Parallel approach of Sobel Edge Detector on Multicore Platform

N.E.A.Khalid, S.A.Ahmad, N.M.Noor, A.F.A.Fadzil and M.N.Taib

***Abstract*—** This paper presents the parallel multicore Sobel edge algorithm which parallelizes the traditional sequential Sobel edge detection algorithm on a parallel multicore platform. Dealing with images, the algorithm inherits repetitive instructions that depend on the image size, thus may slow down the processing speed. The multicore architecture is a ready available resource on ordinary personal computer but often not fully utilized to its utmost potential. Align with this hidden opportunity, Sobel edge algorithm can be implemented on parallel programming paradigm by focusing on the thread operations. This work presents the parallel multicore Sobel edge algorithm which parallelizes the traditional sequential Sobel edge detection algorithm on a parallel multicore platform via Parallel communication software named Message Passing Interface (MPI). The test is being done on ten different images with each image tested in the varying size of 1KxK, 2KxK and 3KxK pixels. Various threads, ranging from two to ten had been performed on Duo and Quad cores. An interesting result shows that in sequential processing Duo core overcome Quad core speed. As for parallel processing, two thread is the best used for Duo core and eight threads is finest for Quad core.

***Keywords*—**Parallel programming, Sobel edge algorithm, Message Passing Interface (MPI), Multicore.

I. INTRODUCTION

PARALLEL programming or parallel computing is an exciting and promising area to be explored today especially due to the decreasing cost of computer hardware [1]. Beside that most organization has a network of computers available in every department. As for universities, there are many computers in lecturer rooms and in computer labs abandon after office hours. These available resources can be utilized by implementing parallel computing.

Manuscript sent on August 15, 2011:

N.E.A.Khalid and N.M.Noor with Faculty of Computer and Mathematical Science, Universiti Teknologi MARA, Shah Alam, 40450, Malaysia as senior lecturer (e-mail: elaiza@tmsk.uitm.edu.my).

S. A. Ahmad is with Faculty of Electrical Engineering, Universiti Teknologi MARA, Shah Alam, 40450, Malaysia as graduate student (e-mail: arpah@ tmsk.uitm.edu.my).

A.F.A.Fadzil is a final year student of Bachelor Degree in Computer Science with Faculty of Computer and Mathematical Science, Universiti Teknologi MARA, Shah Alam, 40450, Malaysia. (ahmadfirdausfadzil@gmail.com)

M.N.Taib currently the Professor and the Dean of Faculty of Electrical Engineering, Universiti Teknologi MARA, Shah Alam, 40450, Malaysia. (phone: 603-5543 5034; fax: 603-5543 5077; e-mail: dr.nasir@ ieee.org).

This area promised a multitasking process and may solve bigger problem in less time. These criteria is very significant in today's era which every application that relate to science and engineering has to deal with a large amount of data and demand the real-time or near real-time performance [2].

Parallel computing is suitable for applications that required huge amount of computer power such as in modeling physical systems in many field of science, medicine and engineering. Modelers, whether to predict the weather or render a scene in the next blockbuster movie, can usually use whatever computing power is available to make the simulations more detail. Vast amount of data, whether customer shopping patterns, telemetry data from space, or DNA sequences, require analysis. This kind of analysis can be utilized on parallel computing platform with the multitasking processes. In biomedical application, such as X-ray image processing is a potential area to be implemented on parallel computing platform because in this area there exists specific solution that do not allow generalization [2].

The advancement of processor technology had produces the multicore computer. Multicore processors are the solution to the ever increasing of computing demand required and stressed upon the processors. As the computing power of the core is restricted by heat and size, duplicating the core for more potential computing resource is the viable current solution. Hence, this additional computing resource on a separate core creates a drive towards changing the application development in order to fully utilize these computing powers. Multicore processors provide a better power consumption without sacrificing the processing speed [14] thus making it a new standard in the CPU market. Utlizing the multicore by parallel programming is a challenge due to its complex interacting facet of performance based on memory consumption, processor utilization and synchronization and communication costs [2]. Trade-off have to be made among these facets to achieve the goal of better performance and utilization. But in implementing parallel solution certain conceptual and programming challenges have to be investigated further.

Image processing is one of the areas that sought tremendous processing prowess. The image that required to be processed is often very large but the processing needs to be fast [3]. When such cases happen, the most imminent solution is by adapting a new and advance hardware that is capable to accommodate the processes [4]. One of the solutions is by using Graphical Processing Unit (GPU) to optimize the imaging algorithms. However, this solution is expensive and the implementation process is complex [16]. Beside that the

graphic card solution such as GPU, creating the gap between the processors and memory speed [17].

The usage of Multicore to speed up image processing task is cheaper solution compare to GPU [18]. Image processing task such as edge detection algorithms are suitable and prove to be successful and economical to be implemented on the desktop personal computers [18]. Clustering algorithm is very practical to be implemented on multicores platform [19]. Wang and friends [19], had tested the k-mean and mean shift, a popular clustering algorithms, on the multicore platform using threads. Their results shows parallel implementation could achieve linear speedup to four threads with various parallelization. Issues about multicore performance such as memory access [17] and load balancing [20] had been investigated and proven that image processing tasks are beneficial to be implemented on multicore via parallel computing paradigm.

Bearing these issues in mind, the better solution towards solving this problem is by fully utilizing the current multicores hardware, to fulfill its utmost potential and subsequently creating a system that is able to accommodate the massive processing requirement [5]. One way to exploit the multicore architecture is by parallelization the operation of multiple threads on different cores using parallel communication software named MPI and examined the performance. While MPI is normally used to be implemented mostly in distributed memory architecture such as SMP cluster [13], it also supports multithreading [12] which in turns allowing it to be implemented on shared memory architecture such as the multicore. Other reason to adapt MPI is also due to the fact that the future works for this paper involves clustering the multicore architectures. Usually, parallelism involving multicore uses OpenMP as the communication model [11].

This paper aims to analyze the performance of multicore architecture on the application of Sobel Edge detector implemented on various thread processing via the parallel programming paradigm.

This paper had been organizes as follows. In section 2, the methodology is described in detail. Section 3 elaborates the results and analysis of the finding. Finally section 4 is the conclusion.

II. Material and Method

A. Material

Ten images of size 3Kx3K pixels are used as the initial images. These images are then resized using imaging tools into smaller dimensions of 1Kx1K and 2Kx2K pixels. The images are colored and roughly are scenery, animals and part of car images. The sizes of the imagers are modified to the specified sizes using Adobe Photoshop software.

B. Sobel Edge Algorithm

The edge detection algorithm named Sobel, that had been used in this work is a well known and established algorithm for detecting an edge in an image [6]. The algorithm aims to identify points in a digital image at which the image brightness changes sharply or more formally has discontinuities [6]. This algorithm has significant parallelism since it operates at pixel by pixel level. Edge detection is one of the central tasks of the lower levels of image processing which exhibit the need to program in parallel [7]. Sobel edge detector is based on the mathematical equation as given below:

$$|G| = \sqrt[2]{Gx^2 + Gy^2} \quad (1)$$

where *x* and *y* is the convolution kernels that is in a form of 3x3 mask. Figure 1 illustrates the convolution kernels that usually used for Sobel operator [5].

-1	0	1
-2	0	2
-1	0	1

0°

-1	-2	-1
0	0	0
1	1	1

90°

Fig. 1: Convolution Kernel

Despite the simple appearance of the formula which looks fairly simple to calculate, in terms of programming it involves a huge number of iterations within the program in to finish the operation.

C. Parallel architecture and Software Design

Parallel manner is more complicated. There are two ways of parallelization, data or task parallelism. This work used data parallelism as a digital image can be split into several parts to be processed by two processors of duo and quad core.

1. Hardware and Software

The multicore processors specification used in this work are duo core and quad core and is described in Table 1.

Table 1: Hardware Used

Component	Description	
# of Processor Cores	2 (Duo)	4 (Quad)
Processor	Intel® Core™ Duo E7500 @ 2.93GHz	Intel® Xeon® E5420 @ 2.50 GHz
RAM	3.46 GB DDR-2 RAM	3.46 GB DDR-2 RAM

The software required to perform the parallel process are Windows XP, Microsoft visual studio, Net Framework. MPICH 2 parallel communication software and CPU Monitoring software for performance measure as shown in Figure 2.

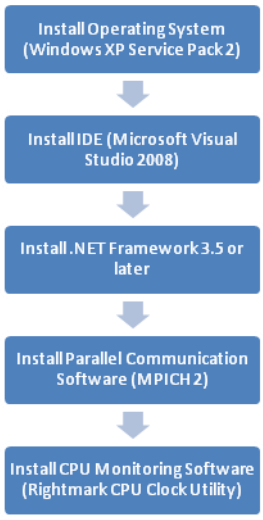

Fig. 2: Software Installation Flow

The parallel communication software used in this project is called the message passing interface or MPI. MPICH 2 is the software that enables message passing in parallel system.

The main role that the message passing interface (MPI) plays in this project is to execute programs in multiple threads, thus enabling all the central processing unit (CPU) to utilize each and every single core in order to accommodate the multiple threads execution. In this paper, we used 2 threads, 4 threads, 6 threads, 8 threads and 10 threads.

There are two ways of parallelization, data or task parallelism.

2. *Data Parallelization Process*

Data parallelism looks to be the simplest and feasible solution in this case, as a digital image can be split into several parts to be processed by different processor's core. The sequential versus parallel process design are depicted in the figure 3.

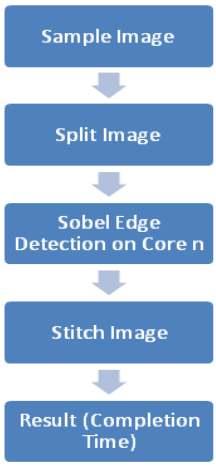

Fig. 3: Data parallelization Model

The data parallelism is achieved by splitting a single image into 2,4,6,8 or 10 subimagse which correspond to 2,4,6,8 and 10 threads used. Example of image being split into two subimages is depicted in Figure 4. After splitting the image, both parts of the image will undergo Sobel edge detector algorithm. Then each of the partition is executed in individual threads on a different core. Figure 3 illustrates how the split image looks after the Sobel edge detection algorithm is applied and how the parts are stitched back together upon the completion of the operation.

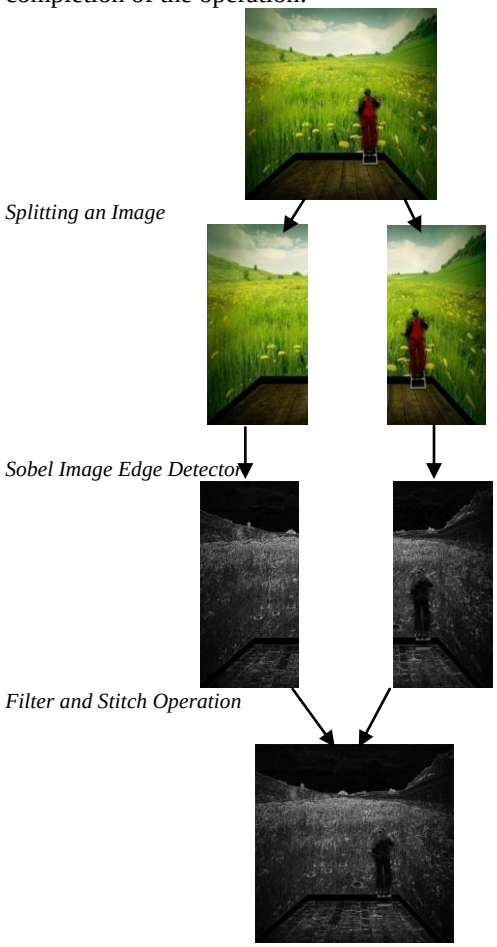

Fig. 4: Data splitting and stitching architecture

Basically, what the project does is that it filters an image from its original source using Sobel edge detection algorithm.

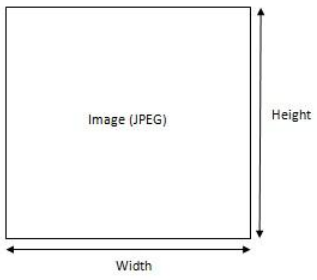

Fig. 5: Data parallelization Model

Figure 5 above illustrates a single image with the size of (Width)m x (Height)n. Therefore, in order for the program to finish its execution, it needs to iterate through the whole pixels of the image and applies the Sobel edge detection algorithm. The number of computation involved is relative towards the resolution of an image. 1000x1000 pixels image will require 1,000,000 numbers of iteration as the program loops through the width and height of the image. The algorithm that is used to filter the image is written in C# as in figure 6.

```
public static Image Mask(Image im)
{
    int[,] masking = new int[,] { { -1, 0, 1 }, { -
2, 0, 2 }, { -1, 0, 1 } };
    Bitmap b, b1;
    b = new Bitmap(im);
    b1 = new Bitmap(im);

    FastBitmap proc1 = new FastBitmap(b);
    FastBitmap proc2 = new FastBitmap(b1);

    proc1.LockImage();
    proc2.LockImage();

    //float max, min;
    for (int i = 1; i < b.Height - 1; i++)
    {
        for (int j = 1; j < b.Width - 1; j++)
        {
           int data3 = Math.Abs(((proc1.GetPixel(j -
1, i - 1).R
           + proc1.GetPixel(j - 1,
           i - 1).G + proc1.GetPixel(j - 1, i -
1).B) / 3) * masking[0, 0] +
           ((proc1.GetPixel(j - 1, i).R +
proc1.GetPixel(j - 1, i).G +
           proc1.GetPixel(j - 1, i).B) / 3) *
masking[0, 1] +
           ((proc1.GetPixel(j - 1, i + 1).R +
proc1.GetPixel(j - 1,i + 1).G +
           proc1.GetPixel(j - 1, i + 1).B) / 3) *
masking[0, 2] +
           ((proc1.GetPixel(j, i - 1).R +
proc1.GetPixel(j, i - 1).G +
           proc1.GetPixel(j, i - 1).B) / 3) *
masking[1, 0] +
           ((proc1.GetPixel(j, i).R +
proc1.GetPixel(j, i).G + proc1.GetPixel(j,
           i).B) / 3) * masking[1, 1] +
           ((proc1.GetPixel(j, i + 1).R +
proc1.GetPixel(j, i + 1).G +
           proc1.GetPixel(j,
           i + 1).B) / 3) * masking[1, 2] +
           ((proc1.GetPixel(j + 1, i - 1).R +
proc1.GetPixel(j + 1, i - 1).G +
           proc1.GetPixel(j + 1, i - 1).B) / 3) *
masking[2, 0] +
           ((proc1.GetPixel(j + 1, i).R +
proc1.GetPixel(j + 1, i).G +
           proc1.GetPixel(j + 1, i).B) / 3) *
masking[2, 1] +
           ((proc1.GetPixel(j + 1, i + 1).R +
proc1.GetPixel(j + 1, i + 1).G +
           proc1.GetPixel(j + 1, i + 1).B) / 3) *
masking[2, 2]);

           if (data3 > 255)
               data3 = 255;

           proc2.SetPixel(j, i,
Color.FromArgb(data3, data3, data3));
        }
    }

    proc1.UnlockImage();
    proc2.UnlockImage();

    return (Image)b1;
}
```

Fig. 6. The coding

The fact that multicore architectures need to be utilized properly in order to get it fully utilize does not help the situation. This means an available resource neglected when it is actually needed. In the end, the situation will lead to a very slow execution time. Below is the program fragment written in C++ that separates a single process into a multi-threaded executions.

```
int maxNum;
int nTasks, rank;

MPI_Init( &argc, &argv );
MPI_Comm_size( MPI_COMM_WORLD, &nTasks );
MPI_Comm_rank( MPI_COMM_WORLD, &rank );

STARTUPINFO si1 = {sizeof (STARTUPINFO)};
PROCESS_INFORMATION pi1;

int MPI_Barrier (MPI_Comm comm);

for (int i = 0; i < nTasks; i++)
{
  if (rank == i)
  {
    cout << "using thread " << i << endl;

  //build cmd line
  string arguments;

  arguments.append("C:\\Paraquest3\\SobelFilter.exe
");

  char *str = new char[arguments.size() + 1];
  std::strcpy ( str, arguments.c_str() );

  system(str);
  break;
  }
}
```

Fig. 7. The coding

The idea of using this approach is to make the CPU fully utilize its resources by separating the process into multiple individual threads. This will allow the core of the CPU to be used effectively.

The program above runs the algorithm by executing the applications using different number of threads. The user will specify the number of threads to be used for the execution.

Performance Evaluation Method

The evaluation of the parallel execution performance is measured with respect to speedup, performance improvement and efficiency with reference to the time taken for both sequential and parallel processing [3].

Speedup measures how much a parallel algorithm is faster than a corresponding sequential algorithm. The speedup calculation is based on Equation 2;

$$\text{Speedup} = \frac{\text{sequential(time)}}{\text{parallel(time)}} \quad (2)$$

The performance improvement depicts measurements relative improvement that the parallel system has over the sequential process. This performance is measured based on Equation 2;

$$\text{Performance Improvement} = \frac{\text{sequential(time)} - \text{parallel(time)}}{\text{sequential(time)}} \quad (3)$$

Efficiency is used to estimate how well-utilized the processors are in solving the problem, compared to how much effort is wasted in communication and synchronization. As for efficiency, the calculation is based on Equation 3;

$$\text{Efficiency} = \frac{\text{sequential(time)}}{\text{no of processor x parallel (time)}} \quad (4)$$

III. Result

The result discussion is divided into the performance of sequential process and parallel processes in the multicore processor.

Table 2: Sequential Result on Intel XEON E5420 and Intel CORE 2 Duo E7500

	1Kx1K		2Kx2K		3Kx3K	
	Quad Core	Duo Core	Quad Core	Duo Core	Quad Core	Duo Core
1	3.52	3.20	13.39	11.94	29.05	25.98
2	3.44	3.10	13.14	11.70	29.02	25.87
3	3.42	3.06	13.03	11.63	28.53	25.44
4	3.39	3.04	12.93	11.56	28.12	25.26
5	3.51	3.17	13.25	11.85	28.91	25.79
6	3.46	3.10	13.19	11.76	29.25	26.17
7	3.48	3.12	13.34	11.92	29.42	26.36
8	3.48	3.11	13.29	11.88	29.46	26.29
9	3.39	3.05	12.92	11.66	28.78	25.78
10	3.54	3.14	13.48	12.07	30.07	26.68

Table 2 is the result of sequential time taken between Xeon E5420, Quad core and Intel CORE 2 Duo, dual core processors. IKx1K, 2Kx2K and 3Kx3K stands for image sizes in pixel values. As expected the larger the image the higher the processing time. It is found that the quad core require more time to process Sobel edge detector compared to the Duo core. This could be explained by the overhead time needed to manage the duo core is lesser than the Quad core. The result is also largely due to the fact that the Quad core is inferior in terms of CPU clock speed, with the Duo core speed clocked at 2.93GHz while the Quad at 2.50 GHz. The result is illustrated graphically in Fig. 8.

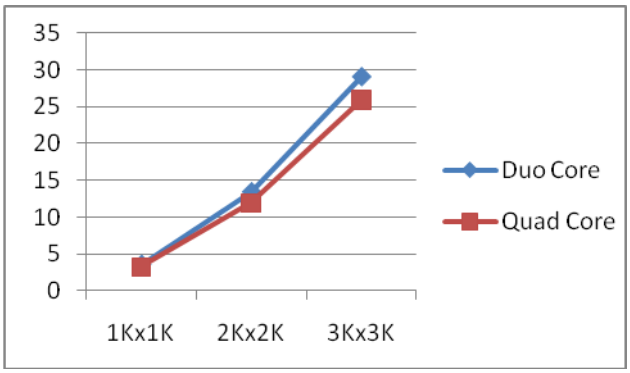

Fig 8: Difference Between Sequential Result on Intel XEON E5420 and Intel CORE 2 Duo E7500

Overall, the sequential result does not provide any substantial findings and it is close towards what was anticipated.

3.1 Parallel Results

The Parallel results are discussed based on speedup, performance improvements, efficiency and CPU Utilization Factor Using the Algorithm in performing the algorithm.

3.1.1 Raw Results Using Duo Core

Fig. 9 – 11, is the result of time (measured in seconds) taken between sequential and various thread (Th) sizes execution for 10 images with sizes of 1Kx1K , 2Kx2K and 3Kx3K respectively by using the Intel Core 2 Duo E7500. Similar to previous sequential results where larger image requires more time to be completed and it is definitely proving its point. The significant difference between execution time for different image sizes is apparent.

The focal point of this result is not base on the image sizes, but instead when executing on different number of threads. It is evident that by increasing the number of threads to 2, the execution time is significantly cut down.

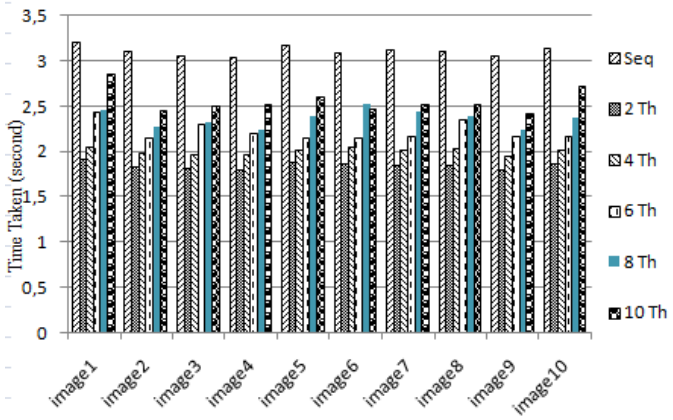

Fig. 9: Raw Results of 10 different images(1Kx1K) execution time using different number of threads on Duo Core

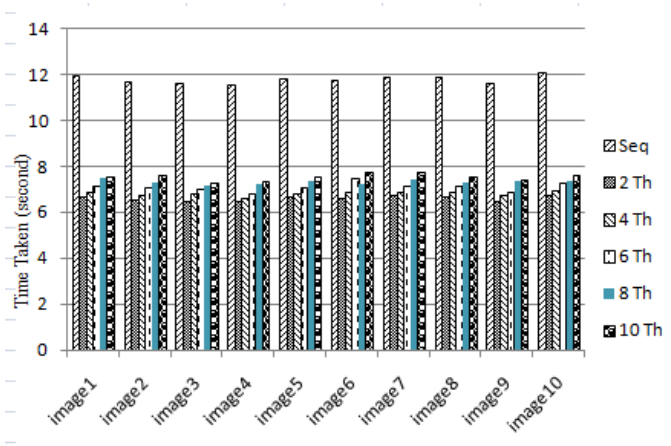

Fig. 10: Raw Results of 10 different images(2Kx2K) execution time using different number of threads on Duo Core

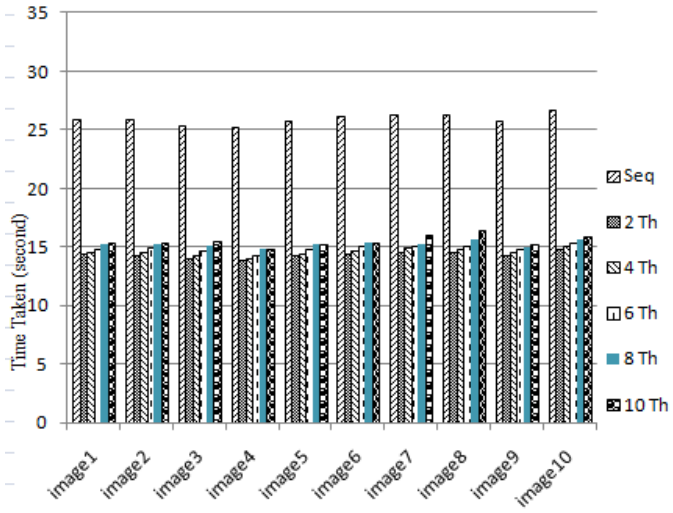

Fig. 11: Raw Results of 10 different images(3Kx3K) execution time using different number of threads on Duo Core

Attempts to improve the execution time further by increasing the number of threads for execution is pretty much failed, with every attempt returns an even slower execution time.

3.1.2 Raw Results Using Quad Core

Fig. 12 – 14, is the result of time (measured in seconds) taken between sequential and various thread (Th) sizes execution for 10 images with sizes of 1Kx1K , 2Kx2K and 3Kx3K respectively by using the Xeon E5420, Quad Core.

The results show that in the beginning, the result by using the quad core processors is similar to its duo core counterpart. Larger images as expected require more execution time and there is no significant difference. Things have otherwise changed when executing using different number of threads. Compare to the duo core processor, quad core still able to improve the execution time after increasing the number of threads to 8.

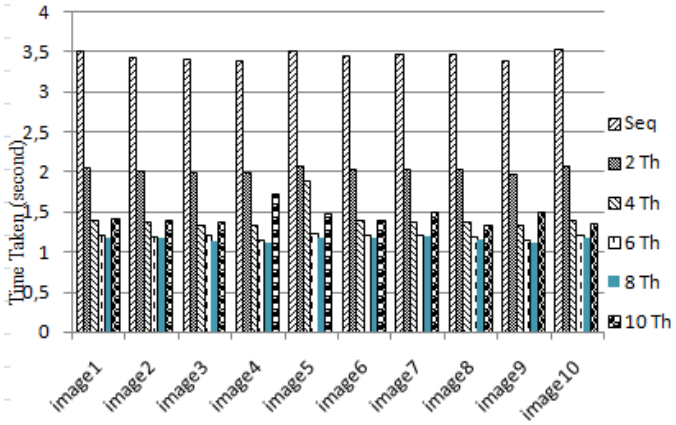

Fig. 12: Raw Results of 10 different images(1Kx1K) execution time using different number of threads on Quad Core

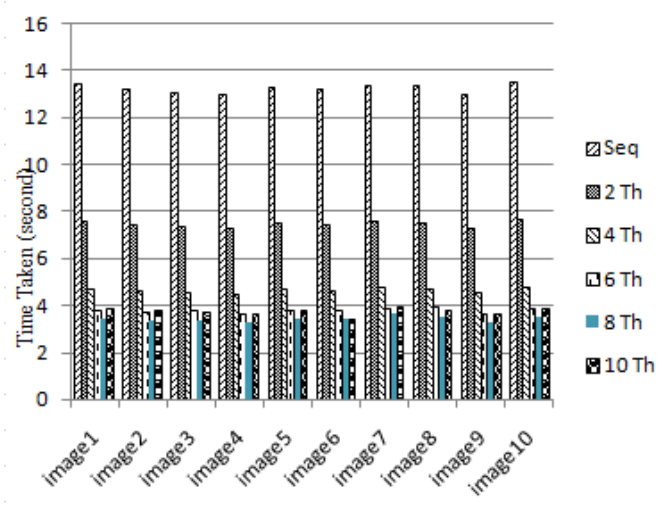

Fig. 13: Raw Results of 10 different images(2Kx2K) execution time using different number of threads on Quad Core

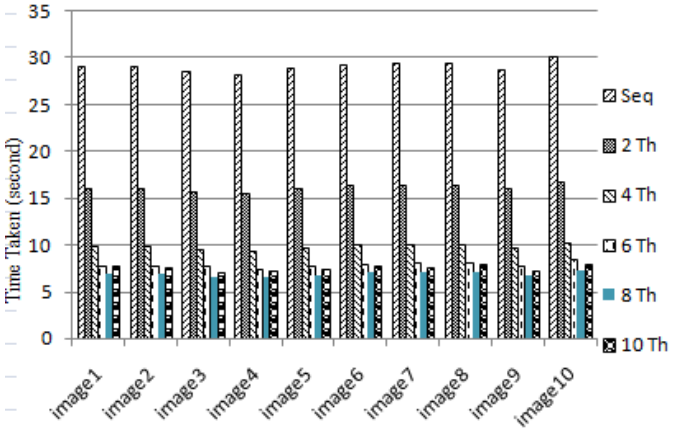

Fig. 14: Raw Results of 10 different images(3Kx3K) execution time using different number of threads on Quad Core

The quad core processor finally shows a different side when executing 10 simultaneous threads. Instead of improving the execution time, the result is a reverse.

3.2.1 Speedup

Fig. 15 and Fig.16 is the results of speedup of the utilization of 2, 4, 6, 8 and 10 thread between Quad and Duo CPU/core respectively. The speed up is based on equation (2) explained previously. Fig.17 shows the comparison for Speedup of in different threads. The results indicate that speedup values for almost all the images processed with duo core are close to two which indicates reasonably good performance whereas the quad core shows the speedup values of around 3.5 thus showing slightly less efficient use of all the processors.

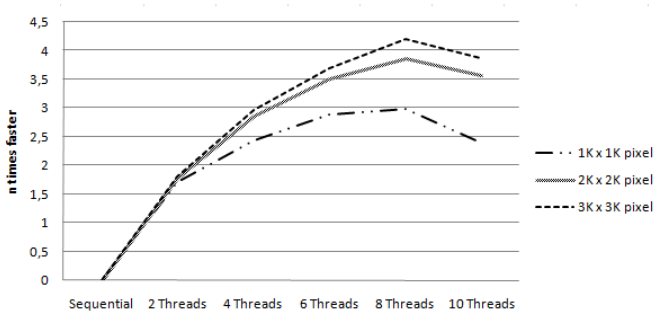

Fig. 15: Speedup of Different thread using Intel Xeon E5420

Two threads process work quite efficiently with the Duo core which reduces slightly with the increased in threads used. However as the number of thread increases, the more efficient the Quad core performance with the exception of ten threads. Thus, this indicates that two threads are best used for Duo core and eight threads are suitable for Quad core.

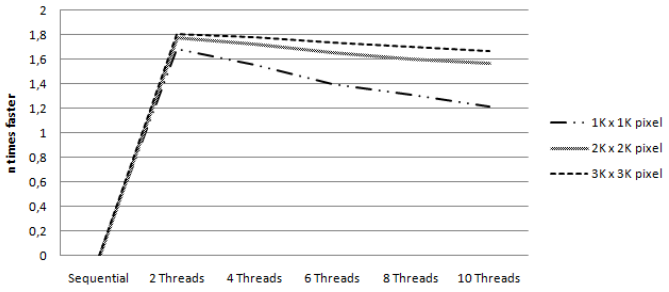

Fig. 16: Speedup of Different thread using Intel Core 2 Duo E7500

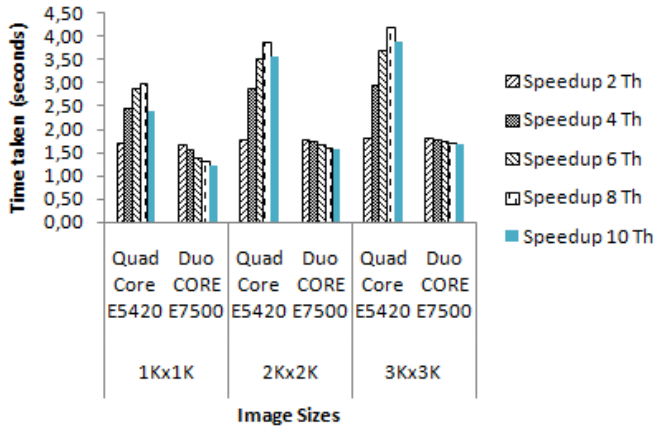

Fig. 17: Histogram of Comparison for Speedup of Different threads

3.2.2. Performance improvement Index

Fig. 18 and 19 show the results of performance improvement index which is based o equation (3) explained in previous section. The histogram of the performance improvement index of Duo and Quad core are depicted in Fig 18. The performance significantly improved for duo core from image with 1Kx1K to 2KX2K, however it does not indicate much improvement between the 2Kx2K and 3Kx3K. It is also found that there is an average of about 0.45 for the duo cluster compared to the quad core process which is mostly above 0.70 for images of 2Kx2k and 3Kx3K. The performance of the Duo core reduces for all case as the number of thread increases however the performance increases steadily with the exception of 10 threads for the Quad core. This indicates the best performance for Duo core is two threads and while eight thread is best used for Quad core compared to the sequential process.

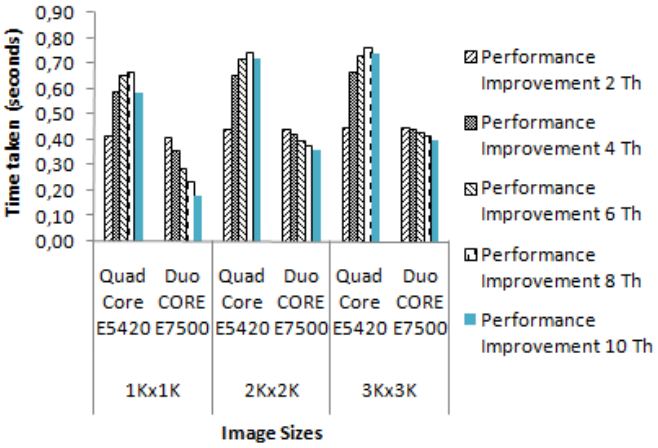

Fig. 18: Histogram of Comparison for Performance Improvement Index of Different threads

3.2.3 Efficiency

Fig.19 display the result of efficency utilization of the processors based om equation (4) earlier explained. The results shows higher efficency for Duo core compared to the Quad core. The larger the number of thread, the lower the efficiency. The reduction is much more significant for the Duo core. Efficiency is highest with the two thread for both Duo and Quad core This is true for all the image sizes. But the larger the image the higher the efficiency.

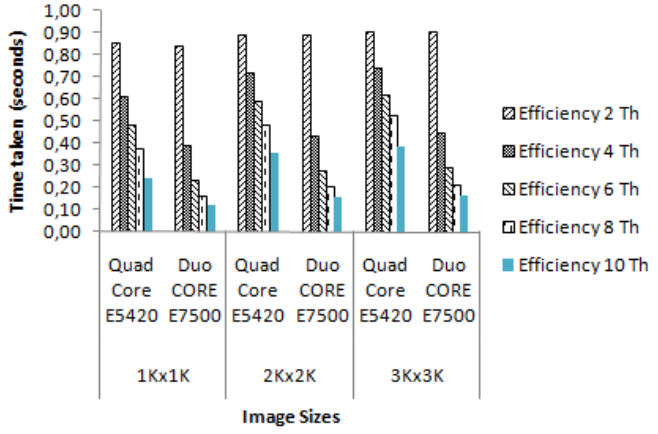

Fig. 19: Histogram of Comparison for Efficiency of Different threads

3.2.4 CPU Utilization Factor Using the Algorithm

The main objective of this paper is to fully utilize the CPU to its utmost potential. Therefore, CPU utilization needs to be monitored via the task manager to determine whether or not it met the criteria intended. Table 3 describes the CPU utilization factor when executing the sequential algorithm on both CPUs respectively.

Table 3: Sequential CPU Utilization Percentage

Processor	CPU Utilization
Quad Core	15%
Duo Core	50%

As suggested by the result above, the CPU utilization factor is nowhere near its full capabilities. The first CPU wasted 85% of the CPU's full capabilities while the second CPU wasted around 50%. Table 4 describes the CPU utilization percentage for each CPU.

Table 4: Parallel CPU Utilization Percentage

Processor	CPU Utilization			
	2Th	4Th	6 Th	8 Th
Quad Core	25%	50%	75%	100%
Duo Core	50%	100%	100%	100%

This research is about utilizing used PC in organization by demonstrating its usability in parallel implementation of image processing algorithm. Sobel edge detection algorithm had been successfully implemented in sequential and parallel manner. Basically both duo and quad have similar trends for all the method of performance such as speedup, performance improvement and the efficiency. Generally the performance measurement methods increases from 1Kx1K to 2Kx2K but reduces at 3Kx3K images. However, the parallel execution remains to outperform the sequential execution as indicated by the positive measurement for speedup and performance improvement but not in terms of efficiency.

The CPU utilization factor in a way, compliment the performance result earlier. The efficiency of the first processor to execute 8 concurrent threads at the same time is evidence as the CPU is only fully utilized when executing at this thread setup.The reason why the second processor failed to emulate the first processor's result is because it already fully utilized the CPU when executing only 2 threads. More threads won't make thing any faster as the resource is already fully utilized.

IV. Conclusion and Recommendation

It is certainly evidence that parallel multicore Sobel algorithm improves the performance of the traditional sequential Sobel algorithm by fully utilize the CPU to its utmost potential. Parallel processing performs better than sequential processing in terms of speed but with a trade off with the performance and the efficiency of utilizing the processors individually. However with the increasing amount of data sizes to be process, multicore provides a welcome alternative for fast processing. This research provides a gateway to identify suitable methods to process large data fast. This initial research can invoke the use of multicore in clustering environment. It also provides some knowledge in balancing the utilization of single core and multicore processors in heterogenous cluster environment. This work is not limited to image processing methods only but can be extended to other processor intensive application.

Acknowledgment

The authors acknowledge with gratitude to Research Management Institute (RMI), UiTM and financial support from E-Science Fund (06-01-01-SF0306) from the Ministry of Science, Technology and Innovation (MOSTI),Malaysia.

References

[1] B. Barney, *Introduction to Parallel Computing*. Retrieved from Lawrence Livermore National Laboratory: https://computing.llnl.gov/tutorials/parallel_comp/, 2010

[2] C .Lin and L.Snyder,"Principles of Parallel programming", Pearson International, 2009

[3] N. Haron, R. Ami, I. A.Aziz, L. T. Jung and S. R.. Shukri, Parallelization of Edge Detection Algorithm using MPI on Beowulf Cluster. *Innovations in Computing Sciences and Software Engineering* . 2010

[4] C. Szydlowski, Multithreaded Technology & Multicore Processors. *Dr. Dobb's Journal,* May 2005.

[5] S.Akhter and J.Roberts, Multi-Core Programming: Increasing Performance through Software Multi-threading, 2006

[6] R.C. Gonzales and R.E Woods," Digital Image Processing", Pearson Education International, 2002

[7] B.Allen, M.Wilkinson Parallel Programming, Techniques and Applications Using Networked Workstations and Parallel Computers, Pearson,2005.

[8] Z.Guo,W.Xu and Z. Chai, Image Edge Detection Based on FPGA. *2010 Ninth International Symposium on Distributed Computing and Applications to Business, Engineering and Science* . 2010

[9] R. L.Rosas, A. D.Luca and F. B. Santillan (). SIMD Architecture for Image Segmentation using Sobel Operators Implemented in FPGA Technology. *2nd International Conference on Electrical and Electronics Engineering (ICEEE) and XI Conference on Electrical Engineering (CIE 2005)*, 2005

[10] C.Szydlowski, Multithreaded Technology & Multicore Processors. *Dr. Dobb's Journal, May 2005*

[11] Abdel-Qader, J. H., & Walker, R. S. (2010). Performance Evaluation of OpenMP Benchmarks on Intel's Quad Core Processors. *WSEAS LATEST TRENDS on COMPUTERS (Volume I)* , 348-355, 2010.

[12] Qi, L., Shen, M., Chen, Y., & Li, J. (2004). Performance Comparison between OpenMP and MPI on IA64 Architecture. *n Proceedings of International Conference on Computational Science'2004.* , 338-397

[13] Huttunen, P., Ikonen, J., & Porras, J.*MPI Communication in SMP Clusters* Retrieved from WSEAS E-Library: www.wseas.us/elibrary/conferences/digest2003/papers/466-257.pdf

[14] Saravanan, V., Chandran, S. K., Punnekkat, S., & Kothari, D. P. (2011). A Study on Factors Influencing Power Consumption in Multithreaded. *WSEAS TRANSACTIONS on COMPUTERS Issue 3, Volume 10, March 2011* , 93-103

[15] Elsamea, A. A., Eldeeb, H., & Nassar, S. (2004). PC cluster as a platform for parallel applications. Retrieved from WSEAS E-Library: http://www.wseas.us/elibrary/conferences/miami2004/papers/484-163.pdf

[16] Image Processing on the Graphics card: GPU beats CPU, available : http://www.commonvisionblox.com

[17] M.G..Benjamin and D.Kaeli, " Stream Image Processing on a Dual-Core Embedded System", in SAMOS 2007, LNCS 4599, Springer-Verlag Berlin Heidelberg, 2007, pp.149-158.

[18] S.A. Ahmad, M.N.Taib, N.E.Khalid, H.Taib, N.M.Noor and A.F.A.Fadzil," Analysis of Parallel Multicore Performance on Application of Sobel Edge Detector", *Presented at conference of The 15th WSEAS International Conference on computers* , July 15-17th 2011, Corfu Island, Greece.

[19] H.Wang, J.Zhao, H.Li and J.wang, "Parallel Clustering Algorithms for Image Processing on Multi-core CPUs", *in Proceeding of 2008 International Conference on Computer Science and Software Engineering*, pp. 450-453.

[20] A.E.Mahdy and H.El-Shishiny, " An efficient load-balancing algorithm for Image Processing application on Multicore Processors", *IFMT '08,*

Proceeding of the 1st International forum on Next-generation multicore/manycore technologies".

N.E.A.Khalid, has received BSc (Comp Sc) from USM, Malaysia in 1985, MSc (Comp Sc) in University of Wales in 1992 and PhD (Comp.Sc) from , Universiti Teknologi MARA, Shah Alam in 2010. Currently hold position as senior lecturer in Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Shah Alam. Her current research interest is in Image Processing, Biomedical, Artificial Intelligence and Parallel Computing.

S. A. Ahmad has received B.Sc (Hon's) in Information Technology from Universiti Utara Malaysia in 1997and M.Sc in Electrical and Electronic Enginnering from Univerisity of The Ryukyus, Japan in 2002. Currently is pursing PhD in Faculty of Electrical Engineering, Universiti Teknologi MARA, Shah Alam. Her current research interest is in Image Processing, Biomedical, Computer Networking and Parallel Computing.

N.M.Noor has received Bachelor of Computer and Information Engineering from UIA, Malaysia and Master of Engineering from RMIT University in 2005. Currently hold position as senior lecturer in Faculty of Computer and Mathematical Sciences, Universiti Teknologi MARA, Shah Alam. Her current research interest is in Image processing, Biomedical and computer network and communication.

A.F.A.Fadzil is a final year student of Bachelor Degree in Computer Science with Faculty of Computer and Mathematical Science, Universiti Teknologi MARA, Shah Alam, 40450, Malaysia.

M.N.Taib has received B.Eng.(Electrical) from Univ. of Tasmania, Australia, MSc (Control System) with Distinction from Univ. Of Sheffield, UK and PhD (Control & Instrumentation), UMIST, UK. Currently he is a Professor and Dean of the Faculty of Electrical Engineering, Universiti Teknologi MARA,. His Professional Membership are as follows: Senior Member, IEEE, Member IET, SIAM, ISA, OSA, SPIE, AIP and ISEBI. His current research interest is on advanced signal processing with applications in intelligent control system, biomedical and pharmaceutical systems, process control & biotechnology, optical fiber sensors and microwave sensors.