Mencari Akar-akar Persamaan Fungsi kompleks Menggunakan Algoritma Genetika Paralel

Ahmad Riza Abdullah, Muhammad Aziz Muslim, S.T., M.T., Ph.D., Waru Djuriatno, S.T., M.T.
Teknik Elektro Universitas Brawijaya
ahmad.riza.abdullah@gmail.com

Abstract

In this paper, a parallel genetic algorithm for finding all roots of complex functional equation based on Message Passing Interface (MPI) is present, research are made in some technical problems for realizing. We describe the design and implement of parallel genetic algorithm for finding roots complex functional equation.

Keywords: *Parallel Genetic Algorithm, Message Passing Interface, finding roots, complex functional equation*

Abstrak

Penelitian ini menyajikan pencarian akar-akar persamaan fungsi kompleks menggunakan algoritma genetika paralel berbasis *Message Passing Interface (MPI)*, penelitian dibuat pada suatu permasalahan teknis untuk kemudian direalisasikan. Kami menjelaskan desain dan implementasi algoritma genetika paralel untuk mencari akar-akar persamaan fungsi kompleks.

Kata Kunci: Algoritma Genetika Paralel, *Message Passing Interface*, pencarian akar-akar, persamaan fungsi kompleks.

I. Pendahuluan

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[1].

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 sensitivitas 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[2].

Oleh karena itu pada penelitian ini kami akan mencoba 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.

1.1. 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

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

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

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

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[3].

1.2. 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: $\boldsymbol{min}|f(z)|$

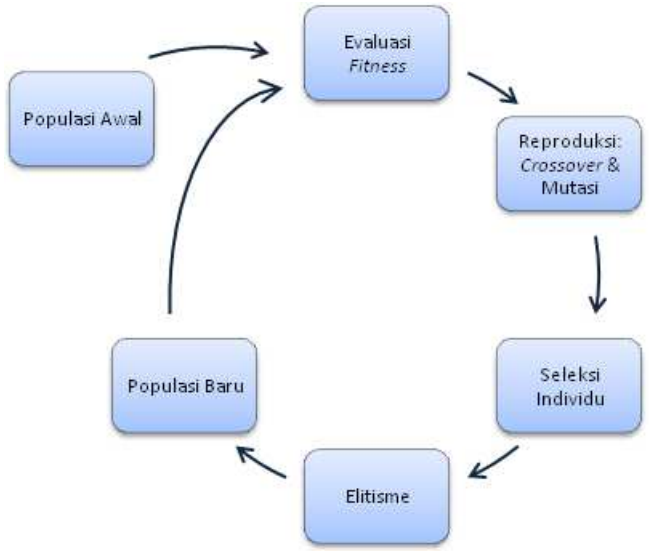
Jika permasalahan memiliki dimensi yang besar, GA akan tidak efisien. Bagaimana menurunkan dimensi permasalahan adalah penting untuk meningkatkan efisiensi GA. Pada algoritma, kami membatasi solusi di dalam

lingkaran atau busur sirkular dari permasalahan. Untuk fungsi kompleks $f(z) = 0$, terdapat dua dimensi variabel: bagian riil x dan bagian imajiner y, kami mereduksi 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[4].

1.3. Siklus Algoritma Genetika[5][6]

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.

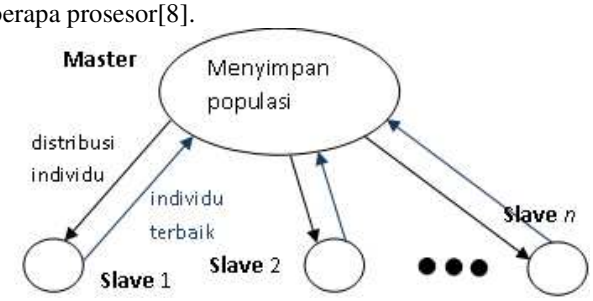
Siklus ini kemudian diperbaiki oleh Zbigniew Michalewicz dengan menambahkan suatu proses elitisme dan membalik proses reproduksi dahulu, kemudian proses seleksi seperti tampak pada gambar 5.



Gambar 1. Siklus algoritma genetika Zbigniew Michalewicz hasil perbaikan dari siklus algoritma genetika yang diperkenalkan oleh David Goldberg[7]

1.4. Algoritma Genetika Paralel

Tipe *global single-population master-slave* GAs akan digunakan pada penelitian ini, skematik diperlihatkan pada Gambar 2. Pada master-slave GA terdapat suatu populasi (GA serial), tetapi evaluasi fitness didistribusikan pada beberapa prosesor[8].



Gambar 2. skematik *master-slave* PGA. *Master* menyimpan populasi, eksekusi operasi GA, dan distribusi individu ke *slaves*. *Slaves* hanya mengevaluasi *fitness* individu

1.5. Message Passing Interface (MPI)[1][5]

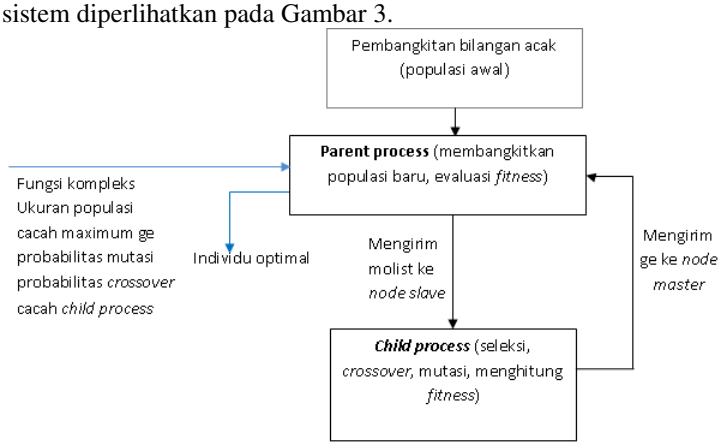
Message Passing Interface atau disingkat MPI adalah bahasa independen untuk protokol komunikasi yang digunakan program paralel pada komputer. MPI diterima banyak komunitas pemrograman paralel.

II. Desain dan Implementasi

Perangkat yang digunakan untuk implementasi sistem komputasi paralel pencarian akar-akar persamaan fungsi kompleks diuraikan sebagai berikut:

2.1. Abstraksi Sistem

Sistem komputasi paralel terdiri dari beberapa komponen yang dapat digambarkan dengan blok diagram sistem diperlihatkan pada Gambar 3.



Gambar 3. Abstraksi sistem komputasi paralel

2.2. Langkah Kerja Sistem

Kami mengadopsi *Master-slave control network* PGA dan murni pola *node programming* (mode Single Program Multiple Data [SPMD]). Langkah-langkah Algoritma Genetika Paralel[9] 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.

2.3. Perangkat Keras

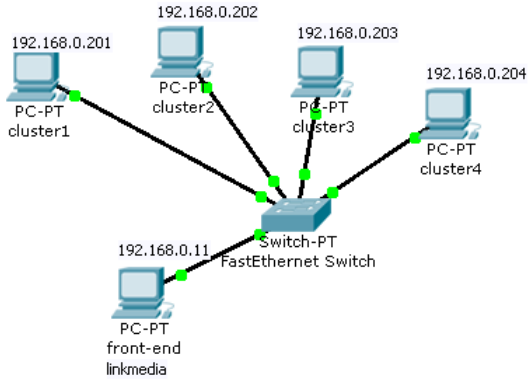
Perangkat keras yang digunakan terdiri atas:

- 4 unit komputer personal: Intel i3 540, DDR3 2 GB. (Gambar 4.)
- 1 unit komputer personal: Intel Dual Core E5400, DDR2 2GB.

- *switch* TL-SG1024D dan perkabelan UTP Cat 5e

Gambar 4. Empat Unit Komputer *Slave*

Konfigurasi jaringan lokal menggunakan topologi bintang diperlihatkan pada Gambar 5.



Gambar 5. Konfigurasi Jaringan lokal Komputer Paralel

2.4. Perangkat Lunak

- GNU/Linux Salix64 14.1: kernel 3.2.29, glibc 2.15, gcc 4.7.1,
- GNU/Linux Salix64 14.0: gcc 4.7.1, glibc 2.15, dan kernel Linux 3.2.29 64 bit,
- Open MPI 1.8.3
- GAlib 2.4.7[10]

2.5. Permasalahan Uji

Parameter Terminasi Algoritma Genetika[9]:
popsize = 50; P_c = 0,92; P_m = 0,02; maxgen = 100
Bila diketahui suatu fungsi kompleks (dengan nilai imajinernya selalu nol)[9][11]:
$f(z) = \cos(z) - \sin(z) = 0$
perkiraan: $z = \rho(\cos\theta + i \sin\theta)$, $\rho \in [0, 20]$, $\theta \in [0, 2\pi)$
presisi : $e1 = 0{,}00001$, $e2 = 0{,}00001$
error : solusi eksak – hasil

III. Pengujian

3.1. *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 1 untuk contoh *Error* Hasil Komputasi dari Sebuah Pengujian serial, dan Tabel 2 contoh *Error* Hasil Komputasi dari Sebuah Pengujian paralel.

Tabel 1. Contoh *Error* Hasil Komputasi dari Sebuah Pengujian serial

Solusi Eksak	Hasil eksekusi	*Error*
0.785398+0.000000i	0.785396+0.000000i	0.000002
-2.356194+0.000000i	-2.356198+0.000000i	0.000004
3.926991+0.000000i	3.927204+0.000000i	0.000213
-5.497787+0.000000i	-5.497753+0.000000i	0.000034
7.068583+0.000000i	7.068649+0.000000i	0.000066
-8.63938+0.000000i	-8.639335+0.000000i	0.000045
10.210176+0.000000i	10.210171+0.000000i	0.000005
-11.780872+0.000000i	-11.78106+0.000000i	0.000188
13.351769+0.000000i	13.351778+0.000000i	0.000009
-14.922565+0.000000i	-14.922554+0.000000i	0.000011
16.493361+0.000000i	16.493342+0.000000i	0.000019
-18.064158+0.000000i	-18.064179+0.000000i	0.000021
19.634954+0.000000i	19.634954+0.000000i	0.000000

Tabel 2. Contoh *Error* Hasil Komputasi dari Sebuah Pengujian paralel

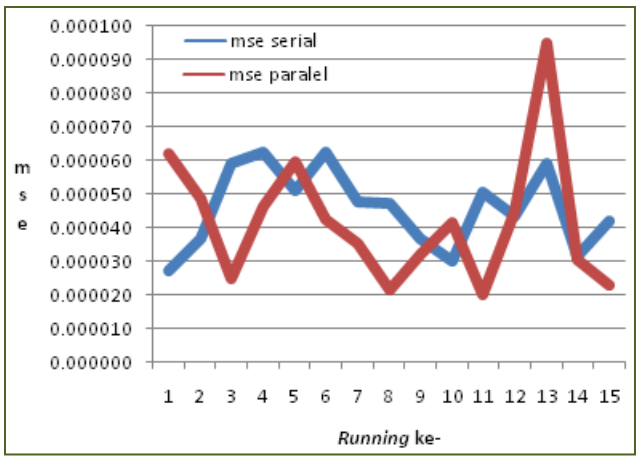
Solusi Eksak	Hasil eksekusi	*Error*
0.785398+0.000000i	0.785408+0.000000i	0.000010
-2.356194+0.000000i	-2.356149+0.000000i	0.000045
3.926991+0.000000i	3.926981+0.000000i	0.000010
-5.497787+0.000000i	-5.497809+0.000000i	0.000022
7.068583+0.000000i	7.068491+0.000000i	0.000092
-8.63938+0.000000i	-8.63935+0.000000i	0.000030
10.210176+0.000000i	10.210196+0.000000i	0.000020
-11.780872+0.000000i	-11.780743+0.000000i	0.000129
13.351769+0.000000i	13.35171+0.000000i	0.000059
-14.922565+0.000000i	-14.922574+0.000000i	0.000009
16.493361+0.000000i	16.493345+0.000000i	0.000016
-18.064158+0.000000i	-18.064162+0.000000i	0.000004
19.634954+0.000000i	19.63494+0.000000i	0.000014

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

Data *mean square error* dari hasil komputasi diperlihatkan pada Tabel 3 dan gambar grafik diperlihat pada Gambar 6.

Tabel 3. *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 6. Mean Square Error

3.3. Performa (Akselerasi)[8][12]

Untuk mengukur performa, pertama hitung rata-rata waktu serial dan rata-rata waktu paralel.

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

Jadi, peningkatan kecepatan sebesar: 1.084943

3.4. Efisiensi

Setelah didapatkan peningkatan kecepatan, langkah selanjutnya adalah mencari seberapa besar efisiensi. *Ep* = *Sp* / *p* = 1.084943 / 4 ≈ 0.271236

Jadi, efisiensinya sebesar: 0.271236

IV. Penutup

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

4.1. 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 sebesar 1.084943 dengan efisiensi 0.271236.

4.2. Saran

Ada beberapa saran untuk penelitian lebih lanjut:

- Penyempurnaan algoritma.
- Perlu dicoba menerapkan *local search* di dalam *broad search*.
- 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.

Referensi

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

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

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

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

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

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

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

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

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

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

[11] 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.

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