Available online www.jocpr.com

Journal of Chemical and Pharmaceutical Research, 2016, 8(2):379-387

Research Article

ISSN : 0975-7384
CODEN(USA) : JCPRC5

GPU constructed image segmentation using first order edge detection operators in CUDA environment

Sanjay Saxena[#], Neeraj Sharma and Shiru Sharma

School of Biomedical Engineering, Indian Institute of Technology(Banaras Hindu University), Varanasi, UP, India

ABSTRACT

This paper presents the command of parallel computing on the GPU in contrast to the substantial computations required in processing of the images. The GPU has long been utilized to quicken 3D applications. With the appearance of anomalous state programmable interfaces, programming to the GPU is simplified and is being utilized to quicken a more extensive class of utilizations. Image segmentation has emerged as an imperative stage in image based applications so our main focus in this work to provide speedup in image segmentation using first order edges detection operators. As it plays a very vital role in various image based application. With the frequency of high level Application Programming Interfaces (CUDA - Compute Unified Device Architecture), the supremacy of the GPU is being leveraged to hasten more common purpose and high performance applications. More precisely, this paper emphasis on CUDA as per its parallel programming platform for first order edges detection operators. Standards are done amongst its parallel execution and its sequential execution.

Keywords: GPU(Graphics Processing Unit), CUDA (Computed Unified Device Architecture), Image Segmentation, Order, Edge Detection,

INTRODUCTION

Now a day, parallel computing plays a very vital role in the field of image processing[1]. Image processing is a well-known and established research field. As we all knows that image processing is a form of signals processing in which the input is an image, and the output can be an image or anything else that endures some expressive processing. There are various stages are present in image processing such as image enhancement, image segmentation, feature extraction, pattern recognition etc. Over and over again, processing of an entire image, where the same steps are applied to every pixel of the image. That means a lot of repetition to do the same work. Newer technology of acquisition of an image allows better quality images to be taken. That are responsible for bigger files in size and longer processing time. CUDA programming model developed by NVidia provides an efficient environment of parallelism on GPUs. This research paper deals with a very important technique of image processing i.e. first order edge detection operation such as Prewitt, Sobel and Roberts edge detector using CUDA on GPU. For this we have divided this paper into several sections next section II describes the brief summary of image segmentation, first order edge detector operators, GPU and CUDA. Section III describes the parallel implementation of edge detection in CUDA programming environment. Experimental Setup and Results have been given in section IV. And finally Discussion and conclusion is given in section V.

I. Brief Overview Image Segmentation, GPU and CUDA

A. Image Segmentation

Image Segmentation is the procedure of segregating an image into its constituent regions and extracting a meaning full information from it[2]. It is one of the important step of analyzing an image. It could be used for occlusion boundary estimation within motion or stereo systems, object recognition, image compression, image editing, or image database look-up.

There are different methods of classifying image segmentation algorithms[2][3].

One of the way to classify segmentation algorithm given as below:

1. Threshold based segmentation: Histogram thresholding and slicing methods are used to segment an image. They may be applied directly to an image, but can also be combined with pre- and post-processing techniques.

2. Edge based segmentation: With this technique, detected edges in an image are anticipated to characterize boundaries of an object, and used to identify these objects.

3. Region based segmentation: In this type of segmentation an edge based method may endeavor to treasure the object boundaries and then locate the object itself by filling them in, a region based method proceeds the conflicting methodology, by (e.g.) preliminary in the central of an object and after that "growing" outward till it encounters the boundaries of the object.

4. Clustering techniques: Although clustering is occasionally used as a synonym for (agglomerative) segmentation techniques, we use it to signify methods that are mainly used in investigative data investigation of high-dimensional measurement configurations. Based on this context, clustering methods endeavor to group together patterns that are comparable in some sagacity. This penalty area is very comparable to what we are endeavoring to do when we segment an image, and certainly some techniques of clustering can willingly be pragmatic for image segmentation.

5. Matching: It is applied when we know what an object we wish to classify in an image (almost) looks similar, we can use this information to discover the object present in an image. This methodology of segmentation is called matching.

In our paper we have focused on edge based segmentation. As edges provides an outline of the object. In the physical plane, edges correspond to the discontinuities in surface orientation, depth, change in material properties and light variations [5].

Edge detection process consist the following steps given as below:

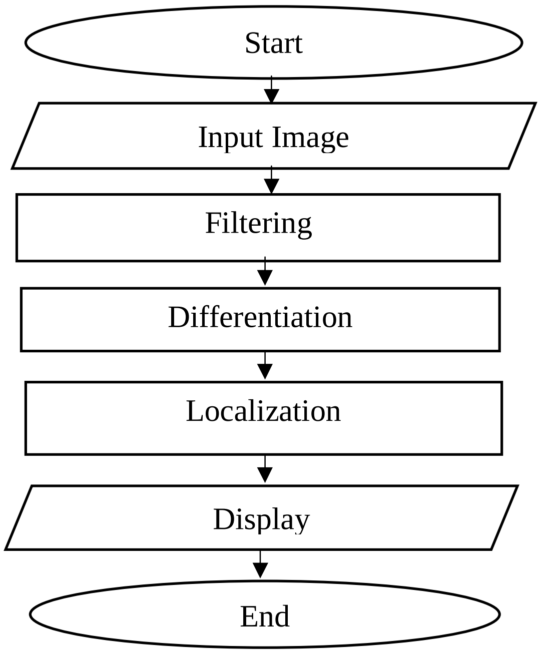

Figure 1: Edge Detection Process[3]

Filtering: In this stage a filter is used to enhance the quality of an image. Generally, Gaussian filters are used for its effectiveness for real time images.

Differentiation: This stage separates the pixels present in the image from other pixels.

Localization: This process includes determining the exact location of the edge.

There are four types of edge detection operators are available: Gradient or derivative filters, template matching filters, Gaussian derivatives and pattern fit approaches**.**

We have focused on First Order Derivative filters discussed as below

B. First order Edge Detection Derivative Operators

Local transitions among different image intensities constitute an edge. Therefore, the aim is to measure the intensity gradients. So edge detectors can be viewed as gradient calculators.

Based on the differential geometry and vector calculus, the gradient operators is represented as below:

$$\nabla = \begin{bmatrix} \frac{\partial}{\partial x} \\ \frac{\partial}{\partial y} \end{bmatrix} \qquad (1)$$

Applying this to image f, we gets

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \qquad (2)$$

Difference between the pixels are quantified by the gradient magnitude. These difference can be extended to 2D as $g_x = \frac{\partial f}{\partial x}$ and $g_y = \frac{\partial f}{\partial y}$. Then the magnitude is given by

$$\nabla f(x,y) = mag(\nabla f(x,y)) = [g_x{}^2 + g_y{}^2]^{\frac{1}{2}} \qquad (3)$$

The gradient direction is given by $\theta = \tan^{-1}\frac{g_x}{g_y}$. (4)

Prewitt Operator:

This method takes the central difference of the neighboring pixels. So the difference can be viewed as below:

$$\frac{\partial f}{\partial x} = f(x+1) - f(x-1)/2 \qquad (5)$$

In 2D this is like

$$f(x+1,y) - f(x-1,y)/2 \qquad (6)$$

The central difference can be obtained by using the mask [-1 0 +1]. This method is very sensitive to noise. Hence to avoid noise, this method does some averaging.

The prewitt approximation using 3 X 3 mask is given as follows:

$$\nabla f \cong |(z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)| + |(z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)| \qquad (7)$$

It masks are given as follows:

$$M_x = \begin{matrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{matrix} \qquad (8)$$

$$M_y = \begin{matrix} -1 & 0 & 1 \\ -1 & 0 & 0 \\ -1 & 0 & 1 \end{matrix} \qquad (9)$$

Sobel Operator:
It also relies on the central difference. It can be viewed as the approximation of the Gaussian derivatives.

A 3 X 3 digital approximation of the sobel operator is given as

$$\nabla f \cong |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)| + |(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)| \qquad (10)$$

It masks are given as follows:

$$M_x = \begin{matrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{matrix} \qquad (11)$$

$$M_y = \begin{matrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{matrix} \qquad (12)$$

To detect the edges in the diagonal direction masks are given below:

$$M_x = \begin{matrix} 0 & 1 & 2 \\ -1 & 0 & 1 \\ -2 & -1 & 0 \end{matrix} \qquad (13)$$

$$M_y = \begin{matrix} -2 & -1 & 0 \\ -1 & 0 & 1 \\ 0 & 1 & 2 \end{matrix} \qquad (14)$$

Roberts operator:
Assume f(x,y) and f(x+1,y) be the neighboring pixels. The difference between the adjacent pixels is obtained by applying the masks [1 -1] directly to the image to get the difference between the pixels. This mathematically defined as follows:

$$\frac{\partial f}{\partial x} = f(x+1, y) - f(x, y) \qquad (15)$$

The approximation of this type of operator can be mathematically defined as follows:

$$g_x = \frac{\partial f}{\partial x} = (z_9 - z_5) \qquad (16)$$

$$g_y = \frac{\partial f}{\partial y} = (z_8 - z_6)$$

Roberts masks of the given cross difference are

$$g_x = \begin{matrix} -1 & 0 \\ 0 & 1 \end{matrix} \quad \text{And} \quad g_y = \begin{matrix} 0 & -1 \\ 1 & 0 \end{matrix}$$

Its magnitude is

$$\nabla f(x,y) = mag\big(\nabla f(x,y)\big) = [g_x{}^2 + g_y{}^2]^{\frac{1}{2}} \qquad (17)$$

And the edge orientation is given as

$$\theta = \tan^{-1} \frac{g_x}{g_y} \qquad (18)$$

C. GPUs (Graphics Processing Unit) and CUDA (Computed Unified Device Architecture) Programming Model:

It is a graphical processing unit which empowers us to run high definitions graphics, which are the request of current computing. GPU computation has delivered an enormous edge above the CPU with reverence to speed of computation. The basic difference between the CPU and GPU are given below defined by [4].

Table 1: Difference Between CPU and GPU

CPU	GPU
It is having Quick caches (great for data reuse)	Numerous of units of math
Fine branching granularity	Fast access to onboard memory
Numerous different processes or threads	Execute a program on each vertexor fragment
High performance on a single thread of execution	High throughput on parallel tasks
CPUs are best for parallelism of task	GPUs are best for parallelism of data
CPU optimized for high performance on sequential codes in terms of caches and branch prediction	GPU optimized for higher arithmetic intensity for parallel nature in terms of Floating point operations

Therefore, it is one of the utmost fascinating areas of research in the arena of recent industrial research and growth. It is gaining approval in performing computation exhaustive jobs and is compatible for the applications of data parallel computation. There are some basic There are several studies have been done till now which describe about the structure of GPU given in [1].

The CUDA (Compute Unified Device Architecture) is a programming model which is developed by NVIDIA. Apart from this that is a software environment deliberated to handle tasks of parallel computing. Its chief concepts are a hierarchy of groups of threads, barrier synchronization and shared memories. These concepts deliver a programming model used for parallelism data, thread and tasks.

CUDA presents a configuration to the programmer of GPU which consist of a collection of threads to execute in parallel. This structure is analogous to the traditional SIMD (Single Instruction Multiple Data) parallel model.
The computation standard of the CUDA defined by [5] is given as follows:

- A program is distributed into blocks that can execute concurrently. It is a cluster of threads mapped to a solitary multiprocessor used by the programmer to share the memory. All available resources are divided equally amongst all threads of concurrent blocks on a single multi-processor. Furthermore, the data is also allocated amongst all threads in a SIMD fashion by the programmer.
- All threads are systematized into warps. It is a cluster of 32 parallel scale threads, that can run parallely on the multi-processors. Furthermore, each warp runs solitary kernel instruction at a while. Assortments of warps are identified as thread block and these threads execute on the identical multiprocessor and usage an amount of the local storage. So, the programmer can spontaneously design the numeral of threads to be accomplished.
- The GPU processes numerous threads concurrently by using the kernel on respectively one of them. A kernel is the code executed by the threads; the executable is shared amongst all of the threads present in the system. In order to classify the restrictive implementation of operations on multi processors, the programmer can easily process individually on a distinctive thread ID.
- Memory hierarchy in the CUDA is in the form of registers, global memory, constant memory plus textures. Register file is the fastest level in hierarchy however only has a restricted amount of space. If we talk about Constant memory, it is a subsection of device memory which is shared among processors and cannot be altered at execution time by a device. Global memory allows r/w, read and write maneuver from all threads, however it is uncached and has extensive latencies. Texture memory is a subsection of the device memory; it is read-only on the device, provides faster cached reads, and allows addressing through a specialized texture unit.

II. Implementation

For efficient implementation we have extensively used CUDA environment on GPU (Version of CUDA and Series of GPU is given in result and discussion section). To use CUDA extensively for purpose of implementation we divided the edge detection into several sections defined as below using different functions of CUDA.

1. ***cudaMalloc( ):***This is used to call CUDA library for the allocation of memory on the device i.e. present GPU on the system. Prototype is given as
cudaMalloc((void **) & device_pixels , Size_Image);

2. ***cudaMemcpy( ):***This is implemented to copy the substances of the host memory to the device memory referenced by device_pixels. Prototype is given as
cudaMemcpy(device_pixels , host_pixels , Size_Image, cudaMemcpyHostToDevice);
basically host_pixelsrepresents the pixels presents on CPU.

3. ***cutCreateTimer( ), cutStartTimer( ) and cutStopTimer( )****:* CUDA requests to create, start and stop the timer by these. Prototype is given as
cutCreateTimer (&timer)
cutStartTimer (timer)
cutStopTimer (timer)

4. ***dim3 block( ) :*** It defines block sizes. For example 32 x 32 for 1024 threads per block.
Prototype is give as
dim3 block(32,32);

5. ***dim3 grid( ):***It articulates size of the grid. If the image is of size 512 x 512, then the grid will be of 32 x 32 and will have 1024 blocks. Subsequently apiece block encompasses 512 threads, this will aggregate to 262144, which is precisely the numeral of pixels in a 512 x 512 image. Prototype is given as
dim3 grid(width/32, height/32);

6. To call the device method suppose named as function_edgedetect_device following prototype are used
function_edgedetect_device<<<grid , block >>> (device_pixels , device_resultPixels , width , height);

7. *blockIdx .x * blockDim .x + threadIdx .x* to find exact location of pixel's x value and *blockIdx .y * blockDim .y + threadIdx .y* to find exact location of pixel's y value have been implemented. These two commands fundamentally define which thread process on which pixel present in the image. As computed above, there are 262144 threads performing on 262144 pixels for 512 X 512 image. Each thread performed on its exclusive pixel and elude processing the pixels preserved by other threads. Subsequently each thread is exclusively recognized by its peculiar thread id, block id and we know the proportions of the block, we can use the above discussed technique to allocate a distinctive pixel coordinate for separately thread to work on.

8. ***cudaThreadSynchronize ():*** This function used to instruct the threads to coordinate before accomplishing auxiliary instructions.

These above are the pre-defined functions present in the CUDA libraries that are used expressively. Basic fundamental is to take input image in the host i.e. CPU, transfer it pixel values to the device i.e. GPU where the specific threads are able to handle specific pixel values that perform edge detection using specific Mask of particular first order edge detector operator described as above and then the segmented image is displayed through the host.

III. Experimental Setup and Results

Three Experiments have been performed with different data sets contains images with different size, i.e. First data set contains 20 images having size 512 X 512, Second data set contains 20 images having size 1024 X 1024 and the third data set are having 20 images of size 3072 X 3072.

All three experiments are having following hardware and software configuration:

Hardware Environment:
Processor: Intel Core i7 -4710 CPU @2.50 GHz
RAM (Random Access Memory): 8 GB
Hard Disk Drive: 1 TB
GPU: Ge Force GTX 860M
Cuda Cores: 640

Software Environment:
System Type: 64 Bit operating System, x – 64- based processor
Visual C++ 2010 Express
CUDA Version 5.0

Note: Speed up is calculated using the parallel implementation of edge detection algorithm and its sequential implementation in same system.

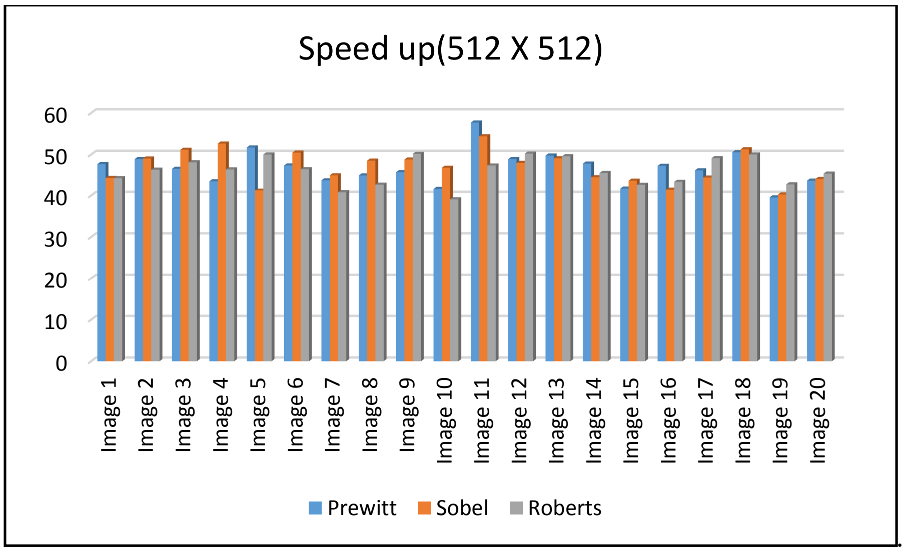

Figure 2: Speed up plot for 20 images of size 512 X 512

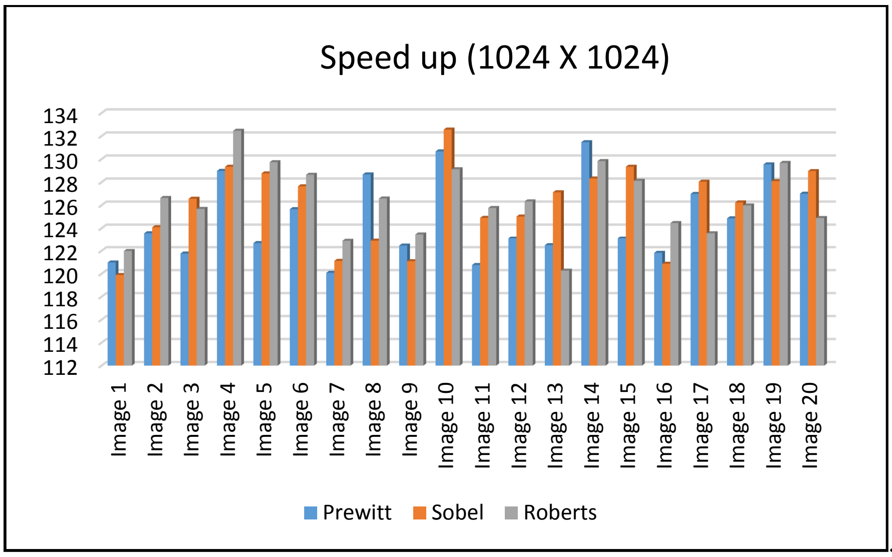

Figure 3: Speed up plot for 20 images of size 1024 X 1024

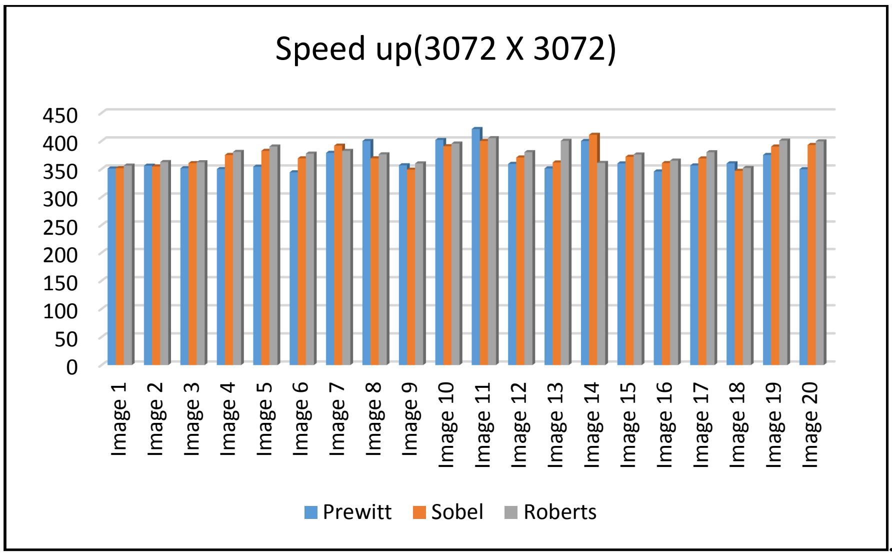

Figure 4: Speed up plot for 20 images of size 3072 X 3072

DISCUSSION AND CONCLUSION

As we have already discussed that GPUs are widely used to accelerate several 3D applications, different kind of simulations and gaming. Different programmable interfaces considered as High level allow this expert technology to be used for general purpose computing. NVidia's CUDA provides the efficient environment to implement this. This paper gives an outline of the type of performance gains in the implementation of the basic and important technique of image processing i.e. first order edge detector operator that can be accomplished by switching sequential programming model over to the parallel programming model. Techniques of Image processing algorithms is a kind of algorithms that work significantly in attaining the best remunerations of CUDA. Generally, algorithms of image processing are such that in which a type of computation is repeated frequently in enormous amounts as we have seen in edge detection. Maximum of these techniques can be processed individually of each other. In our paper we have perfectly utilized CUDA's huge amounts of threads on GPU Ge Force GTX 860Mand got significant results. We have tested 60 images consisting three different size (512 X 512, 1024 X 1024, 3072 X 3072) and found speed up around hundred times in terms of percentage. As image segmentation is one the important step of image analysis for this our paper provides the fast and quick mechanism to do that significantly. As per future aspect we can handle other time consuming techniques of medical image processing using CUDA to provide the quick and prompt results to medical practitioner who are can use it for treatment planning.

REFERENCES

[1] Saxena S., Sharma N., Sharma S., *Research Journal of Applied Science, Engineering & Technology*, 12(2), 223-238, **2016**.
[2] Sharma N., Aggrawal L.M., *Journal of Medical Physics*, 35(1), 3-14, **2010**
[3] Gonzalez R., Woods R.E., *Upper Saddle River, NJ, Prentice Hall*, **2002**.
[4] J. Ghorpade, J. Parande, M. Kulkarni, and A. Bawaskar, "Gpgpu processing in cuda architecture," CoRR, vol. abs/1202.4347, **2012**.
[5] Tse J., "Image Processing with CUDA," Master's Thesis, University Of Nevada, Las Vegas, August **2012**
[6] Salvo RD, Pino C., "Image and video processing on CUDA: State of the art and future directions", *Proceedings of MACMESE* , 60-6,**2011**.
[7] Yang Z., Zhu Y., Pu Y., "Parallel Image Processing Based on CUDA", *International Conference on Computer Science and Software Engineering*, pp. 198-201, **2008**.
[8] Catanzaro B., Su B-Y., Sundaram N., Lee Y., Murphy M., Keutzer K., "Efficient, high quality image contour detection", *IEEE 12th International Conference on Computer Vision*, 2381-2388, **2009**.
[9] Luo S., Jin J.S., Chalup S.K., Qian G., "A liver segmentation algorithm based on wavelets and machine learning", *IEEE International Conference on Computational Intelligence and Natural Computing*, 122-125, **2009**.

[10] Luo Y., Duraiswami R., "Canny edge detection on NVIDIA CUDA", ***2008** IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, **2008**.
[11] Park S. I., Ponce S. P., Huang J., Cao Y., Quek F., "Low-cost, high-speed computer vision using NVIDIA's CUDA architecture", *37th IEEE Applied Imagery Pattern Recognition Workshop*, **2008**.