Satellite image processing using CUDA and Hadoop architecture

Helly M. Patel, Krunal Panchal, Prashant Chauhan, M. B. Potdar

Abstract—With the advancement in digitalization vast amount of Image data is uploaded and used via Internet in today's world. With this revolution in uses of multimedia data, key problem in the area of Image processing, Computer vision and big data analytics is how to analyze, effectively process and extract useful information from such data. Traditional tactics to process such a data are extremely time and resource intensive. Studies recommend that parallel and distributed computing techniques have much more potential to process such data in efficient manner. To process such a complex task in efficient manner advancement in GPU based processing is also a candidate solution. This paper we introduce Hadoop-Mapreduce (Distributed system) and CUDA (Parallel system) based image processing. In our experiment using satellite images of different dimension we had compared performance or execution speed of canny edge detection algorithm. Performance is compared for CPU and GPU based Time Complexity.

Index Terms— Hadoop, CUDA, Image processing, GPU, Map reduce, Distributes System, Parallel system, HPC

— — — — — — — — — — ✍ — — — — — — — — — —

1 Introduction

The rate of image data generation is exceeding much more faster than computational capability to process such data. Major challenges and research opportunity reside in the field of image processing and computer graphics to extract valuable facts from such data. Besides this currently used approaches to process such data are much more costly in terms of resource requirement (hardware and software) and require more execution time for processing.

Especially in case of satellite image processing to have efficient gain High Performance Computing (HPC) workstations are needed which are expensive. Emerging Distributed and parallel computing systems are having potential to process such data in efficient and less expensive manner. Hadoop Map reduce framework is gaining acceptance because of its scalable, fault tolerant and reliable nature to process data in distributed location. Storage and computational power of Hadoop map reduce is extensively used now a days to process larger amount of unstructured and structured data. CUDA on the other side is an emerging parallel computing technology for graphics processing using GPU. As it make use of GPU to process graphics, it provide higher execution speed by using threaded mechanism.

2 Related work

3 Background

3.1 Apache Hadoop

Apache Hadoop [5] [6] is popular, open source, scalable, distributed, java based programming model. Hadoop hides the complex details of parallelization, fault tolerance, data distribution, and load balancing from users [6]. Hadoop consists of two chief components: Hadoop MapReduce and Hadoop Distributed File System (HDFS) [5].

3.2 Hadoop Map reduce

MapReduce is a programming model for the parallel processing of distributed large-scale data [5]. MapReduce paradigm is computing framework of Hadoop composed of a map function that performs filtering and sorting of input data and a reduce function that performs a summary operation. HDFS is a distributed, scalable, and portable file system written in Java for the Hadoop framework, which provides high availability by replicating data blocks on multiple nodes [6]. The MapReduce job needs to undergo two types of machine to complete the process, JobTracker and TaskTracker. A cluster has only one JobTracker, on the NameNode node, which is responsible for scheduling work. And TaskTracker distributed in all DataNode node, is responsible for the execution of tasks [5].

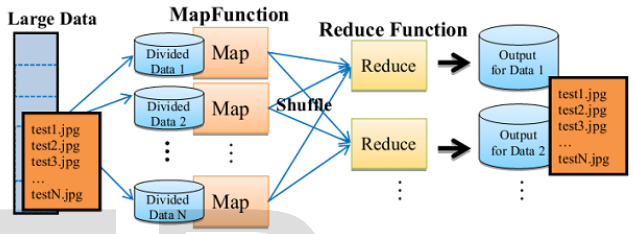

Fig. 1 Workflow in Map reduce Phase [2]

3.3 CUDA (Compute Unified Device Architecture)

A GPU card has many cores which are smaller than the ones present on the CPU but can execute many tasks in parallel [1]. Compute Unified Device Architecture (CUDA) [5] is a C based programming model from NVIDIA that exposes many core capability of GPU for easy development and deployment of general purpose computations [7]. In the CUDA context, the GPU is called as a device, whereas the CPU is called as host. A kernel is a set of computations that is offloaded by the CPU to be executed on the GPU [7]. A CUDA kernel is executed on the GPU by a grid of thread blocks, each consisting of a set of threads [7].

- *Helly Patel is currently pursuing master's degree in Computer engineering in L.J.I.E.T, India. E-mail: hpt1992@gmail.com*
- *Krunal Panchal is currently Assistant professor in computer engineering in L.J.I.E.T, India, E-mail: krunaljpanchal@gmail.com*
- *Prashant Chauhan is currently working as project scientist in Bhaskaracharya Institute for space and geo informatics ,Gandhinagar, Gujarat, India , E-mail:Prashant.mecs@gmail.com*
- *Dr. M. B. Potdar is currently working as project director in Bhaskaracharya Institute for space and geo informatics ,Gandhinagar, Gujarat, India , E-mail:mbpotdar@gmail.com*

Application processing steps on CUDA:

1. Setup the GPU and then Read the input

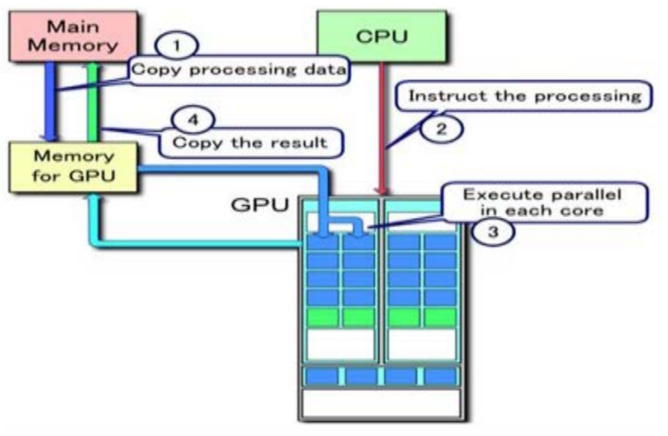

Fig. 2. Application processing flow in CUDA [8]

2. Copy data from main mem to GPU mem

3. CPU instructs the process to GPU

4. GPU executes it in parallel

5. Copy the result from GPU mem to main mem

6. Output the result [8]

3.3 Graphics Processing Unit (GPU)

GPUs are special processors that perform graphical task in a massively parallel manner and thus supplied high processing power [1]. It is most powerful and inexpensive computational hardware which is widely used in the field of Image processing. Massively-parallel threaded GPUs are used to achieve a higher degree of performance and energy efficiency. They can be regarded as massively parallel processors with 10x faster computation and 10x higher memory bandwidth than CPUs [1]. Currently, they are used as co-processors for the CPU.

4 Image Processing for Canny Edge Detection

Description and extraction of features from image is an important task useful for a wide range of application fields such as object recognition, image segmentation, data compression, land-water border etc. Edges in an image are signified by a significant image intensity change which represents important object features and boundaries between objects in an image. This multi step algorithm is considered as a standard and optimal detector among all edge detector algorithm.

Three main objective of algorithm is

1) Detection: The probability of detecting real edge points should be maximized while the probability of falsely detecting non-edge points should be minimized. This corresponds to maximizing the signal-to-noise ratio [8].

2) Localization: The detected edges should be as close as possible to the real edges [8].

3) Number of responses: Minimal number of edges should be detected more than once [8].

Canny's algorithm consists of five major steps:

I. Image smoothing

II. Gradient computation

III. Edge direction computation

IV. Nonmaximum suppression

V. Hysteresis.

I) Image smoothing

Smoothing serves the purpose of eliminating any sudden changes in intensity that may occur from phenomena such as image static, compression artifacts, camera problems, or bright reflections. It will smooth the image to eliminate the noise Eliminating these sudden changes reduces the number of falsely detected edges in the output [8]

II) Gradient Computation

It finds the image gradient to highlight regions with high spatial derivatives, the gradient of the image is used to determine the edge locations, which lie inside areas of high changes in intensity. Gradient computation for each direction is performed through convolving the smoothed image once with a mask that produces a horizontal gradient and once with a vertical gradient mask. The gradient magnitude for a location (X, Y) is obtained by adding the absolute values of position (X, Y) on the horizontal and vertical gradient image together, using the formula [8] $|\nabla| = |\nabla_x| + |\nabla_y|$

$$|\nabla| = |\nabla_x| + |\nabla_y|$$

Put all Equations using Insert Equation menu

III) Edge direction computation Edge direction is computed using the horizontal and vertical gradient images computed in the previous step. The direction for an edge located at location (X, Y) is computed by using the formula $\theta_{x,y} = \tan^{-1}(\nabla_x / \nabla_y)$

Edges are then classified as being in one of four directions by snapping them to their nearest positive 45-degree angle [8].

IV) Nonmaximum suppression

It is used to localize the edges down to a single pixel. Each pixel in the gradient is visited and its magnitude is compared to that of each of its two perpendicular neighbors. Every pixel that does not have a higher magnitude than its neighbors has its value set to zero, and all pixels that are local maxima are retained. Perpendicular neighbor locations are computed based on the edge directions that were previously computed [8]

V) Hysteresis

Many of the edges still present have very small magnitudes ad are not proper edges. Hysteresis is used to eliminate these edges. A high and a low threshold are used for the hysteresis operation. The resultant image from performing nonmaximum suppression is scanned and all pixels that have magnitudes greater than the high threshold are added to the output edge image. Each of the neighbors of a newly added pixel are recursively scanned and are added if they fall below a low threshold [8]

5 Experimentation

In this work, we implemented CPU and GPU based canny edge detection algorithm using Hadoop and CUDA framework respectively. Another is standalone CPU based java program. All three executed programs are compared to analyze performance execution on CPU and GPU.

Following tables shows software and hardware configuration used in experiments.

Table 1

Software configuration of node

Components	*Configurations and Releases*
OS	Ubuntu 15.04 LTS
JDK	1.7.0_79
Hadoop	2.4.0
CUDA	7.5

Mention no of Hadoop Nodes

Table 2

GPU key parameters

CUDA/GPU Specification	*Value / Description*
Name	GeForce GTX 750 TI
Number of Streaming Processors (SMs)	640 CUDA core
Core speed	1020 MHz
Memory	2 GB of GDDR5
Memory clock	5.4 GBPS
Standard Memory configuration	2048 MB

Experiment is performed on publically available satellite image dataset. Performance comparison is done for different for different image dimension. Following figures shows results for canny edge detection algorithm in different images for size. Performance is compared for execution time for different platform. In all the images shown in fig 3. , first row shows original images, second row shows image processed by CPU and third row shows GPU execution result. Table 3 shows comparative analysis of execution time of each algorithm.

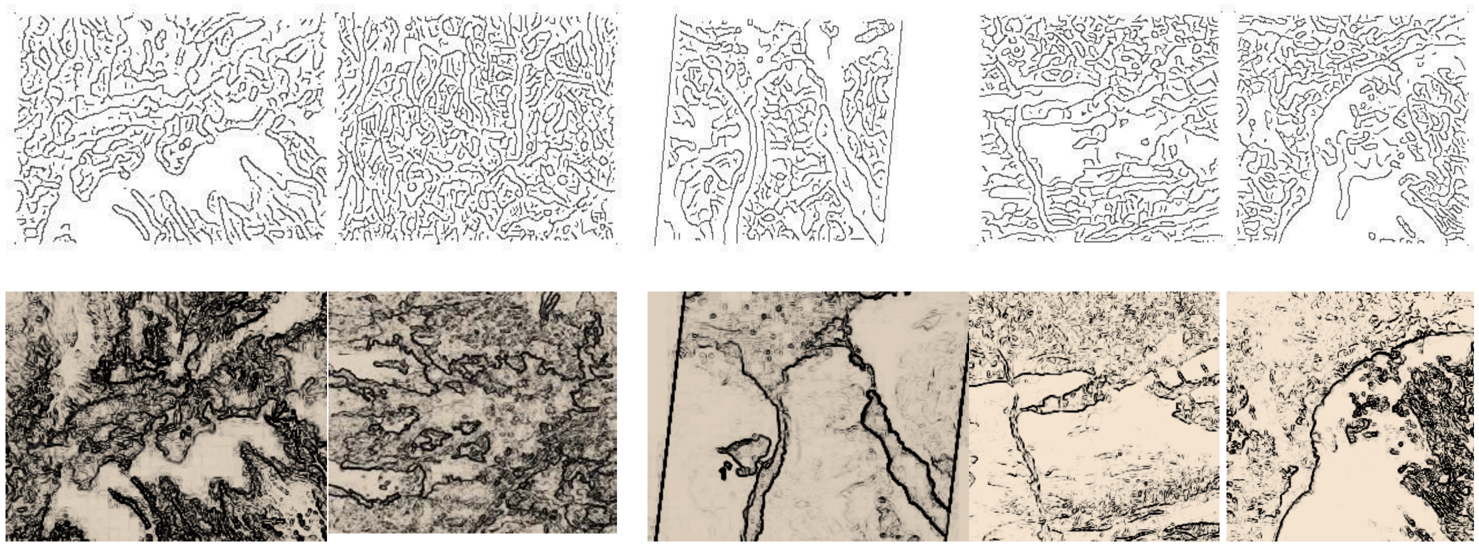

Fig. (a) Fig. (b) Fig. (c) Fig. (d) Fig. (e)

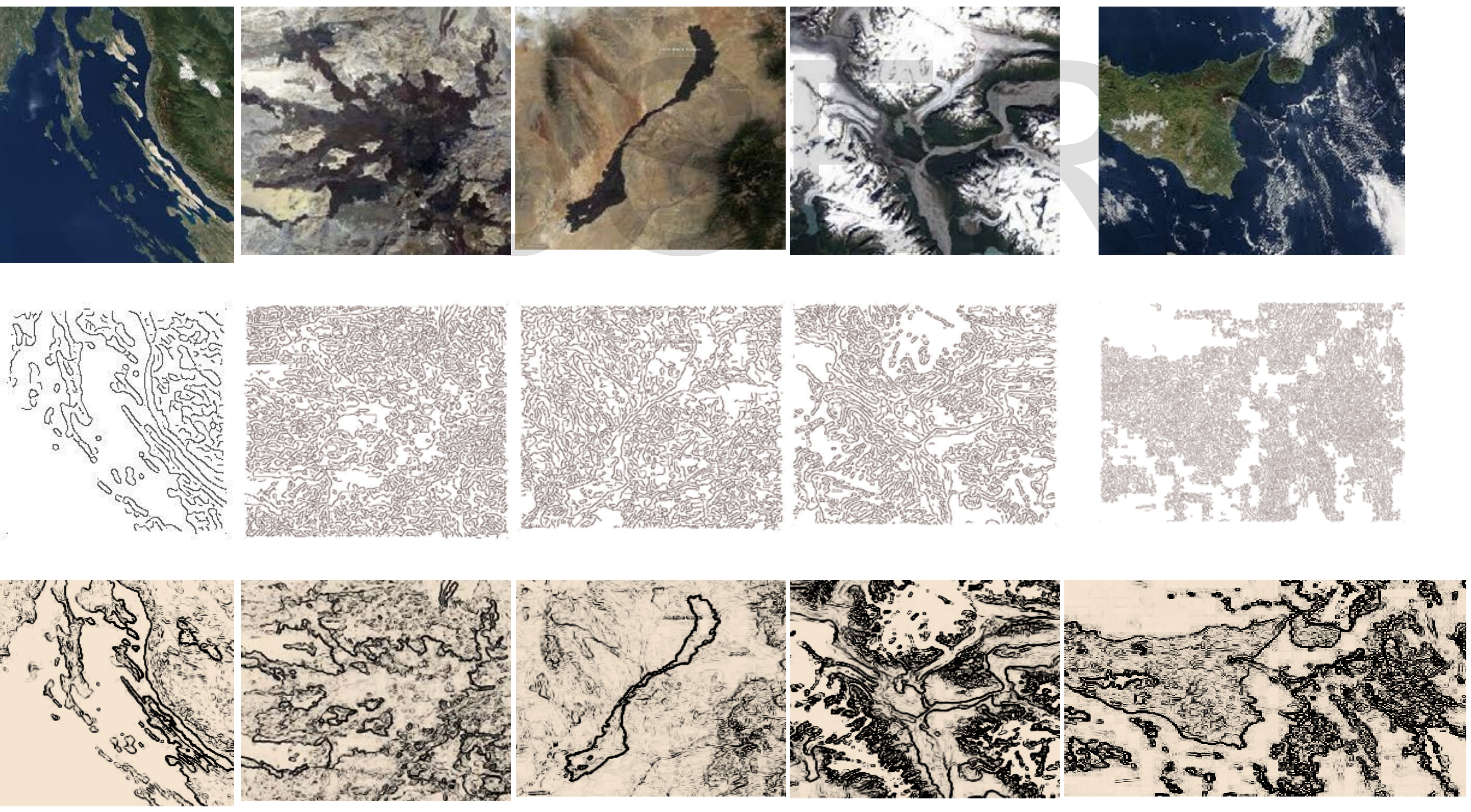

Fig. (f) Fig. (g) Fig. (h) Fig. (i) Fig. (j)

Fig 3. Canny Edge Detection with Satellite Image (original, CPU and GPU Images)

Table 3

Performance comparison for CPU and GPU execution

Image Dimension	Execution time		
	Java (second)	Hadoop (second)	CUDA (mille second)
225*225	2	1	90.41
259*194	3	2	88.47
512*512	7	4	87.09
1024*1024	13	8	85.21
2048*2048	33	27	83.55

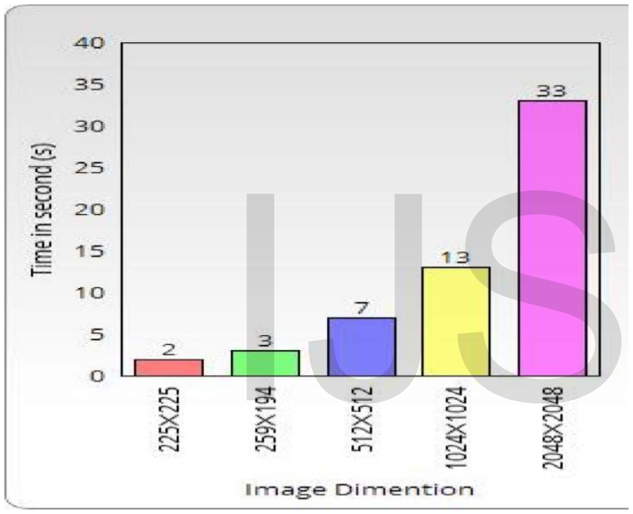

Fig 4.Execution time of java program on CPU

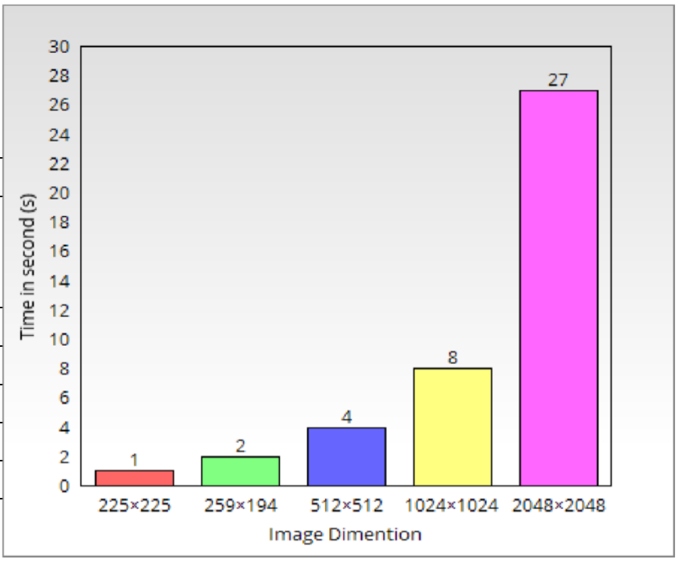

Fig 5. Execution time of Hadoop program on CPU

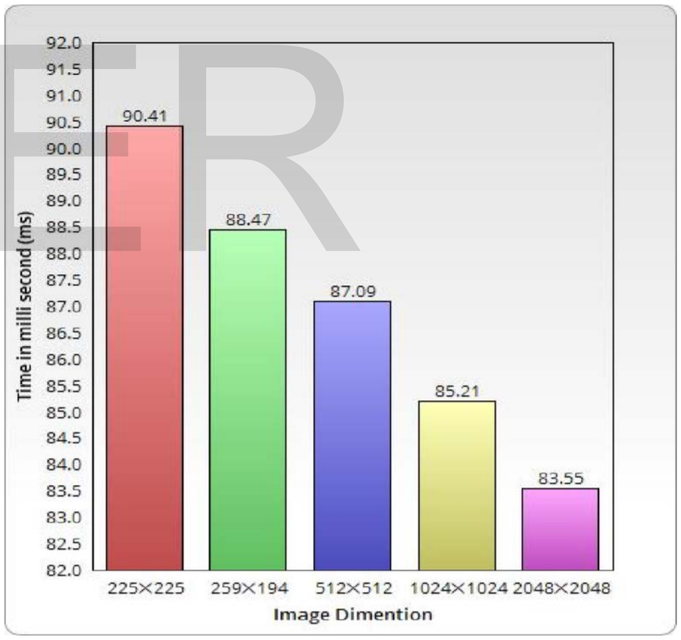

Fig 6. Execution time of Hadoop program on CPU

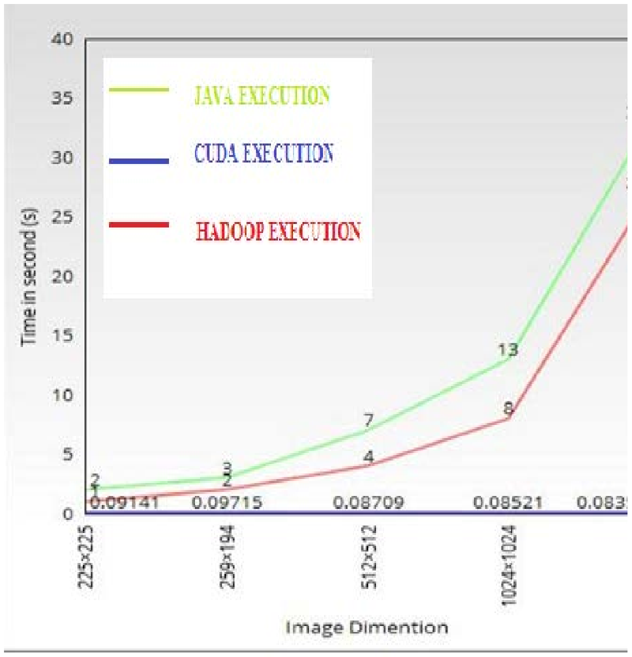

Fig 7 Execution Time of JAVA, Hadoop and CUDA

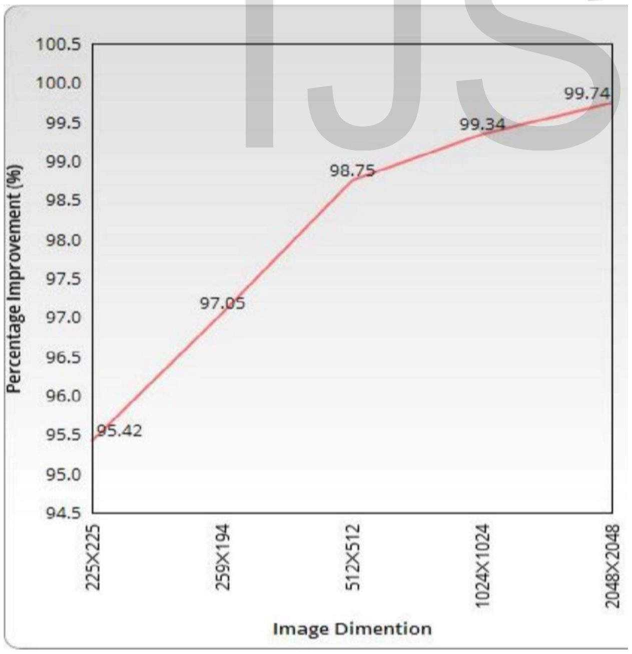

Fig 8 Performance Improvement in execution Time (%) in GPU compare to CPU (Hadoop) execution

Above results shows that for experiment performed for canny edge detection algorithm, (I) for standalone JAVA implementation execution time increase as image dimension increase, (II) For Hadoop based system same scenario happen like java but execution time increases less speedily then Java program.(III) For CUDA based implementation contradictory scenario happen then CPU implementation. Execution time decreases as Image dimension increases. Above experiment shows that GPU are having larger potential to efficiently process images compare to CPU and execution time is much more faster than CPU.

6 Conclusion

Canny edge detection algorithm performed on CPU and GPU shows that for CPU execution time increases with increase in image dimension while for GPU execution time decreases increase in image dimension. Which shows that by using GPU based parallel processing techniques computing power of CPU and GPU is fully utilized. The entire image detection algorithm performed faster for every size input image. For all image sizes, the performance increases gradually in CPU based execution. For the portions of the algorithm performed entirely with the GPU (image smoothing, gradient computation, edge direction computation, and edge classification), the improvement was much larger. The smallest input image was processed 95.4 percent faster by the GPU and the larger input images were processed between 99.3 and 99.7 percent faster.

7 Acknowledgement

This paper is carried out with the full support from Bhaskaracharya Institute for Space Applications and Geo-informatics and the director of Institute Mr. T. P. Singh . I am also thankful to all the members of the institute for supplying the precious data and resources.

8 REFERENCES

[1] Asaduzzaman, Abu, Angel Martinez, and Aras Sepehri. "A time-efficient image processing algorithm for multicore/manycore parallel computing." In *SoutheastCon 2015*, pp. 1-5. IEEE, 2015.

[2] Yamamoto, Muneto, and Kunihiko Kaneko, " Parallel image database processing with MapReduce and performance evaluation, in pseudo distributed mode." International Journal of Electronic Commerce Studies 3, no. 2 (2013): 211-228.

[3] Ryu, Chungmo, Daecheol Lee, Minwook Jang, Cheolgi Kim, and Euiseong Seo, "*Extensible video processing framework in apache hadoop*." ,In Cloud Computing Technology and Science (CloudCom), 2013 ,IEEE 5th International Conference on, vol. 2, pp. 305-310. IEEE, 2013.

[4] Tan, Hanlin, and Lidong Chen," An approach for fast and parallel video processing on Apache Hadoop clusters ." In Multimedia and Expo (ICME), 2014 IEEE International Conference on, pp. 1-6. IEEE, 2014.

[5] Zhang, Gongrong, Qingxiang Wu, Zhiqiang Zhuo, Xiaowei Wang, and Xiaojin Lin. "A Large-scale Images Processing Model Based on Hadoop Platform." In *Proceedings of the Second International Conference on Innovative Computing and Cloud Computing*, p. 51. ACM, 2013.

[6] Zhang, Hong, Zhibo Sun, Zixia Liu, Chen Xu, and Liqiang Wang, "*Dart: A Geographic Information System on Hadoop*." ,In Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on, pp. 90-97. IEEE, 2015

[7] Malakar, Ranajoy, and Naga Vydyanathan, "A CUDA-enabled Hadoop cluster for fast distributed image processing." National Conference on Parallel Computing Technologies (PARCOMPTECH),. IEEE, 2013.0

[8] http://www.hardwarezone.com/feature-cuda-nvidia-turbo-charging-high-performance-computing