International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 10 (2018) pp. 7323-7332

Speed Up Improvement of Parallel Image Layers Generation Constructed By Edge Detection Using Message Passing Interface

Alaa Ismail Elnashar

Faculty of Science, Computer Science Department, Minia University, Egypt.
Associate professor, Department of Computer Science,
College of Computers and Information Technology, Taif University, Saudi Arabia.

Abstract

Several image processing techniques require intensive computations that consume CPU time to achieve their results. Accelerating these techniques is a desired goal. Parallelization can be used to save the execution time of such techniques. In this paper, we propose a parallel technique that employs both point to point and collective communication patterns to improve the speedup of two parallel edge detection techniques that generate multiple image layers using Message Passing Interface, MPI. In addition, the performance of the proposed technique is compared with that of the two concerned ones.

Keywords*:* Image Processing, Image Segmentation, Parallel programming, Message Passing Interface, performance, Edge Detection, Speed up.

INTRODUCTION

Edge detection is a famous image processing methodology that identifies the digital image points at which the image brightness varies sharply or has discontinuities. An edge within an image is defined as discontinuities in image intensity from one pixel to another [1]. Many algorithms [2, 3, 4, 5, 6] were designed and implemented to identify such edges since it is not always easy and possible to locate such edges from complicated images such as real life photos [7].

Many studies [8, 9, 10, 11, 12, 13] have been proposed using threshold techniques to produce high quality edges by selecting an appropriate threshold value. Canny algorithm [26] is one of the most widely used edge detection algorithms, it applies Gaussian smoothing filter [27] on the image using a standard deviation value σ . It then calculates the first derivative in both x and y directions and finds the gradient value. Non-maximum suppression for the gradient value is then applied. Two input parameters T_{high} and T_{low} are used to detect and connect edges. Pixels with values greater than T_{high} are assigned a value 1 in the output, while pixels with values less than T_{low} are assigned the value 0. Pixels with values between T_{high} and T_{low} are assigned the value 1 in the output if they can be connected to any pixel with a value greater than T_{high} through a chain of other pixels with values larger than T_{low} [28]. Finally, the algorithm writes out the edge image to the output image layer file. Implanting and running such techniques may require intensive computations that consume processing time to achieve the resultant image(s).

A. Elnashar [29] introduced two parallel techniques, NASHT1 and NASHT2 that apply edge detection to produce a set of layers for an input image. The proposed techniques generate an arbitrary number of image layers in a single parallel run instead of generating a unique layer as in traditional case; this helps in selecting the layers with high quality edges among the generated ones. Each presented parallel technique has two versions based on point to point communication "Send" and collective communication "Bcast" functions. The presented techniques achieved notable relative speedup compared with that of sequential ones.

In this paper, we introduce a speed up improvement for both NASHT1 and NASHT2. In addition, the performance of the improved technique is compared with that of NASHT1 and NASHT2.

The paper is organized as follows: section 2 presents the related work. In section 3, problem definition is proposed. Section 4 illustrates the two parallel algorithms to be studied, NASHT1 and NASHT2. Speedup improving parallel technique is presented in section 5. The experimental results with their discussion are presented in Section 6.

RELATED WORK

To accelerate edge detection techniques, some studies suggested edge detection parallelization to increase the execution speedup [14, 15, 16, 17]. A comparison between domain decomposition and loop-level is presented in [18]. Christos et al [19] proposed a parallel real-time edge detection technique for field-programmable gate array, FPGAs.

An interactive image processing parallel system designed for manipulating large size images is described in [20].A parallel scheme for large volume 3D image segmentation on a Graphics Processing Unit, GPU, cluster is introduced in [21]. Canny edge detector [26] has been implemented using CUDA [40] system and achieved 50 times speedup compared with CPU system [22].

Prakash et al [23] proposed a technique that combines GPU and multi-cores implementation using MPI [31] to apply edge detection for several images in parallel.

A parallel framework for image segmentation using region based techniques is presented in [24].

Prakash K. Aithal et al. [35] have introduced a parallel edge detection technique of coloured images using MPI by slicing the image and passing the resultant image slices to several parallel processes.

Mohammed Baydoun et al. [36] have improved a parallel implementation that can be used for any image processing application using shared memory. The implementation achieved better results comparing with other image processing techniques.

Shengxiao Niu et al. [37] proposed a new parallel Canny operator for edge tracing without recursive operations, Experimental results on GPU showed that, the improved Canny operator has obtained a high performance.

Wouter Caarls, et al [38] have employed task parallelism with remote procedure call system (RPC) to implement a system that enables an application developer to construct a parallel image processing application with minimal effort. The provided system achieved a significant speedup on SIMD architecture.

Park, I. K. et al [39] explored the design and implementation issues of image processing algorithms on Graphics processing unit (GPUs) with the CUDA framework. In addition, a set of quantitative metrics was proposed. Acceleration achieved for individual algorithms is evaluated in terms of these metrics.

Afshar, Y. et al [41], presented a distributed technique that divides the input image into smaller ones to be distributed among multiple computers to accelerate the image processing algorithm under consideration.

Discrete Region Competition (DRC) algorithm [42] was used for image segmentation and implemented using the PPM library [43, 44, 45]. The distributed-memory scalability of the presented approach overcomes both memory and runtime limitations of single processing and also has better speed-up and scalability

Antonella G., et al [46] presented an approach that enables image processing on cluster of GPUs, using PIMA(GE)2 Lib, the Parallel IMAGE processing GEnoa Library [47] which is provided to the users through a sequential interface to hide the parallelism of the computation. The proposed approach achieved a quite linear speedup on cluster architecture.

PROBLEM DEFINITION

Canny algorithm has been widely used to capture edge information. The quality of the output image layer obtained by edge detection is very sensitive to the input parameters thresholds T_{high} , T_{low} and σ [11] because the scenes and illumination change frequently [25].

The algorithm implementation has become a significant problem, especially for the part of the edge tracing, which consumes a large amount of computing time. So, reducing the consumed time through edge detection parallelization is a desired goal.

NASHT1 and NASHT2 are two message passing interface, MPI parallel techniques with four versions that are based on point-to point and collective communication patterns.

Both techniques were designed and implemented to parallelize layers generation process by distributing layers generation iterations chunks among several parallel processes. In each iteration, the values of input parameters thresholds T_{high} , T_{low} and σ are updated producing a new image layer.

Two versions were developed for each technique. Version1 implementation uses point-to-point communication pattern employing MPI functions "MPI _send" and "MPI_receive". Version2 uses collective communication pattern employing "MPI_Bcast" function.

In general, NASHT1 and NASHT2 with their four versions gained higher speedup than that of sequential ones.

There are three goals for this paper. The first one is to ensure that the quality of generated layers resembles that generated by sequential techniques NASHT1 and NASHT2.

The second goal is to improve the relative speedup achieved by NASHT1 and NASHT2 using the same parallel platform, software and test image used in [29].

The last one is to study the effect of both image size and number of running parallel processes since the performance of MPI programs is affected by various parameters such the number of cores (machines), number of running processes and the programming paradigm which is used [32].

EDGE DETECTION PARALLEL TECHNIQUES, NASHT1 AND NASHT2

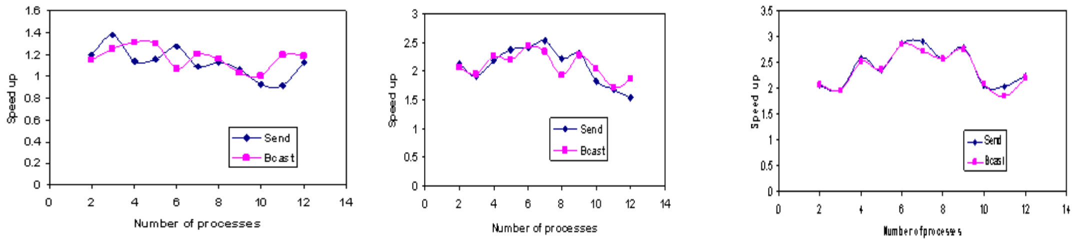
In this section, we present a brief description of NATSHT1 and NASHT2 which are fully described in [29]. Each technique has two versions. Version1 uses MPI point-to-point communication implemented by using MPICH2 [30] "MPI_Send" and "MPI_receive". Version2 uses MPI collective communication implemented by using "MPI_Bcast".

Speedup and the effect of both image size and the number of running parallel process are experimentally examined for the two techniques using three digital images having various sizes ranging from small sized images to large sized ones as shown in Figure 1. Small size, "Brain" image 165x 158, medium size, "Lena" image, 512 x 512 and large size "Picnic" image 1280 x 960.

International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 10 (2018) pp. 7323-7332

Brain, 165x 158 Lena, 512 x 512 Picnic, 1280 x 960

Figure 1. Experiments digital images with their sizes



(a) Brain image (a) Lena image (c) Picnic image

Figure 2. NASHT1, effect of number of running processes for the three examined images

Parallel Technique1: NASHT1

In NASHT1, send version, the root process reads the contents of an image and then passes its data to all other parallel processes. Each one of these processes performs the required computations upon its data chunk and then performs edge detection based on its own values of T_{low}, T_{high} and σ generating a set of image layers that are sent to the root process to be written in separated files.

In NASHT1, bcast version, the root process reads the contents of the input image and broadcasts the image data to all other processes. Then all process including the root execute their own iteration using their data chunks and perform edge detection task based on their own local values of T_{low}, T_{high} and σ.

As in send version, the generated image layers are sent to the root to be saved.

The results show that the relative speedup of both versions of NASHT1, in general, decreases as the number of processes increases for the three examined images as shown in Figure 2, and increases as the image size increases. Although the performance of send version is very close to that of bcast version, the first version gained a slight higher speedup value than the second one.

Parallel Technique2: NASHT2

In NASHT2, send version, the root process reads the input image data to be sent to the processes having odd ranks. Each of these processes manipulates its data chunk and performs edge detection task based on its local values of T_{low}, T_{high} and σ. The generated image layers are sent to the neighbour process having the even ranks to be written. The root process writes its own generated layers.

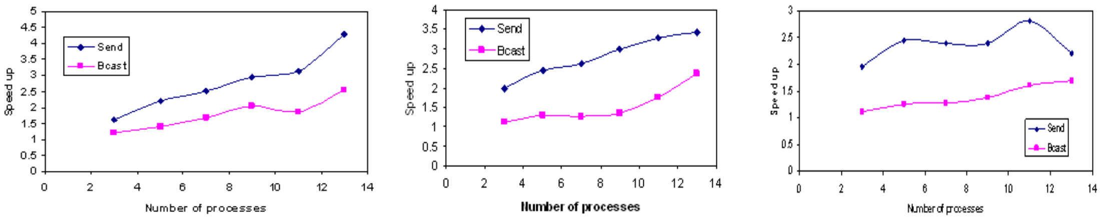
In NASHT2, bcast version, the root process reads the contents of the input image, and broadcasts the image data to all other processes.

Each process executes its iteration upon its data chunk and then performs edge detection task based on its own local values of T_{low}, T_{high} and σ. Once this task is finished within each process except the root process, the generated image layers are sent to the neighbor even ranked process to be written. The root process writes its generated layers directly after terminating edge detection task.

Relative speedup of both versions of NASHT2 increases as the number of process increases as shown in Figure3, since odd ranked processes send their generated layers to the neighbor processes to be written which leads to less overhead on even ranked processes to finalize their amount of work.

It also noticed that, relative speedup of both versions of NASHT2 decreases as the image size increases but still increases as the number of processes increases. Send version performs better than bcast version especially for small sized images except in case of larger number of processes.

International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 10 (2018) pp. 7323-7332



(a) Brain image (a) Lena image (c) Picnic image

Figure 3. NASHT2, effect of number of running processes for the three examined images

PROPOSED PARALLEL TECHNIQUE: NASHT3

In this section, we present NASHT3, a technique that enhance the speedup achieved by both NASHT1 and NASHT2. The proposed technique also has two versions that employ "MPI_Send" and "MPI_Bcast".

The difference between NASHT3 and the other two concerned techniques is the way in which the running parallel processes communicate.

Parallel Technique NASHT3: Send Version

In this technique, the root process reads the contents of an input image and then sends its data to all other processes. All processes including the root process compute their iteration chunks and then perform edge detection task based on their own local values of T_{high}, T_{low} and σ .

Once the computations within each process are finished, the generated image layers are directly written by the process that performed the computations to separate files named based on T_{high}, T_{low} and σ .

The steps of this version are described in figure 4.

We use the visualization software Jumpshot-4 [33, 34] which provides a visual hierarchical structure of the running processes. A jump-shot time line for inter-process communication is shown in Figure 5.

Parallel Technique NASHT3: Bcast Version

In this technique, the root process reads the contents of an input image and broadcasts the image data to all other processes. Each process including the root process computes its iteration chunk and then performs edge detection task based on its own local values of T_{high}, T_{low} and σ . Once the computations within each process are finished, the generated image layers are directly written by the process that performed the computations to separate files named based on T_{low}, T_{high} and σ . The steps of this version are described in figure 6. A jump-shot time line for inter-process communication is shown in Figure 7.

01. Initialize MPI environment;

02. Initialize edge detection parameters T_{high}, T_{low} and σ ;

03. Read the number of layers (N_layers) to be generated;

04. Determine the number of MPI processes (Npr) and their ids;

/*Determine the number of generated layers (N_layers_Pr) for each process */

05. N_layers_Pr = N_layers / Npr ;

06.Update edge detection parameters based on id value;

07. **If id=master then**

08. Read(Image_File_Name, Image, Im_Rows , Im_Cols);

09. Send Image rows (Im_Rows) to all other processes;

10. Send Image columns (Im_Cols) to all other processes;

11. Send Image array to all other processes;

12. for k = 1 to N_layers_Pr

13. **{**

International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 10 (2018) pp. 7323-7332
© Research India Publications. http://www.ripublication.com

14. Edge_Detector (Image_File_Name, Layer, Im_Rows , Im_Cols, T_{high}, T_{low} , σ) ;

15. Form the output Layer_File_Name based on T_{high}, T_{low} and σ ;

16. Write (Layer_File_Name, Layer, Layer_File, Im_Rows , Im_Cols);

17. **}**

18. **EndIf**

/* All processes with id <> master execute the following part */

19. **If id <> mater then**

20. Receive Image rows (Im_Rows) from master process;

21. Receive Image columns (Im_Cols) from master process;

22. Receive Image array from master process;

23. Send Image rows (Im_Rows) and columns (Im_Cols) to process with id = id+1 ;

24. Update edge detection parameters based on id value;

25. for k = 1 to N_layers_Pr

26. **{**

27. Edge_Detector (Image_File_Name, Layer, Im_Rows , Im_Cols, T_{high}, T_{low} , σ) ;

28. Form the output Layer_File_Name based on T_{high}, T_{low} and σ ;

29. Write (Layer_File_Name, Layer, Layer_File, Im_Rows , Im_Cols);

30. **}**

31. **EndIf**

32. Finalize MPI environment

33. End

Figure 4. Parallel Technique NASHT3: Send Version

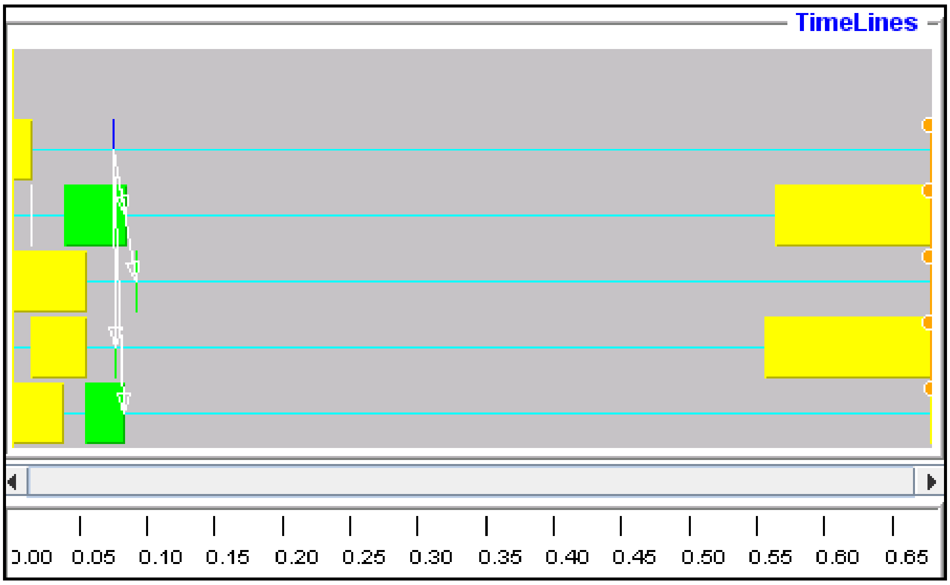


Figure 5. NASHT3 Send Version Inter-Process Communication

01. Initialize MPI environment;

02. Initialize edge detection parameters T_{high}, T_{low} and σ ;

03. Read the number of layers (N_layers) to be generated;

04. Determine the number of MPI processes (Npr) and their ids;

/*Determine the number of generated layers (N_layers_Pr) for each process */

05. N_layers_Pr = N_layers / Npr ;

06. **If id=master then**

07. Read(Image_File_Name, Image, Im_Rows , Im_Cols);

08. **EndIf**

/* All processes execute the following part */

09. Master broadcasts Image rows (Im_Rows) to all processes;

10. Master broadcasts Image columns (Im_Cols) to all processes;

11. Master broadcasts Image array to all processes;

12. Update edge detection parameters based on id value;

13. for k = 1 to N_layers_Pr

14. **{**

15. Edge_Detector (Image_File_Name, Layer, Im_Rows , Im_Cols, T_{high}, T_{low} , σ) ;

16. Form the output Layer_File_Name based on T_{high}, T_{low} and σ ;

17. Write (Layer_File_Name, Layer, Layer_File, Im_Rows, Im_Cols);

18. **}**

19. Finalize MPI environment

20. End

Figure 6. Parallel Technique NASHT3: Bcast Version

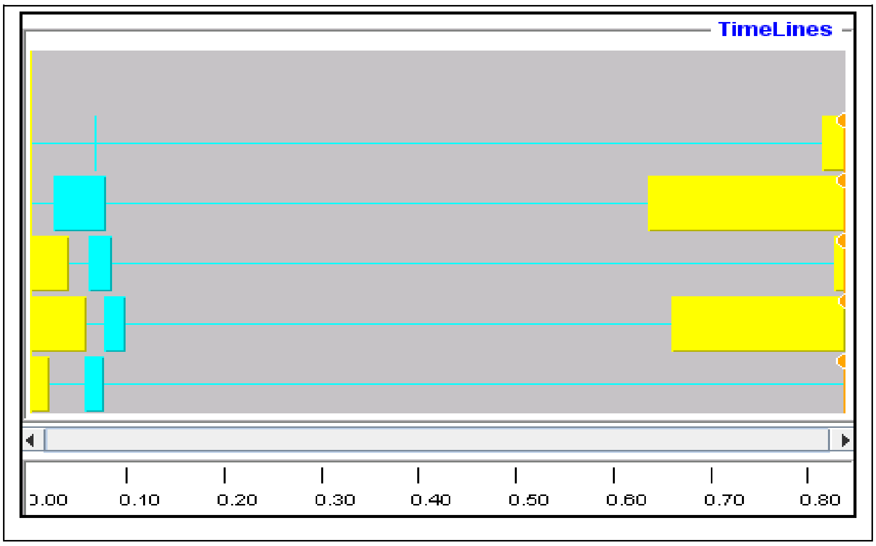


Figure 7. NASHT3 Bcast Version Inter-Process Communication

EXPERIMENTAL RESULTS AND DISCUSSION

To check the achievement of the three goals of our paper, we carried out some experiments using the same hardware and software architectures on which NASHT1 and NASHT2 had been examined and the digital images shown in Figure1.

The first experiment concerns with the quality of generated layers. Its results ensure that NASHT3 with its two versions produced a set of layers that have not any quality differences compared with the layers generated by sequential techniques NASHT1 and NASHT2.

Table 1. Performance Results- NASHT1, NASHT2 and NASHT3

Image / Serial execution time	Parallel Techniques / Speedup					
	NASHT1		NASHT2		NASHT3	
	Send	Bcast	Send	Bcast	Send	Bcast
Brain, 5.26 Sec.	1.38	1.31	4.29	2.53	6.19	4.96
Lena, 46.90 Sec.	2.54	2.44	3.42	2.36	6.49	4.69
Picnic, 215.64 Sec.	2.92	2.85	2.80	1.70	4.37	4.44

Experiment 2 was carried out to study whether any speedup improvement is achieved by NASHT3 compared with NASHT1 and NASHT2.The experiment was designed to generate 720 layers for the input image in parallel.

Table 1 shows the maximum speedup values obtained by running NASHT1, NASHT2 and NASHT3 with several parallel processes on the described system using test input images.

All recorded results show a notable speedup for both NASHT1 and NASHT2 compared with the serial execution time values and a significant speedup improvement of NASHT3 compared with NASHT1 and NASHT2.

Experiment 3 was carried out to study the effect of the number of running parallel process and image size on the relative speedup of the tested technique. Relative speedup is computed by the formula, $Speedup = T_S / T_P$, where T_S is the execution time of sequential technique and T_P is the execution time of corresponding parallel one.

This experiments has two folds, the first one is to study the effect of the number of running parallel processes used in the technique execution on the execution behavior. The second one is to study how the image size affects the performance of tested technique.

In this experiment, the sequential versions that correspond to each parallel technique are executed with an input image and the execution time is then recorded for each version.

Each version of NASHT3 is repeatedly executed with the same input image increasing the number of parallel processes in each run; the execution time and relative speedup of each run are then computed and recorded for each version.

The same scenario is followed with other images having different sizes to observe the effect of image size on execution behavior.

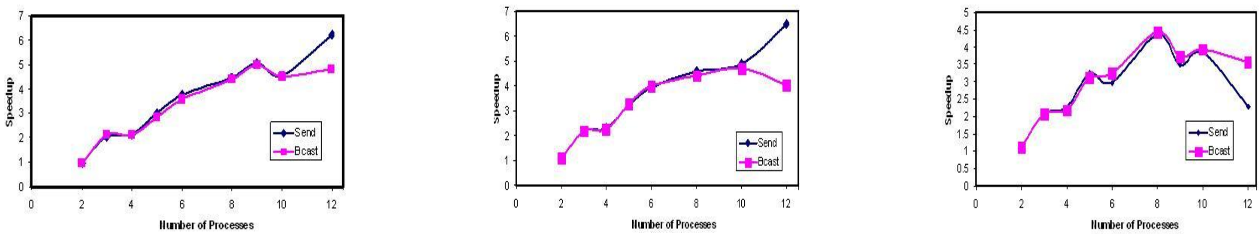
Regarding the effect of the number of running parallel processes, the experimental results show that both versions of NASHT3 scale well with the number of processes compared with the scalability of both versions of NASHT1, this is can be clarified by comparing Figure 2 and Figure 8.

Although NASHT3 obtained a higher speedup than NASHT2, the later one, in general, is slightly more scalable than NASHT3 ignoring the effect of image size as shown in Figure 3 and Figure 8.

Concerning the effect of image size, NASHT1 with its two versions is the only technique that achieves a higher speedup as image size increases. This advantage is hidden behind its lowest speedup compared with the two other techniques as shown in Figure 9.

All of the experimental results discussed above show that the speedup improvement goal of NASHT3 is achieved under the constraints of running processes number and image size. The reasons of NASHT3 improvement are summarized as follows:

1. Its design is simpler than that of NASHT1 and NASHT2. So, its executable implementation has better resources utilization.
2. It is designed in such a way in which all processes share the computations. In contrast, NASHT2 design allows only about the half of running processes to share the computations.
3. It does not suffer from intensive I/O overhead since each running processes writes its own generated layers. In contrast, NASHT1 design preserves only the root process to write all the generated layers received form all other processes.



(a) Brain image (a) Lena image (c) Picnic image

Figure 8. NASHT3, effect of number of running processes for the three examined images

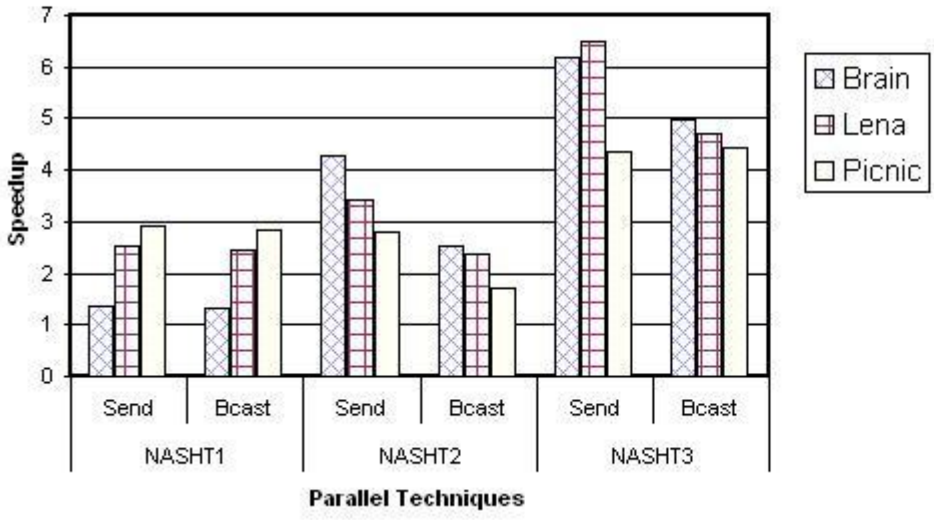


Figure 9. NASHT1, NASHT2 and NASHT3 speedup achievement comparison

CONCLUSION AND FUTURE WORK

We presented a message passing interface, MPI, parallel technique, NASHT3 that applies edge detection to generate a set of layers for an input image. Its main goal is to improve the speedup of two MPI parallel edge detection techniques NASHT1 and NASHT2. The proposed technique was designed implemented and executed using the same parallel software/ hardware architectures used for the two studied techniques, also the same experimental images were used. Each studied technique has two different versions. Send version uses point-to-pint communication and bcast version uses collective communication pattern.

Experiments were carried out to study the effect of both number of processes and also massage size on the techniques performance.

The experimental results for the three studied techniques demonstrated that NASHT3 send version is faster than NASHT1 send version (4.5, 2.6 and 1.5) times for small, medium and large images. NASHT3 bcast version is faster than NASHT1 bcast version (3.8, 1.9 and 1.6) times for small, medium and large images.

NASHT3 send version is faster than NASHT2 send version (1.5, 3.4 and 1.6) times for small, medium and large images.NASHT3 bcast version is faster than NASHT2 bcast version (2.0, 2.0 and 2.6) times for small, medium and large images. In general, NASHT3 achieved a speed up improvement for the two considered techniques. In future, the experimental hardware platform will be extended to study the effect of larger number of processes and also a larger image size on parallel performance.

REFERENCES

[1] Poonam, S. Deokar, (2015) “Implementation of Canny Edge Detector Algorithm using FPGA“ , IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 2 No. 6, pp 112 - 115.

[2] Gholamali Rezai-Rad & Majid Aghababaie, (2006) “Comparison of SUSAN and Sobel Edge Detection in MRI Images for Feature Extraction“, IEEE transaction on Information and Communication Technologies, Vol. 1, pp 1103 - 1107.

[3] Raman Maini & Dr. Himanshu Aggarwal, (2009) “Study and Comparison of Various Image Edge Detection Techniques“, International Journal of Image Processing (IJIP), Vol. 3, No. 1, pp 1-12.

[4] LS. Davis, (1975) “A survey of edge detection techniques”, Computer Graphics and Image Processing, Vol. 4, No. 3, pp 248-260.

[5] Tarek A. Mahmoud & Stephen Marshall, (2008) “Medical Image Enhancement Using Threshold Decomposition Driven Adaptive Morphological Filter“, 16th European Signal Processing Conference (EUSIPCO 2008), EURASIP.

[6] P.Subashini & M.Krishnaveni & Suresh Kumar Thakur, (2010) “Quantitative Performance Evaluation on Segmentation Methods for SAR ship images“, Proceedings of the Third Annual ACM Bangalore Conference,

[7] Saurabh Singh & Dr. Ashutosh Datar, (2013) “EDGE Detection Techniques Using Hough Transform“, International Journal of Emerging Technology and Advanced Engineering, Vol. 3, NO. 6, pp 333 - 337.

[8] Salem Saleh Al-amri & N.V. Kalyankar & Khamitkar S.D, (2010) “ Image Segmentation by Using Thershod Techniques“, Journal of Computing, Vol. 2, No. 5, MAY, pp 83 - 86.

[9] Alexander Drobchenko & Jarkko Vartiainen & Joni-Kristian Kämäräinen & Lasse Lensu & Heikki Kälviäinen, (2011) " Thresholding Based Detection of Fine and Sparse Details", Frontiers of Electrical and Engineering, China, Vol. 6, No. 2, , pp 328 - 338.

[10] Derek Bradley & Gerhard Roth, (2007) "Adaptive Thresholding using the Integral Image", Journal of Graphics, GPU, and Game Tools, Vol. 12, No. 2, pp 13-21.

[11] Simranjit Singh Walia & Gagandeep Singh, (2014) "Color based Edge detection techniques– A review ", International Journal of Engineering and Innovative Technology (IJEIT), Vol. 3, No. 9, March, pp 297-301.

[12] S. Jansi & P. Subashini, (2012) "Optimized Adaptive Thresholding based Edge Detection Method for MRI Brain Images", International Journal of Computer Applications, Vol. 51, No.20, pp 1-8.

[13] Li, J. & Ding, S., (2011) "A research on improved canny edge detection algorithm", in International Conference on Applied Informatics and Communication, pp 102-108.

[14] Ghassan F. Issa & Hussein Al-Bahadili & Shakir M. Hussain, (2012) " Development and Performance Evaluation of A Lan-Based Edge-Detection Tool", International Journal on Soft Computing (IJSC) Vol.3, No.1, pp 121-135.

[15] Mohd Azam Osman & Muqhtar Yassin Mohamad & Rosni Abdullah, (2008) "Parallelizing an Edge Detection Algorithm for Image Recognition to Classify Paddy and Weeds Leaf on Sun Fire Cluster System", 7th WSEAS Int. Conf. on Software Engineering, Parallel and Distributed Systems (SEPADS '08), University of Cambridge, UK, pp 56-60.

[16] Chandrashekar N.S. & Dr. K.R. Nataraj, (2012) "A Distributed Canny Edge Detector and Its Implementation on FPGA", International Journal Of Computational Engineering Research (ijceronline.com) Vol. 2, No. 7, pp 177-181.

[17] A. Z. Brethorst & N. Desai, D. P. Enright & R. Scrofano, (2011) "Performance evaluation of Canny edge detection on a tiled multicore architecture," in Society of Photo-Optical Instrumentation Engineers (SPIE) ConferenceSeries, Vol. 7872, pp. 1–8.

[18] Taieb Lamine Ben Cheikh & Giovanni Beltrame & Gabriela Nicolescu & Farida Cheriet & Sofi`ene Tahar,(2012) "Parallelization Strategies of the Canny Edge Detector for Multi-core CPUs and Many-core GPUs", Conference Proc. of the 10th IEEE Intl. NEWCAS Conference, pp 49-52.

[19] Christos Gentsos & Calliope- LouisaSotiropoulou & Spiridon Nikolaidis Nikolaos Vassiliadis (2010) "Real-Time Edge Detection Parallel Implementation for FPGAs", 17th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2010, Athens, Greece, pp 499-502.

[20] Jones, Donald R. & Elizabeth R. Jurrus & Brian D. Moon & Kenneth A. Perrine, (2003)"Gigapixel-size Real-time Interactive Image Processing with Parallel Computers," Parallel and Distributed Processing Symposium.

[21] Aaron Hagan & Ye Zhao, (2009)" Parallel 3D Image Segmentation of Large Data Sets on a GPU Cluster", 5th International Symposium, ISVC 2009, Las Vegas, NV, USA, pp. 960–969

[22] Luis H.A. Lourenc¸o & Daniel Weingaertner & Eduardo Todt,(2012)" Efficient implementation of Canny Edge Detection Filter for ITK using CUDA", 13th Symposium on Computer Systems, pp 960–969.

[23] Prakash K. Aithal & U. Dinesh Acharya & Rajesh Gopakumar, (2015)"Detecting the Edge of Multiple Images in Parallel", International Journal of Computer, Electrical, Automation, Control and Information Engineering Vol.9, No.7, pp 1442–1445.

[24] Juan C. Pichel & David E. Singh & Francisco F. Rivera, (2007)" A Parallel Framework for Image Segmentation Using Region Based Techniques", Vision Systems: Segmentation and Pattern Recognition, I-Tech, Vienna, Austria, pp 81-98.

[25] J. Gao & N. Liu, (2012)"An Improved Adaptive Threshold Canny Edge Detection Algorithm ", in Computer Science and Electronics Engineering (ICCSEE), International Conference on, pp. 164-168.

[26] J. F. Canny, (1986) "A computational approach to edge detection", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol., No. 6, pp. 679-698.

[27] Samir Kumar Bandyopadhyay, (2011)"Edge Detection in Brain Images", International Journal of Computer Science and Information Technologies, Vol. 2, No.2, pp 884-887.

[28] R.A. Haddad & A.N. Akansu, (1991) "A Class of Fast Gaussian Binomial Filters for Speech and Image Processing," IEEE Transactions on Acoustics, Speech and Signal Processing, Vol. 39, March, pp 723-727.

[29] Alaa Ismail El-Nashar, (2016) "Parallel Generation of Image Layers Constucted by Edge Detection Using Message Passing Interface", International Journal of Computer Science & Information Technology (IJCSIT) Vol.8, No.5, pp 1–17.

[30] W. Gropp, (2002) "MPICH2: A New Start for MPI Implementations", In Recent Advances in PVM and MPI: 9th European PVM/MPI Users' Group Meeting, Linz, Austria.

[31] Bernd Mohr & Jesper Larsson Träff & Joachim Worringen & Jack Dongarra, (2006)"Recent Advances in Parallel Virtual Machine and Message Passing Interface", 13th European PVM/MPI User's Group Meeting Proceedings, Bonn, Germany, pp 17-20.

[32] Alaa Ismail El-Nashar, (2011) "Performance of MPI Sorting Algorithms on Dual–core Processor Windows-

based Systems", International Journal of Distributed and Parallel Systems (IJDPS) Vol.2, No.3, pp 1–14.

[33] A. Chan D. Ashton & R. Lusk, and W. Gropp, (2007)"Jumpshot-4 Users Guide" Mathematics and Computer Science Division, Argonne National Laboratory.

[34] Omer Zaki, Ewing Lusk & William Gropp & Deborah Swider, (1999) "Toward Scalable Performance Visualization with Jumpshot", International Journal of High Performance Computing Applications, Vol. 13, No. 3 pp. 277 – 288.

[35] Prakash K. Aithal & Rajesh G. & Dinesh U. Acharya, (2014) "MPI based edge detection of coloured image using laplacian of gaussian filter", International Journal of Computer Application, pp. 277 – 288.

[36] Mohammed Baydoun & Mohamad Adnan Al-Alaoui & Rony Ferzli. (2013) "Parallel Edge Detection Based on Digital Differentiator Approximation", in 3rd International Conference on Communications and Information Technology (ICCIT-2013): Wireless Communications and Signal Processing, Beirut, pp 371-375.

[37] Shengxiao Niu & Jingjing Yang & Sheng Wang & Gengsheng Chen, (2011)"Improvement and Parallel Implementation of Canny Edge Detection Algorithm Based on GPU", ASIC (ASICON), 2011 IEEE 9th International Conference, pp 641-644.

[38] Wouter Caarls & Pieter Jonker & Henk Corporaal, (2005)" Skeletons and Asynchronous RPC for Embedded Data and Task Parallel Image Processing", IAPR Conference on Machine Vision Application (MVA'05), Tsukuba Science City, Japan, pp 16-18.

[39] I. K. Park & N. Singhal & M. H. Lee & S. Cho, & C. W. Kim, (2011)"Design and Performance Evaluation of Image Processing Algorithms on GPUs", IEEE Transactions on Parallel and Distributed Systems, Vol. 22, No. 1, pp. 91-104.

[40] NVIDIA Corporation, Compute Unified Device Architecture (CUDA), http://developer.nvidia.com/object/cuda.html.

[41] Afshar Y. & Sbalzarini Ivo F., (2016) "A Parallel Distributed-Memory Particle Method Enables Acquisition-Rate Segmentation of Large Fluorescence Microscopy Images", PLoS ONE, Vol. 11, No.4.

[42] Cardinale J. & Paul G. & Sbalzarini Ivo F., (2012) "Discrete region competition for unknown numbers of connected regions", IEEE Trans Image Processing, Vol. 21, No. 8, pp 3531–3545.

[43] Sbalzarini Ivo F. & Walther JH. & Bergdorf M. & Hieber SE. & Kotsalis EM. & Koumoutsakos P., (2006) "PPM—A Highly Efficient Parallel Particle-Mesh Library for the Simulation of Continuum Systems", Journal of Computational Physics, Vol. 215, No. 2,pp 566–588.

[44] Awile O. & Demirel O. & Sbalzarini Ivo F., (2010) "Toward an Object-Oriented Core of the PPM Library", in Proc. ICNAAM, Numerical Analysis and Applied Mathematics, International Conference, pp 1313–1316.

[45] Awile O. & Mitrović M. & Reboux S. & Sbalzarini Ivo F., (2013) "A domain-specific programming language for particle simulations on distributed-memory parallel computers", in Proc. III Intl. Conf. Particle-based Methods (PARTICLES). Stuttgart, Germany.

[46] Antonella Galizia & Daniele D'Agostino & Andrea Clematis, (2013) "An MPI–CUDA library for image processing on HPC architectures", Journal of Computational and Applied Mathematics, Vol. 273, pp 414–427.

[47] Andrea Clematis & Daniele D'Agostino & Antonella Galizia, (2007) " Parallel I/O Aspects in PIMA(GE)2 Lib", Parallel Computing: Architectures, Algorithms and Applications, Vol. 38, pp 441-448.