International Journal of Software Engineering and Its Applications
Vol. 9, No. 5 (2015), pp. 135-144
http://dx.doi.org/10.14257/ijseia.2015.9.5.13

Image Segmentation Using OpenMP and Its Application in Plant Species Classification

M Nordin A Rahman[1], Ahmad Fakhri Ab. Nasir[1], Nashriyah Mat[2] and A Rasid Mamat[1]

[1]*Faculty of Informatics and Computing*
[2]*Faculty of Agriculture, Biotechnology & Food Science*
Universiti Sultan Zainal Abidin, Tembila Campus
22200, Besut, Terengganu, Malaysia
mohdnabd@unisza.edu.my, ahfan_022003@yahoo.com,
nashriyah@unisza.edu.my, arm@unisza.edu.my

Abstract

Segmentation is very important in early stage of image processing pipelines. Final results of image processing are strongly depending on the initial image segmentation quality. A good quality result often comes at the price of high computational cost including computation speed. Image segmentation requires long computation task caused by sequential processing of huge sizes of image and complex tasks. Nowadays, multi-core architectures are emerging as an attractive platform for parallel processing because it has two or more independent cores in a single physical package and their comparatively low cost. In this paper, two parallelization strategies (fine-grain and coarse-grain approach) are proposed for computing leaf image segmentation. The Canny Edge Detector and Otsu thresholding methods are used due to their wide range of usage for leaf segmentation in plant classification. The implementation is developed under multi-core architecture with shared memory multiprocessors. The OpenMP (Open Multi-Processing), an API (Application Programming Interface) is utilized for writing multi-threaded applications in shared memory architecture. The comparative study with two parallelization strategies is discussed further in this paper.

Keywords: *Image processing, image segmentation, parallel processing, OpenMP, leaf shape based classification*

1. Introduction

Plant species classification is one of image processing application that has received high attentions from scientists. In order to achieve better plant classification, the leaf image must have a good segmentation framework as the necessary preparation step before the processes of leaf features extraction and classification. The value of an image processing system is determined first by the classification accuracy, and second, by the computational cost to process the image.

Parallel architectures are considered an efficient solution to cut down the computational cost when implementation in this high-computation application. The motivations for parallel computing are the processing of large amount of data, parallel nature problems and the parallel processor capability offered [1, 8]. However, the potential benefits of multi-core processors do not come for free. In this study, we will show readers some guidelines on how to decide which parallelization strategy is suitable for a given multi-core architecture and tasks to reach peak performance.

This paper evaluates two different parallelization strategies using a quad-core CPU. The study aims at determining guidelines for the efficient parallelization of this

ISSN: 1738-9984 IJSEIA

application. The Canny Edge Detector [9, 11] and Otsu methods [12, 14] are used as a benchmark in the experiments since they have many common parallelism features, and widely used in plant species classification applications. Finally, the article provides a comparison analysis of two different parallel implementations for Canny Edge Detector and Otsu methods on a quad-core CPU.

The rest of the paper is organized as follows: The next section, Section 2 reviews a few related works on previous image segmentation using Canny Edge Detector and Otsu methods thresholding including their algorithms. Section 3 discusses the proposed parallelization strategies for Canny Edge Detector and Otsu methods thresholding. Experimental results of each strategy are presented in Section 4. Conclusion is placed in Section 5.

2. Related Works

There are limited literatures on parallel plant classification application unlike other application, such as for medical imaging application [15]. Generally, the plant classification processes consist of image segmentation, features selection and features classification. Parallel image processing is an alternative way to solve image processing problems that require large times of processing or handling large amounts of data. For instance, the segmentation process using Canny Edge Detection and Otsu methods require lots of memory space and time to process. In parallel processing, the computer program is written for executing multiple tasks that work together to solve a problem [16]. The main idea of parallel image processing is to divide the problem into simple tasks and solve them concurrently, in such a way the total time can be divided between the total tasks.

Domain decomposition and loop-level parallelism are well known techniques to divide the problems in parallel programming. Domain decomposition is used for solving a set of tasks that operate independently on separate partitions of data on multiprocessor architectures [17]. Meanwhile loop-level parallelism is the model that focuses on processes or threads of execution [18]. Conversely, domain decomposition is considered a coarse-grain technique, consisting in equally dividing input data structure into sub data structures to be fed into tasks. Loop-level parallelism is considered a fine-grain technique whereby it consists of parallelizing task loops where iterations can be executed independently. If those techniques are applied in OpenMP environment, the operations are performed by adding specific directives, and the OpenMP runtime will manage the allocation and distribution work to threads.

Otsu method is used as a preprocessing phase in several plant classification applications for extracting the internal characteristics (region-based) of leaf shape. The internal characteristics of leaf shape are aspect ratio, rectangularity, eccentricity, compactness, length, width, area and many more. [19] implemented parallel approach to Otsu method and run their algorithm on multi-core architecture namely Graphics Processing Unit (GPU). The results of their proposed parallel strategy achieved at least 1.6 times faster than sequential algorithm.

Canny Edge Detector method is applied as a preprocessing phase in several plant classification applications for extracting external characteristics (contour-based) of leaf shape. Example of external characteristics for leaf shapes are contour signatures and elliptic Fourier descriptors. Some of researcher also used Canny Edge Detector method to extract the leaf vein structures. Several implementations of Canny Edge Detector method can be found in the literature, using different languages on different hardware platforms [20, 21]. Canny Edge Detector method is also run on a NVIDIA GPU, using fine-grain parallelism [22]. Although this implementation offers the best performance in terms of execution time, there are no comparison works with coarse-grain parallelism.

2.1 Canny Edge Detector Method

The Canny Edge Detector method consists of five stages and one of these stages can be placed in a same "for loop", for example in stage 4 and 5 (for loop (4^{th})). The block format of Canny Edge Detector algorithm is depicted in Figure 1.

The four "for loop" are:

- Image smoothing
- Find the image derivative in the horizontal direction (G_x) and the vertical direction (G_y)
- Image intensity gradient calculation (G) and find the edge direction (θ)
- Non-maximal suppression and hysteresis thresholding calculation.

Sometimes, "for loop" is called "task", for example "for loop (1^{st})" as "task 1". From observations, the task dependencies for all tasks available in Canny Edge Detector method need to be processed in sequence manner whereby the process in task 1 must be completed before task 2 begin its work and so on. All tasks in this algorithm are purely paralleled. The purely parallel task indicates that this loop can be equally distributed over some processors.

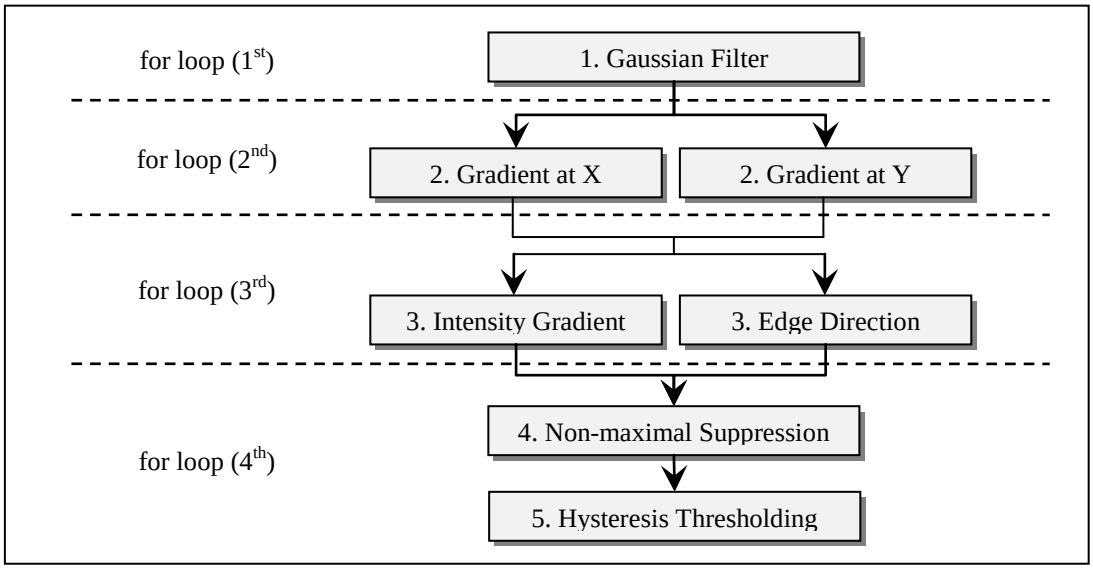

Figure 1. Block Diagram of the Canny Edge Detector Algorithm

2.2. Otsu Thresholding

The Otsu method consists of five stages (see Figure 2). The five "for loop" are:

- Calculate grayscale level image histogram
- Find weight foreground (W_f) and background (W_b), and mean foreground (μ_f) and background (μ_b) for all possible threshold value
- Between class variance (σ_B^2) calculation
- Find the maximum value of (σ_B^2) in order to get threshold value

- Convert image to binary based on the threshold level value.

The task dependencies for all tasks available in Otsu method are same with Canny Edge Detector method in which each task needs to be processed in sequence manner. All tasks for this algorithm are purely paralleled except for task 4 which is strictly sequential. This is because this task is responsible to find a maximum value in a loop. If this loop distributed over different processors, the maximum value will not be generated properly.

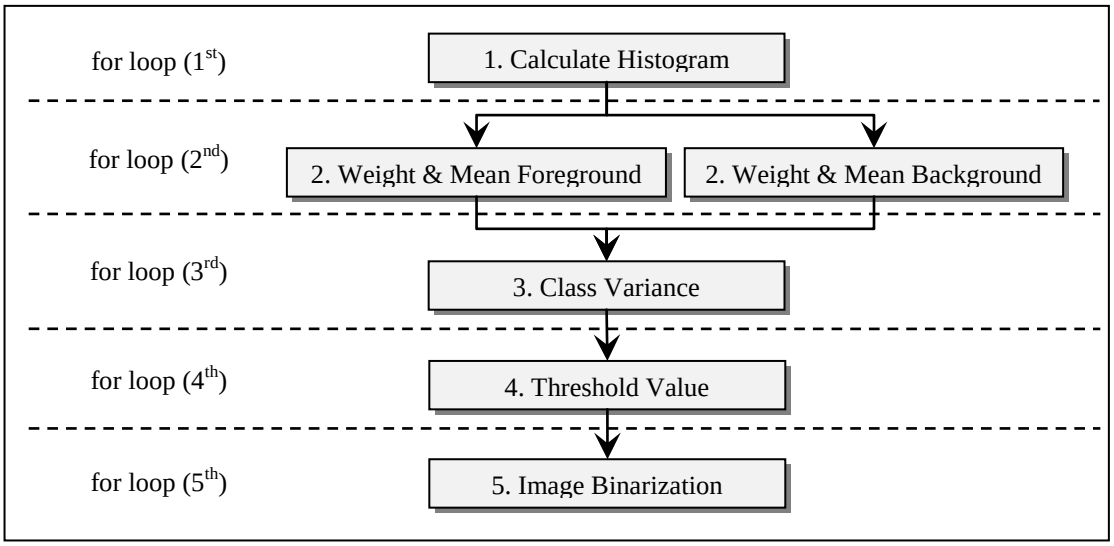

Figure 2. Block Diagram of the Otsu Thresholding Algorithm

3. Parallelization Strategy

Loop-level parallelism is considered as a fine-grain technique. It consists of parallelizing loops where iterations can be executed independently. In OpenMP, this operation is performed by adding *#pragma omp for* directives, and the OpenMP runtime will manage the allocation and distribution work to threads. Loop-level parallelism has a potentially higher reusability than domain decomposition. For example in Canny Edge Detector method, it includes a set of algorithms common to image processing programs, such as Gaussian blur and gradient calculation, whereas their parallel implementation can be easily reused in other image processing programs. However, loop-level parallelism may suffer from the overhead due to the thread's fork-join operations, and from poor data locality.

On the other hand, domain decomposition is considered as a coarse-grain technique, consisting in equally dividing an input data structure into sub data structures. In this study, the input image to the Canny Edge Detector and Otsu methods are divided into sub images with equal sizes, and the original sequential program is executed on each sub image independently and in parallel. This can be done in OpenMP using *#pragma omp sections* directives for splitting up sub image to appropriate processors. The main benefit of data decomposition is to ease parallelization, since a large section of the code base is maintained, even though some additional code is needed to split and merge the data structures. It is also necessary to assess the presence of dependencies between neighboring sub-images: in this case, the partitioning strategy has to be tuned in order to minimize the dependencies' impact on performance. In additional, domain decomposition can avoid the additional overhead caused by fork-join operations, and an increase in data locality.

3.1. Parallelization of Canny Edge Detector

The fine-grain and coarse-grain approach for parallelization of Canny Edge Detector algorithm are shown in Figure 3(a) and Figure 3(b) respectively. In fine-grain, all of tasks are distributed equally among processors. The size of loop is based on the size of input image. In coarse-grain technique, input image is split into sub images with equal size, and all tasks are executed on each sub image independently and in parallel. Based on Figure 3(b), exception implementation is on task 4 because this process need to compare eight neighboring pixels based on the result in intensity gradient and edge direction (task 3). In order to avoid data lost during execution of task 4, the task 4 must wait all the processes in previous sub-images to be finished first before begin its process.

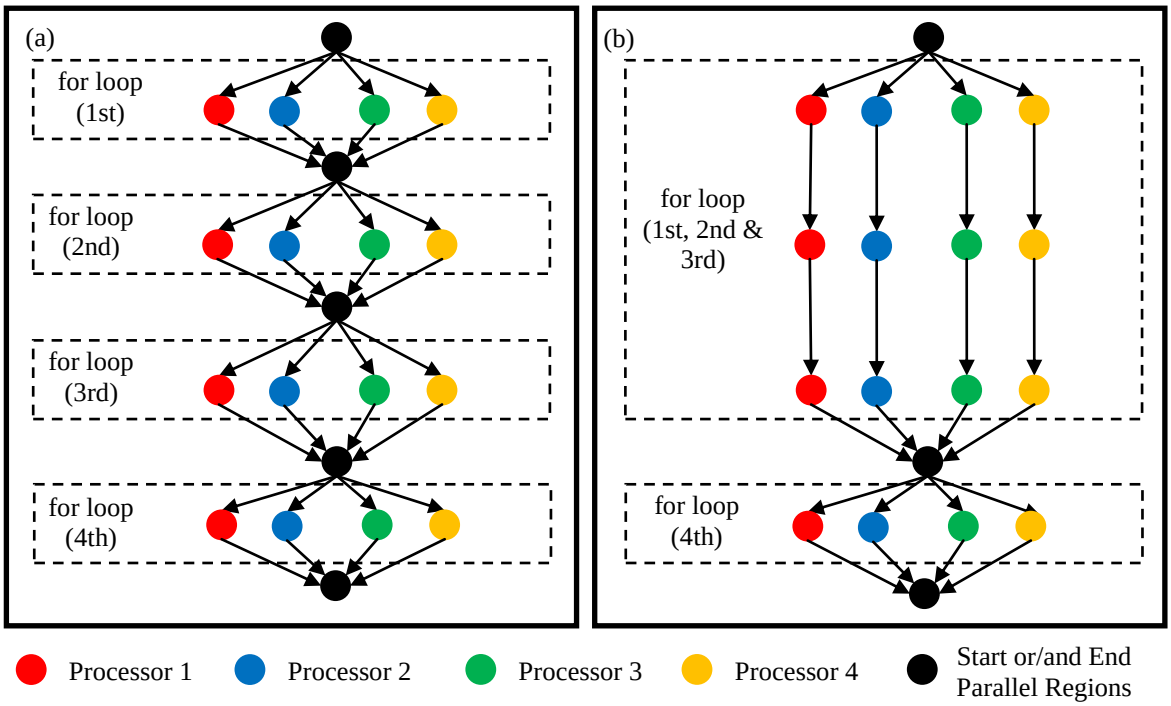

Figure 3. Parallelization Strategies for Canny Edge Detector Method: (a) Fine-Grain (b) Coarse-Grain

3.2. Parallelization of Otsu thresholding

The parallelization strategies for Otsu thresholding are shown in Figure 4(a) and Figure 4(b) respectively. In fine-grain, all tasks are distributed equally among processors except for task 4 (strictly sequential). In coarse-grain technique, input image is split into sub images with equal size. This step can be done only in task 2 and task 3. In the beginning of task 2 to the end of task 3, the loop is getting smaller into constant value of 255, grayscale image level value. To calculate histogram for each of grayscale level (task 1), the conversion of RGB to grayscale image must be completed first before calculating the histogram to avoid data lost. This also occurred in final task (task 5), whereby to convert the original image into binary image, the threshold value must be obtained in a first place. Other tasks in coarse-grain technique are the same with fine-grain technique.

As theoretical overhead and data locality comparison for this algorithm, the OpenMP has to construct six times start or/and end parallel regions in fine-grain at five different places of data locality. Whilst, five time start or/and end parallel regions are needed in coarse-grain technique with four different places of data locality sharing.

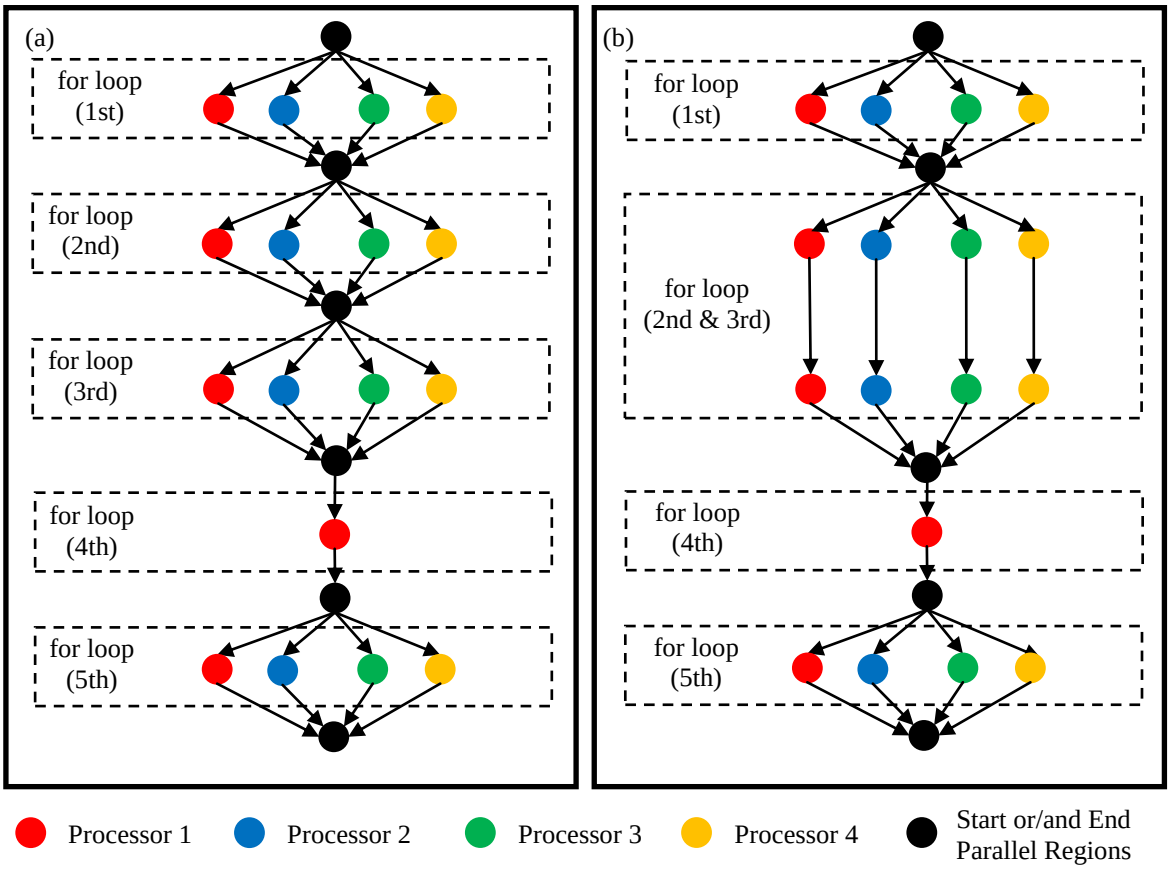

Figure 4. Parallelization Strategies for Otsu Method: (a) Fine-Grain Approach and (b) Coarse-Grain Approach

4. Experimental Results

All developed algorithms are written in C++ language and implemented using GCC 4.3.2 OpenMP compiler under a quad core platform. To evaluate the proposed strategies, the target leaf image size used is 2300x1540. The sample of image used in the experiments is shown in Figure 5. The multi-core CPU platform used is an Intel Core 2 Quad CPU Q8200, where each processor has 4 cores working at 2.33 GHz with 4.00 GB DDR3 memory.

Figure 5. The Leaf *(F. deltoidea)* Image used in the Experiments

The comparative analysis for execution time of fine-grain and coarse-grain parallelism for both Canny Edge Detector and Otsu methods is made on a quad-core CPU. Figure 6(a) and (b) shows that the coarse-grain strategy offers a lower execution time compared to the fine-grain strategy. It was happened because of the number of parallel region constructs overhead more in fine-grain compared to coarse-grain. In Canny Edge Detector method, the OpenMP has to construct five times start or/and end parallel regions in fine-

grain. Meanwhile, only three time start or/and end parallel regions are needed in coarse-grain. The same reason also occurred in Otsu in which the OpenMP has to construct six times start or/and end parallel regions in fine-grain and only five time start or/and end parallel regions are needed in coarse-grain.

The results show a better speedup (coarse-grain vs. fine-grain) in Canny Edge Detector method compared to Otsu method due to the increasing of data locality in Canny Edge detector. In Canny Edge Detector, the different in data locality for coarse-grain and fine-grain happened in task 1, task 2 and task 3. These tasks are depending on the size of image. Task 1, task 2 and task 3 in fine-grain need to access the data in a shared memory in a separate way. Different to coarse-grain, the data in a shared memory are accessed in a same time fashion.

As a result, the coarse-grain approach will contribute more execution time saving for Canny Edge Detector method. Meanwhile in Otsu method, only task 1 and task 5 are depending on the size of image. Task 2, task 3 and task 4 are based on the constant size of grayscale value which is 255. If these tasks are changed into coarse-grain, the size of data that need to be accessed in a shared memory remains the same compared to fine-grain.

The performance of two parallelization strategies is evaluated by calculating its speedup. Speedup shows how much of an improvement is practically possible in the best case. Assume that the speed of processors and its communication is constant; the speedup of k processor(s) is calculated using the following equation:

$$S(k) = \frac{T_1}{T_k}$$

where T_1 is the time required to execute an equivalent sequential program on one processor and T_k is the time required to execute the parallel version of the program on k processors. Figure 7 shows the speedup of Canny Edge Detector and Otsu methods.

Based on the performance evaluation of the two studied parallelization strategies, we can conclude that coarse-grain works well on multi-core architectures. Here, with more tasks can be executed by splitting sub images equally on each processor. Although a small speedup gap between coarse-grain and fine-grain obtained in Otsu method, it is considered as a speedup improvement. As a conclusion, coarse-grain approach is also appropriate to be implemented in Otsu method.

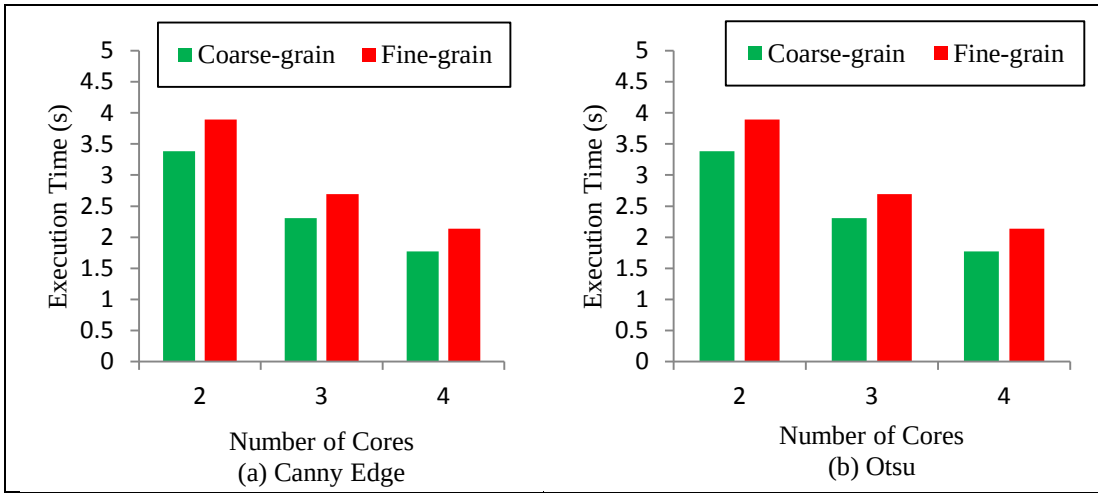

Figure 6. Execution Time (a) Canny Edge Detector Method and (b) Otsu Method

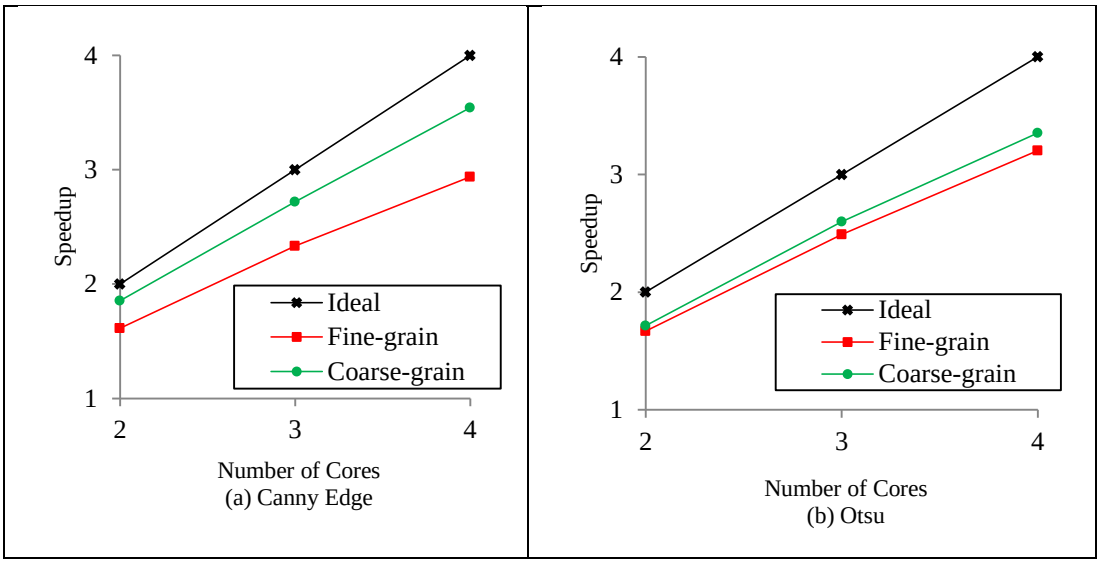

Figure 7. Overall Speedup: (a) Canny Edge Detector Method and (b) Otsu Method

5. Conclusions

The nature of tasks in Canny Edge Detection and Otsu methods are highly suitable for parallel implementation. In addition, the size of image in matrix form also makes the parallel implementation more possible. The main contribution of this paper is to describe two parallelization strategies that are generally not well studied for multi-cores architecture using fine-grain and coarse-grain parallelism. In this paper, experiments have proved that the fine-grain overhead to start a parallel region each time in OpenMP is bigger when constructing more parallel regions. Different to the coarse-grain model, the data have been split to each processor in the beginning of parallel region construct and this situation will reduce data synchronization across threads. Based on the performance evaluation of the two studied parallelization strategies, we can conclude that coarse-grain parallelism is more suitable to be implemented on Canny Edge Detector and Otsu algorithms in order to reduce some of additional time overhead and data locality that exist in OpenMP and multi-cores architecture. Even though the fine-grain model is the easiest way to be coded in OpenMP, it is worth to change the codes into coarse-grain model for more processing time reduction.

In terms of immediate future work, we intend to evaluate the performance of many-core architecture for each strategy used this experiment. The performance on different programming models is also needed to strengthen the effectiveness of these strategies. The parallel processing that guide to plant species classification is partly done and the next other future work is to focus on parallelizing features extraction process and features classification process.

Acknowledgements

The authors duly acknowledged the permission of the Faculty of Bioresource and Food Industry, Universiti Sultan Zainal Abidin (UniSZA) for allowing us to pick and capture leaves of *F. deltoidea* at the faculty nursery in the Tembila Campus.

References

[1] L. Huang, "Extending OpenMP on distributed memory Systems via global arrays", (Thesis, University of Houston, **(2006)**.
[2] R. B. Batista, A. Boukerche and A. C. M. A. de Melo, "A parallel strategy for biological sequence alignment in restricted memory space", Journal of Parallel and Distributed Computing, vol. 68, **(2008)**, pp. 548-561.
[3] M. Nordin A. Rahman, M. Yazid, MSaman, A. Ahmad and A. Osman M Tap, "Parallel Guided Dynamic Programming Approach for DNA Sequence Similarity Search", International Journal of Computer and Electrical Engineering, vol. 1, no. 4, **(2009)**, pp. 402-409.
[4] N. Otsu, "A threshold selection method from gray level histograms", IEEE Trans. on Systems, Man and Cybernetics, vol. 9, **(1979)**, pp. 62-66.
[5] J. Canny, "A computational approach to edge detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6, **1986**, 679-698.
[6] J. Adams, C. Nevison and N. C. Schaller, "Parallel computing to start the millennium", Association for Computing Machinery (ACM) SIGCSE Bulletin, vol. 32, no. 1, **2000**, 65-69.
[7] R. Yang, J. Cai, A. P. Rendell and V. Ganesh, "Cluster OpenMP and the Gaussian code: a preliminary performance analysis", 5th International Workshop on OpenMP IWOMP, Dresden, Germany, **(2009)**.
[8] M Nordin A Rahman, Utilizing MPJ Express Software in Parallel DNA Sequence Alignment. In the Proc. of the IEEE International Conference on Future Computer and Communication, **(2009)**, pp. 567-571.
[9] J. K. Park, E. J. Hwang and Y. Nam, "Utilizing venation features for efficient leaf image retrieval", Journal of Systems and Software, vol. 81, **(2008)**, pp. 71-82.
[10] S. D. Noble and R. B. Brown, "Spectral band selection and testing of edge-subtraction leaf segmentation", Canadian Biosystems Engineering, vol. 50, **(2008)**, pp. 2.1-2.8.
[11] X. Li, H.-H. Lee and K.-S. Hong, "Leaf contour extraction based on an intelligent scissor algorithm with complex background", In the International Conference on Future Computers in Education, **(2012)**, pp. 215-220.
[12] X.-F. Wang, D.-S. Huang, J.-X. Du, H. Xu and L. Heutte, "Classification of plant leaf images with complicated background", Applied Mathematics and Computation, vol. 205, **(2008)**, pp. 916-926.
[13] X. Zheng, "Leaf vein extraction based on gray-scale morphology", International Journal of Image, Graphics and Signal Processing, vol. 2, **(2010)**, pp. 25-31.
[14] T. H. Jaware, R. D. Badgujar and P. G. Patil, "Crop disease detection using image segmentation", World Journal of Science and Technology, vol. 2, no. 4, **(2012)**, pp. 190-194.
[15] A. Fakhri, A. Nasir, M. Nordin, A. Rahman and A. Rasid Mamat, "A study of image processing in agriculture application under high performance computing environment", International Journal of Computer Science and Telecommunications, vol. 3, no. 8, **(2012)**, pp. 16-24.
[16] B. Barney, "Introduction to parallel computing", form: https://computing.llnl.gov/tutorials/ parallel_comp, Retrieved (**2013)**.
[17] K. Sun, Q. Zhou, K. Mohanram and D. C. Sorensen, "Parallel domain decomposition for simulation of large-scale power grids", IEEE/ACM International Conference on Computer Aided Design, **(2007)**, pp. 54-59.
[18] C. Terboven, D. An Mey and S. Sarholz, "Openmp in multicore architectures", International Workshop on OpenMP A Practical Programming Model for the MultiCore Era, **(2008)**, 1-15.
[19] B.M. Singh, R. Sharma, A. Mittal and D. Ghosh, "Parallel implementation of Otsu's binarization approach on GPU", International Journal of Computer Applications, vol. 32, no. 2, **(2010)**, pp. 16-21.
[20] K. Ogawa, Y. Ito and K. Nakano, "Efficient Canny edge detection using a GPU", In the International Conference on Networking and Computing, **(2010)**, pp. 279-280.
[21] N. S. Chadrashekar and K. R. Nataraj, "A distributed Canny edge detector and its implementation of FPGA", International Journal of Computational Engineering Research, vol. 2, no. 7, **(2012)**, pp. 177-181.
[22] Y. M. Luo and R. Duraiswami, "Canny edge detection on NVIDIA CUDA", In the IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops, **(2008)**, pp. 1-8.

