CHAPTER V
IMPLEMENTATION AND TESTING

5.1. Implementation

5.1.1 Build Virtual Parallel Workstation

Before performing the parallel computing, a parallel workstation should be setup. The preparation required to build a parallel workstation need operating system and PVM software. After the preparation are ready, the configuration of parallel computing can be started.

PVM software has an open source license, so it can be obtained for free at "*http://www.netlib.org/pvm3/*". The version of PVM used in this project is PVM 3 for Linux operating systems (*pvm3.4.6.tgz*). After successfully getting the PVM program, the next step is extracting that file to get the PVM root directory.

PVM software requires two main parameters PVM_ROOT and PVM_ARCH. The function of PVM ROOT is to give the information to operating system about the location of PVM root directory. PVM_ARCH function is to give the information about the computer architecture that being used. Both of them can be placed at file "*~/.bashrc*".

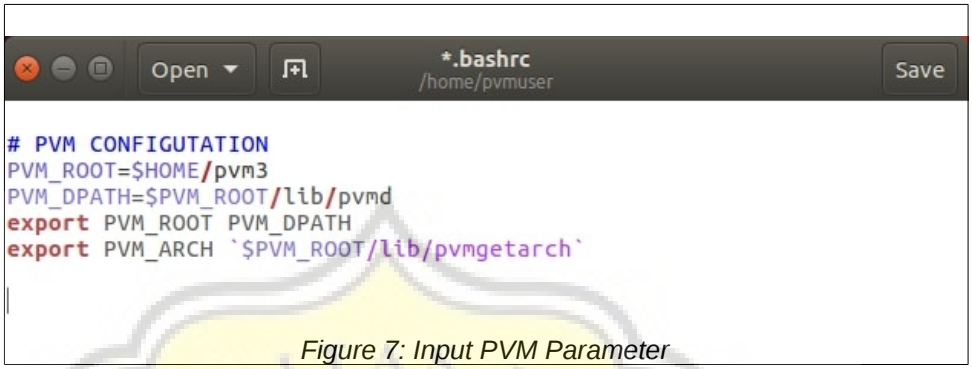

Figure 7: Input PVM Parameter

The next step is installing PVM daemon on each machines that is used. PVM daemon is a service that active at the backgroud process in operating system. PVM daemon function is to give the response of each PVM program request. The PVM daemon installation can be completed with a command "sudo apt-get install pvmd".

5.1.2 Run The Parallel Program with PVM

PVM is a master-slave programming. All commands is controlled by a master program and the tasks are computed by slave programs. In PVM, every computer used is called “a machine”. These machines can be divided into two types: master and slave machine.

To run the program with PVM, master machine is needed to activate PVM software with a command: "*~/pvm3/lib/pvm*". After the PVM software activated, master machine can register the other computers that connected to the network as a slave machine with a command: "*add cpu_hostname*". Master machine can print a list of registered machines with a command: "*conf*". After that, all registered machines will be organized by PVM to be a parallel workstation and PVM is ready to run the program.

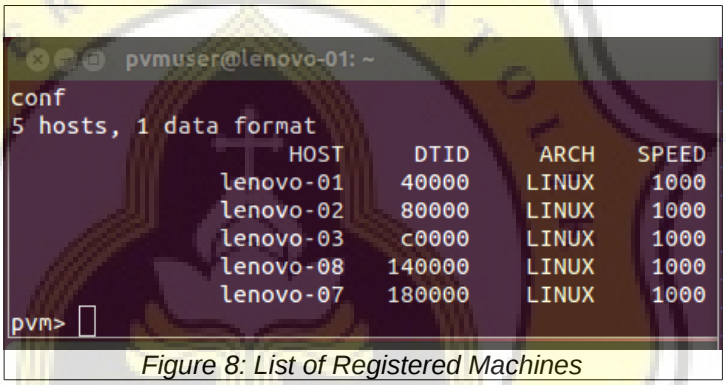

Figure 8: List of Registered Machines

Before run the program, a source code programs must be compiled with a command: "*gcc bMaster.c -o bMaser -lpvm3*". The output program must be in $PVM_ROOT/bin/$PVM_ARCH to be executed with PVM. To run the program with PVM can be done with the following command:

"*$PVM_ROOT/bin/$PVM_ARCH/program_name*"

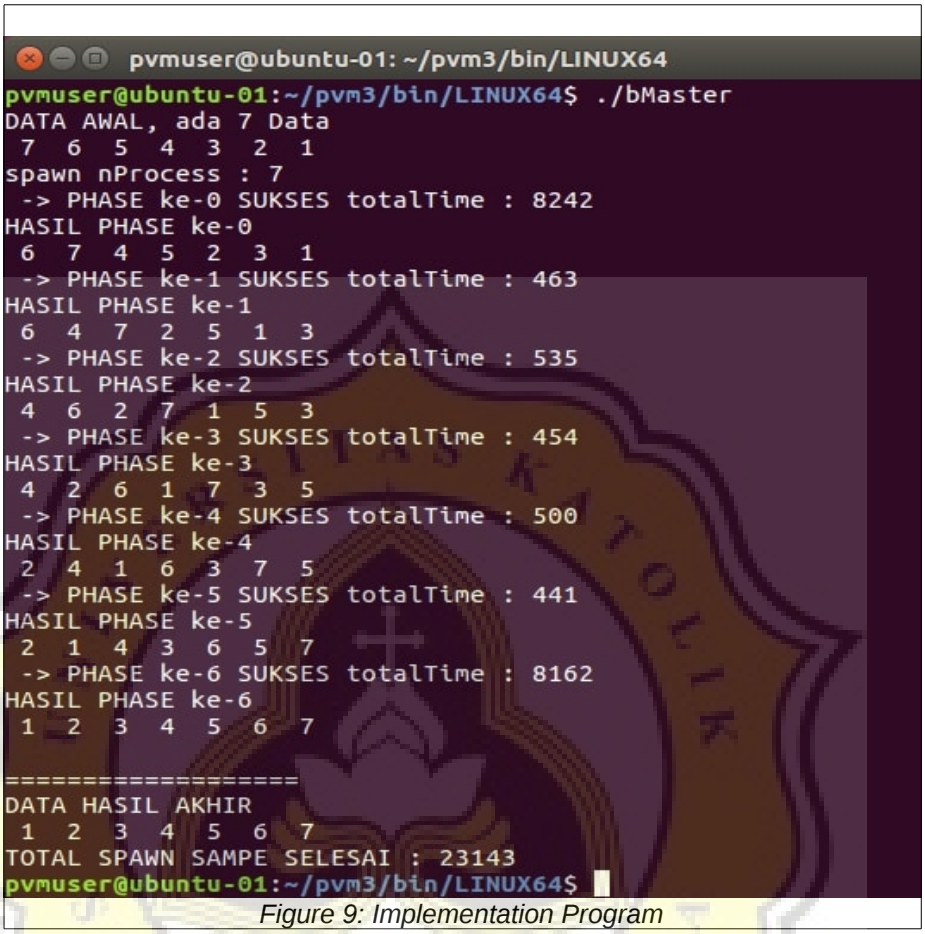

Figure 9: Implementation Program

The screenshot above is the implementation program of bubble sort odd-even transposisiton with PVM. This program have 7 unordered data to ordered in ascending. The program started with reading the unsorted data that saved in txt file. After that, the master program will do initiation and divide the main job into several processes to be performed simultaneously by slave program. Slave program will do their jobs according to the instruction signal from master program.

The next step of this program is doing the calculations for phase-0 (even phase). In phase-0, each slave with an even serial number will send its data to the next odd slave. Then, the odd slave

will be its data to received data. if its data bigger then received data, the odd slave will swap them, and returns the smaller to the even.

The calculation of the odd phase-1 is similar to the phase-0. The odd slave sends its data to the next even slave. After that, the even slave will do the comparation and returns the smaller data to odd slave.

After the job is completed, each process (slave programs) will send a feedback signal to the master program, which indicates the job of that process has been completed. Finally, the master program will collect the data from each slave, and then arranged it into an ordered result.

The calculation result of this program on each phase are:

- Phase-0 : 6 7 4 5 2 3 1
- Phase-1 : 6 4 7 2 5 1 3
- Phase-2 : 4 6 2 7 1 5 3
- Phase-3 : 4 2 6 1 7 3 5
- Phase-4 : 2 4 1 6 3 7 5
- Phase-5 : 2 1 4 3 6 5 7
- Phase-6 : 1 2 3 4 5 6 7

5.2 Testing

There are three experiments conducted on this project. These experiment are the implementation of bubble sort odd-even transposition program with several different conditions. Each experiment performed five times with the same number of data and machines.

5.2.1. Experiment 1.

The first experiment is implementing this program with four variations number of data: 10, 200, 400, and 800. This experiment uses four variations the number of CPUs/machines used: 1, 2, 3, and 4 machines. The purpose of this experiment is to obtain the time required of that program to complete. These results can be seen in the following table.

Table 1: Bubble Sort Implementation 1

Bubble Sort Implementation 1
(time in milisecond)

DATA	1 machine	2 machine	3 machine	4 machine
10	0	0	0	0
10	0	0	0	0
10	0	0	0	0
10	0	0	0	0
10	0	0	0	0
200	130	350	360	340
200	150	360	360	350
200	140	350	350	340
200	140	350	360	340
200	140	350	360	340
400	600	1440	1470	1470
400	600	1420	1440	1470
400	620	1440	1460	1490
400	600	1410	1460	1460
400	600	1420	1440	1490
800	3810	5510	5940	6560
800	3790	5490	5960	6630
800	4210	5430	5890	6630
800	4130	5430	5990	6540
800	3680	5430	6000	6590

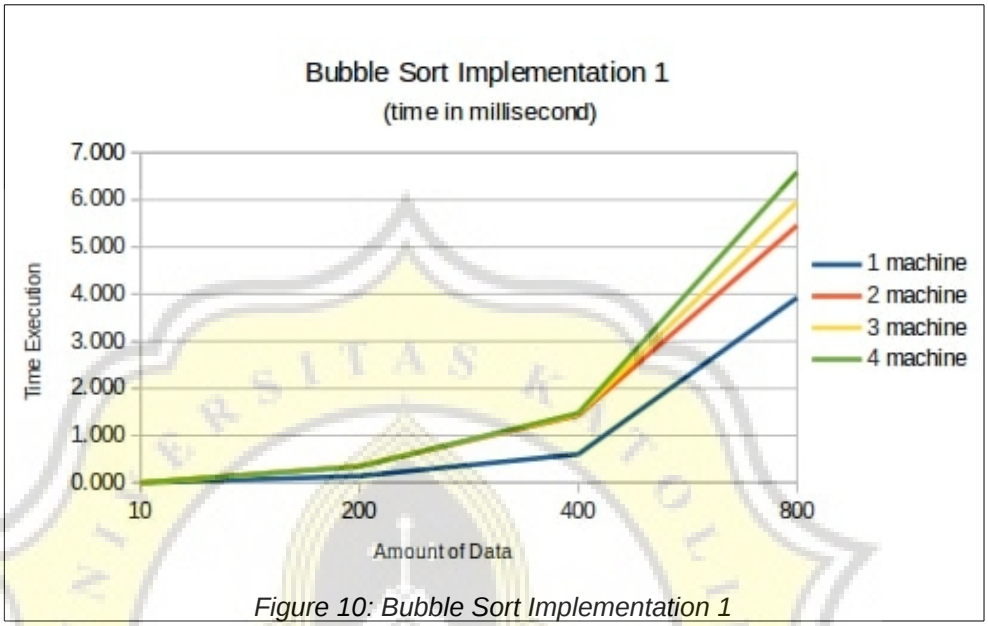

Figure 10: Bubble Sort Implementation 1

From figure 10 it can be seen that the results show that the time required to complete this program is proportionate to amount of data tested. However, the time required increases when the number of CPUs/machines used are increased.

5.2.2. Experiment 2.

The second experiment is implementing this program using ten numbers of data and eight numbers of CPUs/machines. The purpose of this experiment is to obtain more data about the time required by that program with variation of CPUs/machines used. These results can be seen in the following figure.

Table 2: Bubble Sort Implementation 2

Bubble Sort Implementation 2
(time in milisecond)

DATA	1 machine	2 machine	3 machine	4 machine	5 machine	6 machine	7 machine	8 machine
10	12.158	27.472	20.731	21.140	17.549	20.167	17.848	17.385
10	17.024	28.892	20.727	20.980	17.909	20.492	17.276	17.617
10	9.593	27.557	20.940	21.193	18.604	19.592	18.444	17.514
10	13.497	28.097	21.575	21.170	18.328	19.715	17.135	17.444
10	11.748	27.533	20.661	22.040	18.041	19.911	17.152	17.744
10	13.408	28.704	20.587	21.003	17.741	18.813	58.376	65.078
10	13.344	27.961	20.719	21.988	17.652	18.787	50.849	49.571
10	13.568	28.810	20.636	21.465	17.682	18.724	46.180	53.296
10	13.307	28.370	21.249	21.225	18.007	18.892	50.950	65.481
10	9.694	28.057	22.107	21.670	17.889	18.758	53.012	62.457

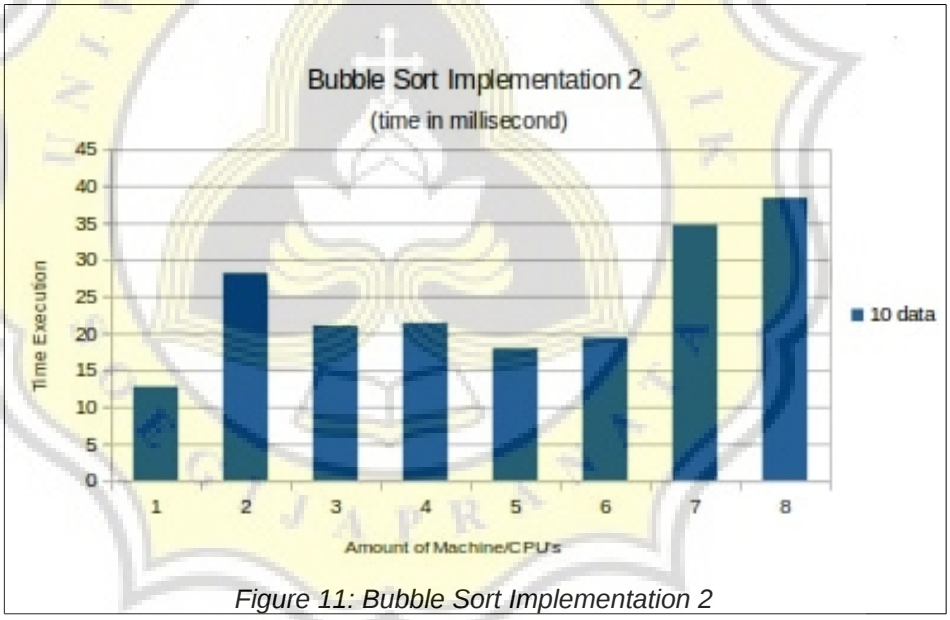

Figure 11: Bubble Sort Implementation 2

From figure 11 it can be seen that the time required to complete this program with 1, 2, 3 and 4 CPUs/machines are become uncertain. The different result happens in the experiments with 5, 6 and 7 CPUs/machines. The results are: the experiment with 5 CPUs/machines the average time is about 17.09 milliseconds, the experiment with 6 CPUs/machines the average

time is about 19.39 milliseconds, and the experiment with 7 CPUs/machines is about 38.36 milliseconds. In these cases when the number of CPUs increase, the time that required to complete the program is not always better.

5.2.3. Experiment 3.

The third experiment is implementing this program using five number of data and two different number of machines used (one and five machines). The purpose of this experiment is to obtain the time required from each process of this program by different number of CPUs/machine used. These results can be seen in figure 12 and figure 13.

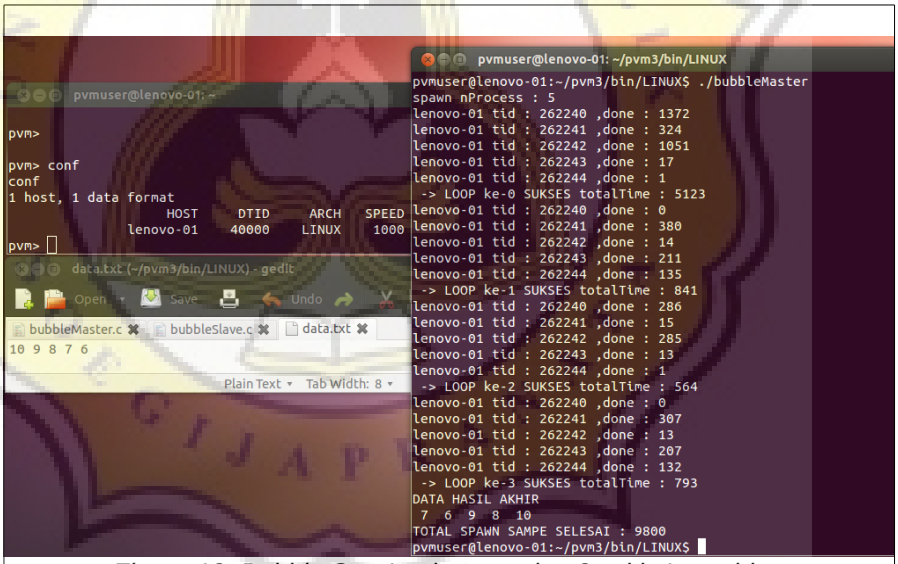

Figure 12: Bubble Sort Implementation 3, with 1 machine

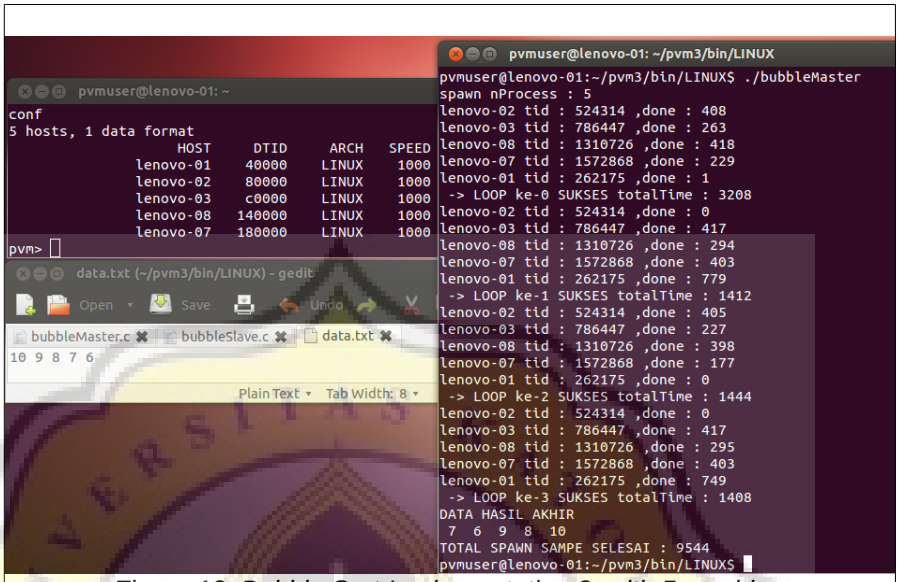

Figure 13: Bubble Sort Implementation 3, with 5 machine

From figure 12 and 13 it can be seen that there are an increasing time when the number of CPUs/machines used are increased. These happened in the master and slaves program.