image processing field and typically the responsible for the biggest fraction of the algorithm's execution time. Runtime of convolution applications, can be considerably deceased by adopting implementations via parallel processors (GPUs). GPUs provide suitable architecture to perform convolution applications.

Convolution is a technique widely used in image processing field, among these uses: smoothing and edge detection.

A 2D convolution operation applied to an input image using a 3 x 3 convolution mask is illustrated in the following figure.

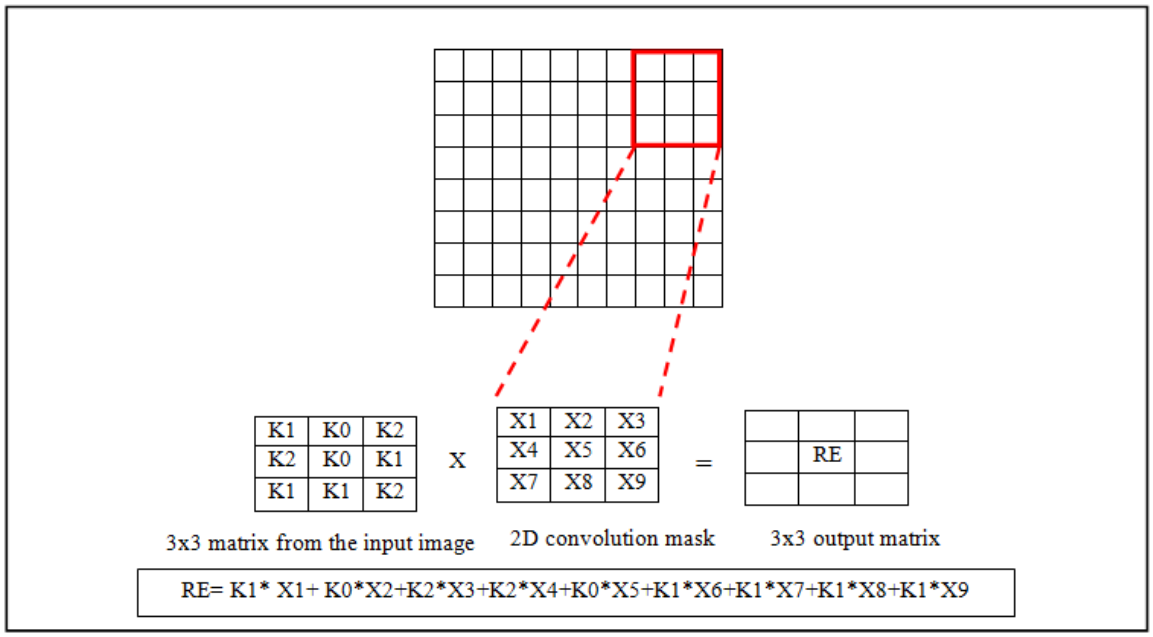


Fig.1. Convolution process

A. *Edge detection: Case of Sobel filter*

In our method, we use the Sobel filter. This is one of the simplest operators that gives correct results. The operator uses convolution matrixes. The 3 × 3 matrix is convoluted with the image to calculate approximations of the horizontal and vertical derivatives. Let I be the input image, Gx and Gy are two images, which at each point contain approximations respectively of the horizontal and vertical derivative of each point. These images are calculated as follows:

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} * I \quad G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I \quad (1)$$

At each point of the image, the approximations of the two images previously calculated combined as follows to obtain an approximation of the gradient norm:

$$G = \sqrt{{G_x}^2 + {G_y}^2} \quad (2)$$

The direction gradient calculated as follow:

$$\theta = arctan\left(\frac{G_y}{G_x}\right) \quad (3)$$

B. *Smoothing: Case of Gaussian Blur*

Smoothing and noise cancellation from image, present a crucial step in many image-processing applications. This type of algorithms have an unfortunate side effect, which could obfuscate and camouflage important image information and results bad effects in the following applications. Gaussian filter present a nonlinear smoothing algorithm that comes at a heavy cost in computational speed, especially when it is used in lager images, since this algorithm perform a great work for each image pixel, however the runtime can be greatly reduced through parallelization as the treatment for each pixel can be done simultaneously.

The convolution performed through multiplexing the input image pixels by a convolution mask (matrix). Gaussian filter is used to attenuate noise that corrupts images. This function is applied in numerous field, it define a smoothing operator, it gives idea about the probability distribution for noise and it is widely used in mathematics. The Gaussian function is as follow:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4)$$

σ is the standard deviation of the distribution (blur factor).
E: Euler number.
x: Horizontal distance to the center pixel.
y: Vertical distance to the center pixel.

The Gaussian function is continuous, so we have to discretize this continuous function to obtain a convolution mask of 5 x 5 pixels as follow:

$$\frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

Fig.2. Gausian Blur kernel

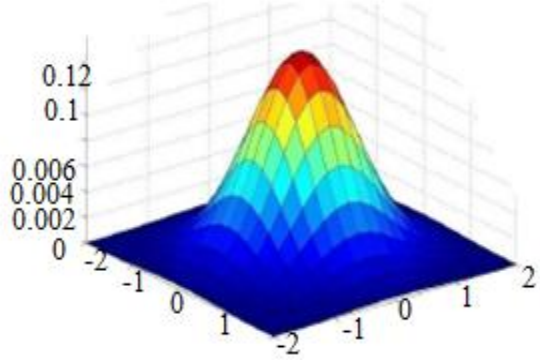


Fig.3. Gaussain distribution for (x,y) = (0,0) and σ = 1

IV. Graphics Processing Unit (GPU) Architecture

Graphic processor units have dramatically developed and evolved during the last decade, they achieve more computational performances comparing to those obtained on CPU. Graphic processors keep getting faster thanks to their architecture, which consist of thousands of cores. Exploring parallelism is the best path to follow for implementation to achieve considerable speed-ups. GPU computing based on heterogeneous programming, which consist on using the graphic processor simultaneously with the CPU (central processor) in a heterogeneous co-processing computing model. GPU is widely use since its appearance. Graphic processor present a very advanced architecture composed by thousands of cores that are suitable to treat thousands of operations concurrently. This allows developers to achieve considerable accelerations in their implementations. During our implementations in the next section, we will use an NVIDIA GPU cards belonging to the tesla [13] range, which are essentially made for scientific computing applications.

Fig.4. Tesla graphic card

A. *Différence between CPU and GPU architectures*

If we put side by side the representations of the architectures of CPU and GPU, we understand why the GPUs are dedicated mainly for the acceleration of the treatments and the massively parallel computations. This is achieved thanks to the large number of ALUs present in these architectures. Whereas GPU architectures use a big number of threads for processing, which is the cause of the considerable accelerations achieved in processing.

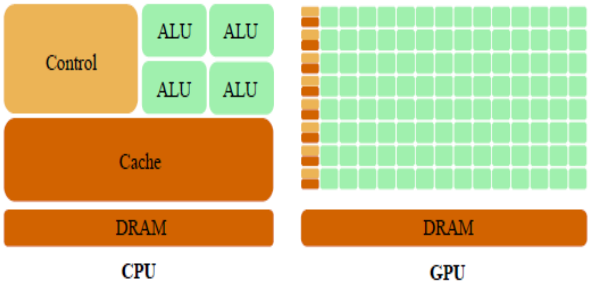


Fig.5. The difference between CPU and GPU architectures[14]

Indeed execution part ALUs present the biggest part in the GPU architecture. GPU deals with complex tasks that require computational parallelization to achieve high speed-up. GPU is very suitable to parallelize tasks, because it consist on a huge number of transistors that are used for sequential tasks as memory latency performance. However, GPU is limited in memory bandwidth and speed, but its capacity to perform thousands of operations concurrently makes it faster than CPU. GPU offer advanced capabilities, where used for 3D game rendering. GPU accelerated computing became increasingly process adopted by developers to optimize computing time of many applications in many fields of research.

B. *CUDA*

In 2006, NVIDIA, the leading manufacturer of high-end GPUs, developed a particular programming language. This programming language called CUDA[16] (Unified Device Architecture). This language is reserved only for NVIDIA brand graphics processors. CUDA is the greatest success story to date in harnessing GPU resources and optimizing parallel calculations. In CUDA technology a program consisting on two parts, the host part deals with the sequential part of the code and the data transfer and the device part, which perform operations in a massively parallel way (parallel portion of the code). In other words the CPU and the GPU parts. Data transfer between CPU and GPU can be done via a bus (PCI Express). Operations performed in a CUDA program called “kernels”, which are executed on device threads under many copy simultaneously, this process allow developers to achieve much better results than other implementations based on other GPU programming languages. Because of the big number of threads [15] present in cuda architecture, CUDA provide the concept of block and grid to manage them efficaciously. Threads performing the same kernels are organized on a grid of threads block. Threads belonging to the same thread block are carried out under the same stream multiprocessor (SM). Also threads within the same threads block can share information and cooperate via shared memory, we note that threads from different threads block can never cooperate.

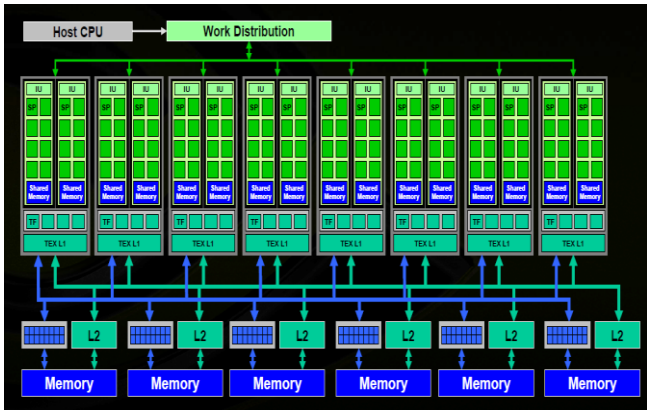


Fig.6. Tesla GPU computing [8]

In CUDA NVIDIA arccchitecture, parallelization is obtained through the execution of thousands of tasks similtaniously on Stream Processor (SP) or CUDA cores. CUDA stream processors can be used to execute integer and floating point instructions. CUDA NVIDIA GPU-based architecturesupports and presents different memoy spaces: global memory, local memory, texture memory, constant memory, shared and register memory.

All blocs-threads can access global memory. Threads within the same bloc can access to the shared memory, while registers presents a local storage for each stream processor.

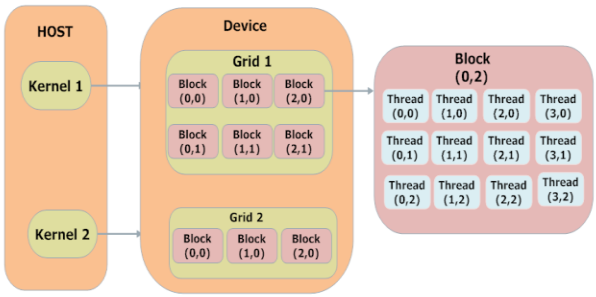


Fig.7. CUDA architecture [16]

V. Experiment and Results

The implementation steps on CUDA NVIDIA adopted for our integral image algorithms are shown in the figure 8 bellow.

GPU consist on numerous [17] SMs (stream multiprocessors) which are composed by stream processors (SPs), which launch a high amount of threads running concurrently. This allows for high level of parallelization, especially [18] SIMD (Single Instruction Multiple Data) i.e (high number of threads running the same task), this technique is very suitable for convolution algorithms as each pixel is treated by the same algorithm. Exploring GPU computing will influence the runtime of developer's algorithms. Based on what we have presented for our algorithms implementations we will use the programming language CUDA developed for NVIDIA GPU to optimize calculation and to accelerate our convolution algorithms treatments.

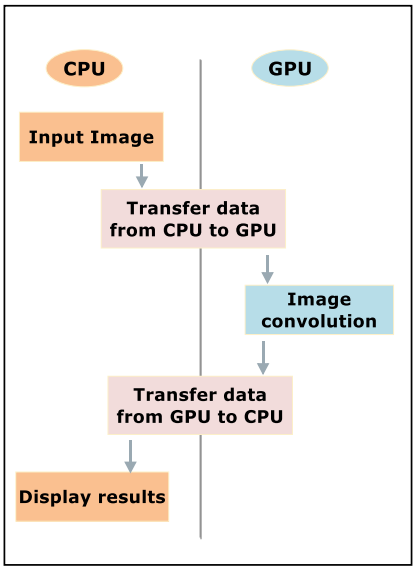


Fig.8. Image Convolution algorithm as implemented on NVIDIA CUDA

To come up with efficient implementation of both Sobel filter and Gaussian Blur algorithm to satisfy real time impositions, we adopt GPU implementation based on the CUDA language. In this section, we propose efficient implementations of our algorithms on the programming language CUDA. We evaluate our algorithms of Sobel filter and Gaussian Blur implemented on CUDA, we compare also our algorithms performances by the same algorithms when they are implemented on CPU and to others previous GPU implementations. Firstly, we have to present our environment tools and materials used for our implementations, and then a discussion of results achieved will be presented.

A. Image processing step on GPU CUDA

1) Image loading: in this step we load the input image from the CPU (host) to the GPU (device) memory, therewith we can apply GPU treatments to the copied image.
2) GPU procesing using CUDA: In this step we have to allocate threads appropriate to the application, we select the number of threads on the GPU architecture. By this way each thread selected perform its processing on the input image pixel. After the thread allocation step we apply the CUDA processing to the input image. CUDA program are also called "kernels", whch are executed using threads selected in the previous step.
3) Results desplaying: After procesing applied to the input image, results are transferred from the GPU memory to the CPU memory. Results presented in our case by using the graphic library OpenCV for (Open Computer Vision). The data transfer between the GPU and the CPU memories present an additional cost for the applicaction.

B. *Hardware environment*

Our experiments were carried out under the GPU programming language CUDA, the version used in our case is 5.0 and for the software implementation we have used visual studio 2010 combined with the graphic library OpenCV of version 2.3.1. Table I summaries the hardware used in this paper.

Table 1. Hardware configuration

Product	**GeForce GT 620**
CUDA Driver Version / Runtime version	5.0
Memory band width	14,4 GB/S
Memory size	1024MB DDR3
Memory bus width	64
OpenGL	4,2
DirectX	11
Bus type	PCI_ Express 2.0 x 16
Memory interface	64 bit DDR3
Maximum digital resolution	2560 x 1600
CUDA cores	96

C. *Sobel filter implementation*

In this section, we tried to evaluate our Sobel filter algorithm on NVIDIA Tesla platform. To ensure the performance obtained by our algorithm, we launched the execution many times and for different image sizes. We compared experiment results obtained on both CPU and GPU. The software execution time are obtained by using the predefined function of time library under C++, while the GPU time are obtained by using the NVIDIA compute visual profiler of CUDA Toolkit 5.0.

In table 2, we present our experiments results of Sobel filter implementation for different image sizes in both CPU and GPU architectures, we note the significant reduction in time consuming by using NVIDIA CUDA GPU. Indeed the execution. Time reduction margin obtained between CPU and GPU varies between 15 and 2047. We note, when we increase the size of the input image we obtain a higher acceleration. The 2048 x 2048 image size used just for algorithm implementation evaluation.

For more details, we present the bar chart of speed-ups obtained of the Sobel filter algorithm implementation using CUDA.

Table 2. Execution time of sobel filter on CPU and GPU

Image size	Sobel filter computing time (ms)		Speed-up
	CPU	GPU	
64 x 64	1.5	0.0962	15
64 x 128	2	0.0963	20
128 x 128	4	0.0965	41
256 x 256	14	0.0966	144
512 x 512	52	0.0969	536
1024 x 1024	199	0.0972	2047
2048 x 2048	786	0.0975	8061

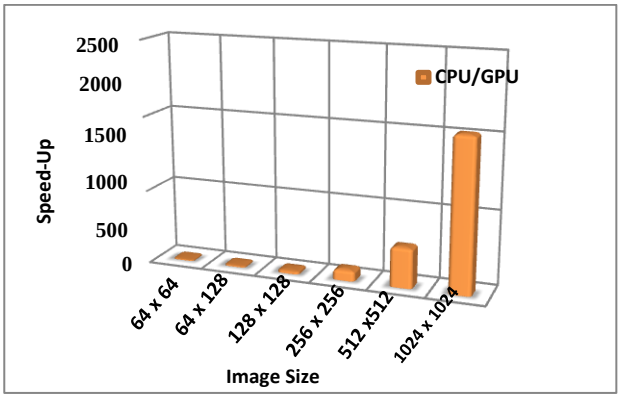


Fig.9. Speed-up factor of sobel filter on CPU and GPU

D. *Gaussian Blur implementation*

In this section, we will be interested to the Gaussian Blur by studying its performances, capacities and its speedup achieved by comparing our experimental implementations results obtained on CPU and GPU architectures. According to results obtained in table 3, we can conclude that GPU implementation boosts the algorithm performances and decrease the time consuming comparing to the same algorithm on CPU. In table3 we provide all experimental results obtained for different image sizes, we summaries results obtained under Tesla GT620 platform, we note a wide gain in execution time by using GPU implementation instead of a CPU implementation. Results showed in table 3 demonstrates the innumerous gain obtained by GPU platform. When the input image size increase, the execution time gain factor increase. Experimental implementations results shows the importance of using GPU platforms for applications implementations. According to Table3, the time-consuming gain margin vary between 2 times for 64 x 64 image size and 245 times for the 1024 x 1024 image size.

Table 3. Execution time of Gaussian Blur on CPU and GPU

Image size	Gaussian Blur computing time (ms)		Speed-up
	CPU	GPU	
64 x 64	0.5	0.293	2
64 x 128	1	0.241	4
128 x 128	2	0.244	8
256 x 256	6	0. 246	24
512 x 512	19	0.247	76
1024 x 1024	61	0.248	245
2048 x 2048	180	0.249	722

We present bellow for more details the bar chart of speedups achieved of Gaussian Blur algorithm implementation.

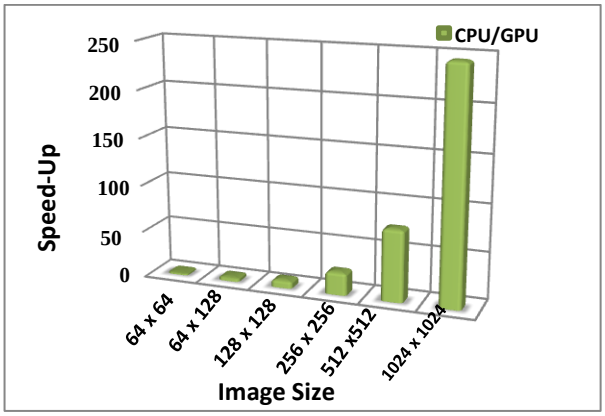


Fig.10. Speed-up factor of Gaussian Blur on CPU and GPU

VI. Conclusion

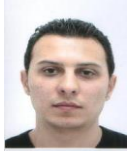
Convolution algorithms, present a basic component in image processing tool and characteristically responsible for the large portion of code requiring high execution time. In this paper, we propose an efficient implementations of two convolution algorithms in our case the Sobel filter and the Gaussian blur on GPU using the application-programming interface CUDA. Results show the high efficiency of our algorithms for all filters algorithms and for all images sizes. We achieve much lower runtime by performing algorithms on GPU NVIDIA CUDA comparing to CPU implementations and to other previous GPU implementations.

References

[1] Rafael C Gonzalez and Richard E Woods. Digital Image Processing. Pearson, 3rd edition, 2007.
[2] Alex Krizhevsky, Ilya Sutskever, and Geo_rey E Hinton. Imagenet Classi_cation with Deep convolutional Neural Networks. In NIPS, pages 1097{1105, 2012.
[3] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. CoRR, abs/1409.1556, 2014.
[4] Ross Girshick, Je_ Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In CVPR, pages 580{587, 2014.
[5] Xilinx. Available from: http://www.xilinx.com, 2006.
[6] NVIDIA. NVIDIA CUDA Programming Guide 2.0. 2008.
[7] M. Chouchene, H. Bahri, F.E. sayadi and M. Atri. “Image Processing Application on Graphics processors”. IEEE Conference on Computer Vision and Pattern Recognition International Journal of Image Processing (IJIP) volume (8): Issue (3), 2014.
[8] S. Jubertie, NVIDIA CUDA Compute Unified Device Architecture, Laboratory of Computer Science of Orleans, 2011.
[9] Afif, M., Said, Y., Bahri, H., & Atri, M. (2016, November). Efficient implementation of sobel filter based on GPUs cards. In Image Processing, Applications and Systems (IPAS), 2016 International(pp. 1-4). IEEE.
[10] Cabello, F., León, J., Iano, Y., & Arthur, R. (2015, September). Implementation of a fixed-point 2D Gaussian Filter for Image Processing based on FPGA. In Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), 2015 (pp. 28-33). IEEE.
[11] Bozkurt, F., Yaganoglu, M., & Günay, F. B. (2015). Effective Gaussian Blurring Process on Graphics Processing Unit with CUDA. International Journal of Machine Learning and Computing, 5(1), 57.
[12] Perrot, G., Domas, S., & Couturier, R. (2016). An optimized GPU-based 2D convolution implementation. Concurrency and Computation: Practice and Experience, 28(16), 4291-4304.
[13] Lindholm, E., Nickolls, J., Oberman, S., & Montrym, J. (2008). NVIDIA Tesla: A unified graphics and computing architecture. IEEE micro, 28(2).
[14] Cuda_c_programming_guide_2.3.
[15] D. Kirk, W. mei Hwu, Chapter 3: CUDA Threading Model, NVIDIA, 2006.
[16] S. Jubertie, NVIDIA CUDA Compute Unified Device Architecture, Laboratory of Computer Science of Orleans, 2011.
[17] Yang, Z., Zhu, Y., & Pu, Y. (2008, December). Parallel image processing based on CUDA. In Computer Science and Software Engineering, 2008 International Conference on (Vol. 3, pp. 198-201). IEEE.
[18] Harish, P., & Narayanan, P. J. (2007, December). Accelerating large graph algorithms on the GPU using CUDA. In HiPC (Vol. 7, pp. 197-208).

Authors’ Profiles

Mouna AFIF Received her master degree in micro and nano electronics from Monastir University in 2016. Currently she is a PHD student at the faculty of sciences of Monastir. She is working on image and video processing implementation on GPUs.



Yahia SAID received the Master„s Degree in Micro-electronics from Faculty of Science of Monastir, Tunisia in 2010. Since 2011, he has been working as a Research Scientist at the Laboratory of Electronics & Micro-electronics, Faculty of Science of Monastir where he

prepares his thesis. His areas of interest include Embedded Processor, Embedded System, Image and Video Processing, and HW/SW Co-design.

Mohamed ATRI received his Ph.D. Degree in Micro-electronics from the University of Monastir, Tunisia in 2001. He has obtained the HDR degree from the University of Monastir in 2011.

He is currently a member of the Laboratory of Electronics & Micro-electronics, Faculty of Science of Monastir. His research includes Circuit and System Design, Image processing, Network Communication, IPs and SoCs.

How to cite this paper: Mouna Afif, Yahia Said, Mohamed Atri, " Efficient 2D Convolution Filters Implementations on Graphics Processing Unit Using NVIDIA CUDA ", International Journal of Image, Graphics and Signal Processing(IJIGSP), Vol.10, No.8, pp. 1-8, 2018.DOI: 10.5815/ijigsp.2018.08.01

