Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 79 (2016) 516 – 524

Procedia Computer Science

7th International Conference on Communication, Computing and Virtualization 2016

Implementation of Image Enhancement Algorithms and Recursive Ray Tracing using CUDA

Mr. Diptarup Saha[a], Mr. Karan Darji[b], Dr. Narendra Patel[c], Dr. Darshak Thakore[d]

[a,b,c,d]*Birla Vishvakarama Mahavidyalaya, Vallabh Vidyanagar, Anand,Gujarat,India*

Abstract

This paper intends to achieve high performance in terms of time by implementing various time consuming application on NVIDIA Graphics Processing Unit (GPU) by using parallel programming model NVIDIA Compute Unified Device Architecture (CUDA). NVIDIA CUDA provides platform for developing parallel applications on NVIDIA GPUs. So it gives developers a platform to build high-end parallel processing applications. This paper implements various image processing algorithms on both Central Processing Unit (CPU) and GPU. Implemented point-to-point image processing algorithms are brightening filter, darkening filter, negative filter and RGB to Grayscale filter. Along with various convolution algorithms that consider value of its neighboring pixels are also implemented. Implemented convolution algorithms are sobel filter for edge detection, low pass filter and high pass filter. Performance analysis of the implemented image processing algorithms is done on both CPU and GPU. Analysis is made on images of resolution 3000 X 3000. Color-ed images are used for point-to-point pixel processing algorithms. Grayscale images are used for all convolution algorithms. Performance analysis done for point-to-point processing algorithms by varying number of threads per block.

Recursive ray tracing is also implemented on GPU, and found performance gain compare to serial algorithm run on CPU.

Peer-review under responsibility of the Organizing Committee of ICCCV 2016

Keywords:CUDA;Image Processing; NVIDIA GPU; Parallel Programming

1. Introduction

The main objective of image processing is to improve quality of pictorial information for better human interpretation and processing o image data for storage, transmission and representation. Image processing is usually an expensive operation for point processing as well as neighborhood processing for large size images. It becomes more expensive for colored images. In recent years Graphics Processing Units (GPU) have become tools for processing in

Peer-review under responsibility of the Organizing Committee of ICCCV 2016
doi:10.1016/j.procs.2016.03.066

parallel large amount of information. NVIDIA developed the CUDA architecture that groups cores of GPU in a vector which can be programmed to reduce processing time over large amount of data.

Fig. 1 Block diagram representing difference between CPU and GPU[1,2]

Drastic performance boost achieved by CUDA is due to architecture of GPU. It contains hundreds ALUs. Hence the focus is concentrated to computation. Using CUDA, the latest NVIDIA GPUs become accessible for computation like CPUs. Unlike CPUs, however, GPUs have a parallel throughput architecture that emphasizes executing many concurrent threads slowly, rather than executing a single thread very quickly. This approach of solving general-purpose problems on GPUs is known as GPGPU (General Purpose Graphics Processing Unit).

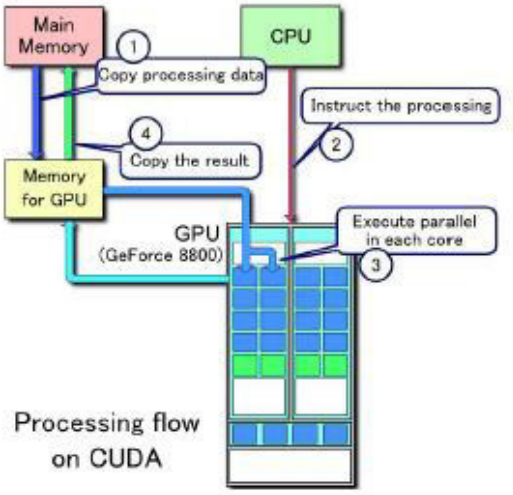

Fig 2 CUDA Process workflow diagram[1]

On start CUDA's compiled code runs like any other application. Its primary execution is happening in CPU. When kernel call is made, application continues execution of non-kernel function on CPU.

In the same time, kernel function does its execution on GPU. This way we get parallel processing between CPU and GPU. This is also known as heterogeneous programming. Memory move between host and device is primary bottleneck in application execution. Execution on both is halted until this operation completes.

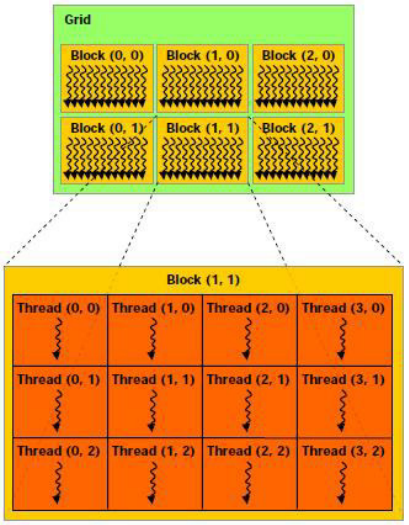

Fig 3 CUDA Grid-Block-Thread Structure[1,2]

As shown in fig 3, Grid contains partitioned coarse grained sub-problems in each block. Further each block contains set of threads which is fine grained. CUDA helps in partitioning the problem into coarse sub-problems that can be solved independently in parallel by blocks of threads[9], and each sub-problem into smaller pieces that can be solved cooperatively in parallel by all threads within the block. Indeed, each block of threads can be scheduled on any of the available multiprocessors within a GPU, in any order, concurrently or sequentially, so that a compiled CUDA program can execute on any number of multiprocessors as illustrated by Figure 1.3 CUDA uses CUDA C as programming language. It is extension of C, with added libraries for CUDA.

2. Image Processing Operations

2.1. RGB to Greyscale

Grayscale is a function that obtain light intensity as a result, instead of colours.

$$G(x, y) = (\text{Red}(x, y) + \text{Green}(x, y) + \text{Blue}(x, y)) / 3; \quad [1]$$

Above equation shows way obtain a grayscale image from coloured image by averaging the three channels i.e. RGB.

2.2Negative Filter

The negative transformation inverts the intensity levels of an image. This is useful in grayscale images when is needed to find some significant characteristics and the predominant color is white (or black) like x-ray plate. Below equation can be used to obtain negative images.

$$G(x,y) = 255 - P(x,y); \quad [2]$$

2.3 Darkening Filter

There are cases when in image light exposure is more, then image appears to be brighter than expected. In such cases darkening filter can be used to reduce the effect of light exposure by reducing the intensity value of pixels.

$$G(x, y) = f(x, y) * b; \; 0<b<1, \text{ b is constant for all pixels} \quad [3]$$

2.4 Brightening Filter

Brightening filter maps the pixel values to higher ones through a function applied to *f(x, y)* or constants values:

$$G(x,y) = f(x,y) * b; \; b>1 \quad [4]$$

2.5 Low Pass Filter

Low pass filter basically removes noise from the image. It removes discontinuous effect if any from the image.
Mask for Low Pass filter:

$$\begin{bmatrix} 0 & 1/8 & 0 \\ 1/8 & 1/4 & 1/8 \\ 0 & 1/8 & 0 \end{bmatrix}$$

2.6 High Pass Filter

High pass filter basically sharpens the image.
Mask for High Pass filter:

$$\begin{bmatrix} 0 & -1/4 & 0 \\ -1/4 & 2 & -1/4 \\ 0 & 1/4 & 0 \end{bmatrix}$$

2.7 Sobel Filter for Edge Detection

Sobel filter is used for edge detection. In vertical Sobel filter mask will be applied vertically.
Mask for Vertical Sobel Filter:

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Mask for Horizontal Sobel filter:

$$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

2.8 Recursive Ray Tracing Algorithm

Recursive Ray Tracing or Global Illumination or indirect illumination is an algorithm that adds more realistic lighting to 3D scenes. RRT takes into account not only the light which comes directly from a light source (direct illumination), but also the light coming from surrounding due to refraction, reflection, shadow etc.

3. Result Analysis

By applying image enhancement filters on image of 3000 X 3000 resolution, execution time of algorithms are measured.

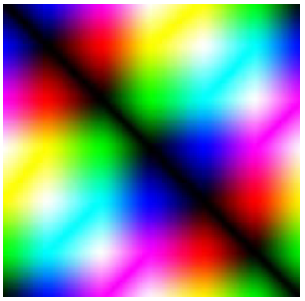

Fig 4 Original PPM Image

Image shown in Fig 4 is the image used for finding experimental result.

3.1 RGB to Greyscale

By implementing RGB to Greyscale filter on CPU for image of resolution 3000 X 3000, we found time taken for execution was 262.00 ms. It was also implemented on GPU with varying number of Threads per Block and its analysis is presented in Table 1

Table 1 RGB to Greyscale

No. of blocks	No. of Threads/block	Time (ms)
27000000	1	11.08
54000	500	21.61
27000	1000	42.86

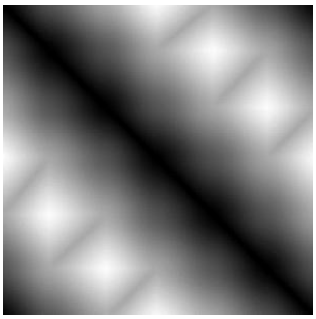

Fig 5 Output Greyscale Image

Fig 5 shows the output of greyscale filter algorithm performed on GPU. Here we have obtained pixel intensity by averaging the RGB channels.

3.2 Negative Filter

By implementing negative filter on CPU for image of resolution 3000 X 3000, time taken for execution was 92.00 ms. It was also implemented on GPU with varying number of Threads per Block and its analysis is presented in Table 2

Table 2 Negative Filter

No. of blocks	No. of Threads/block	Time (ms)
27000000	1	21.03
54000	500	29.02
27000	1000	44.32

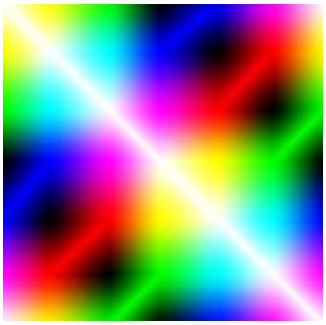

Fig. 6 Output Negative Image

Fig 6 shows the output of negative filter algorithm performed on GPU. By comparing with the original image it can be well observed that diagonal which was of colour Black transformed to white.

3.3 Brightening Filter

By implementing image brightening filter on CPU for image of resolution 3000 X 3000, time taken for execution was 250.00ms. It was also implemented on GPU with varying number of Threads per Block and its analysis is presented in Table 3

Table 3 Brightening Filter

No. of blocks	No. of Threads/block	Time (ms)
27000000	1	27.36
54000	500	46.14
27000	1000	99.39

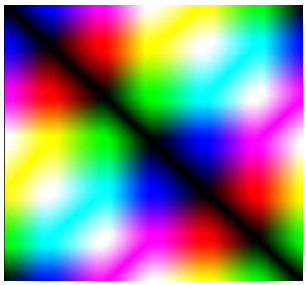

Fig. 7 Output Brightened Image

Fig 7 shows the output of brightening filter algorithm performed on GPU. Here we have brightened the image by 20%. In other words, we have increased the intensity of each pixel by 20%.

3.4 Darkening Filter

By implementing image darkening filter on CPU for image of resolution 3000 X 3000, time taken for execution was 171.00ms. It was also implemented on GPU with varying number of Threads per Block and its analysis ispresented in Table 4

Table 4 Darkening Filter

Nu\. of blocks	No. of Threads/block	Time (ms)
27000000	1	36.71
54000	500	53.78
27000	1000	56.39

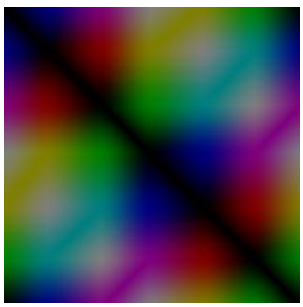

Fig. 8 Output Darkened Image

Fig 8 shows the output of darkening filter algorithm performed on GPU. Here we have darkened the image by 50%. In other words, we have reduced the intensity of each pixel by 50%.

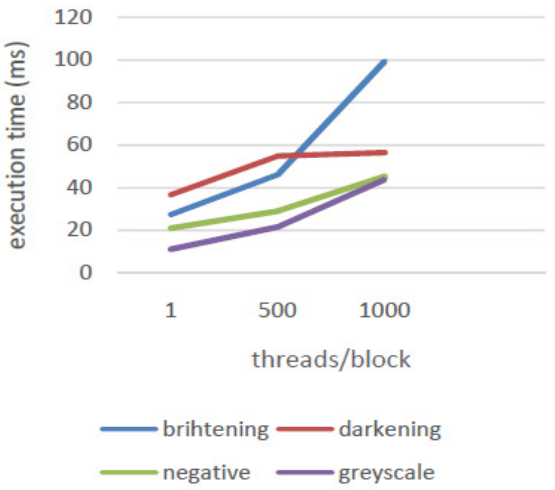

Fig. 9 Performance Analysis for Point to Point image filters on CUDA

*Plot shown in Fig. 9 shows relation between execution time and number of threads per block.

3.5 High Pass Filter and Low Pass Filter

High Pass Filter: By implementing high pass filter on CPU for greyscale image of resolution3000 X 3000, we found time taken for execution was 468.00 ms It was also implemented on GPU and the time taken was 39.3 ms Hence, GPU gives nearly 12 times faster performance.

Fig. 10 Output High Pass Image

Low Pass Filter: Low Pass Filter: By implementing low pass filter on CPU for greyscale image of resolution3000 X 3000, we found time taken for execution was 483.00 ms It was also implemented on GPU and the time taken was 38.1 ms Hence, GPU gives nearly 12 times faster performance.

Fig. 11 Output Low Pass Image

3.6 Sobel Filter for Edge Detection

We have usedsobel mask for edge detection both vertically andhorizontally[5]. An image greyscale image of 3000 X3000 is considered for this purpose. We found timetaken for execution was 297.00 ms. It was alsoimplemented on GPU and the time taken was 28.4ms.Hence, GPU gives nearly 11 times fasterperformance.

Fig 12 Output Vertical Sobel Filter

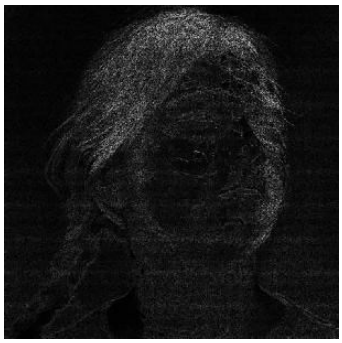

Fig 13 Output Horizontal Sobel Filter

Fig 12 and 13 shows vertically Edge detected image and horizontally edge detected image respectively.

3.7 Recursive Ray Tracing Algorithm

Here we have implemented recursive ray tracing algorithm for 300 passes[7]. With each pass the image becomes more and more clear. Here two spheres and a light source is kept in a box. One sphere is transparent that gives effect of light like refraction, absorption, scattering and et al. Another sphere has reflecting surface on it.Analysis of recursive ray tracing algorithm on GPU is given below:

Samples processed per second: 4411500
Time per pass: 172 ms.
Total time for 300 pass: 51.6 second

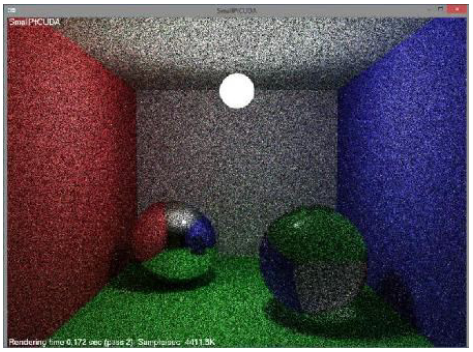

Fig. 14 Recursive Ray Tracing Output atPass 2

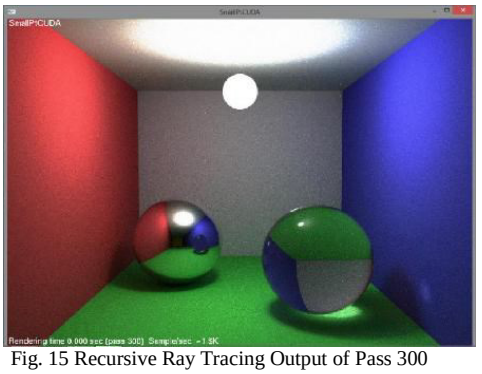

Fig. 15 Recursive Ray Tracing Output of Pass 300

4. Conclusion

In this project we have implemented matrix multiplication as case study. We have implemented various image enhancement algorithms. Finally, by observing and analysing the experimental result we conclude following points:

1. In the applications which involve high inter-thread communication, we have found that with the increase in value of number of threads per block we get faster results.
2. In the applications that are parallel in nature or do not require inter-thread communication, we get better result with lower number of threads per block.
3. Also it was observed while implementing image filters that product of number of blocks and number of thread/blocks should be equal to number of elements to be processed.
4. Recursive ray tracing is highly computation requiring task so we get more benefit of parallelism, here, computation on GPU.

Acknowledgements
We are thankful to all the faculties and laboratory support team of Birla Vishvakarma Mahavidyalaya.

References

1. Courtesy of NVIDIA Corporation
2. NVIDIA CUDA Programming Guide, Version 2.3, Page 3,4
3. Point to point processing of digital images using parallel computing, *by Eric Olmedo, Jorge de la Calleja, Antonio Benitez, and Ma. Auxilio Medina*, page no. 2,3
4. https://developer.nvidia.com/ [This forum has helped lot for development of our project]
5. Digital image processing by Rafel C Gonzalez, third edition, Pearson
6. Massively parallel computing CUDA by Antonino Tumeo & Politecnico di Milano
7. M. Kass, A. Lefohn, and J. Owens, ‘‘Interactive Depth of Field Using Simulated Diffusion on a GPU,’’ tech. report 06-01, Pixar Animation Studios, 2006; http:// graphics.pixar.com/DepthOfField/
8. J. Nickolls et al., ‘‘Scalable Parallel Programming with CUDA,’’ ACM Queue, vol. 6, no. 2, Mar./Apr. 2008, pp. 40-53.
9. S. Ryoo et al., ‘‘Optimization Principles and Application Performance Evaluation of a Multithreaded GPU using CUDA,’’ Proc. 13th ACM SIGPLAN Symp. Principles and Practice of Parallel Programming, ACM Press, 2008, pp. 73-8