BAB I
PENDAHULUAN

1.1. Latar Belakang

Ilmu bukan bergerak tahap demi tahap melainkan revolusi demi revolusi. Selama revolusi bergolak, semua pandangan lama ditantang oleh pandangan yang lebih baru. Lambat laun semakin banyak orang menerima pandangan baru, mungkin karena mereka menerimanya selama menuntut ilmu di masa muda.

Kemajuan teknologi terjadi pun bukan karena pengembangan, tapi karena revolusi. Hal ini terbukti, pengembangan teknologi yang terjadi menyebabkan Hukum Moore semakin tidak relevan untuk meramalkan kecepatan mikroprosesor. Hukum Moore menyatakan kompleksitas sebuah mikroprosesor akan meningkat dua kali lipat tiap delapan belas bulan sekali, sekarang mendekati arah jenuh. Hal ini menyebabkan para peneliti yang haus akan kecepatan komputasi mencari jalan keluar untuk menjawab tantangan dan permasalahan yang mereka hadapi. Walaupun sudah memiliki komputer dengan spesifikasi yang sangat tinggi, apa yang sudah ada ini pun dirasa tetap kurang, karena mereka berusaha memecahkan permasalahan yang lebih besar lagi. Tuntutan yang terus meningkat dari komputer adalah kapasitas yang semakin besar dan kinerja yang semakin cepat, hal ini dapat diperoleh dengan komputasi paralel.

Pada analisis numerik pencarian akar-akar persamaan fungsi kompleks adalah permasalahan yang sulit. Metode iterasi Newton, interpolasi Muller, dan metode gradien adalah metode yang sering digunakan, tetapi metode-metode tersebut menuntut nilai awal yang tinggi. Iterasi *For* dan metode Newton memiliki masalah pada konvergensi, kecepatan konvergen, dan kesensitivan pada nilai awal; aritmatika iterasi sirkular terlalu kompleks untuk dihitung; metode *down mountain* cukup sederhana, tetapi mungkin akan jatuh ke tempat yang ekstrem.

Algoritma Genetika adalah metode efektif untuk menyimulasikan proses seleksi alam, metode ini dapat menemukan solusi yang memuaskan untuk beberapa permasalahan optimasi daripada metode-metode tradisional seperti pencarian *exhaustive* dan metode analitis.

Oleh karena itu pada penelitian ini akan dicoba menggunakan Algoritma Genetika Paralel (*Parallel Genetic Algortithm* [PGA]) untuk mencari akar-akar

persamaan fungsi kompleks, akan diadopsi pada sistem paralel, ini akan mereduksi skala penyelesaian permasalahan dengan metode *domain decomposition*. Algoritma ini sederhana dan praktis. Algoritma paralel akan diimplementasikan pada *Message Passing Interface* atau disingkat MPI adalah bahasa independen untuk protokol komunikasi yang digunakan program paralel pada komputer.

1.2. Rumusan Masalah

Untuk mencari akar fungsi kompleks menggunakan algoritma genetika paralel, maka perlu dilakukan rumusan masalah sebagai berikut:

1. Bagaimana mencari akar-akar persamaan fungsi kompleks?
2. Bagaimana mendesain dan mengimplementasikan algoritma genetika paralel pada pencarian akar-akar persamaan fungsi kompleks?
3. Bagaimana peningkatan kecepatan waktu eksekusi program algoritma genetika paralel dibandingkan dengan algoritma genetika serial dan seberapa efisien?

1.3. Ruang Lingkup

Penyusunan penelitian ini menggunakan batasan masalah sebagai berikut:

1. Pencarian akar dibuat dengan ketelitian 10^{-5}.
2. Pencarian akar-akar persamaan fungsi kompleks dibatasi maksimum tiga belas akar-akar persamaan.
3. Proses komputasi paralel diimplementasikan pada MPI.
4. Daya terpakai untuk proses komputasi tidak dihitung.

1.4. Tujuan

Tujuan penelitian ini adalah:

1. Meningkatkan performa komputasi pencarian akar fungsi kompleks menggunakan Algoritma Genetika Paralel berbasis MPI.
2. Mengetahui efisiensi algoritma genetika paralel berbasis MPI.

BAB II
TINJAUAN PUSTAKA

2.1. Pencarian Akar-Akar Persamaan Fungsi Kompleks

2.1.1 Fungsi Kompleks

Yang dimaksud dengan peubah kompleks z ialah suatu titik umum dari suatu himpunan tertentu pada bidang datar. Suatu definisi formal fungsi peubah kompleks dapat diberikan sebagai pasangan terurut dua bilangan kompleks (z, w) yang memenuhi syarat-syarat tertentu. Ini sebenarnya suatu proses pemadanan yang mengawankan setiap nilai peubah z ke nilai w yang tunggal. Definisi yang kurang formal adalah sebagai berikut.

Misalkan D adalah sebuah himpunan titik-titik pada bidang datar; andaikan dikenakan aturan f yang mengawankan setiap titik z pada D dengan satu dan hanya satu titik w pada bidang datar.

Maka, f dinamakan fungsi peubah kompleks, atau disingkat fungsi kompleks. huruf-huruf lain yang lazim digunakan untuk menunjukkan fungsi adalah g, h, k, ..., dan cara yang paling umum untuk mendefinisikan fungsi adalah menyatakan sebagai persamaan yang berbentuk

$$w = f(z) \qquad (2\text{-}1)$$

Himpunan D itu dinamakan domain fungsi f dan besaran $f(z)$ dinamakan nilai f pada z atau *image* (bayangan) z di bawah f.

2.1.2 Bentuk Kutub; Teorema DeMoivre

Jika $z = x + iy$ adalah bilangan kompleks taknol, $r = |z|$, dan θ mengukur sudut dari sumbu riil positif ke vektor z, maka seperti disarankan oleh Gambar 2.1:

$$x = r\cos\theta \qquad (2\text{-}2)$$

$$y = r\sin\theta \qquad (2\text{-}3)$$

$$z = x + iy \qquad (2\text{-}4)$$

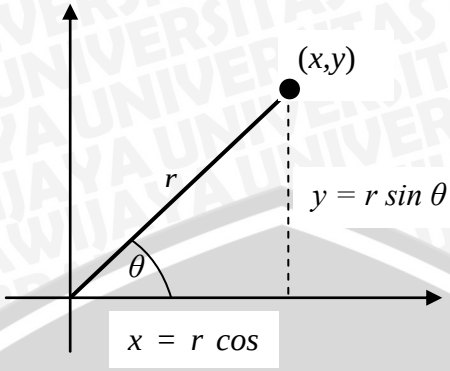
Sehingga dengan demikian $z = x + iy$ dapat dituliskan sebagai

$$z = r\cos\theta + ir\sin\theta \qquad (2\text{-}5)$$

atau

$$z = r(\cos\theta + i\sin\theta) \qquad (2\text{-}6)$$

Ini disebut **bentuk kutub dari z**.



Gambar 2.1. bentuk kutub dari *z*

Sudut θ disebut **argumen** z dan dinyatakan oleh

$$\theta = \arg z \qquad (2\text{-}7)$$

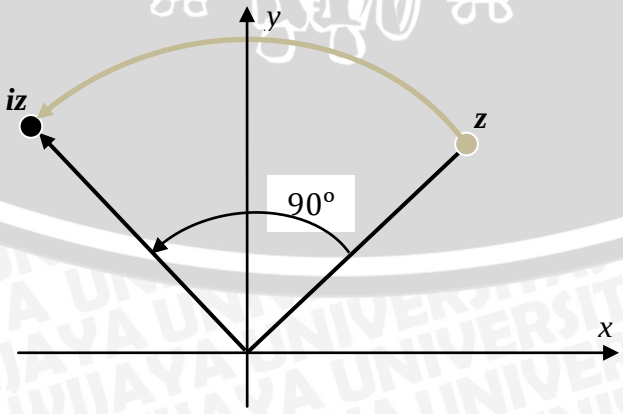
Argumen *z* tidak ditentukan secara unik karena dapat ditambahkan atau dikurangkan sembarang kelipatan 2π dari θ untuk menghasilkan nilai argumen yang lain. Namun demikian, hanya terdapat satu nilai argumen dalam radian yang memenuhi

$$-\pi < \theta \leq \pi \qquad (2\text{-}8)$$

Ini disebut **argumen utama z** dan dinyatakan oleh

$$\theta = Arg\, z \qquad (2\text{-}9)$$

Bilangan kompleks *i* mempunyai modulus 1 dan argumen π/2 (=90^o), sehingga hasil kali *iz* mempunyai modulus sama seperti z, tetapi argumennya 90^o lebih besar dari argumen z. Singkatnya, perkalian *z* dengan *i* memutar z berlawanan arah jarum jam sebesar 90^o (Gambar 2.2)



Gambar 2.2. Perkalian *z* dengan *i*

Jika *n* bilangan bulat positif dan $z = r(\cos\theta + i \sin\theta)$, maka

$$z^n = \underbrace{z \cdot z \cdot z \cdots z}_{n\text{-faktor}} = r^n[\cos(\underbrace{\theta + \theta + \cdots + \theta}_{n\text{-suku}}) + i\ \sin(\underbrace{\theta + \theta + \cdots + \theta}_{n\text{-suku}})] \quad (2\text{-}10)$$

$$z^n = r^n(\cos n\theta + i \sin n\theta) \quad (2\text{-}11)$$

Dalam kasus khusus dimana $r = 1$, dimana $z = \cos\theta + i \sin\theta$, sehingga menjadi

$$(\cos\theta + i \sin\theta)^n = \cos n\theta + i \sin n\theta \quad (2\text{-}12)$$

yang disebut **rumus DeMoivre**. rumus ini akan sahih untuk semua bilangan bulat n.

Rumus DeMoivre dapat dipakai untuk memperoleh akar-akar bilangan kompleks. Jika n bilangan bulat positif, dan z sembarang bilangan kompleks, maka dapat didefinsikan **akar yang ke-*n* dari *z*** adalah sembarang bilangan kompleks yang memenuhi persamaan

$$w^n = z \quad (2\text{-}13)$$

Nyatakan akar ke-n dari z dengan $z^{1/n}$. Jika $z \neq 0$, maka rumus dapat diturunkan untuk akar ke-n dari z sebagai berikut. Misalkan

$$w = \rho(\cos\alpha + i\ \sin\alpha) \quad \text{dan} \quad z = r(\cos\theta + i \sin\theta)$$

Jika diasumsikan bahwa w memenuhi (7-10), maka dari (7-9) menyusul bahwa

$$\rho^n(\cos n\alpha + i\ \sin n\alpha) = r(\cos\theta + i\ \sin\theta) \quad (2\text{-}14)$$

Dengan membandingkan modulus-modulus dari kedua ruas, dapat dilihat bahwa $P^n = r$ atau

$$\rho = \sqrt[n]{r} \quad (2\text{-}15)$$

dimana $\sqrt[n]{r}$ menyatakan akar riil positif ke-n dari r. Lebih lanjut, agar dalam (7.1.2.12) mempunyai $\cos n\alpha = \cos\theta$ dan $\sin n\alpha = \sin\theta$, sudut-sudut $n\alpha$ dan θ haruslah sama atau dibedakan oleh suatu kelipatan 2π. Yakni,

$$n\alpha = \theta + 2k\pi, \qquad k = 0, \pm1, \pm2, \cdots \quad (2\text{-}16)$$

$$\alpha = \frac{\theta}{n} + \frac{2k\pi}{n}, \qquad k = 0, \pm1, \pm2, \cdots \quad (2\text{-}17)$$

Jadi, nilai-nilai $w = P(\cos\alpha + i \sin\alpha)$ yang memenuhi (2-14) diberikan oleh

$$w = \sqrt[n]{r}\left[\cos\left(\frac{\theta}{n} + \frac{2k\pi}{n}\right) + i \sin\left(\frac{\theta}{n} + \frac{2k\pi}{n}\right)\right] \quad k = 0, \pm1, \pm2, \quad (2\text{-}18)$$

Walaupun terdapat takberhingga banyaknya nilai k, dapat diperlihatkan bahwa k = 0, 1, 2, . . . , n – 1 menghasilkan nilai w yang berbeda yang memenuhi (9.14), tetapi semua pilihan k lainnya memberikan duplikatnya. Karenanya, terdapat tepat n buah akar-akar ke-n yang berlainan dari $z = r(\cos\theta + i\ \sin\theta)$, dan ini diberikan oleh

$$z^{\frac{1}{n}} = \sqrt[n]{r}\left[\cos\left(\frac{\theta}{n}+\frac{2k\pi}{n}\right) + i\sin\left(\frac{\theta}{n}+\frac{2k\pi}{n}\right)\right] \quad k = 0, 1, 2, \cdots, n-1 \quad \text{(2-19)}$$

2.1.3. Konversi Penyelesaian Persamaan ke Pencarian Nilai Minimum

Prinsip Argument menyatakan bahwa, jika fungsi $f(z)$ adalah *meromorphic* pada daerah S, r adalah koreksi dari kurva Jordan, di dalam S dan bagian dalam juga termasuk S, $f(z)$ tidak memiliki nol-nol (*zeroes*) dan tiang-tiang (*poles*) pada r, maka

$$N - P = \frac{1}{2\pi i}\int_r \frac{f'(z)}{f(z)}dz \quad \text{(2-20)}$$

Disini, N dan P cacah rata-rata dari nol dan tiang $f(z)$ di dalam r. Saat $f(z)$ analitik di S, persamaannya dapat ditransformasikan menjadi

$$N = \frac{1}{2\pi i}\int_r \frac{f'(z)}{f(z)}dz = \frac{1}{2\pi}\,\Delta r\, Arg\, f(z) \quad \text{(2-21)}$$

Jika $f(z) = u + iv$ analitik pada kurva tertutup sederhana C, maka pada saat yang sama juga analitik pada C, dan $f(z)$ taknol di C, maka cacah akar $f(z)$ pada C adalah sama dengan nilai $\frac{1}{2\pi}$ dikalikan perubahan sudut saat z membuat sirkuit pada C pada arah positif.

2.1.4. Mencari Nilai Minimum

Semua permasalahan penyelesaian persamaan termasuk pencarian akar-akar persamaan fungsi kompleks, dapat dikonversikan ke pencarian nilai minimum dari persamaan, Untuk fungsi kompleks $f(z) = 0$, dapat dikonversi ke optimasi minimum:

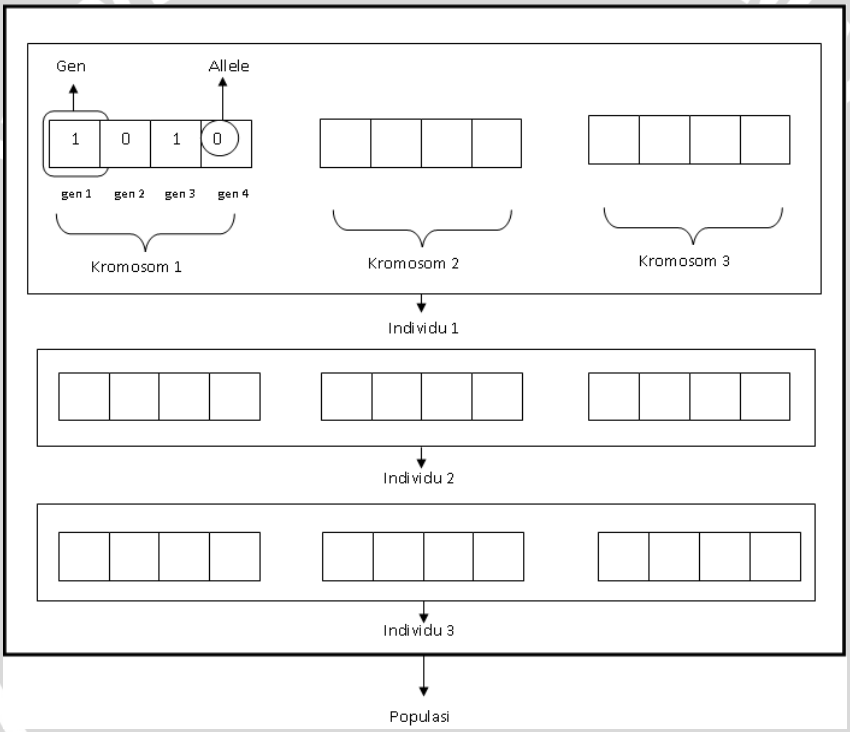
$$\boldsymbol{min}|f(z)| \quad \text{(2-22)}$$

Jika permasalahan memiliki dimensi yang besar, GA akan tidak efisien. Bagaimana menurunkan dimensi permasalahan adalah penting untuk meningkatkan efisiensi GA. Pada algoritma, akan dibatasi solusi di dalam lingkaran atau busur sirkular dari permasalahan. Untuk fungsi kompleks *f(z)* = 0, terdapat dua dimensi variabel: bagian riil *x* dan bagian imajiner *y*, akan direduksi

keduanya menjadi variabel satu dimensi, itu adalah mengonversi pencarian nilai minimal dari $|f(x,y)|$ untuk (x,y) ke pencarian nilai minimum dari $|f(\rho)|$ untuk ρ(radian), ini akan meningkatkan efisiensi GA.

2.2. Algoritma Genetika

Algoritma genetika yang termasuk dalam metode *heruristic* atau *non-determisnistic* adalah suatu model algoritma optimisasi yang sangat umum berdasarkan pada proses seleksi alam. Algoritma ini adalah salah satu dari algoritma evolusi (*evolutionary algorithms*), dan mirip tapi berbeda dengan *stochastic beam search*, algoritma genetika terbentuk dari ide reproduksi seksual (*sexual reproduction*), atau penggabungan-ulang genetika (*genetic recombination*). Visualisasi algoritma genetika diperlihatkan pada Gambar 2.3.



Gambar 2.3. Visualisasi gen, allele, kromosom, individu, dan populasi pada algoritma genetika

2.2.1. Pengertian Dasar Algoritma Genetika

Berikut beberapa pengertian dasar algoritma genetika yang perlu diketahui:

a. Gen (Genotype) adalah variabel dasar yang membentuk suatu kromosom. Dalam algoritma genetika, gen ini bernilai biner, float, integer, maupun karakter.

b. Allele adalah nilai dari suatu gen, bisa berupa biner, float, integer, maupun karakter.
c. Kromosom adalah gabungan dari gen-gen yang membentuk arti tertentu. Ada beberapa macam bentuk kromosom, yaitu:

   Kromosom biner, adalah kromosom yang disusun dari gen-gen yang bernilai biner. Kromosom ini mempunyai tingkat keberhasilan yang tinggi. Jumlah gen pada kromosom biner menunjukkan tingkat ketelitian yang diharapkan. Kromosom ini bagus bila digunakan untuk permasalahan yang parameter dan *range* nilainya tertentu.

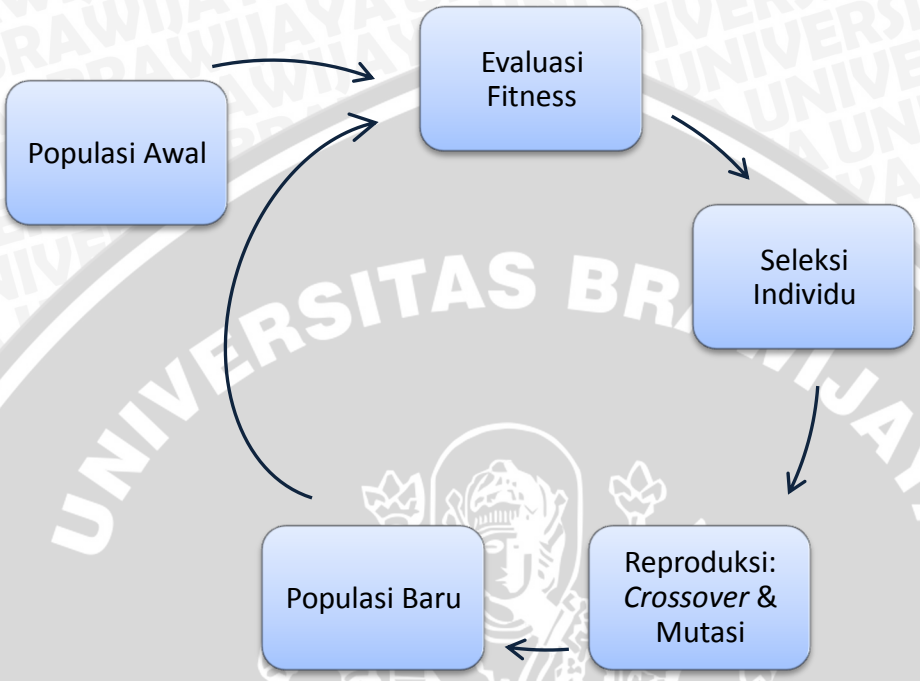
   Kromosom float, adalah kromosom yang disusun dari gen-gen yang bernilai pecahan, termasuk gen yang bernilai bulat. Kromosom ini merupakan model yang jumlah parameternya banyak. Tingkat keberhasilan dari kromosom ini rendah dalam kecepatan (jumlah generasi). Model *crossover* dan mutasi pada kromosom ini sangat berbeda dengan kromosom biner sehingga diperlukan strategi khusus untuk melakukan crossover dan mutasi. Nilai *range* (*min*, *max*) menjadi tidak penting.

   Kromosom string, yaitu kromosom yang disusun dari gen-gen yang bernilai string.

   Kromosom kombinatorial, yaitu kromosom yang disusun dari gen-gen yang dinilai berdasarkan urutan.
d. Individu adalah kumpulan gen, bisa dikatakan sama dengan kromosom. Individu menyatakan salah satu kemungkinan solusi dari suatu permasalahan.
e. Populasi adalah sekumpulan individu yang akan diproses secara bersama-sama dalam satu siklus proses evolusi.
f. Generasi menyatakan satu satuan siklus proses evolusi.
g. Nilai *Fitness* menyatakan seberapa baik nilai dari suatu individu atau solusi yang didapatkan. Nilai inilah yang dijadikan acuan untuk mencapai nilai optimal. Algoritma genetika bertujjuan untuk mencari individu yang mempunyai nilai *fitness* yang paling optimal (bisa maksimum atau minimum, tergantung pada kebutuhan.)

2.2.2. Siklus Algoritma Genetika

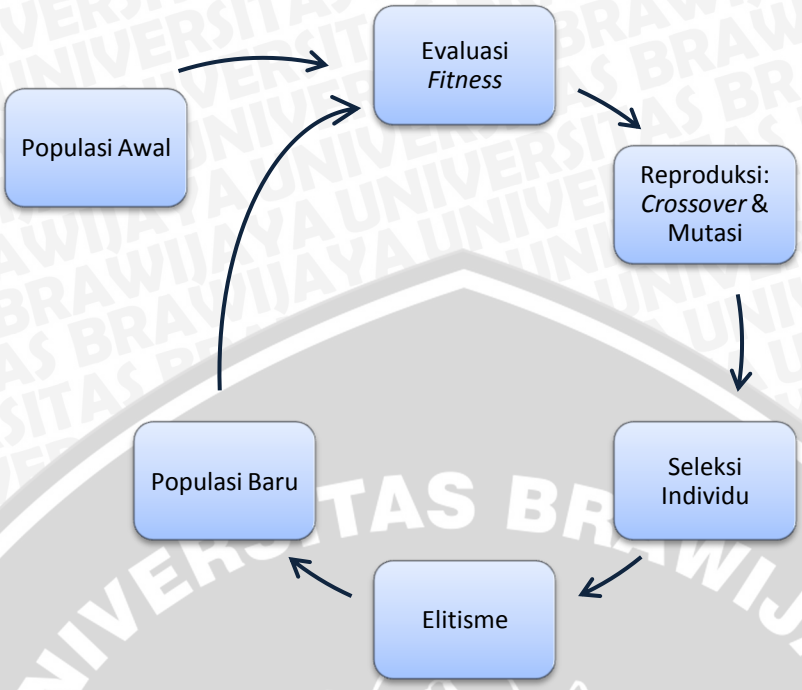
David Goldberg adalah orang yang pertama kali memperkenalkan siklus algoritma genetika yang digambarkan seperti berikut (Gambar 2.4):



Gambar 2.4. Siklus algoritma genetika yang diperkenalkan oleh David Goldberg

Siklus dimulai dari membuat populasi awal secara acak, kemudian setiap individu dihitung nilai *fitness*-nya. Proses berikutnya adalah menyeleksi individu terbaik, kemudian dilakukan *crossover* dan dilanjutkan oleh proses mutasi sehingga terbentuk populasi baru. Selanjutnya populasi baru ini mengalami siklus yang sama dengan populasi sebelumnya. Proses ini berlangsung terus hingga generasi ke-n.

Siklus ini kemudian diperbaiki oleh Zbigniew Michalewicz dengan menambahkan suatu proses elitisme dan membalik proses reproduksi dahulu, kemudian proses seleksi seperti tampak pada Gambar 2.5.



Gambar 2.5. Siklus algoritma genetika Zbigniew Michalewicz hasil perbaikan dari siklus algoritma genetika yang diperkenalkan oleh David Goldberg

2.2.3. Komponen Utama Algoritma Genetika

Untuk mengimplementasikan algoritma genetika, ada **delapan** komponen utama yang harus dilakukan.

2.2.3.1. Teknik *Encoding/Decoding* Gen dan Individu

Decoding (pendekodean) berguna untuk mendekode gen-gen pembentuk individu agar nilainya tidak melebihi range yang telah ditentukan dan sekaligus menjadi varabel yang akan dicari sebagai solusi permasalahan. Jika nilai variabel x yang telah dikodekan tersebut *range*-nya diubah menjadi $[r_b \;\; r_a]$, yaitu r_b batas bawah, r_a batas atas, maka cara untuk mengubah nilai-nilai variabel di atas hingga berada dalam range yang baru $[r_b \;\; r_a]$, disebut *encoding* (pengodean).

1) Pendekodean bilangan real:

$$x = r_b + (r_b - r_a)g \qquad (2\text{-}23)$$

2) Pendekodean diskrit desimal:

$$x = r_b + (r_b - r_a)(g_1 \times 10^{-1} + g_2 \times 10^{-2} + \ldots + g_N \times 10^{-N}) \qquad (2\text{-}24)$$

3) Pendekodean biner:

$$x = r_b + (r_b - r_a)(g_1 \times 2^{-1} + g_2 \times 2^{-2} + \ldots + g_N \times 2^{-N}) \qquad (2\text{-}25)$$

Dengan catatan bahwa N adalah jumlah gen dalam individu.

2.2.3.2. Membangkitkan Populasi awal

Sebelum membangkitkan populasi awal, terlebih dahulu harus menentukan jumlah individu dalam populasi tersebut. Misalnya jumlah individu tersebut N. Setelah itu, barulah membangkitkan populasi awal yang mempunyai N individu secara *random*.

2.2.3.3. Nilai *Fitness*

Nilai fitness menyatakan nilai dari fungsi tujuan. Tujuan dari algoritma genetika adalah memaksimalkan nilai *fitness*. Jika yang dicari nilai maksimal, maka nilai *fitness* adalah nilai dari fungsi itu sendiri. Tetapi jika yang dibutuhkan adalah nilai minimal, maka nilai *fitness* merupakan *invers* dari nilai nilai fungsi itu sendiri. Proses *invers* dapat dilakukan dengan cara berikut.

$$Fitness = C - f(x) \text{ atau } Fitness = \frac{C}{f(x)+\varepsilon} \qquad (2\text{-}26)$$

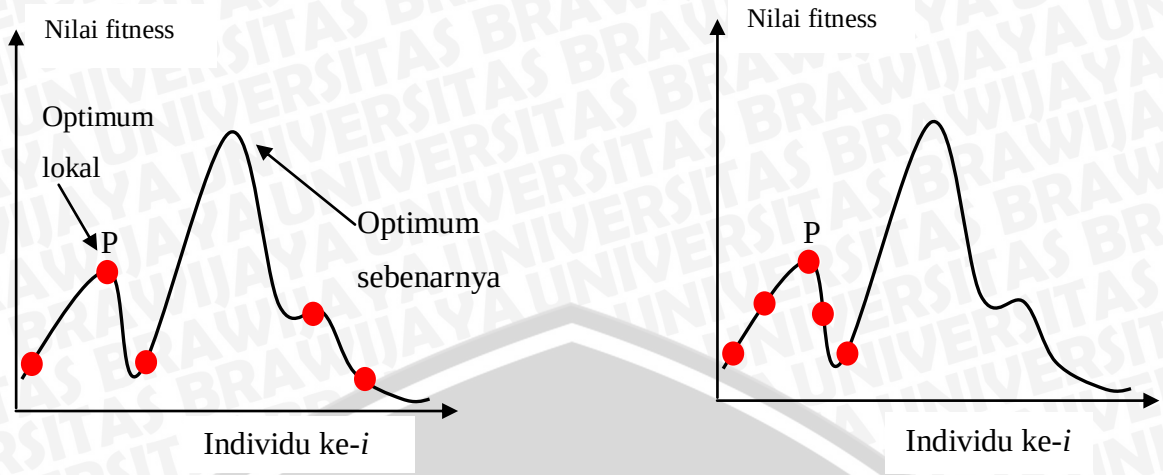
C adalah konstanta dan ε adalah bilangan kecil yang ditentukan untuk menghindari agar tidak terjadi pembagian oleh nol dan x adalah individu.

2.2.3.4. Elitisme

Elitisme adalah prosedur untuk menggandakan individu yang mempunyai nilai fitness tertinggi sebanyak satu (bila jumlah individu dalam satu populasi adalah ganjil) atau dua (bila jumlah individu dalam satu populasi genap). Hal ini dilakukan agar individu ini tidak mengalami kerusakan (*fitness*-nya menurun) selama proses pindah silang maupun mutasi. biasanya individu dengan *fitness* terbaik di-*copy* , kemudian disimpan dalam variabel *temporer_individu*.

2.2.3.5. Seleksi Induk

Kadang kala suatu fungsi tertentu dapat menyebabkan setiap individu mempunyai nilai *fitness* hampir sama. Hal ini bisa berakibat buruk pada saat dilakukan proses seleksi untuk memilih orang tua (induk atau *parent*) karena dapat menyebabkan optimum lokal (Gambar 2.6).



Gambar 2.6. Pada gambar sebelah kiri, individu P mempunyai nilai fitness tertinggi dibandingkan individu lainnya. Pada gambar sebelah kanan, populasi konvergen pada suatu optimum lokal dekat individu P. tidak ada individu yang mencapai nilai optimum sebenarnya

Pada gambar sebelah kiri (Gambar 2.6.), semua individu berada sangat jauh dari titik optimal sesungguhnya. Kebetulan ada satu individu P yang nilai *fitness*-nya tertinggi dibanding individu lainnya. Akibatnya P akan memproduksi banyak anak. Pada generasi tertentu individu P dan anak-anaknya, melalui proses pindah silang dan mutasi, akan menghasilkan individu-individu yang mendekati lokal optimum (gambar sebelah kanan) sehingga terjadilah konvergensi prematur, dimana populasi konvergen pada suatu solusi optimal lokal. Permasalahan inilah yang sering muncul pada algoritma genetika.

2.2.3.6. *Linear Fitness Ranking*

Untuk menghindari hal ini, dibuatlah suatu mekanisme yang disebut dengan *Linear Fitness Ranking (LFR)*. Tujuan dari mekanisme ini sebenarnya adalah untuk melakukan penskalaan nilai-nilai *fitness* dengan menggunakan persamaan berikut:

$$LFR = f_{max} - (f_{max} - f_{min}) \left(\frac{R(i)-1}{N-1}\right) \quad (2\text{-}27)$$

Dengan catatan bahwa:

LFR : nilai *LFR* individu ke-*i*

N : jumlah individu dalam populasi

R(i) : ranking individu ke-*i* setelah diurutkan dari nilai *fitness* terbesar hingga terkecil

f_{max} : nilai *fitness* tertinggi

f_{min} : nilai *fitness* terendah

Pertama, harus dilakukan *sorting* terhadap nilai *fitness* tersebut. Kemudian, seleksi digunakan untuk memilih dua buah individu yang akan dijadikan orang tua, kemudian dilakukan pindah-silang (cross-over) untuk mendapatkan keturunan yang baru. Metode seleksi yang sering digunakan adalah *Roulette-wheel* (Roda Roulette). Dengan menggunakan roda roulette, masing-masing individu menempati potongan lingkaran roda secara proporsional sesuai nilai *fitness*-nya.

2.2.3.7. *Roullete Wheel*

Pemilihan dilakukan secara acak dengan membangkitkan bilangan *random*. Jika probabilitas individu ke-*i* < bilangan *random*, maka individu ke-*i* terpilih sebagai orang tua.

2.2.3.8. *Crossover* (Pindah-silang)

Sebuah individu yang mengarah pada solusi optimal bisa diperoleh melalui proses pindah silang, dengan catatan bahwa pindah silang hanya bisa dilakukan jika sebuah bilangan *random r* dalam interval [0 1] yang dibangkitkan nilainya lebih kecil dari probabilitas tertentu *prob*, dengan kata lain: $r < prob$. Biasanya nilai *prob* diatur mendekati 1. Cara yang paling sederhana untuk melakukan pindah silang adalah pindah silang satu titik potong. Posisi titik potong dilakukan secara *random*.

2.2.3.9. Mutasi

Mutasi dilakukan untuk semua gen yang terdapat pada individu, jika bilangan *random* yang dibangkitkan lebih kecil dari probabilitas mutasi p_m yang ditentukan. Biasanya p_m diatur = 1/N, dimana N adalah jumlah gen dalam individu. Untuk kode biner, mutasi dilakukan dengan cara membalik nilai bit 0 menjadi bit 1, sebaliknya bit 1 diubah menjadi bit 0.

2.2.3.9.1. Teknik Mutasi

Pada gen yang mempunyai tipe data biner, mutasi dilakukan dengan menukar bit 0 menjadi 1, dan bit 1 menjadi 0. Untuk gen yang mempunyai tipe data real, mutasi dilakukan dengan cara menggeser nilai gen termutasi (pada posisi T, dimana T = *random*) sebesar bilangan kecil ε yang ditentukan dalam interval [0 1]. Nilai gen yang baru adalah:

$$x'(r) = x(r) + \varepsilon \qquad (2\text{-}28)$$

2.2.3.10. *Generational Replacement* (Penggantian Populasi)

Penggantian populasi dimaksudkan bahwa semua individu awal dari satu generasi diganti oleh *temporer_individu* hasil proses pindah silang dan mutasi.

2.2.4. Langkah-langkah Algoritma Genetika

Gambaran singkat langkah-langkah Algoritma Genetika sebagai berikut:

Langkah 1: populasi memiliki P individu yang dibangkitkan secara acak dengan *pseudo random generator* disebut populasi awal (*initial population*) dimana individu-individu tersebut mewakili solusi. Ini representasi dari solusi vector pada ruang solusi yang dinamakan solusi awal. Pencarian dimulai dari cakupan yang luas pada ruang solusi.

Langkah 2: Individu sebagai anggota populasi dievaluasi untuk mencari nilai paling objektif dari fungsi.

Langkah 3: Pada langkah ketiga, fungsi objektif memetakan tingkat kebugaran (*fitness*) tiap anggota populasi mengikuti ketentuan operator GA berikut.

1) Operator reproduksi: reproduksi dilakukan dengan seleksi Roda Roulette (*Roulette Wheel*). Dimana memilih kromosom dari populasi awal kemudian dimasukkan pada prosedur kawin (*crossover*).

2) Operator *crossover*: Rasio crossover (0 hingga 1) menentukan kemungkinan memproduksi kromosom baru dari orang tua (*parent*). Contoh, sebuah string 10000100 hingga 11111111 dapat dipindahsilangkan pada setiap setelah posisi ketiga untuk hasil dua keturunan 10011111 hingga 11100100. Operator *crossover* secara kasar meniru rekombinasi biologis diantara dua organisme kromosom tunggal (haploid).

3) Operator mutasi: operator ini secara acak membalik beberapa bit pada kromosom. Contoh, string 00000100 mungkin bisa bermutasi pada posisi kedua menjadi 01000100. Mutasi dapat terjadi pada setiap posisi bit pada string dengan banyak kemungkinan, biasanya sangat kecil (seperti 0,001).

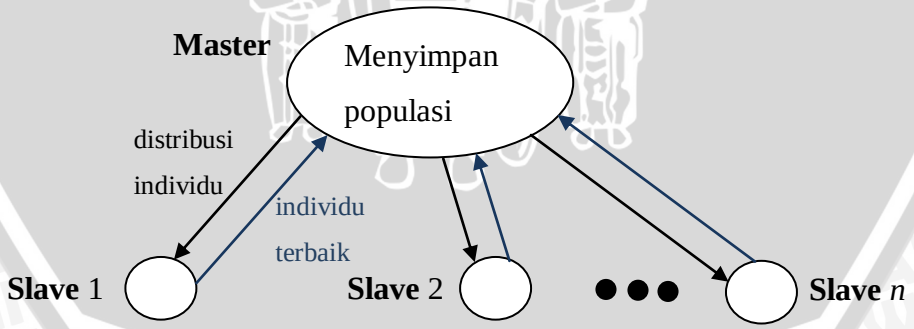
Pseudo code Langkah-langkah kerja Algotitma Genetika diperlihatkan pada Gambar 2.7.

```
Genetic Algorithm()
{
        Population initialization;
        Population evaluation;
        while (Termination criteria satisfied)
        {
                Selection for next population;
                Perform crossover;
                Perform mutation;
                Population evaluation;
        }
}
```

Gambar 2.7. Langkah-langkah kerja Algotitma Genetika

2.3. Algoritma Genetika Paralel

Ada tiga tipe utama PGA: (1) global single-population master-slave GAs, (2) single-population fine-grained, and (3) multiple-population coarse-grained GAs. tipe global single-population master-slave Gas akan digunakan pada penelitian ini. Pada master-slave GA terdapat suatu populasi (GA serial), tetapi evaluasi fitness didistribusikan pada beberapa processors (Lihat Gambar 2.8).



Gambar 2.8. skematik *master-slave* PGA. *Master* menyimpan populasi, eksekusi operasi GA, dan distribusi individu ke *slaves*. *Slaves* hanya mengevaluasi *fitness* individu

2.4. Komputasi Paralel

Komputasi adalah proses yang dijalankan prosesor dalam sebuah rentang satuan waktu yang meliputi:

a. operasi baca: menerima masukan berupa data dengan ukuran tertentu,

b. operasi penghitungan: melakukan operasi aritmatik atau logik terhadap masukan,

c. operasi tulis: mengembalikan hasil penghitungan dengan ukuran tertentu.

Komputasi paralel adalah eksekusi proses pengoperasian aritmatik atau logik yang sejenis secara bersamaan. Sebuah masalah komputasi yang kompleks dibagi menjadi bagian-bagian yang lebih kecil, sederhana, dan dijalankan di beberapa prosesor secara bersamaan sehingga keseluruhan proses dapat diselesaikan dengan lebih cepat.

Pada komputasi sekuensial (serial) masukan diterima dan keluaran dikirim melalui memori internal atau perangkat antarmuka. Sementara pada komputasi paralel sebuah prosesor dapat menerima masukan atau mengembalikan keluarannya melalui *shared memory* atau prosesor lain melalui bus interkoneksi antar prosesor atau jaringan komputer.

Tujuan utama komputasi paralel adalah peningkatan kecepatan komputasi. Hukum Amdahl dan Gustafon menjelaskan peningkatan kecepatan komputasi paralel.

Hukum Amdahl menjelaskan jika kecepatan komputasi 1 prosesor adalah: $S = \frac{1}{\alpha}$ dengan α adalah waktu yang dibutuhkan untuk menyelesaikan rutin komputasi dengan 1 prosesor. Kecepatan komputasi dari beberapa prosesor dijabarkan sebagai:

$S = \frac{1}{\frac{1-\alpha}{P}+\alpha}$ dengan P adalah banyaknya prosesor.

Hukum Gustafon menjelaskan kecepatan komputasi beberapa prosesor dengan persamaan sederhana:

$$S = \alpha + P(1-\alpha) \qquad (2\text{-}29)$$

Peningkatan kecepatan komputasi paralel secara teori dibatasi kendala sekuensial bahwa sebuah rutin tidak dapat dipecah menjadi unit yang lebih kecil dan harus dijalankan secara sekuensial di satu prosesor. Sementara secara

praktek peningkatan kecepatan komputasi paralel dibatasi konfigurasi lingkungan komputasi dan *overhead* rutin program karena komunikasi dan sinkronisasi antar proses.

Program yang dibuat untuk komputasi paralel memiliki kerumitan tersendiri dibanding program sekuensial karena memunculkan kendala baru karena kondisi pacu dan dependensi data antar proses. Kondisi pacu dapat ditangani dengan mekanisme lokalisasi dan *locking* data, sementara dependensi data ditangani dengan mekanisme komunikasi *message* antar proses. Kedua mekanisme tersebut membutuhkan sinkronisasi antar proses yang memunculkan tambahan rutin komputasi yang memperlambat dan menambah kerumitan program.

2.5. *Cluster* Beowulf

Cluster Beowulf adalah *cluster* komoditas yang menggunakan komponen komputer *consumer-grade* dengan harga yang murah (contoh: komponen komputer bekas pakai), menggunakan perangkat lunak gratis dengan lisensi yang bersifat FOSS (*free open source software)* untuk membangun keseluruhan sistem, dan mengakomodasi keperluan komputasi paralel. *Cluster* Beowulf memiliki keuntungan karena stuktur dan metode implementasinya, antara lain: skalabilitas, konvergensi, fleksibilitas konfigurasi, *failover*, kemudahan, kecepatan, dan harga implementasi yang murah dibandingkan sistem *cluster* konvensional.

Komputasi dengan *cluster* Beowulf melibatkan 4 hal dalam pertimbangan sistem:

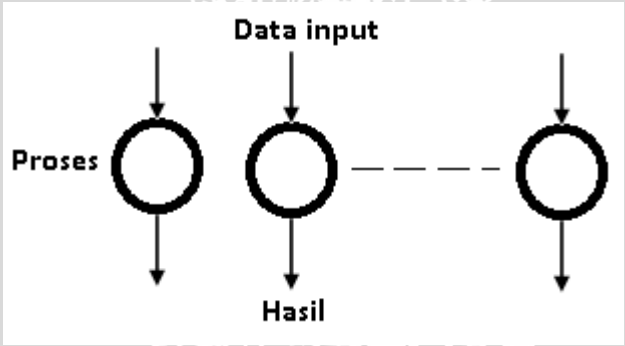
1. sistem perangkat keras,
2. manajemen sumber daya komputasi,
3. pustaka pemrograman paralel,
4. algoritma paralel.

Sistem perangkat keras meliputi aspek komponen perangkat keras tiap *node* komputasi, antarmuka dan topologi jaringan. Manajemen sumber daya komputasi adalah serangkaian perangkat lunak yang digunakan untuk instalasi, konfigurasi dan inisialisasi, administasi, dan pemantauan aktivitas, dan perawatan *cluster*. Pustaka pemrograman paralel adalah *framework* yang digunakan untuk

membuat program komputasi paralel. Algoritma paralel memberikan model dan pendekatan paralelisasi rutin komputasi.

2.6. *Embarrasingly parallel* (Komputasi Paralel Ideal)

Pemrograman paralel meliputi pembagian sebuah permasalahan ke dalam beberapa bagian, dimana prosesor yang terpisah melakukan komputasi dari setiap bagian tersebut. Komputasi paralel ideal adalah salah satu dari pemrograman paralel yang dapat dibagi segera ke dalam bagian yang sepenuhnya independen, yang dapat dieksekusi secara bersamaan. Gambaran itu disebut *Embarrasingly parallel* atau lebih tepat disebut *naturally parallel* atau paralel alami. Paralelisasi persoalan haruslah jelas dan tidak membutuhkan teknik khusus atau algoritma guna mendapatkan solusinya. Idealnya, tidak akan ada komunikasi anatara proses-proses yang terpisah, yakni grafik komputasi yang sepenuhnya tidak berhubungan, seperti Gambar 2.9.



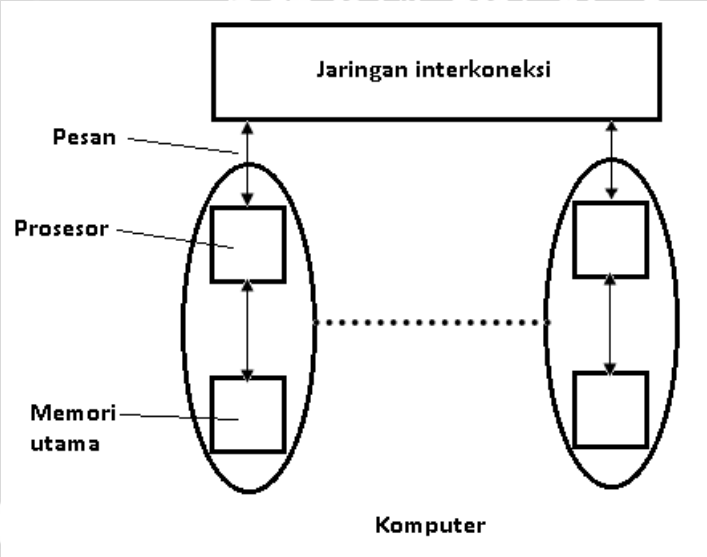
Gambar 2.9. Grafik komputasi yang tidak terhubung (e*mbarrasingly parallel problem*)

Setiap proses membutuhkan data yang berbeda (atau sama) dan memberikan hasil dari input data tanpa membutuhkan hasil dari proses lain. Kondisi tersebut memberikan kecepatan maksimum yang dimungkinkan apabila seluruh prosesor yang ada dapat ditugasi proses untuk total durasi dari komputasi tersebtu. Satu-satunya konsep yang diperlukan di sini hanyalah mendistribusikan data dan memulai proses. Model SPMD (*Single-Program Multiple-Data*) atau satu program dengan banyak data adalah model yang sesuai. Data tidaklah dibagi sehingga multikomputer message-passing adalah hal yang tepat. Jika data yang sama dibutuhkan, maka data tersebut harus disalin ke masing-masing proses. Karakteristik utamanya adalah tidak adanya interaksi antar proses.

Pada praktiknya *embarrasingly parallel computation*, data harus didistribusikan ke setiap proses dan hasilnya dikumpulkan serta dikombinasikan melalui beberapa cara. Hal itu memberi kesan bahwa dari awal hingga akhir, setiap proses harus beroperasi sendiri. Salah satu hal yang umum adalah pengorganisasian *master-slave*

2.7. *Message-Passing Multicomputer*

Bentuk multiprosesor yang menggunakan *shared*-memori dapat dibentuk dengan cara menghubungkan seluruh komputer ke suatu jaringan interkoneksi seperti terlihat pada Gambar 2.10. Setiap komputer terdiri atas satu memori yang tidak dapat diakses oleh prosesor lain. Jaringan interkoneksi memungkinkan pengiriman pesan antar prosesor. Pesan-pesan tersebut membawa data-data program. Sistem multiprosesor seperti ini disebut *massage-passing multiprosesor* atau hanya *multiprocesor*, terutama jika sistem tersebut terdiri atas komputer-komputer yang dapat beroperasi secara independen.



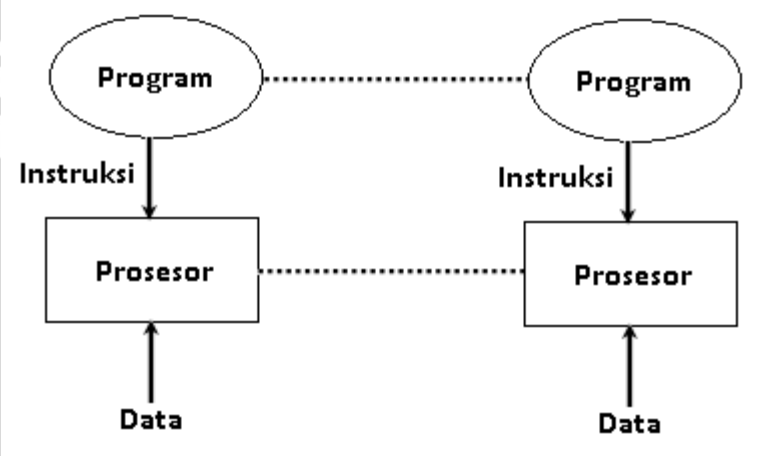
Gambar 2.10. Model Multiprosesor massage passing (multikomputer)

Umumnya, pendekatan yang digunakan adalah menggunakan *routine* massage-passing yang disisipkan pada program sekuensial konvensional yang digunakan untuk pengiriman pesan. Dalam konteks proses, suatu permasalahan dibagi menjadi beberapa proses yang terjadi bersama-sama untuk dijalankan di komputer berbeda. Jika ada 6 proses dan 6 komputer, maka bisa memiliki satu proses yang dijalankan masing-masing. Jika jumlah proses lebih banyak dari komputer yang ada, maka satu komputer menjalankan beberapa proses dengan model pembagian

waktu. Satu-satunya cara untuk mendistribusikan data dan hasil operasi adalah proses-proses tersebut saling berkomunikasi melalui pengiriman pesan.

2.8. *Single-Program Multiple-Data* (SPMD)

Pada struktur *single program multiple data* (Gambar 2.11.) sebuah *source* program tunggal dituliskan dan setiap prosesor akan mengeksekusi salinan miliknya sendiri dari program tersebut, meskipun itu dilakukan independen dan tidak singkron. Source program tersebut dapat dikonstruksi sedemikian rupa sehingga bagian dari program hanya dapat dieksekusi oleh komputer tertentu, bukan komputer lainnya, bergantung pada identitas komputer. Untuk sebuah struktur *master-slave*, program akan memiliki bagian-bagian untuk prosesor *master* dan bagian untuk prosesor *slave*.



Gambar 2.11. Struktur SPMD

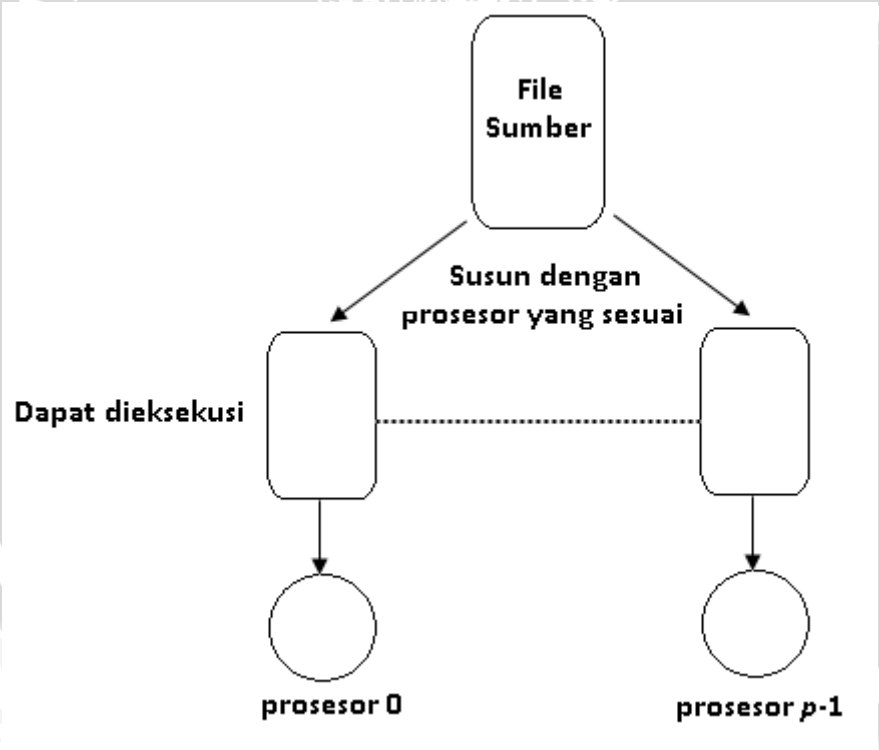
2.9. Pembuatan Proses

Untuk membangun proses dan memulai eksekusi, akan ditentukan metode pembuatan proses. Terdapat dua metode pembuatan proses: Pembuatan proses statis (*static process creation*) dan Pembuatan proses dinamis (*dynamic process creation*).

Pada pembuatan proses statis, semua proses telah ditentukan sebelumnya dan sistem akan mengeksekusi beberapa prosesor yang jumlah tetap. Sedangkan pada pembuatan proses dinamis, proses dibuat sementara eksekusinya diawali pada saat proses lain dieksekusi. Konstruksi pembuatan proses atau pemanggilan pustaka/sistem (*library/system calls*) biasanya digunakan untuk membuat proses. Proses juga bisa musnahkan.

Pada kebanyakan aplikasi, proses-proses biasanya tidak semuanya sama ataupun berbeda sama sekali. Biasanya terdapat satu proses pengontrol, "proses master (*master process*)" dan sisanya adalah "*slave*" atau "*worker*", Kedua istilah tersebut serupa dalam bentuk, tetapi berbeda dalam identifikasi proses (ID) saja.

Untuk Pembuatan proses statis, model single-program multiple-data (SPMD) sangat tepat digunakan. Pada model SPMD, program-program yang berbeda disatukan ke dalam satu program. Di dalam program terdapat statemen kontrol yang akan memilih bagian-bagian yang berbeda untuk setiap prosesnya. Setelah dibentuk program *source*, yang dilengkapi dengan statemen kontrol untuk memisahkan aksi pada setiap prosesor (lihat Gambar 2.12). Setiap prosesor akan memuat salinan dari kode tersebut ke memori lokal untuk eksekusi. Semua prosesor dapat memulai eksekusi kode secara bersamaan. Pendekatan ini yang paling umum untuk salah satu dari sistem message-passing umum, MPI.



Gambar 2.12. Model SPMD

2.10. *Message Passing Interface* (MPI)

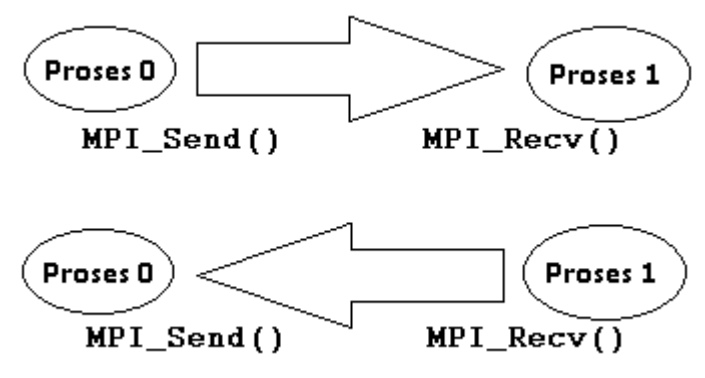
Ada banyak sekali bahasa pemrograman paralel yang diperkenalkan dan sebagian besar merupakan bahasa tingkat tinggi untuk menyederhanakan kompleksitas pemrograman, tetapi bahasa-bahasa tersebut tidak banyak diterima

di komunitas pemrograman paralel meskipun bahasa tersebut dari bahasa C ataupun Fortran.

Message Passing Interface atau disingkat MPI adalah bahasa independen untuk protokol komunikasi yang digunakan program paralel pada komputer. MPI diterima banyak komunitas pemrograman paralel.

2.11. Komunikasi *Point-to-Point* MPI

Mekanisme dasar sistem komunikasi pada MPI adalah proses pertukaran data pada sepasang proses dimana satu sebagai pengirim data dan satu sebagai penerima data, seperti yang diilustrasikan pada Gambar 2.13.



Gambar 2.13. Komunikasi *point-to-point* MPI

Ini yang disebut dengan komuniaksi *point-to-point*. Hampir sebagian besar komunikasi yang terjadi pada MPI didasarkan pada komunikasi *point-to-pointi* sehingga komunikasi ini sangatlah penting dan dasar untuk komunikasi MPI.

MPI menyediakan beberapa fungsi untuk mengirim dan menerima data baik secara *blocking* maupun *non-blocking*. *Blocking send/receive* adalah proses yang tidak akan mengembalikan nilai sampai buffer sudah penuh dengan data yang akan dikirim/diterima dan selanjutnya data dikirim/diterima. Ini artinya kode selanjutnya setelah *Blocking send/receive* tidak akan dieksekusi apabila proses b*locking send/receive* belum selesai. Sedangkan *non-blocking send/receive* akan mengembalikan nilai walaupun data yang dikirim/diterima belum dieksekusi.

Pada MPI, semua operasi komunikasi dieksekusi oleh *communicator*. Sebuah *communicator* mewakili sebuah komuikasi dominan yang merupakan kumpulan proses yang saling tukar-menukar data. Umumnya domain komunikasi yang digunakan adalah `MPI_COMM_WORLD` yang menangkap semua proses yang

terjadi pada program paralel. `MPI_COMM_WORLD` bertipe data `MPI_Comm` yang mana sudah dideklarasikan pada *file header* mpi.h, berikut:

```
typedef int MPI_Comm;
#define MPI_COMM_WORLD    ((MPI_Comm)0x44000000))
#define MPI_COMM_SELF     ((MPI_Comm)0x44000001))
```

BAB III
METODOLOGI PENELITIAN

Metode penelitian yang digunakan dalam penelitian ini meliputi studi literatur, penentuan spesifikasi alat, desain dan implementasi algoritma genetika paralel, pengujian, penarikan kesimpulan dan saran.

3.1. Studi Literatur

Studi literatur yang dilaksanakan berupa kajian pustaka terhadap sumber-sumber bacaan yang relevan sehingga mampu menunjang dalam melakukan pencarian akar-akar persamaan fungsi kompleks menggunakan algoritma genetika paralel. Studi literatur yang diperlukan sebagai bahan acuan dalam melakukan penelitian ini, seperti mempelajari tentang fungsi kompleks, pencarian akar-akar persamaan fungsi kompleks, algoritma genetika, algoritma genetika paralel, komputasi paralel, dan teori – teori lain yang menunjang dalam penyusunan skripsi ini.

3.2. Penentuan Spesifikasi Alat

Perangkat yang digunakan untuk implementasi pencarian akar-akar persamaan fungsi kompleks menggunakan algoritma genetika paralel diuraikan sebagai berikut:

3.2.1. Perangkat Keras

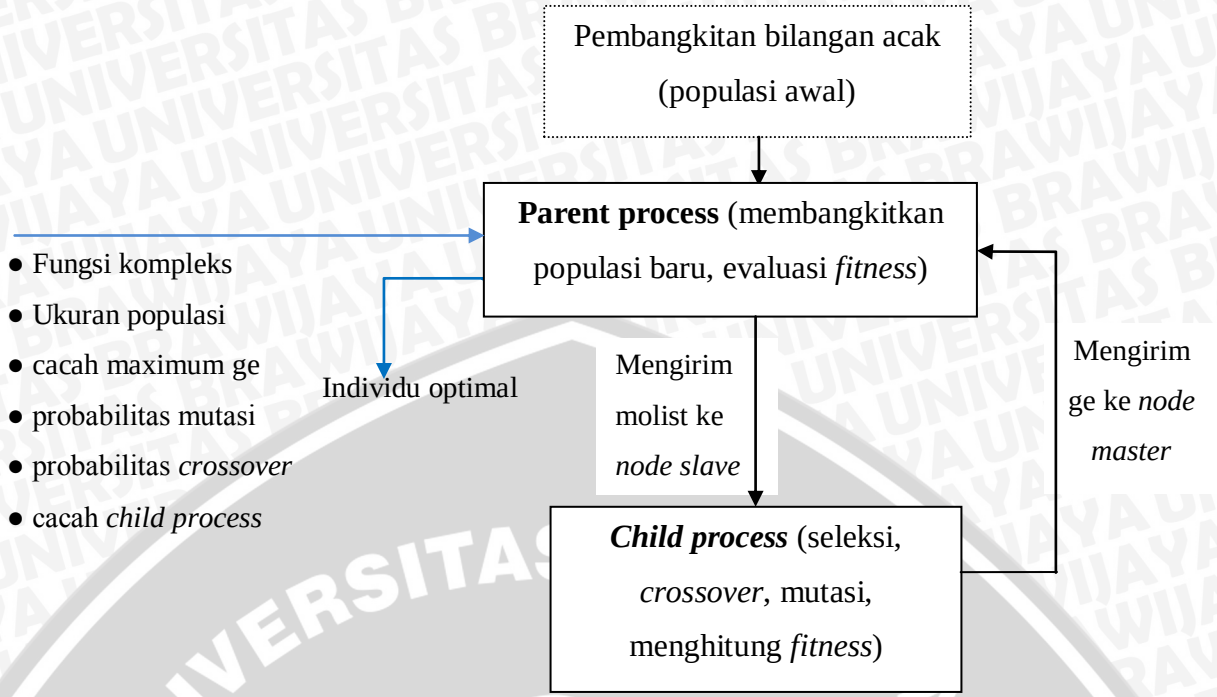
1) 4 unit komputer personal: Intel i3 540, DDR3 2 GB
2) 1 unit komputer personal: Intel Dual Core E5400, DDR2 2GB.
3) *switch* TL-SG1024D dan perkabelan UTP Cat 5e.

3.2.2. Perangkat Lunak

1) GNU/Linux Salix64 14.1: kernel 3.2.29, glibc 2.15, gcc 4.7.1,
2) GNU/Linux Salix64 14.0: gcc 4.7.1, glibc 2.15, dan kernel Linux 3.2.29 64 bit,
3) Open MPI 1.8.3
4) GAlib 2.4.7
5) GAlib-mpi

3.3. Abstraksi Sistem

Sistem komputasi paralel terdiri dari beberapa komponen yang dapat digambarkan dengan blok diagram sistem diperlihatkan pada Gambar 3.1.



Gambar 3.1. Abstraksi sistem komputasi paralel

Keterangan:

1) Populasi awal dibangkitkan secara acak dengan *pseudo random number generator*.

2) *Single-population master-slave Gas* hanya memiliki satu populasi, beberapa *node slave (child process)* mengeksekusi GA (seleksi, crossover, mutasi), dan evaluasi *fitness* dilakukan oleh satu *node master (father process).*

3) Individu optimal diperoleh dari perulangan evaluasi *fitness* hingga didapatkan nilai minimum dari $f(\rho)$ untuk ρ(radian)

3.4. Langkah Kerja Sistem

Akan diadopsi *Master-slave control network* PGA dan pola *node programming* (mode Single Program Multiple Data [SPMD]). Langkah-langkah Algoritma Genetika Paralel adalah sebagai berikut:

Langkah 1: Buat proses (proses ini adalah *parent process*, digunakan untuk menyimpan dan menghasilkan individu yang optimal saat ini, selama GA serial dilakukan).

Langkah 2: *Parent process* menciptakan beberapa *child process* (setiap *child process* digunakan untuk melakukan GA serial).

Langkah 3: Setiap *child process* (termasuk *parent process*) melakukan GA serial, saat *genetic generation* (ge) dari *child process* menjangkau generasi yang

diinginkan. (Disini ge dari *child process* adalah kelipatan 5), *child process* memberikan individu yang optimal pada *parent process*.

Langkah 4: *parent process* memilih yang terbaik dari individu yang optimal dari setiap *child process* (disebut molist), kemudian mengirimkannya ke setiap *child process*.

Langkah 5: Setiap *child process* mengganti individu yang memiliki kebugaran (fitness) rendah pada generasi saat ini dengan molist.

Langkah 6: jika ge=gmax (maka gmax sebagai ge maksimum) lalu pergi ke langkah (7), selain itu pergi ke langkah (3).

Langkah 7: Selesai.

3.5. Pengujian

Sistem tersebut setelah dirancang dan diimplementasikan, maka akan dilakukan pengujian dengan cara pembandingan kinerja waktu. Untuk mengetahui kecepatan eksekusi program akan digunakan stopwatch timer.

Misalkan terdapat dua contoh persamaan. Ambil rata-rata *runtime* dari sepuluh kali *runtime* sebagai *runtime* algoritma karena keacakan (*randomcity*) iterasi dari GA. Hasil runtime dari akselerasi paralel diketahui dari persamaan peningkatan kecepatan.

Peningkatan kecepatan (*speed up*) dapat diformulasikan dengan persamaan:

$$S_p = \frac{T_s}{T_p} \qquad (3\text{-}1)$$

dimana S_p adalah peningkatan kecepatan, T_s adalah waktu yang dibutuhkan untuk menyelesaikan pekerjaan (program komputer) bila dijalankan dalam satu komputer, sedangkan T_p adalah waktu yang dibutuhkan jika pekerjaan dikerjakan bersamaan oleh beberapa komputer.

Ukuran kontribusi cacah prosesor pada komputasi paralel adalah:

$$E_p = \frac{S_p}{p}; (0 > E_p < 1) \qquad (3\text{-}2)$$

dimana S_p adalah peningkatan kecepatan, p adalah cacah prosesor secara paralel. Nilai dari E_p adalah mendekati 1 yang mengindikasikan bahwa algoritma tersebut efisien.

3.6. Penarikan Kesimpulan

Pada tahap ini, kesimpulan dan analisis dari pengujian dipaparkan. Tahap selanjutnya adalah pembuatan saran untuk perbaikan dan pengembangan penelitian selanjutnya.

3.7. SISTEMATIKA PENULISAN

Sistematika penulisan yang digunakan dalam penyusunan skripsi ini adalah sebagai berikut :

BAB I : PENDAHULUAN

Berisi latar belakang, rumusan masalah, batasan masalah, tujuan, hipotesis, dan sistematika penulisan.

BAB II : TINJAUAN PUSTAKA

Berisi tinjauan pustaka atau dasar teori yang digunakan untuk dasar penelitian yang dilakukan dan untuk mendukung permasalahan yang diungkapkan

BAB III : METODOLOGI PENELITIAN

Memberikan penjelasan tentang metode yang digunakan dalam skripsi, meliputi penentuan spesifikasi alat, abstraksi sistem, dan langkah kerja sistem.

BAB IV : DESAIN DAN IMPLEMENTASI

Berisi rancangan konfigurasi, kompilasi dan instalasi sistem.

BAB V : PENGUJIAN

Berisi perbandingan performa dan efisiensi sistem antara komputasi paralel dan serial.

BAB V : PENUTUP

Berisi kesimpulan dan saran.

BAB IV
DESAIN DAN IMPLEMENTASI

Pada bab ini akan dibahas tentang desain dan implementasi algoritma genetika paralel pada permasalahan uji yang akan dikerjakan dalam empat tahap, yaitu tahap penentuan permasalahan uji spesifik, tahap desain algoritma genetika paralel, penyelesaian permasalahan uji dengan algoritma genetika paralel, tahap pengujian.

4.1. Permasalahan Uji

Untuk contoh permasalahan dipilih kasus khusus dimana nilai imajinernya selalu nol, agar memudahkan dalam analisis.

Bila diketahui suatu fungsi kompleks (dengan nilai imajinernya selalu nol):

$$f(z) = \cos(z) - \sin(z) = 0$$

perkiraan : $z = \rho(\cos\theta + i \sin\theta), \rho \in [0, 20], \quad \theta \in [0, 2\pi)$

presisi : $e1 = 0{,}00001$, $e2 = 0{,}00001$

error : solusi eksak – hasil

4.1.1. Solusi Eksak

Tentukan kombinasi $\rho \angle \tan\theta$ pada selang [0, 20] untuk ρ dan selang $(0, 2\pi)$ untuk θ yang meminimumkan fungsi kompleks $f(z) = \cos(z) - \sin(z)$, dengan $z = \rho \cos\theta + i\, \rho \sin\theta$. Jelaslah bahwa nilai minimum untuk fungsi $\cos(z) - \sin(z)$ adalah 0 (atau mendekati 0, bergantung pada presisi).

Jika dilihat secara grafik (Gambar 4.1 dan Gambar 4.2), maka terdapat tiga belas akar-akar persamaan (digambarkan dengan simbol lingkaran merah).

Secara eksak nilai z dapat dicari dengan formulasi sebagai berikut:

$f(z) = \cos(z) - \sin(z) = 0$

$\cos(z) - \sin(z) = 0$

$\cos(z) = \sin(z)$

$1 = \frac{\sin(z)}{\cos(z)}$

$1 = \tan(z)$

akar-akar persamaan pertama di dapatkan dengan cara:

$$z_1 = \tan^{-1}(1) = 45° = \frac{1}{4}\pi = 0{,}785398$$

Karena fungsi tersebut adalah fungsi periodik maka persamaannya dapat ditulis:

$$z_n = \tan^{-1}(1) + k\pi \qquad k = 0, \mp1, \mp2, \mp3 \ldots$$

akar-akar persamaan yang lain dapat dicari:

k = -1; $z_2 = \tan^{-1}(1) - 1\pi = -\frac{3\pi}{4} = -2{,}356194$

k = 1; $z_3 = \tan^{-1}(1) + 1\pi = \frac{5\pi}{4} = 3{,}926991$

k = -2; $z_4 = \tan^{-1}(1) - 2\pi = -\frac{7\pi}{4} = -5{,}497787$

k = 2; $z_5 = \tan^{-1}(1) + 2\pi = \frac{9\pi}{4} = 7{,}068583$

k = -3; $z_6 = \tan^{-1}(1) - 3\pi = -\frac{11\pi}{4} = -8{,}639380$

k = 3; $z_7 = \tan^{-1}(1) + 3\pi = \frac{13\pi}{4} = 10{,}210176$

k = -4; $z_8 = \tan^{-1}(1) - 4\pi = -\frac{15\pi}{4} = -11{,}780972$

k = 4; $z_9 = \tan^{-1}(1) + 4\pi = \frac{17\pi}{4} = 13{,}351769$

k = -5; $z_{10} = \tan^{-1}(1) - 5\pi = -\frac{19\pi}{4} = -14{,}922565$

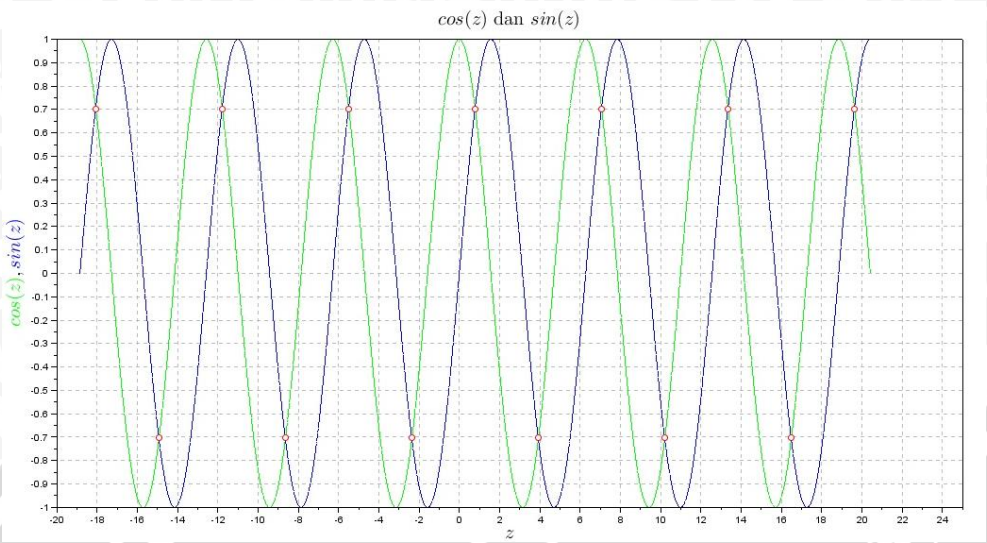
k = 5; $z_{11} = \tan^{-1}(1) + 5\pi = \frac{21\pi}{4} = 16{,}493361$

k = -6; $z_{12} = \tan^{-1}(1) - 6\pi = -\frac{23\pi}{4} = -18{,}064158$

k= 6; $z_{13} = \tan^{-1}(1) + 6\pi = \frac{25\pi}{4} = 19{,}634954$

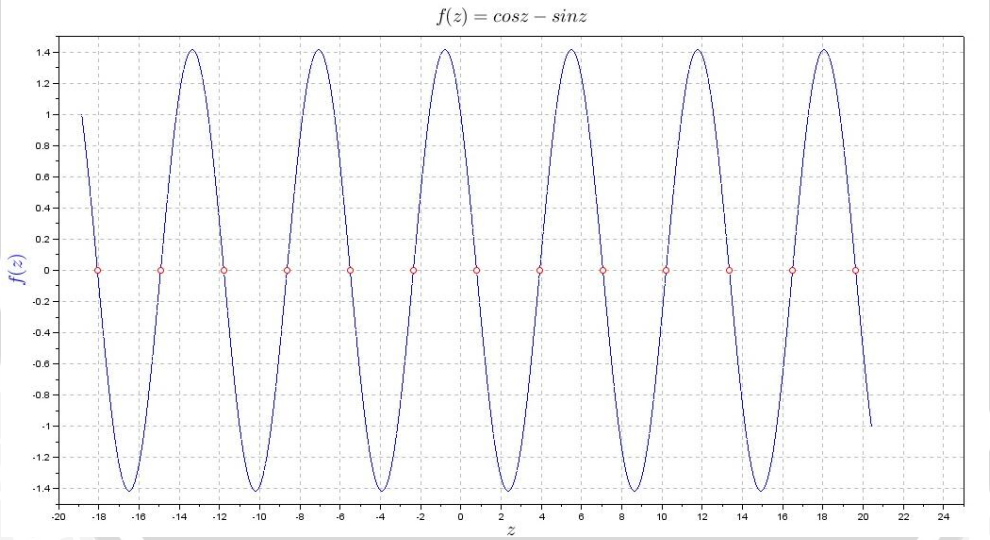
4.1.2. Grafik Fungsi

Secara grafik fungsi tersebut dapat digambarkan sebagai berikut:



Gambar 4.1. Fungsi cos *z* dan sin *z*

Perpotongan antara fungsi cos z dengan fungsi sin z adalah akar-akar persamaannya. Atau dapat juga digambarkan sebagai fungsi f(z) = cos z –sin z seperti gambar di bawah ini.



Gambar 4.2. Fungsi f(*z*) = cos *z* – sin *z*

4.1.3. Nilai ρ dan θ

Dari perhitungan eksak, maka dapat diketahui berapakah nilai ρ dan θ yang memenuhi akar-akar persamaan tersebut, karena bagian imajiner dari fungsi tersebut selalu nol dan akar-akar persamaannya tepat pada sumbu *y*:

$$(\rho \angle \tan\theta)_{\mathrm{n}} \rightarrow \rho\cos\theta + i\,\rho\sin\theta = \tan^{-1}(1) + n\pi$$

Untuk $(\rho \angle \tan\theta)_1$ maka dapat diketahui:

$$\rho\cos\theta + i\,\rho\sin\theta = \tan^{-1}(1) + n\pi$$

$$\frac{\pi}{4}\cos(0) + i\,\frac{\pi}{4}\sin(0) = \tan^{-1}(1) + 0 * \pi$$

Sehingga:

$\rho_1 = \frac{\pi}{4}$ dan $\theta_1=0$

$\rho_2 = \frac{3\pi}{4}$ dan $\theta_2=\pi$

$\rho_3 = \frac{5\pi}{4}$ dan $\theta_3=0$

$\rho_4 = \frac{7\pi}{4}$ dan $\theta_4=\pi$

$\rho_5 = \frac{9\pi}{4}$ dan $\theta_5=0$

$\rho_6 = \frac{11\pi}{4}$ dan $\theta_6=\pi$

$\rho_7 = \frac{13\pi}{4}$ dan $\theta_7=0$

$\rho_8 = \frac{15\pi}{4}$ dan $\theta_8=\pi$

$\rho_9 = \frac{17\pi}{4}$ dan $\theta_9=0$

$\rho_{10} = \frac{19\pi}{4}$ dan $\theta_{10}=\pi$

$\rho_{11} = \frac{21\pi}{4}$ dan $\theta_{11}=0$

$\rho_{12} = \frac{23\pi}{4}$ dan $\theta_{12}=\pi$

$\rho_{13} = \frac{25\pi}{4}$ dan $\theta_{13}=0$

4.2. Desain Algoritma Genetika

Terdapat delapan komponen yang harus ditentukan terlebih dahulu sebelum melakukan implementasi algoritma genetika, delapan komponen tersebut antara lain:

1. Skema Pengodean
2. Nilai *Fitness*
3. Seleksi Orang Tua
4. Skema *Crossover*
5. Skema Mutasi
6. Elitisme
7. Penggantian Populasi

8. Kriteria Terminasi

4.2.1. Populasi Awal

Untuk memperkirakan langkah pertama pada metode algoritma genetika, akan ditentukan masalah sederhana dalam menentukan nilai minimum dari fungsi berikut:

$f(z) = \cos(z) - \sin(z) = 0$

$z = \rho(\cos\theta + i \sin\theta),\ \rho \in [0, 20],\quad \theta \in [0, 2\pi)$

atau

$f(\rho,\theta) = \cos[\rho(\cos\theta + i \sin\theta)] - \sin[\rho(\cos\theta + i \sin\theta)] = 0$

$f(\rho,\theta) = \cos(\rho \cos\theta + i\,\rho \sin\theta) - \sin(\rho \cos\theta + i\,\rho \sin\theta) = 0$

di atas domain *float* 0 sampai 20 untuk variabel ρ dan 0 sampai 2π untuk variabel θ.

Dengan menggunakan aljabar sederhana. f dapat difaktorkan ke dalam bentuk di mana nilai minimum dapat diketahui dari pemeriksaan. Pada bentuk ini, jelas terlihat bahwa minimum dari pertama dari $f(\rho,\theta)$, yaitu 0, muncul ketika $\rho_1 = \frac{\pi}{4}$ dan $\theta_1=0$.

Dalam kaitan dengan ukuran perhitungan masalah, pendekatan pencarian "secara nyata dan seketika" yang mendalam yang hanya menghitung nilai fungsi yang sesuai dengan setiap kemungkinan nilai adalah hal yang praktis. Terdapat lebih dari $1{,}5 \times 10^{12}$ (0 hingga 20 dengan presisi $1/10^5$ dikalikan 0 hingga 2π dengan presisi $1/10^5$) kombinasi (ρ,θ) yang harus dievaluasi, dan walaupun nilai fungsi yang sesuai dengan setiap kombinasi (ρ,θ) dapat dihitung, tetapi diperlukan waktu yang lama pada sebuah prosesor tunggal.

4.2.2. Representasi Data

Langkah pertama dalam menentukan populasi awal dari solusi potensial adalah dengan memilih representasi data yang sesuai dengan setiap individu. Agar sederhana maka akan digunakan *string biner*.

Solusi potensial ke-i terdiri atas dua variabel (ρ_i, θ_i) dimana komponen ρ_i memiliki satu dari 2.000.001 *float* yang mungkin pada interval $0 \leq komponen_{nilai} \leq 20$. Dan komponen θ_i memiliki satu dari 628.319 *float* yang mungkin pada interval $0 \leq komponen_{nilai} \leq 2\pi$. maka panjang kromosom L dapat dirumuskan sebagai:

$$L = \log_2((b-a)10^n + 1)$$

Untuk variabel ρ, *parent process* akan mendistribusikan individu berdasarkan interval

$\text{interval}_1 = (0, \frac{1}{2}\pi)$ atau (0; 1,57080)

$\text{interval}_2 = (\frac{1}{2}\pi, \pi)$ atau (1,57080; 3,14159)

$\text{interval}_3 = (\pi, \frac{3}{2}\pi)$ atau (3,14159; 4,71239)

$\text{interval}_4 = (\frac{3}{2}\pi, 2\pi)$ atau (4,71239; 6,28319)

$\text{interval}_5 = (2\pi, \frac{5}{2}\pi)$ atau (6,28319; 7,85398)

$\text{interval}_6 = (\frac{5}{2}\pi, 3\pi)$ atau (7,85398; 9,42478)

$\text{interval}_7 = (3\pi, \frac{7}{2}\pi)$ atau (9,42478; 10,99557)

$\text{interval}_8 = (\frac{7}{2}\pi, 4\pi)$ atau (10,99557; 12,56637)

$\text{interval}_9 = (4\pi, \frac{9}{2}\pi)$ atau (12,56637; 14,13717)

$\text{interval}_{10} = (\frac{9}{2}\pi, 5\pi)$ atau (14,13717; 15,70796)

$\text{interval}_{11} = (5\pi, \frac{11}{2}\pi)$ atau (15,70796; 17,27876)

$\text{interval}_{12} = (\frac{11}{2}\pi, 6\pi)$ atau (17,27876; 18,84956)

$\text{interval}_{13} = (6\pi, \frac{13}{2}\pi)$ atau (18,84956; 20,42035)

Sehingga panjang kromosom L_{ρ_1} untuk ρ_1 pada selang $[0, \frac{1}{2}\pi]$ dengan presisi n = 5 adalah:

$$L_{\rho_1} = \log_2((\frac{\pi}{2} - 0)10^5 + 1)$$
$$= \log_2(157080{,}6327)$$
$$= 17{,}26115$$
$$= 17 \text{ (dibulatkan)}$$

L_{θ_1} untuk θ_1 pada selang $[0, 2\pi]$ dengan presisi n = 5 adalah:

$$L_{\theta_1} = \log_2((2\pi - 0)10^5 + 1)$$
$$= 19{,}26114$$
$$= 19 \text{ (dibulatkan)}$$

Kromosom *v* adalah gabungan dari dua string biner (ρ_i ,θ_i). Misalnya $\rho_1 = \frac{\pi}{4}$ dan θ_1=0. Dengan menggunakan representasi besaran, representasi biner untuk setiap angka adalah

ρ_1 = 01111111111111111

θ_1= 0000000000000000000

Sedangkan nilai ρ_1 dapat dirumuskan sebagai:

$$\rho_i = r_b + (r_a - r_b) \sum_{k=1}^{n} g_k \times 2^{-k}$$

atau

$$\rho_i = r_b + (r_a - r_b)(g_1 2^{-1} + g_2 2^{-2} + g_3 2^{-3} + \ldots + g_n 2^{-n})$$

Maka

$$\rho_1 = 0 + (1{,}57080 - 0)((0)2^{-1} + (1)2^{-2} + (1)2^{-3} + (1)2^{-4} + (1)2^{-5} + (1)2^{-6} + (1)2^{-7} + (1)2^{-8} + (1)2^{-9} + (1)2^{-10} + (1)2^{-11} + (1)2^{-12} + (1)2^{-13} + (1)2^{-14} + (1)2^{-15} + (1)2^{-16} + (1)2^{-17})$$

$$\rho_1 = (1{,}57080)(0{,}5) = 0{,}78539$$

dan nilai θ_1 dapat dirumuskan sebagai:

$$\theta_1 = 0 + (6{,}28319 - 0)((0)2^{-1} + (0)2^{-2} + (0)2^{-3} + (0)2^{-4} + (0)2^{-5} + (0)2^{-6} + (0)2^{-7} + (0)2^{-8} + (0)2^{-9} + (0)2^{-10} + (0)2^{-11} + (0)2^{-12} + (0)2^{-13} + (0)2^{-14} + (0)2^{-15} + (0)2^{-16} + (0)2^{-17} + (0)2^{-18} + (0)2^{-19})$$

$$\theta_1 = (6{,}28319)(0) = 0$$

4.2.3. Nilai *fitness*

$f(\rho_1,\theta_1) = \cos(\rho_1 \cos\theta_1 + i\,\rho_1 \sin\theta_1) - \sin(\rho_1 \cos\theta_1 + i\,\rho_1 \sin\theta_1) = 0$

$f(0{,}78539;0) = \cos(0{,}78539 \cos 0 + i\ 0{,}78540 \sin 0) - \sin(0{,}78539 \cos 0 + i\ 0{,}78540 \sin 0)$

$f(0{,}78539;0) = \cos(0{,}78539 + 0i) - \sin(0{,}78539 + 0i)$

$f(0{,}78539;0) = 0{,}00001$

error = 0,00001

4.2.4. Stringbiner$_k$

ρ_1 = 01111111111111111

θ_1= 0000000000000000000

tidak ada nilai negatif dari setiap solusi potensial ρ dan θ, jadi tidak ada penanda. Seluruh bit menunjukkan besarnya. Pilihan ini menjamin bahwa besaran tersebut harus berada pada 0 dan 20 untuk ρ, 0 dan 2π (6,28319) untuk θ. Dengan menggabungkan ke dalam string biner, didapatkan representasi dari solusi potensial v_1

v_1 = 01111111111111111000000000000000000

Sekarang sudah dapat menentukan panjang solusi potensial, Lv=36 bit, maka dapat dengan mudah mengenali sebuah populasi awal dengan menggunakan *pseudo random generator* (pembangkit bilangan acak palsu).

4.2.5. Penentuan Parameter Terminasi

Yang disebut dengan parameter di sini adalah parameter kontrol algoritma genetika, yaitu: ukuran populasi (popsize), peluang *crossover* (P_c), dan peluang mutasi (P_m). Nilai parameter ini ditentukan juga berdasarkan permasalahan yang akan dipecahkan. Ada beberapa rekomendasi yang bisa digunakan, antara lain:

- Untuk permasalahan yang memiliki kawasan solusi cukup besar, De Jong merekomendasikan untuk nilai parameter kontrol:

  (popsize; P_c; P_m) = (50; 0,6; 0,001).

- Bila rata-rata *fitness* setiap generasi digunakan sebagai indikator, maka Grefenstette merekomendasikan:

  (popsize; P_c; P_m) = (30; 0,95; 0,01).

- Bila fitness dari individu terbaik dipantau pada setiap generasi, maka usulannya adalah:

  (popsize; P_c; P_m) = (80; 0,45; 0,01).

- Ukuran populasi sebaiknya tidak lebih kecil dari 30, untuk sembarang jenis permasalahan.

Maka, dapat dilakukan pemilihan parameter untuk penelitian ini didasarkan pada rekomendasi dan percobaan. Parameter yang akan digunakan, yaitu antara lain:

popsize = 50

P_c = 0,92

P_m = 0,02

maksimum generasi = 100

4.2.6. Inisialisasi Populasi Awal

Populasi awal dipilih secara acak dengan PRNG (*pseudo random number generator*). PRNG yang digunakan adalah `rand()`.

4.2.7. Seleksi Kromosom Induk dengan Roda Roulette

Langkah-langkah yang harus dikerjakan:

1. Mencari Total *fitness*, *fitness* relatif (p_k) tiap-tiap kromosom dapat dicari dengan persamaan

$p_k = F_k/\text{TotalFitness}$

$p_1 = F_1/\text{TotalFitness}$

$p_2 = F_2/\text{TotalFitness}$ dst.

2. *Fitness* kumulatif (q_k) dapat dicari dengan persamaan

$q_1 = p_1$

$q_2 = q_1 + p_1$

$q_3 = q_2 + p_3$ dst.

3. Bangkitkan bilangan acak r_k sebanyak popsize=50. Lalu bandingkan r_k dengan q_k. Misalkan $r_1 > q_{16}$ dan $r_1 < q_{17}$. Maka v_1 akan terpilih menjadi kromosom baru yang pertama. dst.

4.2.8. *Crossover*

langkah-langkah yang harus dikerjakan:

1) Karena peluang *crossover* (p_c) adalah 0,92; maka diharapkan 92% dari total kromosom akan mengalami *crossover* (45 dari 50 kromosom). Untuk memilih kromosom mana saja yang akan dilakukan *crossover*, bangkitkan bilangan acak antara 0 dan 1 sebanyak 45 buah.

2) Pilih bilangan-bilangan acak yang kurang dari P_c (misalkan r_s), maka kromosom v_s' berhak untuk melakukan *crossover*. Misal bilangan acak pertama r_1=0,98; bilangan ini lebih besar jika dibandingkan dengan p_c (0,92). Dengan demikian kromosom v_1' tidak akan mengalami *crossover*. Misalkan bilangan acak yang kurang dari p_c adalah bilangan ke-2,3,4,6,7,8,9,11,12,13,14,16,17,18,19 dst. Hal ini berarti bahwa kromosom v_2' ,v_3', v_4', v_6', v_7', v_8', v_9', v_{11}', v_{12}', v_{13}', v_{14}', v_{15}', v_{16}', v_{17}', v_{18}', v_{19}' dst. Karena jumlah kromosom yang terpilih ganjil (45), maka harus dibuang salah satu, misalkan kromosom v_{45}' dibuang.

3) Silangkan (v_2' dengan v_3'), (v_4' dengan v_6'), (v_7' dengan v_8'), (v_9' dengan v_{11}'), (v_{12}' dengan v_{13}') dst. Cara *crossover* (misalnya pada v_{18}' dengan v_{19}'): Pilih bilangan acak antara 1 sampai (Lv-1)[1 sampai 36]. Bilangan ini akan menentukan posisi crossover satu titik. Misalkan bilangan itu adalah 23.

v_{18}' = 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 | 0 1 0 1 0 1 0 0 1 0 0 1

v_{19}' = 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 | 0 0 0 0 0 0 0 0 0 0 0 0

Anak (*offspring*) sebagai hasil crossover (v_{18}'' dan v_{19}'') adalah:

v_{18}'' = 0 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1 0 1 | 0 1 0 0 0 0 0 0 0 0 0 0

v_{19}'' = 1 0 1 1 0 1 1 1 1 1 1 1 0 0 1 1 1 0 0 0 0 | 0 0 0 1 0 1 0 0 1 0 0 1

Pemilihan posisi penyilangan dengan mengambil bilangan acak, kemudian dilanjutkan dengan penyilangan. Ini juga dilakukan untuk semua padangan kromosom yang akan disilangkan.

4.2.9. Mutasi

Langkah-langkah yang harus dikerjakan:

1) Hitung jumlah bit yang ada pada populasi, yaitu: popsize * Lv = 50 * 36 = 1800

2) Karena peluang mutasi (p_m) adalah 0,02; maka diharapkan 2% dari total bit akan mengalami mutasi (36 bit). Untuk memilih bit-bit mana saja yang akan dilakukan mutasi, bangkitkan bilangan acak antara 0 dan 1 sebanyak 1800.

3) Misalnya pada kromosom pertama, bit pertama, bilangan acak yang terbentuk 0,592. Bilangan ini lebih besar jika dibandingkan dengan p_m = 0,02; ini berarti bahwa kromosom pertama, bit pertama, tidak akan terkena mutasi. Pada kromosom kedua, bit kedua, bilangan acak yang terbentuk adalah 0,007. Bilangan ini lebih kecil jika dibandingkan dengan p_m = 0,02; ini berarti bahwa kromosom kedua, bit kedua, akan terkena mutasi.

4.2.10. Elitisme

Seperti diketahui bahwa metode seleksi dalam algoritma genetika dilakukan secara random, sehingga ada kemungkinan bahwa kromosom yang sebenarnya sudah baik tidak bisa turut serta pada generasi berikutnya karena tidak lolos seleksi. Untuk itu perlu kiranya ada pelestarian kromosom-kromosom terbaik, sehingga kromosom-kromosom yang sudah baik tersebut bisa lolos seleksi. Muhlenbein mengusulkan adanya perbaikan pada algoritma genetika yang dikenal dengan *Breeder* Gas (BGA). Pada BGA ini digunakan parameter *r*, yang menunjukan kromosom terbaik. Kromosom ini akan tetap dipertahankan pada generasi berikutnya dengan cara menggantikan sebanyak *r* kromosom pada generasi tersebut secara acak.

Sehingga apabila pelestarian kromosom dilakukan:

1) Misalkan probabilitas kromosom terbaik yang akan dilestarikan adalah 0,2 yang berarti bahwa paling tidak 20% kromosom dalam populasi yang telah terjadi (10 kromosom dari 50 kromosom) akan digantikan dengan kromosom terbaik pada populasi awal generasi yang bersangutan.

b. Bangkitkan 50 bilangan random. Misalkan bilangan acak yang kurang dari 0,2 adalah bilangan acak ke: 3, 17, 39, 42. Sehingga kromosom-kromosom pada nomor tersebut akan digantikan dengan kromosom-kromosom terbaik pada populasi awal.

4.2.11. Penggantian Generasi (*Generational Replacement*)

Populasi akhir setelah dilakukan mutasi ini nanti akan dijadikan sebagai populasi awal untuk generasi kedua.

Dengan cara yang sama proses seleksi, *crossover*, mutasi, dan penggantian generasi dilakukan pada generasi kedua sampai ke-100.

4.3. Ikhtisar GAlib

Implementasi GA yang digunakan adalah SimpleGA (*non-overlapping*) pada standar GAlib. Ketika menggunakan *library* GAlib dua kelas yang utama adalah: *Genome* dan *Genetic Algorithm*. Setiap genome merupakan solusi tunggal untuk suatu masalah. *genetic algorithm object* mendefinisikan bagaimana evolusi harus dilakukan. GA menggunakan fungsi tujuan (*objective function*) yang ditentukan sendiri untuk menentukan seberapa '*fit*' tiap genome untuk bertahan

hidup. Penggunaan genome operator (sudah termasuk [*built in*] di dalam genome) dan strategi seleksi/penggantian (sudah termasuk [*built in*] di dalam genome) untuk membangkitkan individu baru. Ada tiga hal yang harus dilakukan sebelum menggunakan GAlib: mendefinisikan representasi, mendefinisikan operator genetik, mendefinisikan fungsi tujuan.

Pada dua item pertama GAlib sangat membantu dengan memberikan banyak contoh dan potongan program, dari sana pengguna dapat membangun representasi dan operator. Pengguna dapat menggunakan *built-in* representasi dan operator dengan sedikit atau tanpa modifikasi.

Template sudah tersedia yang digunakan di beberapa kelas genom, tetapi jika compiler tidak memahami *template* tersebut, GAlib dapat digunakan tanpa template.

Pada GAlib, contoh struktur data genome disebut GAGenome (sebutan lainnya adalah kromosom). Pada penelitian ini akan digunakan GABin2DecPhenotype.

Ada beberapa macam tipe GA. GAlib menyediakan tiga tipe dasar: *simple*, *steady – state*, dan *incremental*. Ketiga algoritma ini berbeda dalam hal cara membuat individu baru dan mengganti individu tua selama evolusi. Pada penelitian ini akan digunakan tipe *simple* GA.

Untuk membuat genome melalui fungsi tujuan, maka kode yang digunakan adalah: GABin2DecGenome genome(pheno, objective); Untuk membuat GA menggunakan genome, maka kode yang digunakan adalah: GASimpleGA ga(genome); Untuk menjalankan GA, maka kode yang digunakan adalah: ga.evolve (seed);.

4.4. Paralelisasi Algoritma Genetika

Secara konsep, masalah untuk menyesuaikan algoritma genetika untuk beberapa prosesor sudahlah jelas. Dua pendekatan yang terpikirkan langsung adalah:

- Setiap prosesor beroperasi secara independen pada sebuah subpopulasi individu yang terisolasi, secara periodik membagi individu terbaiknya dengan prosesor-prosesor lain melalui *migrasi*, atau

- Setiap prosesor melakukan satu bagian dari setiap langkah dalam algoritma – seleksi, penyilangan dan mutasi – pada populasi yang biasa / umum.

Akan dipilih pendekatan pertama, karena sesuai dengan struktur SPMD.

4.4.1. Subpopulasi yang Terisolasi

Paralelisme yang terisolasi seperti itu pernah digunakan untuk mencoba menjelaskan (secara jelas) mata rantai yang hilang pada catatan fosil. Menurut teori *punctuated equilibria* yang memberikan sebuah fondasi untuk teori yang menyatakan bahwa spesies-spesies baru cenderung berubah bentuk secara cepat dalam subpopulasi karena mengikuti perubahan di dalam lingkungan.

Pada pendekatan ini, setiap prosesor secara independen menghasilkan subpopulasi individu awal. Setiap prosesor kemudian membawa generasi *k* dari individu-individu:

- Penanganan evaluasi kesesuaian
- Memilih individu-individu dari subpopulasi untuk digunakan menghasilkan generasi berikutnya
- Melakukan perhitungan penyilangan dan mutasi pada subpopulasinya

Setelah generasi seperti *k* ($k \leq 1$), prosesor-prosesor membagi individu terbaik mereka dengan prosesor *master*.

4.5. *Embarrasingly Parallel* pada Algoritma Genetika

Embarrasingly parallel memiliki kelebihan dan juga kelemahan: alam relatif terisolasi dari subkumpulan populasi. Isolasi/pengasingan dari subpopulasi meningkatkan kemungkinan bahwa setiap subkumpulan akan bergabung di jumlah maksimum lokal dibandingkan jumlah maksimum global dan barangkali tidak mengenai solusi global. Sebagai tambahan, jumlah individu yang dikurangi membuat setiap subkumpulan populasi meningkatkan jumlah generasi yang diperlukan untuk mendapatkan konvergensi.

Di samping itu, fenomena isolasi *local-optima* dalam algoritma genetika dipercaya telah terjadi pula di alam. Para ahli geologi mempelajari lempeng tektonik (bagian besar dari kerak bumi yang bergerak di sekitarnya, di atasnya, dan ke satu sama lain) telah menghasilkan proyeksi komputer terhadap pergerakan lempeng sebelumnya. Salah satu lempeng yang telah lama terisolasi dari

hubungan satu sama lain telah membentuk sebagian benua Australia. Para ahli biologi telah lama mencatat jumlah spesies yang unik di wilayah tertentu, termasuk *wingless birds* dan *large hopping mamals*. Evolusi muncul secara paralel pada wilyah yang terisolasi, solusi yang dihasilkan tidak selalu sama di setiap wilayah. Malahan, hasilnya mungkin adalah jumlah maksimum lokal yang berbeda.

Mekanisme untuk mengurangi isolasi yang efektif telah meningkatkan komunikasi antar prosesor, namun juga memperkenalkan kelemahan. Secara bertahap ada peningkatan komunikasi algoritma dan ada penurunan percepatan efektif pada kasus dimana digunakan banyak prosesor namun prosesor terisolasi.

4.6. Konfigurasi Perangkat Keras

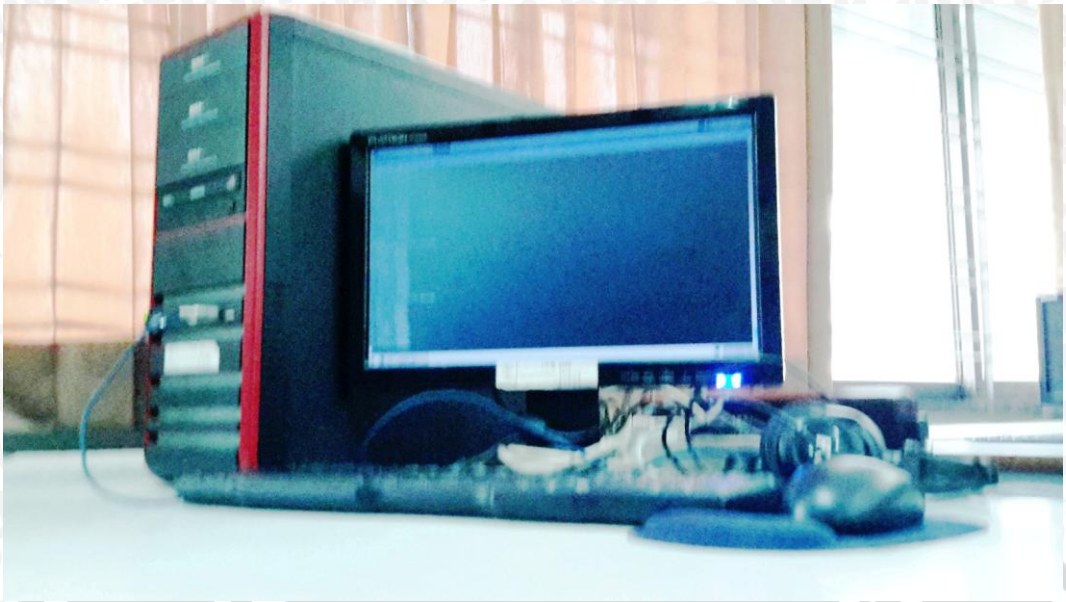
Sistem komputasi paralel menggunakan 5 komputer: 1 *front-end*, 4 komputer slave. Jaringan area lokal sistem komputasi paralel menggunakan *switch* TL-SG1024D dengan perkabelan Cat 5e.

4.6.1. Konfigurasi Perangkat Keras Komputer *Front-end*

Komputer *front-end* adalah komputer tunggal yang digunakan sebagai antarmuka pengguna dan sistem komputasi paralel.

Tabel 4.1 Perangkat Keras Komputer *Front-end*

Prosesor	Intel Pentium E7400 (2M Cache, 2.70 GHz, 800 MHz FSB)
Board	BIOSTAR G31-M7 TE
Memori	2 x 1GB DDR2 6400
Storage	Seagate ST3500414CS Seagate ST3250318AS
NIC	Realtek Semiconductor Co., Ltd. RTL8101E/RTL8102E PCI Express Fast Ethernet controller (rev 02)



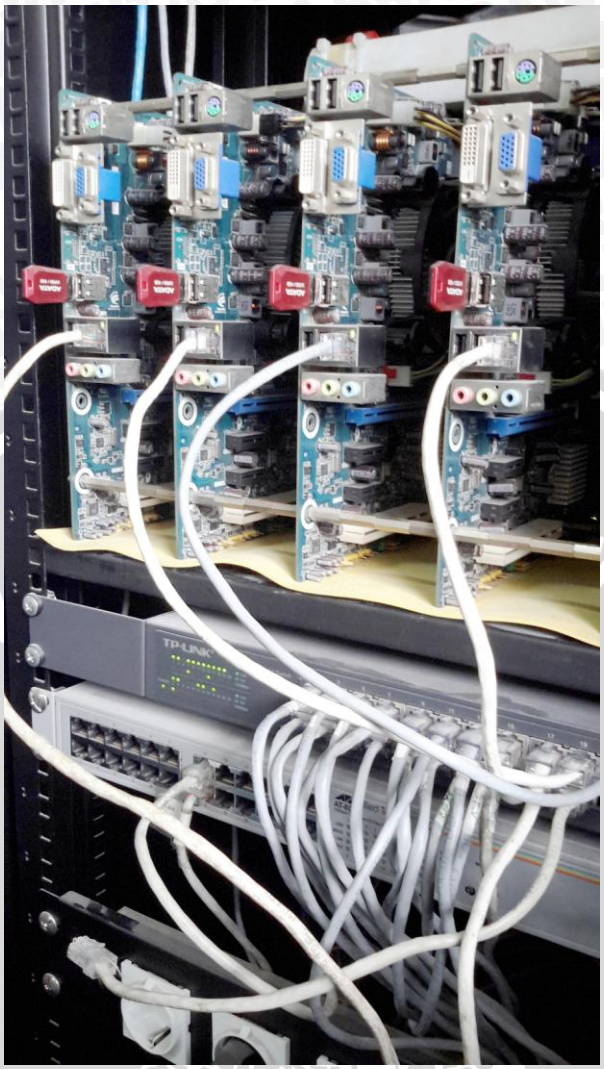
Gambar 4.3. Unit Komputer *Front-end*

4.6.2. Konfigurasi Perangkat Keras Komputer *Slave*

Komputer yang digunakan sebagai komputer *slave* berjumlah 4 unit dan digunakan untuk semua tahap pengujian, 4 unit menggunakan prosesor Intel Core i3-550. Media *boot* yang digunakan adalah flashdrive dengan ukuran nominal 4 GB.

Tabel 4.2 Perangkat Keras Komputer *Slave*

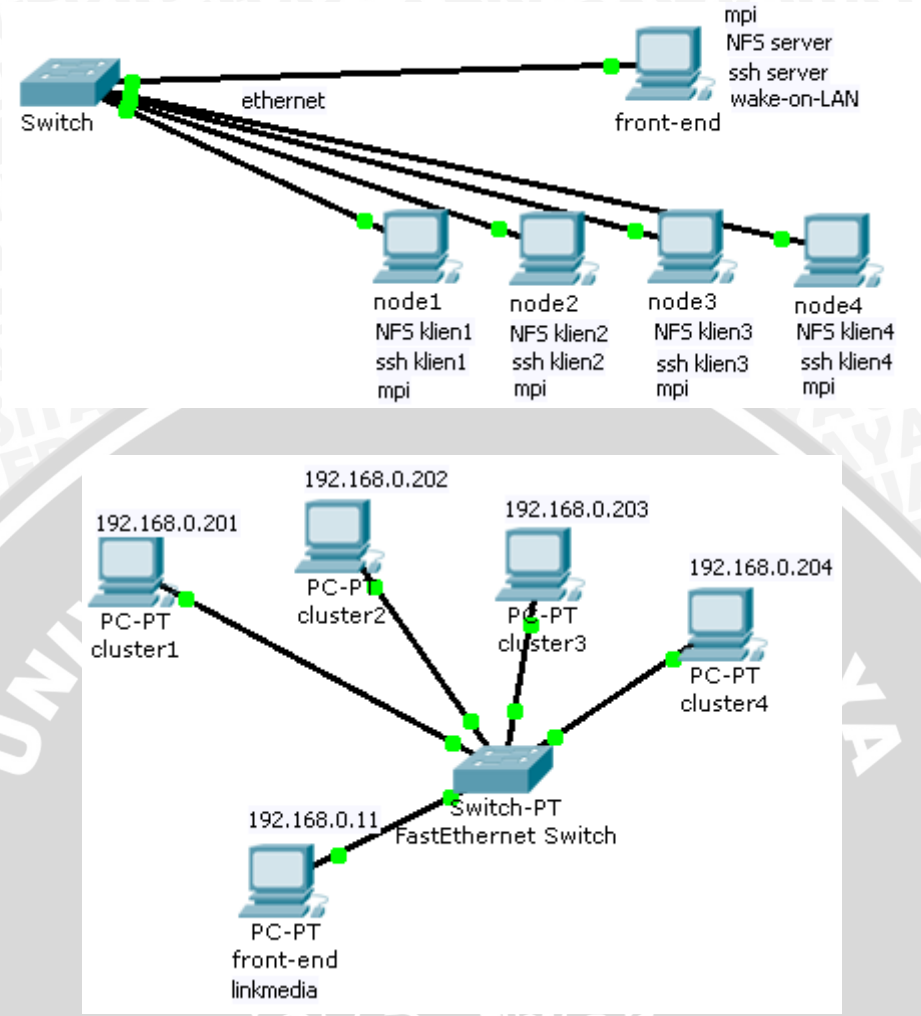
Prosesor	Intel Core i3 550 (4M Cache, 3.20 GHz)
Board	Intel Desktop Board DH55PJ
Memori	2GB DDR3 10600
Storage	ADATA DashDrive UV100 4GB
NIC	Intel Corporation 82578DC Gigabit Network Connection (rev 06)



Gambar 4.4. Empat Unit Komputer *Slave*

4.6.3. Konfigurasi Jaringan Lokal

Jaringan sistem komputasi paralel menggunakan jaringan lokal dengan topologi bintang. NIC dari tiap komputer terhubung langsung dengan *switch* TL-SG1024D melalui kabel UTP dengan kategori 5e. Konfigurasi pengalamatan jaringan lokal menggunakan IPv4 dengan lingkup jaringan 192.168.0.0/24. Gambar 4.5. memperlihatkan topologi jaringan, asosasi nama *host* dan alamat IP.



Gambar 4.5. Jaringan lokal Komputer Paralel

4.7. Konfigurasi Perangkat Lunak

Konfigurasi perangkat lunak sistem komputasi paralel melingkupi konfigurasi jaringan, *environment* sistem operasi, *filesystem* terdistribusi, kompilasi dan instalasi paket program dependensi, dan implementasi program komputasi paralel.

4.7.1. Konfigurasi Komputer *Front-end* (E7400)

Komputer *front-end* bertanggung jawab atas distribusi berkas *executable* program melalui *filesystem* terdistribusi (NFS), kompilasi (mpicc) dan inisialisasi eksekusi (mpirun) program komputasi paralel.

4.7.1.1. Konfigurasi Alamat IP dan Host Komputer *Front-end*

Front-end menggunakan sistem operasi Salix64 14.1 dengan gcc 4.8.2, glibc 2.17, dan kernel Linux 3.10.17 64-bit. Paket-paket dependensi: gcc, nfsutils, openssh merupakan telah terpasang sejak instalasi *default.*

Front-end menggunakan nama *host* linkmedia dengan alamat IP 192.168.0.11. Sistem operasi Salix64 menggunakan berkas `/etc/hosts` untuk mengasosiasikan nama *host* dan alamat IP. Semua komputer dalam sistem komputasi paralel menggunakan berkas `/etc/hosts` yang berisi semua entri semua *hosts* dalam sistem komputasi paralel agar dapat mengenali *hosts* lainnya.

Pengaturan alamat IP *front-end* dilakukan dengan menambahkan entri:

```
IPADDR[0]="192.168.0.11"
NETMASK[0]="255.255.255.0"
```

ke berkas `/etc/rc.d/rc.inet1.conf`.op

Pengaturan nama *host* dilakukan dengan penambahan berkas `/etc/HOSTNAME` dengan entri: `linkmedia.link.elektro.ub.ac.id`

4.7.1.2. Konfigurasi User `mpi`

Sebuah *user* identik diperlukan untuk menjalankan Open MPI di beberapa *user* sekaligus, hal ini mengharuskan adanya instans *user* dengan UID (*user id)* dan GID (*group id*) yang identik di semua *hosts* sistem komputasi paralel. Untuk menambahkan *user* dengan nama mpi dan sandi lewat mpi, berkas `/etc/passwd` ditambahkan entri:

```
mpi:$5$CUgEJj9IK3$/txgiLKoRWmB2VOVQ
/Sh4OsxEMIR0qnYByJcXOtJoK1:2000:2000::/home/mpi:/bin/bash
```

dan berkas /etc/group ditambahkan entri:

```
mpi:x:2000:
```

User mpi perlu memilki direktori *home* agar berbagai berkas dapat disimpan. Konfigurasi yang dilakukan:

```
# mkdir /home/mpi
# chmod 777 /home/mpi
# chown mpi:mpi /home/mpi
```

4.7.1.3. Konfigurasi Server NFS

Distribusi kunci publik SSH dan program komputasi paralel menggunakan NFS (Network File System) sebagai media. Direktori yang di-*export* melalui NFS akan dapat di-*mount* di *node*.

Penambahan direktori *home user* mpi `(/home/mpi)` dengan NFS di *front-end* yang bertindak sebagai server dilakukan dengan pengubahan berkas `/etc/exports` dengan entri:

```
/home/mpi 192.168.0.0/255.255.255.0(rw)
```

Pengaktifan *service* NFS dilakukan dengan cara memberikan moderasi *executable* terhadap berkas `/etc/rc.d/rc.rpc` dan `/etc/rc.d/rc.nfsd`:

```
# chmod +x /etc/rc.d/rc.rpc
# chmod +x /etc/rc.d/rc.nfsd
```

4.7.1.4. Konfigurasi SSH

Eksekusi Open MPI (mpirun) membutuhkan login *remote* dengan SSH di *node* tanpa *prompt* sandi lewat. Hal ini dilakukan dengan penambahan entri di berkas `/home/mpi/.ssh/authrorized_keys` dengan kunci publik *user* mpi di *front-end*.

```
$ cp ~/.ssh/id_ecdsa.pub ~/.ssh/authorized_keys
```

4.7.1.5. Kompilasi dan Instalasi OpenMPI Komputer Front-end

Program Open MPI harus dikompilasi dan dipasang pada *path* yang sesuai di semua komputer sistem komputasi paralel. Langkah konfigurasi dan instalasi adalah sebagai berikut:

```
# wget http://www.open-
  mpi.org/software/ompi/v1.8/downloads/openmpi-
  1.8.3.tar.bz2
# tar -jxvf openmpi-1.8.3.tar.bz2 -C /tmp
# cd /tmp/openmpi-1.8.3
# ./configure && make && make install
# ldconfig
```

Proses instalasi telah berhasil jika mpirun dan mpicc dikenali dalam *path* sistem dan pustaka yang berkaitan berhasil dimuat.

4.7.2. Konfigurasi Komputer *Slave* (4 Unit i3 550)

Komputer *slave* adalah komputer yang digunakan di semua tahap pengujian sistem komputasi paralel, menggunakan alamat IP 192.168.0.201-192.168.0.204, mendapatkan berkas *executable* dan kunci publik SSH melalui direktori NFS. Sistem operasi yang digunakan adalah Salix64 14.0 dengan gcc 4.7.1, glibc 2.15, dan kernel Linux 3.2.29. Paket nfsutils dan openssh telah terpasang sejak instalasi *default*.

Komputer *slave* dikonfigurasi *headless* (tanpa layar dan perangkat I/O) sehingga penyalaan dilakukan dengan paket *wake-on-LAN*.

4.7.2.1. Konfigurasi Alamat IP dan Host Komputer Slave

Pengaturan alamat IP *node* dan nama *host* sama seperti pengaturan alamat IP dan nama *host front-end*.

Tabel 4.3 Asosiasi Nama *Host* dan Alamat IP Komputer *Slave*

Nama *Host*	Alamat IP
cluster1	192.168.0.201
cluster2	192.168.0.202
cluster3	192.168.0.203
cluster4	192.168.0.204

4.7.2.2. Konfigurasi User `mpi`

Konfigurasi *user* mpi di komputer *slave* sama dengan konfigurasi *user* mpi di *front-end*.

4.7.2.3. Konfigurasi Klien NFS

Komputer slave melakukan mounting direktori NFS dari *front-end* secara otomatis tiap *boot* dengan menambahkan entri ke berkas `/etc/fstab`:

```
192.168.0.11:/home/mpi /home/mpi nfs sloppy,rw 0 0
```

4.7.2.4. Konfigurasi SSH

Komputer *slave* mendapatkan kunci publik SSH secara otomatis setelah direktori home user mpi di-mount dan konfigurasi SSH di *front-end* telah dilakukan.

4.7.2.5. Instalasi OpenMPI Komputer Slave

Langkah konfigurasi dan instalasi adalah sebagai berikut:

```
# wget http://www.open-
  mpi.org/software/ompi/v1.8/downloads/openmpi-
  1.8.3.tar.bz2
# tar -jxvf openmpi-1.8.3.tar.bz2 -C /tmp
# cd /tmp/openmpi-1.8.3
# ./configure && make && make install
# ldconfig
```

Program mpirun dan mpicc akan dikenali dalam *path* sistem dan pustaka yang berkaitan akan dimuat jika program dijalankan.

4.7.2.6. Konfigurasi wake-on-LAN

Komputer *slave* dikonfigurasi *headless* sehingga tidak terdapat tombol untuk menyalakan unit komputer, sebagai gantinya *node* dinyalakan dengan paket *wake-on-LAN* dengan alamat MAC (*media access control*) yang bersesuaian.

Tabel 4.4 Asosiasi Nama *Host* dan Alamat MAC Komputer *Slave*

Nama *Host*	Alamat MAC
cluster1	70:71:bc:a3:9e:82
cluster2	70:71:bc:a3:9e:6f
cluster3	70:71:bc:a3:9e:7e
cluster4	70:71:bc:a3:9e:85

Langkah penyalaan komputer *slave* dilakukan dengan menjalankan skrip *shell* `wakeonlan.sh` sesuai dengan argumen alamat MAC komputer *slave* dari *front-end.*

4.7.2.7. Konfigurasi Boot

Komputer *slave* melakukan *boot* dari media *flashdrive,* hal ini menyebabkan kernel perlu menggunakan *initrd* (initial RAM disk). Konfigurasi *boot* dilakukan dengan melakukan instalasi paket kernel *generic* dan modul kernel, pembuatan *initrd* dan pengaturan *bootloader* lilo yang merupakan *bootloader* sistem operasi Salix64.

```
# installpkg kernel-generic-3.2.29-x86_64-1.txz
# cd /usr/share/mkinitrd/
# . mkinitrd_command_generator.sh -a "-w 10" > /boot/new.sh
# . /boot/new.sh
```

Initrd yang dihasilkan akan membuat kernel menunggu selama 10 detik sebelum melakukan *boot* ke partisi *root* karena perangkat *flashdrive* memerlukan waktu yang lebih lama untuk dikenali oleh kernel.

Konfigurasi *bootloader* diperlukan untuk mengatur parameter kernel agar *boot* dilakukan ke perangkat *flashdrive* yang telah dienumerasi menurut UUID oleh udev. Berkas `/etc/lilo.conf` ditambahkan dengan entri:

```
image = /boot/vmlinuz-generic
initrd = /boot/initrd.gz
label = salix
append = "root=/dev/disk/by-uuid/1bb9bf67-3acf-4ae1-be72-9c10653b1f95"
read-only
```

Langkah terakhir konfigurasi *bootloader* adalah instansiasi *boot*loader pada MBR dengan lilo.

```
# lilo -v
```

4.7.2.8. Duplikasi Media Boot Komputer Slave

Duplikasi media *boot* komputer *slave* dilakukan dengan utilitas `dd`:

```
# dd if=/dev/sda of=/dev/sdb bs=4M
```

dengan `/dev/sda` adalah enumerasi *flashdrive* berisi sistem operasi yang telah dikonfigurasi dan `/dev/sdb` adalah enumerasi *flashdrive* yang belum dikonfigurasi.

Media *boot* yang hasil salinan hanya perlu mengulangi prosedur konfigurasi alamat IP dan *host* sesuai alamat IP dan nama *host* masing-masing.

4.8. Implementasi pada Sistem

Setelah perancangan perangkat keras dan perangkat lunak telah dilakukan, langkah selanjutnya adalah mengimplementasikan algoritma genetika pada sistem tersebut.

4.8.1. Pembuatan Proses dan Eksekusi

Seperti pemrograman paralel pada umumnya, komputasi paralel diuraikan menjadi proses konkruen. Pembuatan dan permulaan proses MPI sengaja tidak didefiniskan di dalam standar MPI. Hal ini bergantung pada masing-masing implementasi. Pada proses statis, seluruh proses harus didefinisikan terlebih dahulu sebelum eksekusi dan dimulai secara bersamaan.

Pada penggunaan model SPMD, sebuah program ditulis dan kemudian dieksekusi oleh beberapa prosesor. Bagaimana program-program berbeda itu dimulai bergantung pada baris perintah (*command line*). Contohnya, satu program yang sama bisa mulai dieksekusi oleh empat prosesor yang terpisah secara bersamaan menggunakan perintah: `mpirun prog1 -np 4`.

Sebelum melakukan pemanggilan fungsi MPI, kode harus di-inisialisasi menggunakan `MPI_Init()`. Setelah pemanggilan selesai kode harus ditutup mengguanakan `MPI_Finalize()`. Argumen *command-line* dikirimkan ke `MPI_Init()` agar setup MPI dapat dilakukan.

```
int main(int argc, char **argv)
{
      MPI_Init(&argc, &argv);
        .
        .
        .
      MPI_Finalize();
}
```

(Seperti pada program C sekuensial, `&argc`, jumlah argumen, berisi informasi jumlah argumen, dan `&argv`, vektor argumen, adalah sebuah pointer ke array dari string karakter.)

Awalnya, semua proses terlibat dalam "universe" yang disebut `MPI_COMM_WORLD`. Setiap proses diberi rangking unik, mulai 0 sampai $p - 1$, dimana p adalah jumlah proses. Pada terminologi MPI, `MPI_COMM_WORLD` adalah *komunikator* yang mendefinisikan *jangkauan* operasi komunikasi. Sementara itu, proses-proses memiliki rangking yang terisolasi dengan komunikator. Komunikator yang lain dapat ditentukan dengan untuk beberapa group proses. Untuk program sederhana, penggunaaan `MPI_COMM_WORLD` secara default sudah mencukupi.

4.8.2. Penggunaan Model Komputasi *Single Program Multiple Data* (SPMD)

Model SPMD terasa ideal karena setiap proses akan benar-benar mengeksekusi kode yang sama. Walaupun demikian, secara normal, satu atau lebih prosesor pada semua aplikasi perlu mengeksekusi kode yang berbeda. Agar suatu program dapat melakukan hal itu, perlu disisipkan suatu statemen yang menentukan bagian mana dari kode yang akan dieksekusi oleh setiap prosesor. Oleh sebab itu, model SPMD tidak menghalangi program dengan pendekatan *master-slave*. Namun, baik kode *master* maupun *slave* harus berada dalam program yang sama.

Potongan program berikut mengilustrasikan bagaimana hal itu bisa tercapai.

```
int mpi_tasks, mpi_rank; //deklarasi global
int main(int argc, char **argv)
{

      MPI_Init(&argc, &argv);
      MPI_Comm_size(MPI_COMM_WORLD, &mpi_tasks);
      MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);

     if(mpi_rank == (mpi_tasks-1))
            int u,v; //contoh deklarasi lokal
            master();
     else
            int u,v; //contoh deklarasi lokal
            slave();


      MPI_Finalize();
      return 0;
```

```
}
```

`master()` dan `slave()` berisi prosedur yang akan dieksekusi oleh proses master dan proses slave secara berturut-turut. Penggunaan memori pada model SPMD akan menjadi tidak efisien jika setiap prosesor mengeksekusi kode yang benar-benar berbeda. Namun biasanya hal itu jarang sekali terjadi.

Dalam model SPMD, semua deklarasi variabel global akan diduplikasi pada setiap proses. Variabel yang tidak akan diduplikasi bisa dideklarasikan secara lokal. Artinya variabel dideklarasikan di dalam kode yang hanya dieksekusi oleh proses tersebut.

4.8.3. Implementasi Algoritma Genetika dengan GAlib

Simple Genetic Algorithm melakukan inisialisasi populasi dengan mengkloning individu atau populasi yang dimasukkan ketika *user* membuatnya. Setiap generasi pada algoritma sepenuhnya membuat individui baru dengan seleksi dari populasi sebelumnya kemudian melakukan perkawinan untuk menghasilkan anak baru untuk populasi baru. Proses ini berlanjut hingga menemui kriteria penghentian.

```
float objective(GAGenome &);
int main(int argc, char **argv)
{
      ...
      GABin2DecGenome genome(pheno, objective);
      ...

      GASimpleGA ga(genome);
      ...
      ga.evolve(seed);
}
float objective(GAGenome &)
{
      // Your objective function goes here
}
```

4.8.3.1. Skema Pengodean Kromosom/Individu

```
      ...
#define NBITS_RHO               19
#define MIN_VALUE_RHO           0.5 * M_PI * k
#define MAX_VALUE_RHO           0.5 * M_PI * (k+1)

#define NBITS_THETA             17
#define MIN_VALUE_THETA         0.0
#define MAX_VALUE_THETA         2.0 * M_PI

      ...
```

```
GABin2DecPhenotype pheno;
pheno.add(NBITS_RHO, MIN_VALUE_RHO, MAX_VALUE_RHO);
pheno.add(NBITS_THETA,MIN_VALUE_THETA,MAX_VALUE_THETA);
```

4.8.3.2. Deklarasi Parameter Terminasi

```
// Declare variables GA parameters (termination criteria)
int popsize  = 50; // Population
int ngen     = 100; // Total number of
                    // generations for termination
float pmut   = 0.02;
float pcross = 0.92;
...
...
ga.populationSize(popsize);
ga.nGenerations(ngen);
ga.pMutation(pmut);
ga.pCrossover(pcross);
```

4.8.3.3. Fungsi Obyektif dan Penskalaan *Fitness*

```
GALinearScaling scaling;    // We'll use linear scaling
ga.minimize();  // by default we want to minimize the objective
...
...
ga.scaling(scaling);
```

```
float objective(GAGenome &c) // defined by us
{
    GABin2DecGenome &genome = (GABin2DecGenome &)c;
    float error;
    // Periodic function
    // complex from polar components
    error = fabs(cos(genome.phenotype(0) *
    cos(genome.phenotype(1)))-sin(genome.phenotype(0) *
    cos(genome.phenotype(1))));
    return error;
}
```

4.8.3.4. Skema Seleksi Orang Tua

Secara *default* seleksi yang digunakan pada `GASimpleGA` adalah Roulette Wheel.

4.8.3.5. Elitisme

Elitisme tidak harus digunakan. Tetapi secara default pada `GASimpleGA` Elitisme diaktifkan, artinya individu terbaik dari setiap generasi akan diikutkan pada generasi selanjutnya.

4.8.4. Mencari Akar Lebih dari Satu

Untuk mencari akar-akar persamaan yang jumlahnya lebih dari satu, maka digunakanlah iterasi, hingga akar-akar persamaan maksimum tercapai.

```
#define MAX_ROOTS       13

while(k < MAX_ROOTS)  // how many roots want to be searched
```

```
{

...
k++;
}
```

4.8.5. Pencarian Sekuensial

Digunakan pencarian sekuensial untuk mengatasi munculnya akar-akar persamaan kembar yang mengakibatkan akar-akar persamaan yang lain tidak muncul.

```
int cmp_arr(int arr[], int f, int m) // compare arrays
{
    int i;
    for (i=0; i<m; ++i)
    {
        if (arr[i] == f) return 1;
    }
    return 0;
}
```

4.8.6. Operasi *Blocking* MPI

- MPI_Send()

```
int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)
```

Keterangan parameter dapat dilihat sebagai berikut:

Parameter	Keterangan
buf	: Buffer data yang akan dikirim
count	: Jumlah buffer data
datatype	: Tipe data dari buffer data yang akan dikirim
dest	: Rank tujuan yang akan dikirim data
tag	: Messag tag yang nilainya antara 0 sampai 32767
comm	: Communicator yang digunakan

- MPI_Recv()

```
int MPI_Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)
```

Keterangan parameter dapat dilihat sebagai berikut:

Parameter	Keterangan
buf	: Buffer data yang akan dikirim
count	: Jumlah buffer data
datatype	: Tipe data dari buffer data yang akan dikirim

dest : Rank sumber yang ditunggu data masuk

tag : Messag tag yang nilainya antara 0 sampai 32767

comm : Communicator yang digunakan

status : Status yang telah terjadi pada proses penerimaan data

4.8.7. Migrasi Individu

Pada *library* mpi-galib sudah menyediakan mekanisme migrasi individu yang dapat dilihat di dalam berkas GAPopulation.C, potongan kode program bisa dilihat dibawah ini:

```
...
// Array to store the scores of the individuals
  // The master process:
  if(p.vmpi_rank == (p.vmpi_tasks-1))
  {
    // recives the partial scores
    for(int i=1; i<p.vmpi_tasks; i++)
    {
      is = (int)((float)p.size()/(float)p.vmpi_tasks*((float)i));
      ie = (int)((float)p.size()/(float)p.vmpi_tasks*((float)i+1.0));
      mpi_rc =
MPI_Recv(mpi_score+is, ie-is, MPI_FLOAT, i, 1, MPI_COMM_WORLD, &mpi_Stat);
    }
// and sends the whole array
    for(int i=1; i<p.vmpi_tasks; i++)
      mpi_rc = MPI_Send(mpi_score, p.size(), MPI_FLOAT, i, 1, MPI_COMM_WORLD);
  }
...
```

Proses *master* menerima individu terbaik secara parsial dari beberapa *slave*. Sedangkan sisanya, yaitu proses *child* mengirim hasil individu terbaik pada proses tersebut secara parsial ke proses *master*.

```
   // The rest:
   else
   {
     // sends the partial computed scores
 mpi_rc = MPI_Send(mpi_score+is, ie-is, MPI_FLOAT, 0, 1, MPI_COMM_WORLD);
 mpi_rc =
        MPI_Recv(mpi_score, p.size(), MPI_FLOAT, 0, 1, MPI_COMM_WORLD,
 &mpi_Stat);
   }
 // Update the scores of the individuals
 ...
```

Komunikator yang digunakan adalah `MPI_COMM_WORLD`. Pemetaan proses ke prosesor tidak didefiniskan dalam standar MPI. Tetapi MPI telah mendukung pendefinisian topologi jaringan (meshes, dll). Oleh karena itu, sangatlah mungkin dilakukan pemetaan secara otomatis.

4.8.8. Langkah-langkah Keseluruhan

Langkah-langkah keseluruhan adalah sebagai berikut:

Langkah 1: Inisialisasi komunikator.

Langkah 2: Deklarasi parameter GA (proses s*lave* dan proses *master*).

Langkah 3: Pembuatan dan pemetaan *phenotype* (proses s*lave* dan proses *master*).

Langkah 4: Pembuatan *template* untuk *genome* melalui fungsi tujuan menggunakan *phenotype* yang telah dipetakan (proses s*lave* dan proses *master*).

Langkah 5: Pembuatan GA menggunakan *genome* yang telah dibuat (proses s*lave* dan proses *master*).

Langkah 6: Evolusi dijalankan (proses s*lave* dan proses *master*).

Langkah 7: Penskalaan linier digunakan (proses s*lave* dan proses *master*).

Langkah 8: Migrasi individu terbaik hasil evolusi dari proses *slave* ke proses *master*.

Langkah 9: Menerima individu terbaik dari tiap *slave* dan memilih *score* individu terbaik dari setiap individu (proses *master*).

Langkah 10: Tampilkan hasil pada *user* (proses *master*).

Langkah 11: Jika tiga belas akar telah ditemukan pergi ke langkah 12, selain itu pergi ke langkah 3.

Langkah 12: Terminasi komunikator.

BAB V
PENGUJIAN

5.1. *Error* Hasil Komputasi

Error hasil komputasi didapatkan dari solusi eksak dikurangi hasil komputasi, dipilih salah satu hasil komputasi secara acak sebagai sampel, diperlihatkan pada Tabel 5.1.

Tabel 5.1. Contoh *Error* Hasil Komputasi dari Sebuah Pengujian

Solusi Eksak	Hasil eksekusi serial	Hasil eksekusi paralel	*Error* serial	*Error* paralel
0.785398+0.000000i	0.785396+0.000000i	0.785408+0.000000i	0.000002	0.000010
-2.356194+0.000000i	-2.356198+0.000000i	-2.356149+0.000000i	0.000004	0.000045
3.926991+0.000000i	3.927204+0.000000i	3.926981+0.000000i	0.000213	0.000010
-5.497787+0.000000i	-5.497753+0.000000i	-5.497809+0.000000i	0.000034	0.000022
7.068583+0.000000i	7.068649+0.000000i	7.068491+0.000000i	0.000066	0.000092
-8.63938+0.000000i	-8.639335+0.000000i	-8.63935+0.000000i	0.000045	0.000030
10.210176+0.000000i	10.210171+0.000000i	10.210196+0.000000i	0.000005	0.000020
-11.780872+0.000000i	-11.78106+0.000000i	-11.780743+0.000000i	0.000188	0.000129
13.351769+0.000000i	13.351778+0.000000i	13.35171+0.000000i	0.000009	0.000059
-14.922565+0.000000i	-14.922554+0.000000i	-14.922574+0.000000i	0.000011	0.000009
16.493361+0.000000i	16.493342+0.000000i	16.493345+0.000000i	0.000019	0.000016
-18.064158+0.000000i	-18.064179+0.000000i	-18.064162+0.000000i	0.000021	0.000004
19.634954+0.000000i	19.634954+0.000000i	19.63494+0.000000i	0.000000	0.000014

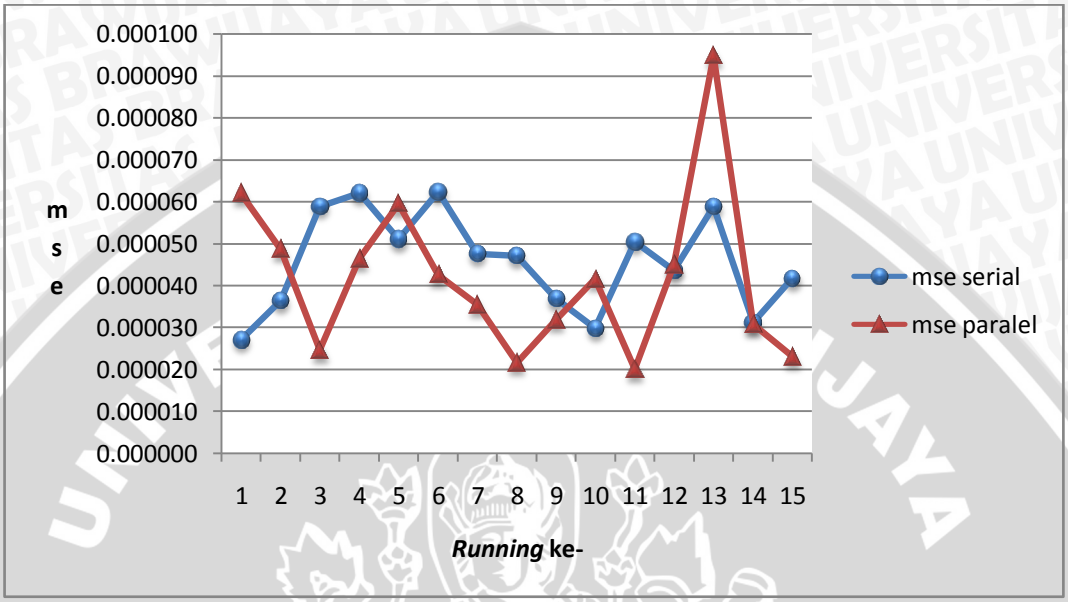
5.2. *Mean Square Error* (Kesalahan Kuadrat Rata-rata)

Data *mean square error* dari hasil komputasi diperlihatkan pada Tabel 5.2 dan gambar grafik diperlihat pada Gambar 5.2.

Tabel 5.2. *Mean Square Error*

Running ke-	*mse* serial	*mse* paralel
1	0.000027	0.000062
2	0.000036	0.000049
3	0.000059	0.000025
4	0.000062	0.000046
5	0.000051	0.000059
6	0.000062	0.000043
7	0.000047	0.000035
8	0.000047	0.000022
9	0.000037	0.000032
10	0.000030	0.000042
11	0.000050	0.000020

12	0.000043	0.000045
13	0.000059	0.000095
14	0.000031	0.000031
15	0.000042	0.000023



Gambar 5.1. *Mean Square Error*

5.3. Kode Program untuk Mengukur Waktu Eksekusi

```
#include <stddef.h>
#include <sys/time.h>
...

double t1,t2,elapsed;
struct timeval tp;
int rtn;

...

int main(int argc, char **argv)
{

    // start timer
    rtn=gettimeofday(&tp, NULL);
    t1=(double)tp.tv_sec+(1.e-6)*tp.tv_usec;

      /*
      Do the work ...
      */

    // stop timer
    rtn=gettimeofday(&tp, NULL);
    t2=(double)tp.tv_sec+(1.e-6)*tp.tv_usec;
    elapsed=t2-t1;
    printf("elapsed time = %f seconds\n", elapsed);
    return 0;
}
```

5.4. Prosedur Pengukuran Waktu Eksekusi

- Letakkan *start timer* pada awal program utama dan *stop timer* pada akhir program utama.
- Eksekusi program, menggunakan bash

```
$ for i in {1..15}; do mpirun -np 4 --hostfile host_i3
./example seed i; done;
```

script tersebut mengeksekusi program `example` sebanyak lima belas kali menggunakan `mpirun`, untuk pengujian serial hilangkan bagian `--hostfile host_i3`

- Catat *elapsed time* yang tampil pada *command-line*.

5.5. Rekapitulasi Waktu Serial

Dari lima belas kali *running* program serial didapatkan lima belas data waktu serial, dapat ditampilakan pada Tabel 5.3. di bawah ini.

Tabel 5.3. Rekapitulasi waktu serial

Ts(seconds)
188.9176
170.0335
115.8477
260.7524
255.4302
179.6951
191.2230
115.4703
135.9766
193.4984
169.0396
160.8663
177.2867
184.1001
210.0859

5.6. Rekapitulasi Waktu Paralel

Dari lima belas kali *running* program paralel didapatkan lima belas data waktu paralel, dapat ditampilakan pada Tabel 5.4. di bawah ini.

Tabel 5.4. Rekapitulasi waktu paralel

Tp (*seconds*)
129.5692
149.1923
135.7561
139.6951
170.2611
247.2510
146.6976
202.3363
234.1125
160.2753
142.8335
104.3855
194.5832
164.1471
175.0947

5.7. Performa (Akselerasi)

Untuk mengukur performa, pertama hitung rata-rata waktu serial dan rata-rata waktu paralel, kemudian masukkan pada persamaan (3-1):

- Rata-rata *Ts*: `180.5482` *seconds*
- Rata-rata *Tp*: `166.4127` *seconds*
- *Sp* = *Ts*/*Tp* ≈ `1.084943`

  Jadi, peningkatan kecepatan sebesar `1.084943`

5.8. Efisiensi

Setelah didapatkan peningkatan kecepatan, langkah selanjutnya adalah mencari seberapa besar efisiensi, menggunakan persamaan (3-2).

$$Ep = Sp / p = 1.084943 / 4 \approx 0.271236$$

Jadi, efisiensiny sebesar `0.271236`.

BAB VI
PENUTUP

Akar-akar persamaan fungsi kompleks berhasil ditemukan dengan algoritma genetika paralel berbasis MPI dengan model SPMD, dan telah direalisasikan dengan sebuah permasalahan uji.

6.1. Kesimpulan

Berdasarkan hasil pengujian dan analisis maka didapatkan beberapa kesimpulan sebagai berikut:

- Penerapan komputasi paralel tidak selalu meningkatan kecepatan secara signifikan.
- Dari hasil pengujian, rata-rata waktu serial `180.5482` *seconds* dan rata-rata waktu paralel `166.4127` *seconds*.
- Peningkatan kecepatan paralel dibanding serial sebesar `1.084943` dengan efisiensi `0.271236`.

6.2. Saran

Ada beberapa saran untuk penelitian lebih lanjut:

- Perlu dicoba menerapkan *local search* di dalam *broad search*, untuk meningkatkan efisiensi pencarian, dan meminimalkan *error* akar-akar persamaan.
- Penyempurnaan algoritma.
- Perlu dicoba pada permasalahan uji lain.
- Perlu dicoba menerapkan parameter terminasi berdasarkan konvergensi.
- Perlu dicoba menerapkan skema algoritma genetika paralel yang lain (*single-population fine-grained* atau *multiple-population coarse-grained GAs*)
- Perlu dicoba mengkombinasikan MPI dengan OpenMP.

DAFTAR PUSTAKA

Anton, Howard. 1987. *Aljabar Linear Elementer*. Alih Bahasa Pantur Silaban, I Nyoman Susila. Jakarta: Penerbit Erlangga.

Kurniawan, Agus. 2010. *Pemrograman Parallel dengan MPI dan C*. Yogyakarta: Penerbit ANDI.

Kusumadewi, Sri., Hari Purnomo. 2005. *Penyelesaian Masalah Optimasi dengan Teknik-Teknik Heuristik*. Yogyakarta: Graha Ilmu.

F. Liu, X. Chen and Z. Huang, "Parallel genetic algorithm for finding roots of complex functional equation," in *Proceedings of 2nd International Conference on Pervasive Computing and Application*, Birmingham, Alabama: IEEE Press, pp. 542-545, 2007.

Michalewicz, Z. 1992. *Genetic Algorithms + Data Structures = Evolution Programs (3ed)*. Berlin: Springer-Verlag.

Paliouras, John D. 1987. *Complex Variables for Scientists and Engineers*. Alih Bahasa Wibisono Gunawan. Jakarta: Penerbit Erlangga.

Reif, John., Sangutevar Rajasekaran. 2008. *Handbook of Parallel Computing: Models, Algorithms and Applications*. New York: Chapman & Hall/CRC Press.

Sutojo, T., Edy Mulyanto, Vincent Suhartono. 2011. *Kecerdasan Buatan*. Yogyakarta: Penerbit ANDI.

Suyanto. 2011. *Artificial Intelligence: Searching, Reasoning, Planning dan Learning*. Bandung: Informatika.

Suyanto. 2008. *Soft Computing: Membangun Mesin Ber-IQ Tinggi*. Bandung: Informatika.

Syaifullah, Indra H. 2014. *Implementasi Pemrosesan Paralel pada Permainan Catur Di Cluster Beowulf*. Skripsi tidak dipublikasikan. Malang: Universitas Brawijaya.

Wall, M. 1995. The GAlib Genetic Algorithm Package (copyright 1995–1996 Massachusetts Institute of Technology; copyright 1996–1999 Matthew Wall). Available at http://lancet.mit.edu/ga.

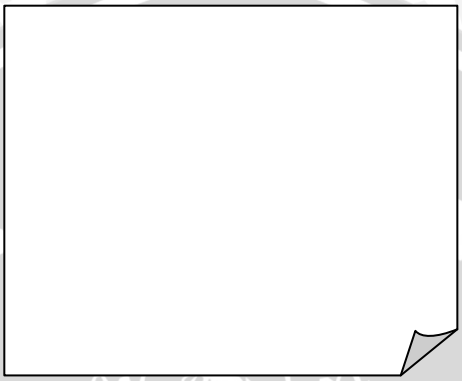
Wall, M.: GAlib; A C++ Library of Genetic Algorithm Components, Massachusetts Institute of Technology. http://lancet.mit.edu/ga/dist/galibdoc.pdf

Wilkinson, Barry., Michael Allen. 2005. *Parallel Programming: Teknik dan Aplikasi Menggunakan Jaringan Workstation & Komputer Paralel*. Yogyakarta: Penerbit ANDI.

H. Wu, Z. Ni, and L. Ni, "Solving roots of complex functional equation based on improved ant colony algorithm," in *Proceedings of International Conference on E-Product E-Service and E-Entertainment*, Henan, China: IEEE press. pp. 1-4, 2010.

"*B0RJA/GAlib-mpi · GitHub*". https://github.com/B0RJA/GAlib-mpi (diakses tanggal 26 Januari 2015)

LAMPIRAN – LAMPIRAN



File example.c:

```
#include <stddef.h>
#include <sys/time.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <complex.h>
#include <ga-mpi/ga.h>
#include <ga-mpi/std_stream.h>
#include "mpi.h"        // Necessary for running the code in parallel

#ifndef M_PI
#define M_PI            3.14159265358979323846
#endif

#define MAX_ROOTS       13

#define NBITS_RHO               19
#define MIN_VALUE_RHO           0.5 * M_PI * k
#define MAX_VALUE_RHO           0.5 * M_PI * (k+1)

#define NBITS_THETA             17
#define MIN_VALUE_THETA         0.0
#define MAX_VALUE_THETA         2.0 * M_PI

double t1,t2,elapsed;
struct timeval tp;
int rtn;

int cmp_arr(int arr[], int f, int m); // declaring cmp_arr functions prototypes
float objective(GAGenome &); // declaring objective functions prototypes

int k;

int mpi_tasks, mpi_rank;
double real[MAX_ROOTS], imag[MAX_ROOTS];
int int_real[MAX_ROOTS];
//~ int int_imag[MAX_ROOTS];
double real_roots[MAX_ROOTS], imag_roots[MAX_ROOTS];

int main(int argc, char **argv)
{
        /* clock_t start = clock(); */
        // start timer
        rtn=gettimeofday(&tp, NULL);
        t1=(double)tp.tv_sec+(1.e-6)*tp.tv_usec;

        // Initialize the MPI environment
        MPI_Init(&argc, &argv);

        // get the number of process
        MPI_Comm_size(MPI_COMM_WORLD, &mpi_tasks);

        // get the rank of process
        MPI_Comm_rank(MPI_COMM_WORLD, &mpi_rank);
```

```
// seed number
unsigned int seed = 0;
for(int i=1 ; i<argc ; i++)
    if(strcmp(argv[i++],"seed") == 0)
        seed = atoi(argv[i]);

// Declare variables GA parameters (termination criteria)
int popsize  = 50; // Population
int ngen     = 100; // Total number of
                    // generations for termination
float pmut   = 0.02;
float pcross = 0.92;

while(k < MAX_ROOTS)  // how many roots want to be searched
{
    // popsize per mpi_tasks must be an integer
    // calculating the popsize for each core:
    popsize = mpi_tasks * int((double)popsize/(double)mpi_tasks+0.999);

    // Create the phenotype for two variables.
    GABin2DecPhenotype pheno;

    // pheno.remove(0); pheno.remove(1);

    // a map of NBITS_RHO bits and range of [MIN_VALUE_RHO,
    // MAX_VALUE_RHO] would use NBITS_RHO bits to represent
    // the number form MIN_VALUE_RHO to MAX_VALUE_RHO
    pheno.add(NBITS_RHO, MIN_VALUE_RHO, MAX_VALUE_RHO);
    //~ pheno.add(19, 0.5 * M_PI * j, 0.5 * M_PI * (j+1));

    // a map of NBITS_THETA bits and range of [MIN_VALUE_THETA,
    // MAX_VALUE_THETA] would use NBITS_THETA bits to represent
    // the number form MIN_VALUE_THETA to MAX_VALUE_THETA
    pheno.add(NBITS_THETA, MIN_VALUE_THETA, MAX_VALUE_THETA);
    //~ pheno.add(17, 0.0, 2.0 * M_PI);

    // Create the template genome using the phenotype map we just made.
    GABin2DecGenome genome(pheno, objective);

    // Now create the GA using the genome and run it
    GASimpleGA ga(genome);
    GALinearScaling scaling;    // We'll use linear scaling
    ga.minimize();  // by default we want to minimize the objective
    ga.populationSize(popsize);
    ga.nGenerations(ngen);
    ga.pMutation(pmut);
    ga.pCrossover(pcross);
    ga.scaling(scaling);

    if(mpi_rank == (mpi_tasks-1))
        ga.scoreFilename("evolution.dat");    // Name of output file for
                                              //scores
    else
        ga.scoreFilename("/dev/null");

    ga.scoreFrequency(1);   // Keep the scores of every generation
    ga.flushFrequency(1);   // Specify how often to write the score to
```

```
                                    // disk
            ga.selectScores(GAStatistics::AllScores);

            // Pass MPI data to the GA class
            ga.mpi_rank(mpi_rank);
            ga.mpi_tasks(mpi_tasks);
            ga.evolve(seed);        // Initialize the GA then evolve it until
                                    // the termination criteria have been
                                    // satisfied

                // Dump the GA results to screen
                if(mpi_rank == (mpi_tasks-1))
                {
                    genome = ga.statistics().bestIndividual();

                    // complex from polar components
                    float rho = genome.phenotype(0);
                    float theta = genome.phenotype(1);
                    double complex z = rho * (cos(theta) + sin(theta) * I);

                    real[k] = creal(z);

                    int_real[k] = (int)real[k]; // convert to integer

                    if(!cmp_arr(int_real, int_real[k], k))
                    {
                        // print the result
                        printf("%f+%fi\n",real[k],imag[k]);
                    }
                    else
                    {
                        k--;
                    }
                    k++;
                }
        }

    // Terminate the MPI environment
    MPI_Finalize();


    // stop timer
    rtn=gettimeofday(&tp, NULL);
    t2=(double)tp.tv_sec+(1.e-6)*tp.tv_usec;
    elapsed=t2-t1;
    printf("elapsed time = %f seconds\n", elapsed);
    /*
    printf("elapsed time = %f seconds",
    double)(clock() - start)/CLOCKS_PER_SEC);
    */

    return 0;
}

float objective(GAGenome &c) // defined by us
{
    GABin2DecGenome &genome = (GABin2DecGenome &)c;
```

```
    float error;
    // Periodic function
    // complex from polar components
    error = fabs(cos(genome.phenotype(0) * cos(genome.phenotype(1)))
            -sin(genome.phenotype(0) * cos(genome.phenotype(1))));
    return error;
}

int cmp_arr(int arr[], int f, int m) // compare arrays
{
    int i;
    for (i=0; i<m; ++i)
    {
        if (arr[i] == f) return 1;
    }
    return 0;
```

Hasil eksekusi serial:

```
0.785389+0.000000i
-2.356195+0.000000i
3.926975+0.000000i
-5.497735+0.000000i
7.068556+0.000000i
-8.639354+0.000000i
10.210143+0.000000i
-11.780996+0.000000i
13.351765+0.000000i
-14.922599+0.000000i
16.493371+0.000000i
-18.064140+0.000000i
19.634947+0.000000i
elapsed time = 188.917583
seconds

0.785377+0.000000i
-2.356192+0.000000i
3.926944+0.000000i
-5.497717+0.000000i
7.068622+0.000000i
-8.639373+0.000000i
10.210199+0.000000i
-11.780999+0.000000i
13.351750+0.000000i
-14.922505+0.000000i
16.493357+0.000000i
-18.064150+0.000000i
19.635000+0.000000i
elapsed time = 170.033533
seconds

0.785392+0.000000i
-2.356150+0.000000i
3.926986+0.000000i
-5.497791+0.000000i
7.068650+0.000000i
-8.639379+0.000000i
10.210031+0.000000i
-11.780973+0.000000i
13.351676+0.000000i
-14.922563+0.000000i
16.493225+0.000000i
-18.064215+0.000000i
19.635058+0.000000i
elapsed time = 115.847749
seconds

0.785394+0.000000i
-2.356280+0.000000i
3.927010+0.000000i
-5.497815+0.000000i
7.068586+0.000000i
-8.639183+0.000000i
10.210122+0.000000i
-11.780959+0.000000i
13.351716+0.000000i
-14.922694+0.000000i
16.493355+0.000000i
-18.064152+0.000000i
19.635089+0.000000i
elapsed time = 260.752410
seconds

0.785413+0.000000i
-2.356192+0.000000i
3.927086+0.000000i
-5.497809+0.000000i
7.068689+0.000000i
-8.639589+0.000000i
10.210168+0.000000i
-11.780973+0.000000i
13.351760+0.000000i
-14.922594+0.000000i
16.493370+0.000000i
-18.064190+0.000000i
19.634980+0.000000i
elapsed time = 255.430234
seconds

0.785355+0.000000i
-2.356163+0.000000i
3.926989+0.000000i
-5.497812+0.000000i
7.068644+0.000000i
-8.639600+0.000000i
10.210170+0.000000i
-11.780971+0.000000i
13.351789+0.000000i
-14.922477+0.000000i
16.493482+0.000000i
-18.064220+0.000000i
19.634922+0.000000i
elapsed time = 179.695124
seconds

0.785396+0.000000i
```

```
-2.356198+0.000000i
3.927204+0.000000i
-5.497753+0.000000i
7.068649+0.000000i
-8.639335+0.000000i
10.210171+0.000000i
-11.781060+0.000000i
13.351778+0.000000i
-14.922554+0.000000i
16.493342+0.000000i
-18.064179+0.000000i
19.634954+0.000000i
elapsed time = 191.222972
seconds

0.785403+0.000000i
-2.356175+0.000000i
3.927066+0.000000i
-5.497798+0.000000i
7.068543+0.000000i
-8.639388+0.000000i
10.210187+0.000000i
-11.781231+0.000000i
13.351791+0.000000i
-14.922584+0.000000i
16.493365+0.000000i
-18.064175+0.000000i
19.634933+0.000000i
elapsed time = 115.470271
seconds

0.785405+0.000000i
-2.356176+0.000000i
3.926992+0.000000i
-5.497756+0.000000i
7.068653+0.000000i
-8.639307+0.000000i
10.210171+0.000000i
-11.780992+0.000000i
13.351788+0.000000i
-14.922558+0.000000i
16.493415+0.000000i
-18.064091+0.000000i
19.634960+0.000000i
elapsed time = 135.976607
seconds

0.785385+0.000000i
-2.356194+0.000000i
3.926974+0.000000i
-5.497786+0.000000i
7.068599+0.000000i
-8.639251+0.000000i
10.210181+0.000000i
-11.780948+0.000000i
13.351712+0.000000i
-14.922583+0.000000i
16.493377+0.000000i
-18.064133+0.000000i
19.634968+0.000000i
elapsed time = 193.498365
seconds

0.785387+0.000000i
-2.356120+0.000000i
3.927006+0.000000i
-5.497794+0.000000i
7.068475+0.000000i
-8.639382+0.000000i
10.210219+0.000000i
-11.780979+0.000000i
13.351896+0.000000i
-14.922570+0.000000i
16.493355+0.000000i
-18.064104+0.000000i
19.634858+0.000000i
elapsed time = 169.039562
seconds

0.785525+0.000000i
-2.356180+0.000000i
3.927028+0.000000i
-5.497839+0.000000i
7.068582+0.000000i
-8.639373+0.000000i
10.209940+0.000000i
-11.780919+0.000000i
13.351751+0.000000i
-14.922565+0.000000i
16.493359+0.000000i
-18.064136+0.000000i
19.634956+0.000000i
elapsed time = 160.866258
seconds

0.785377+0.000000i
-2.356194+0.000000i
3.926993+0.000000i
```

```
-5.497801+0.000000i
7.068525+0.000000i
-8.639379+0.000000i
10.210288+0.000000i
-11.780984+0.000000i
13.352041+0.000000i
-14.922564+0.000000i
16.493403+0.000000i
-18.064141+0.000000i
19.634842+0.000000i
elapsed time = 177.286697
seconds

0.785451+0.000000i
-2.356196+0.000000i
3.926963+0.000000i
-5.497789+0.000000i
7.068633+0.000000i
-8.639426+0.000000i
10.210225+0.000000i
-11.780957+0.000000i
13.351808+0.000000i
-14.922553+0.000000i
16.493356+0.000000i
-18.064143+0.000000i
19.634936+0.000000i
elapsed time = 184.100095
seconds

0.785391+0.000000i
-2.356194+0.000000i
3.927010+0.000000i
-5.497786+0.000000i
7.068498+0.000000i
-8.639382+0.000000i
10.210142+0.000000i
-11.780993+0.000000i
13.351799+0.000000i
-14.922552+0.000000i
16.493356+0.000000i
-18.064010+0.000000i
19.635031+0.000000i
elapsed time = 210.085948
seconds
```

Hasil eksekusi parallel:

```
0.785394+0.000000i
-2.356208+0.000000i
3.926826+0.000000i
-5.497782+0.000000i
7.068702+0.000000i
-8.639379+0.000000i
10.210491+0.000000i
-11.780993+0.000000i
13.351776+0.000000i
-14.922560+0.000000i
16.493357+0.000000i
-18.064109+0.000000i
19.634927+0.000000i
elapsed time = 129.569238
seconds

0.785354+0.000000i
-2.356215+0.000000i
3.926978+0.000000i
-5.497978+0.000000i
7.068611+0.000000i
-8.639298+0.000000i
10.210177+0.000000i
-11.780994+0.000000i
13.351747+0.000000i
-14.922582+0.000000i
16.493364+0.000000i
-18.064148+0.000000i
19.634876+0.000000i
elapsed time = 149.192260
seconds

0.785385+0.000000i
-2.356199+0.000000i
3.926984+0.000000i
-5.497827+0.000000i
7.068581+0.000000i
-8.639392+0.000000i
10.210181+0.000000i
-11.780993+0.000000i
13.351750+0.000000i
-14.922570+0.000000i
16.493352+0.000000i
-18.064168+0.000000i
19.634883+0.000000i
elapsed time = 135.756103
seconds

0.785395+0.000000i
-2.356164+0.000000i
3.926928+0.000000i
-5.497794+0.000000i
7.068667+0.000000i
-8.639372+0.000000i
10.210144+0.000000i
-11.780972+0.000000i
13.351766+0.000000i
-14.922558+0.000000i
16.493367+0.000000i
-18.063933+0.000000i
19.634989+0.000000i
elapsed time = 139.695077
seconds

0.785434+0.000000i
-2.356217+0.000000i
3.926966+0.000000i
-5.497805+0.000000i
7.068614+0.000000i
-8.639340+0.000000i
10.210525+0.000000i
-11.781013+0.000000i
13.351787+0.000000i
-14.922518+0.000000i
16.493371+0.000000i
-18.064158+0.000000i
19.634989+0.000000i
elapsed time = 170.261143
seconds

0.785389+0.000000i
-2.356197+0.000000i
3.926972+0.000000i
-5.497788+0.000000i
7.068519+0.000000i
-8.639381+0.000000i
10.210170+0.000000i
-11.780932+0.000000i
13.351804+0.000000i
-14.922550+0.000000i
16.493041+0.000000i
-18.064168+0.000000i
19.634964+0.000000i
elapsed time = 247.250970
seconds

0.785408+0.000000i
```

```
-2.356149+0.000000i
3.926981+0.000000i
-5.497809+0.000000i
7.068491+0.000000i
-8.639350+0.000000i
10.210196+0.000000i
-11.780743+0.000000i
13.351710+0.000000i
-14.922574+0.000000i
16.493345+0.000000i
-18.064162+0.000000i
19.634940+0.000000i
elapsed time = 146.697550
seconds

0.785401+0.000000i
-2.356232+0.000000i
3.926971+0.000000i
-5.497780+0.000000i
7.068585+0.000000i
-8.639416+0.000000i
10.210235+0.000000i
-11.780920+0.000000i
13.351772+0.000000i
-14.922594+0.000000i
16.493351+0.000000i
-18.064166+0.000000i
19.634971+0.000000i
elapsed time = 202.336328
seconds

0.785416+0.000000i
-2.356186+0.000000i
3.926972+0.000000i
-5.497838+0.000000i
7.068529+0.000000i
-8.639383+0.000000i
10.210196+0.000000i
-11.780980+0.000000i
13.351775+0.000000i
-14.922468+0.000000i
16.493383+0.000000i
-18.064155+0.000000i
19.634950+0.000000i
elapsed time = 234.112548
seconds

0.785406+0.000000i
-2.356206+0.000000i
3.926991+0.000000i
-5.497789+0.000000i
7.068583+0.000000i
-8.639380+0.000000i
10.210165+0.000000i
-11.780969+0.000000i
13.351474+0.000000i
-14.922585+0.000000i
16.493328+0.000000i
-18.064125+0.000000i
19.634925+0.000000i
elapsed time = 160.275312
seconds

0.785399+0.000000i
-2.356194+0.000000i
3.927006+0.000000i
-5.497821+0.000000i
7.068598+0.000000i
-8.639338+0.000000i
10.210175+0.000000i
-11.780968+0.000000i
13.351795+0.000000i
-14.922559+0.000000i
16.493349+0.000000i
-18.064162+0.000000i
19.634962+0.000000i
elapsed time = 142.833497
seconds

0.785446+0.000000i
-2.356195+0.000000i
3.926986+0.000000i
-5.497779+0.000000i
7.068555+0.000000i
-8.639399+0.000000i
10.210067+0.000000i
-11.780969+0.000000i
13.351922+0.000000i
-14.922646+0.000000i
16.493359+0.000000i
-18.064144+0.000000i
19.634973+0.000000i
elapsed time = 104.385464
seconds

0.785396+0.000000i
-2.356195+0.000000i
3.927002+0.000000i
```

```
-5.497802+0.000000i
7.068493+0.000000i
-8.639357+0.000000i
10.210342+0.000000i
-11.781044+0.000000i
13.352004+0.000000i
-14.922591+0.000000i
16.493352+0.000000i
-18.064320+0.000000i
19.635274+0.000000i
elapsed time = 194.583247
seconds

0.785424+0.000000i
-2.356195+0.000000i
3.927007+0.000000i
-5.497817+0.000000i
7.068494+0.000000i
-8.639381+0.000000i
10.210189+0.000000i
-11.780971+0.000000i
13.351774+0.000000i
-14.922609+0.000000i
16.493328+0.000000i
-18.064184+0.000000i
19.634938+0.000000i
elapsed time = 164.147078
seconds

0.785403+0.000000i
-2.356189+0.000000i
3.926996+0.000000i
-5.497822+0.000000i
7.068580+0.000000i
-8.639393+0.000000i
10.210177+0.000000i
-11.780976+0.000000i
13.351778+0.000000i
-14.922569+0.000000i
16.493263+0.000000i
-18.064173+0.000000i
19.634956+0.000000i
elapsed time = 175.094672
seconds
```

***Error* Hasil Komputasi:**

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785389+0.000000i	0.785394+0.000000i	0.000009	0.000004
-2.356194+0.000000i	-2.356195+0.000000i	-2.356208+0.000000i	0.000001	0.000014
3.926991+0.000000i	3.926975+0.000000i	3.926826+0.000000i	0.000016	0.000165
-5.497787+0.000000i	-5.497735+0.000000i	-5.497782+0.000000i	0.000052	0.000005
7.068583+0.000000i	7.068556+0.000000i	7.068702+0.000000i	0.000027	0.000119
-8.63938+0.000000i	-8.639354+0.000000i	-8.639379+0.000000i	0.000026	0.000001
10.210176+0.000000i	10.210143+0.000000i	10.210491+0.000000i	0.000033	0.000315
-11.780872+0.000000i	-11.780996+0.000000i	-11.780993+0.000000i	0.000124	0.000121
13.351769+0.000000i	13.351765+0.000000i	13.351776+0.000000i	0.000004	0.000007
-14.922565+0.000000i	-14.922599+0.000000i	-14.92256+0.000000i	0.000034	0.000005
16.493361+0.000000i	16.493371+0.000000i	16.493357+0.000000i	0.000010	0.000004
-18.064158+0.000000i	-18.06414+0.000000i	-18.064109+0.000000i	0.000018	0.000049
19.634954+0.000000i	19.634947+0.000000i	19.634927+0.000000i	0.000007	0.000027

mse serial	mse paralel
0.000027	0.000062

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785377+0.000000i	0.785354+0.000000i	0.000021	0.000044
-2.356194+0.000000i	-2.356192+0.000000i	-2.356215+0.000000i	0.000002	0.000021
3.926991+0.000000i	3.926944+0.000000i	3.926978+0.000000i	0.000047	0.000013
-5.497787+0.000000i	-5.497717+0.000000i	-5.497978+0.000000i	0.000070	0.000191
7.068583+0.000000i	7.068622+0.000000i	7.068611+0.000000i	0.000039	0.000028
-8.63938+0.000000i	-8.639373+0.000000i	-8.639298+0.000000i	0.000007	0.000082
10.210176+0.000000i	10.210199+0.000000i	10.210177+0.000000i	0.000023	0.000001
-11.780872+0.000000i	-11.780999+0.000000i	-11.780994+0.000000i	0.000127	0.000122
13.351769+0.000000i	13.35175+0.000000i	13.351747+0.000000i	0.000019	0.000022
-14.922565+0.000000i	-14.922505+0.000000i	-14.922582+0.000000i	0.000060	0.000017
16.493361+0.000000i	16.493357+0.000000i	16.493364+0.000000i	0.000004	0.000003
-18.064158+0.000000i	-18.06415+0.000000i	-18.064148+0.000000i	0.000008	0.000010
19.634954+0.000000i	19.635+0.000000i	19.634876+0.000000i	0.000046	0.000078

mse serial	mse paralel
0.000036	0.000049

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785392+0.000000i	0.785385+0.000000i	0.000006	0.000013
-2.356194+0.000000i	-2.35615+0.000000i	-2.356199+0.000000i	0.000044	0.000005
3.926991+0.000000i	3.926986+0.000000i	3.926984+0.000000i	0.000005	0.000007
-5.497787+0.000000i	-5.497791+0.000000i	-5.497827+0.000000i	0.000004	0.000040
7.068583+0.000000i	7.06865+0.000000i	7.068581+0.000000i	0.000067	0.000002
-8.63938+0.000000i	-8.639379+0.000000i	-8.639392+0.000000i	0.000001	0.000012
10.210176+0.000000i	10.210031+0.000000i	10.210181+0.000000i	0.000145	0.000005
-11.780872+0.000000i	-11.780973+0.000000i	-11.780993+0.000000i	0.000101	0.000121
13.351769+0.000000i	13.351676+0.000000i	13.35175+0.000000i	0.000093	0.000019
-14.922565+0.000000i	-14.922563+0.000000i	-14.92257+0.000000i	0.000002	0.000005
16.493361+0.000000i	16.493225+0.000000i	16.493352+0.000000i	0.000136	0.000009
-18.064158+0.000000i	-18.064215+0.000000i	-18.064168+0.000000i	0.000057	0.000010
19.634954+0.000000i	19.635058+0.000000i	19.634883+0.000000i	0.000104	0.000071

mse serial	mse paralel
0.000059	0.000025

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785394+0.000000i	0.785395+0.000000i	0.000004	0.000003
-2.356194+0.000000i	-2.35628+0.000000i	-2.356164+0.000000i	0.000086	0.000030
3.926991+0.000000i	3.92701+0.000000i	3.926928+0.000000i	0.000019	0.000063
-5.497787+0.000000i	-5.497815+0.000000i	-5.497794+0.000000i	0.000028	0.000007
7.068583+0.000000i	7.068586+0.000000i	7.068667+0.000000i	0.000003	0.000084
-8.63938+0.000000i	-8.639183+0.000000i	-8.639372+0.000000i	0.000197	0.000008
10.210176+0.000000i	10.210122+0.000000i	10.210144+0.000000i	0.000054	0.000032
-11.780872+0.000000i	-11.780959+0.000000i	-11.780972+0.000000i	0.000087	0.000100
13.351769+0.000000i	13.351716+0.000000i	13.351766+0.000000i	0.000053	0.000003
-14.922565+0.000000i	-14.922694+0.000000i	-14.922558+0.000000i	0.000129	0.000007
16.493361+0.000000i	16.493355+0.000000i	16.493367+0.000000i	0.000006	0.000006
-18.064158+0.000000i	-18.064152+0.000000i	-18.063933+0.000000i	0.000006	0.000225
19.634954+0.000000i	19.635089+0.000000i	19.634989+0.000000i	0.000135	0.000035

mse serial	mse paralel
0.000062	0.000046

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785413+0.000000i	0.785434+0.000000i	0.000015	0.000036
-2.356194+0.000000i	-2.356192+0.000000i	-2.356217+0.000000i	0.000002	0.000023
3.926991+0.000000i	3.927086+0.000000i	3.926966+0.000000i	0.000095	0.000025
-5.497787+0.000000i	-5.497809+0.000000i	-5.497805+0.000000i	0.000022	0.000018
7.068583+0.000000i	7.068689+0.000000i	7.068614+0.000000i	0.000106	0.000031
-8.63938+0.000000i	-8.639589+0.000000i	-8.63934+0.000000i	0.000209	0.000040
10.210176+0.000000i	10.210168+0.000000i	10.210525+0.000000i	0.000008	0.000349
-11.780872+0.000000i	-11.780973+0.000000i	-11.781013+0.000000i	0.000101	0.000141
13.351769+0.000000i	13.35176+0.000000i	13.351787+0.000000i	0.000009	0.000018
-14.922565+0.000000i	-14.922594+0.000000i	-14.922518+0.000000i	0.000029	0.000047
16.493361+0.000000i	16.49337+0.000000i	16.493371+0.000000i	0.000009	0.000010
-18.064158+0.000000i	-18.06419+0.000000i	-18.064158+0.000000i	0.000032	0.000000
19.634954+0.000000i	19.63498+0.000000i	19.634989+0.000000i	0.000026	0.000035

mse serial	mse paralel
0.000051	0.000059

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785355+0.000000i	0.785389+0.000000i	0.000043	0.000009
-2.356194+0.000000i	-2.356163+0.000000i	-2.356197+0.000000i	0.000031	0.000003
3.926991+0.000000i	3.926989+0.000000i	3.926972+0.000000i	0.000002	0.000019
-5.497787+0.000000i	-5.497812+0.000000i	-5.497788+0.000000i	0.000025	0.000001
7.068583+0.000000i	7.068644+0.000000i	7.068519+0.000000i	0.000061	0.000064
-8.63938+0.000000i	-8.6396+0.000000i	-8.639381+0.000000i	0.000220	0.000001
10.210176+0.000000i	10.21017+0.000000i	10.21017+0.000000i	0.000006	0.000006
-11.780872+0.000000i	-11.780971+0.000000i	-11.780932+0.000000i	0.000099	0.000060
13.351769+0.000000i	13.351789+0.000000i	13.351804+0.000000i	0.000020	0.000035
-14.922565+0.000000i	-14.922477+0.000000i	-14.92255+0.000000i	0.000088	0.000015
16.493361+0.000000i	16.493482+0.000000i	16.493041+0.000000i	0.000121	0.000320
-18.064158+0.000000i	-18.06422+0.000000i	-18.064168+0.000000i	0.000062	0.000010
19.634954+0.000000i	19.634922+0.000000i	19.634964+0.000000i	0.000032	0.000010

mse serial	mse paralel
0.000062	0.000043

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785396+0.000000i	0.785408+0.000000i	0.000002	0.000010
-2.356194+0.000000i	-2.356198+0.000000i	-2.356149+0.000000i	0.000004	0.000045
3.926991+0.000000i	3.927204+0.000000i	3.926981+0.000000i	0.000213	0.000010
-5.497787+0.000000i	-5.497753+0.000000i	-5.497809+0.000000i	0.000034	0.000022
7.068583+0.000000i	7.068649+0.000000i	7.068491+0.000000i	0.000066	0.000092
-8.63938+0.000000i	-8.639335+0.000000i	-8.63935+0.000000i	0.000045	0.000030
10.210176+0.000000i	10.210171+0.000000i	10.210196+0.000000i	0.000005	0.000020
-11.780872+0.000000i	-11.78106+0.000000i	-11.780743+0.000000i	0.000188	0.000129
13.351769+0.000000i	13.351778+0.000000i	13.35171+0.000000i	0.000009	0.000059
-14.922565+0.000000i	-14.922554+0.000000i	-14.922574+0.000000i	0.000011	0.000009
16.493361+0.000000i	16.493342+0.000000i	16.493345+0.000000i	0.000019	0.000016
-18.064158+0.000000i	-18.064179+0.000000i	-18.064162+0.000000i	0.000021	0.000004
19.634954+0.000000i	19.634954+0.000000i	19.63494+0.000000i	0.000000	0.000014

mse serial	mse paralel
0.000047	0.000035

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785403+0.000000i	0.785401+0.000000i	0.000005	0.000003
-2.356194+0.000000i	-2.356175+0.000000i	-2.356232+0.000000i	0.000019	0.000038
3.926991+0.000000i	3.927066+0.000000i	3.926971+0.000000i	0.000075	0.000020
-5.497787+0.000000i	-5.497798+0.000000i	-5.49778+0.000000i	0.000011	0.000007
7.068583+0.000000i	7.068543+0.000000i	7.068585+0.000000i	0.000040	0.000002
-8.63938+0.000000i	-8.639388+0.000000i	-8.639416+0.000000i	0.000008	0.000036
10.210176+0.000000i	10.210187+0.000000i	10.210235+0.000000i	0.000011	0.000059
-11.780872+0.000000i	-11.781231+0.000000i	-11.78092+0.000000i	0.000359	0.000048
13.351769+0.000000i	13.351791+0.000000i	13.351772+0.000000i	0.000022	0.000003
-14.922565+0.000000i	-14.922584+0.000000i	-14.922594+0.000000i	0.000019	0.000029
16.493361+0.000000i	16.493365+0.000000i	16.493351+0.000000i	0.000004	0.000010
-18.064158+0.000000i	-18.064175+0.000000i	-18.064166+0.000000i	0.000017	0.000008
19.634954+0.000000i	19.634933+0.000000i	19.634971+0.000000i	0.000021	0.000017

mse serial	mse paralel
0.000047	0.000022

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785405+0.000000i	0.785416+0.000000i	0.000007	0.000018
-2.356194+0.000000i	-2.356176+0.000000i	-2.356186+0.000000i	0.000018	0.000008
3.926991+0.000000i	3.926992+0.000000i	3.926972+0.000000i	0.000001	0.000019
-5.497787+0.000000i	-5.497756+0.000000i	-5.497838+0.000000i	0.000031	0.000051
7.068583+0.000000i	7.068653+0.000000i	7.068529+0.000000i	0.000070	0.000054
-8.63938+0.000000i	-8.639307+0.000000i	-8.639383+0.000000i	0.000073	0.000003
10.210176+0.000000i	10.210171+0.000000i	10.210196+0.000000i	0.000005	0.000020
-11.780872+0.000000i	-11.780992+0.000000i	-11.78098+0.000000i	0.000120	0.000108
13.351769+0.000000i	13.351788+0.000000i	13.351775+0.000000i	0.000019	0.000006
-14.922565+0.000000i	-14.922558+0.000000i	-14.922468+0.000000i	0.000007	0.000097
16.493361+0.000000i	16.493415+0.000000i	16.493383+0.000000i	0.000054	0.000022
-18.064158+0.000000i	-18.064091+0.000000i	-18.064155+0.000000i	0.000067	0.000003
19.634954+0.000000i	19.63496+0.000000i	19.63495+0.000000i	0.000006	0.000004

mse serial	mse paralel
0.000037	0.000032

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785385+0.000000i	0.785406+0.000000i	0.000013	0.000008
-2.356194+0.000000i	-2.356194+0.000000i	-2.356206+0.000000i	0.000000	0.000012
3.926991+0.000000i	3.926974+0.000000i	3.926991+0.000000i	0.000017	0.000000
-5.497787+0.000000i	-5.497786+0.000000i	-5.497789+0.000000i	0.000001	0.000002
7.068583+0.000000i	7.068599+0.000000i	7.068583+0.000000i	0.000016	0.000000
-8.63938+0.000000i	-8.639251+0.000000i	-8.63938+0.000000i	0.000129	0.000000
10.210176+0.000000i	10.210181+0.000000i	10.210165+0.000000i	0.000005	0.000011
-11.780872+0.000000i	-11.780948+0.000000i	-11.780969+0.000000i	0.000076	0.000097
13.351769+0.000000i	13.351712+0.000000i	13.351474+0.000000i	0.000057	0.000295
-14.922565+0.000000i	-14.922583+0.000000i	-14.922585+0.000000i	0.000018	0.000020
16.493361+0.000000i	16.493377+0.000000i	16.493328+0.000000i	0.000016	0.000033
-18.064158+0.000000i	-18.064133+0.000000i	-18.064125+0.000000i	0.000025	0.000033
19.634954+0.000000i	19.634968+0.000000i	19.634925+0.000000i	0.000014	0.000029

mse serial	mse paralel
0.000030	0.000042

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785387+0.000000i	0.785399+0.000000i	0.000011	0.000001
-2.356194+0.000000i	-2.35612+0.000000i	-2.356194+0.000000i	0.000074	0.000000
3.926991+0.000000i	3.927006+0.000000i	3.927006+0.000000i	0.000015	0.000015
-5.497787+0.000000i	-5.497794+0.000000i	-5.497821+0.000000i	0.000007	0.000034
7.068583+0.000000i	7.068475+0.000000i	7.068598+0.000000i	0.000108	0.000015
-8.63938+0.000000i	-8.639382+0.000000i	-8.639338+0.000000i	0.000002	0.000042
10.210176+0.000000i	10.210219+0.000000i	10.210175+0.000000i	0.000043	0.000001
-11.780872+0.000000i	-11.780979+0.000000i	-11.780968+0.000000i	0.000107	0.000096
13.351769+0.000000i	13.351896+0.000000i	13.351795+0.000000i	0.000127	0.000026
-14.922565+0.000000i	-14.92257+0.000000i	-14.922559+0.000000i	0.000005	0.000006
16.493361+0.000000i	16.493355+0.000000i	16.493349+0.000000i	0.000006	0.000012
-18.064158+0.000000i	-18.064104+0.000000i	-18.064162+0.000000i	0.000054	0.000004
19.634954+0.000000i	19.634858+0.000000i	19.634962+0.000000i	0.000096	0.000008

mse serial	mse paralel
0.000050	0.000020

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785525+0.000000i	0.785446+0.000000i	0.000127	0.000048
-2.356194+0.000000i	-2.35618+0.000000i	-2.356195+0.000000i	0.000014	0.000001
3.926991+0.000000i	3.927028+0.000000i	3.926986+0.000000i	0.000037	0.000005
-5.497787+0.000000i	-5.497839+0.000000i	-5.497779+0.000000i	0.000052	0.000008
7.068583+0.000000i	7.068582+0.000000i	7.068555+0.000000i	0.000001	0.000028
-8.63938+0.000000i	-8.639373+0.000000i	-8.639399+0.000000i	0.000007	0.000019
10.210176+0.000000i	10.20994+0.000000i	10.210067+0.000000i	0.000236	0.000109
-11.780872+0.000000i	-11.780919+0.000000i	-11.780969+0.000000i	0.000047	0.000097
13.351769+0.000000i	13.351751+0.000000i	13.351922+0.000000i	0.000018	0.000153
-14.922565+0.000000i	-14.922565+0.000000i	-14.922646+0.000000i	0.000000	0.000081
16.493361+0.000000i	16.493359+0.000000i	16.493359+0.000000i	0.000002	0.000002
-18.064158+0.000000i	-18.064136+0.000000i	-18.064144+0.000000i	0.000022	0.000014
19.634954+0.000000i	19.634956+0.000000i	19.634973+0.000000i	0.000002	0.000019

mse serial	mse paralel
0.000043	0.000045

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785377+0.000000i	0.785396+0.000000i	0.000021	0.000002
-2.356194+0.000000i	-2.356194+0.000000i	-2.356195+0.000000i	0.000000	0.000001
3.926991+0.000000i	3.926993+0.000000i	3.927002+0.000000i	0.000002	0.000011
-5.497787+0.000000i	-5.497801+0.000000i	-5.497802+0.000000i	0.000014	0.000015
7.068583+0.000000i	7.068525+0.000000i	7.068493+0.000000i	0.000058	0.000090
-8.63938+0.000000i	-8.639379+0.000000i	-8.639357+0.000000i	0.000001	0.000023
10.210176+0.000000i	10.210288+0.000000i	10.210342+0.000000i	0.000112	0.000166
-11.780872+0.000000i	-11.780984+0.000000i	-11.781044+0.000000i	0.000112	0.000172
13.351769+0.000000i	13.352041+0.000000i	13.352004+0.000000i	0.000272	0.000235
-14.922565+0.000000i	-14.922564+0.000000i	-14.922591+0.000000i	0.000001	0.000026
16.493361+0.000000i	16.493403+0.000000i	16.493352+0.000000i	0.000042	0.000009
-18.064158+0.000000i	-18.064141+0.000000i	-18.06432+0.000000i	0.000017	0.000162
19.634954+0.000000i	19.634842+0.000000i	19.635274+0.000000i	0.000112	0.000320

mse serial	mse paralel
0.000059	0.000095

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785451+0.000000i	0.785424+0.000000i	0.000053	0.000026
-2.356194+0.000000i	-2.356196+0.000000i	-2.356195+0.000000i	0.000002	0.000001
3.926991+0.000000i	3.926963+0.000000i	3.927007+0.000000i	0.000028	0.000016
-5.497787+0.000000i	-5.497789+0.000000i	-5.497817+0.000000i	0.000002	0.000030
7.068583+0.000000i	7.068633+0.000000i	7.068494+0.000000i	0.000050	0.000089
-8.63938+0.000000i	-8.639426+0.000000i	-8.639381+0.000000i	0.000046	0.000001
10.210176+0.000000i	10.210225+0.000000i	10.210189+0.000000i	0.000049	0.000013
-11.780872+0.000000i	-11.780957+0.000000i	-11.780971+0.000000i	0.000085	0.000099
13.351769+0.000000i	13.351808+0.000000i	13.351774+0.000000i	0.000039	0.000005
-14.922565+0.000000i	-14.922553+0.000000i	-14.922609+0.000000i	0.000012	0.000044
16.493361+0.000000i	16.493356+0.000000i	16.493328+0.000000i	0.000005	0.000033
-18.064158+0.000000i	-18.064143+0.000000i	-18.064184+0.000000i	0.000015	0.000026
19.634954+0.000000i	19.634936+0.000000i	19.634938+0.000000i	0.000018	0.000016

mse serial	mse paralel
0.000031	0.000031

Solusi Eksak	Hasil eksekusi serial:	Hasil eksekusi parallel:	Error serial	Error paralel
0.785398+0.000000i	0.785391+0.000000i	0.785403+0.000000i	0.000007	0.000005
-2.356194+0.000000i	-2.356194+0.000000i	-2.356189+0.000000i	0.000000	0.000005
3.926991+0.000000i	3.92701+0.000000i	3.926996+0.000000i	0.000019	0.000005
-5.497787+0.000000i	-5.497786+0.000000i	-5.497822+0.000000i	0.000001	0.000035
7.068583+0.000000i	7.068498+0.000000i	7.06858+0.000000i	0.000085	0.000003
-8.63938+0.000000i	-8.639382+0.000000i	-8.639393+0.000000i	0.000002	0.000013
10.210176+0.000000i	10.210142+0.000000i	10.210177+0.000000i	0.000034	0.000001
-11.780872+0.000000i	-11.780993+0.000000i	-11.780976+0.000000i	0.000121	0.000104
13.351769+0.000000i	13.351799+0.000000i	13.351778+0.000000i	0.000030	0.000009
-14.922565+0.000000i	-14.922552+0.000000i	-14.922569+0.000000i	0.000013	0.000004
16.493361+0.000000i	16.493356+0.000000i	16.493263+0.000000i	0.000005	0.000098
-18.064158+0.000000i	-18.06401+0.000000i	-18.064173+0.000000i	0.000148	0.000015
19.634954+0.000000i	19.635031+0.000000i	19.634956+0.000000i	0.000077	0.000002

mse serial	mse paralel
0.000042	0.000023