IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol.-6, Issue-7, July 2018 E-ISSN: 2347-2693

Performance Enhancement of Edge Detection Methods for Human Bone Fracture X-Ray Image Using Graphical Processors

Saima Iram[1*], Jabir Ali[2], Pradeep Kumar[3]

Computer Science and Engineering, Noida Institute of Engineering & Technology, Greater Noida, Uttar Pradesh, India[1]
Computer Science and Engineering, Noida Institute of Engineering & Technology, Greater Noida, Uttar Pradesh, India[2]
Computer Science and Engineering, Noida Institute of Engineering & Technology, Greater Noida, Uttar Pradesh, India[3]

[*]Corresponding Author: saima.iram01@gmail.com, Tel.: +91 9990657656

Available online at: www.ijcseonline.org

Accepted: 10/Jul/2018, Published: 31/Jul/2018

***Abstract*—** Edge detection is a crucial step in medical imaging and in a no. of other image processing applications, such as face-identification or recognition, and other classification problems. Various methods have been developed for edge detection based on applications and edge types. Some of the most common techniques used are Sobel, Prewitt, Robert, LoG and Canny etc. However, most of these methods for edge detection of various images (including x-rays image) is a computationally expensive process in terms of both time and space. Because of this delay the patients and the doctors do not get instant information or imaging reports (for example regarding fractured bone in case of x-rays). This ultimately leads to delayed diagnosis and treatment of the patient. In this work we present our findings of research related to an important edge detection technique which involve finding image gradient. We emphasize that our approach is equally valid for many different kinds of edges in an image and not just for fractured bone. To eliminate latency issue we used a graphical processor with CUDA API to implement an image gradient. The graphical processors are massively parallel processors that come inside a graphics card and have become a standard piece of hardware on all modern day computing systems including portable hand-held device. We emphasize that alternate solutions such as FPGA (Field Programmable Gate Array) and ASIC (Application Specific Integrated Circuit) based solutions are much costlier and take much longer time for development as compared to a graphical processor which is programmable using C-CUDA. We compared our implementation's performance with respect to a CPU-only implementation. To prove our idea we used an algorithm which is a parallel version of naïve serial algorithm. Thanks to GPU's enormous amount of computational units, our GPU-implementation shows several fold speed ups with respect to a standard CPU-only implementation. Our proof-of-concept (PoC) developed as part of this research, thus establish that the GPU stands a very good candidate for such edge detection problems where we need faster results, i.e. in real time or in near real-time.

Keywords— Digital Image, Edge detection, GPU, CUDA, X-ray, gradient

I. INTRODUCTION

Medical Image processing or imaging is an important means for early detection of various diseases, including Cancer etc. One of the main tasks in such imaging is inspection of a region of inertest by doctors. This involves edge detection as a crucial step which is a crucial step in medical imaging. Edge detection is the heart of many image processing applications including face recognition, and various classification problems. Edge detection algorithms are often developed based on type of edge and the application where it will be used. Among all the techniques of edge detection most famous techniques include: Sobel, Prewitt, Robert, LoG and Canny etc. However, due to nature of the problem these methods of edge detection of various images (including x-rays image) is a highly computationally expensive task. Due to this delay, the patients and the doctors have to wait for long time to get the report and subsequently start the treatment. This is an undesirable solution.

In this work we describe our approach to speed up edge detection process related to an important edge detection method that involves finding image gradient using image convolution. In particular to eliminate delay issue discussed above we proposed using a graphical processor and programming it using C-CUDA API to implement an image gradient. CUDA is an extension to C programming that enables a C knowing programmer to exploit the resources of a GPU or Graphical Processing Unit for general applications.

The remainder of this paper discusses the above ideas and has been further organized in several sections and subsections. Section II presents a summary of the literature survey that we conducted. The introduction to the GPU and CUDA API, and motivation for using a GPU has been discussed in Section III. Our main approach has been discussed in Section IV. In this section we described system architecture and our algorithm. In this work we have compared our GPU implementation's performance with respect to a CPU implementation. The results of such performance comparisons are presented in Section V followed by an in-depth analysis of the results in the same section. Our results show that at least for larger size images the gain by using GPU is around 8 times. This ultimately means that a report that is generated normally in 60 minutes will now be available in 8 min. Finally, in section VI we have discussed important conclusions drawn from this research.

II. LITERATURE REVIEW

A number of related works has been sighted in literature by researchers in the area of edge detection, including those for X-rays. We present here most important researchers conducted recently.

In digital Image processing research many edge detection techniques have been proposed. Among the various developed algorithms which are meant to extract edges from digital images, Gradient based operators like Robert, Prewitt, Sobel are very common. The algorithm proposed by Canny in 1986 is considered as one of the ideal edge detection algorithm. It is one of the best choices for images that have noise. Canny's edge detection algorithm aims at reducing the likely-hood of detecting false edge, and has been designed to give sharp edges [1-6].

In [7] the authors have presented a modified Canny Edge Detection algorithm with an aim to detect the boundaries of spine disc image from the noisy image, which is normally a very difficult task. The authors have verified the results with medical database and are found to be an optimal result.

The authors in [9] have presented a technique for reduction of blur which is more computationally efficient. Such blur is normally caused when the images are shaken due to hand movements, which are registered by mobile and other portable units. Based on the various quantitative measures of image quality, the authors have proven that their technique outperforms similar techniques used for image deblurring. Further, they proved that the technique is more efficient in terms of computations. They have presented a GPU implementation of their own technique.

GPU have been applied in a many areas apart from medical imaging. Such as in [10], the authors have demonstrated a successful application of GPU for solving a computation related problems in Ad-hoc networks.

Cache coherence is one of the common features in shared memory multiprocessors including GPUs. But implementing such techniques is quite difficult, especially in systems with discrete CPUs and GPUs. Such systems are produced by different vendors and may give rise to many compatibility issues. The authors in [11] have proposed a selective caching mechanism to solve this problem of hardware cache coherence implementation

III. INTRODUCTION TO GPU AND CUDA

A. Introduction

In this present age of technology and innovation, with the market driven to deliver at the most competitive price, it is quite safe to assume that virtually, almost every person owns a desktop. Using a computer has now become a lot easier; every aspect of an operation is now being done through Graphic User Interfaces. Apple and Microsoft are going head to head in trying to the deliver the best possible, fluid and seamless user interface in their operating systems, and at the same time, the transition from serial to parallel computing has made modern processors a lot more powerful, meeting the needs of several applications including the fields of scientific research and entertainment.

B. Motivation

On one hand Moore's law is still technically valid. But on the other hand for all practical intents and usefulness it is no longer as meaningful as it was. Of course, we can still double the number of transistors that can be cost effectively put on a chip or IC. It continues to double every two years or so as

per the Moore's postulation of 1965. However, the performance gains that this has traditionally promised\, has stopped many years ago. In fact from the year 2002 onward CPU manufacturers started producing dual core CPUs. So instead of increasing clock speeds (a traditional way of increasing speed of CPU in line with Moore's Law) , that enables software to automatically execute faster, CPU manufacturers have now started putting more number of cores, or CPUs, in the single chip. Since a long time now, most home computers have come with at least 2, and recently even 4, and 8 cores of CPUs.
But despite having these extra cores or processors have not been able to speed up computers. It does not give much gain in the way of appreciable benefits to computer users. It is also not clear if they could be of much use in coming future.

The main point about multiple cores CPU is that the programmer has to write software in such a way that it takes advantages of the multiple cores or processors provided on the CPU chip. So this put lot of pressure on the programmer, because in contrast of doing programming in a traditional step by step process, or serially, programmers now need to ensure that their apps are developed in such a way that it works many jobs or tasks in parallel (a kind of multithreading). However, this is proving to be a very difficult and complex job for an average programmer.

A direct consequence of Moore's Law is the Speed vs. Power Dissipation factor. With clean and green computing taking prime importance, chip manufacturers are ensuring their products consume less power, without sacrificing performance. It can be seen that chip makers have stopped increasing the clock speed. For example both Intel & AMD are not increasing clock speed appreciably. These chip makers are in fact increasing computational units per chip to increase the computational power of the CPU.

In essence, from the above facts we understand two important observations:

1. CPU chips annual speed up is not growing every year now
2. CPU (multiple) cores or processors are becoming more in number but are largely under-utilized.

We need an alternative, one that fills most of these loop-holes efficiently, and promises to sustain itself in the years to come: the Graphics Processing Unit, or simply, the GPU.

C. Cuda Fundamentals

There is a fairly minimal amount of terminology that is needed to help understand the programming model used in the CUDA framework.

- An individual GPU will be referred to as a device.
- The CPU will be referred to as a host.

With respect to NVidia's G80 GPU chip, it appears the computation of a grid, block and thread is distributed as follows:

- Grid → GPU: An entire grid is handled by a single GPU chip.
- Block → Multiprocessor: The GPU chip is organized as a collection of multiprocessors (MPs), with each multiprocessor responsible for handling one or more blocks in a grid. A block is never divided across multiple MPs.
- Thread → Stream Processor: Each MP is further divided into a number of stream processors (SPs), with each SP handling one or more threads in a block

The GPU differs from a CPU in its ability to process dozens of thousands of threads simultaneously. Each thread is scalar; and does not require packing data into 4-component vectors, which is more convenient for most tasks. The number of logical threads and thread blocks surpasses the number of physical execution units, which gives good scalability for the entire model range.

- Each thread uses IDs to decide what data to work on:
- Block ID: Blocks can be either one-dimensional or two-dimensional.
- Thread ID: Threads can be referenced either in one, two or three dimensions.

This feature of allowing the programmer to visualize the arrangement of threads greatly simplifies memory addressing when processing multidimensional data. This finds itself a particularly useful feature in applications such as:

- Image processing
- Solving PDEs on volumes

IV. OUR APPROACH

The basic approach in coding for the host part is as given below:

1. Define the functions for Convolution.
2. Input the image in the form a random matrix to be convolved. We call this image matrix
3. Define pointers to the kernel, image matrix and the resultant convolved image matrix.
4. Allocate memory dynamically on the Host using malloc() function,
5. Convert image matrix data into a 2D array whose contents can be manipulated.
6. Define the Kernel.
7. Perform Convolution on each component.
8. Measure the time taken for the calculation, and display the total time taken for the entire image data.
9. Write the convolved data in into output Image Matrix array.
10. Free the pointers to kernel, source image, and resultant arrays from the CPU memory for other activities.

A. How does our system work?

The following figure shows the system diagram for our idea.

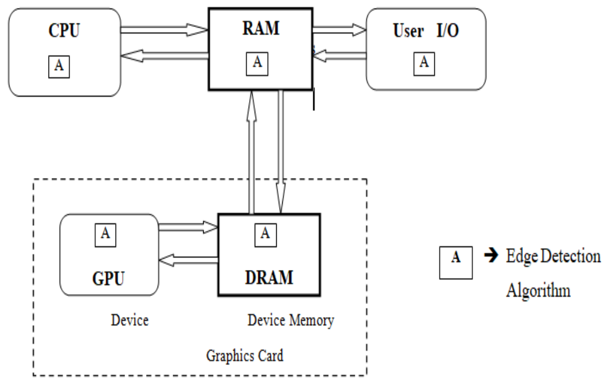

Figure 1. System diagram for GPU based Edge detection technique

Our system is a heterogeneous system since it has two different kinds of processor: CPU and GPU. Building application on Heterogeneous architecture using CUDA includes the application's data flow in both CPU and Graphics card. The figure explains the complete architecture. As can be seen from this figure we can offload expensive operation in Gradient calculation on the GPU, while all less costly jobs are performed on a CPU. Of course we need to tweak the algorithm so that it matches that of the GPU and becomes compatible to it.

B. The approach

The basic approach in coding for the host part is as given below:

1. Define the functions for Convolution.
2. Input the image in the form a random matrix to be convolved. We call this image matrix
3. Define pointers to the kernel, image matrix and the resultant convolved image matrix.
4. Allocate memory dynamically on the Host using malloc () function,
5. Convert image matrix data into a 2D array whose contents can be manipulated.
6. Define the Kernel.
7. Perform Convolution on each component.
8. Measure the time taken for the calculation, and display the total time taken for the entire image data.
9. Write the convolved data in into an output Image Matrix array.
10. Free the pointers to kernel, source image, and resultant arrays from the CPU memory for other activities.

C. The parallel approach

In general, GPUs solve to 2D image processing problem as a special case of 3D image processing task. In graphics everything is handled as polygons and in case of 2D processing a quadrilateral polygon is aligned to the desired image-screen rectangle and rendered. This rendering could be onto the screen or into the frame buffer. There are millions of transistors on the NVIDIA GPUs, which can be dynamically allocated for various graphics related operations. Such operations include geometry and pixel computation

D. The algorithm

Our basic approach has been shown on the following figure:

1. First we put a portion of the image into a device memory data structure,

2. Then we do a point-wise multiplication of a portion of the data which is equal to the size of the filter. This is what we do in parallel by many threads.

3. The final step is then to put this addition into the output image matrix in DRAM of card.

It may be noted that as per the above scheme individual thread block handles 1 block in the matrix. Thus every individual thread is responsible for generating a single output pixel.

V. RESULTS AND ANALYSIS

A. Results

In this section we present some of the results obtained so far in our research. Our implementation of the gradient via convolution process on the GPU (GT 640) yields the results as given in Table 1:

Table 1. Timing in ms for GPU implementation for different 3 image sizes and radius

Kernel Radius	Time (ms) (Size=256)	Time (ms) (Size=512)	Time (ms) (Size=1024)
1	0.48	1.447	4.64
2	1.488	2.89	11.335
4	3.97	8.842	36.14
8	9.25	32.11	120.8

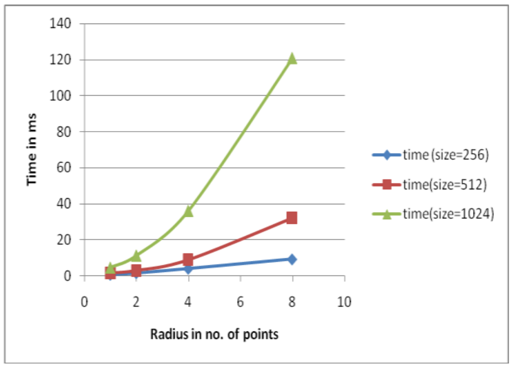

Figure 2. Performance of GPU implementation (KSize=8).

In Figure 1 we show the performance of a GPU implementation for Kernel Size=8.

Table 2. Timing in ms for GPU implementation for 3 different image size and radius

Size	CPU time(ms)	GPU time(ms)	Speed-up
256	53.96	9.7	5.56
512	223	33	6.75
1024	908	120	7.6

B. Analysis

Table 1 show that as we increase the size of the image or the size of the kernel, the computation time on even a GPU is increasing almost exponentially. We infer from the Table 2, however, that the GPU can process data approximately 8 times faster than the Dual Core CPU, courtesy – the massively parallel architecture. We observed that at least for larger size images the gain is quite appreciable. This ultimately means that a report that is generated normally in 60 minutes will now be available in 8 min. For small size kernel and images it looks like there is large overhead and the gain is not much.

We observe certain limitations in our kernel for gradient calculation that we have written for a GPU. (We have not used shared memory in our implementation, but we believe can be used for improving performance). We list some of these based on our observation:

1. It can be observed that for a practical given filter kernel size, the points on the boundary of the shared memory (SM, if we use shared memory at all) array will depend on points not available in the memory segments considered. As can be seen around the image portion inside a thread block, we must have an extra portion of the points with width of the r i.e. kernel radius. This is a must have for filtering the particular image block. Therefore, every individual thread block has to load extra pixels in such situations. Our implementation has not taken this into account this consideration.

2. If we account for those extra pixels as discussed above, we need to launch extra threads. These extra threads will not participate in calculations post loading of the data. Thus, it will be wastage of resources.

3. CUDA provides optimized specialized mathematical operations to run in less clock cycles. We have not used these operations in our code. For e.g., the mul24() multiplies the lower

24 bits of the 32 bit integers, in 4 clock cycles, instead of the normal 16 clock cycles for 32 bit integer multiplication. However, care has to be taken so that, the numbers being multiplied occupy less than 24 bits, otherwise valid data is lost, with the multiplication yielding incorrect results. An implementation with such special operations optimized for GPU architecture is expected to give more performance advantage as compared to without using it.

VI. CONCLUSION

This paper introduces our method for edge detection by employing a GPU. The edge detection is a time consuming operation, especially in case of high resolution medical images. Edge detection is studied as a multi-stage process, in which the compatibility between boundary and edge is emphasized. We conducted a study based on image edge detection methods which provide insight into more widely used as edge detection techniques. We have described edge detection techniques such Gradient-based Laplacian Robert, Prewitt, Sobel, Canny detection methods. The gradients in such algorithms can be approximated using convolution operation and is often implemented using the same. However, since such gradient or convolution operations are computationally complex and computationally costly we proposed using a GPU to achieve our real-time goal. A real time or a faster response is of great help to a doctor for a faster diagnosis. A GPU is basically a massively parallel processor and consist of thousands of cores. It was originally meant for graphics applications, but thanks to CUDA platform, it became possible to use it for non-graphical applications such as ours. Our algorithms run in parallel on these GPU cores (to increase performance) in parallel. The alternatives to the GPU include FPGA and ASICS, but these are 100s of time costlier and also involve very time consuming tasks for developing even a very simple system.

To prove our point we implemented this time consuming operations i.e. gradient calculations using image convolution on a GPU. While doing so, we first studied the architecture of the GPU in detail and then changed the algorithm of trivial serial operation and made it a parallel one, compatible to GPU architecture. Finally, we compared the GPUs performance with a CPU implementation and found several times speedup (around 8 times for large images) as compared to the normal serial CPU implementation. A faster response means a quicker availability of reports for the Doctors inspection and diagnosis. In our case a 60 minute report will now be available in 8 minutes, thereby considerably saving the time of the doctor and patient. Total time saved for a large number of patients and thereby the gains can be accordingly estimated.

REFERENCES

[1] Marr and E. Hildrith, “Theory of Edge Detection,” Proc. Royal Society of London, B207, pp. 187–217, 1980.

[2] James Clerk Maxwell, DIGITAL IMAGE PROCESSING Mathematical and Computational Methods.

[3] R .Gonzalez and R. Woods, Digital Image Processing, ,Addison Wesley, 1992, pp 414 - 428.

[4] S. Sridhar, Oxford university publication. , Digital Image Processing.

[5] Shamik Tiwari , Danpat Rai & co.(P) LTD. “Digital Image processing”

[6] J. F. Canny. “A computational approach to edge detection”. IEEE Trans. Pattern Anal. Machine Intell., vol.PAMI-8, no. 6, pp. 679-697, 1986 Journal of Image Processing (IJIP), Volume (3) : Issue (1)

[7] Geng Xing, Chen ken , Hu Xiaoguang “An improved Canny edge detection algorithm for color image” IEEE TRANSATION ,2012 978-1-4673-0311-8/12/$31.00 ©2012 IEEE.

[8] Punarselvam, E., & Suresh, P. (2011). Edge Detection of CT scan Spine disc image using Canny Edge Detection Algorithm based on Magnitude and Edge Length. 3rd International Conference on Trendz in Information Sciences & Computing (TISC2011). doi:10.1109/tisc.2011.6169100

[9] Nikolic, M., Tuba, E., & Tuba, M. (2016). Edge detection in medical ultrasound images using adjusted Canny edge detection algorithm. 2016 24th Telecommunications Forum (TELFOR). doi:10.1109/telfor.2016.7818878

[10] Chang, C., & Kehtarnavaz, N. (2015). Computationally efficient image deblurring using low rank image approximation and its GPU implementation. Journal of Real-Time Image Processing, 12(3), 567-573. doi:10.1007/s11554-015-0539-x

[11] Sher Jung, Rajendra Kumar Sharma, GSZRP: Graphics-hardware based Optimized Secure Zone Routing protocol, IJARSE (ISSN: 2319-8354),Volume No.06, Issue No. 12, December 2017

[12] Agarwal, N., Nellans, D., Ebrahimi, E., Wenisch, T. F., Danskin, J., & Keckler, S. W. (2016). Selective GPU caches to eliminate CPU-GPU HW cache coherence. 2016 IEEE International Symposium on High Performance Computer Architecture (HPCA). doi:10.1109/hpca.2016.7446089

[13] “Convolution,” Wikipedia, 20-May-2018. [Online]. Available: http://en.wikipedia.org/wiki/Convolution. [Accessed: 23-May-2018].

[14] R. Farber, “CUDA, Supercomputing for the Masses: Part 1,” Dr. Dobb's. [Online]. Available: http://www.drdobbs.com/high-performance-computing/207200659. [Accessed: 23-May-2018].

[15] “Weather, Atmospheric, Ocean Modeling, and Space Sciences,” NVIDIA. [Online]. Available: http://www.nvidia.com/object/weather.html. [Accessed: 23-May-2018].

[16] “The AI Computing Company | NVIDIA.” [Online]. Available: https://www.bing.com/cr?IG=746A37FE329B48A9922870D343DAB2B1&CID=0CAFD34A11B26EA30FA3D8B2104F6F1E&rd=1&h=FZfALwVHxE_S9H2WNKhtElNoV47kgQDk68vgEU4&v=1&r=https://www.nvidia.com/en-us/about-nvidia/ai-computing/&p=DevEx.LB.1,5549.1. [Accessed: 23-May-2018].

Authors Profile

Ms. Saima Iram received the Bachelor of Technology degree in computer science and engineering from Priyadarshini College of Computer Sciences, Greater Noida, G.B.Nagar, Uttar Pradesh in 2013 and currently pursuing Master of Technology in Computer Science & engineering from Noida Institute of Engineering & Technology, Greater Noida, Uttar Pradesh .Her research area is digital image processing.

Mr. Jabir Ali received the B.Tech degree in Information Technology from Invertis Institute of engineering & technology, Bareilly in 2009, and M.Tech degree in Computer Science & engineering from Jaypee University of Information Technology, Solan, India, in 2011 and he joined Translam institute of technology & management in 2011 and Sunderdeep Group of colleges as an Assistant Professor in 2012. In 2013, he joined the Jaypee university of information Technology, as a research scholar and submitted his Ph.D. in 2018. Currently he is working in Noida Instititute of Engineering & Technology. His current working domain is in Image and Video copyright protection. His research interest includes image security, Video watermarking, Signal Processing and Network Security.