Image Processing in Windows: Acceleration in GPGPU

Prawar Poudel, *M.Sc. Computer Engineering, The University of Alabama in Huntsville*

***Abstract*—Digital Image Processing has gained certain amount of ubiquity in present world. Availability of computing devices with extreme computing power at everybody's disposal has made taking photos and manipulating them a luxury. As a learning opportunity, we have implemented Digital Processing Operation in Images as an application in Windows platform. Acceleration of the algorithms implemented is done using NVIDIA GeForce 980 GPU using CUDA/C. Here we obtain speedup of as high as 32 times in certain operations.**

***Index Terms*—Digital Images, Image Analysis, Image Denoising, Image Edge Detection**

I. Introduction

DIGITAL image processing is the process of operating on image using computers. When the concept of image operation was first conceived, the operation was intended for some specialized purposes only. This was because the cost of operating was high considering the lack of availability of computing equipment at the time. Military, Medical, Remote sensing were some of the areas that used image processing. [1]

With the Moore's Law being followed by the computing machines, and reducing size of electronic equipment and increasing affordability has brought these electronic equipment in hands of most of the people. With this increase in penetration of the devices, more features are added to them, and taking and storing images is one of them.

GPUs are available in most of the processors today. Raspberry Pi of as low as 5$ also has Broadcom VideoCore GPU running at 400Mhz [2]. It is only matter of time that these GPU are made general purpose. So as a good future scope we implemented some algorithms of basic image processing in general purpose GPU that is available at our disposal.

Following sections introduce the basic operation and other technical aspects of the project. Section II details the digital image processing and image representation in general. Section III details the algorithms that are implemented. Section IV introduces implementation approach. Section V details the analysis results. Section VI details the help received from review and feedback. Section VII and Section VIII talk about the platform targeted and conclusion respectively. Section IX details the final section which is future enhancements, and more stretching of the project.

II. Image Representation and Digital Image Processing

Computers are extensively used for operating in images. Numbers are what computers understand. So intake of images from real world, which is analog, and converting them to numbers for computer understandable forms requires some operation which certainly does not fall in our scope of this work. But as a shake of necessity we will be dealing with brief introduction.

A. Binary Images

Image is divided into smallest possible representable form called Pixel. These pixels each is given certain value for the computer to be able to identify or represent it inside itself. In case of binary image, the value for each pixel is either '1' or '0'. The higher value represents the higher contrast while the lower value represents the lower contrast. So there is only two level in the entire image.

Fig I and Fig II show the binary image pixel representation and the actual image.

1	1	0	1	1	1	0	1
1	1	0	1	0	1	0	1
1	1	1	1	0	0	0	1
0	0	0	0	0	0	0	1
1	1	1	1	0	1	0	1
0	0	0	1	0	1	0	1
1	1	0	1	0	0	0	1
1	1	0	1	0	1	1	1

Fig. 1 Binary Image Pixel values

Fig. 2 Binary Image [3]

B. Gray Scale Images

Advancement took a step further in terms of representing images and 256 possible values were devised to represent each pixel value. This means that each pixel could have any value from 0 to 255. This increased the range of possible colors that a particular position in an image could mimic. Since the value range was 0-255, these could easily be represented by an 8-bit variable.

Fig 3 and Fig 4 shows the gray scale pixel values in an arbitrary image and the possible color ranges that it could represent.

Submitted 08/08/2017. "This work was done as project for the course CPE 613 General Purpose Graphics Processing Unit Processing."

The author is a graduate student at the University of Alabama in Huntsville. He can be contacted at pp0030@uah.edu.

88	82	84	88	85	83	80	93	102
88	80	78	80	80	78	73	94	100
85	79	80	78	77	74	65	91	99
38	35	40	35	39	74	77	70	65
20	25	23	28	37	69	64	60	57
22	26	22	28	40	65	64	59	34
24	28	24	30	37	60	58	56	66
21	22	23	27	38	60	67	65	67
23	22	22	25	38	59	64	67	66

Fig. 3 Gray Scale Pixel values

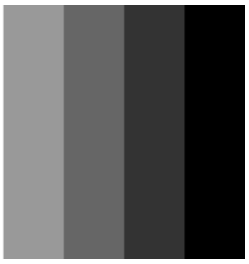

Fig. 4 Sample Gray Scale Colors

C. *RGBA Channels*

Color images are represented using composite of three layers of individual color components. There three color components are Red, Green and Blue colors. The forth channels is Alpha channels which defines the opacity of the image. Advancement took a step further in terms of representing images and 256 possible values were devised to represent each pixel value. This means that each pixel could have any value from 0 to 255. This increased the complexity of images and also increased the number of colors possible that could be represented in each pixel.

Fig 5 and Fig 6 shows the three composite layers that make up the final image.

Fig. 5. Three composite layers or channels that comprise the image representation

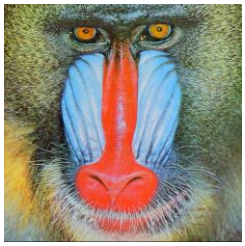

Fig. 6. Final Image made up of three channels from Fig 5 [4]

Digital Operation in image means reading these channels into a computer or a computing machine and operating on them. We can get a hold of individual channels and operate to create the final image.

III. ALGORITHMS IMPLEMENTED

The complexity of the image operation depends on the kind of algorithm used, and the operation itself. If the operation on image is simple operation such as RGB to Gray Scale Conversion, the operation is simple, while DeNoising of the image would be somewhat complex than the DeNoising, but is still not a complex cooperation. Edge Detection of an image using filter involves complex operation of convolution but if the mask kernel is of fixed size, the operation is simple enough to be operated.

A. *Mean Filtering*

Mean filtering is the operation that takes a pixel and performs the stencil operation among its neighbors. The weight provided to the pixels is same, so there is no added influence on the resulting pixel based on the original pixel. Mean filtering is basically used to remove the detail and also blurs the image.

Convolution with the following mask kernel results in the Mean filtered image output.

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Fig. 7 Mean Filter Mask Kernel [6]

B. *Gaussian Blur*

This is also another operator that blurs the image and removes the concentrated artifacts in the image so that detail in the image is removed. But since this provides some weights to the pixels based on the neighborhood, this is more effective than the Mean filtering operation.

Convolving with the following mask kernel results in the Gaussian Blurred Image as in Fig 8.

1/16	2/16	1/16
2/16	4/16	2/16
1/16	2/16	1/16

Fig. 8 Gaussian Filter Mask Kernel

C. *RGB To Gray Scale Operation*

RGB to Gray Scale conversion is the simplest of the operation that I have implemented. This operation is basically the point to point operation where 3 channels of the RGB is used to calculate the Gray scale equivalent of the image.

The basic operation will be to take the average among the three channels and assign the value as the Gray scale output. But because that these colors have in our eye; certain formula is devised for this purpose. The equation is shown in Fig 9

$$Gr = R^* 0.299 + G^*0.587 + B^*0.114$$

Fig. 9. RGB To Gray Scale Conversion Formula [5]

D. *Edge Detection*

Edges in images are determined by finding the difference in the intensity value of any pixels with that of its neighbor. It again implies the application of stencil operation and thus involves convolution. The filter used in our case are all 3x3 filters.

1) *Sobel Edge Detection Operator*

Sobel Edge Detection operator is the edge detection operator that calculates the difference of an image pixel with its neighbor in both x- and y- direction. Here in either of the cases the immediate neighbor in the concerned direction is given some

weightage than the diagonal neighbor. Fig 10 and Fig 11 shows the Sobel Filter for x- and y- direction respectively.

-1	0	1
-2	0	2
-1	0	1

Fig. 10 Sobel Dx-filter for edge detection [7]

1	2	1
0	0	0
-1	-2	-1

Fig. 11 Sobel Dy-filter for edge detection [7]

Convolving the image with either of these filters would give the edge to some extent, but there is another approach. This approach uses Pythagoras theorem to find the vector sum of these two directions.

2) Prewitt Edge Detection Operator

Prewitt operator is similar to Sobel operator. It differs only from the fact that in Sobel for the immediate non-diagonal neighbor in the concerned direction, certain weights is given, but in Prewitt Operator all the neighbors are treated equally, meaning none is given higher priority.

-1	0	1
-1	0	1
-1	0	1

Fig. 12 Prewitt Dx-filter for edge detection [7]

1	1	1
0	0	0
-1	-1	-1

Fig. 13 Prewitt Dy-filter for edge detection [7]

We can also perform the vector sum of these to find the final and better result.

3) Laplace Edge Detection Operator

Laplace edge detector operator taken into consideration for the immediate neighbor, so there is no directional neighborhood relation as in previous operator considered.

Convolving with matrix as shown in Fig 14 would result in the edge detection.

E. Contrast Enhancement

0	1	0
1	-4	1
0	1	0

Fig. 14. Laplace Edge Detection Kernel [8]

Images taken do not always have the contrast or brightness value in the range desired. Sometimes the contrast in concentrated on the darker region, while sometimes image pixels are concentrated toward the brighter region.

Here we will be dealing with three such algorithm that follow in the following subsections.

1) Linear Stretching

Linear stretching operation of pixel value maps the original pixels to final pixels on one-to-one basis. The pixel value concentrated in the middle of the spectrum is stretched so that they are mapped to wider region in linear fashion.

Fig 15 shows the formula for implementation of the linear stretching operation image.

2) Power Law

Power law is used for either really dark images or too bright

$$pix_{final,i} = 255*(pix_{initial,i}-pix_{min})/(pix_{max}-pix_{min})$$

Fig. 15. Linear Stretching Formula

images. This law is based on a power equation which is shown in figure 16.

Power Law is used to either stretch the dark region or expand

$$Pix_{final,i} = c * pow(Pix_{initial,i}, r)$$

Fig. 16. Power Formula

bright region. For r>1, the bright region is expanded while for r<1, the dark region is stretched.

3) Histogram Equalization

It is one of the technique that stretches the histogram of the pixel intensity from a concentrated region in image and distributes it equally. In other words, based on the number of the images pixels at certain value, the final pixel intensity value is going to change.

In this operation, first histogram of all the pixel intensity is calculated, and the minimum non-zero cumulative histogram is mapped to 0 while maximum is mapped to 255.

IV. Implementation Approach

A. Mean Filtering

Since the operation is straight forward, no special implementation approach must be considered. Since the mask kernel is 3x3 always, the implementation was in-place implementation.

Fig 17 and Fig 18 shows the images input and output of the operation for one image.

Fig. 17 Input Image for Mean Filtering

Fig. 18 Output Image for Mean Filtering

B. Gaussian Blurring

The implementation approach followed similar case as in Mean Filtering. The convolution is always done with 3x3 matrix so the implementation was in place.

Fig 19 and Fig 20 shows the input and output for the Gaussian Blurring Operation.

Fig. 19 Input Image for Gaussian Blur

Fig. 20 Output Image for Gaussian Blur

C. RGB To GrayScale Conversion

The procedure is straight forward approach where output at each location is calculated based on the input pixel value at that location.

Fig 21 and Fig 22 show the input and output image of the operation

Fig. 21 Input Image for GrayScale Conversion

Fig. 22 Output Image for GrayScale Conversion

D. Edge Detection

1) Laplace Edge Detection

For Laplace edge detection, the input image has to first be converted to GrayScale form and the convolved with the mask discussed in the section earlier.

Fig 23 and 24 show the input and output image for the Laplace Edge Detection.

Fig. 23 Input Image for Laplace Edge Detection

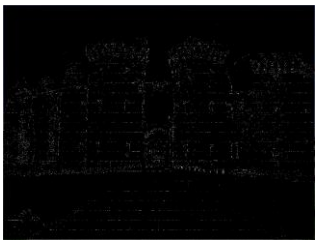

Fig. 24 Output Image for Laplace Edge Detection

2) Sobel Edge Detection Dx

Sobel Operator was applied to the input image. The image was first converted to gray scale and then convolution operation was performed.

Fig 25 and Fig 26 show the input and output of the Sobel Dx Edge Detection.

Fig. 25 Input Image for Sobel Edge Detection Dx

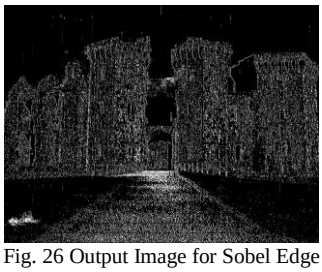

Fig. 26 Output Image for Sobel Edge Detection Dx

3) Sobel Edge Detection Dy

Sobel Operator was applied to the input image. The image was first converted to gray scale and then convolution operation was performed.

Fig 27 and Fig 28 show the input and output of the Sobel Dy Edge Detection.

Fig. 27 Input Image for Sobel Edge Detection Dy

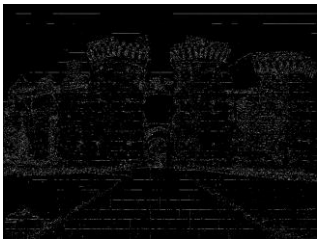

Fig. 28 Output Image for Sobel Edge Detection Dy

4) Sobel Edge Detection Dm

In this calculation, Dx component as well as Dy component are calculated. Then the vector sum of both is calculated in kernel, The input and the resulting edge detected image is shown in Fig 29 and Fig 30.

Fig. 29 Input Image for Sobel Edge Detection Dm

Fig. 30 Input Image for Sobel Edge Detection Dm

5) Prewitt Edge Detector

Implementation approach for all Dx, Dy and Dm are similar to the Sobel filter operator. Fig 31,31,33,34,35,36 show the input and output respectively of all the three approaches repectively.

Fig. 31 Input Image for Prewitt Edge Detection Dx

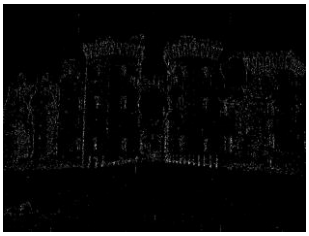

Fig. 32 Output Image for Prewitt Edge Detection Dx

Fig. 33 Input Image for Prewitt Edge Detection Dy

Fig. 34 Input Image for Prewitt Edge Detection Dy


Fig. 35 Input Image for Prewitt Edge Detection Dm


Fig. 36 Output Image for Prewitt Edge Detection Dm

E. Contrast Enhancement

1) Linear Stretching

The maximum and minimum of the pixel values are required for contrast enhancement. So, they have to be figured out first before proceeding to calculate the final intensity value at each point in image. Besides this the operation is fairly easy.

Fig 37 and 38 show the initial and final image respectively.


Fig. 37 Input Image for Linear Stretching


Fig. 38 Output Image Linear Stretching

Fig 39 and Fig 40 shows the histogram of the images before and after the application of linear stretching.

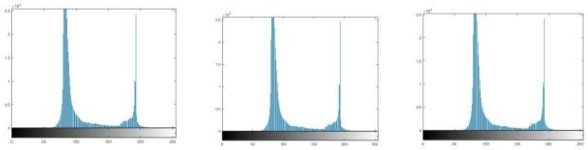
Fig. 39 Histogram Before Linear Stretching

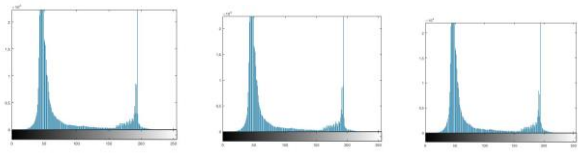
Fig. 40 Histogram After Linear Stretching

2) Power Law

Implementation of power law was a bit tricky since it needs to call the function *pow()*. This was a barrier for speedup in this case.

Fig 41 and Fig 42 shows the image before and after Power Law.

3) Histogram Equalization

Following are the steps that were followed for implementing the Histogram Equalization.

i. Compute Histogram of the pixels 0-255


Fig. 42 Output Image for Power Law

ii. Computer Cumulative Histogram for all the pixels 0-255

iii. Find minimum of the Cumulative Pixel

iv. For all the pixel value apply the formula shown in Fig 43

The histogram was stored in shared memory so the operation was faster. This is because the value of histogram is to be accessed by all the threads, and update on it.

The greater part of this implementation is the calculation of

```
Pixel Val = 255*(CUM_HIST[Initial_Val]-MIN)/NumberOfPixels
```

Fig. 43 Formula For Histogram Equalization

cumulative histogram operation. We have used Hillis-Steele Scan algorithm for this case [3]

Figure 44 and Fig 45 shows the input and output images respectively. Fig 46 and 47 how the histogram of the input and output images.

F. Convolution 2D

A 2D convolution kernel is implemented where an user can play with her own kernel mask. Since for most basic image processing operation, 3x3 stencil operation will do, it is custom developed highly efficient for this case.


Fig. 44 Input Image for Histogram Equalization


Fig. 45 Output Image for Histogram Equalization

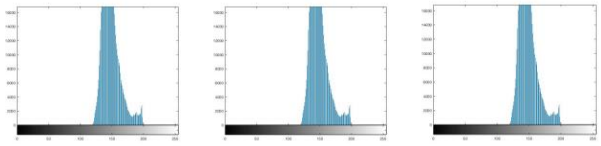
Fig. 46 Histogram Before Histogram Equalization

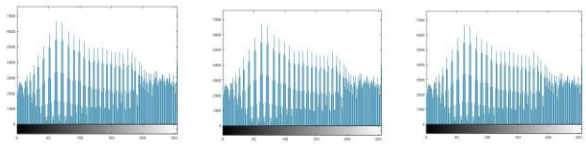
Fig. 47 Histogram After Histogram Equalization

Fig 48 and Fig 49 show the input and output of the operation respectively. The mask kernel used for test operation used corresponds to Sobel Dx filter.


Fig. 48 Input Image for 2D convolution

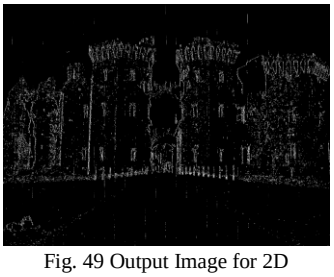
Fig. 49 Output Image for 2D convolution

V. Analysis Result

Timing Analysis using windows API **QueryPerformanceCounter** is used to measure time. Fig 50 through Fig 56 show the result of the timing measurements.

Fig 57 shows the speedup gains from the application for various input cases.

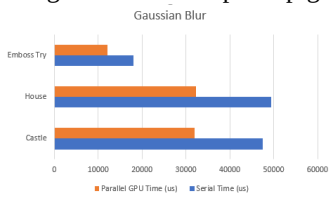

Fig. 50 Timing Comparison for Gaussian Blur Operation

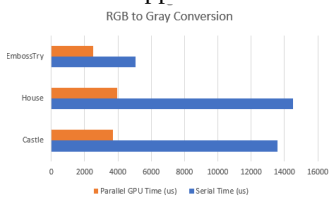

Fig. 51 Timing Comparison for Grayscale Conversion Operation

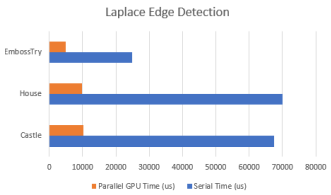

Fig. 52 Timing Comparison for Laplace Edge Detection Operation

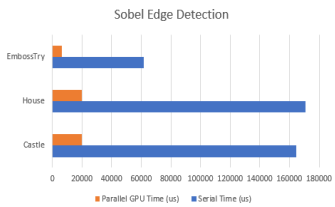

Fig. 53 Timing Comparison for Sobel Edge Detection Operation

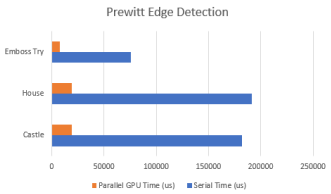

Fig. 54 Timing Comparison for Prewitt Edge Detection Operation

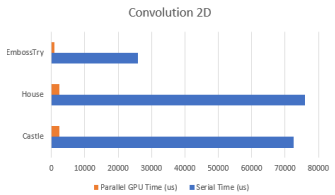

Fig. 55 Timing Comparison for Gaussian Blur Operation

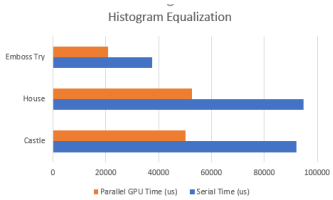

Fig. 56 Timing Comparison for Histogram Equalization

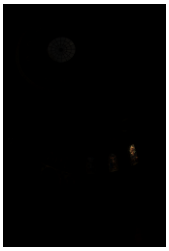

Fig. 41 Input Image for Power Law

	Castle	House	EmbossTry
Convolution 2D	31.11021	32.46022	28.99441
R2G conv	3.647217	3.655841	2.027178
Laplace Edge Detection	6.630781	7.135088	5.118571
Sobel Edge Detection	8.212121	8.706731	9.727503
Prewitt Edge Detection	9.546437	10.05768	9.80249
Gaussian Blur	1.488413	1.534915	1.4906
Histogram Eq	1.834102	1.802575	1.810808

Fig. 57 Speed Ups for various cases

VI. Help From Review

This project was intended to be taken as stepping stone to the field of Digital Image Processing. So the project does was in a sense of dilemma on how to move forward about this project. Some basic operations in image was already implemented, bu the specific issues raised in Review help guide the project in right direction.

Following deals the issues raised and handled during the course of project.

A. Create Goals For Project

It was raised in one of the review that goals needed to be set for such a project that involves implementing multiple problem. So the goal was set to implement simple algorithm in the initial part, and move on to the complex algorithms later on.

As a result, blurring, denoising, RGB to Gray scale conversion was done in initial phase. Later, edge detection algorithm, and tome enhancement operation was done.

B. Provide User chance to play around with custom convolution

As a result of this review I was able to implement the Convolution kernel that accepts Mask kernel from user nad performs the operation on image. The speed up gained in this case is about 32 percent in some case tested.

C. Compare Result

Although I planned to take this project as GPU acceleration of image processing, comparing and profiling time would automatically be a part of it. But the review inspired me to implement the CPU code myself, and I did so. The comparison result presented in the previous section are both my version of implementation, one being in CPU and the other being in GPU.

VII. Target Employed

All the work in the project is done in Windows platform. The development environment used was Visual Studio Community 2015. The hardware used is NVIDIA GeForce GTX960 which is MaxWell based architecture.

All the programming was done using CUDA/C GPU and C++ for host programming.

VIII. Conclusions

This project was hands-on learning opportunity for me in the field of Digital Image Processing. I got to know about many operations that I would have not known otherwise. Over the course of the project, skill of GPU programming using CUDA was developed.

One important takeaway is that sometimes host code may not allow to launch kernel using <<<>>> symbol, so the use of API cudaLaunchKernel will come handy.

IX. Further Enhancements

As an immediate further enhancement, the convolution kernel can be made to accommodate mask kernel of any size.

Launching of threads from the Host in GPU is considered to be in machine that has 1024 threads at least per block, as is the

development machine that I am working on, this can be made more flexible.

Since edge detection has been done, this can be enhanced to incorporate some machine learning solver that identifies object in the images.

Prawar Poudel is a graduate student in the University of Alabama in Huntsville where he is doing is Masters of Science in Computer Engineering. He is expected to graduate Spring 2018.

X. BIBLIOGRAPHY

[1] "Digital Image Processing," [Online]. Available: https://en.wikipedia.org/wiki/Digital_image_processing.

[2] B. Benchoff, "Hackaday," 28 02 2016. [Online]. Available: http://hackaday.com/2016/02/28/introducing-the-raspberry-pi-3/.

[3] "Parallel Prefix Sum on the GPU," [Online]. Available: http://www.umiacs.umd.edu/~ramani/cmsc828e_gpusci/ScanTalk.pdf.

[4] [Online]. Available: https://courses.cs.washington.edu/courses/cse576/book/ch3.pdf.

[5] [Online]. Available: https://2.bp.blogspot.com/.

[6] "Stack Overflow," [Online]. Available: https://stackoverflow.com/questions/687261/converting-rgb-to-grayscale-intensity.

[7] Y. Wang, "EL5123 /BE6223 ---- Image Processing," [Online]. Available: http://eeweb.poly.edu/~yao/EL5123/syllabus_F11.htm.

[8] A. Majumdar, "Visual Computing," [Online]. Available: http://www.ics.uci.edu/%7Emajumder/VC/CS211-F16.htm.

[9] [Online]. Available: https://www.evl.uic.edu/aej/525/lecture04.html.