Improve Computational Complexity of Sobel Edge Detection using Parallel Contract Anytime Algorithm

Department of Computer Science and Engineering
School of Engineering and Computer Science
BRAC University

Supervisor **Dr. Jia Uddin**

Md. Kamal Hossain	12101073
Md. Asif Ibtehaz	14341001
Md. Assaduzzaman Ashique	12301017

DECLARATION

We, hereby declare that this thesis is based on the results found by ourselves. Materials of work found by other researcher are mentioned by reference. This Thesis, neither in whole or in part, has been previously submitted for any degree.

Supervisor

Dr. Jia Uddin
Ass professor
BRAC University
Department of Computer Science & Engineering
jia.uddin@bracu.ac.bd

Authors

Md. Kamal Hossain
12101073
Shajal16@gmail.com

Md. Asif Ibtehaz
14341001
Ibtehaz.shawon@gmail.com

Md. Assaduzzaman Ashique
12301017
ashique12301017@gmali.com

Contents

List of Figures

List of Tables

Preface

Image processing has a vast impact on today's world. Every steps we are giving forward being dependent on image processing kinds of thing. This is call modernization of the world. And in this world image processing is needed as we are giving vision to our robots, we are operating robotic arms to do operations there we need high quality camera with good encoding software. Again we are trying to shorten the process of any decision taken visually by test or judging by cameras there we need image processing.

Whenever image processing topic comes there is always some limitations and its time range or time limit that needed to do the process. In normal CPU it's so tough and risky to run a complete image processing as this might damage the machine. For high quality CPU it's okay but till now it takes a lot of time to do a good quality image processing. And sometimes may be after waiting a long time the expected result is not there so it increases the disappointment.

Another problem is we cannot get a result partially based on the present methods of image processing or edge detection. We have to wait the whole time where a partial processed could have worthy.

Based on these problems stated above we came with an idea where we can resolve both the problems based on hardware and software solution.

Now for that we have to go through some algorithms and to solve the second problem only solution is using anytime algorithm where we can stop the program anytime we want to and get a result that will help. For initial work stage we chose only edge detecting as edge detecting is the main part of image processing.

When it comes to solve the first problem none but the solution came in to our head is parallel computing and that's been made easy by NVIDIA releasing support for GPU's to use with CPU's. This GPU technology is the main part here and also challenge for as.

Anytime algorithm is a artificial intelligence based algorithm. And the main challenge was to implement it on NVIDIA CUDA C programming. Where we have gone through many books and web sites and the outcome was disappointed for us. As once only the anytime algorithm was theoretically solve using parallel computing but they failed to implement practically. That was a bad news for us as all other so far implemented AI algorithms in parallel computing doesn't work as great as thought in theoretically.

Then we decided to partition the problem and started working on it. We have to wait two month long to get our first progress. And it was implementing only Sobel in CUDA C. and so we did it. Then our next part was to do something with anytime algorithm. We had to go through tons of references and works done by others for anytime algorithm and till then all the works for anytime algorithm was done in CPU based models only. Never in any parallel things.

After making a process model of anytime algorithm for GPU programming we did it successfully. We are now able to get a result on given time. If we give the process till how many we want to get the final result now our can do that.

We did successfully implemented anytime algorithm with Sobel edge detection method in GPU computing by NVIDIA that is CUDA C. Comparing the result of the output and other conventional CPU edge detection process we came to an end with 4 times faster edge detection process.

Acknowledgement

With great contentment, we would like to express our gratitude to all the people who helped us through this research period. First of all, special thanks are due to Assistant Professor Dr. Jia Uddin, our advisor, who gave us invaluable advice and instructions and helped us to implement this research work by giving advises how should we proceed. What should we do in time where we were stuck in work progress he helped us all the time. Providing us with vast resources he made our work easier.

Then we would like to thank Dr. Md. Haider Ali our honorable Head of the Department. He is a very kind person to permit us to start our research work. And because of him we learnt a lot about Image processing. As well as Dilruba Showkat Mam's given idea about image processing course did helped a lot.

Our great gratitude to also our honorable teacher Rubel Biswas Sir and Professor Mohammad Zahidur Rahman sir. Because of them we get interested on Artificial Intelligence. Where we did give a think to solve our problem by Artificial intelligence.

We want to thank Moin Mostakim Sir because of him we could think about algorithms to solve any problem and how to approach towards algorithm.

We would be so grateful to Annajiat Alim Rasel Sir. He helped us to get allocated Lab room to do our research work there.

Then we would like to thank those people who were in the Lab with us. Though they had other research topic but they sometimes helped us a lot to complete our work properly.

Finally, we would like to thank our parents. We owe a lot to them. Without their support we might not come this far. They helped us in every aspects of life. They encourage us to continue our research. This work would not be came to light without their collective help.

Abstract

Edge detection is a considerably important factor in image or video processing. Detecting the edges of an image play a significant role in image segmentation, data compression, well matching, and image reconstruction. There are several approaches available to detect the edges of an image. In this paper we focus on Sobel edge detection using contract-time anytime algorithm in CUDA. To reduce the computational complexity we implemented our proposed edge detection method using an NVIDIA GPU. In the experimental setup we have used NVIDIA GTX 550Ti GPU along with AMD FX8150 Processor and 8 GB RAM. Finally, we measure speedup as well as quick, moderate and final (3steps of contract) of our proposed parallel implemented model. Comparing with conventional serial CPU based edge detection we have experienced maximum 4X speedup of proposed implementation for 16 block dimension.

Chapter I

Introduction

1.1 Introduction

Edge detection from a color image is a very important and basically critical area in low level image processing. For performing high speed industrialized application based on image processing, edge detection is a mandatory thing to enhance work rate as well as accuracy. A number of researchers works on several edge detection algorithms and they give different responses and details to the different input images [1-7]. Edge detection quality has a great impact on realization of complex automated computer/machine vision systems [1]. Among them, the Sobel edge detection algorithm is much more popular than simple gradient operators due to its property to counteract the noise sensitivity and easier implementation process [2]. While using Sobel operator for GPU takes much less time than CPU. Again, the use of Interruption-Algorithm for image processing much less time efficient [3]. Moreover, in case of canny edge detection in GPU time process seems efficient but not enough for real time [4]. The use of anytime algorithm for GPU architecture makes it run faster in association with Dijkstra's algorithm [5, 10]. In addition, anytime algorithm seems much efficient when it is used for observing different tasks [6]. Interruptible Anytime Algorithm for image processing is much faster than normal image processing algorithms and also gives the privilege of getting output in different stage of time [7]. That is why, we are choosing contract-time anytime algorithm in coordination with Sobel operator for proposed parallel implementation.

1.2 Contribution Summary

The proposed edge detection of images was designed by passing through five main steps: Image input taking in CPU, image conversion (gray scale) using the Sobel Algorithm along with Contract-time Anytime Algorithm in GPU, essential Sobel Algorithm was used to detect the edge of the image. We also used Contract-time Anytime Algorithm here for faster processing in GPU Edge detection calculation of an image was done in CUDA environment. Finally, processed image and calculated time is showing in CPU.

1.3 Thesis Orientation

The rest of this thesis is organized as follows: Chapter II discusses on the premises of our thesis Parallel Computing and different components of parallel computing, anytime algorithm, its features and Sobel Edge Detection Technique. Chapter III discusses our proposed model. Chapter IV describes Experiment and result analysis on both CPU and CUDA Environment. Chapter V will tell the Application of this thesis. Finally, Chapter VI will conclude this paper with our future direction regarding this research.

Chapter II

Background Study

2.1 Parallel Computing - Definition

Parallel computing is operating on the idea that large problems can be divided into smaller ones, which are then solved at the same time. All of the calculations are carried out at the same time.

2.1.1 General-Purpose Parallel Computing Architecture

NVIDIA introduced CUDA™, a general purpose parallel computing architecture, a new parallel programming model and instruction set architecture. It leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems. It is more efficient way than on a CPU. CUDA comes with a software environment that allows developers to use C as a high-level programming language. Other languages, application programming interfaces, or directives-based approaches are supported (i.e.) FORTRAN, DirectCompute, OpenCL, OpenACC.

2.1.2 Kernels

CUDA C extends C by allowing the programmer to define C functions, called kernels. When called, are executed N times in parallel by N different CUDA threads. In regular C functions, it will be executed only once. A kernel is defined using the **__global__**. It specifies the number of CUDA threads that execute when a kernel is called using a new <<<…>>> execution configuration syntax. Each thread that executes, given a unique thread ID. It is accessible within the kernel through the built-in threadIdx variable. As an illustration, the following sample code adds two vectors A and B of size N and stores the result into vector C.

```
1.  // Kernel definition
2.  __global__ void VecAdd(float* A, float* B, float* C){
3.  int i = threadIdx.x;
4.  C[i] = A[i] + B[i];
5.  }

6.  int main() {
7.  ...     // Kernel invocation with N threads
8.  VecAdd<<<1, N>>>(A, B, C);
9.  ... }
```

Here, each of the N threads that execute VecAdd() performs one pair-wise addition.

2.1.3 Thread Hierarchy

ThreadIdx is a 3-component vector. The threads can be identified using a one-dimensional, two-dimensional, or three-dimensional thread index. It forms a one-dimensional, two-dimensional, or three-dimensional thread block respectively. This provides a way to invoke computation across the elements in a vector, matrix, or volume.

The index of a thread and its thread ID relate to each other in a straightforward way. For a one-dimensional block, they are the same. For a two-dimensional block of size (Dx, Dy), the thread ID of a thread of index (x, y) is (x + y Dx). For a three-dimensional block of size (Dx, Dy, Dz), the thread ID of a thread of index (x, y, z) is (x + y Dx + z Dx Dy).

As an example, the following code adds two matrices A and B of size N x N and stores the result into matrix C:

```
1.  // Kernel definition
2.  __global__ void MatAdd ( float A[N][N], float B[N][N],  float C[N][N]) {
3.  int i = threadIdx.x;
4.  int j = threadIdx.y;
5.  C[i][j] = A[i][j] + B[i][j];
6.  }
7.  int main() {
8.  ...     // Kernel invocation with one block of N * N * 1 threads
9.  int numBlocks = 1;
10. dim3 threadsPerBlock(N, N);
11. MatAdd<<<numBlocks, threadsPerBlock>>>(A, B, C);
12. ... }
```

There is a limit to the number of threads per block. On current GPUs, a thread block may contain up to 1024 threads. A kernel can be executed by multiple equally-shaped thread blocks. The total number of threads is equal to the number of threads per block times the number of blocks. Blocks are organized into a one-dimensional, two-dimensional, or three-dimensional grid of thread blocks as illustrated by Figure 1.

A thread block size of 16x16 (256 threads), is a common choice. Thread blocks are required to execute independently. This allows thread blocks to be scheduled in any order across any number of cores.

2.1.4 Memory Hierarchy

Each thread has private local memory. They have shared memory visible to all threads of the block and shared the same lifetime as the block. All threads have access to the same global memory.

There are also two additional read-only memory spaces accessible by all threads: the constant and texture memory spaces. The global, constant, and texture memory spaces are persistent across kernel launches by the same application. Figure 3 illustrates a basic memory hierarchy block.

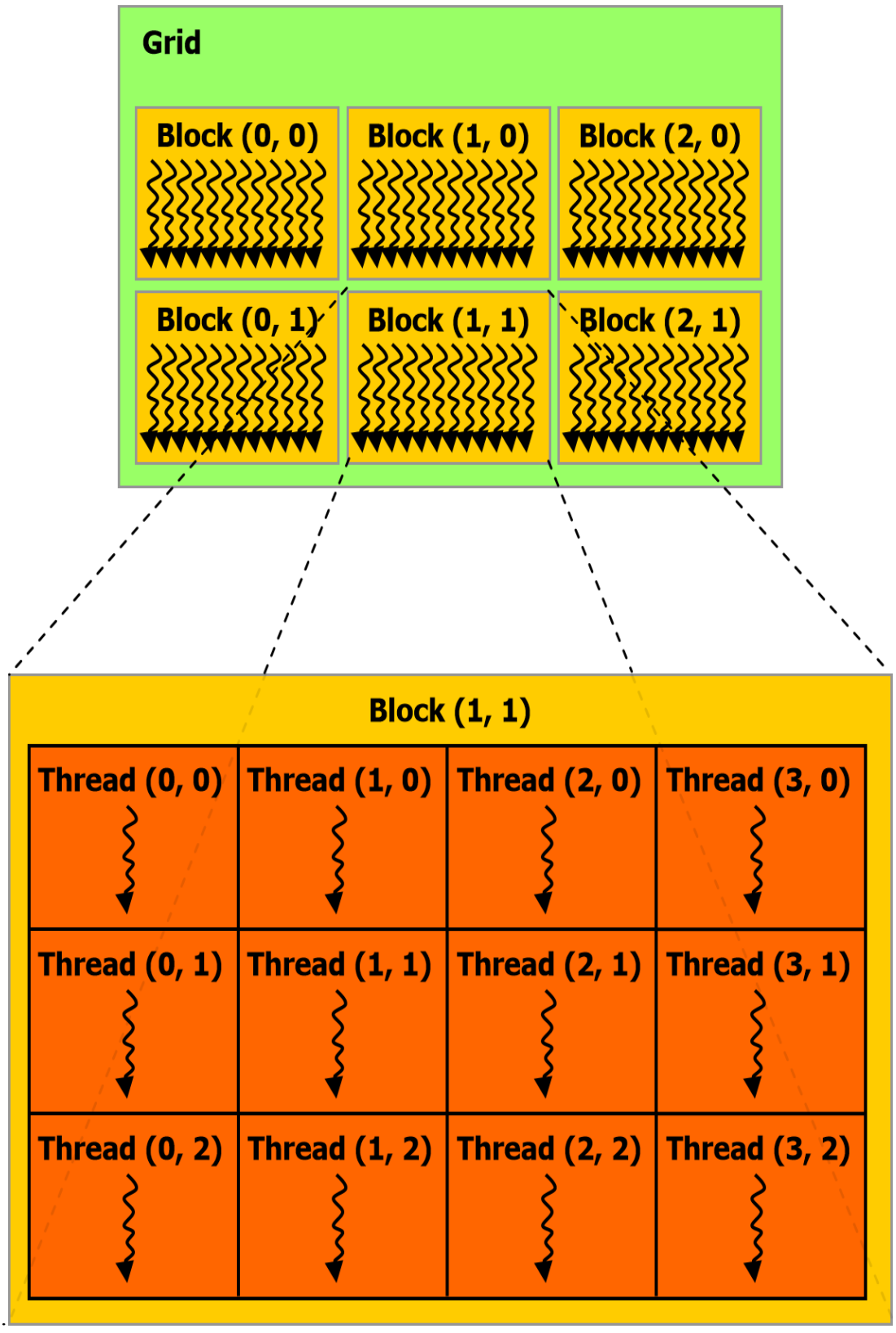


Figure 1. Grid of Thread Blocks

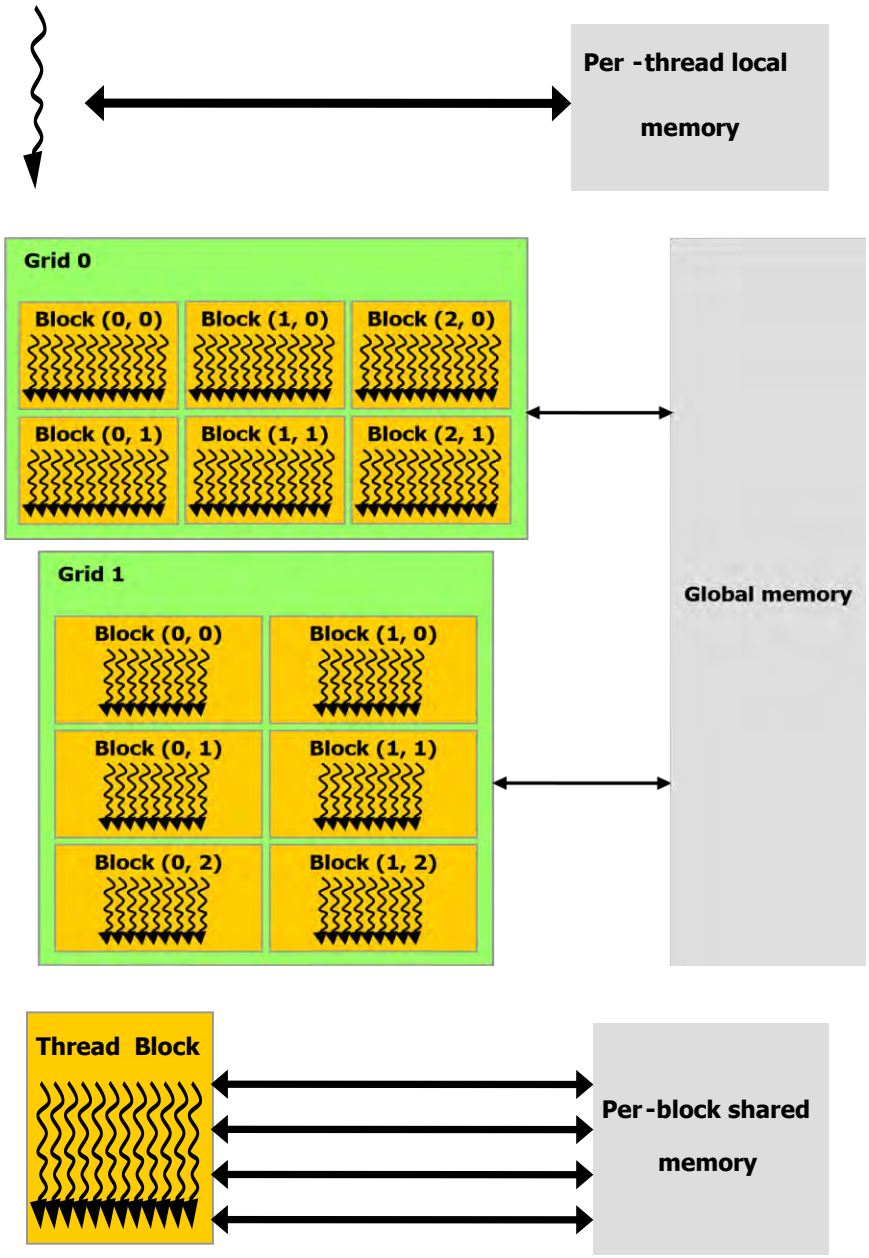


Figure 2. Memory Hierarchy

A multithreaded program is partitioned into blocks of threads. It executes independently from each other. A GPU with more multiprocessors will automatically execute the program in less time than a GPU with fewer multiprocessors.

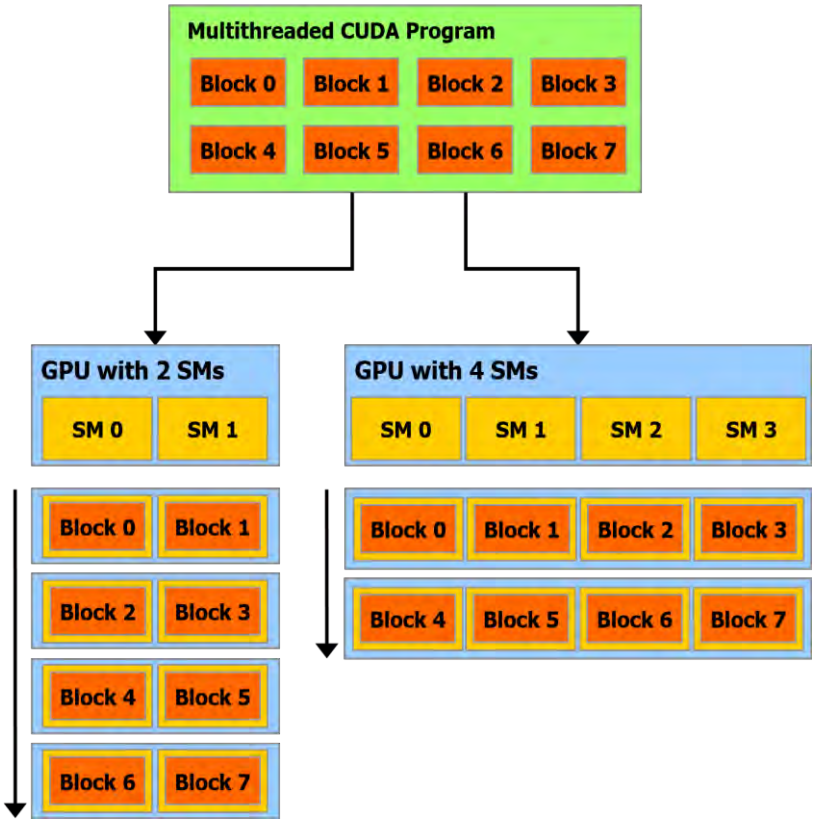


Figure 3. CUDA Multithreaded Programming Model

2.2 Anytime Algorithm

Anytime algorithm is a class of algorithm whose quality of results improves gradually as computation time increases. However, it offers trade-off between the resource consumption and output quality. The most wonderful feature of anytime algorithm is that it can be stopped anytime and an approximate result will be given based on the so far calculated data. It is the best part of anytime algorithm that other algorithm's cannot do. In case of any other algorithm, the program will crash if it is being try to stop in the middle of the calculation. They have to run the whole process to give a result.

Anytime algorithm is suited for the problem which has the trade-off between the processing (computation) time and the accuracy. The accuracy will be improved as the computation time

increases. The computation of anytime algorithm extends the traditional idea of computational procedures by allowing to return many possible approximate answers to any given inputs. The specialty of anytime algorithm is the use of well-defined quality measures to monitor the progress in problem solving. It also allocates the computational resources effectively.

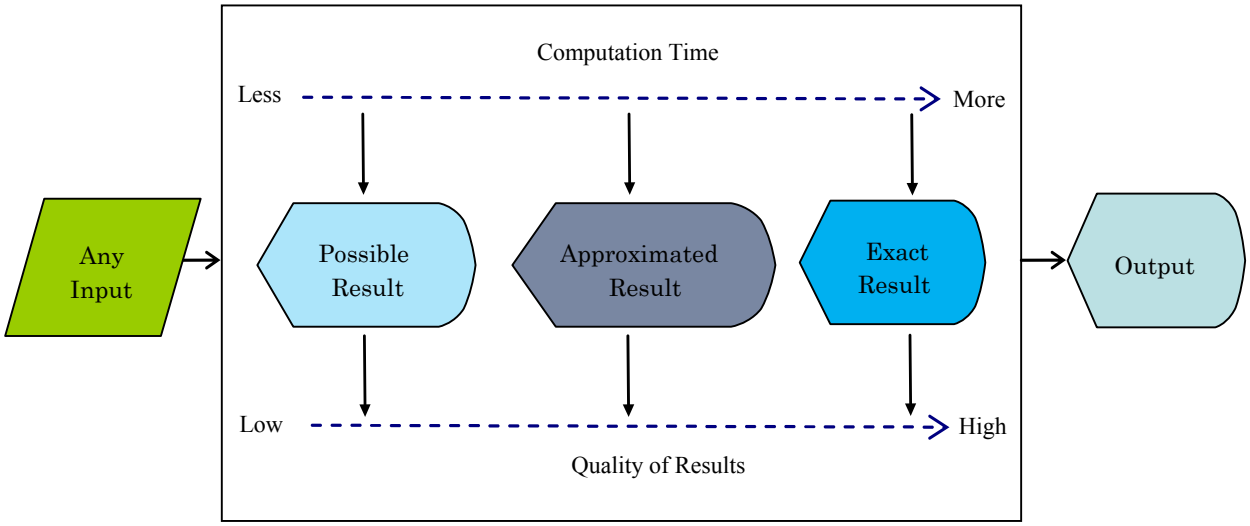


Figure 4. Workflow of anytime algorithm.

According to S. Zilberstein's paper [11] metrics of various kind can be used to get the quality of the result which is produced by anytime algorithm.

2.2.1 Certainty

This metrics reflects the certainty whether the result is correct. Certainty can be expressed using probabilities, certainty factors, or any other approaches.

2.2.2 Accuracy

This metric reflects the difference between the approximate result and the exact answer. Many anytime algorithms can provide a guarantee of a bound on the error, where the bound is reduced over time.

2.2.3 Specificity

This metric reflects the level of detail of the result. Anytime algorithm always produces the correct results, but the level of detail increases over time.

2.2.4 Properties of Anytime Algorithm

This section explains the properties of anytime algorithm according to S. Zilberstein's paper [11]. Anytime algorithm has the properties that satisfy the following features –

- **Measurable quality:** Quality of result can be defined exactly.
- **Recognizable quality:** Quality of an approximated result is easy to determine at intermediate processing time.
- **Monotonicity:** Quality of result is an increasing function of time and input quality.
- **Consistency:** Quality of result is connected with computation time and input quality.
- **Diminishing returns:** The solution's quality improves much larger than previous stages of computation and diminishes over time.
- **Interruptibility:** The algorithm can be stopped at any time and given some answer.
- **Preempt ability:** The algorithm can be stopped and started again at any time with minimal overhead.

2.2.5 Type of Anytime Algorithm

There are two types of anytime algorithm, (i.e.,) contract and interruptible

2.2.5.1 Contract

Contract based anytime algorithm provides the result in a given time frame. Although the algorithm can produce the results for any given time allocation, it might not be able to produce the required result. If the algorithm is interrupted before the expiration of the allocation, it might

not be able to produce the required result. In case of contract anytime time algorithm, there are few ways to be adopt. Among them, one is knowing the full time of the program and another is till which processes the program can give results.

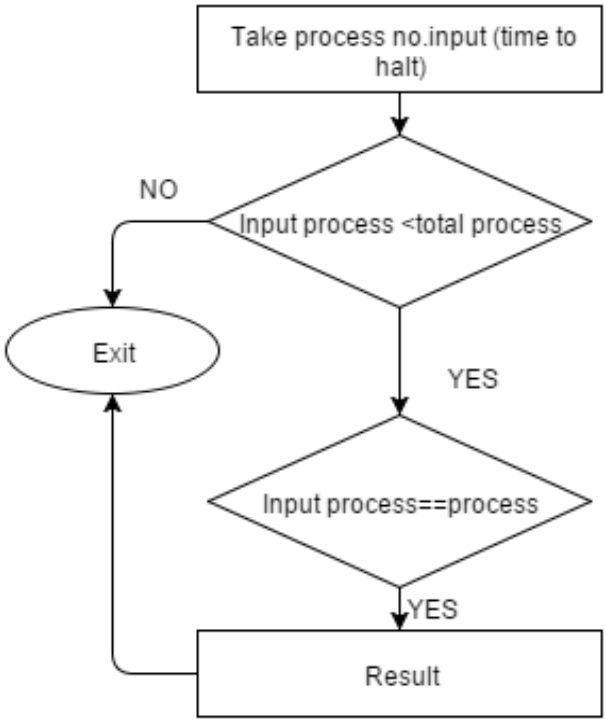


Figure 5. Model of contract-time Anytime Algorithm.

2.2.5.2 Interruptible

Anytime algorithm also produce an acceptable or required result based on the requirement when interrupted. The total run time of this algorithm is unknown. It can provide the output at any step of the result.

2.2.6 Differences Between Interruptible and Contract Anytime Algorithms

2.2.6.1 Interruptible

- Total execution time is unknown.
- Can be interrupted at any time.

- It is always contract algorithms.
- It is more complicated to construct than contract algorithm
- Flexible and widely applicable.

2.2.6.2 Contract

- Total execution time must be known in advance.
- Cannot be interrupted at any time, if it is interrupted between the execution time, it cannot provide the required result.
- It is not interruptible algorithms.
- Easier to construct.

2.2.7 Sample Algorithm of anytime algorithm

1. *Result* ← INITIALIZATION-STEP (*Input(x, y)*)
2. REGISTER-RESULT (*Result)*
3. x ← 0; y ← 0;
4. While (x < h)
5. {
    a. While (y < w)
    b. {
        i. *Output(x, y)* ← *Input(x, y);*
        ii. y ← y + 2;
    c. }
    d. SIGNAL (TERMINATION)
    e. HALT
6. }
7. w ← w/2;
8. h ← h/2;

2.2.8 Reduction Theorem

Reduction theorem allows the construction of contract anytime algorithms as an *intermediate* step, before the system is made interruptible. (S. Zilberstein, 1993)**.**

For any contract algorithm, an interruptible algorithm **B** can be constructed such that for any particular input **qB(4t) ≥ qA(t).**

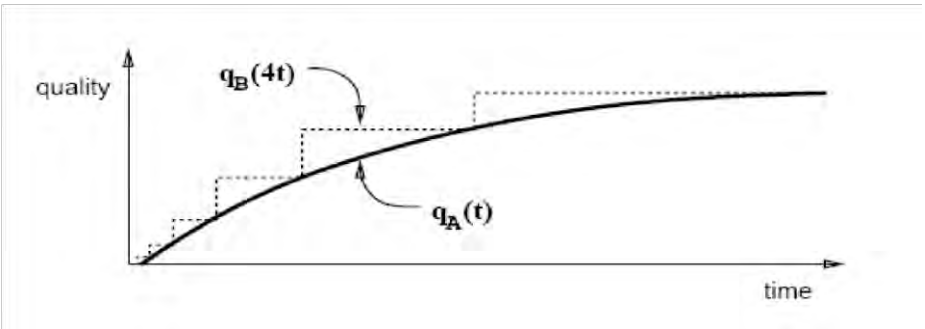


Figure 6. Performance profiles of interruptible and contract algorithms

2.3 Sobel Algorithm

The Sobel operator is widely used for edge detection in images. It is based on the computing the approximation of the gradient of image intensity function. The Sobel filter uses two 3 x 3 spatial masks to calculate the gradient. Two filters are Sx and Sy.

$$S_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

$$S_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

Figure 7. Sobel operator Convolution Kernel/Mask.

Sobel operator is based on convolving the image with a small, separable and integer valued filter in both horizontal and vertical directions. The local edge strength is defined as the gradient magnitude is given by equation 1. Equation 2 can give approximate magnitude for the computation, much faster to compute the gradient.

$$S=\sqrt{(S_x^2+S_y^2)} \quad (1)$$

$$|S| = |S_x| + |S_y| \quad (2)$$

The above mentioned scheme is for grayscale images. For color images (RGB color space), this scheme is applied separately for each color component. Final color edge map of color image is computed by using the edge maps of each color component.

Chapter III

Proposed Model

3.1 Proposed Model

Figure 8 shows the block diagram of proposed parallel implementation of CPU-GPU based edge detection method. To evaluate our proposed model have utilized different test images. First of all, we have taken the images as input. As the images are color images, we converted it into gray scale images. The process ran in GPU and we used contract-time anytime algorithm to make the conversion process faster, as depicted Figure 8.

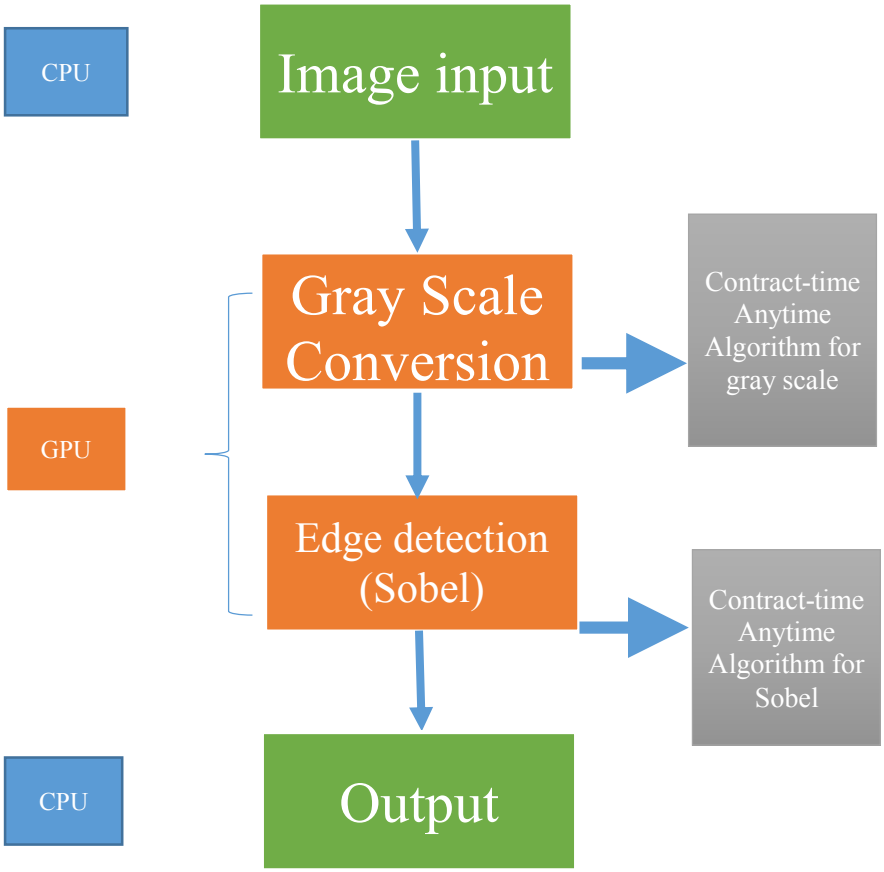


Figure 8. Proposed Model

After that, the edge detection process runs in GPU. Again, used contract-time anytime algorithm to detect the edges. We have calculated the time for Sobel operator in CUDA environment and took the time for processing and compared results. Our all the outputs shows in CPU that were calculated in GPU that is the primary aspiration of parallel implementation of Sobel operator along with any time algorithm. Figure 10 is a Sobel operator matrix that we used to calculate value for detecting edges in our CUDA environment. Here we used the general Sobel operator gradient matrix with CUDA. That detects edges and the time taken here is less than normal Sobel operator in CPU.

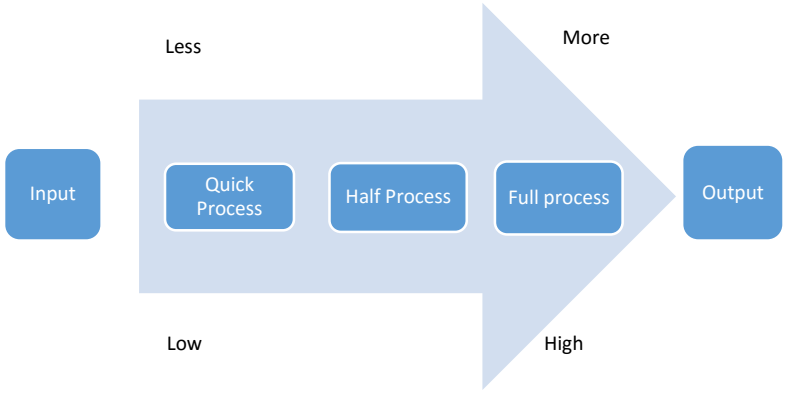


Figure 9. Contract-time Anytime Algorithm task processing.

In Figure 7 Sx represents horizontal convolution mask and followed by Sy represents vertical convolution mask. These convolution mask is being used for calculating the gradient.

Sobel operator 2D gradient based measurement is performed on an image. High spatial frequency which correspond to edges is mainly used to perform the measurement. For measurement we use Equation 1 which is the equation for gradient magnitude. In addition, Equation 2 can give approximate magnitude for the computation, which is much faster to compute the gradient. Figure 9 shows the task processing structure of any image input. From starting point it takes less time to compute but quality of processing is low. That is the quick process of contract-

time anytime algorithm. Gradually for half process and complete process of the program gives better output.

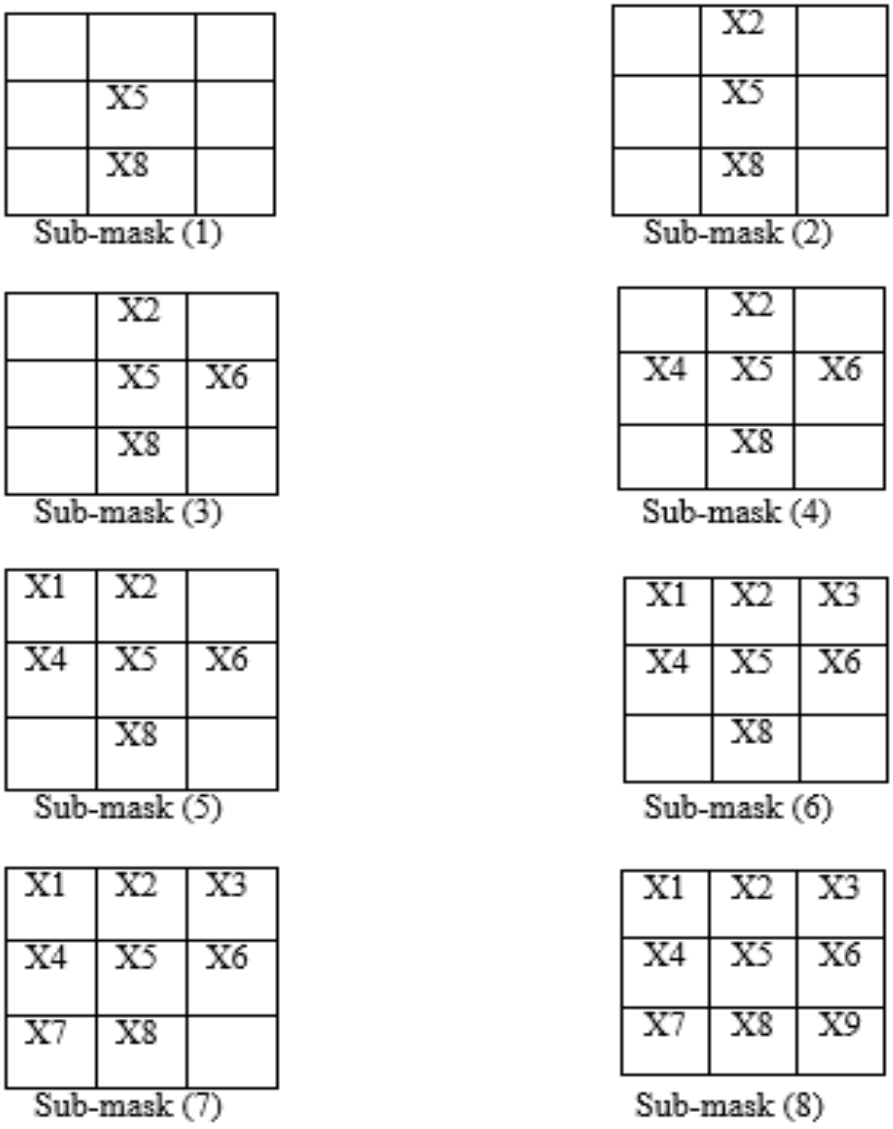


Figure 10. 3x3 sub-mask filters (1-8).

Figure 10 depicted how we divide every image into eight 3x3 sub-masks to use Sobel operator and use contract-time in different time period. We initially experiment only 3 contracts using these sub-masks and calculate process time in GPU and CPU system.

Chapter IV

Experiment and Result Analysis

4.1 Experimental Environment and Tools

In the experimental setup, we have used AMD FX 8150 CPU, 8 GB RAM with a GTX 550_{TI} GPU. We have also used Visual Studio 2013, CUDA ToolKit 7.5, OpenGL, Java 7, .NET C++, Eclipse Mars editor and Create a graph (online tool). Table I describes the detailed specification of GTX 550_{TI} GPU.

Table I. GTX 550ti GPU Engine Specs

Parameters	**Value**
CUDA Cores	192
Graphics Clock (MHz)	900
Processor Clock (MHz)	1800
Texture Fill Rate (billion/sec)	28.8
Processor Clock (MHz)	1800
Total amount of shared memory per block	49152 bytes
Maximum number of threads per block	1024
CUDA Driver Version / Runtime Version	7.5 / 7.5

4.2 Experimental Results of Parallel Sobel Detection

To evaluate our proposed edge detection model, we have used a 4096 image [Image I] and a 1920x1080 image [Image II]. Figure 11(a) is our sample image [Image I] for this experiments. Figure 11 (c) and (d) are the two outputs of input we present in Figure 11 (a).

Figure 11. (a) And (b) are the Sample Image I and II.
(c) And (d) are the output of Sobel 16 and 32 block.

Figure 11 (c) & (d) are two outputs for our experiment image. For large pixel images we capture the real image in 4096x4096 texture and then we send it to the GPU for gray scale conversion and edge detection. CPU execution time is the total time to read the image and printing the output. GPU time is the time for kernel that is calculating the total time to execute the kernel in GPU. We have taken block dimension 16 and 32 to calculate threads lowest threads for 16 block dimension is 256. And for 32 block dimension uses highest 1024 threads. For our experiment, we have calculated the blocks using Equation 3.

$$Block\ size = (X1\ ,\ Y1) \qquad (3)$$

$$X1 = W/Bd$$

$$Y1 = H/Bd$$

Where, width and height of image are from input image and block dimension is our default value. Width of image = W, Height of image = H, and Block dimension = Bd.

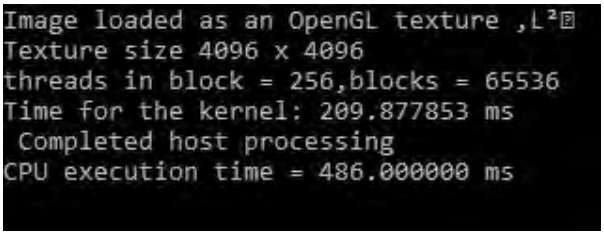


Figure 12. Console Result for 16 block dimension.

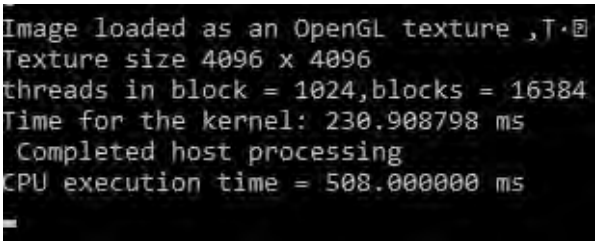


Figure 13. Console Result for 32 block dimension.

This program process 4K ultra HD images that is a normal conventional CPU programming Sobel edge detection cannot do. As we have also implemented Sobel for conventional CPU programming language with 3840 x 2400 image [Image 1] but it cannot read the image.

4.3 Experiment with Parallel Contract-time anytime and Sobel Detection

For experimenting our contract time algorithm with Sobel we have used a 1920x1080 image [Image II], which is presented in Figure 11(b).

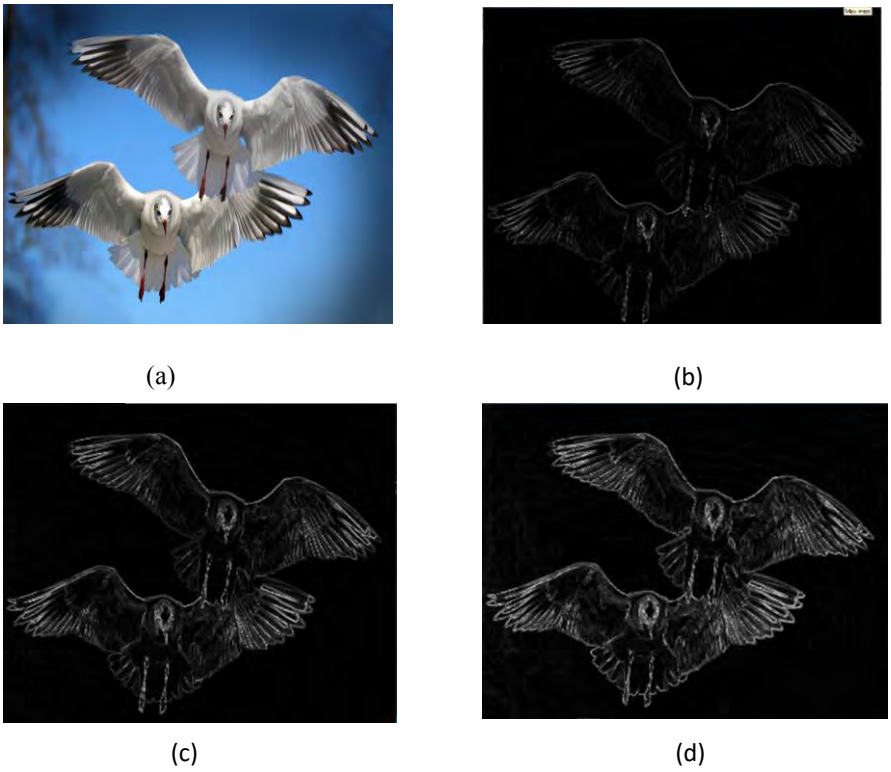
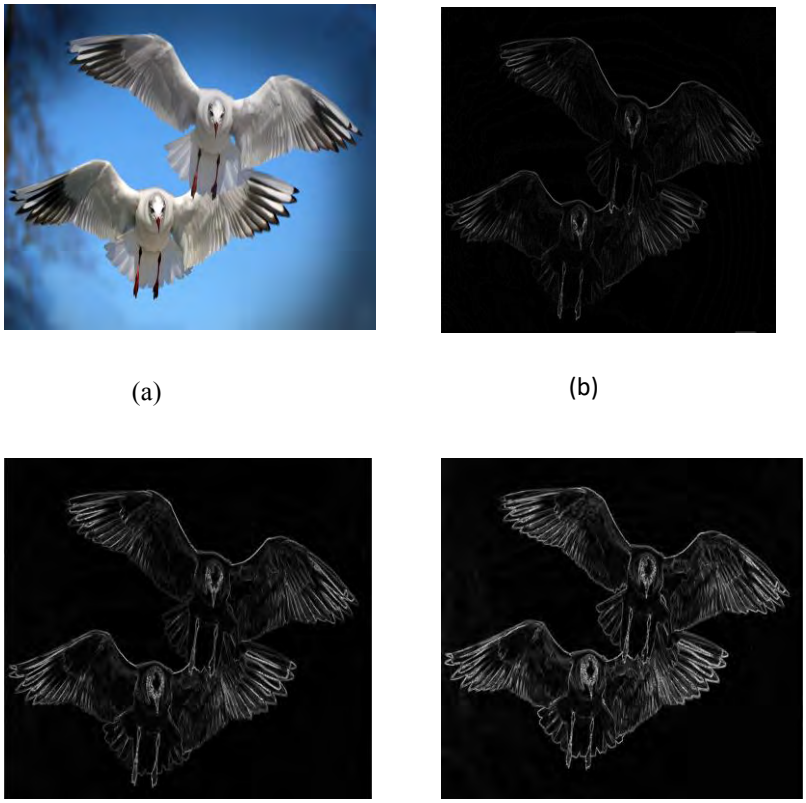


Figure 14. Three contract-time process test for 32 block dimension Test (b, c, and d).

Figure 14 shows that we have used 32 block dimension for test contract-time anytime algorithm. Where we have got different output and execution time from 3 contract of our program. Test 1 is the result of quick process. Test 2 is for half process and Test 3 is for Full process. Test 1 takes comparatively less time than test 3. Same goes for the image tested in 16 block dimension that's showed in Figure 15.

Figure 16 illustrates the sample outputs of our process. Where process of our algorithm is anytime algorithm output [Output 1], here user defines which process will be calculated. This output is for 16 block dimension half process. We have processed an image also in conventional CPU programming language to compare our results. Conventional CPU result is shown in Figure 16 (b).



(a) (b)

(c) (d)

Figure 15. Three contract-time process test for 16 block dimension test (b, c and d).

```
Insert
2 for quick process
4 for half process
6 for full preocess
4
Image loaded as an OpenGL texture ,▯·▯
Texture size 2048 x 1024
threads in block = 1024,blocks = 2048
Time for the kernel: 27.027777 ms
 Completed host processing
CPU execution time = 108.000000 ms
```

```
Sobel Enhancement
Output file name: >> output1920_output
Output file extension : >> jpg
Done > Sobel
480 ms
```

Figure 16. Sample outputs.

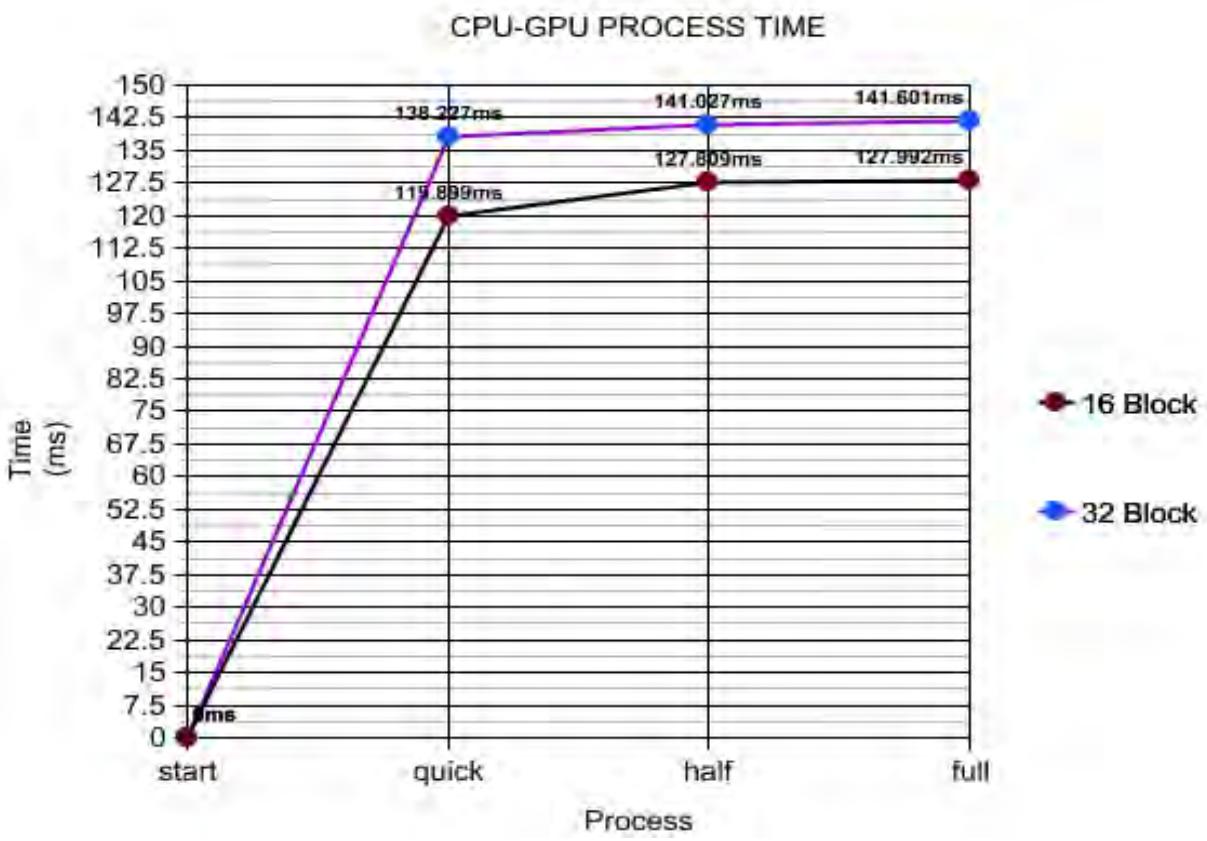


Figure 17. Process versus Time for 16 and 32 block dimension.

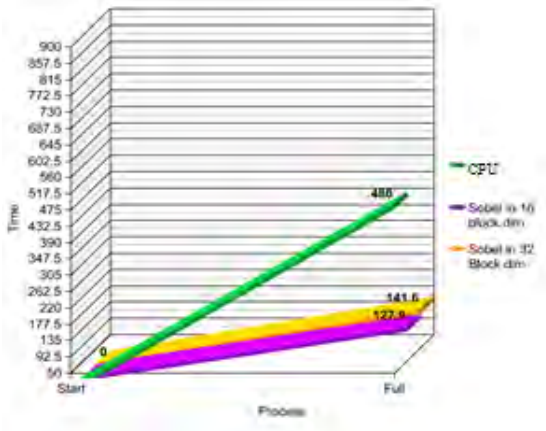


Figure 18. Time Comparison of 1920 x 1024 Input Image.

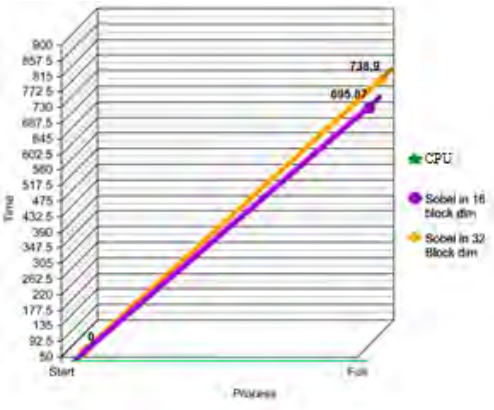


Figure 19. Time Comparison of 4096 x 4096 Input Image.

Table II. Result Comparison

Process		CPU – Gpu Time (ms)	Threads per block	Block size	CPU Execution time (ms)	Speedup (against CPU)
16 block Dimension	quick	119.899	256	8192	480	4.003x
	Half	127.809				3.7x
	full	127.992				3.75x
32 block dimension	quick	138.227	1024	2048	480	3.47x
	half	141.027				3.4x
	full	141.601				3.38x

Table II represents the comparison of our program with conventional CPU programming. Comparing with the CPU program we have calculated speedup of our program and for 16 block low it is 4.003x and for high quality edge detection it is 3.75x. Figure 17 graph represents the detailed comparison between 16 and 32 block dimension with respect to execution time.

Chapter V

Application

5.1 Application

Edge detection is a fundamental feature of Image Processing. It is the basic step of image analysis. The purpose of edge detection is to discover the information about the shapes and the reflectance or transmittance in an image. We can apply edge detection in many sectors.

Our first thought of applying this technology in game of cricket. In Cricket, there are some contradiction in giving out. To determine, whether it was out or not, they took help of snicko technology (sound wave based). In this case, we can apply our proposed model instead of snicko technology.

Figure 20. Sample Image III

Figure 21. Sample Image IV

Figure 20 and Figure 21 are the sample image III and sample image IV to use in our application. Figure 22 and Figure 23 are the same output of III and sample image IV are the output image. It is clearly visible that whether the ball touched the bat or not.

Figure 22. Output Image III

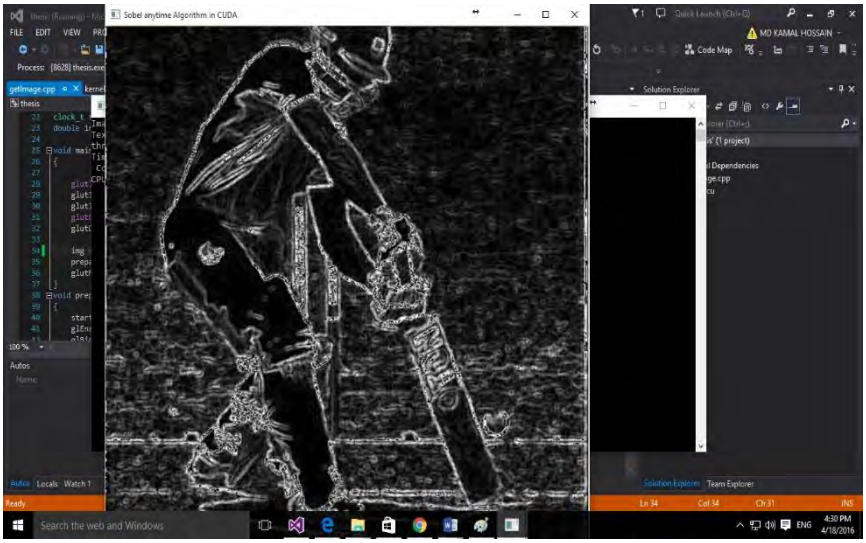


Figure 23. Output Image IV

We can also implement our edge detection methods on medical science (detect the edge of lungs CT image), shape and object recognition, traffic management and line detection from blurry image.

Chapter VI

Conclusion and Future Works

6.1 Conclusion

This paper presented a new parallel edge detection method using Sobel and Contract Anytime Algorithm. As a parallel platform we utilize an NVIDIA GTX GPU and 8 Core CPU. For sample test images, we calculate the execution time of proposed CPU-GPU parallel method and conventional CPU based algorithm. In addition, by varying thread and block sizes, we observed the effect of computation time. Experimental results show that the proposed parallel implementation exhibits above 4X speedup over the conventional serial implementation.

6.2 Future Works & Limitations

Among the various difficulties and limitations we have faced, the most important one is the Visual Studio 2013 extension error with CUDA C environment. Moreover, CUDA C 7.0 toolkit did not permit some of our coding technique. So, we have to wait till CUDA C toolkit 7.5. In addition, due to lack of Kepler Architecture GPU, we were not able to implement the interruptible anytime algorithm. Normal GTX architecture that we used and are available to us does not support the techniques required.

We are interested to extend this thesis to compute image in real time using contract-based anytime algorithm. Furthermore, we like to implement interruptible anytime algorithm. Again, we also want to implement the above method of computing edge detection in video formats as well.

References

[1] M. B. Ahmad and T. S. Choi, “Local threshold and boolean function based edge detection,” IEEE Transactions on Consumer Electronics, vol. 45, no. 3, pp. 674–679, 1999.

[2] T. A. Abbasi and M. U. Abbasi, “A novel FPGA-based architecture for Sobel edge detection operator,”International Journal of Electronics, vol. 94, no. 9, pp. 889–896, 2007.

[3] W. Kywe, D. Fujiwara, and K. Murakami., “Scheduling of Image Processing Using Anytime Algorithm for Real-time System,” Pattern Recognition, 2006. ICPR 2006. 18th International Conference on, vol.3.2006.

[4] Ogawa, Kohei, Y. Ito, and K. Nakano. “Efficient Canny Edge Detection Using a GPU.” First International Conference on Networking and Computing. 2010.

[5] M. Rahul, and A. A. Saba. “Anytime Algorithms for GPU Architectures.” 2011 IEEE 32nd Real-Time Systems Symposium, 2011.

[6] Baxter, J W, J. Hargreaves, N. Hawes, and R. Stolkin. “Controlling Anytime Scheduling of Observation Tasks.” Research and Development in Intelligent Systems XXIX: 219-24, Oct 2012.

[7] W. Kywe, D. Fujiwara, and K. Murakami, “An Approach to Linear Spatial Filtering Method based on Anytime Algorithm for Real-time Image Processing,” 18th International Conference on Pattern Recognition (ICPR'06), vol. 4, no. 12, 2012.

[8] Rostov Kremlin Available from:
<https://wallpaperscraft.com/wallpaper/rostov_velikij_kreml_rossiya_kh ram_103672>

[9] Gulls Available from:
https://wallpaperscraft.com/download/gulls_birds_flying_flapping_1064 66/1600x900

[10] J. Uddin, E. Oyekanlu, C.H. Kim, and J. M. kim, "High Performance Computing for Large Graphs of Internet Applications using GPU," International Journal of Multimedia and Ubiquitous Engineering, Vol. 9, No. 3, pp. 269-280, 2014.

[11] S. Zilberstein, ‚Using Anytime Algorithms in Intelligent Systems', AI Magazine, vol. 17, no. 3, pp. 73-83, (1996).

[12] S. Zilberstein and S. J. Russell. In S. Natarajan (Ed.), ‚Approximate Reasoning Using Anytime Algorithms, Imprecise and Approximate Computation', Kluwer Academic Publishers, (1995).

[13] J. Grass and S. Zilberstein. In M. Pittarelli (Ed.), ‚Anytime Algorithm Development Tools', SIGART Bulletin Special Issue on Anytime Algorithms and Deliberation Scheduling, 7(2):20-27, (1996).

[14] Harley R. Myler and Arthur R. Weeks, ‚The handbook of image processing algorithms in C', Prentice-Hall PTR, 1993.

[15] R.C. Gonzalez and R.E. Woods, ―Digital image processing,‖ Prentice Hall, 2nd Edition, 2002.

[16] R.K. Sandhu, P.S. Maan, ―A spatial-domain filter for digital image De-noising used for Real time applications‖, International Journal of Computer Science and Technology, IJCST Vol. 2, Issue 3, September 2011.

[17] J.R. Parker, ‚Algorithms for image processing and computer vision', John Wiley & Sons, Inc. U.S.A, 1997.

[18] I. Pitas, ‚Digital Image Processing Algorithms and Applications', John Wiley & Sons, Inc. U.S.A, 2000.

[19] A. K. Jain, ‚Fundamentals of Digital Image Processing', Prentice-Hall, Inc. U.S.A, 1989.

[20] J. Cheng, M. Grossman, T. McKercher, Professional CUDA C programming, Wrox, a Wiley brand, INpolis, IN, 2014.

[21] I. Hatzilygeroudis, V. Palade, Combinations of intelligent methods and applications: proceedings of the 3rd International Workshop, CIMA 2012, Montpellier, France, August 2012, Springer, Heidelberg, 2013.

[22] J. Sanders, E. Kandrot, CUDA by example: an introduction to general-purpose GPU programming, Addison-Wesley, Upper Saddle River, NJ, 2011.

[23] T. Masters, Deep belief nets in C and CUDA C, n.d.

[24] http://www.profc.udec.cl

[25] http://www.eng.iastate.edu/ee528/sonkamaterial/chapter_1.htm

[26] http://www.eng.iastate.edu/ee528/sonkamaterial/chapter_2.htm#Image%20functions

[27] http://en.wikipedia.org/wiki/

[28] http://www.nvidia.com/object/what-is-gpu-computing.html

[29] http://www.nvidia.com/object/cuda_home_new.html

[30] http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz465OFzRvq

[31] http://www.nvidia.co.kr/content/cudazone/download/showcase/kr/Tutorial-DKIRK.pdf

[32] https://blogs.nvidia.com/blog/2013/05/03/trump-card-why-a-pro-poker-player-bet-on-cuda

[33] http://docs.nvidia.com/cuda/

[34] http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2012-02-20/07-intro_to_cuda_c.pdf

[35] https://devblogs.nvidia.com/parallelforall/easy-introduction-cuda-c-and-c

[37] https://devblogs.nvidia.com/parallelforall/using-shared-memory-cuda-cc

[38] http://www.pgroup.com/resources/cuda-x86.htm

[39] http://developer.download.nvidia.com/compute/developertrainingmaterials/presentations/cuda_language/Introduction_to_CUDA_C.pptx

[40] http://cs.unc.edu/~prins/Classes/633/Readings/CUDA_C_Programming_Guide_4.2.pdf

[41] http://www.codeproject.com/Articles/202792/Using-Cudafy-for-GPGPU-Programming-in-NET

[42] http://developer.download.nvidia.com/books/cuda-by-example/cuda-by-example-sample.pdf

[43] Create A Graph (Online tool) - https://nces.ed.gov/nceskids/createagraph/