Rancang Bangun Komputasi Paralel pada Algoritma *Longest Common Subsequence* untuk Perbandingan Genom Menggunakan *Framework* OpenMPI

Evaria Ayu Nurjana, Waskitho Wibisono, dan Hudan Studiawan  
Jurusan Teknik Informatika, Fakultas Teknologi Informasi   
Institut Teknologi Sepuluh Nopember (ITS)  
Jl. Arief Rahman Hakim, Surabaya 60111 Indonesia  
*e-mail*: {waswib,hudan}@if.its.ac.id

*Abstrak*— Proses mencari *global alignment* dari dua genom dengan cara membandingkan urutannya merupakan salah satu permasalahan yang sering ditemui pada komputasi biologi. Salah satu cara untuk menyelesaikan permasalahan tersebut adalah mengimplementasikan algoritma *Longest Common Subsequence* (*LCS*) pada proses pencarian *global* *alignment* dari kedua genom. Data dari genom sendiri sangatlah besar sehingga dibutuhkan suatu teknik yang dapat mengurangi waktu komputasi. Hal tersebut dapat tercapai dengan cara mengimplementasikan komputasi paralel terhadap algoritma *LCS*. Oleh karena itu Tugas Akhir ini mengimplementasikan Algoritma *LCS* yang diparalelkan. Tujuan dari Tugas Akhir ini adalah untuk mencari global alignment dari dua genom serta menganalisa dan membandingkan kinerja algoritma *LCS* secara sekuensial serta paralel. Implementasi algoritma *LCS* secara paralel menggunakan *library* OpenMPI. Berdasarkan hasil uji coba, waktu eksekusi *LCS* secara paralel terbukti lebih cepat daripada sekuensial dan panjang karakter yang dapat diproses pada komputasi paralel lebih banyak daripada sekuensial.

*Kata Kunci*—*LCS*, OpenMPI, *Dynamic Programming*, genom.

# PENDAHULUAN

T

iap makhluk hidup memiliki DNA (*deoxyribonuclei acid*) yang merupakan kode genetik pada makhluk hidup [1]. *DNA* terdiri dari 2 pasang 4 huruf alfabet (A, T, C, dan G) dimana A (*adenine*) berpasangan dengan T(*thymine*), dan C\(*cytosine*) berpasangan dengan G(*guanine*). Pasangan tersebut disebut dengan *base* *pairs*. Genom sendiri adalah kumpulan semua DNA di suatu makhluk hidup. Ukuran dari genom sendiri sangatlah besar yaitu mulai dari ratusan *base* *pairs* hingga jutaan *base* *pairs*.

*Genes* adalah bagian dari DNA dan membentuk sebuah protein. Protein merupakan urutan 20 huruf alfabet yang disebut dengan asam amino [1].

*Biological* *sequence* adalah representasi dari urutan simbol – simbol yang digunakan untuk mencari kesamaan dari dua urutan genom. Perbandingan dari *biological* *sequence* merupakan salah satu masalah yang ada pada komputasi biologi. Perbandingan tersebut dilakukan untuk mencari dua gen yang *homologous*. Gen *homologous* adalah dua gen yang mempunyai gen ansestor yang sama. Kesamaan urutan genom dapat berguna untuk menyelediki fungsi umum dari organisme yang sesuai dan untuk memahami bagaimana gen terorganisir dan berkaitan dalam beberapa fungsi [1].

*Dynamic* *Programming* adalah teknik pemecahan masalah dari submasalah yang ada. Submasalah muncul ketika solusi masalah dari submasalah yang berkaitan muncul sehingga dapat dikatakan bahwa submasalah satu dengan yang lainnya berkaitan. Penyelesaian masalah menggunakan *Dynamic Programming* yaitu menyimpan solusi dari tiap submasalah pada suatu tabel hingga masalah yang sebenarnya dapat terselesaikan.

Penelitian ini mengimplementasikan algoritma *Longest Common Subsequence* (*LCS*) dalam komputasi paralel. Algoritma *Longest Common Subsequence* merupakan salah satu dari algoritma *dynamic programming* yang sering digunakan dalam permasalahan yang luas [2], [3] termasuk di bidang bioinformatika. Algoritma ini mencari *LCS* dari dua *string.* Algoritma paralel sendiri merupakan sebuah pendekatan untuk memecahkan suatu masalah dengan menggunakan beberapa prosesor [4]. Data dari genom sangatlah besar sehingga dengan *Longest Common Subsequence* yang diparalelkan diharapkan mampu mengurangi waktu eksekusi untuk menemukan *global alignment* dari dua genom.

# ANALISIS DAN PERANCANGAN

## Penjelasan Umum Algoritma LCS

Pada subbab ini menjelaskan secara umum mengenai algoritma paralel *LCS*. Sebagai contoh, jika ada dua *string* *X* dan *Y* dan *p* proses dengan |*X*|=*m* dan |Y|=*n*, maka cara paralelnya adalah:

*String X* dan *Y* dibaca di tiap komputer, dimana *string* *Y* dibagi menjadi beberapa *p* proses bagian dengan ukuran , sehingga tiap proses *Pi*, 0 ≤ *i* < *p*, memproses bagian ke-*i* dari s*tring* *Y* misalnya ada 10 karakter yaitu AAACAGTAAA dengan empat proses maka *P­*0 memproses bagian dari *string Y*(*Y*0....*Y*2) yaitu AAA, *P*­1 memproses bagian dari *string Y*(*Y*3....*Y*5) CAG, *P­*2­­ *­­*­memproses bagian dari *string Y*(*Y6* dan *Y*7) TA, dan *P*­3­ memproses bagian dari *string Y*(*Y*8dan *Y*9) AA.



Jika ada empat proses yang berjalan. proses yang pertama kali berjalan adalah proses pertama *P*­0, selanjutnya proses *P­*0­ dan *P­*1*­­*, kemudian proses *P­*0­, *P­*1*­­*, dan *P­*2, kemudian proses *P­*0, *P*1, *P*2, dan *P*3, lalu menjadi *P*­1*­*, *P*­2­, dan *P*3 dan seterusnya hingga terakhir yang memproses adalah proses paling terakhir yaitu *P*3.

Tiap proses memiliki ukuran *vector* yaitu. Pada Tabel 1 menunjukkan proses perhitungan *LCS* tiap proses. *P*­0­ menghitung *LCS* dengan seluruh *string X* yaitu TAAGTTATTAT dengan bagian pertama dari empat bagian *string Y* yaitu AAA. *P­­­1­* menghitung *LCS* seluruh *string X* dengan bagian kedua dari empat bagian *string Y* yaitu CAG begitu juga seterusnya. Variabel *k* pada Tabel 1 menunjukkan perulangan pada proses perhitungan *LCS*. Pada *k*=1 yang pertama kali berjalan yaitu proses *P1*, dengan membandingkan *X*[0]=T dengan bagian pertama dari *string Y* dan hasilnya disimpan pada *vector L*.

Gambar 1. Desain Arsitektur Program dengan OpenMPI

Proses *P*0 mengirimkan kolom paling terakhir yaitu *L*[0][2] ke proses *P*1 dan proses *P*1 menerima dan menyimpan pada *vector* paling pertama *L*[0][0] yang berwarna untuk melanjutkan perhitungan. Perhitungan pada proses *P*1 membutuhkan kolom paling terakhir dari proses *P*0 yang disimpan pada *vector* paling pertama pada bagian *vector L* diproses *P*1. Proses *P*1 menghitung string *X*[0]=T dengan bagian kedua dari *string* *Y*. selanjutnya kolom paling terakhir dari proses *P*1 yaitu *L*[0][3] dikirim ke proses *P*2. Begitu juga dengan proses *P*2, untuk melanjutkan perhitungan dibutuhkan kolom paling terakhir dari proses *P*1 yaitu *L*[0][3]. Proses *P*2 menghitung *A*[0] dengan bagian ketiga *string* *Y* dan proses *P*3 menghitung *A*[0] dengan bagian keempat dari *string* *Y*.

Lebih jelasnya, proses perhitungan dimulai pertama kali oleh proses *P*0. Proses *P*0 menghitung *X*[0] dengan bagian pertama *string* *Y*. Selanjutnya proses *P*­0 mengirim kolom terakhir yaitu *L*[0][2] ke proses *P*­1­­.

Tabel 1. Proses Perhitungan LCS Secara Paralel

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 proses | | | | | *alpha*=0.125 | | | | | | | | | | | |
| *L* | | *Y* | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 | 6 | 7 | 8 | 8 | 9 | 10 |
| 0 | 1 | 2 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 0 | 1 | 2 | 3 |
| *k* | *X* |  | A | A | A |  | C | A | G |  | T | A |  | A | A |  |
| 1 | 0 | T | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 2 | A | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| 10 | 3 | G | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 |
| 13 | 4 | T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 3 |
| 16 | 5 | T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 20 | 6 | A | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 4 | 4 | 4 | 4 | 4 |
| 23 | 7 | T | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 5 | 5 | 5 |
| 26 | 8 | T | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 5 | 5 | 5 |
| 29 | 9 | A | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | 5 | 5 | 5 |
| 32 | 10 | T | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 5 | 6 | 6 |
| Proses | | | *P*0 | | | *P*1 | | | | *P*2 | | | | | *P*3 | |

Proses *P*­1 melanjutkan perhitungan *X*[1] dengan bagian pertama *string* *Y* dan proses *P*­1 menerima kiriman dari proses *P*­0 dan menyimpannya pada kolom pertama pada proses *P*­1 yaitu *L*[0][0] kemudian proses *P*1 melakukan perhitungan.Proses *P*1 mengirim kolom terakhir setelah perhitungannya selesai ke proses *P*2. Proses *P*2 menerima dan melakukan perhitungan. Proses *P*0 mengirim data ke proses *P*1 yaitu *L*[1][2] selanjutnya melanjutkan ke perhitungan *X*[3] dengan bagian pertama dari *string Y*. Proses *P*1 menerima dan melakukan perhitungan.

Pada bagian kolom berwarna merupakan indeks asli *string B* sedangkan kolom berwarna  adalah indeks yang menyesuaikaikan dengan ukuran pada *vector* *L* bagian kolomnya yaitu (*n*/*p*) + 2.

Variabel *alpha* sendiri digunakan untuk mengurangi panjang karakter yang diproses tiap proses [5] yang ditunjukkan pada Tabel 1. Terlihat pada Tabel 1 bahwa di tiap *k*, tiap proses hanya memproses satu huruf pada karakter *X*. Desain arsitektur perangkat lunak secara umum menggunakan OpenMPI ditunjukkan pada Gambar 1.

## Proses Perhitungan LCS

Proses perhitungan *LCS* adalah proses untuk mendapatkan index terpanjang dari dua *string*. Sebagai contoh *string X* = TAAGTTATTA *string Y* = AAACAGTAAA.

Persamaan 2.1 yaitu *F*[*i,j*]*=*0jika *i=0* atau *j=0*. *i* adalah indeks dari *string A* sedangkan *j* adalah indeks dari *string B*. Nilai dari *F*[0,0] hingga *F*[0,10] adalah nol ditunjukkan pada Tabel 2.

Mencari nilai F[7,0] didapatkan dengan menggunakan persamaan 2.2.Sebagai contoh pada Tabel 3 *A*6*=B*6*, A*6*=T* dan *B*6*=T* maka nilai dari *F*[1,7] adalah *F*[0,6]+1 = 0+1=1.

Selanjutnya, mencari nilai *F*[1,2]. Indeks *i=*1 dan *j*=2 maka tidak masuk pada persamaan 2.1. *A0=T* dan *B1*=*A* maka persamaan 2.2 tidak dapat digunakan*.* Terakhir, nilai *F*[1,2] didapatkan dengan menggunakan persamaan 2.3. Kita ambil nilai maksimal dari *F*[1,1] dan *F*[0,2] yaitu nol maka nilai dari *F*[1,2]=0.

Tabel 2. Tabel Potongan Perhitungan LCS (a)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *F* | *Y* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| *X* |  | A | A | A | C | A | G | T | A | A | A |  |
| 0 | T | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Tabel 3. Tabel Potongan Perhitungan LCS (b)

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *F* | *Y* | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| *X* |  | A | A | A | C | A | G | T | A | A | A |  |
| 0 | T | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |

Tabel 4. Tabel dengan Indeks Khusus

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 proses | | | | | *alpha*=0.125 | | | | | | | | | | | |
| *L* | | *Y* | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 | 6 | 7 | 7 | 8 | 9 | 10 |
| 0 | 1 | 2 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 0 | 1 | 2 | 3 |
| *k* | *X* |  | A | A | A |  | C | A | G |  | T | A |  | A | A |  |
| 1 | 0 | T | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | A | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| Proses | | | *P*0 | | | *P*1 | | | | *P*2 | | | *P*3 | | | |

## Proses Menghitung LCS dengan Indeks Khusus

Seperti yang telah dijelaskan pada bahwa pada bagian kolom terdapat dua indeks yaitu kolom yang berwarna  dan berwarna  yang ditunjukkan pada Tabel 4. Kolom yang berwarna merupakan indeks asli dari *string B* sedangkan kolom yang berwarna  merupakan kolom yang menyesuaikan dengan ukuran kolom pada *vector* *L* yaitu (*n*/*p*) + 2.

Pada proses perhitungan *LCS*, penyesuaiannya dilakukan pada penyimpanan nilai indeks *LCS*. Sebagai contoh, pada proses kedua, nilai indeks kolom yang digunakan untuk menyimpan adalah kolom berwarna . Normalnya, kolom yang digunakan adalah kolom berwarna . Proses kedua memproses kolom 2 sampai kolom 5 seperti yang ditunjukkan Tabel 4. Normalnya untuk menyimpan nilai misalnya *L*[0][5] tetapi karena menyesuaikan dengan ukuran kolom *L* yaitu (*n*/*p*) + 2 maka nilai tersebut disimpan di *L*[0][3]. Nilai indeks kolom 3 didapat dari nilai indeks kolom berwarna yaitu 5 dikurang dengan nilai indeks kolom berwarna paling pertama yaitu 2.

## Proses Mencetak LCS

Proses mencetak *LCS* dilakukan untuk membangun dan menghasilkan *LCS* dari dua *string* yang telah dihitung panjang *LCS*nya [6]. Proses ini dimulai dari proses paling terakhir, sebagai contoh pada Tabel 5 terdapat 4 proses dengan menggunakan *alpha*=0.125.

*L*[10][10] bernilai 6 yang berarti panjang *LCS*nya adalah 6. Dimulai dari *L*[10][10], jika karakter pada *X*[*i*-1] dan *Y*[*originalColumn*-1] sama maka dimasukkan karakter tersebut bagian dari *LCS*. Jika tidak sama, nilai dari *L*[*i*-1][*j*] dan *L*[*i*][*j*-1] dibandingkan. Berpindah ke kiri jika nilai *L*[*i*-1][*j*] lebih besar dari *L*[*i*][*j*-1] selain itu berpindah ke atas.

Tabel 5. Proses mencetak LCS

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| rank 4 | | | | *alpha*=0.125 | | | | | | | | | | | | | |
| *L* | | *Y* | 0 | 1 | 2 | 2 | 3 | 4 | 5 | 5 | 6 | 7 | 7 | 8 | 9 | 10 |
| 0 | 1 | 2 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 0 | 1 | 2 | 3 |
| *k* | *X* |  | A | A | A | A | C | A | G | G | T | A | A | A | A |  |
| 1 | 0 | T | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 4 | 1 | A | 0 | 0 | **0** | **0** | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 2 | A | 0 | 1 | 1 | 1 | **1** | **1** | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2 |
| 10 | 3 | G | 0 | 1 | 2 | 2 | 2 | 2 | **2** | **2** | 2 | 2 | 2 | 2 | 3 | 3 |
| 13 | 4 | T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | **3** | 3 | 3 | 3 | 3 | 3 |
| 16 | 5 | T | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | **3** | 4 | 4 | 4 | 4 | 4 |
| 20 | 6 | A | 0 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | **4** | **4** | 4 | 4 | 4 |
| 23 | 7 | T | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | **5** | 5 | 5 |
| 26 | 8 | T | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | **5** | 5 | 5 |
| 29 | 9 | A | 0 | 1 | 2 | 2 | 3 | 3 | 3 | 3 | 3 | 4 | 4 | **5** | **5** | 5 |
| 32 | 10 |  | 0 | 1 | 2 | 2 | 3 | 3 | 4 | 4 | 4 | 4 | 4 | 5 | 6 | 6 |
| Proses | | | *P*0 | | | *P*1 | | | | *P*2 | | | *P*3 | | | | |

Tiap proses mempunyai nilai *vector* *L* yang berbeda maka dalam mencetak *LCS* juga ada mekanisme tersendiri. Kolom yang diproses dibatasi sesuai dengan kolom yang diproses tiap proses. Semisal, proses *P*3 hanya memproses dari kolom ketujuh hingga kolom kesepuluh. Karakter *LCS* yang ditemukan disimpan pada suatu array semisal *LCS* dan dikirim ke proses *P*0. Nilai dari *L*[10][10] yaitu 6 dikirim ke proses *P*0 untuk mengalokasikan karakter – karakter yang akan digabung dari beberapa proses.

Indeks awal dan akhir dari karakter yang disimpan serta karakternya di setiap proses dikirim ke proses *P*0. Proses *P*3 mengirim indeks awal yaitu 5, indeks akhir yaitu 4, dan karakter *LCS*nya yaitu A dan A ke proses *P*1. Baris, kolom, indeks, dan kolom asli yang terakhir dikirim ke proses sebelumnya, semisal dari proses *P*4 mengirim ke proses *P*3, proses *P*3 mengirim ke proses *P*2. Proses *P*4 mengirim baris terakhir yaitu 6, kolom terakhir yaitu 0, kolom asli terakhir yaitu 7 serta indeks *LCS* terakhir yaitu 4 ke proses *P*3.

## Proses Mencetak LCS dengan Indeks Khusus

Sama halnya dengan proses perhitungan *LCS* dengan indeks khusus, proses mencetak *LCS* juga menggunakan indeks khusus. Pada proses mencetak *LCS* penyesuaian dilakukan pada penyocokan karakter *LCS* dimana indeks yang disesuaikan adalah kolomnya. Untuk menyesuaikan ke indeks yang asli, kolom yang berwarna  ditambah dengan kolom paling terakhir pada indeks yang asli yang ada diproses sebelumnya. Sebagai contoh, pada proses ketiga, huruf T berada pada indeks asli ke 6, maka dari itu indeks 1 yang berwarna ditambah dengan indeks asli pada kolom terakhir diproses dua yaitu 5 seperti yang ditunjukkan pada Tabel 5.

# UJI COBA

## Skenario Uji Coba 1

Skenario uji coba 1 adalah dengan melakukan perhitungan tes waktu eksekusi dan panjang maksimal karakter yang dapat diproses di tiap komputer menggunakan program *LCS* secara sekuensial. Selanjutnya dilakukan perhitungan tes waktu eksekusi untuk tiap komputer dengan proses yang berbeda proses (2, 4, 10, 20, 30, 40, dan 50 proses) dengan panjang karakter maksimal yang didapat pada proses sekuensial. Hasil uji coba dapat dilihat pada Gambar 2 dan Gambar 3.

Gambar 2. Waktu Eksekusi Sekuensial Tiap Komputer

Gambar 3. Waktu Eksekusi Paralel Terhadap Banyak Proses

## Skenario Uji Coba 2

Skenario perhitungan tes waktu eksekusi program *LCS* secara sekuensial menggunakan satu komputer. Hal ini telah dilakukan di uji coba skenario 1. Selanjutnya melakukan eksekusi untuk program *LCS* secara paralel menggunakan 13 komputer dan banyak proses proses (9, 18, 27, 36, 45 dan 54 proses). Panjang *dataset* disesuaikan dengan hasil panjang maksimal di tiap komputer pada skenario uji coba 1. Panjang karakter yang digunakan dari hasil uji coba skenario 1 yaitu 72.660, 52.150, 45.080, 40.950, 38.220, 36.820, 28.280, dan 28.210 karakter. Hasil uji coba dapat dilihat pada Gambar 4.

## Skenario Uji Coba 3

Skenario uji coba 3 adalah melakukan tes waktu eksekusi program secara paralel menggunakan panjang karakter yang berbeda. Skenario uji coba 3 ini juga dilakukan untuk mengetahui panjang karakter maksimal yang dapat diproses. Program *LCS* paralel dieksekusi menggunakan 13 komputer dengan 108 proses. Panjang karakter yang digunakan yaitu 4.830, 25.830, 46.830, 67.830, 88.830, 109.830, dan 130.830 karakter. Hasil uji coba dapat dilihat pada Gambar 5.

## Skenario Uji Coba 4

Skenario uji coba 4 yaitu melakukan tes waktu eksekusi untuk mengetahui panjang karakter ketika metode paralel lebih lambat dari sekuensial. Pada skenario uji coba 2a, panjang karakter paling sedikit yaitu 28.210 karakter. Dengan 28.210 karakter, hasil uji coba tersebut menunjukkan bahwa paralel masih lebih cepat dari sekuensial maka dari itu panjang karakter di skenario uji coba 2e ini dibatasi hingga 28.000 karakter. Panjang karakter yang digunakan yaitu 210, 350, 700, 1.050, 1.750, 3.500, 7.000, 10.500, 14.000, 17.500, 21.000, 24.500, dan 28.000 karakter. Hasil uji coba dapat dilihat pada Gambar 6.

Gambar 4. Grafik Hasil Waktu Eksekusi Paralel Banyak Komputer dan Proses

Gambar 5. Grafik Hasil Waktu Eksekusi Paralel dengan 108 Proses

Gambar 6. Grafik Hasil Uji Coba 2e Waktu Eksekusi Terhadap Panjang Karakter

Tabel 6. Hasil Keluaran Program LCS

|  |  |
| --- | --- |
| Metode | Hasil Keluaran |
| Manual | AAGTAAAGTTAATATTTTAACAAATTTTAAATTAA |
| Paralel *LCS* | AAGTAAAGTTAATATTTTAACAAATTTTAAATTAA |

## Skenario Uji Coba 5

Skenario uji coba 5 yaitu melakukan uji coba fungsionalitas terhadap algoritma paralel *LCS*. Hasil keluaran dari program paralel *LCS* akan dibandingkan dengan keluaran *LCS* dari perhitungan manual. Panjang karakter yang digunakan yaitu 50 karakter. Hasil keluaran secara manual dan dari program paralel *LCS* dapat dilihat pada Tabel 6.

# KESIMPULAN/RINGKASAN

Dari hasil uji coba yang telah dilakukan dapat diambil beberapa kesimpulan sebagai berikut:

1. Waktu eksekusi paralel LCS lebih cepat dari sekuensial ketika panjang karakter dimulai dari 3.500 karakter pada semua proses (27, 54, 81, dan 108 proses).

2. Semakin banyak proses maka waktu eksekusi paralel *LCS* pada satu komputer semakin lama.

3. Penggunaan 13 komputer belum dapat mengeksekusi seluruh karakter pada dataset dengan masing – masing panjang karakter yaitu 777.079 dan 580.076 karakter. Maksimal karakter yang dapat dieksekusi yaitu 130.830 karakter.

4. Waktu eksekusi sekuensial lebih cepat ketika program *LCS* dijalankan dengan banyak proses pada satu komputer.

5. Keluaran dari algoritma *LCS* paralel terbukti benar dengan membandingkan hasil keluaran program *LCS* paralel dengan perhitungan manual.

DAFTAR PUSTAKA

|  |  |
| --- | --- |
| [1] | N. F. Almeida Jr, E. N. Caceres, C. E. R. Alves dan S. W. Song, “Comparison of Genomes using High-Performance Parallel Computing,” dalam *Proceedings of the 15th Symposium on Computer Architecture and High Performance Computing*, 2003. |
| [2] | D. Abraham dan N. S. Raj, “Approximate String Matching Algorithm for Phishing Detection,” dalam *International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, New Delhi, 2014. |
| [3] | M. Elhadi dan A. Al-Tobi, “Duplicate Detection in Documents and WebPages using Improved Longest Common Subsequence and Documents Syntactical Structures,” dalam *Fourth International Conference on Computer Sciences and Convergence Information Technology*, Seoul, 2009. |
| [4] | A. Grama, A. Gupta, G. Karypis dan V. Kumar, Introduction to Parallel Computing, Second Edition, Addison Wesley, 2003, p. 534. |
| [5] | C. E. R. Alves, E. N. Cáceres, F. Dehne dan S. W. Song, “A Parallel Wavefront Algorithm for Efficient,” dalam *International Conference on Computational Science and Applications 2003*, Canada, 2003. |
| [6] | GeeksforGeeks, “Printing Longest Common Subsequence,” [Online]. Available: http://www.geeksforgeeks.org/printing-longest-common-subsequence/. [Diakses 11 May 2016]. |
| [7] | “Genetics Home Reference,” National Library of Medicine (US), 14 June 2016. [Online]. Available: https://ghr.nlm.nih.gov/primer/hgp/genome. [Diakses 16 June 2016]. |
| [8] | “What is DNA?,” National Library Medicine (US), 14 June 2016. [Online]. Available: http://ghr.nlm.nih.gov/handbook/basics/dna. [Diakses 15 June 2016]. |
| [9] | “What is a gene?,” National Library of Medicine (US), 14 June 2016. [Online]. Available: https://ghr.nlm.nih.gov/primer/basics/gene. [Diakses 15 June 2016]. |
| [10] | “What is Heredity?,” Genetic Science Learning Center, 22 June 2014. [Online]. Available: http://learn.genetics.utah.edu/content/inheritance/intro/. [Diakses 15 June 2016]. |
| [11] | “NCBI National Center for Biotechnology Information,” [Online]. Available: http://www.ncbi.nlm.nih.gov/. [Diakses 21 June 2015]. |