IMPLEMENTASI ALGORITMA PARTICLE SWARM OPTIMIZATION DAN KOMPUTASI PARALEL UNTUK MENYELESAIKAN PERSAMAAN ROSENBROCK DAN ALGORITMA EIGENFACE

Particle Swarm Optimization On Parallel Computing For Solving Rosenbrock Function And Eigenface Algorithm

Bayu Wibisana1[1], Lalu A. Syamsul Irfan A.2[1], I G. Pasek Suta W.3[2]

ABSTRAK

Penelitian ini akan membahas mengenai bagaimana membuat program yang mengimplementasikan algoritma Particle Swarm Optimization (PSO) dan komputasi paralel untuk menyelesaikan persamaan Rosenbrock dan algoritma Eigenface. Permasalahan yang muncul adalah semakin banyak partikel yang digunakan maka semakin lama waktu komputasinya.

Penelitian ini akan membandingkan waktu komputasi antara program yang berjalan secara standalone dengan yang telah diprogram secara paralel. Serta mengkaji pengaruh perubahan konstanta PSO (partikel, c1, c2 dan w) dalam melakukan optimasi.

Hasil penelitian ini menunjukkan penggunaan komputasi paralel dapat mempercepat waktu komputasi dari PSO. Percepatan yang didapat saat menjalankan 6144 partikel pada 16 rank sebesar 28.5 kali lebih baik daripada saat dijalankan pada program standalone. Konstanta PSO yang hasilnya relatif baik pada saat c1=2 dan c2=0.5. penggunaan bobot inertia (w) dapat meredam laju pergerakan partikel. Pengujian PSO pada penyelesaian algoritma eigenface didapatkan nilai J(W) dengan eror relatif sebesar 0.956%.

Kata Kunci: *Particle Swarm Optimization (PSO), Komputasi Paralel, Rosenbrock, Eigenface, Message Passing Interface (MPI).*

ABSTRACT

This research will discuss about how to create a programs that implements The Particle Swarm Optimization (PSO) algorithm and parallel computing to solve Rosenbrock function and Eigenface algorithm. The problem that arises is that more particles used the longer the computation time.

This research will compare the computation time between programs running in standalone with a programs has been programmed in parallel. So reviewing the effect of constant changes in PSO (particle, c1, c2, w) in the optimization.

The results showed the use of parallel computing can speed up the computation time of the PSO. Acceleration obtained when running 6144 particle on 16 ranks is 28.5 times better than when run on standalone program. PSO constant were relatively has good result when c1=2 and c2=0.5. The use of inertia weights (w) can reduce the rate of movement of the particles. Tests on PSO for completion Eigenface algorithm get value J(W) with a relative error 0.956%.

Keywords: *Particle Swarm Optimization (PSO), Parallel Computing, Rosenbrock, Eigenface, Message Passing Interface (MPI).*

PENDAHULUAN

Komputasi paralel merupakan salah satu solusi murah untuk mendapatkan kecepatan komputasi. Idenya sendiri berasal dari bagaimana menggabungkan beberapa PC sedemikian rupa sehingga PC-PC tadi seolah-olah menjadi sebuah PC dengan kemampuan komputasi yang jauh lebih baik. Komputasi paralel banyak dikembangkan untuk melakukan peramalan cuaca global, *data ming* dan sebagainya. Pemanfaatan komputasi paralel di atas tidak lepas dari pemilihan metode pemrograman paralel yang tepat serta penggunaan algoritma-algoritma yang sesuai.

Salah satu metode optimasi yang cukup populer adalah menggunakan algoritma *Particle Swarm Optimization* (PSO). Yang mana merupakan sebuah algoritma yang didasarkan pada perilaku kawanan serangga, burung atau herbivora. Sebuah partikel

[1] Jurusan Teknik Elektro Fakultas Teknik Universitas Mataram , Nusa Tenggara Barat, Indonesia
[2] Jurusan Teknik Informatika Fakultas Teknik Universitas Mataram, Nusa Tenggara Barat, Indonesia
Email: [1]bayu.wibisana@yahoo.com , [1]irfan@unram.ac.id , [2]gpsutawijaya@te.ftunram@ac.id

diasumsikan sebagai sebuah solusi. Pada umumnya, dalam penggunaan algoritma PSO, rentang penggunaan partikelnya berkisar antara 20-30 partikel saja. Kebanyakan permasalahan sebenarnya hanya membutuhkan sekitar 10 partikel untuk mendapatkan hasil yang baik. Untuk beberapa kasus yang sulit atau permasalahan khusus, penggunaan 100 hingga 200 partikel bisa dilakukan. Namun seiring dengan bertambahnya penggunaan partikel yang ada, waktu komputasi yang digunakan juga akan bertambah. Sehingga penggunaan komputasi paralel mungkin saja untuk dilakukan.

Sedangkan pada *image processing*, metode yang banyak digunakan adalah algoritma *eigenface*. *Eigenface* merupakan sekumpulan *standarize face ingredient* yang diambil dari analisis statistik dari banyak gambar wajah. Kumpulan gambar-gambar tadi diambil pada kondisi pencahayaan yang sama kemudian dinormalisasi dan diproses pada resolusi yang sama (misal m x n), kemudian citra tadi diperlakukan sebagai vektor dimensi m x n dimana komponennya diambil dari nilai piksel citra. Sehingga ketika kita punya data gambar yang banyak dengan ukuran piksel yang besar dibutuhkan waktu yang lama untuk melakukan *image processing*-nya. Sehingga komputasi paralel mungkin saja untuk dilakukan.

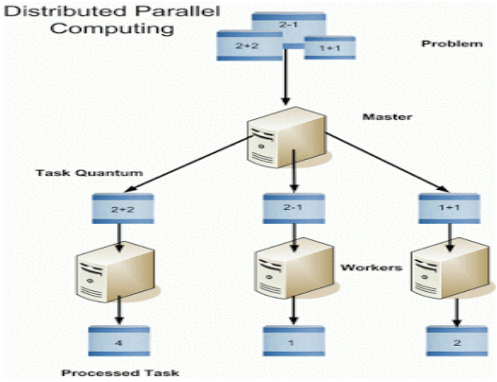
Komputasi paralel diharapkan bisa menjadi solusi untuk mempercepat waktu komputasi dari algoritma PSO dan implementasi-implementasinya. Dengan cara membagi partikel-partikel dan data lainnya ke PC-PC yang terhubung secara paralel tadi.

Komputasi Paralel. Dalam penelitiannya Annas (2013) menjelaskan jika Komputasi paralel adalah salah satu teknik melakukan komputasi secara bersamaan dengan memanfaatkan beberapa komputer independen secara bersamaan. Ini umumnya diperlukan saat kapasitas yang diperlukan sangat besar, baik karena harus mengolah data dalam jumlah besar ataupun karena tuntutan proses komputasi yang banyak.

Wilkinson (2010) menjelaskan bahwa dalam komputasi paralel *platform* komputer yang digunakan bisa berupa komputer dengan beberapa prosesor maupun beberapa komputer yang terhubung dengan cara terntentu. Selain itu diperlukan interaksi antara masing-masing bagian, baik untuk transfer data maupun sinkronisasi. Namun, seberapa jauh peningkatan kecepatan dapat dicapai bergantung pada masalah dan peran paralelisme. Satu hal yang menyebabkan kemampuan komputasi paralel menjadi tidak terbatas adalah peningkatan kecepatan suatu prosesor secara berkelanjutan yang akan meningkatkan pula kecepatan komputer paralel. Dengan demikian, akan selalu ada problem dengann tantangan besar yang tak bisa diselesaikan dalam waktu memadai menggunakan komputer saat ini.

Annas (2013) menjelaskan yang menjadi tujuan utama dari pemrograman paralel adalah untuk meningkatkan performa komputasi. Semakin banyak hal yang bisa dilakukan secara bersamaan (dalam waktu yang sama), semakin banyak pekerjaan yang bisa diselesaikan.

Mesin Paralel. Untuk melakukan aneka jenis komputasi paralel ini diperlukan infrastruktur mesin paralel yang terdiri dari banyak komputer yang dihubungkan dengan jaringan dan mampu bekerja secara paralel untuk menyelesaikan satu masalah. Untuk itu diperlukan aneka perangkat lunak pendukung yang biasa disebut sebagai *middleware* yang berperan untuk mengatur distribusi pekerjaan antar *node* dalam satu mesin paralel.



Gambar 1 Skema pemecahan masalah pada komputasi paralel.

Pemrograman Paralel adalah teknik pemrograman komputer yang memungkinkan eksekusi perintah/operasi secara simultan (komputasi paralel), baik dalam komputer dengan satu (prosesor tunggal) ataupun banyak (prosesor ganda dengan mesin paralel) CPU. Bila komputer yang digunakan secara bersamaan tersebut dilakukan oleh komputer-komputer terpisah yang terhubung dalam suatu jaringan komputer lebih sering istilah yang digunakan adalah sistem terdistribusi (*distributed computing*).

Message Passing Interface Dalam bukunya, Kurniawan (2010) menjelaskan bahwa MPI adalah bahaasa independen

untuk protolol komunikasi yang digunakan dalam pemrograman paralel pada komputer.

MPI (*Message Passing Interface*) adalah spesifikasi API (*Application Programming Interface*) yang memungkinkan terjadinya komunikasi antar komputer pada network dalam usaha untuk menyelesaikan suatu tugas.

Dalam MPI terdapat operasi untuk melakukan komunikasi secara kolektif antara lain:

1. Sinkronisasi Barrier. MPI Barrier akan melakukan *blocking* semua proses yang terjadi sampai semua proses pada group comm. Sudah mencapai MPI Barrier
2. Broadcast. operasi *broadcast* itu terjadi ketika suatu proses dalam spesifik *group* mengirim ke seluruh proses pada spesifik *group*.
3. Gather dan Scatter. Gather adalah proses pengambilan data dari semua proses pada spesifik *communicator*. Scatter adalah proses pengiriman data dari satu proses ke semua proses pada spesifik *communicator*.

Particle Swarm Optimization. Dalam tulisannya, Hassan (2005) menyatakan jika algoritma particle swarm optimization pertama kali diciptakan Kennedy dan Eberhart saat mencoba untuk mensimulasikan koreografi dan gerakan kawanan burung sebagai bagian dari studi sosial kognitif dan kecerdasan kelompok pada populasi biologi. Dalam PSO, satu set solusi yang diinisialisasi secara acak merambat diruang desain terhadap solusi optimal atas sejumlah iterasi didasarkan pada informasi yang dimiliki oleh kawanan tersebut. PSO terinspirasi oleh kemampuan kawanan burung, ikan dan hewan untuk beradaptasi pada lingkungan, menemukan sumber makanan dan menghindari predator dengan pendekatan "berbagi informasi".

Persamaan matematis untuk pembaharuan kecepatan (*velocity*) adalah:

$v_{i\,t+1} = wv_{it} + c_1 r_1(pbest - x_t) + c_2 r_2(gbest - x_t)$ ..... (1)

Sedangkan untuk pembaharuan posisi adalah:

$x_{t+1} = x_t + v_{it}$ ..... (2)

Keterangan:

$v_{i\,t+1}$: kecepatan saat ini pada waktu t.
w : bobot inersia, konstan.
c_1 : bobot kognitif (personal atau lokal), konstan
r_1 : variabel *random* dengan range [0,1]
$pbest$: posisi terbaik yang ditemukan oleh partikel.
x_t : posisi saat ini
c_2 : bobot global
r_2 : variabel random dengan range [0,1]
$gbest$: posisi terbaik yang ditemukan oleh partikel dalam kawanan sejauh ini.
x_{t+1} : posisi terbaru.

BCCD Dalam penelitiannya, Annas (2013) menjelaskan BCCD (*Bootable Cluster CD*) merupakan suatu *live* cd yang berisi *tool-tool clustering*. Berbasis debian dan berlisensi GNU dibentuk oleh Paul Gray dan tim nya dari Universitas Nothern Iowa dalam grup Earlham *College Cluster Computing Group*. *Cluster live* CD ini diciptakan untuk memfasilitasi kebutuhan konstruksi komputasi paralel dan keperluannya. Seringkali penggunaan *resource* untuk komputasi paralel yang rumit dan mengorbankan sistem operasi yang telah terpakai di dalam *node*. Oleh sebab itu kurangnya sumber daya dari server komputasi paralel dapat diakomodasi dengan penggunaan *live* CD *cluster* tanpa mengubah sistem yang ada.

Fungsi Rosenbrock pertama kali diperkenalkan oleh Howard H. Rosenbrock pada tahun 1960. Persamaan ini biasa digunakan sebagai problem uji performa dari algoritma optimasi. Persamaan Rosenbrock ini kadang disebut *banana function* (fungsi pisang) atau rosenbrock *valley* karena bentuk dari garis konturnya.

Adapun persamaan dari fungsi ini yaitu:

$f(x,y) = (a-x)^2 + b(y-x^2)^2$ ....(3)

Mempunyai global optimum pada $(x,y) = (a,a^2)$, dimana $(x,y) = 0$. Atau bisa pula diartikan global optimum dari persamaan ini terletak pada titik (1,1). Biasanya nilai dari $a = 1$ dan $b = 100$, url_1(2014)

Eigenface. Menurut Minartiningtyas (2013) dalam artikel yang dibuatnya, kata *eigenface* sebenarnya berasal dari bahasa Jerman "*eigenwert*" dimana "*eigen*" artinya karakteristik dan "*wert*" artinya nilai. *Eigenface* adalah salah satu algoritma pengenalan pola wajah yang berdasarkan pada *Principle Component Analysis* (PCA) yang dikembangkan MIT. *Eigenface* merupakan kumpulan *eigenvector* yang digunakan untuk masalah *computer vision* pada pengenalan wajah manusia.

Tahapan perhitungan dari *eigenface*:

1. Menyiapkan data dengan membuat suatu himpunan S yang terdiri dari seluruh *training image* $(\Gamma_1, \Gamma_2, \dots, \Gamma_M)$.

   $S = \{\Gamma_1, \Gamma_{2,\dots,\Gamma_M}\}$ .....(4)
2. Ambil nilai tengah atau *mean* (Ψ)

   $\Psi = \frac{1}{M}\sum_{n=1}^{M} \Gamma_n$ .....(5)
3. Cari selisih (Φ) antara *training image* (Γi) dengan nilai tengah (Ψ)

$\Phi_i = \Gamma_i - \Psi$.....(6)

4. Menghitung nilai matriks kovarian (C)

$C = \frac{1}{M}\sum_{n=1}^{M} \Phi n \Phi_n^T$ ..... (7)

$L = A^T A$ ..... (8)

$L = \Phi_m^T \Phi_n$ .....(9)

5. Menghitung *eigenvalue* (λ) dan *eigenvector* (v) dari matriks kovarian (C)

$C\ x\ v_i = \lambda_i v_i$ .....(10)

6. Setelah *eigenvector* (v) diperoleh, maka *eigenface* (μ) dapat dicari dengan:

$\mu_i = \sum_{k=1}^{M} v_{ik}\, \Phi_k$ .....(11)

Seperti yang dijelaskan sebelumnya jika algoritma eigenface berdasarkan pada algoritma *Principle Component Analysis* (PCA). Dan menurut penelitian yang dilakukan Wijaya (2013) mengenai klasik *Principle Component Analysis* bekerja didasarkan pada kriteria seperti Persamaan di bawah ini:

$$J(W) = {}^{\arg}{}^{max}_{w}(W^T C_g W) \quad \text{.....(12)}$$

Keterangan:

- W = Matriks.
- W^T = Transpose dari matriks W
- C_g = Kovarian
- J(W) = optimum dari matriks $W^T C_g W$

Pada penelitian ini nantinya persamaan diatas ini yang akan digunakan sebagai persamaan evaluasi pada kasus 2. Nilai partikel-partikel yang dibangkitkan dan diperbaharui disetiap iterasinya akan bertindak sebagai W dalam persamaan ini. Sedangkan kovarian didapat dari perhitungan dari sekumpulan data-data yang dibangkitkan nantinya.

METODE PENELITIAN

Dalam penelitian ini menggunakan alat dan bahan baik dari perangkat penelitiannya maupun perangkat pengujiannya dengan penjabaran sebagai berikut:

1. Sebuah Notebook komputer Samsung NP27564V-K01 sebagai perangkat pembantu penelitian dalam pembuatan laporan dan pembuatan *script* program.

Tabel 1 Spesifikasi notebook penelitian

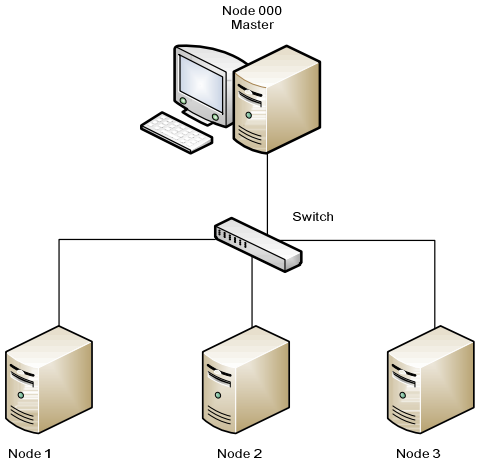
Perangkat	Keterangan
Processor	AMD E2-2000 APU 1.75 GHz
Memori	4.00 GB
Hardisk	500 GB
Kartu Ethernet	Realtek PCIe FE Family Controller
Sistem Operasi	Linux Mint 15 (Olivia) dan Windows 7 Home Premium 32-bit
Software Pendukung	- GCC GNU C Compiler - Code::Blocks IDE - OpenMPI *library* - simulink Matlab r2014a

2. 1 buah perangkat Switch, dalam penelitian ini digunakan Switch D-LINK DES-1024R+.
3. Kabel-kabel UTP straight through, CAT 5 sebanyak 4 buah dengan tipe *male to male*.
4. komputer sebagai *node* pengujian (1 sebagai *node master* dan 3 sebagai *node slave*), dalam hal ini dipergunakan 4 komputer yang bersifat homogen pada Laboratorium Jaringan dan Komputasi Fakultas Teknik Universitas Mataram.
5. 4 buah Live CD BCCD v3-3898 Perangkat Lunak
6. Media storage sebagai penyimpanan pada mesin kluster, dalam penelitian ini menggunakan flash disk.

Tabel 2 Spesifikasi komputer pengujian (node master dan slave)

Perangkat	Keterangan
Processor	Intel(R) core (TM) i3-3220 CPU@3.30GHz (4 CPUs), ~3.3GHz
Memori	2.00 GB
Hardisk	160 GB (Optional)
Kartu Ethernet	Realtek PCIe FE Family Controller RTLE8023
Drive Optik	LG HL-DT-ST DVDRAM GH22NS40
Sistem Operasi	BCCD Live CD dan Windows 7

Setelah perangkat pengujian siap, maka komputer pengujian yang berjumlah 4 tersebut akan dirangkai sedemikian sehingga menyerupai gambar di bawah ini



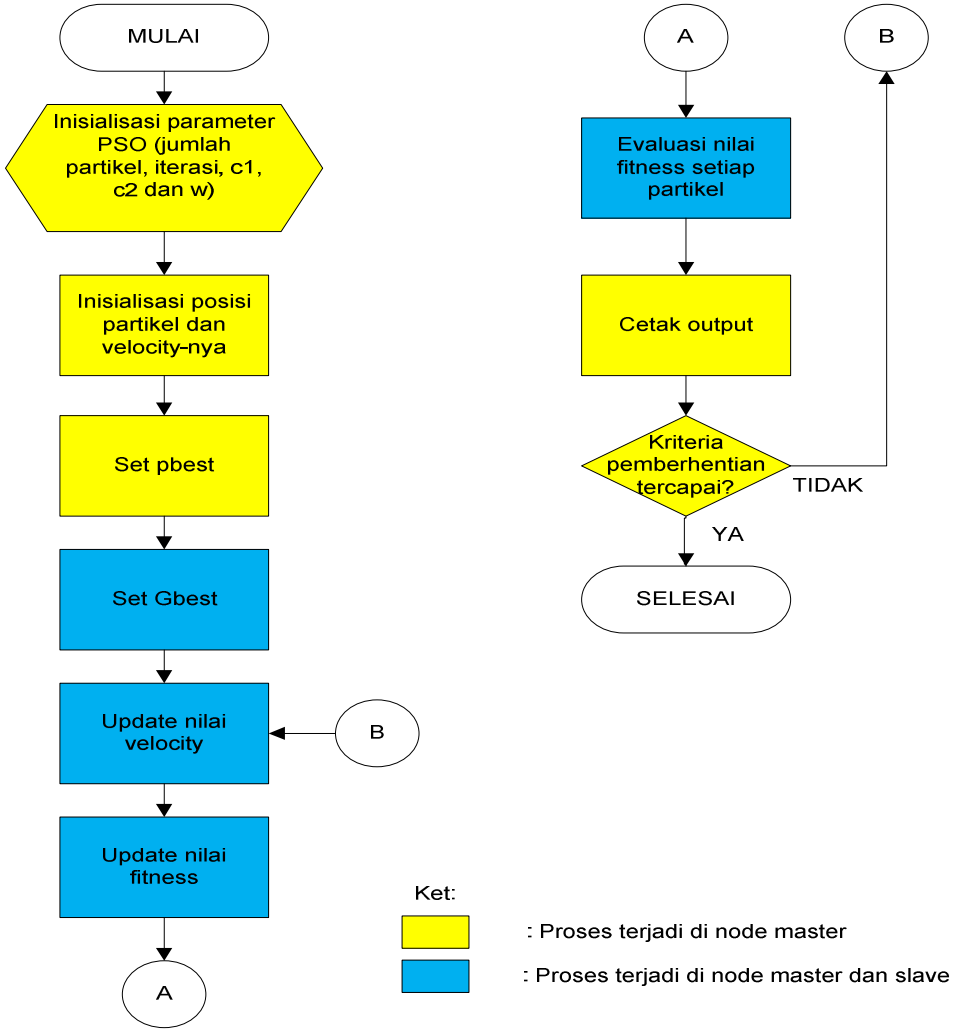
Gambar 2 Ilustrasi perancangan sistem

Tahap selanjutnya adalah pembuatan program PSO. Program PSO yang dibuat akan diujikan dalam 2 kasus yaitu:

1. Kasus pertama, menyelesaikan persamaan Rosenbrock.
   Adapun program PSO akan dibuat untuk menyelesaikan persamaan Rosenbrock berikut ini:

$$f(x,y) = (1-x)^2 + 100(y-x^2)^2 \quad \text{..... (13)}$$

Dan perancangan sistemnya dapat dilihat berikut ini:



Gambar 2 Perancangan sistem PSO saat komputasi paralel

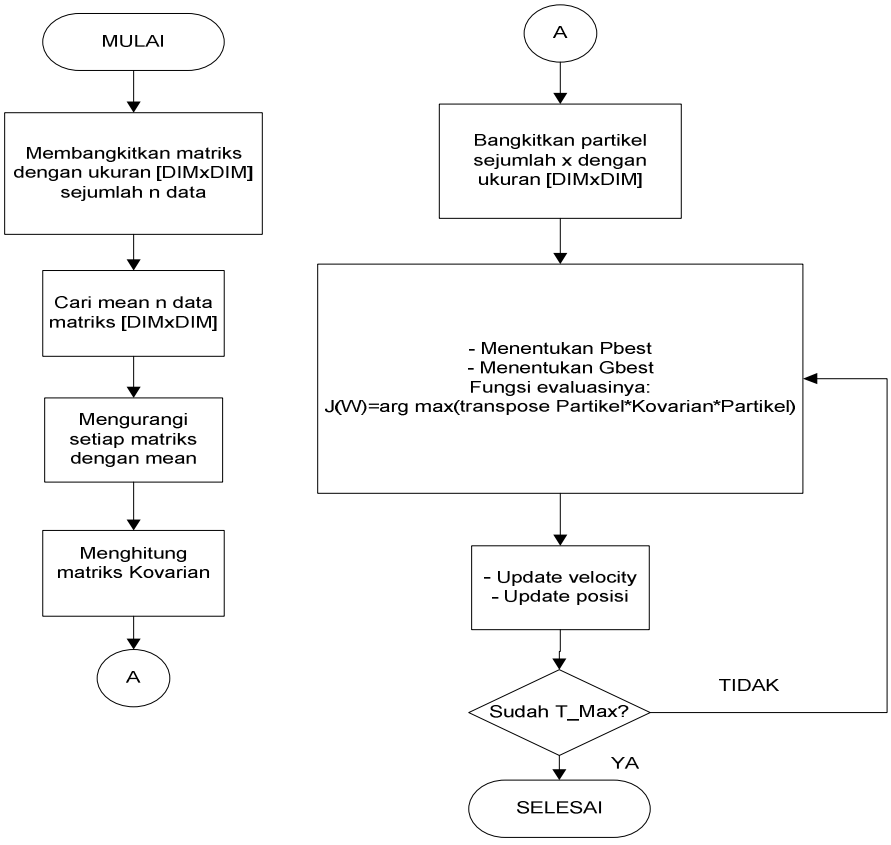
tahapan pengujian yang dilakukan pada kasus pertama ini adalah:

a. Pengujian algoritma, cara mengujinya adalah membandingkan output dari program PSO yang dibuat dengan hasil perhitungan secara manual untuk data yang sama, hal ini dilakukan untuk mengetahui apakah program PSO sudah berjalan sesuai algoritmanya.
b. Menguji pengaruh perubahan konstanta PSO dalam melakukan optimalisasi, cara mengujinya adalah dengan melakukan serangkaian percobaan dengan kombinasi c1, c2, jumlah partikel dan w yang berbeda. Akan ditinjau pengaruh perubahan konstanya PSO dalam melakukan optimasi pada saat iterasi maksimum yang sama (500 iterasi).
   Konstanta yang digunakan:
   - Partikel = {48, 192, 384, 3072, 6144}
   - c1 = {0.5, 1, 2}
   - c2 = {0.5, 1, 2}
   - w = {0.5, 1}
c. pergerakan partikel, pengujian dilakukan dengan cara menggunakan 20 partikel saja, kemudian output akan ditampilkan di setiap iterasi tertentu yang ingin dilihat setiap partikelnya berada diposisi mana. Rentang pergerakan partikel disebar pada interval -50 sampai dengan 50.

d. PSO saat komputasi paralel, PSO saat dikomputasi secara paralel, program yang dibuat akan dimodifikasi agar bisa berjalan pada lingkungan paralel. Program akan dijalankan pada 1, 2, 3 dan 4 komputer, kemudian dicatat waktunya serta dibandingkan dengan waktu komputasinya dengan saat *standalone*.

2. Kasus kedua, PSO menyelesaikan algoritma eigenface.
   Program dibuat dengan rancangan sebagai berikut ini.

PSO saat komputasi paralel, PSO saat dikomputasi secara paralel, program yang dibuat akan dimodifikasi agar bisa berjalan pada lingkungan paralel. Program akan dijalankan pada 1, 2, 3 dan 4 komputer,



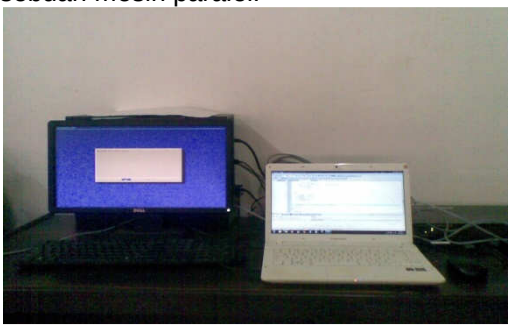
Gambar 3 diagram alir *Principle Component Analysist* dengan PSO

Program yang dibuat akan diuji dengan cara:

a. Menguji algoritmanya dengan mencari nilai kovariannya, dari sejumlah data, akan dihitung nilai kovariannya dengan bantuan matlab kemudian hasilnya dibandingkan dengan saat program berjalan.
b. Menguji perhitungan J(W)-nya dengan caramembandingkan output program dengan perhitungan menggunakan matlab dengan data yang sama. hal ini dilakukan untuk melihat pengaruh PSO dalam membantu pencarian eigenface.
c. Menguji dengan komputasi paralel. Sama halnya dengan kasus sebelumnya.

HASIL DAN PEMBAHASAN

Dalam penelitian ini perangkat keras disusun sedemikian rupa sehingga menjadi sebuah mesin paralel.



Gambar 4 Mesin paralel yang dibangun

Kasus 1 (PSO menyelsaikan persamaan Rosenbrock)

Program yang dibuat dibandingkan dengan perhitungan manual. Apabila sudah berjalan dengan tepat maka kemudian diuji pengaruh perubahan konstanta PSO dalam melakukan optimasi ditinjau dari output akhir gbestnya saat iterasi maksimum (500 iterasi). Partikel disebar dengan rentang persebaran sebesar -50 sampai dengan 50.

Dari hasil pengujian ini dapat ditarik hipotesa yaitu:

1. Gbest-nya relatif baik saat menggunakan c1 = 2, c2 = 0.5 dan bobot inertia (w) = 0.5.
2. Penggunaan bobot inertia dapat meredam laju partikel dalam melakukan penjelajahan.
3. Penggunaan c2 yang besar cendrung menghasilkan gbest yang keluar jalur.
4. Jumlah partikel tidak mempengaruhi hasil gbest secara signifikan

Selanjutnya ditinjau dari waktu komputasinya saat iterasi maksimum (500 iterasi) tercapai.

Tabel 3 Pengaruh perubahan konstanta terhadap nilai gbest

Konstanta			Gbest akhir dengan jumlah partikel [x,y]				
C1	C2	w	48	192	384	3072	6144
0.5	0.5	0.5	0.49,0.32	0.5,0.2	1.10,1.18	0.22,0.1	1.36, 1.86
0.5	1	0.5	0.46,0.26	0.6,0.38	0.6,0.31	72.4,-8.6	-73.3, 24
0.5	2	0.5	0.9,0.7	84,39.3	-76.4,19.6	-132.,0.67	102, -36.8
1	0.5	0.5	1.95,3.84	1.46,2.2	0.66,0.5	1.60,2.47	0.52, 0.33
1	1	0.5	2.41,5.80	76.6,-34.6	1.35,1.79	1.06,1.12	69.6, -20.3
1	2	0.5	0.60,0.37	-71.4,-90.3	78.65,-68	-146,39.4	84.4, 11.5
2	0.5	0.5	0.50,0.33	0.86,0.8	0.58,0.43	0.8,0.65	0.43, 0.24
2	1	0.5	1.27,1.71	0.26,0.09	-69.3,-24.8	-70.3,17	-82, 14.4
2	2	0.5	113,55.6	139.5,-43.6	-70.2,-27.4	-95,22.1	-86.2, 36.6
0.5	0.5	1	-94.4,30.4	77.1,37.75	76.2,-30.9	79.1,40.7	81.5, -2.1
0.5	1	1	-80,-5.39	-100.6,6.34	-86.1,-40.1	90.1,141	77.8, -8.4
0.5	2	1	-89.7,0.42	-116.4,36	-92.6,-172	79.8,29.3	187, -64.4
1	0.5	1	-82,37.5	-82.06,71.2	73.4,-24.7	-70.9,2.79	88.4, 20.7
1	1	1	70.6,-19.8	112,-112	-71,-21.4	-94,-26.7	76, -72.5
1	2	1	105,129	68.7,-11	126,-17.4	98.4,85.4	83.2, 32.2
2	0.5	1	87.91,64.5	83,213.8	-77.5,11.6	87.1,0.01	83.4, 18.2
2	1	1	102.8,60.7	72.25,13.6	-117.6,-66	74.8,26.9	74.4, 16.4
2	2	1	122.1,71.6	-89.8,-18.4	-108,13	90.5,87	85.6, -26.2

Tabel 4 Pengaruh perubahan jumlah partikel terhadap waktu program *standalone*

Konstanta			Partikel waktu eksekusi program dalam second (s)				
C1	C2	w	48 (s)	192 (s)	384 (s)	3072 (s)	6144 (s)
0.5	0.5	0.5	17.207	38.969	57.408	408.955	756.991
0.5	1	0.5	20.608	36.769	60.091	422.496	815.055
0.5	2	0.5	16.396	40.186	66.597	447.518	849.936
1	0.5	0.5	20.327	36.322	62.790	394.353	783.839
1	1	0.5	17.113	37.924	57.549	386.522	815.304
1	2	0.5	17.347	36.457	66.550	437.097	830.904
2	0.5	0.5	16.895	35.116	55.926	392.356	773.028
2	1	0.5	16.723	34.757	69.139	423.433	837.269
2	2	0.5	17.285	39.031	64.350	443.634	861.200
0.5	0.5	1	17.940	50.232	70.637	461.932	903.928
0.5	1	1	16.786	38.048	74.256	494.833	981.476
0.5	2	1	21.263	45.833	94.318	567.279	1198.472
1	0.5	1	19.016	38.563	70.590	485.020	967.108
1	1	1	14.524	62.494	68.219	497.048	1034.828
1	2	1	16.037	48.656	79.545	561.149	1199.970
2	0.5	1	16.318	44.335	78.109	550.447	1175.010
2	1	1	16.536	44.491	79.935	555.096	1185.555
2	2	1	20.561	54.366	86.908	623.829	1341.649
Rata-rata waktu			17.716	42.364	70.162	475.166	961.751

Dapat dilihat jika perubahan konstanta c1, c2 dan w tidak mempengaruhi waktu komputasi. Sedangkan penambahan jumlah partikel sangat berpengaruh dalam peningkatan waktu komputasi.

Saat rentang persebaran partikel-nya diperkecil menjadi -2 sampai dengan 2 dan -1 sampai dengan 1 program yang dibuat berjalan dengan lebih baik. Karena pada

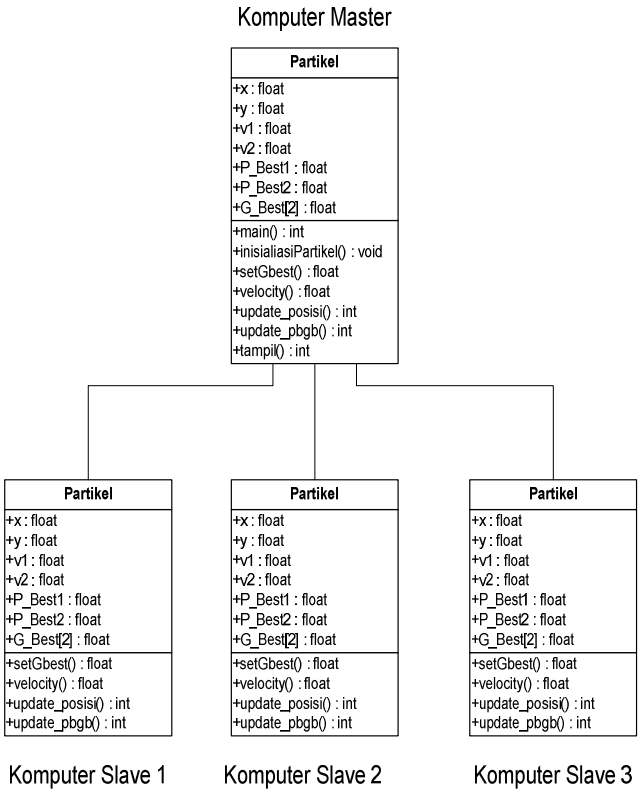
permasalahan ini kemungkinan menemukan optimalnya lebih tinggi saat rentang persebarannya dipersempit mendekati nilai optimum 1,1 untuk penyelesaian persamaan Rosenbrock.

Untuk program PSO yang dikomputasi paralel,perlu dilakukan pengujian algoritmanya dengan cara membandingkan hasilnya dengan perhitungan manual. Dalam penelitian ini hasil antara uji secara manual dengan saat program berjalan hasilnya sudah sama. sehingga dapat dikatakan program berjalan dengan benar.

Selanjutnya pengujian saat PSO dikomputasi secara paralel. Bagian-bagian yang dikomputasi paralel dan algoritmanya.

1. Set gbest.
   a. Pada rank master (rank ke-0) Simpan variabel-variabel ke dalam penyimpanan sementara untuk mempermudah pemecahan dan pengumpulan data nantinya.
   b. Bagi data pada array penyimpanan sementara sama rata ke seluruh rank aktif dengan MPI Scatter.
   c. Masing-masing rank akan mengevaluasi nilai gbestnya sehingga aka nada gbest sejumah rank.
   d. Gbest masing-masing rank akan dikirim ke master dengan bantuan MPI Gather.
   e. Pada master, semua nilai gbest tadi akan dievaluasi sehingga menyisakan 1 buah gbest yang paling baik saja yang akan digunakan untuk proses selanjutnya

2. *Update* nilai *velocity* dan posisi.
   a. Simpan variabel-variabel (x, y, pbest, gbest, velocity) ke penyimpanan sementara untuk mempermudah proses pemecahan data. Evaluasi nilai gbest lama dan simpan pada A.
   b. Data pada penyimpanan sementara akan dibagi sama rata ke rank aktif dengan MPI Scatter. Untuk gbest akan dibroadcast ke seluruh rank sehingga rank akan memperoleh gbest bernilai sama.
   c. Setiap rank akan meng - update velocitynya dan meng-update posisinya.
   d. Dengan MPI Gather, kumpulkan velocity baru dan posisi baru yang telah diupdate dari setiap rank ke rank *master*.



Gambar 5 UML program komputasi paralel

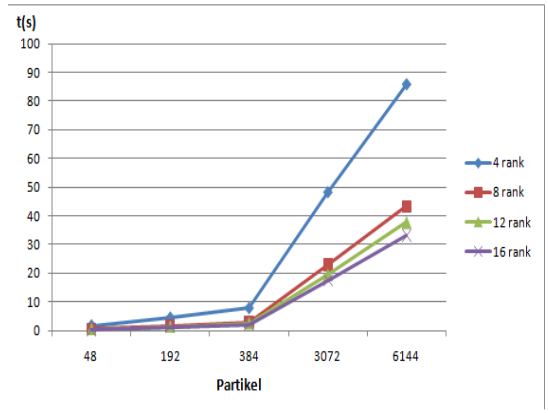
3. Evaluasi pbest dan gbest.
   a. Simpan variabel x, y, pbest dan gbest ke penyimpanan sementara.
   b. Data pada penyimpanan sementara akan dibagi sama rata ke rank aktif dengan MPI Scatter. Untuk gbest akan dibroadcast ke seluruh rank sehingga rank akan memperoleh gbest bernilai sama.
   c. Evaluasi nilai semua x dan y simpan hasil evaluasinya pada himpunan A. Evaluasi semua pbestnya dan simpan di himpunan B.
   d. Masih di setiap rank, bandingkan nilai A dan B. jika A lebih baik maka pbest adalah posisi baru, jika B lebih baik, maka pbest tetap seperti sebelumnya.
   e. Masih di setiap rank, untuk gbestnya. Evaluasi nilai pbestnya sehingga mendapatkan 1 yang terbaik dari setiap ranknya.
   f. Kumpulkan semua pbest baru dang best ke rank master dengan MPI Gather. Evaluasi kembali nilai gbest masing-masing rank simpan di B.
   g. Bandingkan A dan B (hasil evaluasi gbest sebelumnya) apabila A lebih baik, maka gbest yang digunakan adalah gbest lama, jika yang lebih baik B maka gbest yang baru akan digunakan.

Hasil dari pengujian program PSO dalam menyelesaikan persamaan Rosenbrock ditinjau dari perubahan jumlah rank terhadap waktu komputasinya dapat dilihat berikut ini

Tabel 5 Perbandingan waktu komputasi dengan jumlah rank

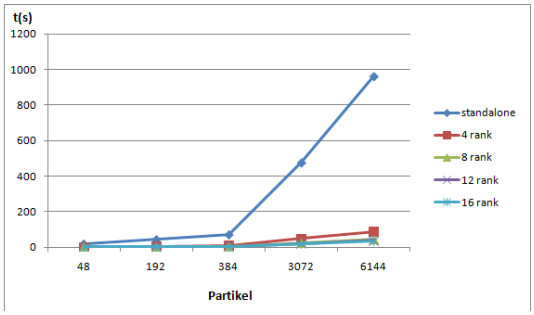
Jumlah Partikel / Proses	48 (s)	192 (s)	384 (s)	3072 (s)	6144 (s)
MPI 4 *rank*	1.68670	4.38816	7.72977	48.11524	85.73486
MPI 8 *rank*	0.46836	1.56640	2.87929	22.84469	43.32892
MPI 12 *rank*	0.40588	1.28482	2.44862	19.45933	37.75846
MPI 16 *rank*	0.29517	1.16667	2.14523	17.52626	33.07862

Dari pengujian pengaruh jumlah rank terhadap waktu didapatkan grafik sebagai berikut.



Gambar 6 Grafik pengaruh jumlah rank terhadap waktu komputasi

Apabila dibandingkan pengujian secara paralel dengan standalone didapatkan grafik perbandingan sebagai berikut



Gambar 7 Grafik perbandingan waktu komputasi saat standalone dengan komputasi paralel

Kasus kedua (PSO menyelesaikan algoritma eigenface)

Untuk menguji apakah program yang dibuat sudah benar. Kita perlu melakukan pengujian:

1. Membandingkan nilai kovarian pada hasil eksekusi program dengan perhitungan manual dengan bantuan matlab
2. Membandingkan nilai determinan J(W) hasil eksekusi program dengan perhitungan manual menggunakan matlab.

Pada penelitian ini untuk poin a, hasilnya sama, sedangkan untuk poin b terjadi

perbedaan antara eksekusi program dengan keluaran matlabnya. Dari sana dapat dicari persen error relatifnya.

$$\varepsilon_r = \left|\frac{1696775929856 - (1.6984e + 12)}{(1.6984e + 12)}\right| \times 100\%$$

$$= 0.956\%$$

KESIMPULAN

Berdasarkan hasil penelitian yang telah dilakukan maka dapat disimpulkan menjadi beberapa poin sebagai berikut.

Program PSO yang dibuat sudah sesuai dengan algoritma PSO setelah dilakukan perbandingan antara pengujian secara *manual* dengan saat program dijalankan baik saat komputer standalone maupun saat telah dikomputasi secara paralel.

Konstanta algoritma PSO yang hasilnya relatif baik saat menggunakan c1=2 dan c2=0.5 untuk rentang penyebaran partikel -50 sampai dengan 50. Sedangkan penggunaan bobot inertia (w) pada program yang dibuat dapat meredam laju pergerakan partikel.

Pencarian optimal dengan PSO untuk persamaan Rosenbrock lebih mudah dilakukan ketika pengujian dilakukan dengan rentang penyebaran partikelnya diperkecil menjadi -1 sampai dengan 1 dan -2 sampai dengan 2.

Waktu rata-rata yang digunakan untuk menjalankan program *standalone* untuk jumlah partikel terbesar yakni 6144 adalah sebesar 961.751s. sedangkan dengan komputasi paralel dengan rank 16 diperoleh waktu komputasi sebesar 33.07862s.

Percepatan yang diperoleh saat partikel uji terbesar (6144) pada komputasi paralel saat menggunakan 4 *rank* sebesar 11 kali lebih baik dari program *standalone*, saat menggunakan 8 *rank* sebesar percepatan yang didapat 22.2 kali lebih baik dari program *standalone*, saat menggunakan 12 rank percepatan yang diperoleh sebesar 25.5 kali lebih baik dari program *standalone* dan saat menggunakan 16 rank percepatan yang diperoleh sebesar 28.5 kali lebih baik dari program *standalone*.

Dari perbandingan uji coba menggunakan Matlab dengan hasil eksekusi program *standalone* yang telah dibangun. Nilai kovariannya keduanya sama. Sedangkan untuk nilai J(W)-nya terjadi error relatif sekitar 0.956%.

Untuk program Eigenface dengan PSO yang dikomputasi secara paralel tidak dapat diselesaikan karena terkendala batas waktu penelitian. Hal ini juga disebabkan pada pembuatan program ditemukan kendala-kendala yaitu program tidak dapat mengirim dan menerima data-data berupa banyak matriks pada *rank-rank* aktifnya. Sehingga proses perhitungan tidak dapat diselesaikan.

DAFTAR PUSTAKA

Annas, M. 2013. "*Studi Komputasi Paralel dengan Kasus Perhitungan Matriks Besar*".

Hassan, R., Cohanim, B., Weck, O.,. 2005. "*A Comparison of Particle Swarm Optimization and the Genetic Algorithm*". 46th AIAA/ASME/ ASCE/AHS/ASC Structures, Structural Dynamics & Materials Conference, Austin, Texas, 18 – 21 April 2005

Kurniawan, A. 2010. "*Pemrograman Paralel dengan MPI & C*". ANDI. Yogyakarta

Minartiningtyas, B.A, 2013. "*Algoritma Eigenface*" http://informatika.web.id/algoritma-eigenface.htm, diakses tanggal 19 November 2014

Wijaya, I.G.P.S., Uchimura, K., Koutaki, G., 2013. "*Face Recognition Using Holistic Features and Within Class Scatter-Based PCA*", GSTF International Jurnal on Computing (JoC), Vol. 3 No. 2.

Wilkinson, B., Allen, M.,2010."*Parallel Programming Teknik dan Aplikasi Menggunakan Workstation dan Komputasi Paralel*".Edisi 2, ANDI. Yogyakarta

url_1.2014."*Rosenbrock Function*" http://mathworld.wolfram.com/RosenbrockFunction.html diakses tanggal 12 November 2014