PEMROSESAN GRAF BERSKALA BESAR SECARA PARALEL (PARALLEL PROCESSING OF LARGE SCALE GRAPH) Ufra

Neshia[1], ZK Abdurahman Baizal[2], Izzatul Ummah[3]

[1]Ilmu Komputasi, Fakultas Informatika, Universitas Telkom

[1]ufraneshia94@gmail.com, [2]bayzal@gmail.com, [3]izzatul.ummah@gmail.com

Abstrak

Masalah pencarian rute terpendek untuk graf statis berskala besar dapat dilakukan dengan menggunakan algoritma yang dijalankan secara paralel. Dua algoritma yang dapat digunakan adalah algoritma Djikstra yang bersifat greedy dan algoritma Bellman-Ford yang menggunakan dynamic programming. Dijkstra lebih sulit diparalelkan namun memiliki waktu eksekusi yang relatif lebih cepat dibandingkan Bellman-Ford yang mudah diparalelkan namun waktu eksekusinya lebih panjang. Optimasi kedua algoritma dilakukan dengan menyimpan data graf dalam bentu Compact Spare Row, dan mendesain algoritma agar membagi data antar prosesor dengan efisien dan meminimalisir komunikasi antar prosesor. **Kata Kunci**: paralel, shortest path, graf, algoritma Djikstra, algoritma Bellman-Ford, performansi.

1. Pendahuluan

Dewasa ini komputasi, termasuk bidang komputasi paralel, terus berkembang dengan pesat. Hal ini menjadikan algoritma yang berjalan secara paralel juga makin banyak digunakan. Algoritma paralel biasa digunakan untuk mengolah data dalam jumlah yang sangat banyak, sehingga tidak efektif jika dilakukan secara seri.

Graf adalah masalah yang banyak dibahas dalam bidang komputasi. Graf dapat digunakan untuk menggambarkan hubungan antar sistem dalam jumlah besar. Ada banyak sistem yang digambarkan dalam graf dengan jutaan simpul, sehingga lebih cocok dikerjakan secara paralel daripada sekuensial.

Salah satu masalah klasik graf adalah "Shortest Path Problem", yaitu menentukan rute terpendek antara dua simpul dalam suatu graf yang saling berhubungan. Aplikasi dari pemecahan masalah ini sangatlah luas, misalnya perancangan rute transportasi yang efisien, perancangan suatu jaringan komunikasi, dan lain sebagainya. Untuk memecahkan masalah rute terpendek dalam graf, dapat dilakukan dengan bantuan komputer. Ada banyak algoritma yang dirancang untuk keperluan ini, seperti algoritma Djikstra (Djikstra, 1959) dan algoritma Bellman-Ford (Bellman, 1958). Keduanya termasuk algoritma rute terpendek yang paling sederhana dan paling tua, sehingga dewasa ini diimplementasikan dengan banyak perubahan dan versi.

Kedua algoritma ini dibangun dengan prinsip komputasi serial. Dengan kecepatan komputer yang sekarang, komputer dengan spesifikasi rendah sekalipun dapat menjalankan kedua algoritma dengan sangat baik. Namun masalah muncul ketika graf yang menjadi permasalahan bertambah jumlah simpul dan sisinya, hingga mencapai jutaan bahkan lebih simpul. Semakin besar graf yang diproses semakin lama pula waktu komputasi yang dibutuhkan sehingga menjadi tidak efisien. Oleh karena itu muncul ide untuk memodifikasi kedua algoritma agar dapat dijalankan secara paralel, dengan harapan dapat memangkas waktu komputasi yang dibutuhkan.

Setelah kedua algoritma diimplementasikan secara paralel, kemudian akan dianalisis performansinya, agar diketahui apakah performansi implementasi secara paralel lebih baik daripada implementasi secara serial, untuk pemrosesan graf berskala besar.

2. Dasar Teori

2.1. Algoritma Paralel

Komputasi paralel adalah penggunaan lebih dari satu sumber daya komputasi seperti prosesor dalam waktu bersamaan untuk memecahkan suatu masalah. Masalah yang hendak dipecahkan secara paralel akan dibagi menjadi beberapa sub masalah, dimana beberapa sub masalah dikerjakan secara sekaligus pada beberapa prosesor yang berbeda (Barney, 2007).

Langkah pertama untuk melakukan komputasi secara paralel adalah mendesain pemrograman paralel yang dibutuhkan. Dalam bukunya yang berjudul "Parallel Programming for Multicore and Cluster System" , Rauber dan Runger menuliskan:

"Langkah pertama dalam melakukan pemrograman paralel adalah mendesain algoritma paralel untuk masalah yang diberikan. Proses desain dimulai dengan membagi proses komputasi yang dibutuhkan menjadi beberapa bagian, yang disebut task, yang kemudian akan dikomputasikan dalam core atau prosesor dalam hardware paralel. Proses pembagian task bisa menjadi sangat rumit karena ada banyak kemungkinan pembagian untuk satu algoritma yang sama" (Rauber, Runger 2007)

Untuk dapat melihat perbedaan program paralel dan sekuensial, perlu dilakukan analisis performansi. Dua teknik yang paling sering dipakai adalah perhitungan speed up dan perhitungan efektifitas. Speed up ada perbandingan waktu eksekusi program paralel relatif terhadap program serialnya. Rumus speed up $S(n)$ dari suatu program dengan waktu eksekusi paralel $T(n)$ dan waktu eksekusi serial $T(n)$ adalah:

$$S(n) = \frac{T(n)}{T(n)}$$

(Rauber dan Runger, 2007)

2.2. *Masalah Jalur Terpendek (Shortest Path Problem)*

Bedasarkan artikel yang ditulis oleh Alexander Schrijver pada tahun 2012, sejarah tentang pencarian jalur terpendek sudah sangat tua, namun riset matematis dalam bidang ini masih bisa dikatakan lebih baru dari pada bidang lain, dikarenakan permasalahan dan pemecahannya yang relatif sederhana. Hal ini terlihat dengan ditemukannya teknik pemecahan yang sama oleh ilmuan yang berbeda dalam waktu nyaris bersamaan. Beberapa referensi klasik misalnya Wiener (1873), Lucas (1882) and Tarry (1895). Pemecahan yang lebih modern mulai muncul pada tahun 1950-an, dan masih berlaku hingga sekarang dalam bentuk algoritma jalur terpendek, diantaranya algoritma Djikstra dan Bellman-Ford.

2.3. Algoritma Dijkstra

Algoritma ini diperkenalkan oleh E.W. Djikstra pada tahun 1959 dalam makalahnya yang berjudul "A Note on Two Problems in Connexion with Graphs". Ada dua masalah yang diperkenalkan dalam makalah tersebut, yang pertama adalah mencari jalur terpendek yang ada di antara n buah simpul dan yang kedua adalah mencari jalur terpendek dari satu simpul ke simpul lain.

Pseudocode algoritma Djikstra:

```
DJIKSTRA (G,w,S)
        INITIALIZE _SINGLE _SOURCE (G,S)
        S=Ø
         Q = G.V
        while Q≠Ø
                u=EXTRACT_MIN (Q)
                S=S U {u}
                for each vertex v ∈ G.adj[u]
                        Relax(u,v,w)
```

Sumber buku "Introduction to Algorithms" oleh Cormen dkk.

2.4 Algoritma Bellman-Ford

Diperkenalkan oleh Richard Bellman pada 1958, pada tahun yang hampir bersamaan Lester R. Ford Jr mengemukakan algoritma yang sama sehingga algoritma ini pada akhirnya dikenal dengan nama Bellman-Ford. Melakukan relaksasi setiap sisi sebanyak |V|-1 kali.

Pseudocode Bellman-Ford:

```
BELLMAN-FORD (G,w,S)
        INITIALIZE -SINGLE-SOURCE (G,S)
        for i=1 to |G.V|-1
                for each edge (u,v) ∈  G.E
                        RELAX (u,v,w)
        for each edge (u,v) ∈ G.E
                if v.d > u.d+w(u,v)
                        return FALSE
                return TRUE
```

3. Perancangan Sistem

Pada tugas akhir ini, dirancang dua buah algoritma parallel bedasarkan algoritma shortest path yang sudah ada, yaitu Dijkstra dan Bellman-Ford. Algoritma dirancang untuk diimplementasikan pada graf statis berskala besar yang tidak mengandung *negative cycle*. Untuk menambah efektifitas algoritma, data graf disimpan dalam format Compact Sparse Row diimplementasikan menggunakan array dinamis. Implementasi algoritma menggunakan bahasa C dengan *middleware* MPI. Algoritma Dijkstra diparalelkan dengan membagi array vertex Q dan array jarak minimum sementara d[] ke beberapa prosesor, yang mana masing-masing prosesor akan mengerjakan algoritma Dijkstra serial pada data yang menjad bagiannya, dengan tetap melakukan komunikasi per iterasi untuk memastikan data yang dimilikinya memiliki jarak (*distance*) minimum. Algoritma Bellman-Ford akan diparalelkan dengan membagi data graf ke beberapa prosesor, dan setiap prosesor akan melakukan algoritma Bellman-Ford serial menggunakan data sisi graf yang menjadi bagiannya. Pada setiap iterasi juga akan dilakukan komunikasi antar prosesor untuk menjamin keakuratan hasil.

4. Pengujian dan Analisis

4.1. Implementasi dan Optimasi Paralel Algoritma Dijkstra

Algoritma Dijkstra mencari jalur terpendek dari satu simpul ke semua simpul lain dalam suatu graf terarah tanpa *negative cycle.* Langkah-langkah algoritma Dijkstra:

1. Memberikan nilai prediksi jarak terpendek d(v) untuk semua simpul
2. Memilih simpul dengan d(v) terpendek
3. Melakukan relaksasi untuk semua simpul yang bertetangga dengan simpul terpilih
4. Menandai simpul yang sudah terpilih
5. Mengulangi 1-4 hingga seluruh simpul diberi tanda.

Ada dua data yang digunakan dalam algoritma ini, yaitu data graf dan data hasil. Untuk optimasi, data graf disimpan ke dalam bentuk CSR (Column Sparse Row), yaitu bentuk kompresi sederhana dari matriks jarang. Tipe data CSR digunakan untuk memudahkan akses ke setiap bagian data graf dan memperkecil memori yang digunakan bila dibandingkan dengan menggunakan matriks biasa. Data hasil terdiri atas dua buah array dinamis, yaitu d[ ] dan predecessor[ ], data ini merupakan output dari algoritma dan akan melalui tahap relaksasi pada setiap iterasi.

Untuk algoritma parallel, pertama dilakukan pembagian data. Prosesor selain prosesor 0 (root) akan memiliki satu buah kopi lengkap dari graf data, serta satu buah partisi dari data hasil. Data hasil memiliki panjang N. Dengan jumlah prosesor P, setiap prosesor akan menyimpan N/P bagian data hasil. Setelahnya dimulai algoritma parallel:

0. Inisialisasi: mengkopi data graf ke semua prosesor dan membagi data hasil.
1. Setiap prosesor selain prosesor 0 memilih simpul dengan d(v) terpendek.
2. Simpul terpilih lokal dikirim ke prosesor 0, dipilih simpul d(v) terpendek, hasilnya dikirim lagi ke semua prosesor.
3. Melakukan relaksasi ke semua simpul yang bertetangga.
4. Simpul terpilih ditandai
5. Kembali ke langkah 1 hingga semua simpul lokal diberi tanda
6. Kirim semua partisi data hasil ke prosesor 0. Gabung menjadi data utuh.

Pseudocode Algoritma Dijkstra Paralel:

Input: *Graf berarah CSR, simpul source S, banyak prosesor SIZE, rank prosesor RANK, list simpul Q*
output: *data hasil RESULT*

```
/*Inisialisasi*/
if (RANK!=0)
        localRESULT<-partisi(RANK)
        localCSR<-CSR
        foreach v ∈ V do
                localRESULT.d[v]<-INF
                localRESULT.predecessor[v]<-NIL
        end
        localQ<-partisi(Q,RANK)
else
        RESULT globalRESULT
        foreach v ∈ V do
```

```
                globalRESULT.d[v]<-INF
                globalRESULT.predecessor[v]<-NIL
        end

        globalSHORTEST<-S dGlobalShortest<-0
        sendToSlaves(globalSHORTEST,dGlobalSHORTEST)

endif


/*Bagian Utama*/
while ALL localQ not EMPTY do
        if (RANK!=0)
                        receiveFromMaster(globalSHORTEST, dGlobalSHORTEST)
                        relax(localRESULT,localCSR,globalSHORTEST, dGLOBALSHORTEST)
                        decreaseIfExist(localQ,globalSHORTEST)
                        localSHORTEST<-extractMin(localQ,localRESULT)
                        sendToMaster(localShortest,localResult.d[localShortest])
        else
                        receiveFromSlaves(shortest[],dShortest[])
                        globalShortest<-extractMin(shortest[],dShortest[])
                        sendToSlaves(globalSHORTEST,dGlobalSHORTEST)
        endif
endwhile


if (RANK!=0)
        sendToMaster(localResult)
else
        receiveFromSlave(RESULT[])
        globalRESULT=combineResult(RESULT[])
endif
end
```

Dengan algoritma ini, pengiriman data antar prosesor dapat diminimalisir, dengan hanya melakukan pengiriman 2 integer per iterasi dan satu pengiriman array di akhir iterasi. Setiap prosesor slave membutuhkan kopi utuh dari data graf, agar mampu memperoleh data yang dibutuhkan untuk proses relaksasi tanpa menunggu komunikasi dari prosesor lain.

Prosesor yang bekerja terbagi atas satu prosesor master, biasanya prosesor 0, yang bertugas menerima data simpul terpendek lokal dari prosesor lain dan mengirim kembali data simpul terpendek global pada setiap iterasi dan kemudian mengumpulkan kembali partisi-partisi data hasil dari prosesor lain untuk digabungkan menjadi data utuh. Sisa prosesor, disebut slave, adalah prosesor selain master. Prosesor-prosesor slave adalah prosesor yang mengerjakan algoritma Dijkstra serial pada setiap partisi data yang dimilikinya. Karena pembagian tugas yang demikian, maka untuk dapat benar-benar bekerja secara parallel algoritma tersebut minimal harus dijalankan menggunakan 3 prosesor, 1 master dan 2 atau lebih slave.

4.2. Implementasi dan Optimasi Paralel Algoritma Bellman-Ford

Algoritma Bellman-Ford melakukan pencarian jalur terpendek dengan cara melakukan |V| x |E| kali relaksasi untuk semua simpul. Hal ini menjadikan Bellman-Ford jauh lebih lambat relatif dengan waktu eksekusi Dijkstra namun bagian relaksasi algoritma ini dapat diparalelkan dengan lebih mudah dan efektif. Langkah-langkah algoritma Bellman-Ford serial:

1. Memberikan nilai prediksi jarak terpendek d[v] untuk semua simpul
2. Untuk setiap sisi E(u,v), lakukan relaksasi pada list d[v]
3. Ulangi langkah 2 sebanyak jumlah sisi |V|-1

Sama seperti algoritma Dijkstra, pada algoritma Bellman-Ford dilakukan optimasi tidak langsung dengan cara menyimpan data graf dalam bentuk CSR dan menggabungkan array d[] dan predecessor[] dalam bentuk tipe data baru hasil. Optmasi langsung pada badan algoritma adalah dengan cara membagi data graf untuk diproses oleh masing-masing prosesor.

Pada algoritma parallel Bellman-Ford, prosesor-prosesor yang digunakan terbagi atas satu prosesor master dan sisanya prosesor slave. Setiap prosesor slave menerima sebagian data graf dalam bentuk partial CSR dan satu kopi utuh data hasil. Pembagian data graf bedasarkan jumlah sisi yang dimiliki, di mana setiap prosesor slave menyimpan sejumlah |E|/(P-1) sisi. Berikut langkah-langkah algoritma parallel:

0. Inisialisasi: untuk semua slave dan master, buat satu buah kopi tipe data hasil sepanjang |N| simpul dan inisialisasi. Untuk setiap slave, lakukan pembagian data graf.
1. Untuk setiap slave, lakukan relaksasi data hasil sebanyak |E|/P kali. Kirim data hasil setelah relaksasi ke master.
2. Master akan memerima data terkirim dan membandingkan setiap d[v] pada data hasil, d[v] yang terkecil akan dikopi ke data hasil lokal milik master. Setelah selesai, data hasil master, yang sudah merupakan data hasil dengan d[v] minimum global, dikirim kembali ke seluruh slave
3. Slave menerima data terkirim dan mengupdate data hasil local.
4. Ulangi langkah 1-3 sebanyak |N| kali.

Setelah iterasi selesai, setiap master dan slave akan memiliki data hasil lokal dengan d[v] terkecil.

Pseudocode Algoritma Bellman-Ford Paralel:

```
Input: Graf berarah CSR, simpul source S,  banyak prosesor SIZE, rank prosesor RANK,
 output:  data hasil RESULT
 /* Inisialisasi */

 lokalResult<-createResult;
 source<-S
 foreach v ∈ V do
        lokalResult.d[v] <-INF
        lokalResult.predecessor[v] <-NIL
 endfor
 lokalResult.d[source] <-0

 if (RANK!=0)
        lokalCSR<-partisi(CSR,RANK)
 else
        sendToSlaves(lokalResult)
 endif


 foreach v ∈ V do
        if (RANK!=0)
                receiveFromMaster(lokalResult)
                relax(lokalResult,lokalCSR)
                sendToMaster(lokalResult)
        else
                receiveFromSlaves(arrayLokalResult[])
                lokalResult<-getMinDistance(arrayLokalResult[])
                sendToSlaves(lokalResult)
        endif
 endfor
```

4.3. Pengujian Algoritma Dijkstra dan Bellman-Ford Secara Serial dan Paralel

Hasil implementasi algoritma serial dan parallel Dijkstra dan Bellman-Ford diujicobakan pada komputer dengan sistem operasi Windows, menggunakan data graf berskala besar yang disimpan dalam file .txt. Algoritma parallel diujicobakan dengan menggunakan 3 dan 4 prosesor. Hasilnya dirangkum dalam tabel di bawah ini:

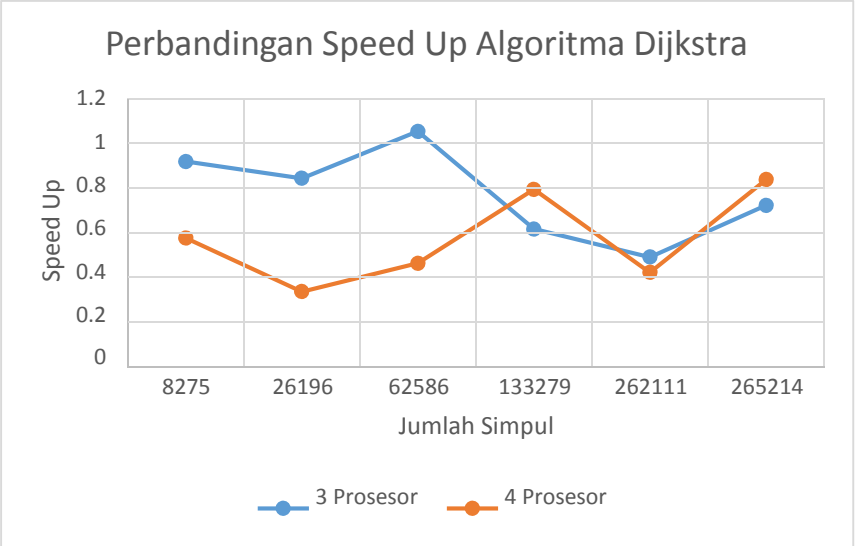
No	Jumlah Simpul	Jumlah Sisi	Waktu Eksekusi (s)		
			Serial	Paralel 3 Prosesor	Paralel 4 Prosesor
1	8275	103689	1.2	0.8	1.3
2	26196	28980	5.68	4.7	6.7
3	62586	147882	32.8	24	31
4	133279	396161	116	154	187
5	262111	1234877	531	637	1073
6	265214	420031	869	916	1196

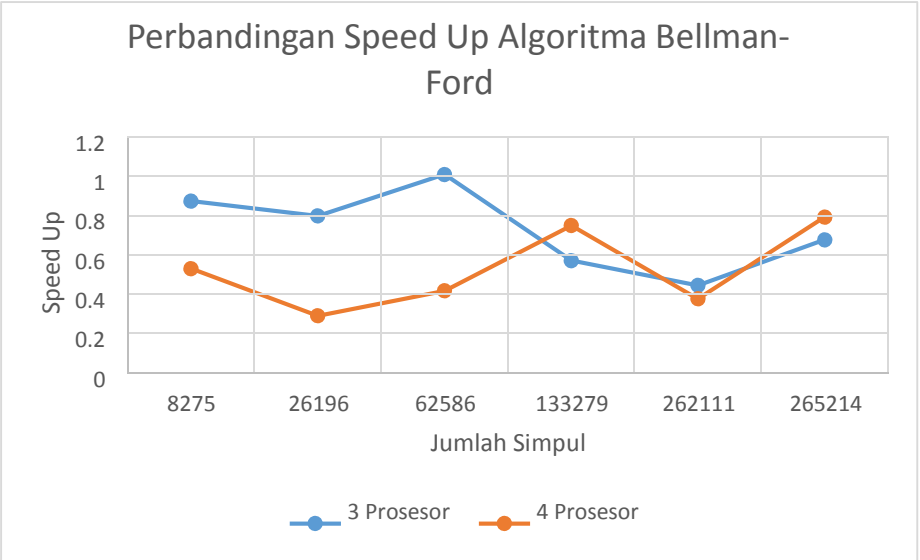
Tabel 4.1 : Data Hasil Eksekusi Algoritma Dijkstra Serial dan Paralel

No	Jumlah Simpul	Jumlah Sisi	Waktu Eksekusi (s)		
			Serial	Paralel 3 Prosesor	Paralel 4 Prosesor
1	8275	103689	6.67	11.5	20.445
2	26196	28980	15.3	45	70
3	62586	147882	119	254.81	502
4	133279	396161	638.9	800.1	1009
5	262111	1234877	3550	8336	10110
6	265214	420031	2610	3100	3970

Tabel 4.2 Data Hasil Eksekusi Algoritma Bellman-Ford Serial dan Paralel

Kedua tabel di atas adalah hasil dari menjalankan program Dijkstra dan Bellman-Ford terhadap 6 set data graf dengan jumlah simpul di bawah 300000 dan jumlah sisi di bawah 1,5 juta. Batasan jumlah simpul dan sisi dikarenakan batasan jumlah data yang dapat dijalankan di komputer yang digunakan untuk eksekusi. Dengan menggunakan hardware yang memiliki spesifikasi lebih besar dimungkinkan untuk menggunakan data yang lebih besar. Secara umum, untuk data yang sama, terlihat bahwa waktu eksekusi algoritma Bellman Ford lebih besar daripada Dijkstra.





Untuk membandingkan kecepatan algoritma serial dengan parallel, dapat digunakan pula rumus Speed Up, yaitu $S(n) = \frac{T^*(n)}{T_p(n)}$, dimana T*(n) adalah waktu ekssekusi serial dan Tp(n) adalah waktu eksekusi parallel dengan p prosesor.

Dari data di atas, terlihat bahwa peningkatan kecepatan kedua algoritma masih kurang efektif, dikarenakan speed up 4 prosesor lebih kecil daripada speedup 3 prosesor, yang menunjukkan bahwa untuk data-data yang telah diujicobakan, peningkatan jumlah prosesor malah membuat algoritma semakin lambat. Diperlukan ujicoba lebih lanjut dengan data dan jumlah prosesor yang lebih besar untuk membuktikan efektifitas dari algoritma parallel yang telah diimplementasikan.

Ketika membandingkan algoritma parallel Dijkstra dengan parallel Bellman didapati perbandingan performansi keduanya masih sama dengan serial, dimana performansi Bellman lebih buruk daripada Dijsktra, diakibatkan jumlah relaksasi yang dilakukan Bellman-Ford jauh lebih banyak daripada Dijkstra.

5. Kesimpulan dan Saran

5.1. Kesimpulan

Bedasarkan implementasi yang telah dilakukan dapat disimpulkan bahwa Bellman-Ford yang bersifat dinamis lebih mudah diparalelkan dibandingkan dengan Dijkstra yang bersifat *greedy*. Namun waktu eksekusi algoritma Bellman-Ford jauh melebihi Dijkstra untuk jumlah simpul yang sama. Kemudian optimasi yang dapat dilakukan untuk algoritma Dijkstra dan Bellman-Ford parallel adalah dengan cara menyimpan data graf dalam bentuk CSR, membagi data yang dibutuhkan prosesor slave secara rata dan meminimasi jumlah komunikasi antar prosesor. Setelah diparalelkan, ternyata tidak terdapat perubahan signifikan antara perbandingan kecepatan algoritma Bellman-Ford dengan Dijkstra, di mana untuk simpul sampai dengan 260.000 simpul kecepatan Bellman Ford masih jauh di bawah Dijkstra parallel.

5.2. Saran

Kedua algoritma dapat memproses data yang lebih besar jika dijalankan di komputer dengan spesifikasi yang lebih tinggi. Kemudian, optimasi yang dilakukan pada Dijkstra dan Bellman-Ford dapat diaplikasikan juga pada algoritma shortest path lain. Kecepatan pada Bellman-Ford dapat ditingkatkan jika dapat dilakukan pengurangan jumlah relaksasi yang dilakukan, dengan tidak mengurangi akurasi algoritma.

Daftar Pustaka

[1] Bader, David A., et al. 2007. *An Experimental Study of a Parallel Shortest Path Algoritm for Solving Large-Scale Graph Instances.* Workshop on Algorithm Engineering and Experiments (ALENEX), New Orleans.
[2] Bader, David A., Madduri, Kamesh. 2006. *Parallel Algorithm for Evaluating Centrality Indices in Real World Network.* IEEE.
[3] Bellman, Richard. 1956. *On a Routing Problem*. Rand Corporation.
[4] Chakaravarthy, Venkatesan T., et al. 2014. *Scalable Single Source Shorest Path Algorithms for Massively Parallel System.* IEEE.

[5] Cormen, Thomas H., et al. 2009. *Introduction to Algoritms Third Edition*. The MIT Press.
[6] Djikstra, E.W. 1959. *A Note on Two Problems in Connexion with Graphs.* Numerische Mathemathik 1, 269-271.
[7] Edmonds, Nick., et al. 2006. *Single Source Shortest Path with Parallel Boost Graph Library*. Indiana University.
[8] Felner, Ariel. 2011. *Position Paper: Djikstra Algorithm versus Uniform Cost Search or a Case Against Djikstra's Algoritm.* Proceeding, The Fourth International Symposium on Combinatorial Search (47-51).
[9] Munir, Rinaldi. 2004. *Bahan Kuliah ke-4 IF2251 Strategi Algoritmik: Algoritma Greedy (Lanjutan)*. Departemen Teknik Informatika Institut Teknologi Bandung. Bandung.
[10] Rauber, Thomas., Runger, Gudula. 2010. *Parallel Programming for Multicore and Cluster System.* New York City. Springer.
[11] Schrijver, Alexander. 2012. *On the History of the Shortest Path Problem*. Dokumenta Mathematica Extra Volume ISMP.