Parallel Edge Detection using Sobel Algorithm with Contract-time Anytime Algorithm in CUDA

Md Kamal Hossain, Md Assaduzzaman Ashique, Md Asif Ibtehaz, and Jia Uddin
Computer Science and Engineering Department, BRAC University
Dhaka 1212, Bangladesh.
E-mail: shajal16@gmail.com, ashique12301017@gmail.com, ibtehaz.shawon@gmail.com, jia.uddin@bracu.ac.bd

Abstract - Edge detection is a considerably important factor in image or video processing. Detection of edges plays a significant role in image segmentation, data compression, well matching, and image reconstruction. Among several edge detection approaches 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 CUDA. In the experimental setup we have used NVIDIA GTX 550Ti GPU along with AMD FX8150 Processor and 8 GB RAM. Finally, we measure speedup using 3 steps of contract-time anytime of our proposed parallel implementation model. Comparing with conventional serial CPU based edge detection we have experienced maximum 4X speedup of proposed implementation for 16 block dimension.

Keywords: Edge detection; CUDA; Anytime algorithm; Parallel Computing; Sobel; NVIDIA

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, 8]. 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 co-ordination with Sobel operator for proposed parallel implementation.

The rest of this paper is organized as follows: Section II describes the detailed of Anytime Algorithm. Section III presents details about the proposed Sobel edge detection using Contract-time anytime algorithm in an NVIDIA GPU in CUDA, and Section IV contains experimental results of analysis and comparison. Finally, Section V concludes this paper.

Figure 1. Model of contract-time Anytime Algorithm.

2 Contract-time anytime algorithm

Anytime algorithm is a kind of algorithm that searches for better result instead of calculating the final result [3, 5, and 7]. There are mainly two types of anytime algorithm: Interruption and Contract-time. Interruption anytime is the algorithm that continues running and can be stopped anytime to get the final result. For contract-time anytime it is a little different, as final result is generated based on user input of
processes or time. Figure 1 is the presentation of contract-time anytime algorithm, which we used in our proposed edge detection algorithm.

3 Proposed model

Figure 2 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 interruptive anytime algorithm to make the conversion process faster, as depicted Figure 2.

After that, the edge detection process runs in GPU. Again, used interruptive 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.

\[ S = \sqrt{S_x^2 + S_y^2} \]  
\[ |S| = |S_x| + |S_y| \]

Where \( S_x \) represents horizontal convolution mask and followed by \( S_y \) 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 3 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 conventional CPU programming.

Figure 3. Sobel operator Convolution Kernel/Mask.

Figure 4. Contract-time Anytime Algorithm task processing.

Figure 5. 3x3 sub-mask filters (1-8).
Figure 4 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 5 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.

4 Experimental results and analysis

In the experimental setup, we have used AMD FX 8150 CPU, 8 GB RAM with a GTX 550ti GPU, which specification is given in the Table I.

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

4.1 Experimental Results of Parallel Sobel Detection

To evaluate our proposed edge detection model, we have used a 3840x2400 image [Image 1] and a 1920x1080 image [Image2]. Figure 6 is our sample image [Image 1] for this experiments. Figure 7 and Figure 8 are the two outputs of input we present in Figure 6.

Figure 9 and Figure 10 are two outputs for our experiment image, Image I. 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. Maximum amount of threads for 16 block dimension is 256 and for 32 block dimension is 1024 threads. For our experiment, we have calculated the blocks using Equation 3.

\[
\text{Block Size} = (X_1, Y_1) \hspace{1cm} (3)
\]

\[
X_1 = \frac{W}{B_d} \\
Y_1 = \frac{H}{B_d}
\]

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 = \(B_d\).
A conventional CPU based Sobel edge detection is able to compute the edge detection for our sample Image, 1920 x 1080 pixel image. However, it is unable to compute a 3840 x 2400 pixel image. Figure 11 depicts the information related to conventional CPU based implementation.

**Sobel Enhancement**

```java
Exception in thread "main" java.lang.IllegalArgumentException: Pixel
 at java.awt.image.PixelInterleavedSampleModel.<init>(PixelInterleavedSampleModel.java:145)
 at Main.getMain(Main.java:87)
```

Figure 11. Conventional CPU Error Output.

### 4.2 Experiment with Parallel Contract-time Anytime and Sobel Detection

For experimenting our contract time algorithm with Sobel, we have used a 1920x1080 image [Image 2], which is presented in Figure 12, in two different 16 and 32 block dimensions.

![Sample image 2](image)

**Figure 12. Sample image 2.**

Figure 13 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 14. Here, we have changed our block dimension to 16 to measure the computation time when we are using less amount of blocks.

![Test Image 2](image) ![Test 1(quick)](image) ![Test 2(half)](image) ![Test 3(full)](image)

**Figure 13. Three contract-time process test for 32 block dimension (b, c and d).**

![Test Image 2](image) ![Test 1(quick)](image) ![Test 2(half)](image) ![Test 3(full)](image)

**Figure 14. Three contract-time process test for 16 block dimension test (b, c and d).**

Figure 15 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. Output 2 is a sample output for conventional CPU based program.

![Output 1] Anytime algorithm output

![Output 2] Conventional CPU output

**Figure 15. Sample output of parallel anytime and conventional CPU**
Figure 16. Process versus Time for 16 and 32 block dimension.

Figure 16 graph represents the detailed comparison between 16 and 32 block dimension with respect to execution time.

Figure 17. Process versus Time of 1920 x 1024 Input Image.

Figure 17 presents the comparison graph of conventional CPU based program and both 16 and 32 block dimension in GPU. Where conventional CPU based program took 480 ms and for our sobel in parallel process its 141.6ms and 127.9ms for 32 block dimension and 16 block dimension to compute a 1920 x 1024 pixel image.

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.

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU-GPU time (ms)</th>
<th>Threads in Block</th>
<th>Block size</th>
<th>CPU Execution Time (ms)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 block dimension</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quick</td>
<td>119.899</td>
<td>256</td>
<td>8192</td>
<td></td>
<td>4.003x</td>
</tr>
<tr>
<td>Half</td>
<td>127.809</td>
<td></td>
<td></td>
<td></td>
<td>3.7x</td>
</tr>
<tr>
<td>Full</td>
<td>127.992</td>
<td></td>
<td></td>
<td></td>
<td>3.75x</td>
</tr>
<tr>
<td>32 block dimension</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quick</td>
<td>138.227</td>
<td>1024</td>
<td>2048</td>
<td></td>
<td>3.47x</td>
</tr>
<tr>
<td>Half</td>
<td>141.027</td>
<td></td>
<td></td>
<td></td>
<td>3.4x</td>
</tr>
<tr>
<td>Full</td>
<td>141.601</td>
<td></td>
<td></td>
<td></td>
<td>3.38x</td>
</tr>
</tbody>
</table>

5 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 References

International Conference on Pattern Recognition (ICPR’06), vol. 4, no. 12, 2012.

