International Journal of New Computer Architectures and their Applications (IJNCAA) 8(1): 1-10
The Society of Digital Information and Wireless Communications, 2017 ISSN 2220-9085 (Online); ISSN 2412-3587 (Print)

Acceleration of Canny Edge Detection Algorithm Using Parallel Clusters

Njood S.Alassmi and Soha S. Zaghloul, PhD.
College of Computer & Information Sciences, Department of Computer Science, King Saud University, Riyadh
435920199@student.ksu.edu.sa, smekki@ksu.edu.sa .

ABSTRACT

Image processing is a computational operation that requires many CPU cycles for simple image transformation. It takes every pixel of an image to perform a transformation to a new image. The image can be divided into smaller chunks, with same transformation operations being implemented on each. Thus, image processing is a good candidate for running on a parallel processor to improve the speed of computation when there are multiple images to be processed. In fact, this research focuses on Canny edge detection as a case study of probing parallelism. This work presents the design and implementation of sequential and parallel edge detection algorithms that are capable of producing high-quality results and performing at high speed. Therefore, this research aims to improve the Canny edge detection algorithm in terms of speed and scalability with different sizes of images. The algorithm is implemented using parallel clusters on KACST's SANAM supercomputer. It is found that there is a valuable gained speedup with respect to the sequential version.In addition, it is found that more parallelism is explored in larger image sizes with Canny edge detector.

KEYWORDS

Canny Edge Detection, Sequential Implementation, Parallel Processing, Parallel Cluster, SANAM.

1 INTRODUCTION

The importance of image detection algorithms is increasing with the advance of technology due to the recent expansion of images in industry. The tools that acquire images became handy to an ample number of users. These are interested in assessing the quality of the acquired images. Image edge detection is one of the most essential methods in image processing for image quality assessment, image analysis, image pattern recognition, and computer vision. Actually, the importance of image detection comes from the fact that it is the first step to be conducted on images.

In the literature, many filters are used to conduct the image edge detection. These include Prewitt, Canny, Roberts, and Sobel. In [1], the researchers proved that although the Sobel operator is the simplest amongst the previously mentioned ones; however, it is sensitive to noise. On the other hand, the Canny operator applies Gaussian filter to remove any noise in the image. Therefore, better results are guaranteed. In addition, [2] found that almost 90% of the edges are correctly detected using Canny operator.

However, the Canny edge detection algorithm suffers from complex computations and therefore, is time consuming. Given an image of size mxn, the time complexity of the Canny algorithm is found to be O(mn log mn). In order to make the algorithm produce its results in real time, parallelism is needed.

There are two main parallel programming models; namely, the shared memory processor (SMP) and the parallel clusters - also known as message-passing processors (MPP). This classification is based on the way the memory is shared between the multiprocessors [3]. In SMP, the processors communicate with each other through a common memory. On the other hand, MPP is a distributed memory parallel architecture. Communication between processors is conducted through message passing. A hybrid architecture of both SMP and MPP is also available.

The aim of this research is tomaximize the speedup, the efficiency, and the scalability of the parallel program. Also, the accuracy of the detected edges should be at least the same as the sequential version. The layout of this research is as follows: Section 2 exposes similar work in the literature. Section 3 and Section 4 detail the sequential and parallel versions of Canny algorithm respectively. Section 5 explains the performance metrics to be measured in the experiments. Results are shown and analyzed in Section 6. Section 7 concludes the research and gives a hint about the future work.

2 REVIEW OF RELATED STUDIES

This research focuses on Canny edge detection as a case study to implement the parallelism concept. In order to implement the Canny edge detection algorithm, a series of steps must be followed. The first is to filter out any noise in the original image before attempting to detect any edges. This can be done by applying the Gaussian filter that can be computed using a simple mask. After smoothing the image and removing the noise, the next step is to find the edge strength by measuring the gradient of the image. Subsequently, the direction of the edge should be computed using the gradient in the x and y directions. After the edge directions are known, non-maximum suppression should be applied, which is used to trace along the edge in the edge direction and suppress any pixel value. This results in a thin line in the output image. The final stage is edge tracking by hysteresis; the final edges are determined by suppressing all edges that are not associated with the definite edge [4]. Y. Kong et al. [5] implemented the Canny edge detection method and found that almost 90% of the edges are correctly detected. However, disadvantages include intensive and complex computations, ensuring that it is time consuming. The overall time complexity is found to be O (m*n log m*n), where m and n are the width and length of the image in terms of pixels.

With the prevalence of parallelism, image edge detection algorithms are implemented in various parallel environments, including GPUs, SMPs, and MPPs. In [6] and [7], the authors proved the efficacy of the GPU in conducting millions of pixel calculations involved in image processing in parallel. In addition, [8] examined the edge detection in both sequential and parallel algorithms in order to measure the speedup. The results indicate that the parallel implementation is about 262 times faster than the sequential implementation.

In [9], the authors proposed a parallel Canny algorithm for the embedded CPU and GPU heterogeneous. The results showed that the proposed parallel Canny algorithm achieved nearly 50 times speedup on the embedded systems. They used differently sized images for the tests. For 512×512 images, the runtime of a Canny algorithm in sequential implementation is found to be over 1,5898 milliseconds. Otherwise, Canny improved the runtime to a level of 3,310 milliseconds by using embedded CPUs and GPUs.

In [10], a parallel version of the Canny algorithm is applied on a SMP environment. The maximum gained speedup is around 1.7 when using three threads for an image size of 2000x2000 pixels.

In [11]-[13], the edge detection algorithm is applied on parallel clusters. In [11], the authors used 4 parallel clusters, and achieved a gained speedup mostly above 0.7 for a 2Kx2K image. In [12], the authors applied K-means clustering. It is found that the algorithm is more accurate in edge recognition; however, it takes more time (3.32 seconds as opposed to 1.28 seconds for Canny). Finally, in [13], the authors used 10 parallel clusters and the execution time is found to be 0.0311 seconds for an image size of 2.5K x 2.5 K.

3 SEQUENTIAL ALGORITHM

The Canny edge detection algorithm is proved to provide the optimal edge detector [14].The algorithm goes through four steps before reaching the final result. The input image is read into an array of size m×n where m and n are the width and length of the image respectively in terms of number of pixels. The algorithm starts by storing the 2-D

image in a 1-D array to simplify processing in later phases. Then, the algorithm visits every pixel of the image and makes the modifications as detailed below. The algorithm goes through a series of steps [15]:

1. **Noise Removal:** The first step is to filter any noise in the original image before attempting to detect edges. This is done using Gaussian filter computed by a simple mask and exclusively used in the Canny algorithm.
2. **Gradients Calculation:** After eliminating the noise and smoothing the image, gradients of the image are taken for each edge, gradient magnitude (M) represents the edge intensity.
3. **Thresholding:**
   Two thresholds, upper and lower, are used to detect image edges. This allows for greater flexibility than using a single threshold approach.
4. **Edges Classification:**
   The thresholding is used to detect the final edges in an image by using the two thresholding values of v_low and v_high. In this step, the pixels are classified into three categories in terms of the value of the magnitude M. If a pixel gradient is higher than the high threshold (M >v_high), the pixel is accepted as a strong edge. If (v_low ≤ M <v_high), the pixel is classified as a weak edge and waits for thejudgement of the edge points. If (M <v_low), the pixel is classified as a non-edge, and therefore, rejected.The diagram in Fig.1 shows the main steps of the approach for Canny edge detection.

The diagram in Fig.1 depicts the previously detailed steps of the approach for Canny edge detection. A major disadvantage of Canny is its intensive and complex computations making it time consuming. In order to measure the overall program complexity, the complexity of each step is first calculated independently. Steps 1 to 3 are implemented by image convolution. For an *mxn* image size, the time complexity is therefore *O (mn log mn)* for the first three steps. The final step of thresholding is implemented by the final step of thresholding is implemented by selecting all the high values and removing the low values. This can be done in time *O (mn)* [15]. Therefore, by taking the highest complexity, it is found that the overall time complexity is *O (mn log mn)*.

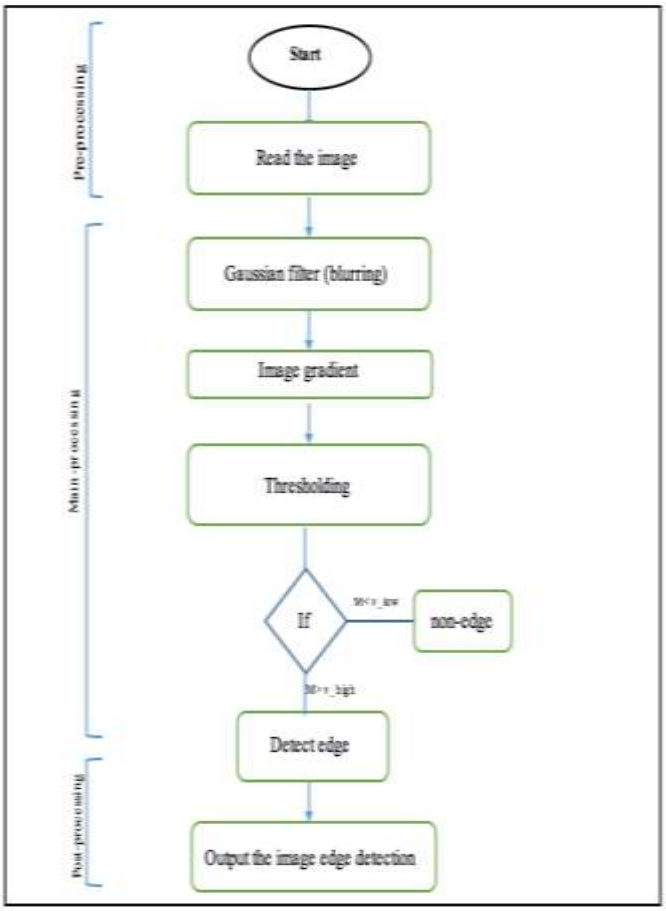


Figure 1. The flow chart of the sequential version for Canny Edge Detection steps.

4PARALLEL ALGORITHM ON CLUSTERS

Clustering is a technique to configure multiple machines belonging to a general network for parallel purposes. A cluster consists therefore of a set of nodes interconnected by high technology network.

The algorithm for parallel clusters is divided into three main parts: the partitioning of the image, the implementation of the Canny filter, and finally, the merging of sub-images.

- Image Partitioning

The first step to make use of data parallelism on clusters, is to partition the image. This is explained as follows: the image is divided into chunks; each chunk is to be processed independently. In this experiment, the partitioning is conducted more than once to conclude the relationship between the execution and the number of partitions. The aim is to find the number of partitions that yield the highest speed-up.

Fig.2 explains the image partitioning by deciding the chunk size in terms of number of rows and columns. The image is partitioned to equal-sized chunks. It is assumed that the size of the image is divisible by the number of chunks, and therefore, there are no left-overs. For example, if the numbers of rows and columns are set to three, the algorithm divided the original image into nine chunks. Then, each sub-image is saved and assigned to a node.The partitioning process is conducted in parallel. In other words, all sub-images are produced and distributed simultaneously.

Canny edge detection is then applied on each sub-image independently and simultaneously. Thus, all sub-images are processed in parallel.

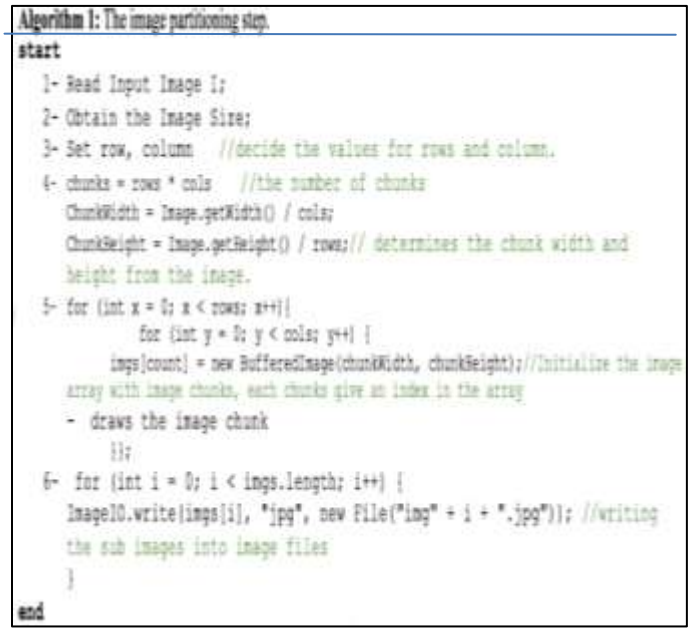


Figure 2. The Pseudo-code of the the image partitioning.

-Image Merging

As soon as a sub-image is processed; it is sent to a specified node for merging. This results in the original image after being subject to the Canny operator. Fig. 3 illustrates the pseudo-code for merging the sub-images. First, the same chunk size - in terms of numbers of rows and columns - as previously determined in the partitioning step is used. Then, all sub-image files are sent to a specific node that is responsible to execute the merging process. Finally, all sub-images are read in order, to get the final original image after applying the Canny edge detection.

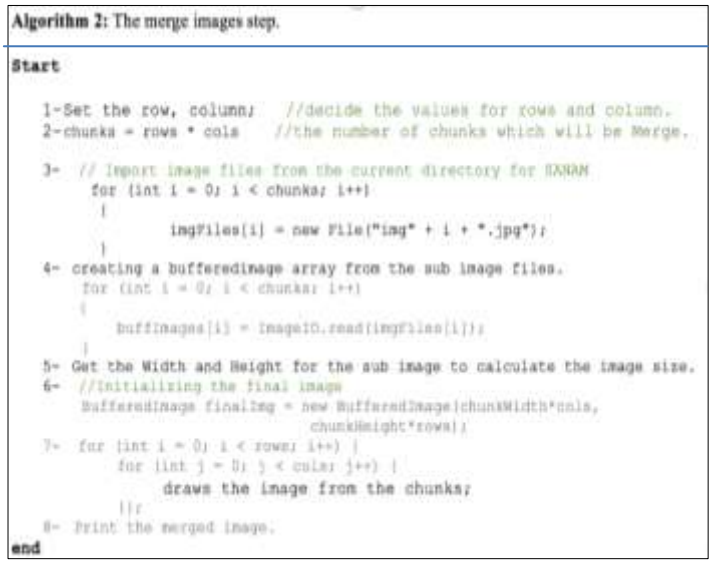
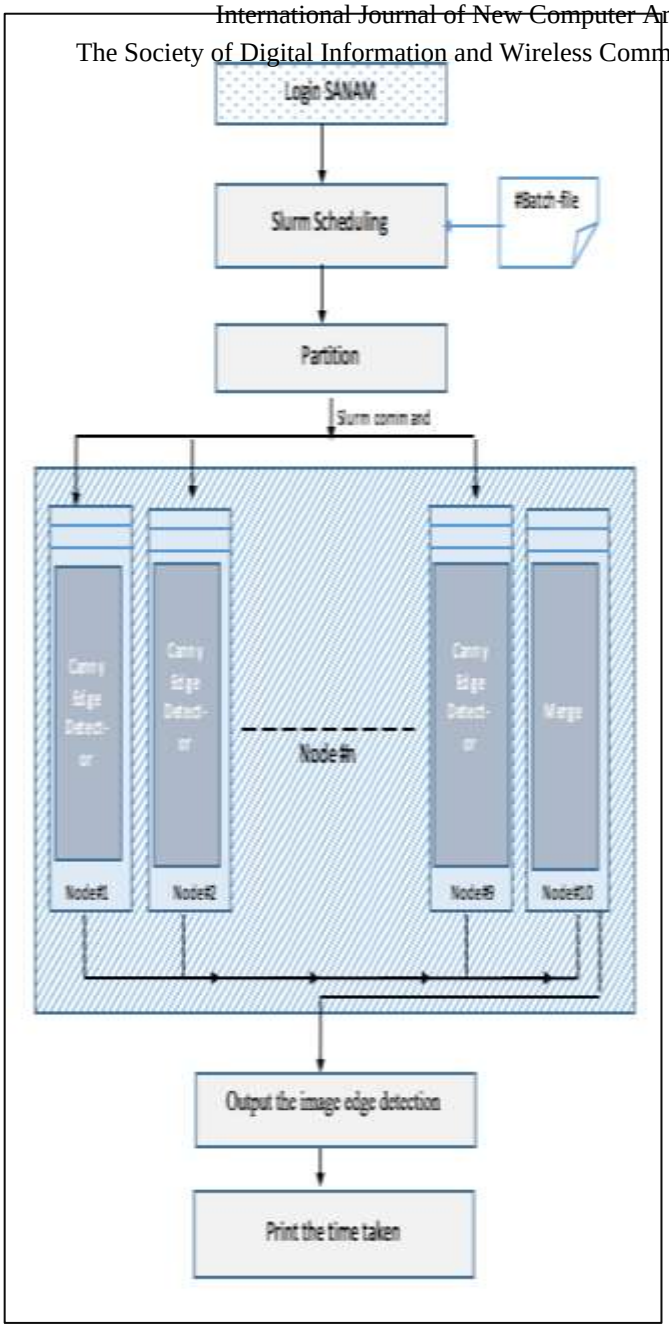


Figure 3. The Pseudo-code of the image Merging partitioning.

- Applying Canny Filter

The parallel edge detection algorithm follows the data parallelism which is most suitable to almost all image processing applications [1]. Parallel image processing is suitable for the clusters environment if there are little sequential dependencies. Consequently, there is minimum message passing involved between the computing processors [16]. Each node applies the sequential version of the Canny algorithm on different partitions simultaneously. The diagram in Fig. 4 depicts the previously detailed steps of the approach for edge detection cluster.

Figure 4. The architecture of parallel cluster.



5 PERFORMANCE METRICS

The performance of the parallel algorithm is measured by comparing it to the sequential algorithm. Three main metrics are used to measure such improvement; namely, the speedup, the efficiency, and the scalability. In addition, the experiments target to find the number of partitions that give the highest speed up in the parallel clusters environment. In addition, the accuracy of the edge detection is of major concern. These are detailed in the following sections

- Speed up

The speedup (S) is defined as the ratio of the execution time of the serial implementation (*Tseq)* over the execution time of the parallel implementation*(Tpar)*. Let *T(N, 1)* be the time required for the best serial algorithm to solve a problem of size *N* on one processor; and *T(N, K)* be the time for a given parallel algorithm to solve the same problem of the same size *N* on *K* processors. Thus, speedup is defined as [12]:

$$S(N, K) = \frac{T\,seq\,(N,1)}{T\,par\,(N,K)} \quad (1)$$

-Efficiency

The Efficiency (*E*) is a metric that identifies how close a program is to the ideal speedup. In other words, it measures the extent to which the program uses the hardware resources efficiently. *E* is a fraction between 0 and 1. A program whose *E* is closer to 1 makes better resources utilization. As the program's efficiency is closer to **1**, as it is closer to the ideal status. Efficiency is therefore measured using the following formula [13]:

$$E(N, K) = \frac{\text{Actual Speedup}}{\text{Ideal Speedup}} = \frac{\text{Speedup (N,K)}}{K} \quad (2)$$

-Scalability

The scalability of a program measures its ability to handle larger amounts of data. A program's size-up is the size of the parallel version running on *K* processors relative to the size of the sequential version running on one processor for a given running time *T*. This is expressed in the following formula [13]:

$$\text{Size-up}\,(T, K) = \frac{N\,par\,(T,K)}{N\,seq\,(T,1)} \quad (3)$$

6 EXPERIMENTAL RESULTS

In this section, we undergo a set of experiments on the parallel version of the Canny filter. The experiments' environment is detailed in Section 6.1. Section 6.2 explains the followed methodology in conducting all experiments. Section 6.3 investigates the optimum number of partitions that gives the

least execution time. Section 6.4 explores the speedup of the parallel program. Section 6.5 and Section 6.6 examine the efficiency and scalability respectively. Section 6.7 demonstrates the validity of the parallel program in edge detection. Finally, Section 6.8 compares our results with previous similar works.

6.1 Experiment Environment

In this research, the Canny image edge detector is implemented in two versions: the sequential, the and the parallel clusters. Both versions are implemented on the Saudi SANAM supercomputer.

SANAM belongs to King Abdel-Aziz City for Science and Technology (KACST). It is ranked second in the Green500 worldwide list of the most energy-efficient computers, as per the listing of November 2012. The KACST's supercomputer comprises standard servers connected via the high-speed network InfiniBand. It consists of 210 servers with 3,360 processor kernels [17]. In terms of computing speed, SANAM is about 40% faster than the German supercomputer "LOEWE-CSC". In addition, it requires only one-third of the power per computing operation. This is achieved by using a larger number of high-performance graphic chips in conjunction with software optimizations, as well as by using energy-efficient storage chips.

SANAM works under the Linux operating system. On the other hand, it uses Slurm: a highly scalable cluster workload manager and job scheduling system. It is available as an open-source basis and can deliver fault-tolerant cluster workload management for Linux clusters of various sizes [19]. A server includes two Intel Xeon E5−2650 CPUs (8 Hyper-Threading-enabled cores per processor). Each of the 256 server nodes has two AMD FirePro S10000 dual GPUs. Every node contains 128 GiB of main memory and has an on-board InfiniBand HCA [18].

6.2 Methodology

The measurement of the run time is a basic step in this research for both sequential and parallel codes. This is essential since the speedup is measured in terms of the execution times of both versions. However, practically speaking, the execution time of a program is rarely the same for multiple successive runs. This is because the underlying operating system is active all the time. Since it is impossible to control the OS activity, then a program is run multiple times (from 7 to 10 times), and the minimum value is recorded for the experiment. Taking the minimum value corresponds to the least interference of the operating system activities during the program run.

6.3 Optimum Number of Partitions

On the other hand, Table 1 depicts the execution time of the parallel version run on 10 nodes with different number of partitions. The parallel version is run on the same images used with the sequential version.

The table proves experimentally that for each image size, the execution time decreases with the increase of the number of partition. However, there is a different turning point for each image size: these are 50 partitions for the 1024x1024 image, and 60 partitions to the other two image sizes. At these turning points, the execution time begins to increase again. Fig. 5 depicts this relationship.

This is due to the increasing efforts to partition/merge larger number of sub-images. In addition, the limitation of the available hardware resources with respect to the number of partitions affects the execution time.

It is also noticed that the execution time increases with the increase of the image size for the same number of partitions. This is explained to the fact that the chunk sizes are equal for the three sizes; therefore, the number of sub-images increases with the increase of the image size.

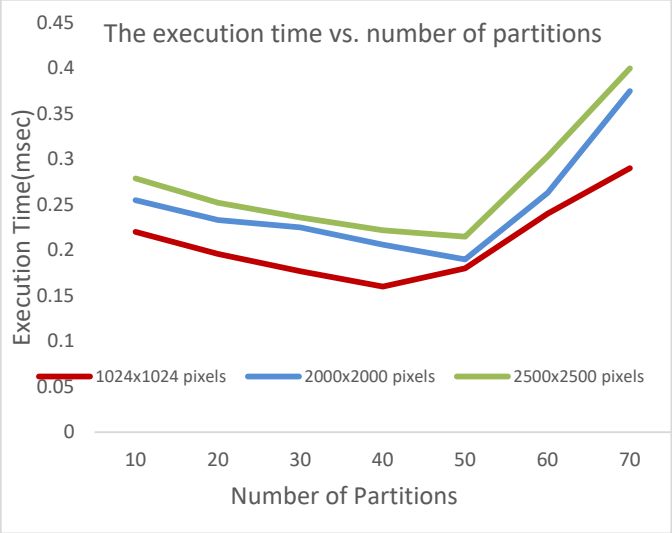


Figure 5.Relation of Execution Time with Number of Partitions.

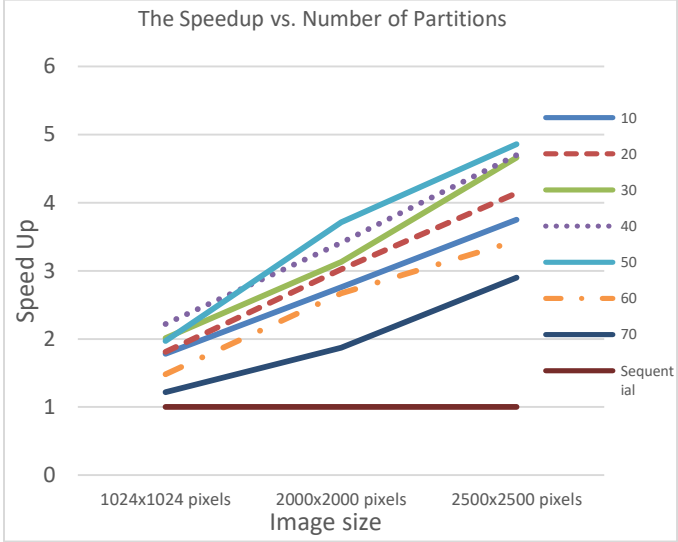


Figure 6. The speedup of SANAM Canny edge detection for the different number of partitioning.

6.4 Measuring Speedup

In this set of experiments, our aim is to test the performance of the parallel algorithm on Clusters. The sequential program is run on a single node; whereas the parallel version is destined to run on 10 nodes. In the parallel version, the same program runs with different sets of data on each node. Experiments are applied on three image sizes to measure the running time according to the methodology previously explained in section 6.2. The results of the sequential run are shown in Table 2.

It is noticed from Table 2, that the execution time is directly proportional with the image size. These results are expected since more processing is needed for larger images.

The minimum execution times for the parallel version are taken for the three image sizes. These correspond to number of partitions equal to 40, 50 and 50 for 1024x1024, 2000x2000 and 2500x2500 pixel images respectively. By applying formula (1), the speedup is found to be 1.97, 3.71, and 4.87 respectively for the three image sizes. More comprehensive results on all numbers of partitions are illustrated in Fig. 6.

From the graph, it may be concluded that the parallel version reduces the execution time of the sequential Canny filter. In addition, it is noticed that it is more beneficial to use the parallel version in case of larger images since it yields the maximum speed-up.

6.5 Efficiency

Table 3 and Fig. 7 depict the results of the efficiency experiments numerically and graphically respectively. Obviously, the efficiency increases with the increase of the number of partitions till a peak at which it starts to decrease. The maximum obtained efficiencies are 0.22, 0.37 and 0.49 for the three images respectively.

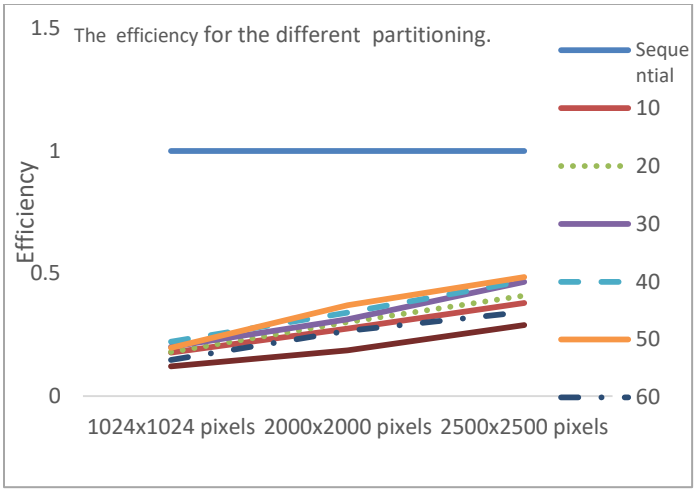


Figure 7. Relationship of Eefficiency of the parallel Canny with Number of Partitions.

6.6 Scalability Measurement

The scalability of a parallel program is performed as follows: The sequential program is run many times with differenct image sizes till the run crashes. The last size before the crash is recorded as *Nseq*. This represents the largest problem size that can be handled by the sequential program. The same step is repeated for the parallel program. *Npar* represents the largest size that can be handled by the program. Then, the scalability factor, or the sizeup, is calculated according to formula (3) above.

Since the sequential program runs on a single node, and the parallel version on ten nodes, *Nseq* and *Npar* are denoted as *N1* and *N10* respectively.

Fig. 8displays the execution time of different image sizes till the program crashes. The blue and red lines in the graph represent the results of the sequential and the parallel versions respectively.

From the graph, the program crashes at sizes equal 5Kx5K and 9Kx9K for the sequential and parallel programs respectively. Therefore, the sizeup is calculated as follows:

$$\text{Scalability factor (10 nodes)} = \frac{N10}{N1} = \frac{9000}{5000} = 1.8$$

A shared memory processor (SMP) environment lacks scalability [10]. However, this limitation is when using SANAM clusters, which can deal with the images of larger sizes.

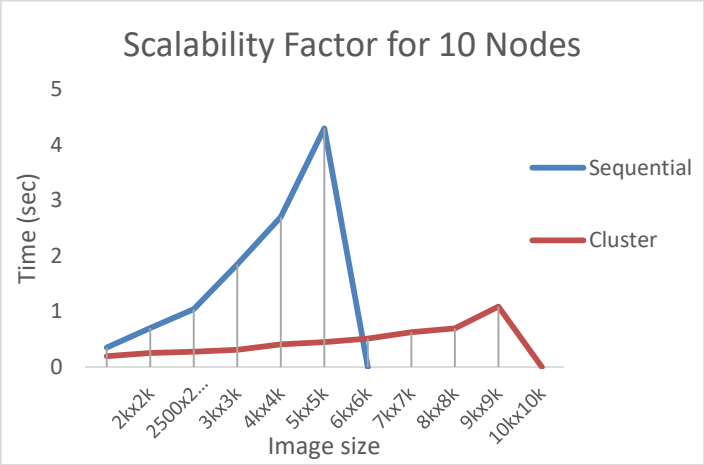
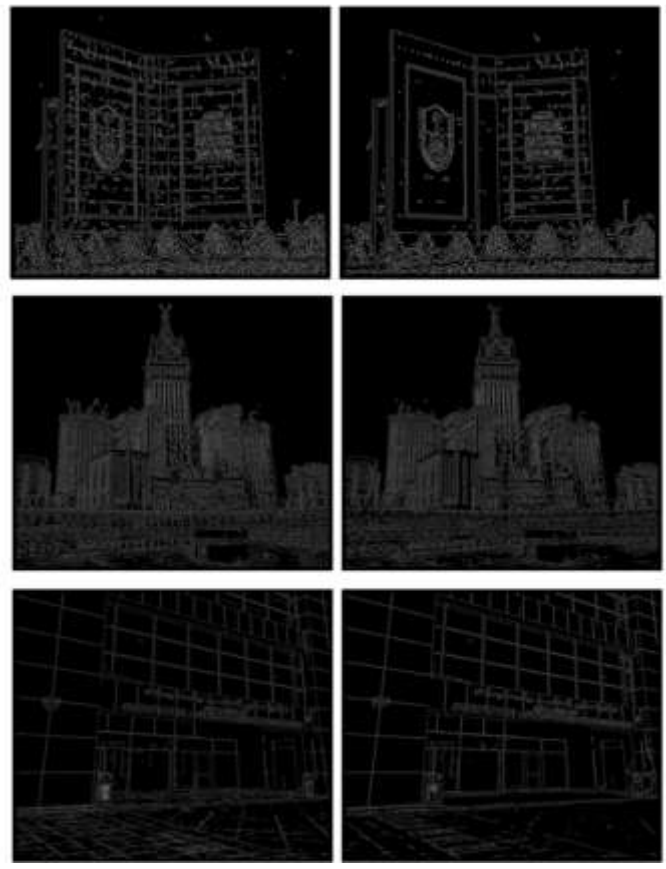


Figure 8. Relationship of Exectution time with Image Size.

6.7 Edges Accuracy

In this section, the accuracy of the parallel program is investigated. The importance of such experiment is to ensure the validity of the parallel program. Fig. 9 shows the original images in (a); the resulting detected edges using sequential program and the Cluster programs are shown in (b) and (c) respectively. Obviously, the edges are more precise in the images resulting from the parallel program compared to their counterparts produced from the sequential program.

(a)



(b) **(c)**

Figure 9. The edge detection results of the three images by using sequential and Cluster versions.

6.8 Comparison with Similar Work

In this section, the speedup results are compared to previous similar work. Our comparison is limited to those studies that used the same sizes of images as our experiments for more accurate conclusions.

In [11], they used an image 1024x1024 pixels in their experiment by using only 4 clusters. Our program outperforms by 81.98%. In [12], the experience is conducted on an image of size 2000x2000 pixels. SANAM gave a better result rate by 81.13%. Finally, in [13], they applied the approximation experiment using 10 clusters. The results are almost the same, slightly in favor of SANAM's result with a percentage close to 0.81% The comparisons are summarized in Fig. 10.

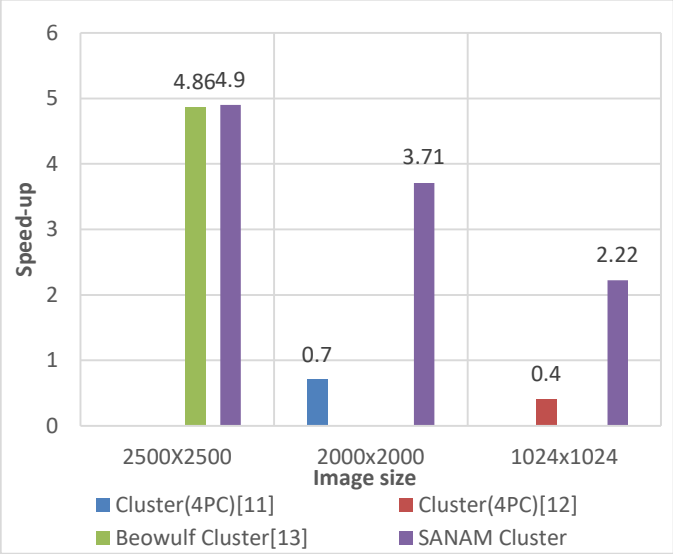


Figure 10. Comparison with Similar Work.

7 CONCLUSION

This researchimplements the Canny edge detection algorithm on parallel clusters. Ten nodes are used on KACST's SANAM supercomputer.

The image is partitioned, the Canny filter is applied on each sub-image simultaneously, and then the sub-images are re-collected and merged to give the edges of the big image.

Experiments target to find the optimum number of partitions. Also, the speedup, the efficiency, and the scalability of the parallel program are investigated. In addition, the accuracy of the detected images is examined. It is found that the speedup, the efficiency, and the scalability are drastically increased. In addition, the edges are more precise as compared to the results of the sequential program.

Our work is compared to previous similar work, and it is found that it gives the highest speedup.

ACKNOWLEDMENT

We would like to extend our gratitude to King Abdel-Aziz City for Science and Technology (KACST) in Riyadh for allowing us to use their SANAM supercomputer at all times. Not only this, but the staff was also of great support and helpful; replying to our posed questions promptly albeit their heavy duty load.

REFERENCES

[1] M. Raman R., and H. Aggarwal. "Study and comparison of various image edge detection techniques". International Journal of Image Processing (IJIP), Vol. 3, No. 1, 2009.

[2] A.-A.-N., Y. Kong, and M. N. Hasan, "Performance analysis of Canny's edge detection method for modified threshold algorithms," 2015 International Conference on Electrical & Electronic Engineering (ICEEE), 2015.

[3] B. Parhami. "Introduction to Parallel Processing: Algorithms and Architectures". Boston, MA: Springer US, 2002.

[4] M. Raman R., and H. Aggarwal. "Study and comparison of various image edge detection techniques," International journal of image processing (IJIP) Vol.3 no.1, 2009.

[5] A.-A.-N., Y. Kong, and M. N. Hasan, "Performance analysis of Canny's edge detection method for modified threshold algorithms," 2015 International Conference on Electrical & Electronic Engineering (ICEEE), 2015.

[6] M. R. Vahidi, M. M. RiahiKashani and A. Bagheri, "An efficient gradient based algorithm for improving performance of image edge detection," International Journal of Computer Applications, 103(4), 2014.

[7] A.-A.-N., Y. Kong, and M. N. Hasan, "Performance analysis of Canny's edge detection method for modified threshold algorithms," 2015 International Conference on Electrical & Electronic Engineering (ICEEE), 2015.

[8] A. Jain, A. Namdev and M. Chawla, "Parallel edge detection by SOBEL algorithm using CUDA C," 2016 IEEE Students' Conference on Electrical, Electronics and Computer Science (SCEECS), pp. 1-6, 2016.
[9] D. Yuefan. "Applied Parallel Computing," Singapore, US: Imperial College Press, 2012.
[10] NjoodAlAssemi, Soha S. Zaghloul, Phd. Speeding Up Canny Edge Detection Using Shared Memory Processing. International Journal of New Computer Architectures and their Applications (IJNCAA). Vol. 7, No. 3, September 2017, pp. 68 – 76.
[11] N. E. A. Khalid, N. M. Noor, S. A. Ahmad, M. H. Rosli, and M. N. Taib, "Parallel Laplacian Edge Detection Performance Analysis on Green Cluster Architecture," Communications in Computer and Information Science Digital Enterprise and Information Systems, pp. 308–316, 2011.
[12] W. Ju, J. Liu and S. Jin, "An improved clustering based on edge detection method," 2016 35th Chinese Control Conference (CCC), Chengdu, pp. 4026-4030, 2016.
[13] Haron N., Amir R., Aziz I.A., Jung L.T., Shukri S.R., "Parallelization of Edge Detection Algorithm using MPI on Beowulf Cluster," Innovations in Computing Sciences and Software Engineering, Springer, 2010.
[14] A. Muntasa, "Hybrid Method Based Retinal Optic Disc Detection," International Journal of New Computer Architectures and their Applications, vol. 5, no. 3, pp. 102–106, 2015.
[15] G. M. H. Amer and A. M. Abushaala, "Edge detection methods," 2015 2nd World Symposium on Web Applications and Networking (WSWAN), Sousse, pp. 1-7, 2015.
[16] A. Kaminsky, "BIG CPU, BIG DATA: Solving the World's Toughest Computational Problems with Parallel Computing" 2nd ed, Boston, MA: Course Technology, Cengage Learning, 2015.
[17] KACST The Saudi Supercomputer "SANAM" is the World's 2nd Leader in Energy Efficiency. [Online]. Available: https://www.kacst.edu.sa/eng/about/news/Pages/news3841117-3854.aspx . [Accessed: 28-Feb-2017].
[18] D. Rohr ndS.Kalcher., "An Energy-Efficient Multi-GPU Supercomputer," 2014 IEEE Intl Conf on High Performance Computing and Communications, 2014 IEEE 6th Intl Symp on Cyberspace Safety and Security, 2014 IEEE 11th Intl Conf on Embedded Software and Syst (HPCC, CSS, ICESS), Paris, pp. 42-45, 2014.
[19] "Slurm Workload Manager," Slurm Workload Manager. [Online]. Available: https://slurm.schedmd.com/. [Accessed: 05-Dec-2017].

Appendix

Table 1: Results of parallel execution time in SANAM.

Number of Nodes = 10 Nodes, Time= milliseconds.			
Image size / Partition	1024x1024 pixels	2000x2000 pixels	2500x2500 pixels
10	200	255	279
20	196	233	252
30	177	225	236
40	160	206	222
50	180	190	215
60	240	263	303
70	290	375	360

Table 2: Results of sequential execution time in SANAM.

Number of Nodes = 1 Node			
Image size	**1024x1024 pixels**	**2000x2000 pixels**	**2500x2500 pixels**
Time (milliseconds)	355	704	1045

Table 3: Efficiency of Parallel Program for Various Number of Partitions.

Number of Nodes = 10 Nodes			
Image size / Partition	1024x1024 pixels	2000x2000 pixels	2500x2500 pixels
10	0.178	0.276	0.375
20	0.181	0.302	0.414
30	0.201	0.313	0.466
40	0.222	0.341	0.470
50	0.197	0.371	0.490
60	0.148	0.267	0.344
70	0.122	0.187	0.290