VIRTUAL PARALLEL ENVIRONMENT USING PVM CASE STUDY BUBBLE SORT ALGORITHM

Iwan Pratama

Program Studi Teknik Informatika, Unika Soegijapranata Semarang

Ignatius.iwan93@gmail.com

Abstract

Parallel computing is a technique to solve a problem using many CPUs. To perform parallel computing, it requires parallel computer, which is commonly known as supercomputer or multiprocessor computer. Nowadays, a supercomputer is still very expensive. Therefore, parallel algorithm is difficult to be applied in general. To resolve this problem, virtual parallel workstation is created. Virtual parallel workstation is a network of computers controlled by a program that can regulate the allocation of computational processes to processors across the entire network. This project uses PVM (Parallel Virtual Machine) to implement the parallel workstation environment to perform Bubble Sort Algorithm in parallel approach.

Keywords: *Parallel Computing, PVM, Bubble Sort*

Pendahuluan

Komputasi paralel adalah sebuah metode perhitungan dengan memanfaatkan banyak prosesor/CPU sekaligus untuk menyelesaikan sebuah permasalahan secara simultan. Implementasi komputasi paralel memerlukan algoritma dan arsitektur komputer paralel (komputer dengan multi prosesor). Arsitektur komputer paralel umumnya dikenal sebagai *superkomputer* (multiprosesor komputer), sampai saat ini *superkomputer* masih tergolong sangat mahal sehingga komputasi paralel masih sulit untuk diterapkan secara umum.

Terkait dengan permasalahan tersebut, membangun *virtual paralel workstation* bisa menjadi solusi untuk dapat mengimplementasikan komputasi paralel dengan biaya yang lebih terjangkau. *Virtual paralel workstation* sebenarnya adalah beberapa komputer tunggal yang terhubung dengan jaringan, dikendalikan oleh perangkat lunak yang mampu mengatur seluruh alokasi proses komputasi untuk dipetakan ke setiap prosesor yang ada dalam satu jaringan. Salah satu perangkat lunak *virtual paralel workstation software* adalah PVM (*Parallel Virtual Machine*).

Penelitian tentang struktur SPO kalimat dan kata kelas telah banyak diteliti. Banyak metode dan berbagai algoritma yang digunakan untuk

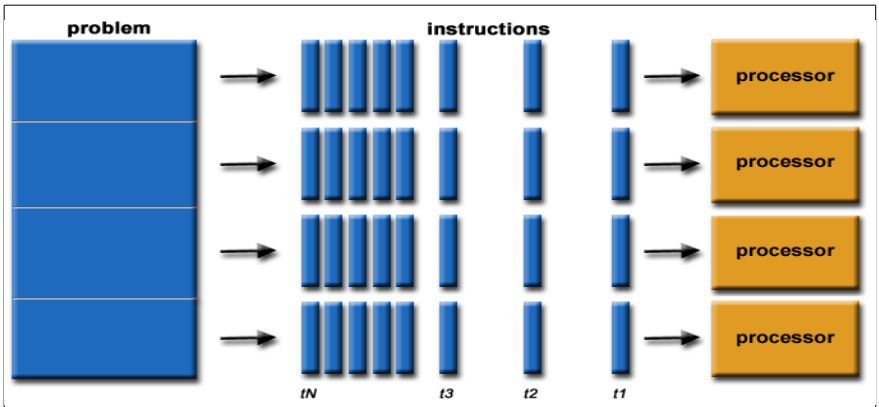
memecahkan dan merumuskannya. Metode dengan penyelesaian finite state automata yang belum pernah digunakan atau dilakukan dalam penelitian sebelumnya untuk menyelesaikan dan menentukan struktur kalimat dan kata kelas dan menggunakan metode meletakkan file sementara untuk proses pembelajaran dalam penentuan program kelas kata.

Ada tiga kelas untuk mendefinisikan kata dalam program, yaitu kata benda, kata sifat dan kata kerja. Dalam tiga kelas kata-kata berikut, program ini dapat menentukan struktur kalimat subjek, frase subjek, predikat dan frase predikat dalam kalimat. Menentukan kelas kata menggunakan algoritma finite state automata dan metode pembelajaran dari file sementara, semakin program ini akan lebih canggih, dan dapat menentukan banyak kelas kata. Dengan begitu banyak kelas kata yang dapat didefinisikan oleh program, struktur kalimat dapat diketahui.

Landasan Teori

2.1. Komputasi Paralel

Komputasi paralel adalah sebuah teknik untuk melakukan perhitungan menggunakan banyak prosesor / CPU secara simultan. Konsep kerja komputasi paralel adalah membagi tugas utama menjadi beberapa bagian yang lebih kecil untuk kemudian dapat dikerjakan oleh banyak prosesor / CPU secara bersamaan. Dari konsep tersebut maka, kemampuan komputasi paralel dapat ditingkatkan sesuai dengan jumlah prosesor / CPU yang digunakan.



Gambar 1: Konsep Komputasi paralel

2.2. PVM (*Parallel Virtual Machine*)

PVM (*Parallel Virtual Machine*) adalah sistem perangkat lunak yang memungkinkan pengguna untuk membangun sebuah paralel *workstation* menggunakan komputer yang heterogen. PVM dibagi menjadi dua bagian, bagian pertama adalah PVM *Daemon* (pvmd) dan bagian lainya adalah PVM *Library*. *Daemon* adalah sebuah program yang berjalan secara terus menerus di belakang layar dari sistem operasi. Sedangkan *Library* adalah sekumpulan sumberdaya yang membantu programmer dalam mengembangkan sebuah program. PVM *library* tersedia untuk bahasa pemprograman C dan Fortran.

2.3. Bubble Sort Algorithm

Algoritma bubble sort adalah termasuk algoritma pengurutan yang tertua dan paling sederhana. Algoritma bubble sort bekerja dengan membandingkan setiap data satu persatu dengan data disebelahnya dalam sebuah list. Kemudian melakukan pertukaran data, jika hasil perbandingannya lebih besar untuk pengurutan ascending. Proses tersebut akan diulang sebanyak *n* kali, sehingga algoritma bubble sort memiliki tingkat kompleksitas digambarkan dengan satuan *Big O* adalah sebesar O(n2).

Metodologi Penelitian

1. Mencari Referensi

Langkah pertama dari projek ini adalah mengumpulkan referensi yang terkait dengan komputasi paralel. Referensi tersebut dapat berupa artikel, buku, dan penelitian yang telah dilakukan oleh seseorang.

2. Menganalisa permasalahan dan membuat desain program

Langkah kedua adalah menganalisa dan mempelajari bagaimana algoritma paralel bekerja. Setiap algoritma mempunyai alur komputasi yang berbeda-beda, begitu pula dengan algoritma paralel yang memiliki perbedaan dengan algoritma sequential.

3. Membangun paralel *workstation*

Langkah ketiga adalah melakukan persiapan untuk membangun sebuah paralel workstation. Membangun paralel workstation dimulai dengan mempersiapkan perangkat keras komputer, sistem operasi, instalasi jaringan, dan konfigurasi perangkat lunak.

4. Melakukan implementasi dan pengujian

Langkah keempat adalah melakukan implementasi dan pengujian. Untuk melakukan implementasi komputasi paralel, sebuah program yang dirancang dengan algoritma paralel terlebih dahulu harus dibuat. Setelah itu, paralel workstation akan siap untuk diuji dengan melakukan beberapa pengujian menggunakan beberapa kondisi keadaan.

5. Mengambil kesimpulan dan membuat laporan

Langkah kelima adalah mengambil kesimpulan dan membuat laporan dari beberapa hasil pengujian dan implementasi yang sudah dilakukan.

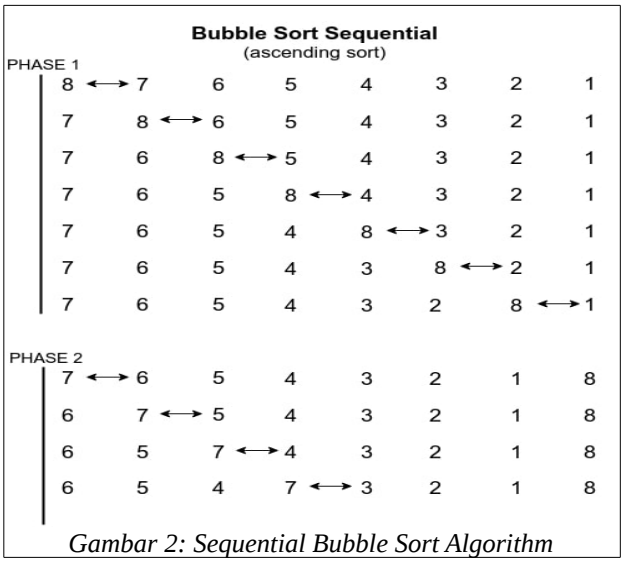
Hasil dan Pembahasan

4.1. Analisa dan Desain

Projek ini adalah implementasi paralel komputing menggunakan *software* PVM dengan studi kasus permasalahan kompleksitas yang ada pada algoritma bubblesort. PVM adalah termasuk *master-slave programing*, yang mana program dibagi menjadi 2 bagian: *master program* dan *slave program*. Pada eksekusinya *master program* akan dieksekusi pertama dan bertanggungjawab untuk melakukan inisiasi, membangkitkan *slave program*, mengatur alokasi proses, dan pengumpulkan data. Selain itu, *slave program* bertugas untuk menyelesaikan perhitungan/tugas yang diberikan oleh *master program*.

4.1.1. Bubble Sort Algorithm in Single Processor

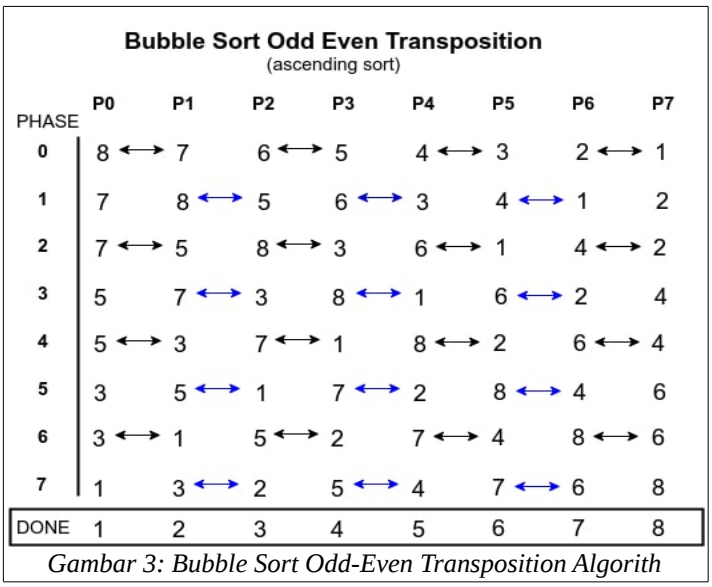
Pada *bubblesort algorithm single processor*, setiap langkah penggelembungan (pertukaran data) diselesaikan dengan multilevel iterasi, untuk kasus terburuk membuat bubblesort memiliki kompleksitas yang digambarkan dalam satuan *Big O level* adalah sebesar O(n2). Rumus perhitungan untuk ascending bubblesort dengan 8 data tak terurut adalah, sebuah prosesor tunggal membutuhkan 7 langkah untuk menyelesaikan *phase* 1, sebanyak 6 langkah untuk dapat menyelesaikan *phase* 2, membutuhkan 5 langkah untuk menyelesaikan *phase* 3 dan seterusnya hingga seluruh data terurut (diulang n-1 kali). Jadi jika langkah ini dihitung seluruhnya, prosesor tunggal membutuhkan 28 langkah (7 + 6 + 5 + 4 + 3 + 2 + 1 + 0) untuk dapat menyelesaikan pekerjaan tersebut.



Gambar 2: Sequential Bubble Sort Algorithm

4.1.2. *Bubble Sort Algorithm in Parallel Computing*

Implementasi algoritma komputer secara paralel memiliki berbagai macam teknik. Salah satunya adalah teknik *pipeline*. Pada teknik *pipeline*, masalah utama dibagi menjadi serangkaian tugas yang dapat diselesaikan oleh beberapa proses terpisah, di mana setiap proses tidak mengganggu proses lainnya.



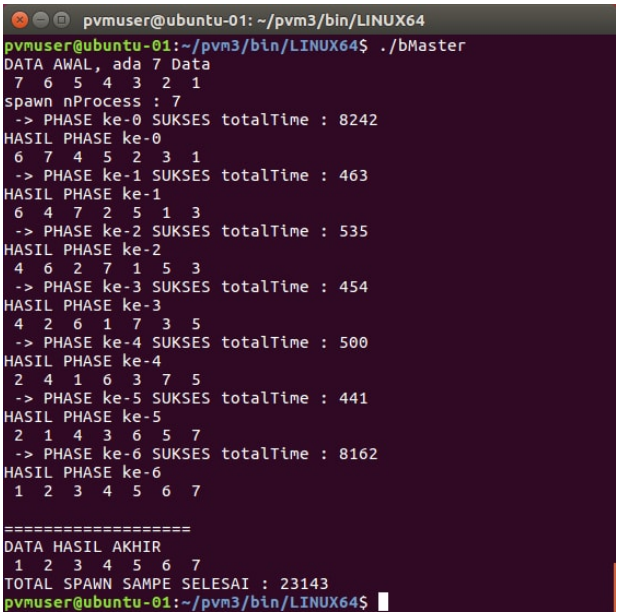
Gambar 3: Bubble Sort Odd-Even Transposition Algorith

Algoritma bubblesort dengan pendekatan *pipeline* disebut dengan "*Bubble Sort Odd-Even Transposition*", teknik ini menghasilkan pola perhitungan *phase* ganjil dan genap. Pada eksekusinya, setiap data dipetakan dalam sebuah proses (*P*) dan diselesaikan melalui beberapa *phase*. Pada *even-phase*, setiap proses dengan nomor genap akan membandingkan data miliknya dengan data

pada proses ber-nomor ganjil ditempatkan setelahnya. Kemudian jika hasil perbandingan lebih besar maka akan dilakukan pertukaran data. Begitu juga pada *odd-phase*, setiap proses ganjil akan membandingkan data miliknya dengan data pada proses genap di Hasil dari algoritma *Bubble Sort Odd-Even Transposition* dengan 8 data tak terurut membutuhkan 8 langkah atau dalam satuan *Big O level* adalah sebesar O(n). Itu berarti sangat dimungkinkan jika algoritma bubblesort dengan pendekatan paralel memiliki kompleksitas perhitungan yang lebih efisien dibandingkan bubblesort dengan *single* prosesor.

4.2. Implementation and Testing

Menjalankan program dengan PVM software memerlukan 2 parameter utama yaitu: PVM_ROOT and PVM_ARCH. Dalam implementasinya, setiap komputer yang terhubung dalam PVM paralel workstation disebut dengan istilah "*machine*". Untuk dapat mengeksekusi paralel program, terlebih dahulu master *machine* harus mengaktifkan PVM software dan mendaftarkan komputer-komputer yang terhubung untuk dapat bekerja sebagai *slave machine*.



Gambar 4: Implementation Program

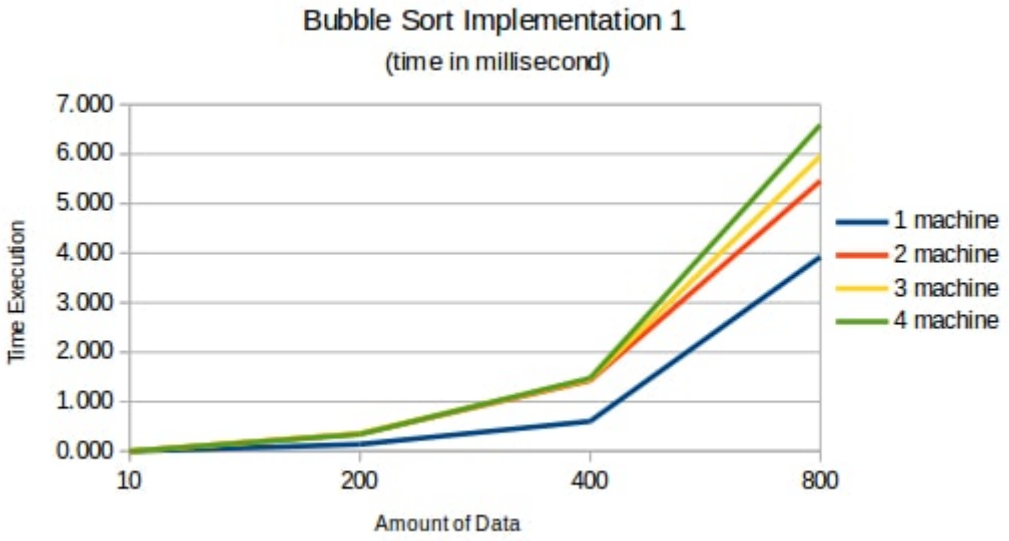
Gambar di atas merupakan implementasi dari program pengurutan bubble sort odd-even transposition menggunakan 7 bilangan tak terurut. Program ini terbagi menjadi 2 bagian, yaitu : *master* dan *slave* program. Master program dieksekusi pertama kali oleh master *machine*, yang bertugas untuk melakukan inisiasi(mendapatkan data awal), membangkitkan slave program, dan membagi tugas utama untuk dapat dikerjakan secara simultan/paralel. Setiap slave program

yang dibangkitkan memiliki tugas yang sama, dan bekerja sesuai dengan instruksi/perintah yang diberikan oleh *master* program.

Dalam project ini ada tiga percobaan yang dilakukan. Percobaan yang dilakukan adalah melakukan ujicoba terhadap *bubble sort odd-even transposition* program dengan beberapa kondisi yang berbeda. Setiap percobaan dilakukan sebanyak lima kali pada jumlah data dan *machine* yang sama.

4.2.1. Percobaan 1

Percobaan pertama adalah implementasi program dengan empat variasi jumlah data yang berbeda: 10, 20, 400, dan 800. Percobaan ini menggunakan empat variasi jumlah *machine* yang digunakan: 1, 2, 3, dan 4 *machine*. Tujuan dari percobaan ini adalah untuk mendapatkan waktu yang dibutuhkan program untuk menyelesaikan sebuah pengurutan bubble sort secara paralel. Hasil percobaan dapat dilihat dalam grafik berikut.



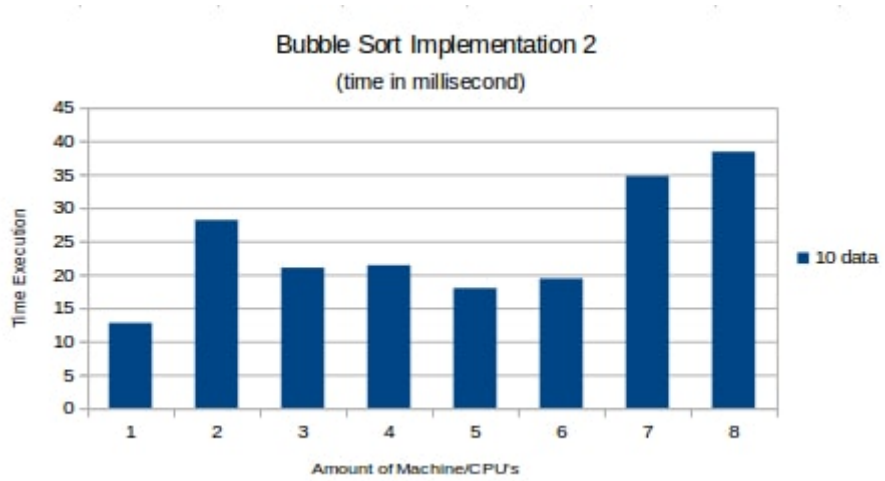
Gambar 5: Bubble Sort Implementation 1

Dari gambar 5 dapat dilihat bahwa hasil waktu yang dibutuhkan untuk menyelesaikan program ini adalah sebanding terhadap jumlah data yang diuji. Namun, waktu yang dibutuhkan meningkat ketika jumlah *machine* yang digunakan bertambah.

4.2.2. Percobaan 2

Percobaan kedua adalah implementasi program dengan 10 data menggunakan 8 *machine*. Tujuan dari percobaan ini adalah untuk mendapatkan

lebih banyak data dari waktu yang dibutuhkan program, dengan variasi jumlah *machine* yang lebih banyak. Hasil percobaan dapat dilihat dalam grafik berikut.

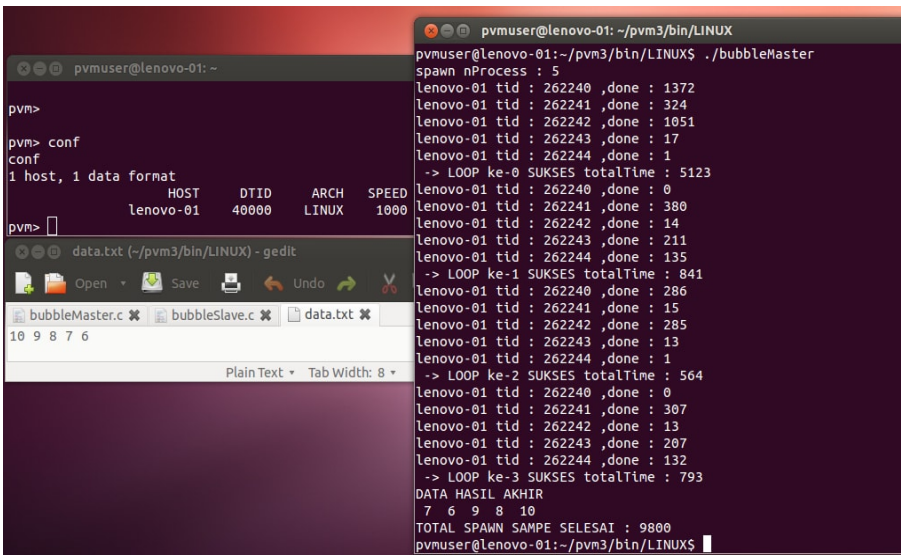


Gambar 6: Bubble Sort Implementation 2

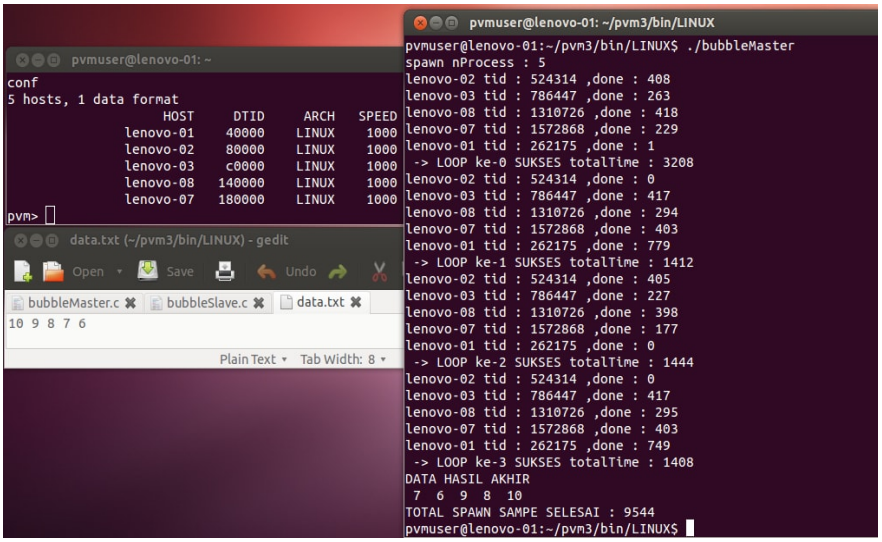
Dari gambar 6 dapat dilihat bahwa waktu yang dibutuhkan program pada percobaan menggunakan 1, 2, 3, dan 4 *machine* adalah tidak dapat ditentukan. Hasil berbeda terjadi pada percobaan dengan menggunakan 5, 6, 7, dan 8 *machine*. Hasil dari percobaan menggunakan 5 *machine* menghasilkan rata-rata waktu yang dibutuhkan adalah sebesar 17.09 miliseconds, pada percobaan menggunakan 6 *machine* menghasilkan waktu rata-rata sebesar 19.39 miliseconds, dan percobaan menggunakan 7 *machine* menghasilkan waktu rata-rata sebesar 38.36 miliseconds, dan terus meningkat pada percobaan menggunakan 8 *machine*. Dalam kasus ini ketika jumlah CPU ditingkatkan, performa dari program tidak selalu dapat lebih baik.

4.2.3. Percobaan 3

Percobaan ketiga adalah implementasi program pengurutan dengan 5 data, menggunakan 2 jumlah variasi *machine* yang berbeda (satu dan lima *machine*). Tujuan dari percobaan ini adalah untuk mendapatkan waktu yang dibutuhkan program pada setiap proses yang diselesaikan oleh setiap *machine*. Hasil percobaan dapat dilihat dalam gambar berikut.



Gambar 7: Bubble Sort Implementation 3, with 1 machine



Gambar 8: Bubble Sort Implementation 3, with 5 machine

Dari gambar 7 dan 8 dapat dilihat bahwa ada peningkatan waktu ketika jumlah *machine* yang digunakan ditingkatkan. Hal ini terjadi pada inisiasi *master* program dan pada setiap penyelesaian tugas dari *slave* program.

Kesimpulan

Berdasarkan hasil dari projek ini dapat disimpulkan bahwa PVM (*Virtual Paralel Machine*) bisa menjadi solusi untuk membangun sebuah komputer paralel *workstation*. Cara kerja PVM adalah bekerja sebagai *middleware*, mensimulasikan paralel *workstation environment* dengan menggunakan beberapa komputer tunggal yang terhubung dalam satu jaringan. Komputasi paralel memerlukan

algoritma khusus yang memungkinkan sebuah permasalahan untuk dapat dikerjakan secara paralel. Pada project ini menggunakan algoritma *bubblesort* untuk diterapkan pada pendekatan algoritma paralel. Penerapan algoritma *bubblesort* secara paralel dapat menghasilkan kompleksitas perulangan yang lebiih baik, digambarkan dengan satuan *Big O* adalah sebesar O(n).

Akan tetapi dari beberapa hasil pengujian yang dilakukan dalam projek ini, ditemukan bahwa penambahan jumlah CPUs/*machine*s yang digunakan tidak selalu memberikan peningkatan kecepatan. Dikarenakan ada banyak faktor yang mempengaruhi kecepatan dari sebuah komputasi, beberapa faktor di antaranya adalah: koneksi jaringan, kondisi dari setiap CPU/*machine* yang digunakan, pembagian proses algoritma pemprograman yang kurang efektif, dll.

Daftar Pustaka

[1] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, Vaidy Sunderam. *PVM: Parallel Virtual Machine A Users's Guide and Tutorial for Network Parallel Computing*, 1994. Diakses 28 September 2016 dari http://www.netlib.org/pvm3/book/

[2] Barry Wilkinson & Michael Allen. "*Parallel Programming: Teknik dan Aplikasi Menggunakan Jaringan Workstation & Komputer Paralel*". 2010

[3] Intan Liswandini, Budhi Irawan, and Irzaman, "Studi Komparatif Antara Paralel Virtual *Machine* (PVM) dan Message Passing Interface (MPI) Dengan Memanfaatkan Local Area Network (LAN)", Diakses 30 September 2016 dari http://elib.unikom.ac.id/download.php?id=4636

[4] Zaid Abdi Alkareem Alyasseri, Kadhim Al-Attar, Mazin Nasser and ISMAI, "Parallelize Bubble and Merge Sort Algorithms Using Message Passing Interface (MPI)". Diakses 28 September 2016 dari https://arxiv.org/pdf/1411.5283.pdf

[5] http://www.netlib.org/pvm3/