Indonesian Journal of Electrical Engineering and Computer Science
Vol. 7, No. 3, September 2017, pp. 861 ~ 866
DOI: 10.11591/ijeecs.v7.i3.pp861-866

Performance of Parallel Computing in Bubble Sort Algorithm

Rihartanto*[1], Arief Susanto[2], Ansar Rizal[3]
[1,3]Department of Information Technology, State Polytechnic of Samarinda, Jl. Cipto Mangunkusumo, Kampus Gn. Lipan Samarinda Seberang, Samarinda, Indonesia 75112
[2]Faculty of Engineering, Muria Kudus University, Gondangmanis Kudus Jawa Tengah, Indonesia, 59327
Corresponding author, e-mail: rihart.c@gmail.com*

Abstract
The performance of an algorithm can be improved using a parallel computing programming approach. In this study, the performance of bubble sort algorithm on various computer specifications has been applied. Experimental results have shown that parallel computing programming can save significant time performance by 61%-65% compared to serial computing programming.

Keywords: *Parallel Programming; Bubble Sort; OpenMP*

1. Introduction

Currently, many types of computers have multi-processor or multicore specifications that allow computers to perform faster [1]. However, parallel processing methods that utilize parallel programming in order to improve the computer performance can also be present. In principle, parallel programming assigns tasks to cores in computer processors to speed up queuing processes [2] or improve computing performance. Meanwhile, the measurement of parallel programming performance is the speed (speed up).

Therefore, various methods can be used to work on parallel programming, including Message Passing Interface (MPI) and OpenMP (Open Multi Processing). Several research with OpenMP method has been used, among others [3] has implemented hybrid message passing interface (MPI) and OpenMP method in distribution and sharing memory of each node. The results have shown that OpenMP was good enough in constructing node structures in memory. Then, [4] have utilized OpenMP to resolve bubble sort using Intel Core i7-3610QM, (8 CPUs). The dataset was used HAMLET, PRINCE OF DENMARK. The results also showed that OpenMP was fast enough using 8 threads. Then, [5] have also implemented MPI to solve bubble and merge sort algorithms using Intel Core i7-3610QM, (8 CPUs). The results show that MPI can complete the data structure was also quite fast.

In this paper, the OpenMP method will be used to solve the bubble sort. This paper consists of four sections. Introduction explains the motivation of writing a paper. The second section, the methodology is to describe the model used. The third part, the results and discussion, and the final part is the conclusion of this study.

2. Methodology

2.1. *OpenMP*

OpenMP (Open Multi-Processing) is an application programming interface (API) that contains a number of compiler drivers and libraries in order to support parallel programming. The OpenMP (Open Multi-Processing) is an application programming interface (API) that contains a number of compiler drivers and libraries to support parallel programming. The OpenMP was developed for FORTRAN then C / C ++. The OpenMP API consists of three main components, namely compiler directive, routine library runtime and environmental variable [6,7] First, the compiler directive is used in order to show the part of the program to be done in parallel. In C / C ++ this directive is declared with ***#pragma omp*** and with include file ***omp.h***. Second, the Runtime library provided in OpenMP includes ***omp_get_num_procs()*** to find out

Received May 17, 2017; Revised July 25, 2017; Accepted August 10, 2017

the number of processors, ***omp_get_max_threads()*** to find out the maximum number of threads allowed and ***omp_get_num_thread()*** to see which threads are active. Finally, environment variables are used to control the parallel working environment, include ***omp_num_thread, omp_dynamic*** and ***omp_nested***, Figure 1.

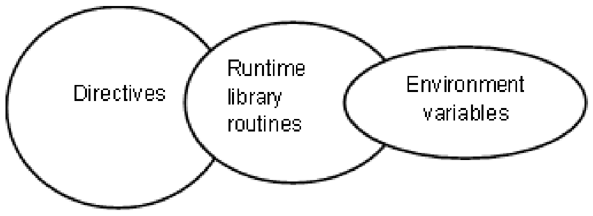

Figure 1. OpenMP Components

OpenMP also uses a fork-and-join model parallelism process[6]. In general, each program is run as a single process called parent thread. Whereas, the parallel thread is made branched from the parent thread which does its respective and synchronized tasks. Then, reassembled into the parent thread Figure 2.

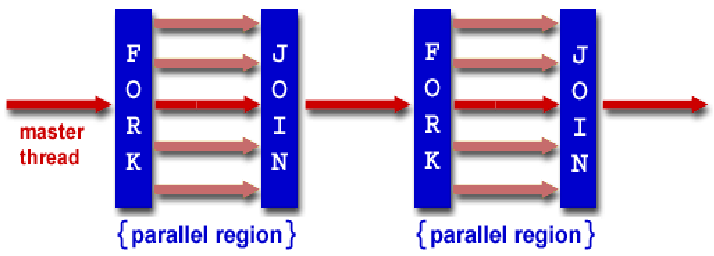

Figure 2. Fork/Join Model

2.2. Bubble Sort Algorithm

One of the computer processes is sorting data. There are many algorithms for sorting data such as bucket sort, bubble sort, insertion sort, selection sort, heapsort, merge sort, etc. Where, each algorithm has advantages and disadvantages [4, 5], [8].

In this research, bubble sort algorithm without additional optimization will be used. The bubble sort algorithm is chosen because it is easy to learn and implement. In general, bubble sort works by comparing two array elements that are side by side, and exchanging data if given conditions are met [4]. This process continues until there is no more data exchange. However, bubble sort algorithm also has performance weaknesses such as algorithm complexity *$O(n^2)$* for worst case and average case, especially on large amount of data. While, the best performance bubble sort is on the data that has been sorted *$O(1)$*.

For example, sorting the smallest to largest values. Where, 5 element array contains of "5 1 4 2 8". The bubble sort stages are as follows.

First stage:

(**5 1** 4 2 8) ⟶ (**1 5** 4 2 8), swap 5 > 1.
(1 **5 4** 2 8) ⟶ (1 **4 5** 2 8), swap 5 > 4
(1 4 **5 2** 8) ⟶ (1 4 **2 5** 8), swap 5 > 2
(1 4 2 **5 8**) ⟶ (1 4 2 **5 8**),

Second stage:

(**1 4** 2 5 8) ⟶ (**1 4** 2 5 8)
(1 **4 2** 5 8) ⟶ (1 **2 4** 5 8), swap 4 > 2
(1 2 **4 5** 8) ⟶ (1 2 **4 5** 8)
(1 2 4 **5 8**) ⟶ (1 2 4 **5 8**)

Third stage:
(**1 2** 4 5 8) → (**1 2** 4 5 8)
(1 **2 4** 5 8) → (1 **2 4** 5 8)
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

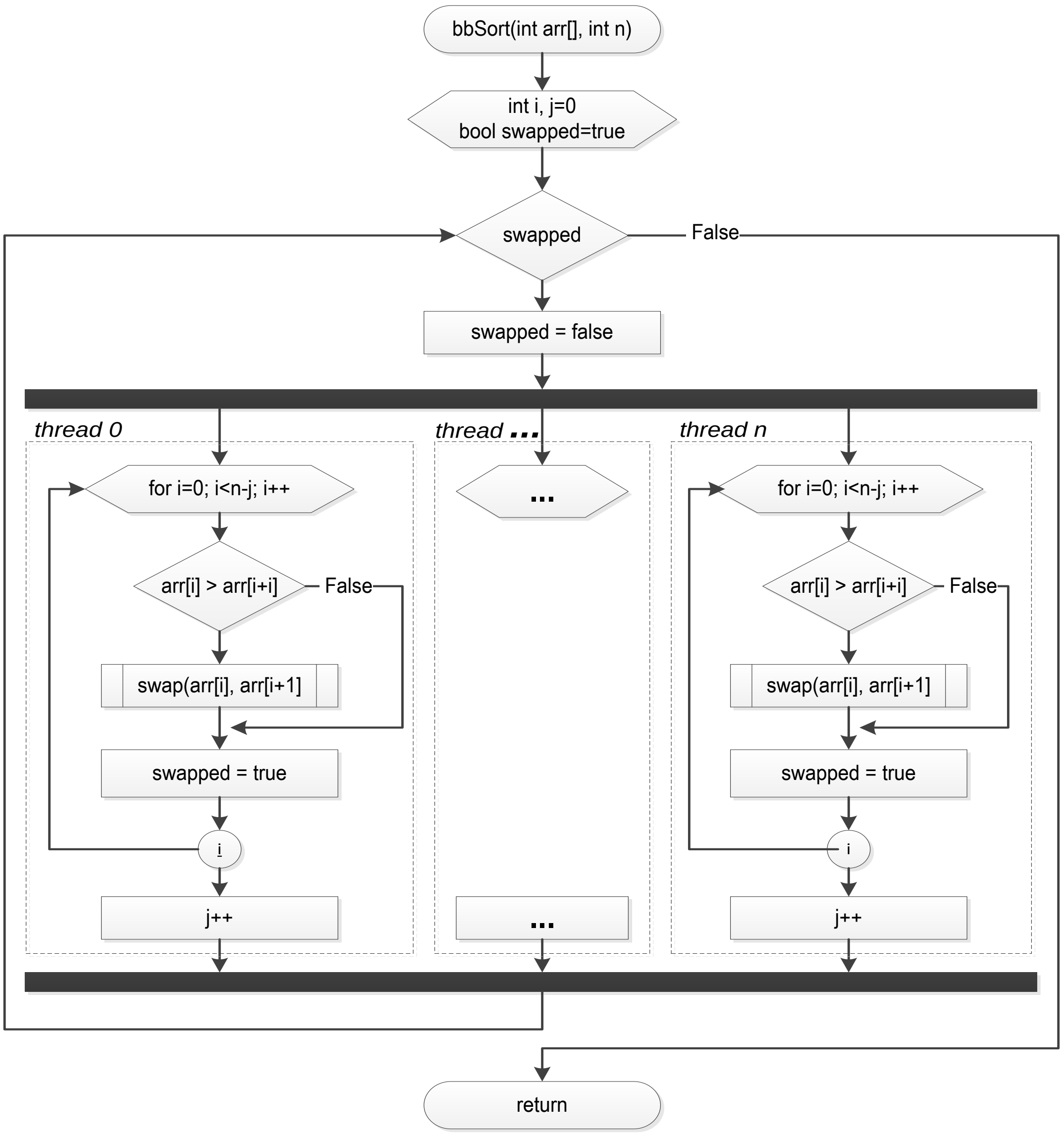

Figure 3. Flowchart of parallel processing

Flowchart of parallel processing can be seen in Figure 3, that *0… n* is total thread of processors then created it by using OpenMP. Later, shared variable of *arr[]* and other variable every thread.

2.3. Datasets

In this study, integer numbers generated with random functions and stored in text files have been used. Each dataset consists of 100, 500, 1000, 2500, 5000, 10000, 15000, 25000, 50000 and 100000. Meanwhile, increasing the data size in order to obtain significant changes by non-linier has been completed. Then, the bubble sort algorithm for serial and parallel sorting has been applied.

3. Results and Discussion

In this experiment, serial and parallel processes with the bubble sort algorithm have been applied. Meanwhile, two units of laptops with different specifications, especially low and high processors have been used. First, a laptop with Intel Celeron (R) CPU processor N2920 @ 1.86GHz Quadcore, 2GB RAM, Windows 8.1 Home SP1 64bit. Second, a laptop with Intel processor specifications (R) Core™ i7-2620M CPU @ 2.70GHz Quadcore, 8GB RAM, Windows XP Pro SP2 64bit. However, both processors have 4 cores.

The testing process by reading data from text file and storing it into an array with the appropriate size has been prepared. After that, serial and parallel sorting processed were 10 times by using bubble sort algorithm have been applied. Meanwhile, second unit as a time velocity measurement has been applied Figure 4.

In this experiment, 100 to 5000 datasets have a fast processing time of <1 sec in both serial and parallel have been generated. Meanwhile, 100000 to 500000 datasets have parallel processing time of 66% to 72% in unit 1, Figure 5, and 61% to 65% in unit 2 has been resulted, Figure 6. Furthermore, 250000 and 500000 datasets were tested only on unit 2 which have parallel processing time of 67% to 70% or 21.77 min have been resulted. This indicated that parallel processes performance with large data sizes especially in the process time has been able to be resolved quickly than the serial processes. Serial and parallel test results are shown in Table 1.

Table 1. Comparison of serial and parallel sorting time

Datasets	Unit Test 1		time saving	Unit Test 2		time saving
	serial	Parallel		serial	parallel	
100	0.0000	0.0031	-0.31%	0.0000	0.0000	0.00%
500	0.0160	0.0031	80.63%	0.0016	0.0000	99.84%
1000	0.1250	0.0222	82.24%	0.0140	0.0016	88.57%
2500	0.7180	0.0202	97.19%	0.0295	0.0140	52.54%
5000	0.2891	0.1047	63.78%	0.1063	0.0469	55.88%
10000	1.1595	0.3938	66.04%	0.4421	0.1719	61.12%
15000	2.6250	0.8515	67.56%	1.0001	0.3642	63.58%
25000	7.2735	2.2111	69.60%	2.7890	0.9827	64.77%
50000	29.1202	8.3453	71.34%	11.1734	4.0062	64.15%
100000	116.7610	32.2123	72.41%	44.8110	15.4439	65.54%
250000	-	-	-	326.8910	104.7314	67.96%
500000	-	-	-	1306.3700	389.3852	70.19%

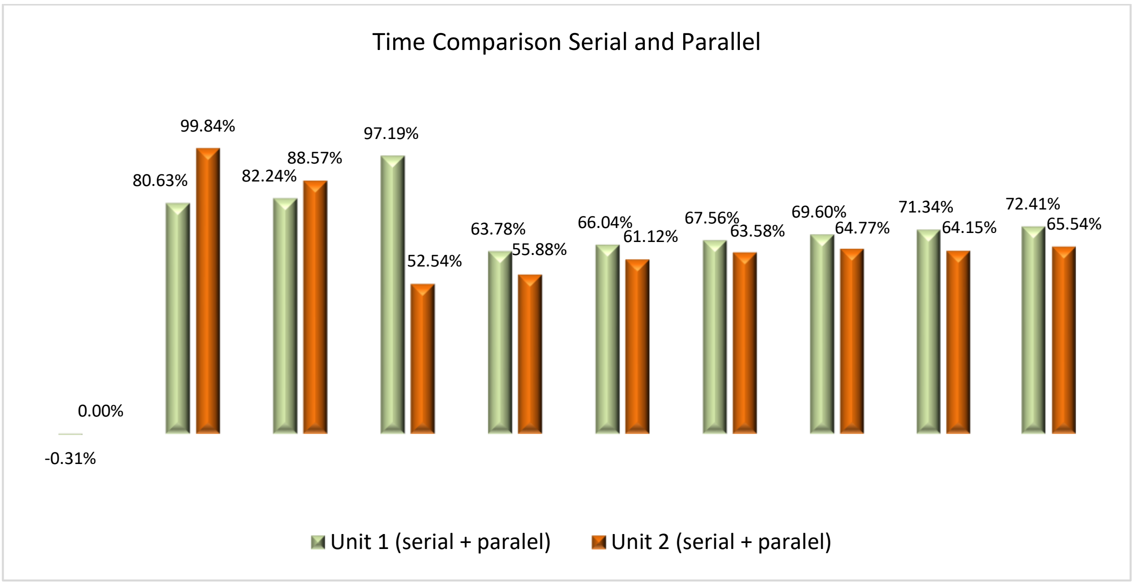

Figure 4. Time comparison of serial and parallel

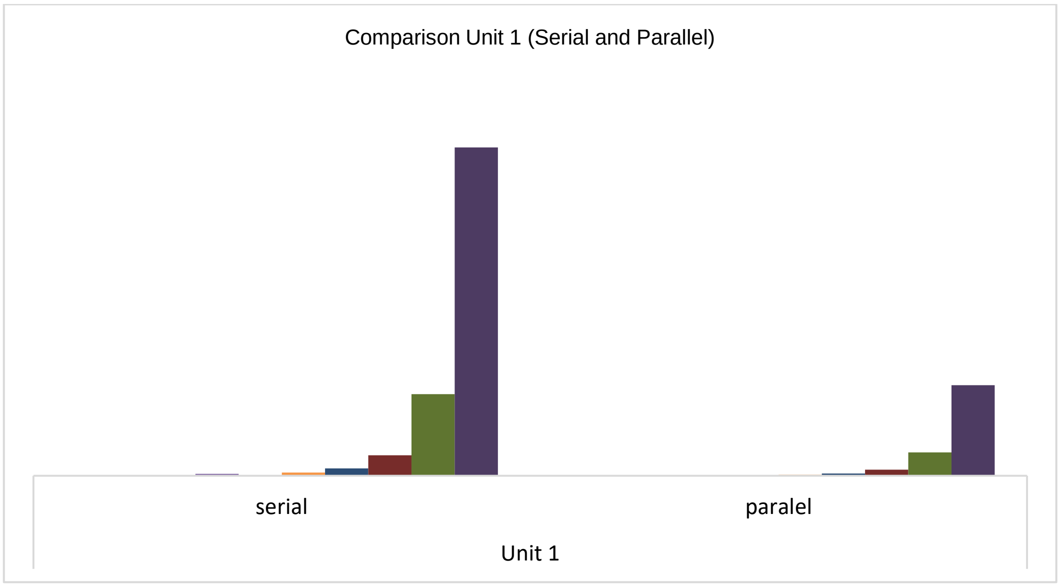

Figure 5. Time comparison of serial and parallel in unit 1

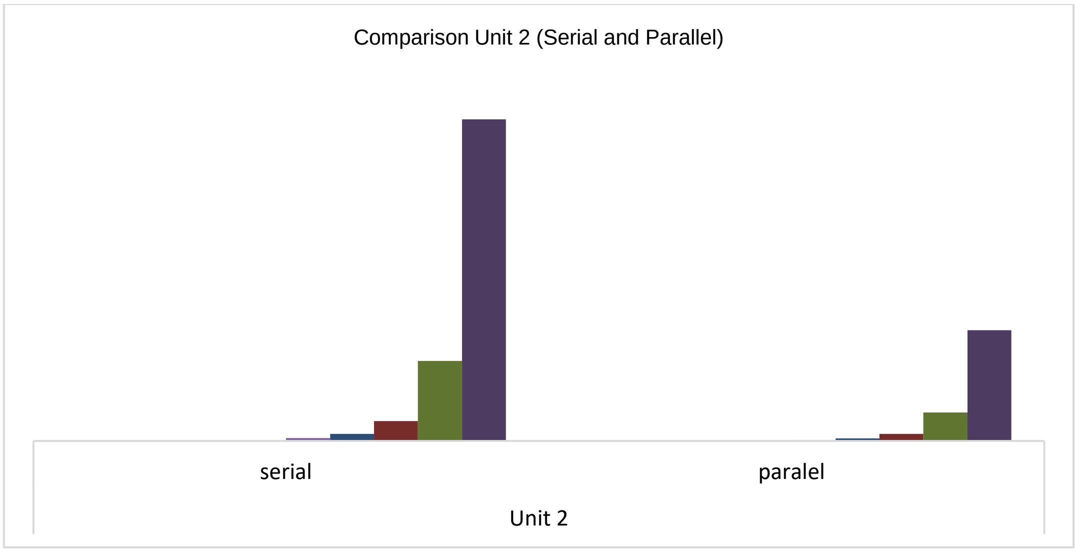

Figure 6. Time comparison of serial and parallel in unit 2

4. Conclusion

Bubble sort algorithm for serial and parallel processing is presented. Based on results, bubble sort algorithm using parallel processing have a good performance in sorting processes especially in large data than serial processes. Performance optimization of parallel processes by utilizing the whole processor core for future research.

References

[1] Xu M, Xu X, Yin M, Zheng F. *Optimized Merge Sort on Modern Commodity Multi-core CPUs*. Telkomnika. 2016;14(1):309–18.

[2] Indrawan G, Sitohang B, Akbar S. *Review of Sequential Access Method for Fingerprint Identification.* Telkomnika. 2012;10(2):335–42.

[3] Jin H, Jespersen D, Mehrotra P, Biswas R, Huang L, Chapman B. *High performance computing using MPI and OpenMP on multi-core parallel systems*. Parallel Comput. 2011;37(9):562–75.

[4] Alyasseri ZAA, Al-Attar K, Nasser M. *Parallelize Bubble Sort Algorithm Using OpenMP*. Int J Adv Res Comput Sci Softw Eng. 2014;4(1):2277–128.

[5] Alyasseri ZAA, Al-Attar K, Nasser M. *Hybrid Bubble and Merge Sort Algorithms Using Message Passing Interface*. J Comput Sci Comput Math. 2015;5(4):66–73.

[6] OpenMP. *OpenMP Application Program Interface*. OpenMP Forum, Tech Rep. 2013;(July):320.

[7] R. Chandra, R. Menon, L. Dagum, D. Kohr, D. Maydan and JM. *Parallel Programming in OpenMP*. San Francisco: ACADEMIC PRESS; 2001. 249 p.

[8] Al-dabbagh SSM, Barnouti NH. *Parallel Quicksort Algorithm using OpenMP*. Int J Comput Sci Mob Comput. 2016;5(July):372–82.