Gaussian Blur through Parallel Computing

Nahla M. Ibrahim, Ahmed Abou ElFarag and Rania Kadry
Department of Computer Engineering, Arab Academy for Science and Technology and Maritime Transport, Alexandria, Egypt

Keywords: CUDA, Parallel Computing, Image Convolution, Gaussian Blur, Google Colaboratory.

Abstract: Two dimensional 2D convolution is one of the most complex calculations and memory intensive algorithms used in image processing. In our paper, we present the 2D convolution algorithm used in the Gaussian blur which is a filter widely used for noise reduction and has high computational requirements. Since, single threaded solutions cannot keep up with the performance and speed needed for image processing techniques. Therefore, parallelizing the image convolution on parallel systems enhances the performance and reduces the processing time. This paper aims to give an overview on the performance enhancement of the parallel systems on image convolution using Gaussian blur algorithm. We compare the speed up of the algorithm on two parallel systems: multi-core central processing unit CPU and graphics processing unit GPU using Google Colaboratory or “colab”.

1 INTRODUCTION

Parallel computer systems popularity is highly increasing for their ability to solve complex problems and to deal with the significant increase in the data sizes and huge data sets (Bozkurt et al, 2015). Image processing has moved from the sequential approach to the parallel programming approach as the image processing applications such as image filtering and convolution consume time, resources and the complexity increases with the image size (Reddy et al, 2017). Convolution is known to be a complex mathematical operation that is highly used in image processing (Novák et al, 2012). The parallel programming approach for the image processing is done through a multi-core central processing unit CPU and graphics processing unit GPU. GPU with the presence of the multithreaded parallel programming capabilities provided by CUDA gained more popularity with image processing applications as it increased the speedup factor by hundreds to thousands compared to the CPU as the CPU speedup factor is limited to the number of available cores (Reddy et al, 2017).

This paper studies the performance of Image convolution using Gaussian filter technique on parallel systems using Google Colaboratory platform and studies the effect of using Google Colaboratory platform. The paper uses two different parallel systems: multi-core central processing unit CPU and graphics processing unit GPU. The paper is organized as follows: Section 1 presents the introduction on the parallel systems: multi-core central processing unit CPU and graphics processing unit GPU, Gaussian Blur filter and related work. Section 2 presents the experiment, the architecture of the platform used, the results and the final section is the conclusion.

2 PRELIMINARY

2.1 CPU and GPU

A Multicore CPU is a single computing component with more than one independent core. OpenMP (Open Multi-Processing) and TBB (Threading Building Blocks) are widely used application programming interfaces (APIs) to make use of multicore CPU efficiently (Polesel et al, 2000). In this study, a general purpose platform using Python pymp tool is used to parallelize the algorithm. On the other hand, GPU is a single instruction and multiple data (SIMD) stream architecture which is suitable for applications where the same instruction is running in parallel on different data elements. In image convolution, image pixels are treated as separate data elements which makes GPU architecture more suitable for parallelizing the application (Polesel et al,

Ibrahim, N., ElFarag, A. and Kadry, R.
Gaussian Blur through Parallel Computing.
DOI: 10.5220/0010513301750179
In *Proceedings of the International Conference on Image Processing and Vision Engineering (IMPROVE 2021)*, pages 175-179
ISBN: 978-989-758-511-1

2000). In this study, we are using CUDA platform with GPU since CUDA is the most popular platform used to increase the GPU utilization. The main difference between CPU and GPU, as shown in Figure 1 (Reddy et al, 2017), is the number of processing units. In CPU it has less processing units with cache and control units while in GPU it has more processing units with its own cache and control units. GPUs contain hundreds of cores which causes higher parallelism compared to CPUs.

2.2 NVIDIA, CUDA Architecture and Threads

The GPU follows the SIMD programming model. The GPU contains hundreds of processing cores, called the Scalar Processors (SPs). Streaming multiprocessor (SM) is a group of eight SPs forming the graphic card. Group of SPs in the same SM execute the same instruction at the same time hence they execute in Single Instruction Multiple Thread (SIMT) fashion (Lad et al, 2012). Compute Unified Device Architecture (CUDA), developed by NVIDIA, is a parallel processing architecture, which with the help of the GPU produced a significant performance improvement. CUDA enabled GPU is widely used in many applications as image and video processing in chemistry and biology, fluid dynamic simulations, computerized tomography (CT), etc (Bozkurt et al, 2015). CUDA is an extension of C language for executing on the GPU, that automatically creates parallelism with no need to change program architecture for making them multithreaded. It also supports memory scatter bringing more flexibilities to GPU (Reddy et al, 2017). The CUDA API allows the execution of the code using a large number of threads, where threads are grouped into blocks and blocks make up a grid. Blocks are serially assigned for execution on each SM (Lad et al, 2012).

2.3 Gaussian Blur Filter

The Gaussian blur, is a convolution technique used as a pre-processing stage of many computer vision algorithms used for smoothing, blurring and eliminating noise in an image (Chauhan, 2018). Gaussian blur is a linear low-pass filter, where the pixel value is calculated using the Gaussian function (Novák et al, 2012). The 2 Dimensional (2D) Gaussian function is the product of two 1 Dimensional (1D) Gaussian functions, defined as shown in equation (1) (Novák et al, 2012):

$$G(x,y) = \frac{1}{2\Pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (1)$$

where (x,y) are coordinates and ‘σ’ is the standard deviation of the Gaussian distribution.

The linear spatial filter mechanism is in the movement of the center of a filter mask from one point to another and the value of each pixel (x, y) is the result of the filter at that point is the sum of the multiplication of the filter coefficients and the corresponding neighbor pixels in the filter mask range (Putra et al, 2017). The outcome of the Gaussian blur function is a bell shaped curve as shown in Figure 1 as the pixel weight depends on the distance metric of the neighboring pixels (Chauhan, 2018).

The filter kernel size is a factor that affects the performance and processing time of the convolution process. In our study, we used odd numbers for the kernel width: 7x7, 13x13, 15x15 and 17x17.

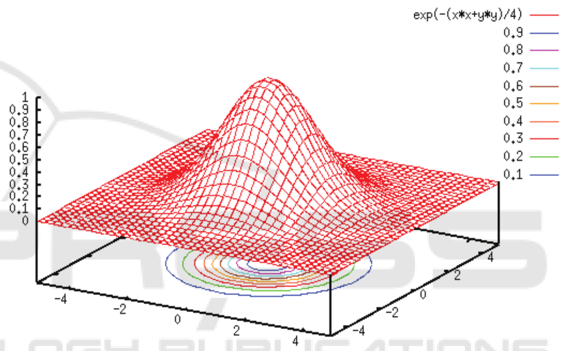

Figure 1: The 2D Gaussian Function.

2.4 Related Work

Optimizing image convolution is one of the important topics in image processing that is being widely explored and developed. The effect of optimizing Gaussian blur by running the filter on CPU multicore systems and its improvement from single CPU had been explored by Novák et al. (2012). Also, exploring the effect of running Gaussian blur filter using CUDA has been explored by Chauhan (2018). In the previous studies, the focus was getting the best performance from multicore CPUs or from GPU compared to the sequential code. Samet et al. (2015) presented a comparison between the speed up of real time applications on CPU and GPU using C++ language and Open Multi-Processing OpenMP. Also, Reddy et al. (2017) presented a comparison between the performance of CPU and GPU was studied for image edge detection algorithm using C language and CUDA on NVIDIA GeForce 970 GTX. In our study we are exploring the performance improvement between two different parallel systems, CPU multicore and GPU using python parallel libraries pymp and CUDA respectively. We are using Intel Xeon CPU and NVIDIA Tesla P100 GPU.

3 EXPERIMENTAL SETUP

In our experiment, we are using Google Colaboratory or "colab" to run our code. Google Colaboratory is a free online cloud-based Jupyter notebook environment that allows us to train PyCUDA and which gives you easy, pythonic access NVIDIA's CUDA parallel computation API. To be able to run our code on Google Colabs, we used the Python programming language. Python pymp library is used for the multiprocessor code. This package brings OpenMP-like functionality to Python. It takes the good qualities of OpenMP such as minimal code changes and high efficiency and combines them with the Python Zen of code clarity and ease-of-use. Python PyCUDA library is used to be able to call CUDA function in our python code and PyCUDA gives you easy, pythonic access to NVIDIA's CUDA parallel computation API.

3.1 Architecture of the Used GPU

NVIDIA Tesla P100 GPU accelerators are one of the advanced data center accelerators, powered by the breakthrough NVIDIA Pascal™ architecture and designed to boost throughput and save money for HPC and hyperscale data centers. The newest addition to this family, Tesla P100 for PCIe enables a single node to replace half a rack of commodity CPU nodes by delivering lightning-fast performance in a broad range of HPC applications (Nvidia Corporation, 2016). CUDA toolkit 10.1 is used.

3.2 Architecture of the Used CPU

Intel Xeon is a high-performance version of Intel desktop processors intended for use in servers and high-end workstations. Xeon family spans multiple generations of microprocessor cores. Two CPUs available with two threads per core and one core per socket (http://www.cpu-world.com/CPUs/Xeon/).

3.3 Experiment Results

Three images are used in our experiment, their particular sizes are of 256 x 256, 1920 x 1200 and 3840 × 2160. We ran our Gaussian filter algorithm on the images using different kernel sizes and calculated the average processing time of 10 runs. First, the color channels of the images are being extracted to red channel, green channel and blue channel. In the second step the convolution of each channel respectively with the Gaussian filter. Using the pycuda library in python code, a CUDA C function had been used to be able to run the code on the GPU. The data is transferred from host to device and after the convolution operation is being brought back to the host. In the end we merge all the channels together to get the output of the blurred image. The processing time calculated is the time taken by the convolution function of the filter on the CPU using two threads and on the GPU as shown in Table 1 and Table 2 respectively. The processing time and speedup are shown in Figure 2 and 3 respectively. Figures 4 and 5 respectively show the original image and blurred image of resolution 256 x 256 using 7 x 7 filter kernel.

From the results of Bozkurt et al. (2015), it is observed that our performance speedup on GPU was higher than the one proposed by Bozkurt et al. (2015) by more than 3 times.

Table 1: CPU time.

Image resolution (pixels)	Kernel size	Processing time (s)
256 x 256	7 x 7	8.2
	13 x 13	34.9
	15 x 15	47.8
	17 x 17	62.3
1920 x 1200	7 x7	429.3
	13 x 13	1358.2
	15 x 15	1882.9
	17 x 17	2294.15
3840 × 2160	7 x 7	1296.4
	13 x 13	5322.2
	15 x 15	7347.02
	17 x 17	9225.1

Table 2: GPU time.

Image resolution (pixels)	Kernel size	Processing time (s)
256 x 256	7 x 7	0.00200
	13 x 13	0.00298
	15 x 15	0.00312
	17 x 17	0.00363
1920 x 1200	7 x 7	0.02726
	13 x 13	0.03582
	15 x 15	0.04410
	17 x 17	0.054377
3840 × 2160	7 x 7	0.06950
	13 x 13	0.11282
	15 x 15	0.14214
	17 x 17	0.17886

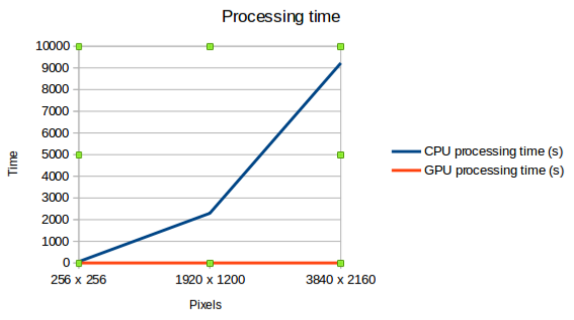

Figure 2: Processing Time.

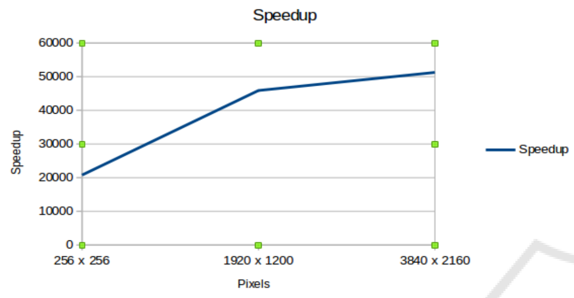

Figure 3: Speedup.

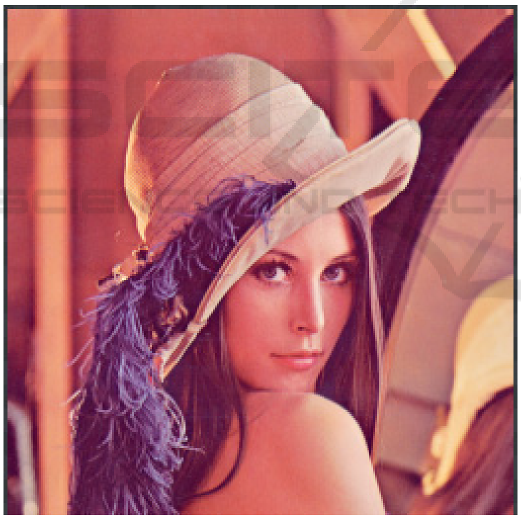

Figure 4: Original Image.

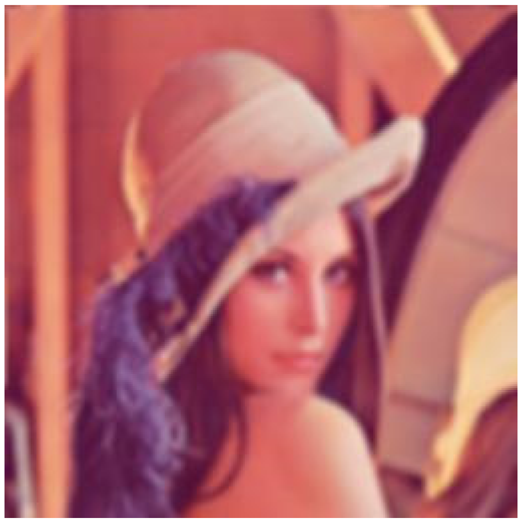

Figure 5: Blurred image using 7 x 7 filter kernel.

4 CONCLUSIONS

Google Colaboratory platform showed great results for using their free access to run our code on the GPUs. Better performance and speedup were gained using CUDA GPU architecture when compared to multicore CPU. The graphs in Figures 2 and 3 show that the processing time for the GPU is almost zero compared to the CPU processing time. Increasing the number of threads for the CPU could give better results but still it will not be able to fill the gap clearly shown between CPU and GPU. GPU is a better candidate for applications having complex operations where the same operation is applied to a big set of data as convolution. Image convolution with Gaussian blur filter is better handled with GPU and gives much better performance.

As future work, to run the same algorithm on different multicore CPU and GPU. Also to investigate the image transfer time from the CPU to GPU and optimize the transfer time of the image for better performance and speedup.

REFERENCES

Andrea Polesel et al, 2000, Image Enhancement via Adaptive Unsharp Masking. IEEE Transactions on Image Processing, VOL. 9, NO. 3

Ritesh Reddy et al, 2017, Digital Image Processing through Parallel Computing in Single-Core and Multi-Core Systems using MATLAB. IEEE International Conference On Recent Trends in Electronics

Information & Communication Technology (RTEICT). India

Jan Novák et al, GPU Computing: Image Convolution. Karlsruhe Institute of Technology.

Shrenik Lad et al, Hybrid Multi-Core Algorithms for Regular Image Filtering Applications. International Institute of Information Technology. Hyderabad, India

Ferhat Bozkurt et al, 2015, Effective Gaussian Blurring Process on Graphics Processing Unit with CUDA. International Journal of Machine Learning and Computing, Vol. 5, No. 1. Singapore, Asia.

Munesh Singh Chauhan, 2018, Optimizing Gaussian Blur Filter using CUDA Parallel Framework. Information Technology Department, College of Applied Sciences. Ibri, Sulatanate of Oman.

B. N. Manjunatha Reddy et al, 2017, Performance Analysis of GPU V/S CPU for Image Processing Applications. International Journal for Research in Applied Science & Engineering Technology (IJRASET). India.

Ridho Dwisyah Putra et al, 2017, A Review of Image Enhancement Methods. International Journal of Applied Engineering Research ISSN 0973-4562 Volume 12, Number 23 (2017), pp. 13596-13603. India.

Nvidia Corporation, 2016, NVIDIA® TESLA® P100 GPU ACCELERATOR. NVIDIA Data sheet.

The CPU World website [http://www.cpu-world.com/CPUs/Xeon/]

R. Samet et al, 2015, Real-Time Image Processing Applications on Multicore CPUs and GPGPU, International Conference on Parallel and Distributed Processing Techniques and Applications. USA.