Using Compute Unified Device Architecture (CUDA) in Parallelizing Different Digital Image Processing Techniques

Mikaela Nadinne A. Silapan and Joseph Anthony C. Hermocilla

***Abstract*— Graphics Processing Units (GPUs) have been conventionally used in the acceleration of 2D, 3D graphics and video rendering. Because of its performance and capability, the GPU has evolved into a highly parallel programmable processor that specializes in memory bandwith utilization and intensive computation. For operations involving graphics, GPUs offer a lot of gigaflops of processing prowess.**

The programmability and competency of GPU in the field of general-purpose computing is exemplified through the implementation of image processing techniques using Compute Unified Device Architecture (CUDA) which makes image processing be implemented with a high-speed performance.

I. Introduction

Different fields of scientific study use images to extract and emphasize significant data. An image is a function of two variables, $f(x, y)$, where x and y are spatial coordinates. The value of f at a coordinate pair (x, y) is called the gray value or intensity of the image at that point. Each pair of coordinates (x, y) is called a pixel. Images can contain hundreds of thousands of pixels. A digital image is created when the values of f for x, y and its intensity are finite [1]. The digital images may be modified, enhanced and processed for various purposes and with different techniques. These techniques are then implemented using some language and the data produced are used for analysis.

The area of study which refers to processing digital images using digital computers is known as digital image processing. One of the most important application of image processing is the enhancement of visual information. The implementation of digital image processing techniques has been successful through the use of computers. However, the computations required for processing digital images involve intensive geometrical and mathematical calculations. In addition, it will make the processor do these calculations for all of the pixels in the image; thus, while computing for values, the processor accesses the memory in a higher rate. It is an operation wherein large matrices are involved, assuming that the image is a two-dimensional array with each cell as a pixel.

The objectives of the field of improving visual perception, including digital image processing, using computers have long been achieved by man [1] through two-dimensional and three-dimensional graphics rendering, simulations, displays and etc. One of the most important tools that lead to the outstanding innovation in visual computing today is the graphics processing unit (GPU).

GPUs were designed for single instruction execution within multiple machines and for accessing multiple pixels in parallel to enhance the quality of computer graphics. Inspite of limitations in executing a single instruction, advances in the capability of GPUs has been already within our reach. As a result, GPU has become a tool not only for graphics rendering, giving rise to General Purpose GPU (GPGPU) computing [2].

A. Significance

During the early days of computers, computer graphics became important for the representation of results written by different kinds of programs. Even though the CPU can satisfy thedemands of some programs using only its resources, some programslook for highly accelerated 3D graphics pipeline [3].

In 2003, standard GPUs with 32 bit floating point numbers and programmable Vertex and Fragment processors were introduced in the market [3]. According to Owens et al., the modern GPU is not only a powerfulgraphics engine but also a highly parallel programmable processor featuring peak arithmetic and memory bandwidth [4]. The GPU is a processing unit built to accelerate the rendering of graphics in display and has played an important role in the field of mainstream computing. The original design of GPUs has long been developed into a more powerful card that even non-graphics applications can use it. The development of the graphics card continues, allowing more powerful processors to be built. The CPU communicates with the GPU through a graphics connector: PCI Express or AGP slot in the motherboard. The connector becomes responsible of transferring all commands and data from the CPU to the GPU and has evolved alongside the GPU itself for the pastyears. The original AGP slot, 32 bits wide, ran at 66 MHz, with a transfer rate of 264 MB/sec. As years go by, transfer rate doubles each available bandwidth, from 2x to 8x. In 2004, the PCI Express standard has reached a maximum bandwidth of 4GB/sec, simultaneously available to and from the GPU [5].

The current GPU can perform concurrent floating point operationsby having hundreds of processors. These processors have onchip memory of 128MB to 4GB which is accessed by the GPU with a faster rate than the CPU accessing the system memory (RAM). The GEForce 8 GPU architecture achieves up to 70GB per second memory transfers when used in some applications, relatively higher than the ordinary system memory interface which resulted to 6.4GB per second.

© 2011 ICS University of the Philippines Los Baños

Component	Bandwidth
GPU Memory Interface	35 GB/sec
PCI Express Bus (x16)	8 GB/sec
CPU Memory Interface	6.4 GB/sec

TABLE I

AVAILABLE MEMORY BANDWIDTH IN CPU AND GPU [5]

Given that there is a lot of internally available bandwidth and the GPU can access the memory on its own and with a substantially higher rate [6], algorithms that run on the GPU can take advantage of this bandwidth to achieve superior performance improvements.

The architecture of GPUs is specifically designed to perform intensive mathematical operations and highly parallel computations: the requirements of being an effective graphics rendering unit. The GEForce 6 GPU Series and higher has the following components in their graphics pipeline [7]:

- GPU Front End
- Vertex Processors
- Primitive Assembly Unit
- Rasterization and Interpolation Unit
- Fragment Processors
- Raster and Frame Buffer Operations Unit

Only two of these components are programmable: the vertex processors and the fragment processors.

The vertex processor allows a program to alter the vertices in the object. The user specifies how the program transforms each vertex. A vertex cache, which is useful for fetching and computing, contains the vertex data. The vertices can be produced with different types of objects: points, lines or triangles. The primitive assembly unit accepts the vertices fromthe vertex processor and decides what object to use.

The rasterization unit calculates which pixels are covered by each primitive; the pixels blocked by objects with nearer depth value are discarded. The pixels which were not discarded now carries its depth and color information, moreover, fragments are produced. Each pixel is mapped on texture wth a ratio of 1:1. Like the vertices, texture data is stored in a cache to minimizethe bandwidth requirements.

The texture and fragment processor applies the fragment program in each fragment, independently with each other. The two processors work on squares of four pixels or quads. They performthe computations for the texture level of each quad, one at a time. The fragment processors operate on hundreds of quads at a time, with each processor operating on one quad. Basically, one instruction is executed a lot of times, hiding the latency of fetching texture data from the cache. The fragments go through the pipelines many times. Computations and memory access are done everytime.

The order in which each fragment is rasterized is the order when the fragment leaves the fragment processor. Then the data is written in the frame buffer, ready for rendering [5]. The design of GPU provide more units devoted to data processing rather than memory managing. These data processing units in GPU allow parallelization, where only single instruction is executedin all units. [8] With all the processors running parallel portions of an application, acceleration is met. Based on the nature and features of the GPU, it is one of the most suitable device to use in accelerating the execution ofalgorithms, such as image processing.

According to Marathe, the following characteristics of image processing algorithms served as the motivation for its parallelization [7]:

1) Image processing algorithms deal with large volumes of data of a particular type
2) the algorithms demand computational power which can be optimized through parallelization
3) they require realtime processing

On the other hand, GPU implements many graphics primitive operations that satisfies the need for faster rendering of graphics than CPUs. Specifically, GPUs are efficient in performing the following operations:

1) Fast parallel floating point processing
2) single instruction multiple data operations
3) high computation per memory access

A very prominent operation in processing images is convolution. It is defined as a mathematical operation involvingtwo functions f and g, resulting in a third function. In image processing, the convolution operation calculates the weighted average of a pixel's neighborhood [9], using a convolution mask, which is a matrix of the weights of each neighboring pixel. Each value in the neighborhood is multiplied with a corresponding value in the convolution mask. These masks may have varying dimensions and as it increases dimension, performing convolutions become time-costly.

The most common dimension of convolution masks is 3x3 but there are many algorithms that use a 65x65 dimensional mask. For instance, when performing a convolution in an 800x800 image, there are about 65x65x800x800 memory operations, array lookups and arithmetic operations such as multiplication and addition.

The characteristics of image processing algorithms and GPU's capability make them compatible in terms of implementation. Therefore, GPU can respond to our need for parallelization, which is deemed as the future of computing. Due to this fact, computer scientists have utilized GPU in making faster algorithms for processing images. There are many publications and researches that involve the GPU in image processing. They have successfully implemented image processing operations through the programmable pipeline of GPU and its fragment and vector pipelines.

To help programmers in writing algorithms for the GPU, a new tool has been developed by NVIDIA: Compute Unified Device Architecure or CUDA. The design of CUDA ables programmers to use the graphics unit to perform operations that are usually done by CPU.

Image processing techniques involve applying the same mathematical operation in all the pixels of the image. With thischaracteristic of image processing, CUDA can offer immense difference in the rate of performing these operations by runningthe code on each concurrently executing threads and produce the same output. The GPU's texture memory available for use by CUDA can be used to store the image for faster pixel value fetching.

CUDA is a scalable programming model for software environment and parallel computing. The technology is available for a large amount of money but performance rate achieved could be up to 500 GFLOPS (floating point operation per second).

B. Objectives

This study aims to make the algorithms of image processing yield faster output without compromise to the quality of the resulting image. The techniquesthat will be implemented in the study are thresholding, gamma correction, image brightening, smoothing, edge detection, dilation and erosion. Specifically, this study aims:

1) To parallelize the selected image processing techniques using CUDA
2) To render the output image of GPU and CPU implementation and
3) To measure the running time of each process for comparison

II. Review of Related Literature

A. GPU as Tool for Acceleration of Algorithms

The people behind SINTEF ICT, an independent research group, used the GPU, with 32 bit floating point arithmetic and programmable vertex and fragment shaders, as an important computational resource within Computer Aided Graphics Design (CAGD) and Partial Differential Equation (PDE) based simulations. Results in solving partial differential equations show a speedupby a factor between 10 and 20 in comparison with a Pentium 2.8 GHz CPU. According to the project, the speedup is best achieved when shaders perform many floating point instructions rather than with a few instructions [3]. The project used GPU for a non-graphics application and achieved a considerably high increase in performance.

In molecular biology, researchers use supercomputers to be able to satisfy the system requirements of their study. The presentation of Schulten at the GPU Technology Conference showed how powerful GPU is in terms of simulating virus infections, molecular dynamics and E.coli bacteria activity and other researches such as light energy consumption of nature, protein synthesis, radial distribution functions, quantum chemistry visualization and protein folding [10]. These kind of scientific researches require producing fast response, e.g. investigation of drug resistance of the swine flu virus. Results show that GPU's capability merely equals the result of the simulations done with supercomputers.

Last April 2011, Lactuan presented his research in the Institute of Computer Science, University of the Philippines Los Baños. He implemented the simulation of the growth of cancer cells using CUDA. According to his research, each cell can grow, stay the same or die, independently with each other and the probabilities of one cell and what will happen to it after an amount of time was computed. The use of CUDA in the simulation is efficient but when the matrix has been increased, GPU has been outperformed by CPU. The author mentioned that the CUDA code can still be improved [11]to attain better results.

B. Parallelization of Image Processing

In 1999, Hopf released a publication that discussed image processing with GPUs, being the first of its kind. He mentioned in his research that image processing techniques such as filtering and feature extraction like edge detection have some complexities. For that, there are no other way to make the process interactive enough for the users' visual perception. Therefore, he proposed that graphical units be used to avoid slowness when implementing the visual transformation cycles [12]. Hopf used wavelet transformation using Fast Fourier Transformation (FFT) in manipulating images. This study did not adopt the FFT approach, however, some filtering techniques and edge detection were implemented.

Colantoni, programmed the GPU to process color images. He used the capability of current graphics cards, the programmability features, in the analysis and processing of color images. He implemented five algorithms: local mean filtering, RGB to Lab and RGB to HSV color spaces conversions, local principal component analysis and anisotropic diffusion filtering. NVIDIA NV30 graphics card have been used in the research and the implementation of the algorithms in GPU yield faster output than CPU [13]. The algorithm that will be adopted from his research is local mean filtering.

In 2007, Narain used GPU in the application of color correction, convolution, wavelet transforms, anistropic diffusion, depth of field, HDR and tone mapping. In convolution, a technique that will also be used in this study, Narain created a kernel for theconvolution mask and passed each coordinate (x, y) as data arrays. The convolution function is computed using the formula $g[x, y] = \Sigma f[x + i, y + j] * h[i, j]$. This operation normally requires NxN texture lookups, where N is the dimension of the image. The paper proposed a method that limits the number of texture lookups. Using cache coherence, texture lookups are fast and for this, the texture memory readily available through CUDA was used in this study. The results of Narain's research show that GPU outstands the performance of CPU in all cases. The different filtering techniques implemented in this study used the convolution concept discussed in Narain's [14].

Trajano implemented different image processing routines for execution in single and distributed processors. He mentioned that the time consumed by the application of the techniques are considered in the process, thus, using distributed systems instead of a single processor in manipulating images [15]. The routines that he implemented consists of thresholding, brightness and contrast adjusting, inverting, gamma, smoothing, detecting edges, eroding and dilating. This study will implement all of the techniques for comparison. His study parallelized the image processing techniques using distributed systems called nodes while this study will implement the parallelization using multiple concurrent threads.

C. Image Processing Parallelization Using GPUs

Payne et al. has studied the advantage of GPU over CPU in image processing tasks that require convolving the image with a mask [16] and as a result of the study, straight-forward 2D convolutions demonstrated a ratio of 130:1in terms of

computation time between the GPU and CPU, respectively. Averagely, their tests attained a speed-up of 59:1.

Castaño-Díez et al. presented three different implementations of SART, two of them using CUDA, and one with a texture-based approach using Cg [17]. All the implementations achieved approximately 50 times speedup, compared to their CPU equivalents. The Cg implementation performed the best, however, writing and debugging it required considerably higher effort than in CUDA, as noted by the authors. The GPU-accelerated weighted back projection, used for 3D reconstruction from 2D electron micrographs was tested, as well. This included low-pass filtering of the acquired images, rotation, Fourier space weighting, and backprojection into a 3D image. They attained speedups of up to 10 times on graphics hardware.

The OpenVIDIA project explores the notion of using computer graphics hardware in reverse to accelerate the computer vision task of image analysis even though graphics hardware was initially designed for rendering images, or image synthesis. Furthermore, OpenVIDIA explores a parallel graphics for vision" architecture created by placing multiple computer graphics cardson a single motherboard. This creates a low cost, commodity, architecture for hardware accelerated computer vision and signal processing. The GPU has been applied to other calculations beyond graphics [6].

Due to the high performance rate achieved by using the GPU, particularly through CUDA, this study will be able to test the computing capabilities of the GPU and how CUDA can utilize these capabilities.

III. Methodology

The study was conducted using an ASUS K43S machine with an Intel Core i3 processor with the following specifications:

- Clock Rate: 2.2GHz
- 2GB RAM
- 640GB HDD

The graphics card that will be used is the NVIDIA GeForce GT520M. The specifications of the graphics card are given below:

- Global Memory - 1GB
- 48 Multiprocessors
- Maximum threads per block: 1024
- Clock Rate: 1.48GHz

A. System Requirements

In desktop machines, the driver for the graphics device must be installed. It can be downloaded from NVIDIA's website. As for this study, the device driver were pre-installed because the driver differs from manufacturer to manufacturer. To complete the study, the following were installed:

- NVIDIA Computing SDK 3.2 (OpenGL and GLUT included)
- CUDA 3.2 Toolkit
- Microsoft Visual C++ 2008 Express Edition
- SDL Image
- GLUI

Compiling CUDA programs were difficult when done using NVCC, the CUDA compiler. Visual C++ 2008 was installed, to avoid problems in compiling and to handle the linking and executing of the project, and other stuffs related to the programming environment. OpenGL is a cross-platform Application Programming Interface (API) developed by Silicon Graphics, for writing application that produce 2D and 3D graphics. It is directly operable with CUDA, so rendering of the CUDA kernel's output is easier with OpenGL. GLUT is an OpenGL utility toolkit used for developing a simple windowing application. This makes the rendering of the output visible on screen, in a simple window. The SDL Image library is an extension of SDL library which is written in C++ and is widely used for game development. It offers a lot of image processing functions like loading and manipulating images. This library was used in loading the image data for CUDA. GLUI library is the UI-building utility developed by the makers of GLUT, to add more useful interface in the GLUT window such asbuttons, panels, text boxes, spinners and etc.

B. Image Processing Techniques

The image processing routines implemented in the study were based from the study conducted by Trajano [15] in 2010.

1) Thresholding: A technique used for detecting object pixels in an image. The decision rule for marking object pixel is that its value should be greater than a particular value or threshold. Otherwise, it will be marked as a background pixel. The background pixels are white while the object is black after the application of the technique.

2) Brightness Adjustment: The brightness of an imagecan be changed using the formula $f(i) = i(1 - p) + p$ where i is the intensity value of a pixel, p is the brightening parameter and $f(i)$ is the new value of the pixel [18].

3) Invert: Inverting the image will result to an image where the pixels that are originally bright will become dark and vice versa. The formula for inverting an image is computed as $f(i) = 1 - i$ where i is the intensity value of a pixel and $f(i)$ is the new value of the pixel [18].

4) Gamma Correction: Gamma Correction routine modifies the contrast of an image. The contrast can be modified using the formula $f(i) = i^p$ where i is the intensity value of a pixel, p is the contrasting parameter and $f(i)$ is the new value of the pixel [18]. The value of p ranges from 0 to 5.

5) Smoothing: Smoothing an image comprises the application of convolution using a smoothing filter. The technique that will be used for smoothing the image is mean filtering.

6) Edge Detection: The technique that will be used is the Sobel edge detection algorithm which uses the Sobel horizontal and vertical filter.

7) Dilation: In dilating gray images, the value of the output pixel is the maximum value of all the pixels in the input pixel's neighborhood. In a binary image, if any of the pixels is set to the value 1, the output pixel is set to 1. [9]

8) Erosion: In eroding gray images, the value of theoutput pixel is the minimum value of all the pixels in the inputpixel's neighborhood. In a binary image, if any of the pixels isset to 0, the output pixel is set to 0 [9].

9) Sharpening: Images are sharpened to enhance the features such as lines and colors. To obtain this enhancements, a difference operator filter must be used. A high pass filter will be used to sharpen the image.

C. Parallelization of the Image Processing Techniques

Parallel programming is exhausted by the GPU as it has multiple processors to execute the commands concurrently. The architecture of the GPU, an abstract design of a modern GPU, is shown in Figure 1. The green boxes correspond to a processor which performs arithmetic operations. Each processor has direct access to the on-chip memory, the orange boxes at the left. CUDA utilizes the capability of the GPU through the different key parallel abstractions it offers: zillions of lightweight threads, hierarchy of concurrent threads, lightweight synchronization primitives and shared memory model for cooperating threads [19].

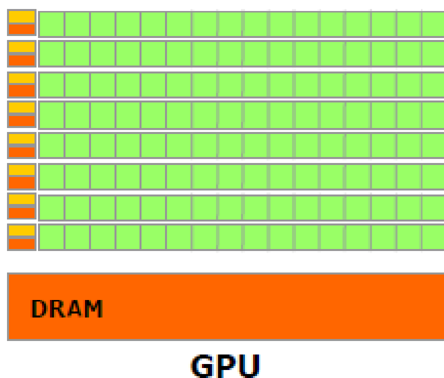

Fig. 1. The GPU Architecture [8].

When programmed through CUDA, the GPU can be seen as a compute device which can execute a high number of threads in parallel. It operates as a co-processor to the CPU, the host device [8]. With this, there are two types of codes in a CUDA program: a host code and kernel code. Figure 2 shows the graphical illustration of the flow of commands and where it will be executed.

Multiple threads can be processed simultaneously because of the hundreds of processors that perform the operation. The kernel code launches the threads thatwill execute the image processing codes. The threads executed on the GPU form a 1D, 2-D, or 3-D structure, called a block. The blocks are arranged in 1D or 2-D layout, called a grid. The structure that will be used in this study is two-dimensional, to be compatible with the layout of an image.

Each thread can be exclusively defined by its coordinates within a block, and by coordinates of its block within a grid. Thus, each block in the grid has a unique ID and each thread in the block has their unique thread ID [20], similar to how a cell in an array is accessed. This type of execution consists of successive kernel execution in the device as Figure 3 shows, allowing more room for data processing.

Each block of threads is executed by a single multiprocessor, on a single shared memory, so all threads of a single block

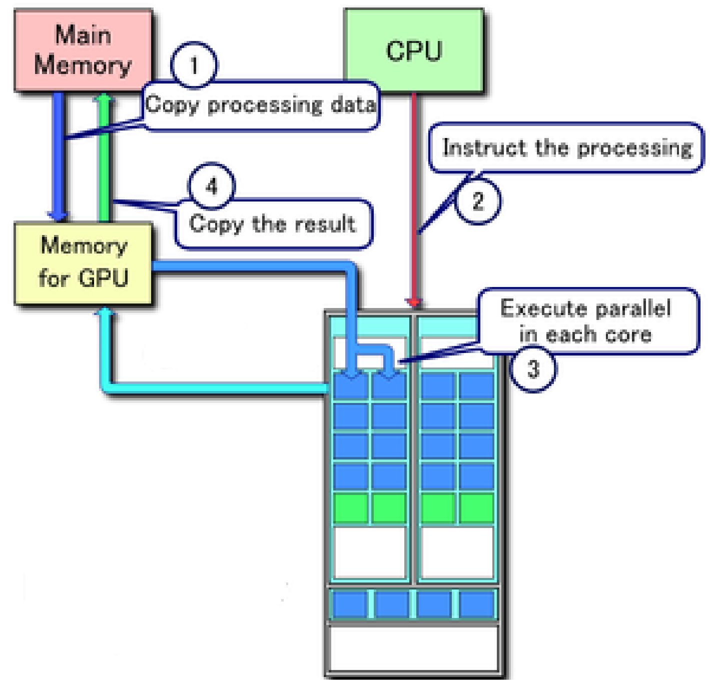

Fig. 2. The processing flow of CUDA programs.

can share. The number of blocks that each multiprocessor can process depends on how many registers per thread and how much shared memory per block were required by the kernel. The multiprocessorś registers and shared memory are split among all the threads of the batch of blocks [8].

D. Implementation in CUDA

The image processing kernels implemented in CUDA was launched using a grid of thread blocks with the dimension imageW/blockwidth x imageH/blockHeight with blockWidth x blockWidth threads each. BlockWidth is a constant with value 8, meaning there will be 8x8 threads executed in each block. For better illustration, suppose the image width and height is 160. The grid of blocks will be of size 20x20 and each block will have to execute 64 threads. The number of threads will add up to the total number of pixels in the image.

If the threads allocated is less than the number of pixels in the image, there will be parts of the image that will not be processed. The number of threads and the dimension of the grid of blocks is the most essential part of the kernel launch.

Parallelization is achieved by implementing the same operation in all the pixels, with one thread accessing each pixel, computing the values concurrently. Each lightweight thread has a unique thread ID, similar to the index of each cellin a 2D array, so consistency of data is ensured. In CUDA, we need not loop around the image from its width to height, as the threads running perform operations in a per-pixel manner. The threads accessing each pixel in every kernel launch parallelized the image processing techniques as each thread concurrently executes the operation to all the pixels.

Thread execution do not finish at the same time, so synchronization is important. The synchronization primitive in

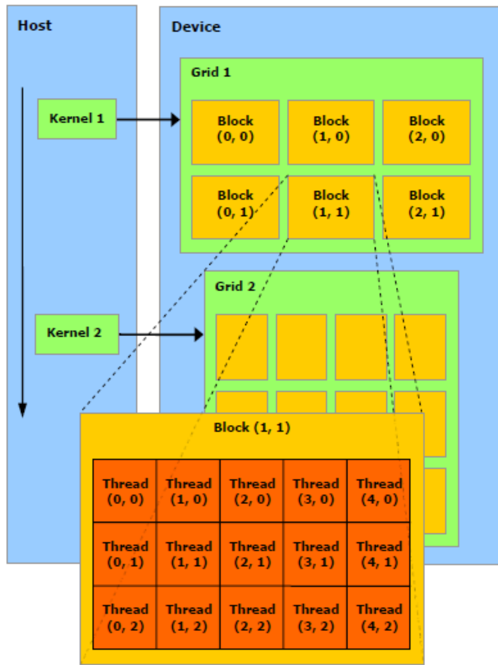

Fig. 3. A diagram showing the grid of thread blocks, the thread blocks and the threads in each block. [8].

CUDA can be called after the threads have been executed, for accurate results. The processing techniques were implemented in CUDA using texture memory, device memory and global memory. The most utilized of these is the texture as it contributed to the speedup of the processing kernels. Texture fetching is relatively faster than looking up global memory because of the cache. The image result is mapped onto the texture memory, and is then accessible to each thread, without the hassle of passing the 2D array containing the image data in every kernel launch.

E. Rendering the output using OpenGL

An OpenGL texture and buffer is created at the very beginning of the program because the buffer is where the data is fetched in order to get it rendered on screen. After creating the texture, parameters are set, and the image data is copied to the texture.The texture is created and image data is copied into texture using the functions, respectively:

```
glGenTextures(1, &textureID);
glTexImage2D(..., source_image);
```

OpenGL buffers are then created and initialized using the following functions:

```
glGenBuffers(1, &BufferID);
glBufferData(..., source_image,...);
```

The output image data from CUDA is then stored in an OpenGL frame buffer by registering the OpenGL buffer with CUDA's resource, to be able to use the CUDA's resource when OpenGL renders the output image.

GLUT is responsible for executing the main thread. It handles such events by specifying which function it'll call once a key has been pressed or if a button is pressed. It has an idle and reshape routines that handles what to do when the user doesn't do anything on the window and what will happen to the rendered output when the window is reshaped, respectively. The most important function in the GLUT toolkit is its display function, as it handles what will be rendered on the window. The calling of the CUDA kernels are inside the display GLUT function as it is where data is rendered real-time.

These are the GLUT functions that was implemented in the project:

- Idle - for handling what the window will display when the program is idle
- Reshape - for reshaping the window
- Display - for rendering the data
- Control Callback - for handling action events

F. User Interface

The main application interface consists of two subwindows: the controls and the rendering window.

1) Controls Window: This window consists of four panels, as shown in figure 4. The first one is where the user can browse an image to be loaded by the application. The second panel is composed of image enhancement options: brightness and contrast adjustment, threshold and iteration value and mask and box radius adjustment. The options are dependent on the selected image processing technique, some options will be disabled if not applicable with the selected technique. The third panel contains the image processing technique that will be performed by the selected processor, that will in turn be displayed on the rendering window. The last panel is where the user can select which processor will execute the processing techniques and where the user can see the time of execution of each.

2) Rendering Window: This is where the user can see the output of the processing techniques. The images rendered were scaled to fit the screen of the machine, but the aspect ratio is maintained so the image's visual appearance will not be affected.

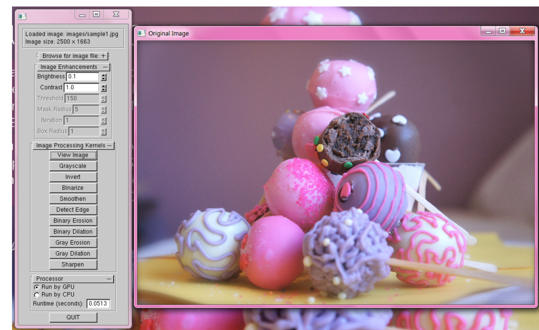

Fig. 4. The main application window, the left panel consists of the controls and the other is the rendering screen.

G. Performance Evaluation

The running time of the CPU functions were calculated using the commands:

Fig. 5. One of the test images with dimension 5800 x 4350

Fig. 6. Output image after applying grayscale

```
float runtime;
clock_t start, stop;
assert((start = clock())!=-1);

//image processing code here

stop = clock(); runtime = (float)
(stop-start)/CLOCKS_PER_SEC;
```

On the other hand, CUDA kernels' execution time is measured using the commands:

```
unsigned int timer;
float runtime;
cutCreateTimer(&timer);
cutStartTimer(timer);

//image processing code here

cutStopTimer(timer);
runtime = cutGetTimerValue(timer)/1000;
```

Each processing technique kernel or function returns the running time, in seconds, measured using the above implementation, with data type float. The performance of each processor in executing the stated processing techniques will be measured using a simple benchmark that executes the techniques for a given number of cycles or iterations. The running time of each process is accumulated and the average is computed.

IV. Results and Discussion

The study involved creating a new function to load different image formats. SDL Image was the library used to load image data from disk. The image formats tested in the application were png, jpeg and bmp. There is a limitation in the size of image to be used. Images of dimension 6000 or higher cannot be handled by the application, SDL Image library returns false when allocating memory for the image. The test images used were of dimension 1280 x 800, 2500 x 1663 and 5800 x 4350.

The test image used, as shown in Figure 8, is selected because of the certain features in the image that can be extracted or enhanced where the results will be easier to see. Such features where the edges, the exposure and the colors.

CUDA's syntax made the implementation easier because of the familiarity with the C and C++ language. The selected image processing routines were successfully parallelized using CUDA through the successive thread execution after launching the kernel. Each thread was executed concurrently in multiple processors. Consistency of the data was achieved using CUDA's synchronization routine.

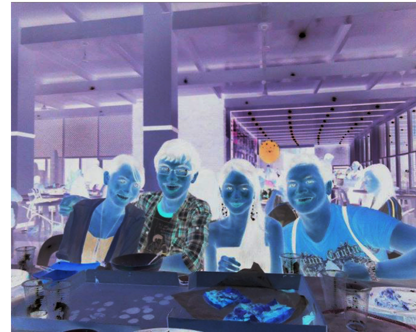

Fig. 7. Output image after applying invert

The results of the parallelized and sequential routines showed minimal output differences but the parallel execution of the algorithms showed faster execution time. All the processing techniques have been executed and rendered a lot of times faster using the GPU. Figure 5 shows the comparison of the running time for each using different images and they have showed equivalent results. The GPU didn't took a second to execute each of the techniques, except the smoothing algorithm using the largest test image.

The execution time of the CPU functions was also reflected in rendering the output images. When rendering results from the CPU, the application takes approximately 30 seconds to

Fig. 8. Output image after applying edge detection

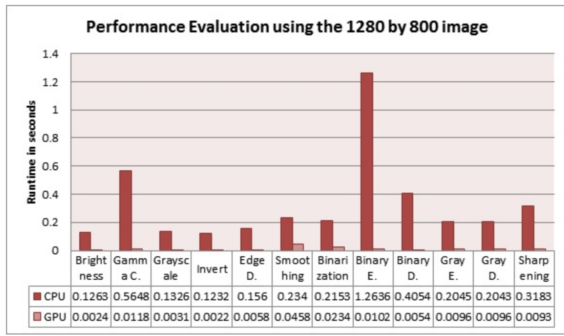

Fig. 9. The running time of the processing routines for test image 1.

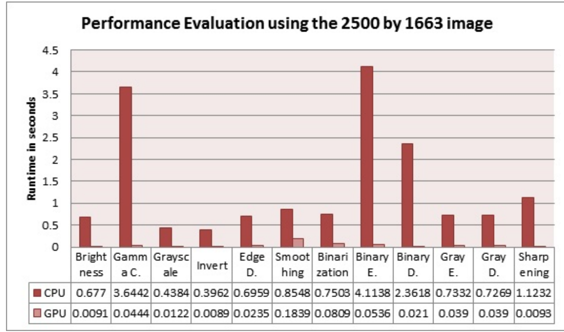

Fig. 10. The running time of the processing routines for test image 2.

respond using the largest test image, with dimension 5800 by 4350. The convolution-based image processing techniques such as sobel edge detection, erosion and dilation require each processor to access multiple memory locations each iteration. The result of implementing these convolution-based techniques in the GPU yield an execution time of less than a second.

The GPU outperformed the CPU of the machine used in the study by executing the image processing routines more than a hundred times faster in some algorithms. Figure 12 shows the ratio between the running time of each implemented techniques using all the test images.

V. Conclusion

The algorithm for each technique need not be different from the CUDA code to be parallelized. Instead, CUDA's architecture allows parallelization through SIMT execution, textures, shared memory and successive kernel execution. The hundreds of processors performing arithmetic operations in the GPU and CUDA's capability of launching thousands of independent threads has helped achieve parallelization.

Similar result was reflected in Trajano's study of parallelizing image processing techniques through distributing the tasks to different nodes. His study showed faster results during the parallelization [15]. Therefore, the parallelization of image processing techniques showed greater speed compared to sequentially executed operations.

The running time of the GPU kernels are relative to the size of the image used. The larger the size of the image, the slower the running time in the GPU but the values are relatively near.

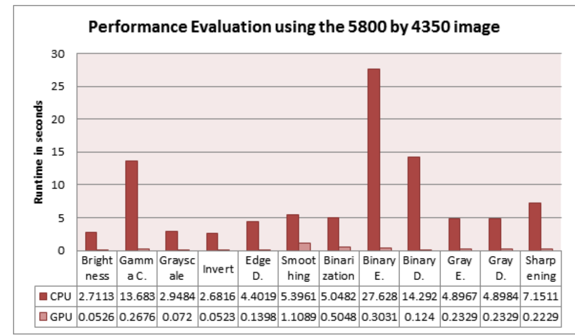

Fig. 11. The running time of the processing routines for test image 3.

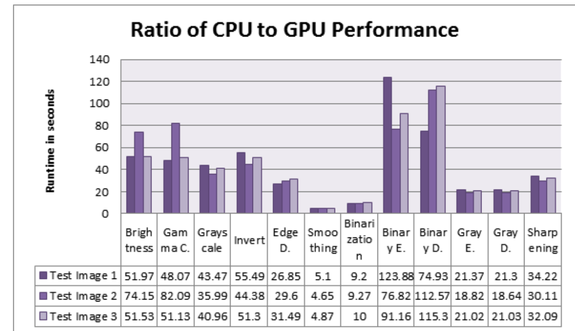

Fig. 12. Ratio of the running time of CPU and GPU-based image processing.

This is an effect of parallelization, given that the image size hardly affected the results.

Floating-point operations and memory location fetching are accelerated using CUDA, and the GPU, based on the results of the gamma correction technique and the convolution-based techniques. In addition, all the processes, ranging from copying the image to convolution, the GPU has outperformed the CPU with more than a hundred times faster execution time.

Each output image is rendered real-time by the use of OpenGL and CUDA. OpenGL renders the image faster when the data is coming from the CPU. Developers has created functions for CUDA-OpenGL inter-operability, so the use of the two in this study really created an advantage in terms of the rendering of output. With this, results of CUDA kernels are best rendered using OpenGL.

Image processing techniques implemented in CUDA exhausts the GPU for general purpose programming. Using CUDA, the GPU's capability for accelerating processes, even without the knowledge of how the GPU graphics pipeline, has been made possible. This could open doors for programmers to develop more powerful softwares and simulations with greater performance.

One important factor to consider in assessing the speed of the implementation is the percentage that the processor allocate its service for such applications. Only 25 percent of the CPU was used by the application, as shown by the task manager application in Windows 7. With this, the study can't fully assess the capability of the CPU when handling compute-intensive operations such as image processing.

ACKNOWLEDGEMENT

I like to thank my adviser, Sir Joseph Anthony Hermocilla, for his support and motivation for my Special Problem proposal. Sir, thanks for the support and for answering my questions everytime I consult. I especially thank my parents, Alicia and Rolando, for the support, for the guidance, support and inspiration. Thank you for making this possible. To my sisters and brothers, even though you make me more stressed while I was doing this, thank you for understanding. I definitely thank Papa Lord for giving me this opportunity to study and for this life He has given me. I also would like to give special thanks to my chimuy, for his encouragement and comfort in times of depression and stress. Thank you for the support and for being always there for me.

REFERENCES

[1] R. Gonzales and R. Woods, *Digital Image Processing*. Prentice Hall, 2002, ch. 1.
[2] J. Fung and S. Mann. (2008) Using graphics devices in reverse: Gpu-based image processing and computer vision.
[3] T. Dokken, T. R. Hagen, and J. M. Hjelmervik. (2005) The gpu as a high performance computational resource.
[4] J. D. Owens *et al.*, “Gpu computing,” presented at the IEEE Xplore, 2008.
[5] E. Kilgariff and R. Fernando, “The geforce 6 series gpu architecture,” *GPU Gems 2*, vol. 30, no. 1, pp. 4–6, 2005.
[6] J. Fung. (2005) Openvidia: Parallel gpu computer vision. NVIDIA Corporation. [Online]. Available: http://openvidia.sourceforge.net
[7] A. Marathe. (2009) Digital image processing with gpu.
[8] NVIDIA. (2007) Nvidia cuda programming guide. NVIDIA Corporation. USA.
[9] J. Matthews. Convolution and correlation. [Online]. Available: http://www.generation5.org/content/2002/convolution.asp
[10] K. Schulten, “Gpu technology conference,” presented at the GTC, San Jose Convention Center, California, 2010.
[11] L. K. Lactuan, “Implementing simulation of cancer cell growth using compute unified device architecture,” Undergraduate Special Problem, University of the Philippines, Los Baños, Apr. 2011.
[12] M. Hopf and T. Ertl, “Hardware-based wavelet transformations,” presented at the Workshop of Vision Modeling and Visualization, 1999.
[13] P. Colantoni, N. Boukala, and J. D. Rugna, “Fast and accurate color image processing using 3d graphics cards,” presented at the 8th International Fall Workshop: Vision Modeling and Visualization, Munich, Germany, 2003.
[14] R. Narain. (2007) Image processing on gpus. [Online]. Available: http://gamma.cs.unc.edu/courses/gpgpu-s07/lectures/imgproc.pdf
[15] A. Trajano, “Implementation of selected image processing routines for single and distributed processors,” Undergraduate Special Problem, University of the Philippines, Los Baños, Apr. 2010.
[16] B. R. P. et al. (2005) Accelerated 2d image processing on gpus.
[17] D. Castano-Diez *et al.*, “Performance evaluation of image processing algorithms on the gpu,” *Journal of Structural Biology*, vol. 164, no. 1, pp. 153–160, 2008.
[18] V. Mariano. Image fundamentals.
[19] D. Luebke. (2008) Beyong programmable shading: In action.
[20] J. Seland. (2008) Cuda programming. [Online]. Available: http://heim.ifi.uio.no/ knutm/geilo2008/seland.pdf

Mikaela Nadinne A. Silapan She is an undergraduate student under the BSComputer Science program of the Institute of Computer Science, University of the Philippines Los Baños. She likes shopping and singing, loves basketball and her friends. Her ultimate favorite color is pink. She is a proud member of The CPS Triangle and has been the Scholastics Committee Head of the organization from 2010-2012.