

IMPLEMENTATION OF SERIAL AND PARALLEL BUBBLE SORT ON FPGA

Dwi M J Purnomo¹, Ahmad Arinaldi¹, Dwi T Priyantini¹, Ari Wibisono¹, and Andreas Febrian²

¹Faculty of Computer Science, Universitas Indonesia, Kampus Baru UI, Depok, 16424, Indonesia

²Department of Engineering Education, Utah State University, 4160 Old Main Hill
Logan, Utah, 84322, United States of America

E-mail: dwimarhaendro@gmail.com, ari.w@cs.ui.ac.id

Abstract

Sorting is common process in computational world. Its utilization are on many fields from research to industry. There are many sorting algorithm in nowadays. One of the simplest yet powerful is bubble sort. In this study, bubble sort is implemented on FPGA. The implementation was taken on serial and parallel approach. Serial and parallel bubble sort then compared by means of its memory, execution time, and utility which comprises slices and LUTs. The experiments show that serial bubble sort required smaller memory as well as utility compared to parallel bubble sort. Meanwhile, parallel bubble sort performed faster than serial bubble sort to implement the algorithm on FPGA.

Keywords: *Sorting, bubble sort, serial bubble sort, parallel.*

Abstrak

Sorting adalah proses yang banyak dilakukan di dunia komputasi. Pemanfaatannya meliputi berbagai macam bidang, dari penelitian hingga industri. Dewasa ini terdapat banyak macam algoritma untuk *sorting*. Salah satu algoritma yang paling sederhana tapi cukup akurat adalah *bubble sort*. Pada studi ini, *bubble sort* diimplementasikan pada FPGA. Implementasi dilakukan pada pendekatan serial dan paralel. *Bubble sort* serial dan paralel dibandingkan penggunaan memori, waktu yang diperlukan untuk mengimplementasi, dan utilitas yang terdiri dari *slice* dan LUT. Eksperimen yang dilakukan menunjukkan bahwa *bubble sort* serial memerlukan lebih sedikit memori dan utilitas dibandingkan dengan *bubble sort* paralel. Sementara itu, *bubble sort* paralel memerlukan waktu lebih sedikit untuk mengimplementasikan algoritma di FPGA.

Kata Kunci: *Sorting, bubble sort, bubble sort serial, bubble sort parallel.*

1. Introduction

Sorting is a process that can be utilized in many applications. The applications vary from its origin computer science to other fields such as management, economic, etc. In computer science, sorting can be used to sort data either ascending or descending which is usually required in algorithm such as evolutionary algorithm. In management, one example is in risk management. The decision is taken based upon risk calculation. Before the decision is made the risk will be sorted to find the smallest. The number of data to be sorted are also vary from small number (e.g. less than 100) to large number of data depends on the application. There are also many variations of the algorithm in sorting process.

Several algorithms to undertake the sorting process are selection sort, merge sort, insertion sort, Heap sort, Radix sort, and bubble sort. Heap sort, Radix sort, and merge sort are powerful to sort large number of data [1]. Meanwhile the selection

sort, insertion sort and bubble sort are powerful for few data.

Selection and bubble sort are almost similar in term of the algorithm. The difference of those two algorithms lie on the array utilization in selection sort. In the selection sort, the sorting is based on the maximum value on each array, whereas in bubble sort each component is swapped one by one. Therefore it leads to the complexity different for both the algorithms. The complexity of bubble sort is $O(4n^2)$, whereas selection sort is $O(2n^2)$ [2]. Nevertheless, for small number of input, bubble sort is the simple but powerful algorithm compared to the others [3].

Bubble sort can be utilized to sort N numbers whether ascending or descending. Basically, bubble sort is undertaken in three main steps [4]. Firstly, the algorithm would step each input from the first to the last. The first position is occupied whether by smallest number or largest number, depend on the orientation of the sorting process (i.e. ascending

or descending). Secondly, the two adjacent input then would be compared. Finally, if the two adjacent inputs are in wrong order, the algorithm would swap it to the right order. The aforementioned steps would be repeated until no swap is required. Therefore, the final result would be numbers from the smallest to the largest for ascending or from the largest to the smallest for descending.

Research on bubble sort have been conducted in various application. Firstly, parallel bubble sort to utilize the concept in parallel computing [5]. Parallel computing means several calculations undertaken simultaneously [5]. Secondly study on bubble sort to assess the performance of visualization to promote the theory of understanding based on application rather than syntax [6]. Thirdly, bubble sort approach for channel routing [7]. Finally, study that compare serial and parallel computing on bubble sort with statistical bond [8]. This paper used statistical approach rather than mathematical approach. Therefore, it is represented in variant system rather than exact value like in mathematical approach [8].

Sorting is a basic process, hence it can be implemented in various platform. Field Programmable Gate Array (FPGA) is one of a desirable platform to implement sorting process. FPGA is still widely employed, especially in the industry. The advantage of using FPGA over another device is that in FPGA there will be no redundancy because the gate utilized in FPGA has not been prescribed yet [9]. Therefore, the utilization of resource is energetically effective [10]. Moreover, it is also mentioned that FPGA has considerable performance [10].

Sorting implementation on FPGA have been researched by many researchers. Srivasta et al. [10] proposed hybrid design for large scaling sorting on FPGA. Other research on high-speed parallel scheme for data sorting on FPGA. [11,12]. Parallel sorting which was conducted by Sogabe [13] and Martínez [14]. Last but not least is comparison study of many sorting algorithms covering parallel merge sort, parallel counting sort, and parallel bubble sort on FPGA [15].

The focus of this study is to implement bubble sort algorithm on FPGA. Bubble sort was chosen due to its simplicity and accuracy if the number of input data is fairly small. This study would contribute on two aspects. Firstly, the implementation of bubble sort on FPGA both serial and parallel approach. Secondly, the comprehensive comparison between serial and parallel approach on bubble sort algorithm implementation on FPGA.

The remainder of this paper is organized as follows: section two would elaborate bubble sort section three would explain the research method,

section four would shows the results and give some discussion, and the last section is conclusions.

2. Methods

Serial Bubble sort

Generally, serial bubble sort sequentially compares two number from leftmost to rightmost [3]. The order depends on whether it is ascending or descending. If it is ascending the largest number would be in the rightmost, otherwise would be in the leftmost. The schematic of serial bubble sort is delineated in Figure 1.

As shown in Figure 1, each consecutive step is undertaken by comparing two adjacent number. On the top left of the figure, two leftmost adjacent numbers are compared. In case of ascending the smaller number would be put in left side. On the contrary, in case of descending the larger number would be positioned in the left side.

The next step is to compare the next two numbers. Similar to the first step those number are compared and swapped. The same steps continue until the last number and called first stage. After all the number in the first stage has been compared, the rightmost of the number will be fixedly positioned. Therefore in the next stage, the aforementioned number would not be compared again.

The second stage then compared the 2 leftmost number again. It continues until one number before the last number similar to the first stage. In Figure 1 the fixedly last number is colored orange. Thus, it distinct which number can still be compared (blue), which cannot (orange).

The next stage until last stage is executed similarly, the only difference is that in the next stage, the rightmost number is reduced by one number as shown in Figure 1. The stage finish when all the

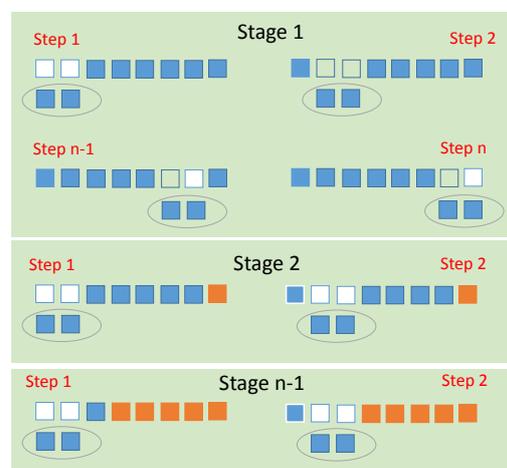


Figure. 1. Scheme of serial bubble sort.

number in each position has been compared to the number which position is right. Therefore, the position of each number will be prescribed based on the value of the number. The pseudo code for serial bubble sort is shown in algorithm 1.

As shown in the pseudo code, there are two loops in total bubble sort algorithm. The first loop is the top loop. The top loop define the stage of the algorithm. The number of stage is the number of data minus 1.

Meanwhile, the second loop is the bottom loop. It defines the step of the algorithm. The number of the step depends on the stage. It is number of data minus the undergoing stage.

Parallel Bubble sort

The idea of parallel bubble sort is to create parallel swapping. When in the serial bubble sort there is only 1 comparing process in parallel bubble sort there are $n/2$ (n is total input) comparing process. The schematic of one sorter is depicted in Figure 2.

As shown in Figure 2, there are n input data that would be sorted (i.e. $in_1, in_2, \dots, in(n)$). The first step is comparing each two adjacent numbers which is undertaken in swapper. After the first swap, the first and last number is located into the first and last output respectively (i.e. out_1 and out_n). Meanwhile, the other number after the first swap will be compared again with its adjacent number. However, the adjacent number is now changed. In the first swap process, the numbers to be compared

```

Algorithm 1: Serial Bubble Sort
1  % Initialization
2  Input_Data;
3  Number_of_Data;
4
5  for i in 1 to Number_of_Data-1 do
6    for j in 1 to Number_of_Data-i
7      do
8        compare Input_Data(i) with
9        Input_Data(i+1);
10       If Input_Data(i)>
11       Input_Data(i+1) then swap;
12     end for;
13   end for;

```

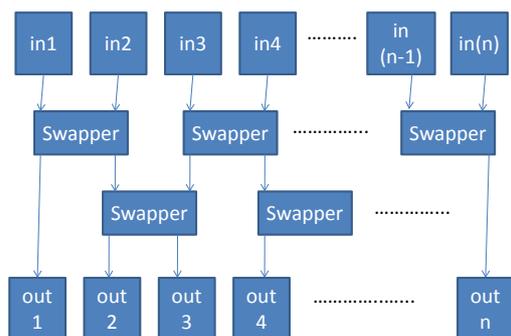


Figure 2. Scheme of steps in sorter in one stage.

are odd number followed by even number. Whereas in the second swap the numbers to be compared are even number followed by odd number.

Similar with the serial bubble sort, in parallel bubble sort there are also many stages. In each stage, contains several steps which has been previously elaborated. The schematic of the total stages in parallel bubble sort is shown in Figure 3. As shown in Figure 3, there are $n-1$ sorter. Each sorter would sort all the input (n input). Inside each sorter, there are two swappers as mentioned in the previous explanation.

Therefore, each number will compared to all the other numbers and placed in the right position. The pseudo code of parallel bubble sort is shown in Algorithm 2. As shown in the pseudo code, there are three loops in total bubble sort algorithm. The first loop is the top loop. The top loop define the stage of the algorithm. The number of stage is the number of data minus 1.

Meanwhile, the second and third loop is the bottom loop. It defines the step of the algorithm.

```

Algorithm 2: Parallel Bubble Sort
1  % Initialization
2  Input_Data;
3
4  for i in 1 to Number_of_Data-1
5    do
6      for i in 0 to
7        Number_of_Data/2-1 do
8        compare Input(2*i+1) with
9        Input(2*i+2);
10       If Input(2*i+1)>
11       Input(2*i+2) then swap;
12     end for;
13
14     for i in 1 to
15       Number_of_Data/2-1 do
16       compare Input(2*i) with
17       Input(2*i+1);
18       If Input(2*i)> Input(2*i+1)
19       then swap;
20     end for;
21   end for;

```

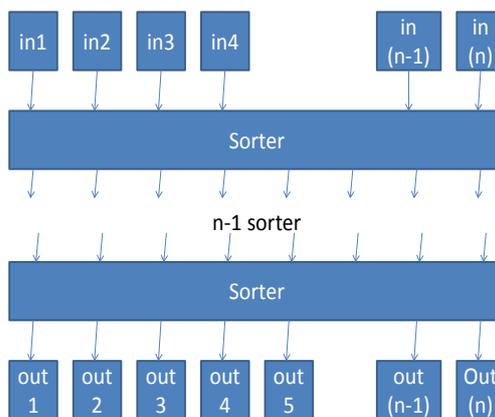


Figure 3. Scheme of stages in parallel bubble sort.

The second loop is the first swapper, whereas the third loop is the second swapper. The number of the step in the first swapper is half of number of data minus. Meanwhile, in the second swapper is half of number of data minus 1.

Serial Bubble Sort Implementation on FPGA

The implementation of serial bubble sort on FPGA contains two main steps. The steps are number representation and component implementation. This sub-section would rigorously discuss those steps.

Number Representation

In this study, the number is represented as 6 bit two's complement. The representation of the number is shown in Figure 4.

Component Implementation

There are four levels of component in serial bubble sort. Firstly, the top level is bubble sort itself which sort all the input into the right position. Secondly, swapper which swap two inputs if the condition is met. Thirdly, comparator which compare two inputs. Lastly, adder which was utilized to subtract two inputs to define the larger number in the comparator component.

Figure 5 shows one example of bubble sort component. In this example, only 8 inputs and outputs used. In the real experiment the number of input and output is varied. In this component the in-

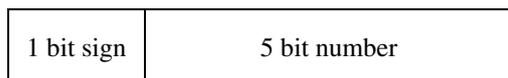


Figure 4. Number representation.

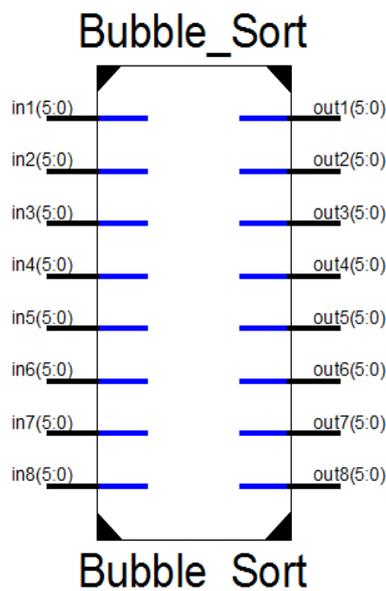


Figure 5. Bubble sort.

put would be ordered. The outputs are input that have been arranged whether ascending or descending. Inside this component, there are many swappers which is shown in Figure 6.

Figure 6 delineates the structure of the swapper which form top level bubble sort. In the first stage there are n-1 swappers i.e. to compare 2 adjacent numbers from 1 to n-1 with 1 increment. The next stage the number of swapper is reduced one each level. It means that the swapping process is reduced one step from the previous stage.

Inside swapper component there is a comparator component and multiplexer. The structure of swapper component is depicted in Figure 7. The comparator is delineated as one component, whereas the multiplexer is the combination of logic gates. The multiplexer is used to define the output after swapping process. Therefore, it has 2 outputs.

Inside comparator component there is an N bit adder. The structure of swapper component is depicted in Figure 8. The comparing process is undertaken by adding the first input with the negative of the second input. Finally, inside N bit adder there are 6 full adders which undertake the adding process.

Parallel Bubble Sort Implementation on FPGA

Similar to the serial bubble sort, the implementation of parallel bubble sort on FPGA contains two main steps. The steps are number representation and component implementation. The number representation in parallel bubble sort is similar with serial bubble sort. Therefore, this sub-section wou-

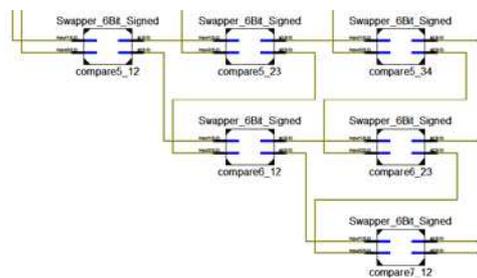


Figure 6. Structure of the Bubble Sort.

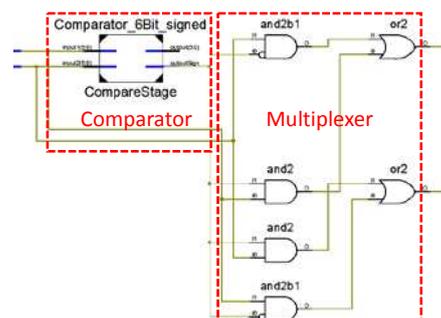


Figure 7. Structure of Swapper.

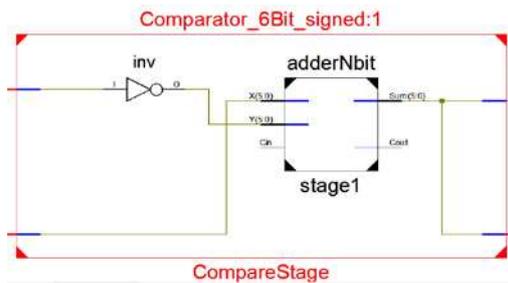


Figure 8. Structure of Comparator.

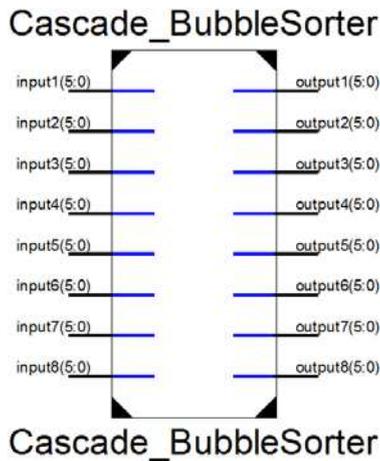


Figure 9. Top level sorter.

Id only sophisticatedly discuss the component implementation.

Component Implementation

There are five levels of component in parallel bubble sort. Firstly, top level sorter which sort all the input into the right position. Thirdly, bottom level sort which sort each two numbers. Thirdly, swapper which swap two inputs if the condition is met. Fourthly, comparator which compare two inputs. Lastly, adder which was utilized to subtract two inputs to define the larger number in the comparator component.

Figure 9 shows one example of top level sorter component. In this example, only 8 inputs and outputs used. In the real experiment the number of input and output is varied. In this component the input would be ordered. The outputs are input that have been arranged whether ascending or descending. Inside this component, there are many swappers which is shown in Figure 10.

Figure 10 delineates the structure of the bottom level sorter which form top level sorter. There are n-1 bottom level sorter to form top level sorter with n inputs and n outputs. The large view of the bottom level sorter is delineated in Figure 11.

Bottom level sorter consist of n inputs and n outputs (n = 8). Each bottom sorter represent the

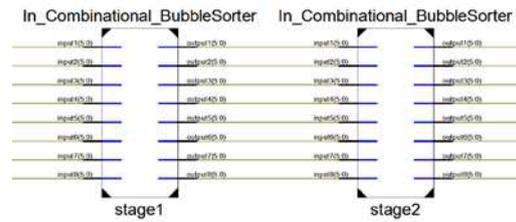


Figure 10. Structure of bottom level sorter.

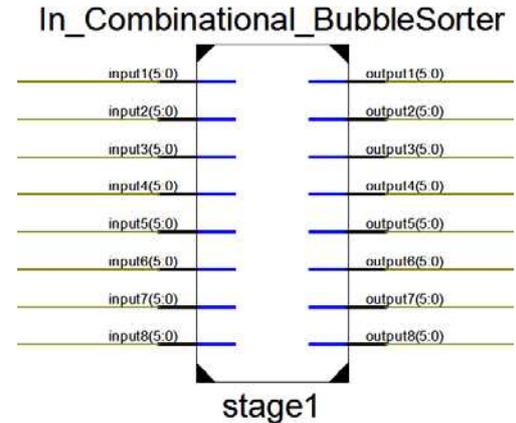


Figure 11. Bottom level sorter.

sorter in the block diagram which has previously been elucidated in Figure 3. Inside bottom level sorter there are n-1 swapper as shown in Figure 12.

The n input from top level sorter will be mapped in bottom level sorter. The inputs would enter the swapper and positioned as shown in Figure 12. If the order is wrong, then it would be swapped. Otherwise, no swapping would be undertaken. Inside swapper there is comparator in which have adder and multiplexer inside it. Swapper, comparator, and adder component in parallel bubble sort are similar with those in serial bubble sort.

Scenario

In this study, both serial and parallel bubble sort would be implemented on FPGA. The input would be varied from 4 to 40. Its increment is 2. Therefore, the input variations are 4, 6, 8, 10, 12, 14, ..., 40. The increment is defined to be two, because in parallel bubble sort it would be efficient if the number of input is even number.

The device utilized is Spartan 3A. Meanwhile the value of the inputs are varied from negative and positive. There are also similar number to test the performance of the sorter completely.

There are three main performance criteria employed in this study. The first criteria is execution time. The time required by the algorithm too implemented in the FPGA is calculated and compared.

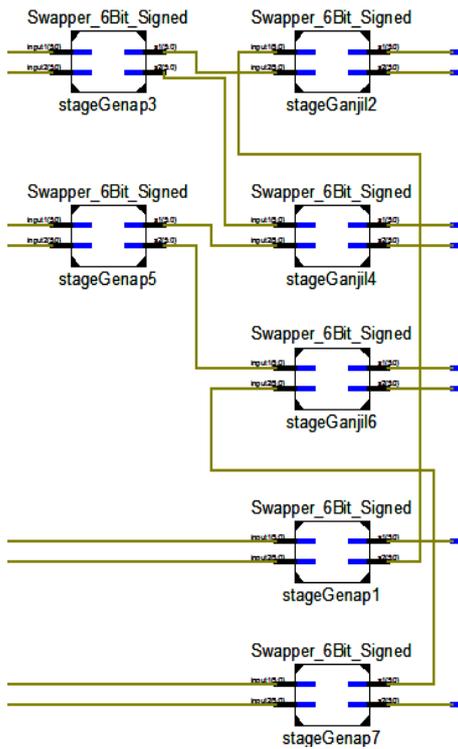


Figure 12. Structure of swapper in parallel bubble sorter.

The second criteria is memory. Memory required by the algorithm to operate is assessed and compared. Lastly, resources needed by the algorithm. Resources that compared cover slices and LUTs.

3. Results and Analysis

Serial Bubble Sort

From the experiments it is evident that serial bubble sort can be employed to sort the numbers. From 4 to 40 inputs, all variation could successfully be sorted by serial bubble sort. One example of 8 input is shown in Figure 13. The representation of the number before and after sorted are listed in Table 1.

The accuracy of serial bubble sorter is 100%. This is due to the fact that there is no number that is not compared by the others. Therefore, it covered the worst case.

To improve the system by means of its memory as well as its execution time, optimization procedure can be taken into account. By utilizing optimization process the unnecessary processes can be overlooked. Hence the time and resources can be reduced.

Parallel Bubble Sort

From the experiments it is apparent that parallel bubble sort can be utilized to sort the numbers. From 4 to 40 inputs could successfully be sorted by paral-

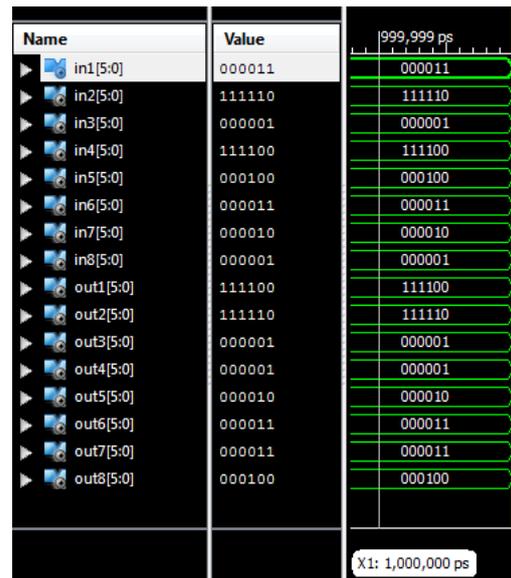


Figure 13. Result of 8 inputs serial bubble sorter.

TABLE 1
RESULT OF SERIAL BUBBLE SORT

Before Sorted		After Sorted	
Binary	Decimal	Binary	Decimal
000011	3	111100	-4
111110	-1	111110	-1
000001	1	000001	1
111100	-4	000001	1
000100	4	000010	2
000011	3	000011	3
000010	2	000011	3
000001	1	000100	4

lel bubble sort. One example of 8 input is shown in Figure 14. The representation of the number before and after sorted are listed in Table 2.

The accuracy of parallel bubble sorter is 100%. This is due to the fact that all number in the input is compared by the others. Therefore, it cope with the worst case.

To improve the system i.e. reducing memory as well as its execution time, optimization procedure can be empowered. By employing optimization process the unnecessary processes can be eradicated. Therefore, the time and resources utilization can be reduced.

Comparison of Serial and Parallel Bubble Sort

In this study, serial bubble sort and parallel bubble sort are compared. The comparison was focused on the execution time and resources. The execution time comparison is depicted in Figure 15. Meanwhile the resources by means its memory is delineated in Figure 16.

In execution time, parallel bubble sort perform better than serial bubble sort. Parallel bubble sort can sort the input faster than serial bubble sort.

TABLE 2
RESULT OF PARALLEL BUBBLE SORT

Before Sorted		After Sorted	
Binary	Decimal	Binary	Decimal
111110	-2	111100	-4
111111	-1	111101	-3
111101	-3	111110	-2
111100	-4	111111	-1
000100	4	000001	1
000011	3	000010	2
000010	2	000011	3
000001	1	000100	4

Name	Value	999,999 ps
input1[5:0]	111110	111110
input2[5:0]	111111	111111
input3[5:0]	111101	111101
input4[5:0]	111100	111100
input5[5:0]	000100	000100
input6[5:0]	000011	000011
input7[5:0]	000010	000010
input8[5:0]	000001	000001
output1[5:0]	111100	111100
output2[5:0]	111101	111101
output3[5:0]	111110	111110
output4[5:0]	111111	111111
output5[5:0]	000001	000001
output6[5:0]	000010	000010
output7[5:0]	000011	000011
output8[5:0]	000100	000100

Figure 14. Result of 8 inputs parallel bubble sorter.

TABLE 3
MEMORY AND TIME COMPARISON BETWEEN SERIAL AND PARALLEL BUBBLE SORT

Number of Input	Memory (kB)		Time(s)	
	Serial	Parallel	Serial	Parallel
4	288288	301104	49.18	17.7
8	301472	326896	50.34	26.42
12	331936	376496	76.35	25.81
14	349344	443696	91.27	32.05
16	353184	482608	106.68	40.84
18	378784	529712	129.84	53.52
20	388640	534448	136.52	70.39
22	417440	590896	191.17	94.22
24	450080	656304	201.25	135.36
26	483872	725488	266.02	206.48
28	522336	802160	365.61	276.3
30	561248	883824	333.17	370.87
32	603552	967088	415.3	422.21
34	651808	1068848	497.61	535.47
36	701536	1166000	711.03	634.98
38	754336	1242144	918.59	634.12
40	808224	1351264	1065.18	904.92

This is due to the fact that several process in parallel bubble sort can be executed parallel. Therefore, it does not have to wait the previous process done to be executed. Meanwhile, in serial bubble sort, the next process would be undertaken after the previous process have finished. Even though the more process executed at the same time would cause the

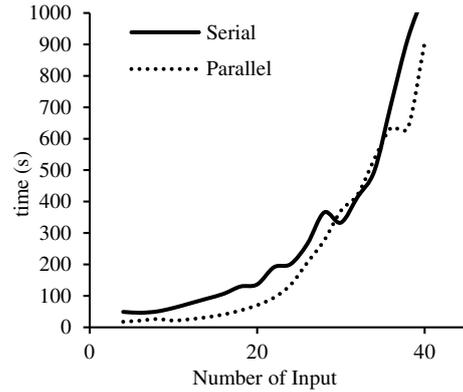


Figure 15. Execution time comparison between serial and parallel bubble sort.

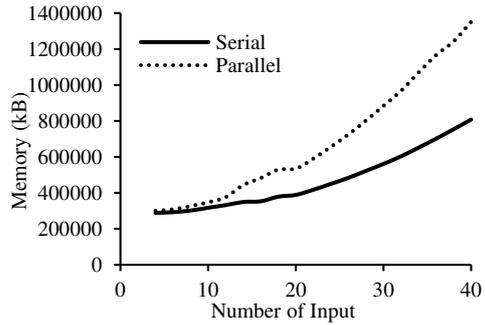


Figure 16. Memory comparison between serial and parallel bubble sort.

system take longer time to finish it, the result shows that the compensation does not affect the final result.

In terms of memory, parallel bubble sort required memory larger than serial bubble sort. Parallel bubble sort required larger memory because it undertook several processes at the same time. Therefore, each process would take larger capacity of memory.

This phenomenon is the compensation of faster process in parallel bubble sort. Serial bubble sort can be employed with limited memory capacity, whereas parallel bubble sort can operate faster. The detail comparison between serial bubble sort and parallel bubble sort are listed Table 3 for memory and time comparison.

4. Conclusion

In this paper, bubble sort has been successfully implemented on FPGA. Serial bubble sort required smaller memory as well as utilities, i.e. slices and LUTs compared to parallel bubble sort. Meanwhile, parallel bubble sort is faster than serial bubble sort to undertake the processes.

References

- [1] R. Abirami, "VHDL Implementation of Merge Sort Algorithm" *International Journal of Computer Science and Communication Engineering*, Vol. 3, No. 2, pp. 15-18, 2014.
- [2] R. Edjlal, A. Edjlal, and T. Moradi, "A Sort Implementation Comparing with Bubble Sort and Selection Sort" *In 3rd International Conference on Computer Research and Development*, pp. 380-381, 2011.
- [3] W. Min, "Analysis on Bubble Sort Algorithm Optimization" *In International Forum on Information Technology and Applications*, pp. 208-211, 2010.
- [4] H. Sutopo, "Multimedia Based Instructional Development: Bubble Sort Visualization" *In 3rd International Conference on Software Engineering and Service Science*, pp. 791-794, 2015.
- [5] R. Rashidy, S. Yousefpour, and M. Koochi, "Parallel Bubble Sort Using Stream Programming Paradigm" *In 5th International Conference on Application of Information and Communication Technologies*, pp. 1-5, 2011.
- [6] P. Bellström and C. Thorén, "Learning How to Program through Visualization: A Pilot Study on the Bubble Sort Algorithm" *In 2nd International Conference Applications of Digital Information and Web Technologies*, pp. 90-94, 2009.
- [7] S. S. Chen, C. H. Yang, and S. J. Chen, "Bubble-Sort Approach to Channel Routing" *In IEE Proceedings - Computers and Digital Techniques*, pp. 415-422, 2000.
- [8] S. K. Panigrahi, S. Chakraborty, and J. Mishra "Statistical Bound of Bubble Sort Algorithm in Serial and Parallel Computations" *In Electrical, Electronics, Signals, Communication and Optimization*, pp. 1-6, 2015.
- [9] D. M. J. Purnomo, M. R. Alhamidi, A. Wibisono, and M. I. Tawakal, "Investigation of Flip-Flop Performance on Different Type and Architecture is Shift Register with Parallel Load Applications" *Jurnal Ilmu Komputer dan Informasi*, pp. 87-95, 2015.
- [10] A. Srivastava, R. Chen, V. K. Prasanna, and C. Chelmis, "A Hybrid Design for High Performance Large-scale Sorting on FPGA" *In ReConfigurable Computing and FPGAs*, pp. 1-6, 2015.
- [11] S. Dong, X. Wang, and X. Wang, "A Novel High-Speed Parallel Scheme for Data Sorting Algorithm Based on FPGA" *In 2nd International Congress on Image and Signal Processing*, pp. 1-4, 2009.
- [12] F. A. Alquaied and M. A. AlShaya, "A Novel High-Speed Parallel Sorting Algorithm Based on FPGA" *In Saudi International Electronics, Communications and Photonics Conference*, pp. 1-4, 2011.
- [13] Y. Sogabe and T. Maruyama, "FPGA Acceleration of Short Read Mapping based on Sort and Parallel Comparison" *In 24th International Conference on Field Programmable Logic and Applications*, pp. 1-4, 2014.
- [14] J. Martínez, R. Cumplido, and C. Feregrino, "An FPGA-based Parallel Sorting Architecture for the Burrows Wheeler Transform" *In International Conference on Reconfigurable Computing and FPGAs*, pp. 7-14, 2005.
- [15] S. Bique, W. Anderson, M. Lanzagorta, and R. Rosenberg, "Sorting Using the Xilinx Virtex-4 Field Programmable Gate Arrays on the Cray XD1" *In CUG 2008 Proceedings*, pp. 1-12, 2008.