IJCSE International Journal of Computer Sciences and Engineering Open Access
Research Paper Vol-6, Special Issue-4, May 2018 E-ISSN: 2347-2693

Parallel Processing Edge Detection Methods for MR Imagery Volumes using CUDA Enabled GPU Machine

P. Sriramakrishnan[1], T. Kalaiselvi[2]* and K. Somasundaram[3]

[1,2,3]Dept. of CSA, The Gandhigram Rural Institute (Deemed to be University) Gandhigram, Tamil Nadu, India.

*Corresponding Author: kalaiselvi.gri@gmail.com, Tel.: 9486362843

Available online at: www.ijcseonline.org

***Abstract*—** Many algorithms in the field of image processing support high degree of inherent parallelism. Edge detection is one of the most important processes in medical image processing. Edge detection is an independent process to support parallel computation of each pixel intensity changes by their neighbourhood pixels. Magnetic resonance imaging (MRI) scanner provides stack of 2D slices with millions of pixels and thus require much time for edge detection process in central processing unit (CPU) systems. In the proposed work, graphics processing unit (GPU) based parallel edge detection methods are developed for MRI volume using compute unified device architecture (CUDA). Each pixel operation in edge detection is independent and thus GPU provides high level data parallelism using threads per voxel method. Basic edge detection operators such as Roberts, Prewitt, Sobel, Marr- Hildreth and Canny are used in this experiment. The computational time of parallel GPU-CUDA based methods were compared with the serial CPU implementation. Results showed that parallel implementation is about 11× to 98× times faster than the serial CPU implementation.

***Keywords*—** Edge detection, GPU, CUDA, Roberts, Prewitt; Sobel, Marr- Hildreth, Canny.

I. INTRODUCTION

Magnetic resonance imaging (MRI) provides the huge amount of data about the soft tissue characteristics of human body that helps to improve the diagnosis more accurate and reliable than other imaging modalities [1]. Manual diagnosis of MR images is a time consuming and painstaking task to the clinician due to large number of 2D slices produced by the MRI scanner [2]. In recent years, rapid development of semiconductor results the integration of much transistors in a single unit and named as GPU in order to solve the computational problems. GPU programming is a new technique to improve the speed and quality of the medical image applications [3]. CUDA is a platform for GPU programming to obtain data parallelism using single instruction multiple threads (SIMT) concept.

Edge detection is a pre and post processing technique for extracting features for classification, object segmentation and visualization in medical image processing [4]. Edge detection is an independent process to each pixel and computed from intensity changes of its neighborhood pixels. Edge detection in MRI volume is an expensive process in CPU. To reduce the computational time, we proposed parallel edge detection methods for MRI volume using CUDA.

Currently, several parallel algorithms were developed for edge detection using GPU-CUDA. Gong and Hao implemented a parallel edge detector for Roberts operator using GPU [5]. The experiment showed that the method given ten times faster than traditional Roberts edge detector implemented in CPU. Jain et al., proposed a method parallel for Sobel operator [6]. They achieved the speed gain upto 262× – 943× for two kernel functions and 120× – 455× for three kernel functions. Hossain et al. developed a parallel method for sobel edge detector using NVIDIA GTX 550Ti GPU [7]. The computation time of the parallel method was compared with convention CPU time and accelerated upto 4×.

Ogawa et al., proposed a efficient implementation of Canny edge detection on GPU using CUDA [8]. The experimental results showed that the implementation of parallel Canny edge detection achieved 61× faster than the serial implementation in CPU. Cheikh et al. implemented a parallel Canny edge detector in multi-core CPU and many-core GPU using the matrix laboratory (MATLAB), C++, Open Multi-Processing (OpenMP) and CUDA [9]. The parallel method reduced the computational time upto 33×. Emrani et al., developed a parallel method for various edge detection techniques using CUDA, open source computer vision (OpenCV), and MATLAB [10]. The comparison of results showed that the parallel Canny method on GPU using the CUDA platform given 2× – 100× speeder than CPU based implementation using the OpenCV and MATLAB platforms.

This paper proposed parallel edge detection methods for 3D MR volumes using GPU-CUDA computing. In edge detection process, pixel operations are independent from each other and GPU can provide high level of data

parallelism using threads per voxel. The popular basic edge detectors such as Roberts, Prewitt, Sobel, Marr- Hildreth and Canny are used in this experiment. NVIDIA Quadro K5000 GPU has Kepler architecture used to implement these parallel edge detection methods. The computational time of parallel methods were compared with the serial implementation using speedup folds metrics (×).

The rest of this paper is organized as follows; Section II presents the introduction about GPU programming and CUDA computing. Section III describes the traditional edge detection methods; Section IV reports parallel implementation of edge detectors; Section V discusses the results of this experiment; Finally, section VI concludes the paper.

II. GPU-CUDA

GPU has massively parallel computing architecture that connected to CPU and completely separated from motherboard. GPU architecture includes M number of streaming multiprocessors (SM), N number of co-processor per SM, ALU's, control units and various types of memories to support data parallelism [11]. NVIDIA introduced its own GPU programming language called CUDA in 2006. CUDA is an extension of the C programming and executed in either host (CPU) or device (GPU) with their memory support. Host code does not support data parallelism and device code exhibit rich amount of data parallelism.

Programming architecture of GPU – CUDA is shown in figure 1. CUDA is capable to execute large amount of parallel threads and the threads are organized into grids and thread blocks. Device code executes the thread into two level of parallelism which includes parallel block in grid and parallel thread in the block. Each block (blockIdx) and thread (threadIdx) has a unique ID. The total number of threads in a block is 1024.

Figure 2 shows the memory architecture of GPU – CUDA. GPU memories are register memory, local memory, shared memory, global memory, constant memory and texture memory [12]. Threads have their own limited register and local memory. Frequently accessed variables are stored in registers. Every block has its own shared memory of size 16 KB or 48KB and all threads in a block can access the shared memory. Constant memory is a read only memory that can be accessed by all threads in a grid. Texture memory is used in visualization process and the size is 32 KB per multiprocessor. Global memory is a largest memory that provides read and write access to all threads. The global memory supports transfer the data to and from the device for computation.

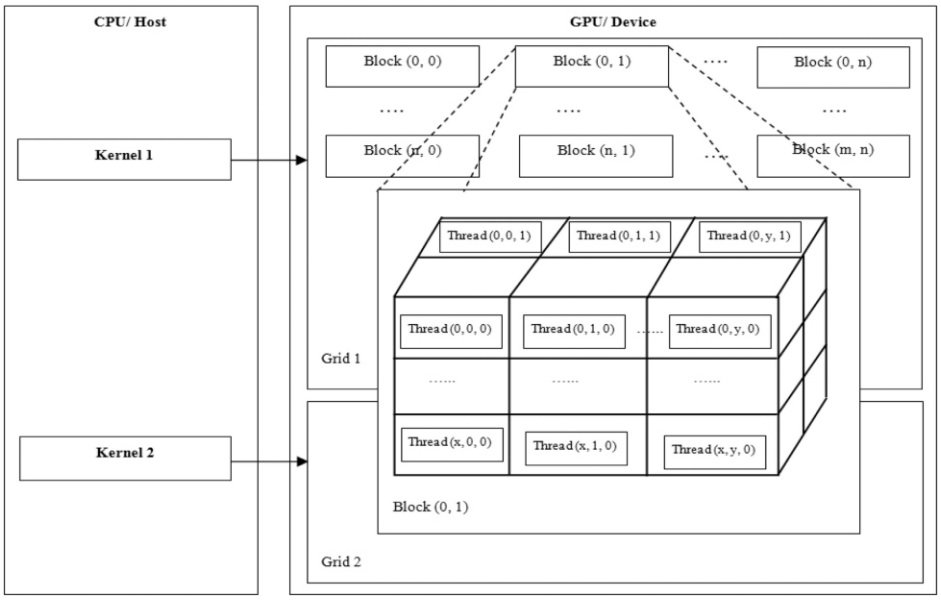

Figure 1. Programming architecture of GPU- CUDA

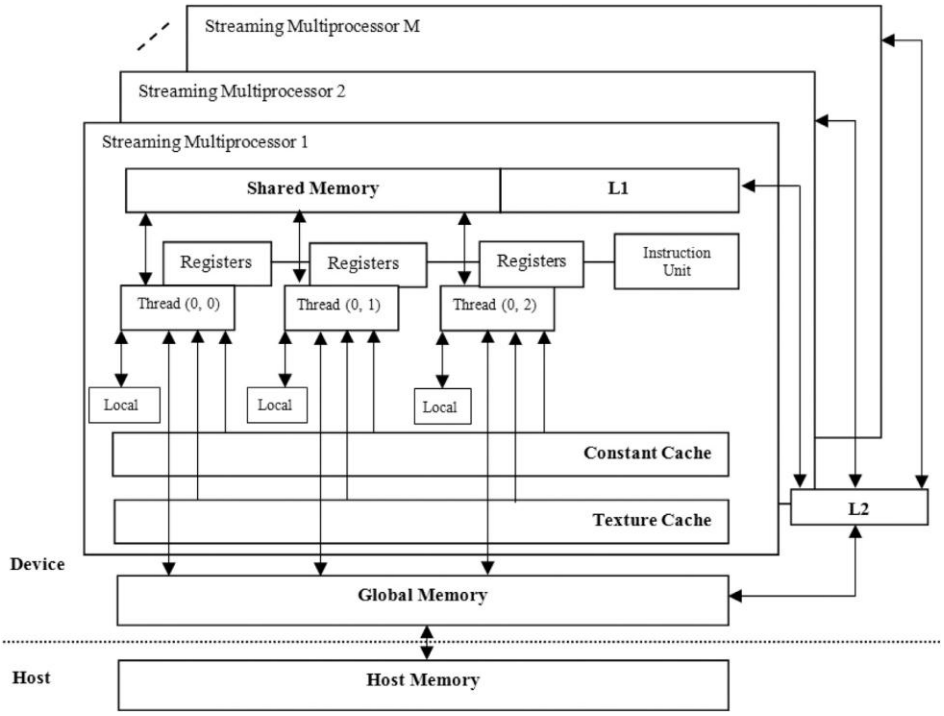

Figure 2. Memory architecture of GPU-CUDA

III. EDGE DETECTION METHODS

Edges are the set of connected pixels occurs at the boundary between two different regions of an image. Edge detection always demands in the field of object identification, feature extraction and classification. Edges can be obtained using first order derivatives or gradient and mathematically defined as follows:

$$\nabla I(i,\ j) = \hat{i}\frac{\partial I(i,j)}{\partial i} + \hat{j}\frac{\partial I(i,j)}{\partial j} \tag{1}$$

where, $I(i,\ j)$ be the input image, $\frac{\partial I(i,j)}{\partial i}$ is the gradient (G_i) in the horizontal direction and $\frac{\partial I(i,j)}{\partial j}$ is the gradient (G_j)in the vertical direction.

The gradient magnitude and direction can be defined as:

$$|G| = \sqrt{G_i^2 + G_j^2} \tag{2}$$

$$\theta = \tan^{-1}\left(\frac{G_j}{G_i}\right) \tag{3}$$

Convolution of an image I by a kernel G is given by

$$I'(v,u) = \sum_{(i,j)\in R_H} I(u-v, v-i).G(i,j) \tag{4}$$

Finally a threshold is used to determine the edges from $I'(u,v)$. The following briefly discusses the basic edge detection methods used for the experiment.

A. *Roberts Cross Edge Detector*

Lawrence Roberts proposed an edge detector operator in 1965 [13]. The operator used two 2 × 2 kernels to detect the intensity changes in the horizontal and vertical direction. The Roberts filter operator approximates the intensity gradient of the brightness using following two different kernels:

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \qquad G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{5}$$

where, G_x and G_y are horizontal and vertical kernels.

B. *Prewitt Edge Detector*

Prewitt edge detector proposed by Judith M.S. Prewitt in 1970 [14]. This operator includes two 3 × 3 kernels which are convolved with the original image to calculate the changes in the horizontal and vertical directions. Two kernels are:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \tag{6}$$

C. *Sobel Edge Detector*

Sobel operator consists of couple of 3 × 3 kernels like Prewitt. The kernels undergo a convolution with the image to calculate the approximations of horizontal and vertical derivative.

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \qquad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{7}$$

D. Marr-Hildreth Edge Detector

Marr and Hildreth proposed an edge operator combined with Gaussian filter and Laplacian operator [15]. This gradient based operator uses the Laplacian to take the second derivative and thus called as Laplacian of Gaussian (LOG). This operator marked a pixel as an edge where the second derivative of the kernel is zero. Second derivative computed as:

$$\nabla^2 I(i,\ j) = \frac{\partial^2}{\partial i^2} I(i,j) + \frac{\partial^2}{\partial j^2} I(i,j) \tag{8}$$

The Gaussian filter can be defined as:

$$G(i,j) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{i^2 + j^2}{2\sigma^2}\right) \tag{9}$$

LOG operator is computed from

$$LOG = \frac{\partial^2}{\partial i^2} G(i,j) + \frac{\partial^2}{\partial j^2} G(i,j) \tag{10}$$

where, σ is the standard deviation.

LOG having two steps are smooth the image using Gaussian filter and convolute the image using Laplace kernel. The 3× 3 kernel defined as

$$G = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{11}$$

E. Canny Edge Detector

Canny edge detector is a well known edge detection algorithm in image processing which is derived from Marr-Hildreth [16]. This having some features such as well localized edge points, minimum spurious edge response and only one response to a single edge. Canny edge detection consists of four steps: Gaussian smoothing, gradient magnitude computing, non-maxima suppression, hysteresis thresholding.

In step 1, Gaussian smoothing is used to remove the noise before taking the edge from an image. It has two dimensional kernel to convolute the image and thus reduces the noise. In step 2, Sobel filter with 3 × 3 kernel is used to compute the gradient magnitude of an image. Intensity changes in horizontal and vertical direction computed by the kernel using Eq. (7). Gradient magnitude (G) of edge for each pixel computed using Eq. (2). Third step detected the thin edges from an image and local maximum along with the gradient direction. Step four is used to detect the final edges based on two threshold values and categories the edges into strong edge, weak edge and non edge. Finally edge pixels determined from strong and weak edges.

IV. PARALLEL EDGE DETECTION

Essential needs for parallel implementation of edge detection algorithms are discussed. Edge detection is an independent process to each pixel and supports the parallelization using threads. This section discusses the parallel edge detection algorithms such as Roberts, Prewitt, Sobel, Marr- Hildreth and Canny. Generally, edge detection algorithms convolute the two kernels over the image in horizontal as well as vertical direction. Serial implementation this process takes a considerable time using CPU. Further, MR brain volumes contain numerous slices that required more time to implement the edge detection process.

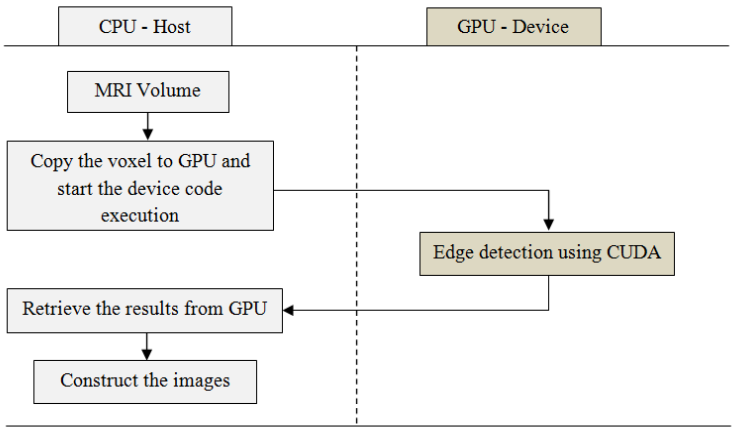

Figure 3. Block diagram of the proposed method for parallel edge detection using GPU

Parallel method creates the threads which are equal to number of voxel in the MR brain volume in order to compute the edge detection process using GPU CUDA. Edge of all pixels in an image as well as all the images in a volume could be computed simultaneously. The method effectively utilizes the hardware resources available in GPU. Parallel implementation mechanism of the proposed work for edge detection algorithms are given in this section. Block diagram of the parallel edge detection process is shown in figure 3.

MRI volume is taken as the input which contains nearly 120 brain images. The voxel intensity data are stored in the form of one dimensional array and transfer to the GPU memory using memcpy() function in CUDA programming. All the above processes are implemented in CPU and then start the device code for parallel execution and produce the results. Threads are simultaneously access the instructions in device code and called as single instruction multiple data (SIMD). Final results are copied back to the host for further processing.

Device code contains kernel operation to compute edge pixel. The kernel helps the parallel threads to access each pixel and compute the edge value. Computed results are sending back to the CPU from GPU in order to construct the resultant image. Most common suggested number of threads per block is 256 [17]. Heterogeneous CPU and GPU implementation is more suitable for time complex algorithms [2]. The generalized algorithm for edge detection using GPU is explained below. Remaining part of this section discusses the parallel implementation details of each edge detection algorithms.

Algorithm:- 1 Edge detection using GPU CUDA

Input: N input images, edge detection kernel (m × m), number of images (N) and size of image (weight and height)

1. Create an array for input (IN) and output (H ,V)
2. Store the image intensities to 1D array
3. Allocate the memory in GPU
4. Copy the data from host to device
5. Calculate the number of blocks and threads
6. Start the GPU execution
7. i = threadIdx.x + blockDim.x * blockIdx.x;
8. **if** i > 0 and i < N*width*height **do**
9. compute the kernel operator
10. **end if**
11. Resultant intensity stored in H and V arrays
12. Copy the data from device to host
13. Construct the images

Output: MRI volume with edges

Thread can access the pixel intensity in parallel and compute the Roberts operator to detect the edges. Parallel implementation of Roberts edge detection broken down to three steps. First, input MR image intensity s stored in IN array and copy to the GPU video memory. Second, the computation is done using two 2 × 2 kernels and sum up the values of kernel operator to calculate the edges. Then the two kernels values in horizontal and vertical direction are stored in H and V array. Third, the resultant H and V arrays are transferred back to the CPU memory.

A 2D separable filter can be divided into two 1D filter and that require $m+n$ operations instead of $m*n$ to produce each output pixel. Here m and n are the width and height of the kernel. If a kernel (G) is called separable, it can be broken down into convolution of two kernels:

$$G = G_1 * G_2 \tag{12}$$

Sobel and Prewitt operators are separable and Sobel kernel can be written as:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -1 \\ -2 \\ -1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \tag{13}$$

The separable filters with 3 × 3 kernel reduce the operations from O (mn) to O ($m+n$). Here six operations performed instead of nine and it will be effective if kernel size is increases.

Prewitt and Sobel operator implemented using the separable concept in GPU. Therefore, the convolution is performed in the x direction first followed by the y direction. These two steps are carried out by a device code [18]. LOG doesn't support the separable filter and implemented like Roberts operator.

Two device codes used to implement the parallel canny edge detection. Parallel Canny used heterogeneous CPU and GPU implementation. Gaussian filter, non-maximum suppression and threshold detection process are implemented in CPU. Sobel operate convolute over an image in parallel using GPU. Finally resultant edge pixels are transferred back to the CPU memory for reconstructing the images.

V. PARALLEL EDGE DETECTION

The proposed implementation is similar to that offered by MATLAB and further translated the MATLAB function to C++ and implemented it in CUDA. Our multi-core CPU platform is a Pentium dual core, where each processor has 2 cores working at 2.3 GHz. Our GPU platform is the NVIDIA Quadro K5000 developed under Kepler architecture. This platform includes 1536 streaming processors (SP) or cores distributed on 8 streaming multiprocessors (SM) as 192 SP per SM. Each core is working at 1.4 GHz, and 4GB. This GPU supports compute capability 3.0, 1024 threads per

block, 173 GB/s memory bandwidth, 3540 million transistors and 49152 registers per block.

The materials used for this work is obtained from multimodal brain tumor segmentation (BraTS 2015) training datasets. The dataset contains 120 brain tumor images with the size of 240 × 240 pixels. All the images in the dataset are skull stripped and resampled to isotropic 1mm × 1mm × 1mm resolution. The parallel method extracted the edges of given MR tumor volume. The result of edge detection algorithms on sample tumor image are given in figure 4. The resultant images showed the exact boundary of the tumor to assist the clinicians in diagnosis process. The speedup metrics used to measure the acceleration of parallel method in terms of folds (×). The speedup folds computed from the time difference taken between serial implementation and parallel implementation. In the time calculation, image read operations are not included in CPU and GPU implementation.

$$Speedup\ folds\ (\times) = \frac{Computation\ time\ of\ serial\ implementation\ in\ CPU}{Computation\ time\ of\ parallel\ implementation\ in\ GPU} \quad (14)$$

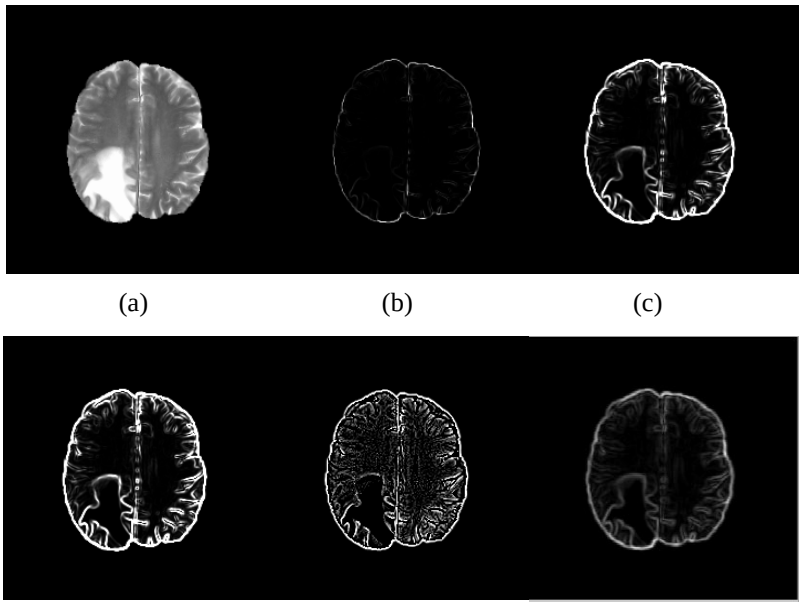

Figure 4 Results of edge detection algorithms on sample brain tumor image. (a) sample image (b) Roberts (c) Prewitt (d) Sobel (e) LOG (f) Canny

Table 1. Computation time taken by the edge detection algorithms for various size of volumes in CPU

Edge Detection Algorithms/ seconds	Number of Images				
	= 1	= 10	= 50	= 100	= 120
Roberts	0.105213	0.131241	0.24390	0.44556	0.455213
Prewitt	0.373870	0.425459	2.22890	4.38564	4.920487
Prewitt Separable	0.136870	0.220459	0.67510	1.89984	2.676887
Sobel	0.062857	0.418859	2.31260	4.27555	5.240987
Sobel Separable	0.017987	0.202959	0.74320	1.48034	2.322987
Marr-Hildreth	0.069557	0.329592	0.66910	1.09804	3.525287
Canny	12.160857	124.98045	370.2135	817.67934	1119.646057

Table 2. Computation time taken by the edge detection algorithms various size of volumes in GPU

Edge Detection Algorithms/ seconds	Number of Images					Maximum Speedup (×)
	= 1	= 10	= 50	= 100	= 120	
Roberts	0.000498	0.003372	0.014762	0.030775	0.041963	11
Prewitt	0.000729	0.004246	0.02211	0.053732	0.063942	77
Prewitt Separable	0.000649	0.002139	0.01538	0.03432	0.0588	46
Sobel	0.000431	0.002590	0.011677	0.041714	0.053394	98
Sobel Separable	0.003251	0.005783	0.00943	0.0341	0.0483	48
Marr-Hildreth	0.000641	0.003134	0.015313	0.029139	0.040742	87
Canny	1.5309	3.528	8.3720	13.452	16.4529	68

The computation time taken by the parallel method on GPU and their maximum speedup folds are given in Table 2. The results showed that the computational time taken by the parallel methods in GPU almost saturated when handling more number of images and threads. Graph comparisons of each method with their parallel implementation are given in figure 5.

Generally Roberts method has less computation with 2 × 2 mask convolute to an image. The results showed that the parallel implementation of Roberts method accelerated up to 11× speedup folds than the CPU implementation for detecting the edges of MRI volume. Prewitt operator with its 3 × 3 kernel took more computation time than Roberts. The performance of Prewitt increased 77 × speedup folds in GPU implementation. Prewitt separable kernel reduces the computation in CPU as well as GPU and yielded 46× speedup folds than the serial implementation. Sobel and Sobel separable kernel accelerate the computation time for MRI volume upto 98× and 48 × speedup folds respectively. Marr-Hildreth edge detection yielded 87× speedup folds than CPU.

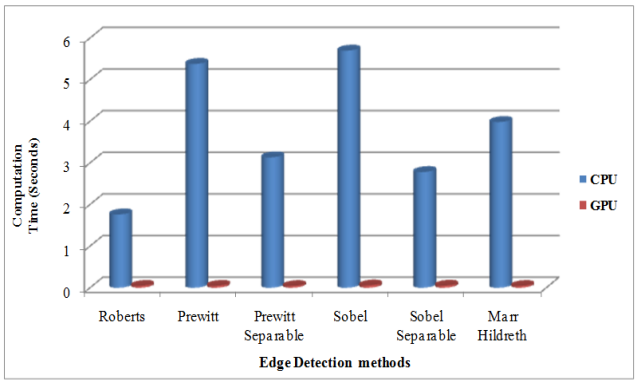

Figure 5. Computation time of Serial and Parallel implementation of edge detection algorithms in CPU and GPU

Graph comparison of Canny edge detection on CPU and GPU as shown in figure 6. Canny edge detector yields 87× and 68× speedup folds computational gain in GPU. Convolute the sobel kernel over the image only done in parallel implementation for Canny edge detection and remaining steps are implemented in CPU. Due to the bandwidth limitation of GPU, data transfer between CPU and GPU takes considerable time in computation. Finally the parallel implementation of this work achieved speed up execution by 98 × compared to the CPU based implementation.

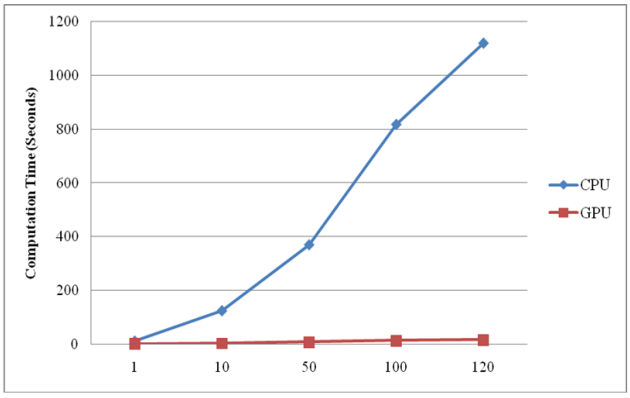

Figure 6. Graph comparison of Canny edge detection for varying size of volumes on CPU and GPU

VI. Conclusion and Future Scope

In this work, edge detection algorithms are described and implemented using MATLAB and CUDA platforms for MRI brain volumes. Five popular basic edge detection methods are presented and analyzed along with the possible optimization in CPU using separable kernel. The comparison of results obtained by these algorithms running in GPU CUDA achieved 98× speed folds than the CPU based implementation. The experimental results indicate parallel implementation of edge detector definitely yields high performance enhancement in medical images. In future, the proposed parallel methods will be tested in other general image based applications such as object identification, gait analysis, etc.

Acknowledgment

The authors gratefully acknowledge the support of NVIDIA Corporation Private Ltd., USA with the donation of the QUADRO K5000 GPU used for this research.

References

[1] J.L. Prince, J.M. Links, *"Medical Imaging Signals and Systems"*, Pearson Prentice Hall, 2nd Edition, 2014.

[2] D. Kirk, W. Hwu, *"Programming Massively Parallel Processors - A Hands-on Approach"*, 3rd Edition, pp. 1-514, 2016.

[3] T.Kalaiselvi, P. Sriramakrishnan, *"Rapid brain tissue segmentation process by modified FCM algorithm with CUDA enabled GPU machine"*, International Journal of Imaging System and Technology, pp. 1–12, 2018. DOI: 10.1002/ima.22267

[4] M. Chouchene, F.E. Sayadi, Y. Said, M. Atri, R. Tourki, *"Efficient implementation of Sobel edge detection algorithm on CPU, GPU and FPGA"*, International Journal of Advanced Media and Communication, Vol. 5, No. 2, pp. 105-117, 2014.

[5] H.X. Gong, L Hao, *"Roberts edge detection algorithm based on GPU"*, Journal of Chemical and Pharmaceutical Research, Vol. 6, No. 7, pp. 1308-1314, 2014

[6] A. Jain, A. Namdev, M. Chawla, *"Parallel Edge Detection By Sobel Algorithm Using CUDA C"*, In the proceedings of the International Conference on Electrical, Electronics and Computer Science, India, pp. 1-6, 2016.

[7] M. Kossain, M.A. Ashique, M.A. Ibtehaz, J. Uddin, *"Parallel Edge Detection using Sobel Algorithm with Contract-time Anytime Algorithm in CUDA"*, Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), CSREA Press, pp. 130-135, 2016

[8] K. Ogawa, Y. Ito, K. Nakano, *"Efficient Canny edge detection using a GPU"*, In the proceedings of First International Conference on Networking and Computing (ICNC), Japan, pp. 279-280, 2010.

[9] T.L.B. Cheikh, G. Beltrame, G. Nicolescu, F. Cheriet, S. Tahar, *"Parallelization strategies of the canny edge detector for multi-core CPUs and many-core GPUs"*, In the Proceedings of the International on New Circuits and Systems Conference (NEWCAS), Canada, pp. 49-52, 2012.

[10] Z. Emrani, S. Bateni, H. Rabbani, *"A New Parallel Approach for Accelerating the GPU-Based Execution of Edge Detection Algorithms"*, Journal of medical signals and sensors, Vol. 7, No. 1, pp. 33-42, 2017.

[11] T. Kalaiselvi, P. Sriramakrishnan, K. Somasundaram, *"Survey of using GPU CUDA programming model in medical image analysis"*, Informatics in Medicine Unlocked, Vol. 9, pp. 133 – 144, 2017

[12] F. Rob, CUDA application design and development, 1st Ed. Elsevier, pp. 1–336, 2011.

[13] L. Roberts, J. Tippet, *"Machine Perception of Three Dimensional Solids"*, Optical and Electro-Optical Information Processing, Cambridge, MA: IT Press; 1965.

[14] J.M. Prewitt, *"Object Enhancement and Extraction"*, Vol. 75, New York, Academic Press 1970.

[15] D. Marr, E. Hildreth, *"Theory of Edge Detection"*, Proceedings of the Royal Society of London, Series B, Biological Sciences, Vol. 207, No. 1167, pp. 187–217, 1980.

[16] J.F. Canny, *"A computation approach to edge detection"*, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 8, No. 6, pp. 769-798, 1986.

[17] S. Cuomo, P.D. Michele, F. Piccialli, *"3D data denoising via nonlocal means filter by using parallel GPU strategies"*, Computational Mathematical Methods in Medicine, pp. 1–14, 2014.

[18] V. Podlozhnyuk, *"Image convolution with CUDA*", 2007.

Authors Profile

Sriramakrishnan P. is a Research Scholar (Full-time) in the Department of Computer Science and Applications, Gandhigram Rural Institute - Deemed University, Dindigul, India. He received his Bachelor of Science (B.Sc.) degree in 2011 from Bharathidasan University, Trichy, Tamilnadu, India. He received Master of Computer Applications (M.C.A) degree in 2014 from The Gandhigram Rural Institute- Deemed University, Dindigul, Tamilnadu, India. He worked as Software Engineer in the Dhvani Research and Development Pvt. Ltd, Indian Institute of Technology Madras Research Park, Chennai during January 2014 – March 2015. He is currently pursuing Ph.D. degree in The Gandhigram Rural Institute - Deemed University. His research focuses on Medical Image Processing and Parallel Computing. He has qualified UGC-NET for lectureship in June 2015.

Kalaiselvi T. is currently working as an Assistant Professor in Department of Computer Science and Applications, The Gandhigram Rural Institute, Dindigul, Tamilnadu, India. She received her Bachelor of Science (B.Sc) degree in Mathematics and Physics in 1994 & Master of Computer Applications (M.C.A) degree in 1997 from Avinashilingam University, Coimbatore, Tamilnadu, India. She received her Ph.D degree from The Gandhigram Rural University in February 2010. She has completed a DST sponsored project under Young Scientist Scheme. She was a PDF in the same department during 2010-2011. An Android based application developed based on her research work has won First Position in National Student Research Convention, ANVESHAN-2013, organized by Association of Indian Universities (AUI), New Delhi, under Health Sciences Category. Her research focuses on MRI of human Brain Image Analysis to enrich the Computer Aided Diagnostic process, Telemedicine and Teleradiology Technologies.

Somasundaram K. received his Master of Science (M. Sc) degree in Physics from the University of Madras, Chennai, India in 1976, the Post Graduate Diploma in Computer Methods from Madurai Kamaraj University, Madurai, India in 1989 and the Ph.D degree in theoretical Physics from Indian Institute of Science, Bangalore, India in 1984. He is presently working as Professor at the Department of Computer Science and Applications, Gandhigram Rural Institute, Dindigul, India. He was senior Research Fellow of Council Scientific and Industrial Research (CSIR) Govt. of India, in 1983. He was previously a Researcher at the International Centre for Theoretical Physics, Trieste, Italy and Development Fellow of Commonwealth Universities, at Edith Cowan University, Perth, Australia. His research interests are image processing, image compression and medical imaging. He is also a member of IEEE USA.

