Parallel Image processing with CUDA

***Abstract*--Gaussian blur is a common model for image processing. 2D Gaussian blur operations are used in many image processing applications. The execution times of these operations can be rather long, especially when a mass number of images are involved. GPUs, on the other hand, are massively parallel platforms with attractive price, used extensively in the recent years for acceleration of data parallel computation. CUDA is a C-based programming model proposed by NVIDIA for leveraging the parallel computing capabilities of the GPU for general purpose computations. This project processes a large number of images with Gaussian Blur by utilizing the parallel computing platform based on CUDA. Meanwhile, the sequential implementation on CPU was also accomplished as a contrast. The experimental evaluations show that, as the data set grows, the GPU based implementation can be as much as 7.09 times faster than the CPU based implementation, indicating that the larger the data set is, the better the benefits are in using CUDA.**

Keywords—CUDA, image processing, Gaussian Blur

1. INTRODUCTION

Graphics cards are widely used to accelerate gaming and 3D graphics applications. The GPU (Graphics Processing Unit) of a graphics card is built for compute-intensive and highly parallel computations. With the prevalence of high-level APIs (CUDA - Compute Unified Device Architecture), the power of the GPU is being leveraged to accelerate more general purpose and high-performance applications. It has been used in accelerating database operations, solving differential equations, and geometric computations.

Image processing is a well-known and established research field. It is a form of signals processing in which the input is an image, and the output can be an image or anything else that undergoes some meaningful processing. Altering an image to be brighter, or darker is an example of a common image processing tool that is available in basic image editors. Often, processing happens on the entire image, and the same steps are applied to every pixel of the image. This means a lot of repetition of the same work. Newer technology allows better quality images to be taken. This equates to bigger les and longer processing time. With the advancement of CUDA, programming to the GPU is simplified. The technology is ready to be used as a problem-solving tool in the field of image processing [1][2].

This project shows the vast performance gain of using CUDA for image processing. Section 2 gives an overview of the GPU, and gets into the depths of CUDA, its architecture and its programming model. Section 3 consists of the experimental section of this thesis. It provides both the sequential and parallel implementations of image blurring. Section 4 shows the experiment results and Section 5 concludes.

2. OVERVIEW OF APPROACH

2.1 Data Preprocessing

The input data consist of 200 bitmap images, filenames of which are renamed in increasing order starting from 1. This preprocess can be done by a Java program called FileRename. The algorithm is simply iterating all files in the folder and change their names one by one. Before renaming the input data, we must collect enough bitmap images as a simulation of huge workload. Since it is not necessary to use entirely different 200 images, we can repeatedly copy and paste images until we have 200 ones. Then choose 10, 100, 200 as the input number of images to gradually display the benefits of GPU.

2.2 Gaussian blur

Image smoothing is a type of convolution most commonly used to reduce image noise and detail. This is generally done by applying the image through a low pass filter. The filter will retain lower frequency values while reducing high frequency values. The image is smoothed by reducing the disparity between pixels by its nearby pixels.

Image smoothing is sometimes used as a preprocessor for other image operations. Most commonly, an image is smoothed to reduce noise before an edge detection algorithm is applied. Smoothing can be applied to the same image repeatedly until the desired effect is achieved.

A straightforward way to achieve smoothing is by using a mean filter. The idea is to replace each pixel with the average value of all neighboring pixels including itself. One of the advantages of this approach is its simplicity and speed. However, a main disadvantage is that outliers, especially ones that are farthest away from the pixel of interest can create a misrepresentation of the true mean of the neighborhood.

Another way to smooth an image is to use the Gaussian Blur [3]. The Gaussian Blur is a sophisticated image smoothing technique because it reduces the magnitude of high frequencies proportional to their frequencies. It gives less weight to pixels further from the center of the window. The Gaussian function is defined as:

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

where$\sigma$is the standard deviation of the distribution. The discrete kernel at (0,0) and σ= 1 is shown in Figure 2.1.

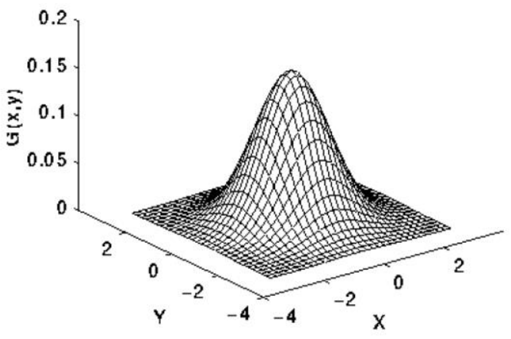


Figure 2.1 Discrete kernel at (0,0) andσ= 1

The Gaussian filter works by using the 2D distribution as a point-spread function. This is achieved by convolving the 2D Gaussian distribution function with the image. We need to produce a discrete approximation to the Gaussian function. This theoretically requires an infinitely large convolution kernel, as the Gaussian distribution is non-zero everywhere. Fortunately, the distribution has approached very close to zero at about three standard deviations from the mean. 99% of the distribution falls within 3 standard deviations. This means we can normally limit the kernel size to contain only values within three standard deviations of the mean. In this project, the kernel has a size equal to 5, and the standard deviation is 1.

2.3 The NVIDIA CUDA Programming Framework

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 2.2.

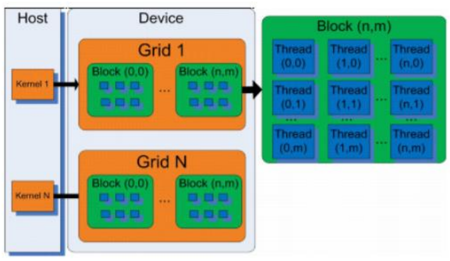


Figure 2.2 Threads and Block Structure of CUDA

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.

2.4 GPU computation in image processing [4]

Most computer vision and image processing tasks perform the same computations on a number of pixels, which is a typical data-parallel operation. Thus, they can take advantage of SIMD architectures and be parallelized effectively on GPU. Several applications of GPU technology for vision have already been reported in the literature.

3. ARCHITECTURE

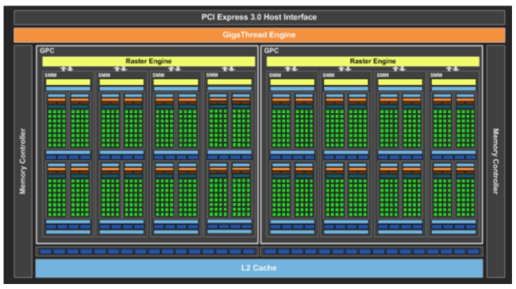


Figure 3.1 GTX 960M architecture

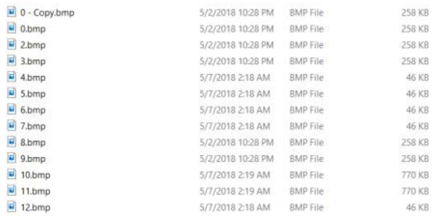
The GPU for this project is NVIDIA GeForce GTX 960M based on Maxwell architecture. Compared to previous architecture, Kepler, Maxwell has been optimized in several details to increase power efficiency. Smaller Streaming Multiprocessors (SMM) with only 128 ALUs (Kepler: 192) and an optimized scheduler should lead to better utilization of the shaders. A Maxwell SMM with 128 ALUs can offer 90 percent of the performance of a Kepler SMX with 192 ALUs. GM107 features 5 SMMs and thus 640 shader cores, 40 TMUs and 16 ROPs (128-bit interface).

Another optimization is the massively enlarged L2 cache (GM107: 2 MB). The increased size can handle some of the memory traffic to allow for a relatively narrow memory interface without significantly reducing the performance [5].

4. IMPLEMENTATION DETAILS

4.1 File name collision

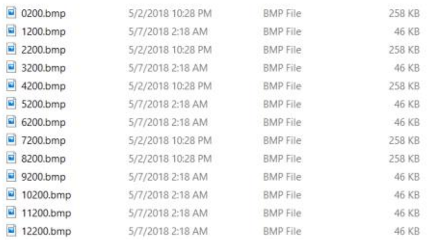
Although the FileRename Java program can rename the filenames in most cases, there is still a potential filename collision in the renaming files process. That is because when the program modifies the filename, if the filename already exists in the file folder. The Windows system will by default add a '-copy' suffix after the output filename of the program. But that is not the file name needed by the project's input. The solution is simply run the FileName programs twice. In the first execution, change all the images' filenames start above 200. This can prevent any filename that collide with number between 0 – 200. In the second execution, change the filenames to let them lie between 0-200.



```
0 - Copy.bmp    5/2/2018 10:28 PM   BMP File   258 KB
0.bmp           5/2/2018 10:28 PM   BMP File   258 KB
2.bmp           5/2/2018 10:28 PM   BMP File   258 KB
3.bmp           5/2/2018 10:28 PM   BMP File   258 KB
4.bmp           5/7/2018 2:18 AM    BMP File    46 KB
5.bmp           5/7/2018 2:18 AM    BMP File    46 KB
6.bmp           5/7/2018 2:18 AM    BMP File    46 KB
7.bmp           5/7/2018 2:18 AM    BMP File    46 KB
8.bmp           5/2/2018 10:28 PM   BMP File   258 KB
9.bmp           5/2/2018 10:28 PM   BMP File   258 KB
10.bmp          5/7/2018 2:19 AM    BMP File   770 KB
11.bmp          5/7/2018 2:19 AM    BMP File   770 KB
12.bmp          5/7/2018 2:18 AM    BMP File    46 KB
```

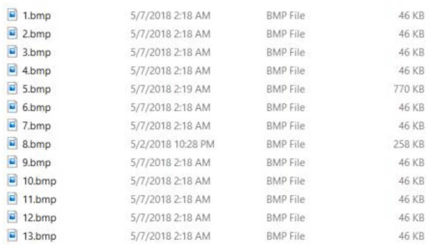
Figure 4.1 Possible file name collision

Figure 4.1 shows the possible file name collision. Since the folder contains a file called '0.bmp' already, the new file named '0.bmp' will be added a suffix to by Windows.



```
0200.bmp    5/2/2018 10:28 PM   BMP File   258 KB
1200.bmp    5/7/2018 2:18 AM    BMP File    46 KB
2200.bmp    5/2/2018 10:28 PM   BMP File   258 KB
3200.bmp    5/7/2018 2:18 AM    BMP File    46 KB
4200.bmp    5/2/2018 10:28 PM   BMP File   258 KB
5200.bmp    5/7/2018 2:18 AM    BMP File    46 KB
6200.bmp    5/7/2018 2:18 AM    BMP File    46 KB
7200.bmp    5/2/2018 10:28 PM   BMP File   258 KB
8200.bmp    5/2/2018 10:28 PM   BMP File   258 KB
9200.bmp    5/7/2018 2:18 AM    BMP File    46 KB
10200.bmp   5/7/2018 2:18 AM    BMP File    46 KB
11200.bmp   5/7/2018 2:18 AM    BMP File    46 KB
12200.bmp   5/7/2018 2:18 AM    BMP File    46 KB
```

Figure 4.2 First execution



1.bmp	5/7/2018 2:18 AM	BMP File	46 KB
2.bmp	5/7/2018 2:18 AM	BMP File	46 KB
3.bmp	5/7/2018 2:18 AM	BMP File	46 KB
4.bmp	5/7/2018 2:18 AM	BMP File	46 KB
5.bmp	5/7/2018 2:19 AM	BMP File	770 KB
6.bmp	5/7/2018 2:18 AM	BMP File	46 KB
7.bmp	5/7/2018 2:18 AM	BMP File	46 KB
8.bmp	5/2/2018 10:28 PM	BMP File	258 KB
9.bmp	5/7/2018 2:18 AM	BMP File	46 KB
10.bmp	5/7/2018 2:18 AM	BMP File	46 KB
11.bmp	5/7/2018 2:18 AM	BMP File	46 KB
12.bmp	5/7/2018 2:18 AM	BMP File	46 KB
13.bmp	5/7/2018 2:18 AM	BMP File	46 KB

Figure 4.3 Second execution

Figure 4.2 is the first execution of the FileRename Program. The file start at least above 200, so as to avoid any collision in second execution. Figure 4.3 is the result after second execution. No name collision exists anymore.

4.2 Parameters

Table 4.1 shows the detailed hardware and software parameters for this project.

CPU	Intel Core i5 4210H
GPU	NVIDIA GeForce 960M
OS	Windows 10

Table 4.1 Summarization of parameters

5. EVALUATION

5.1 CPU implementation result

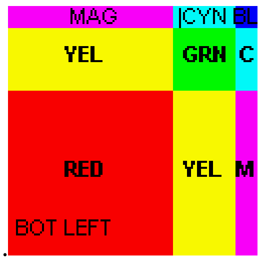


Figure 5.1 Original image

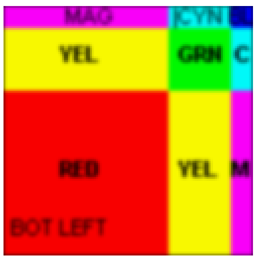


Figure 5.2 Gaussian Blur image

Figure 5.1 is the original image. Figure 5.2 is the image after Gaussian Blur. Table 5.1 shows the time needed in CPU to process the images.

number of input images	CPU(ms)
10	219
100	4031
200	8859

Table 5.1 Mean CPU time

The CPU time is the mean of 10 times recorded data.

5.2 GPU implementation result

The input and output are the same as the CPU implementation because they both use the same blur technique. However, the processing time is reduced vastly. Table 5.2 is the summarization of mean GPU time.

number of input images	GPU(ms)
10	172
100	1062
200	1781

6. CONCLUSION AND FUTURE WORK

Graphics cards have widely been used to accelerate gaming and 3D graphical applications. High level programmable interfaces now allow this technology to be used for general purpose computing. CUDA is the first of its kind from the NVidia tech chain. It is fundamentally sound and easy to use. This project gives an introduction of the type of performance gains that can be achieved by switching over to the parallel programming model.

Image processing algorithms is a category of algorithms that work well in achieving the best benefits out of CUDA. Most algorithms are such that a type of calculation is repeated in massive amounts. This is perfect for utilizing CUDA's massive amounts of threads. Most of these algorithms can be processed independently of each other, making it ideal to spawn threads to perform these calculations simultaneous [6].

The experiment of this project affirms the claim that the larger the data set, the better the benefits are in using CUDA. For 10 input images, the CPU time is 1.27 times as much as GPU time. For 100 images, the multiple becomes 3.80. For 200 images, CPU time is 4.97 times as much as GPU time.

This project introduces CUDA and its benefits, but it does not stop here. A lot of future work can be done. Experiments can be done by using different size grids and blocks. The project is likely to improve with smarter memory usages. A lot can still be explored beyond this project.

REFERENCES

[1] Yang Z, Zhu Y, Pu Y. Parallel image processing based on CUDA[C]//Computer Science and Software Engineering, 2008 International Conference on. IEEE, 2008.

[2] Nguyen H. GPU gems 3[M]. Addison-Wesley Professional, 2007.

[3] Adams A, Gelfand N, Dolson J, et al. Gaussian kd-trees for fast high-dimensional filtering[C]//ACM Transactions on Graphics (ToG). ACM, 2009, 28(3): 21.

[4] Castaño-Díez D, Moser D, Schoenegger A, et al. Performance evaluation of image processing algorithms on the GPU[J]. Journal of structural biology, 2008.

[5] Enmyren J, Kessler C W. SkePU: a multi-backend skeleton programming library for multi-GPU systems[C]//Proceedings of the fourth international workshop on High-level parallel programming and applications. ACM, 2010: 5-14.

[6] Zhang N, Chen Y, Wang J. Image parallel processing based on GPU[C]//Advanced Computer Control (ICACC), 2010 2nd International Conference on. IEEE, 2010.