ISSN 2229-5518

High-speed processing of larger Edge-detection filters using NVIDIA CUDA Architecture

First Sowjanya Latha. Mudumba, Second K. Nageswara rao, Third Yugandhar Garapati

Abstract — Image processing on large images is highly computation intensive. General purpose CPUs with Multiple Cores can provide some level of computational speed. Spatial filters used in Image Processing are inherently parallel. By using a Parallel computing Architecture such as CUDA "Compute Unified Device Architecture", huge improvements can be achieved in processing times. NVidia's GPUs that follow CUDA achieved significant improvement in Canny Edge detection algorithm. Typical Canny edge-detection filter is of size 3x3. In many real applications, the edge detection needs higher size of filters. The aim of this paper is to study the improvement in processing times on larger edge detection filters when Nvidia's GPUs are used for computation in comparison to processing the same on General purpose CPUs. In this paper, we describe some of the benefits associated with implementation of canny using CUDA and we are looking at possible development pitfalls, solutions and performance results.

Index terms— Edge Detection, CUDA, GPU, Parallel Processing

1. Introduction

Many image processing applications operates on higher resolution images, which is highly computation intensive and more time consuming, especially if multiple operations have to be performed on an image. This problem becomes worse when the input is very large. General purpose CPUs we can some level of parallelism in processing. Using a parallel computing architecture "CUDA" (Compute Unified Device Architecture)[3] we can achieve processing time improvements. Recent GPUs (Graphics processing unit) came with multiple cores, which can be used for general purpose parallel computation. GPU is connected with global memory and follows SIMD (Single Instruction Multiple Data) architecture, so that we can launch multiple parallel threads for to perform an operation on each pixel with this we can achieve high computational speed. Where general purpose CPUs follows SISD (Single Instruction Single Data), we can obtain some level parallelism with CPU having multiple cores using Parallel constructs, which is very cost effective. GPU is a specialized unit generally used for 2D and 3D graphics acceleration. Implementation of image processing algorithms on GPU, reduces the overload on CPU and allows CPU to perform other operations while algorithm is running on GPU and reduces the processing time as well as with low cost.

Edge detection algorithms are most commonly used algorithms in image processing. Canny edge detection algorithm is most commonly used edge detection algorithm to detect edges in an image. Following are the improvements made in Canny edge detection method among all other edge detection methods.1. Low error rate. 2. Well localization of edge points. 3. To have only one response to single edge. Based on these criteria, the edge detector first smoothes the image to eliminate the noise in an image. It then finds the gradient and nonmaximum suppression is applied and then hysteresis thresholding is applied to reduce the gradient array. Typically filter sizes we used in canny edge detection algorithm is 3x3. Real time image processing applications operates on higher resolution images with larger filters. Usually edge detection algorithm[1] consists of four steps. **Step1:** Gaussian filtering S**tep2:** Sobel filtering **Step3:** non maximum Suppression **Step4:** hysteresis thresholding. In Step1 Gaussian filter is applied to remove the noise from the image by applying Gaussian filter. In our application we tried 3x3, 5x5 and 7x7 size filters. The main advantages of small filters is it is very quick to compute and only few input pixels need to be examined to determine value of each output pixel and here the simple operations we used are addition and subtraction. A main disadvantage of small kernel is very sensitive to noise and produces weak response to genuine edges unless they are very sharp. Figures 1(a) is an examples of Gaussian filter of size 5x5. In Step2 we used a pair of sobel kernels to detect edges and intensity changes in horizontal and vertical directions. In our application we used filters of size 3x3 and 5x5, In figure2(a) represents the horizontal sobel kernel **dx** of size 5x5, figure2(b) represents the vertical sobel kernel **dy** of size 5x5. Technically, it is a discrete operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the results of the sobel operator are either the corresponding gradient vector or norm of the vector. By applying these **dx** and **dy** filters in horizontal and vertical directions, we obtain edges in the image, then compute the magnitude M and gradient direction θ at each pixel using **dx** and **dy**. Below are the following formulas

M= $\sqrt{dx2 + dy2}$, gives the magnitude of the pixel.

$\theta = \tan^{-1}\frac{dy}{dx}$, gives the gradient of the pixel

0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04
0.04	0.04	0.04	0.04	0.04

Figure1(a) 5x5 Gaussian filter

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

Figure2(a) 5x5 sobel filter dx

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

Figure2(b) 5x5 sobel filter dy

Step4 is non-maximum suppression, in this local maximum along with gradient direction is detected. Step5 is hysteresis thresholding, we are considering two threshold values t_Low and t_High. The pixels whose **M > t_High** are strong edge pixels, if **t_Low<M<t_High** are weak edge pixels and if **M<t_Low** are considered as non edge pixels. Based on this classification (1) a strong pixel is an edge pixel. (2) a weak edge pixel is an edge pixel if it is adjacent to an edge pixel (3) Non edge pixel is not an edge

We implemented edge detection algorithm on CUDA by segmenting the image into number of blocks and then filter is applied on each pixel within a block by launching number of parallel threads for each pixel depending on the GPU memory.

Usage of larger filters in implementation of edge detection algorithm on GPU is limited depending on the memory of the GPU.

2. Implementation

CUDA hardware architecture consists of SM (Streaming Microprocessors) and Global memory. Each streaming microprocessor consists of eight processor cores and threads, which runs in parallel. All the threads in Streaming Microprocessor can access the shared memory which is shared among all threads in a Streaming Microprocessor. Global memory is larger in size and can be accessed by all Streaming microprocessors. However, the accessing time of global memory is generally longer than shared memory access time; here the shared memory of each streaming microprocessor is used like as cache memory of a global memory.

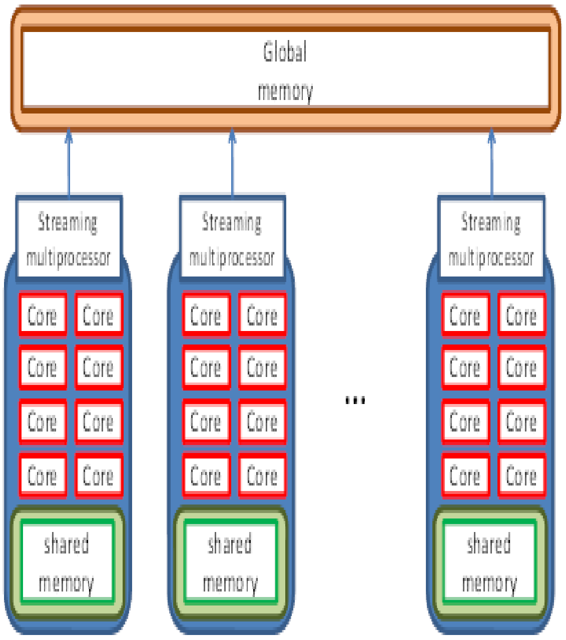

Figure3 CUDA hardware (GPU) Architecture

In our implementation we used global memory for reading and writing operations. The input image is stored in the global memory because shared memory is too small.

Implementation of the steps followed in the edge detection algorithm is shown diagrammatically in the Figure4.

Gaussian filter:

In this we load the input image into global memory of the GPU. Applied Gaussian filter on each pixel by launching number of parallel threads. A Gaussian filter is used for the initial step of smoothing as it has a simple mask. The mask used here is usually smaller than the actual image, So that the mask can slid over the image, manipulating the square of pixels at a time. Using CUDA we can perform these manipulations in parallel by launching multiple threads, so that it improves the performance time. The output of Gaussian filter is written on to global memory of the GPU.

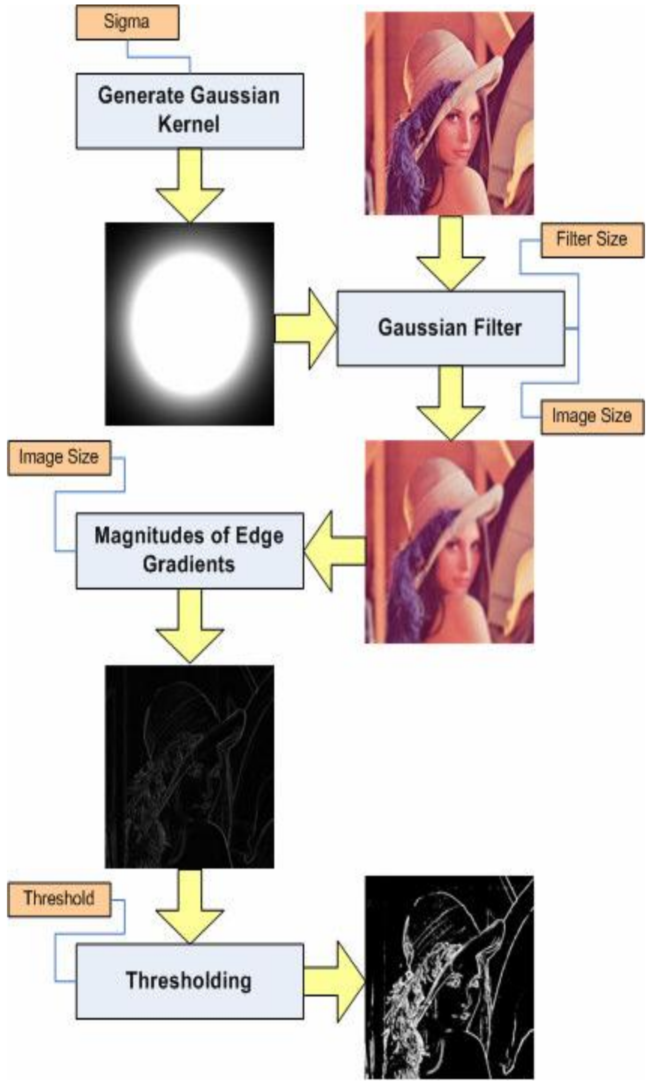

Figure4 Implementation of Canny edge detection algorithm using CUDA

Sobel filter:

It reads the output of the Gaussian filter from the global memory of the GPU, and applies horizontal filter in X-direction and vertical filter in Y-direction in parallel on each pixel, the output of the sobel filter is written back to global memory of the GPU. After that magnitude M and gradient θ is computed and the result is written on to global memory of the GPU.

Non-Maximum Suppression:

Using magnitude M and the gradient θ is computed on each pixel, these values are used for to find whether the pixel is local maximum or not. If the pixel is local maximum, it is recorded into global memory as edge pixel.

Hysteresis Threshold:

In this we applied two threshold values to determine whether the pixel is strong edge pixel, weak edge pixel or not an edge. If M>t_High then it is classified as Strong edge pixel, if t_Low<M<t_High then it is consider as weak edge pixel and if M<t_Low then it is classified as no edge pixel.

3. Experimental Results

For implementation, we used a single CPU PC with 4GB RAM and I5 processor, GPU used is NVIDIA GeForce GT530. For the purpose of comparison of the performance and results we implemented conventional software approach running on a single CPU, same algorithm which runs on GPU. Following table shows performance comparison between CUDA and GPU.

Name of the Kernel	Characteristic Feature	Image Size	Average Time observed CPU (in ticks)	Average Time observed GPU (in ticks)
Edge Detection Algorithm	Color Images	3456x5184 Bmp image data.	3x 3: 8752 5x 5: 18720 7x 7: 33509	3x 3: 858 5x 5: 1887 7 x 7: 3416
Edge Detection Algorithm	Gray Images	3456x5184	3x 3: 4150 5x 5: 7255 7x 7: 11934	3x 3: 562 5x 5: 1014 7x 7: 1685

With the use of CUDA achieved processing time improvements 9 times of CPU ticks. Another advantage of using CUDA is we can optimize the time of CPU while the image processing algorithm runs on GPU.

The processing time improvement using CUDA is also depends on the GPU hardware, which we used for the computation. Segmentation of processing problems and algorithms is a big threat. Proper understanding of the GPU architecture is required to take full advantage of the available hardware.

4. Conclusion

In this paper, we implemented edge detection algorithm on CUDA with varying sizes of filters of sizes 3x3, 5x5 and 7x7. Comparing the performance of CPU with GPU,GPU gave 9 times speed up in terms of ticks. Here we can reduce the workload on CPU by transferring the computations on to GPU, So that we can optimize the performance of the CPU and GPU to achieve better performance in processing of the algorithms. We used a GPU-GT530 having capacity of 1GB with 98 cores for our implementation. We got GPU stopped working problem during edge detection algorithm on a color image of size (3456 X 5184) when we use a filter mask size of 9x9, but with the same 9X9 filter, edge detection algorithm is successfully on a grayscale (single channel) Image of size (3456 X 5184) and shows 9 times performance improvement in terms of CPU ticks. For smaller images which is having resolution (1200 X 800) it ran up to 35x35 filter and shows continuous improvement(9 times performance improvement when compare to CPU). The edge detection algorithm for color image takes more time when compare to gray image because color image contains each pixel information in three channels (RGB), for gray image information is stored in a single channel. We may achieve the better processing time improvements with GPU having many cores. The processing times and performance of the GPU also depends on type of the hardware we used for processing.

References

- [1] GPU Kohei Ogawa, Yasuaki Ito, "Efficient Canny Edge Detection Using a GPU" in *2010 First International Conference on Networking and Computing*

- [2] R Farivar, D Rebolledo, E Chan,R Campbell, "A Parallel Implementation of K-Means Clustering on GPUs" in *Proceedings of the 14th International Conference on High Performance Computing, 2007, pp. 197-208*

- [3] NVIDIA, *NVIDIA CUDA Programming Guide, July 2009*

- [4]*NVIDIA_CUDA_Tutorial_No_NDA_Apr08 by NVIDIA corporation*

Sowjanya Latha. Mudumba pursuing M.Tech in Mother Teressa Institute Of Science and Technology, sathupally. Areas of interest are Networking and computing.
Email_id :latha.mudumba.it@gmail.com

K.Nageswararao working as a Associate Professor in Mother Teressa Institute Of Science and Technology, Sattupalli, Khammam(Dist).
Email_id :nageswararaokapu@yahoo.com

Yugandhar Garapati working as a Associate Prof in Mother Teressa Institute Of Science and Technology, Sattupalli, Khammam(Dist).
Email_id : yugandhar.garapati@gmail.com