Parallel Implementation of Image Matching with MPI

Sheikh, Ismail
Dept. of Electrical & Computer Engineering (ECE)
Ryerson University, Toronto, Canada
misheikh@ryerson.ca

Alfonso Oviedo, Alejandro Emerio
Dept. of Electrical & Computer Engineering (ECE)
Ryerson University, Toronto, Canada
alejandro.alfonso@ryerson.ca

Dr. Nagi Mekhiel
Dept. of Electrical & Computer Engineering (ECE)
Associate Professor, Ryerson University, Toronto, Canada
nmekhiel@ee.ryerson.ca

Abstract *— in this paper, the performance of parallel computing will be thoroughly discussed in the domain of image matching. The concept of image matching is widely used in the areas of security, medical and computer vision which require comparing two images for similarities. However, depending on the size of images, it is highly possible that the application computation cannot be handled in a single processor running a sequential algorithm. In order to overcome this limitation, parallel computing is introduced through the Message Passing Interface (MPI) library. In this project, for the comparison of two images, both images are first converted into grayscale and then are compared using the Sum of Square Differences (SSD) algorithm. Further, a parallel network of 12 processors was implemented to calculate the performance of the SSD algorithm between both images. The performance gain of 12, 8, 4 and 2 processors was compared with the performance of a single processor. The comparison results presented a linear relationship between the performance gain and the number of processors used for execution. Hence, it proves that there are significant benefits of parallelism on SSD applications.* **Keywords—MPI; parallel; multiprocessor; image matching; SSD; performance; gain; distributed memory**

I. INTRODUCTION

Image matching is widely used and a really important process in the field of digital image processing. There are many other fields which are heavily dependent on image processing such as security, medical and computer vision [1]. The term image matching refers to searching a template image inside a bigger main image through similarities between both images' pixels.

Some of the most common methods of image comparison are sum of absolute differences (SAD), sum of square differences (SSD) and normalized cross correlation (NCC) [2].

SAD algorithm is about taking the absolute difference between each pixel of the main image with the corresponding pixel of the template image. The minimum SAD value on the main image will result in the most similar portion of the template image.

SSD algorithm is about dividing the main image into multiple independent blocks of the size of the template image and calculating the sum of the square difference between both images. SSD algorithm outputs the comparison value to be exactly zero for two identical images.

NCC algorithm is about normalizing each pixel of both images by subtracting it from the mean and dividing it by the standard deviation of the image, before the comparison. The comparison using NCC algorithm results in a single value between 1 and -1, where 1 refers to a perfect match between both images and -1 indicates completely anti-correlated comparison.

All these methods are computational intensive due to repetitive operations on tremendous amount of data and require parallel processing for better performance. SAD is the least computational intensive algorithm but the communication overhead of parallel processing is very large which results in poor performance. NCC is a highly computational intensive algorithm since it requires the calculation of mean and standard deviation of both images before the comparison. However, the SSD algorithm is not as computational intensive as NCC and provides really low communication overhead as compared to SAD when processed in the parallel network. Thus, this paper uses SSD algorithm for the parallel implementation of the image matching for the best performance.

II. PARALLEL DISTRIBUTION MODELS

There are two types of multiprocessor models, Shared Memory Multiprocessor (SMM) and Distributed Memory Multiprocessor (DMM). In SMM models, the computation is divided among different threads of a single CPU using OPENMP. Similarly, in DMM models, the computation is divided in the network of several computers using MPI. This research paper mainly focuses on the DMM model since it provides better performance compared to SMM models due to high quality of work-load distribution [4][5]. However, even for the DMM models, it is really important that the problem is decomposed and executed independently among all the computers in the network. There are four key steps of parallel programming which can be used to design good parallel code.

A. *Decomposition*

Decomposition is about breaking the problem into small independent tasks. The goal is to find as many independent tasks so that each processor is busy all the time [6][7].

B. *Assignment*

Assignment is about converting decomposed tasks into processes. The objective is to balance work load among all the processors in the network [6].

C. *Orchestration:*

Orchestration is about labeling or naming data. The purpose of orchestration is to reduce synchronization and communication overhead from the processes before they can be assigned to processors for the computation [6].

D. *Mapping:*

Mapping is the last step in parallel programming. In this, each individual process is assigned to each available processor in the network [6]. Each individual processor computes their assigned data and upon completion, sends the final result to the host processor.

Thus, through these four steps of parallel programming, the image matching computation can be decomposed into small independent tasks. These four steps also reduce synchronization overhead and cost of communication which ultimately results in the best performance of parallel computation.

III. SSD ALGORITHM

The SSD algorithm is based on the comparison of two images, the image template (T), and a known image which we try to find inside the strange image (S). The image T is the size of NxN and it is presented by the function T(m, n); where (m, n) are the coordinates of each pixel. On the other hand, S is the size of MxM, a bigger image than T. The image S is presented by the function $S(m, n)_{ij}$; where the representation is only a sub-image of S, and same size as T. The coordinates of the top-left corner are presented by i and j; the starting point where T is being search on S [1]. The SSD algorithm is defined by eq (1) below.

$$D(i,j) = \sum_{m=1}^{N}\sum_{n=1}^{N}[S(m,n)_{ij} - T(m,n)]^2 \qquad (1)$$

As D gets closer to zero the similarity gets higher.

In this research, we decided to use the SSD algorithm since it is simple and computational intensive. These two factors motivate the idea of using parallel programming models to accelerate the process. In addition, each computed D is the completely independent to the others, which is another key factor for building a parallel model.

IV. SINGLE PROCESSOR

The analytical results of the single processor model performance of the SSD problem is defined by (2). Where T_{SP} is the time a single processor takes to compute all the D values in an S image. T_{PC} is the time that any specific processor takes to compute one cycle of pixel sum of squared differences. It linearly depends on T_{PC} and exponentially on S and T sizes. In this specific model, S and T have their own height (H) and width (W).

$$T_{SP} = (S_H - T_H - 1) * (S_W - T_W - 1) * (T_H * T_W) * T_{PC} \qquad (2)$$

Fig. 1 below shows the maximum of computation level is reached when T size is half of S size.

V. MPI

MPI is a parallel model based on messages passing among processes. The message passing approach is another factor that can also affect the performance of the parallel model beside the computation time [4]. According to this argument, we developed a model to determine the behavior of the parallel approach and from it, the ideal computation power is balanced with communication over-head (Com-oh) for the problem to really benefit from parallelism.

A. *Master-Slave Mode*

When Master-Slave mode is implemented, the master provides workload distribution and collects results. Each processor is assigned the computation of all the Ds defined by the same i for all values of j which means a complete column of S by T width.

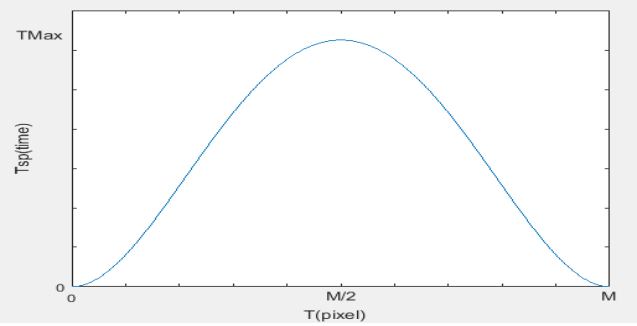

Fig. 1. Highest computation time when T size is half of S size

This approach infers that each processor compares all its own Ds and finally only has to send one D, the minimum D, to the master; resulting in a very low Com-oh. Thus, this Com-oh linearly depends on the amount of processors of the model. Equation (3) presents the model according to these arguments, where T_{MP} is the time a multiprocessor system takes to compute all the Ds values from one S. P is the number of processors used in the system and T_{comOH} is the time of Com-oh per processor [8]. T_{comOH} depends on the network topology, bandwidth, the time that each slave processor has to spend preparing the message and the time that the master processor has to spend receiving and interpreting messages from slaves.

$$T_{MP} = (T_{SP}/P) + (T_{comOH} * P) \qquad (3)$$

A three dimension plot is shown in Fig. 2 below to present the relationship between the number of processors, communication overhead and $T_{MP.}$

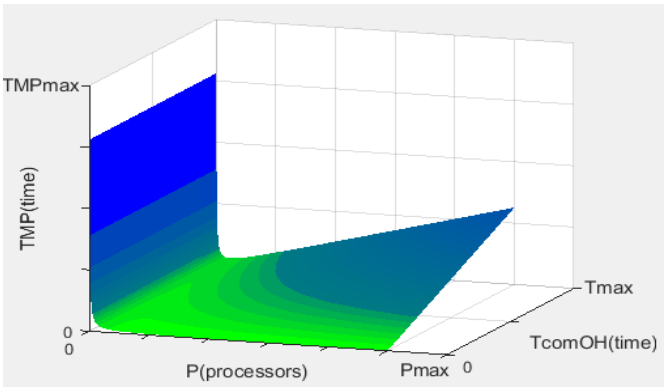

Fig. 2. Behaviour of Tmp vs P and TcomOH

Fig. 2 above presents the dependency of T_{MP} with the number of processors and the T_{comOH}. In the above figure, x-axis is presented by T_{ComOH} and y-axis represents the number of processors in the network and finally the z-axis represents the T_{MP}. It is very clear in the figure that the time a multiprocessor takes to compute one full cycle over S drops dramatically with the increase in the number of processors used in the system. However, if the T_{comOH} is really large, it increases T_{MP} in accordance to the number of processors.

The performance gain is another significant objective on these systems. The goal is having the gain as close as possible to the number of processors, to benefit from the implemented parallel system. Equation (4) gives the gain function depending on the time of a single processor over multiprocessor [8].

$$Gain=T_{SP}/T_{MP} \qquad (4)$$

A three dimension plot is shown in Fig. 3 below to present the relationship between the performance gain, number of processors and time for communication over-head.

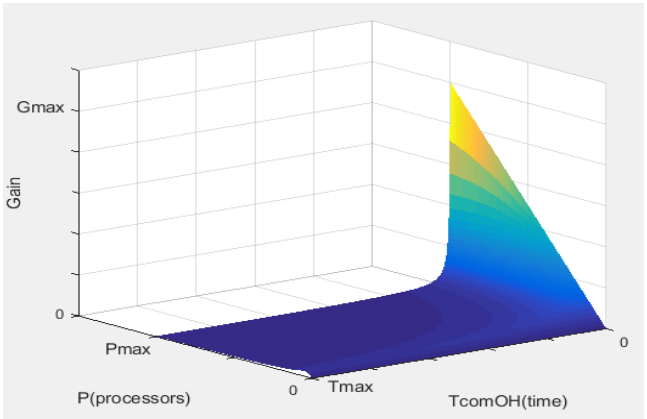

Fig. 3. Behaviour of Tmp vs P and TcomOH

In Fig. 3 above, the x-axis presents the communication overhead, the y-axis shows the number of processors and the z-axis presents the performance gain of the application. It is clear in Fig. 3 that the system performance gain is high when T_{comOH} is close to zero. Similarly as the T_{ComOH} increases, it dramatically reduces the gain of the system. Finally, when the T_{ComOH} is maximum, any additional number of processors has no impact on the performance gain of the system.

VI. Method of Image Matching

In order to compare two images through parallel computation, the main image S has been divided into multiple blocks of the same size as the template image T. After the decomposition, a copy of the template image T and the decomposed blocks are assigned to each processor. Each processor computes the SSD of both images and stores the minimum value of SSD; thus it has no communication overhead. Each processor can compute the assigned block without waiting to receive any data from other processors. Thus, it results in the best performance of parallel computing. After the execution of all decomposed blocks, each processor in the system sends its minimum value of SSD as well as the point of location to the host processor. Host processor, upon receiving the SSD results, compares it with its own SSD value. If the received SSD value is higher than the SSD value stored in the host processor, it is ignored. Similarly, if the received SSD value is smaller than the SSD value stored in the host processor, the host processor updates its SSD value to a new value. Finally, after finding the minimum SSD value between two images, the host processor sequentially converts the main image to RGB from grayscale and draws red lines of the same size as the template image on the point where the SSD is minimum. This process of image matching can be seen in the Fig. 4 below.

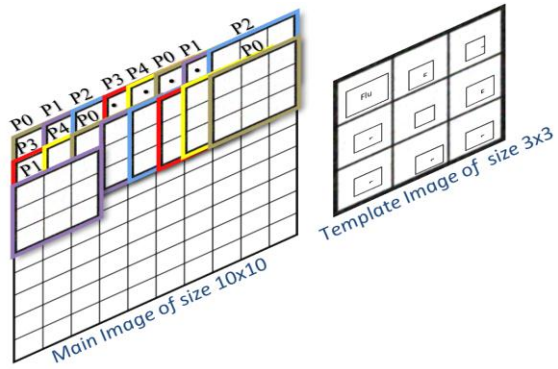

Fig. 4. Sample of Template and Main image

The Fig. 4 above shows the decomposition of the application with the main image size to be 10x10 and template image to be 3x3. The problem is executed in the system of 5 computers. In the decomposition process, there will be 64 independent blocks and processors p0, p1, p2 and p3 compare 13 blocks and p4 only compares 12 blocks.

VII. Performance Gain

Using the same image matching process as in section V, the performance of the parallel processor can be easily compared by running its computation sequentially in one computer and in MPI. The time in seconds was recorded and considered the

computational time of the application. The application details used for the result comparison are processor: Intel® Core™ i7-3770 with CPU clock speed of 3.40GHz. Size of main image is 3000 x 2000 pixels, and size of the template image is 500 x 500 pixels.

Table 1 below, shows the performance of the application in one computer, and in the network of 2, 4, 8 and 12 computers.

TABLE I. PERFORMANCE OF SYSTEM OF 1, 2, 4, 8, AND 12 COMPUTERS

Processors	Time (Sec)	Time (Min)	Gain
1	4724.79	78.74	1.00
2	2363.62	39.39	1.99
4	1183.32	19.72	3.99
8	593.00	9.88	7.95
12	396.59	6.61	11.92

One processor running sequential algorithm took 4724.79 seconds (78.74 minutes) to find template image inside the main image. However, running the same application on the network of 12 processors took only 396.59 seconds (6.610 minutes) to compute the result.

Further, the gain of the parallel program is dependent on the decomposition of the problem. Reducing the communication overhead, latency and delay in the decomposition will result in higher performance gain. Fig. 5 below represents the achieved gain of running the application in up to 12 processors.

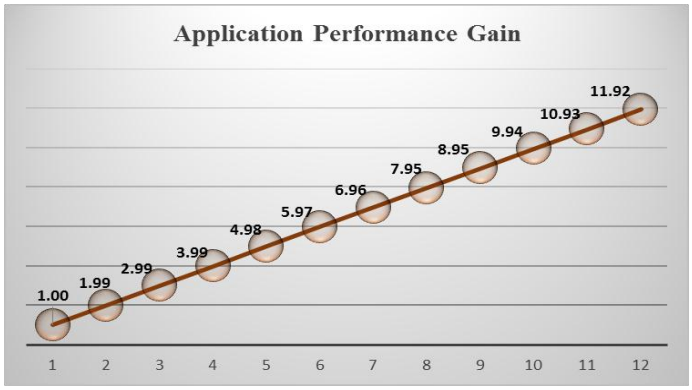

Fig. 5. Application Performance gain in the network

In Fig. 5 above it is clear that there is an almost linear relationship between number of processors in MPI and application performance gain due to the low communication over-head. This linear relationship means that the implemented parallel program in this paper benefits the image matching performance. However, as the number of processors increases the gain slightly starts to reduce. This is due to the increase in communication overhead and the saturation of performance gain. Thus, it is really important to determine the number of processors required for the computation in order to achieve the best performance of parallel computation.

VIII. CONCLUSION

This paper represents an almost linear relationship between the number of processors in the network and the application computation of the image matching application using the SSD algorithm. This relationship is achieved through the reduction of communication overhead. However, it is really important to determine the required number of processors for the application to avoid the gain saturation problem. The best performance gain of parallel implementation of the application is only possible if the cost of communication is at its minimum. In this paper, a unique approach of the SSD algorithm of image comparison is suggested to compute the results parallel in the network of computers with the minimum cost of communication. In the SSD algorithm, the image matching computation is divided in such a way that each computer in the network can execute their assigned workload independently without waiting for another processor to complete its execution.

IX. FUTURE WORK

At this stage, we are able to successfully match only template images if they exist in the main image with similar resolution. The SSD algorithm might result in a different value for a different resolution. In order to improve our approach, we can add some other layers of algorithms to scale the templates before comparison. This will allow us to compare all possible resolutions of the main and template images.

REFERENCES

[1] Liang Zong; Yanhui Wu, "A Parallel Matching Algorithm Based on Image Gray Scale," in Computational Sciences and Optimization, 2009. CSO 2009. International Joint Conference on , vol.1, no., pp.109-111, 24-26 April

[2] Pham, I.; Jalovecky, R.; Polasek, M., "Using template matching for object recognition in infrared video sequences," in Digital Avionics Systems Conference (DASC), 2015 IEEE/AIAA 34th , vol., no., pp.8C5-1-8C5-9, 13-17 Sept. 2015

[3] Jin Lu; Kaisheng Zhang; Ming Chen; Ke Ma, "Implementation of parallel convolution based on MPI," in Computer Science and Network Technology (ICCSNT), 2013 3rd International Conference on , vol., no., pp.28-31, 12-13 Oct. 2013

[4] Xueping Luo; Jinping Bai; Yunping Chen; Ling Tong, "Parallel implementation of MPI-based SAR image soil moisture inversion," in Geoscience and Remote Sensing Symposium (IGARSS), 2013 IEEE International , vol., no., pp.1692-1695, 21-26 July 2013

[5] Xiaoxin Tang; Mills, S.; Eyers, D.; Zhiyi Huang; Kai-Cheung Leung; Minyi Guo, "Performance Tuning on Multicore Systems for Feature Matching within Image Collections," in Parallel Processing (ICPP), 2013 42nd International Conference on , vol., no., pp.718-727, 1-4 Oct. 2013

[6] Culler, D., & Singh, J. (1999). Parallel computer architecture: A hardware/software approach. San Francisco: Morgan Kaufmann.

[7] Hartmann, O.; Kunemann, M.; Rauber, T.; Ruger, G., "A decomposition approach for optimizing the performance of MPI libraries," in Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International , vol., no., pp.8 pp.-, 25-29 April 2006

[8] Ruixin Ding; Junming Wu; Qinghua Huang, "Parallelism of Extended-Field-of-View Sonography based on Scale Invariant Feature Transform," in Biomedical Engineering and Informatics (BMEI), 2011 4th International Conference on , vol.1, no., pp.426-429, 15-17 Oct. 2011