Prosiding Seminar Nasional Teknologi Informasi dan Kedirgantaraan : Transformasi Teknologi untuk Mendukung Ketahanan Nasional, Yogyakarta, 13 Desember 2018
SENATIK 2018, Vol. IV, ISBN 978-602-52742-0-6
DOI: http://dx.doi.org/10.28989/senatik.v4i0.143

PARALEL PROCESSING FOR VERTEX TRIPLE NUMBER SEARCH PROCESSES IN NOT DIFFERENT GRAPH USING SHARED MEMORY MODELS

Astika Ayuningtyas
Program Studi Informatika
Sekolah Tinggi Teknologi Adisutjipto
Jl. Janti, Blok-R, Lanud Adisucipto Yogyakarta
Email : astika@stta.ac.id

Abstract

Parallel processing is an effective strategy for describing large problems in smaller problems, in which computing techniques use two or more computers to perform a task simultaneously by optimizing the resources of existing computer systems. This study uses parallel processing techniques, in which the object of the search is the process of finding the number of triple vertices not represented in the graph. The parallel worker process then divides the workload into each process and distributes it to other threads in the system to solve a problem. The model used in parallel processing is shared memory. This model optimizes the performance of threads on the processor. The experiments carried out make it possible to conclude that the processing speed of a parallel system is influenced by the number of processes and the number of vertices. On a single computer, the larger the process, the shorter the process. The same with the increase in the number of vertices resulting in a longer time.

Keyword: Parallel Processing, Shared Memory, Triple Vertex, Thread

Abstrak

Pemrosesan paralel merupakan salah satu strategi yang efisien yang memungkinkan untuk menguraikan masalah yang besar menjadi lebih kecil, dimana teknik komputasinya menggunakan dua atau lebih komputer untuk menyelesaikan suatu tugas dalam waktu yang simultan dengan cara mengoptimalkan *resource* pada sistem komputer yang ada. Penelitian ini memanfaatkan teknik paralel dalam pemrosesannya, dimana obyek yang akan menjadi fokus penelitian adalah proses pencarian jumlah *triple vertex* pada graph tidak berarah. Proses kerja dari paralel nantinya dengan membagi beban kerja perhitungan setiap proses kali dan mendistribusikannnya pada *thread-thread* lain yang terdapat dalam sistem untuk menyelesaikan suatu masalah. Model yang digunakan pada pemrosesan paralel ini adalah *shared memory*. Model ini memaksimalkan kinerja *thread* yang ada pada prosesor. Berdasarkan uji coba yang dilakukan dapat disimpulkan bahwa kecepatan pemrosesan suatu sistem paralel dipengaruhi oleh jumlah proses dan jumlah verteks. Pada komputer tunggal semakin besar proses maka akan menambahkan sedikit waktu pada pengerjaannya. Begitu pula dengan bertambahnya jumlah verteks mengakibatkan waktu yang diperlukan semakin besar.

Kata Kunci : Pemrosesan Paralel, *Shared Memory, Triple Vertex, Thread*

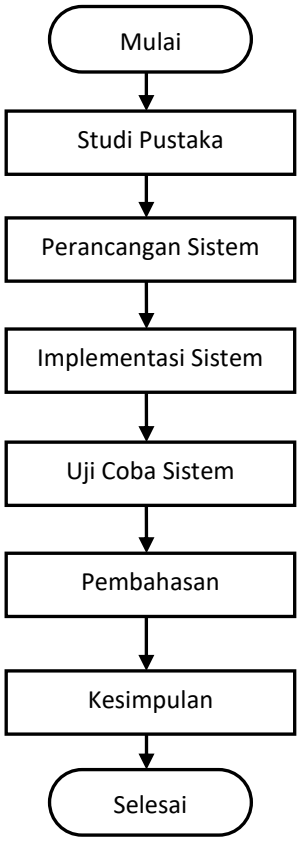
1. Pendahuluan

Pemrosesan paralel merupakan salah satu strategi yang efisien yang memungkinkan untuk menguraikan masalah yang besar menjadi lebih kecil [1], dimana teknik komputasinya menggunakan dua atau lebih komputer untuk menyelesaikan suatu tugas dalam waktu yang simultan dengan cara mengoptimalkan *resource* pada sistem komputer yang ada [2]. Konsep

paralel semakin banyak pekerjaan yang bisa dilakukan secara bersamaan, maka akan semakin cepat pekerjaan tersebut terselesaikan. Seperti contoh kasus pencarian jumlah triple vertex pada graph tidak berarah yang melibatkan operasi logika di dalamnya yaitu operasi perulangan. Jika proses pencarian tersebut sudah melibatkan data yang banyak tentunya akan sangat membutuhkan waktu yang semakin panjang untuk menyelesaikan perhitungannya. Adanya konsep paralel yang membagi pekerjaan besar menjadi lebih kecil dapat menjadi salah satu solusi untuk menyelesaiakn pencarian pada kasus tersebut.

Pada model pemrosesan paralel itu sendiri ada tiga macam, yatu *Network of Workstation*, *Shared Memory*, dan *Graphics Processing Unit* (GPU) [3]. Dari ketiga model tersebut yang akan diimplementasikan untuk kasus perhitungan perkalian matriks dan vektor ini adalah model *shared memory*. Model ini memaksimalkan kinerja *thread* yang ada pada prosesor. Untuk penerapan dari model *shared memory* dalam penelitian ini adalah OpenMP. OpenMP memiliki dua tipe data yaitu data *shared* dan *private* data. Data *shared* dapat diakses oleh semua *thread* dan mempunyai nama yang sama sedangkan *private* data hanya dapat diakses oleh satu *thread* [4] [5]. Pada penelitian yang lain terkait pemrosesan parallel dan *grid computing* telah dibahas [6-9] digunakan sebagai bahan pertimbangan pada penlitian ini.

Penelitian ini diangkat karena peneliti tertarik dengan konsep pembagian beban pada proses yang ditawarkan di teknik komputasi paralel untuk menyelesaiakn kasus pencarian jumlah *triple vertex* pada graph tidak berarah. Penerapan pada model paralel ini menggunakan *shared memory* yaitu OpenMP. Sistem yang dibangun tidak berbasis GUI, sistem ini digunakan untuk menguji apakah konsep paralel yang diterapkan pada pencarian jumlah *triple vertex* di graph tidak berarah menjadi alternatif yang tepat untuk mengatasi pencarian data dengan skala yang lebih besar dibandingkan dengan menggunakan sistem secara sekuensial.



Gambar 1. Diagram Alir Metodologi Peneltian

2. Metodologi Penelitian

Bagan alir menjelaskan tentang metodologi penelitian yang dilakukan dengan langkah-langkah yang terlihat pada Gambar 1.

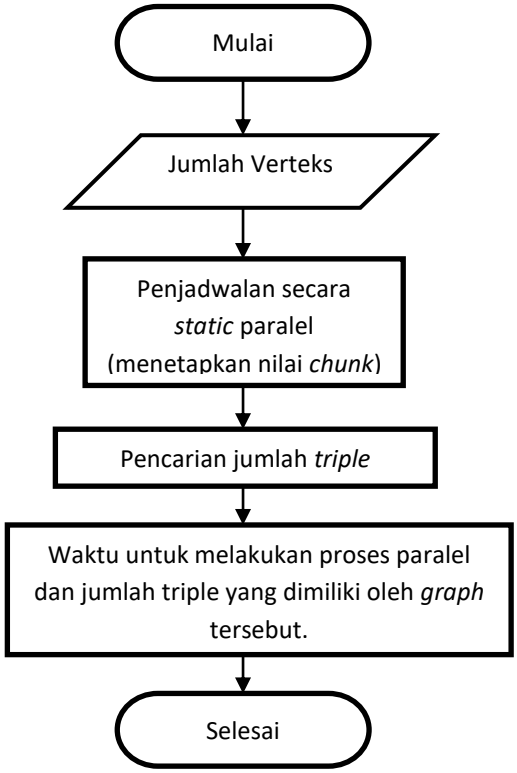
1. Studi Pustaka

   Studi pustaka untuk penelitian ini dilakukan dengan mencari, membaca dan mengumpulkan informasi, dokumen-dokumen serta teori-teori yang mendukung, seperti dari buku teks, artikel, jurnal, literatur-literatur serta *browsing* di internet.

2. Perancangan Sistem

   Pada perancangan sistem, program akan dibuat menggunakan bahasa pemrograman C dengan memanfaatkan *library* OpenMP untuk model pemrograman paralelnya. Masukkan dari program yang dibuat ini berupa argc dan argv yang sering digunakan untuk pemrograman dengan *command line interprater*, dimana masukkan yang diminta adalah jumlah verteksnya. Terdapat dua model program yang digunakan untuk proses pencaian jumlah triple secara paralel, adapun model yang dirancang antara lain:

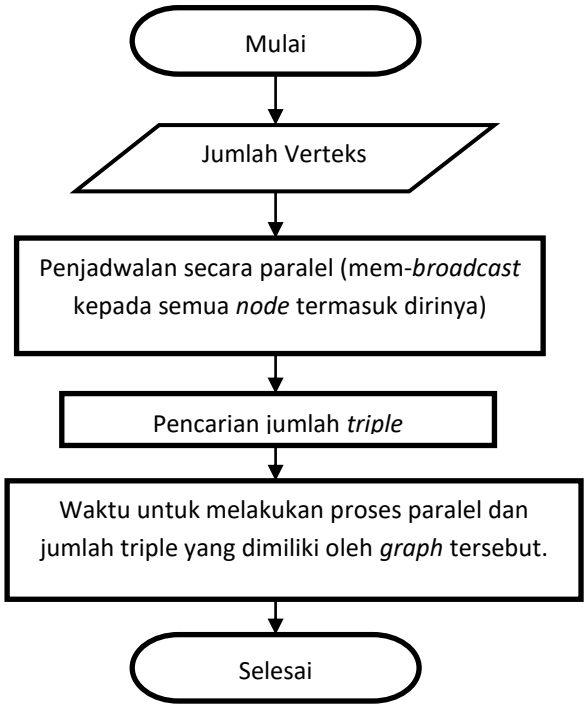
   a. Proses Pencarian *Triple* Model Pertama
      Penerapan untuk mencari *triple* pada *matrix adjacency* tidak berarah pada model pertama dengan menggunakan OpenMP (Gambar 2). Memanfaatkan penggunaan `#pragma omp for schedule` untuk mengatur penjadwalan, dilakukan secara *static* dengan ukuran *chunk* yang disesuaikan dalam beberapa kali percobaan dan memanfaatkan penggunaan `#pragma omp parallel` untuk mengatur penjadwalan dan membagi tugas berdasarkan *thread* yang ada untuk menghitung jumlah *triple* yang terbentuk dari verteks yang diinputkan.



Gambar 2. Diagram Alir Proses Pencarian *Triple* Model Pertama

b. Proses Pencarian *Triple* Model Kedua

Penerapan untuk mencari *triple* pada *matrix adjacency* tidak berarah pada model kedua dengan menggunakan OpenMP (Gambar 3). Memanfaatkan penggunaan `#pragma omp for schedule` untuk mengatur penjadwalan, dilakukan secara *static* dengan ukuran *chunk* yang disesuaikan dalam beberapa kali percobaan dan memanfaatkan penggunaan `#pragma omp parallel` untuk mengatur penjadwalan dengan mem-*broadcast* kepada semua *node* termasuk dirinya, kemudian masing-masing *node* mengerjakan proses yang sama namun dengan nilai berbeda sesuai dengan penomoran ID *node* yang dimiliki. Ini bertujuan untuk menghindari kesamaan nilai yang dikerjakan masing-masing *node*. Setelah itu, untuk mendapatkan total jumlah *triple* dari masing-masing *node* digunakan `#pragma omp critical{ triple+=temp;}`.



Gambar 3. Diagram Alir Proses Pencarian *Triple* Model Kedua

3. Implementasi Sistem

Dalam melakukan penelitian ini dibangun sebuah sistem yang menerapkan pemrosesan paralel untuk proses pencarian jumlah *triple vertex* pada *graph* tidak berarah menggunakan model *shared memory,* dimana sebuah sistem dibutuhkan *hardware* dan *software* yang dapat menunjang perancangan sistem.

1. Spesifikasi *Hardware*

Hardware (perangkat keras) merupakan perangkat secara fisik ada, dapat dilihat dan dipegang. Sistem perangkat keras secara fungsional terdiri dari *input, process, output* dan *memory*. Adapun spesifikasi *hardware* yang digunakan dalam pengaplikasian sistem ini, yaitu:

1. Intel (R) Core(TM) i3-5005U CPU @ 2,00GHz (4CPUs)
2. 12288 MB RAM
3. 500 GB Hardisk

4. CD-ROM drive
5. VGA (*higher resolution* monitor)
6. Keyboard, Mouse Standar
7. Monitor 14.1”

2. Spesifikasi *Software*

   Software (perangkat lunak) merupakan perangkat yang sifatnya abstrak yang berisi instruksi, program, prosedur, pengendali, pendukung dan aktivitas-aktivitas pengolahan perintah pada sistem komputer. Adapun spesifikasi minumum *software* yang dibutuhkan dalam pengaplikasian sistem ini, yaitu:
   1. Sistem Operasi Windows 10 Pro-64bit
   2. Bahasa Pemrograman C/C++.
   3. Pustaka OpenMP.

4. Uji Coba Sistem

   Setelah program dengan dua model jadi, maka dilakukan pengujian. Pengujian dilakukan pada masing-masing model paralel yang telah sebelumnya dirancang. Data masukkan yang dilakukan untuk pengujian bervariasi dengan harapan dapat melihat seberapa efektif dan efisienkah model pemrograman paralel yang telah diterapkan untuk melakukan pencarian jumlah *triple vertex* pada graph yang tidak berarah.

5. Pembahasan

   Hasil dari uji coba sistem yang dibangun akan diuraikan pada bagian ini untuk tujuan menganalisa hasil dari sistem yang telah diimplementasikan menggunakan model pemrograman paralel *shared memory.*

6. Kesimpulan

   Hasil dari pembahasan akan disimpulkan hasilnya.

3. Hasil dan Pembahasan

Penerapan untuk mencari triple pada matrix adjacency tidak berarah dengan menggunakan OpenMP. Memanfaatkan penggunaan `#pragma omp for schedule` untuk mengatur penjadwalan, dilakukan secara statis dengan ukuran *chunk* yang disesuaikan dalam beberapa kali percobaan menghasilkan *task* yang semakin kecil pada *chunk* yang bernilai 100 dan 10 dengan kombinasi sesuai *source code* yang terlihat pada Gambar 4.

```
int triple = 0;
#pragma omp for schedule(static,100)
       for(i=0;i<n-1;i++)
{
#pragma omp for schedule(static,100)
       for(j=i+1;j<n;j++)
{if(matriks[i][j]==1)
{
#pragma omp for schedule(static,10)
       for(k=j+1;k<n;k++)
       {if(matriks[i][k]==1 && matriks[j][k]==1)
       triple++;
}}
}}
```

Gambar 4. Implementasi Pencarian *Triple* Model Pertama

Penerapan untuk mencari *triple* pada *matrix adjacency* tidak berarah dengan menggunakan OpenMP. Memanfaatkan penggunaan #pragma omp parallel untuk mengatur

penjadwalan dan membagi tugas berdasarkan *thread* yang ada untuk menghitung jumlah *triple* yang terbentuk dari verteks yang diinputkan (seperti yang terlihat pada Gambar 5).

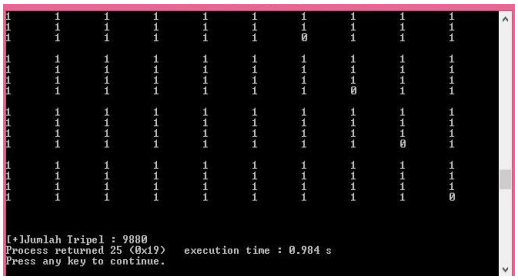
```
#pragma omp parallel
 {     int i,j,k,temp=0,
       me=omp_get_thread_num(),
       nth=omp_get_num_threads();
            for(i=me;i<n;i+=nth)
            {for(j=0;j<n;j++)
              {if(matriks[i][j]==1 && i!=j)
                {
                for(k=j+1;k<n;k++){
                     if(matriks[i][k]==1 && matriks[j][k]==1 && i!=k)
                     temp++;
  }}}}
            #pragma omp critical
            {triple+=temp;
       }}}}
```

Gambar 5. Implementasi Pencarian *Triple* Model Kedua

Kedua model implementasi pencarian tersebut menjelaskan bahwa, untuk mencari *triple* diperlukan tiga buah perulangan yang berfungsi untuk melihat apakah verteks i,j,k memiliki *adjacency* atau tidak. Jika semua verteks tersebut memiliki *adjacency*, maka ketiga verteks tersebut merupakan sebuah *triple*.

Untuk pencarian jumlah *triple* dilakukan secara paralel pada model program pertama, dengan membuat *loop* tiga tingkat dan kemudian memparalelkan dengan cara penjadwalan secara statis dengan menetapkan nilai *chunk* yang telah dilakukan beberapa uji nilai untuk hasil waktu *task* yang semakin kecil didapatkan diproses perulangan kesatu dan kedua menggunkan ukuran *chunk* 100 dan di proses perulangan yang ketiga menggunkan *chunk* 10. Sedangkan untuk pencarian jumlah *triple* dilakukan secara paralel pada model program kedua, yang pertama adalah mem-*broadcast* kepada semua node termasuk dirinya, kemudian masing-masing *node* mengerjakan proses yang sama namun dengan nilai berbeda sesuai dengan penomoran ID *node* yang dimiliki. Ini bertujuan untuk menghindari kesamaan nilai yang dikerjakan masing-masing *node* dan kemudian untuk mendapatkan total jumlah *triple* dari masing-masing *node* digunakan `#pragma omp critical{ triple+=temp;}`

Beberapa percobaan dilakukan dengan menginputkan banyaknya verteks. Pada model kedua program dilakukan pengujian jumlah verteks yang sama untuk membandingkan waktu pengerjaannya. Hanya saja penghitungan waktu pengerjaan pada *count triple* di model program pertama tidak tampil, hanya di model program kedua yang berhasil tampil *time* dari *count triple*. Hasil dari beberapa percobaan dapat dilihat pada Gambar 6 dan 7.



Gambar 6. Contoh Uji Coba Program Model Pertama

Gambar 7. Contoh Uji Coba Program Model Kedua

Verteks	X1	Y1	X2	Y2	Z
300	4455100	9.47	568821	8.81	0.03
400	10586800	15.78	1344279	14.85	0.09
500	20708500	23.76	2596702	21.63	0.17
600	35820200	36.02	4499984	30.14	0.31
700	56921900	46.79	7132116	40.47	0.45

Gambar 8. Cuplikan Hasil Percobaan Model Program Pertama dan Kedua

1. Keterangan Hasil Analisa Program Model Pertama

Proses analisa menggunakan SPSS untuk mengetahui sejauh mana hubungan ketergantungan antara variabel yang terlibat (Gambar 9 sampai dengan Gambar 11) yaitu : jumlah verteks, waktu penghitungan *triple*-nya (*time find counttriple*), jumlah *triple*, dan waktu yang dibutuhkan untuk menjalankan program hingga diperolah hasilnya (*execution time*). Fungsi yang dimanfaatkan adalah analisa regresi dan korelasinya. Korelasi utnuk mengetahui tingkat hubungan variabel yang terlibat sedangkan regresi digunakan untuk memprediksi variabel terikat dengan inputan variabel bebas yaitu jumlah verteks (diinputkan pada awal program berjalan).

Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.992[a]	.984	.979	2.19378

a. Predictors: (Constant), Verteks

b. Dependent Variable: TimeCountTriple1

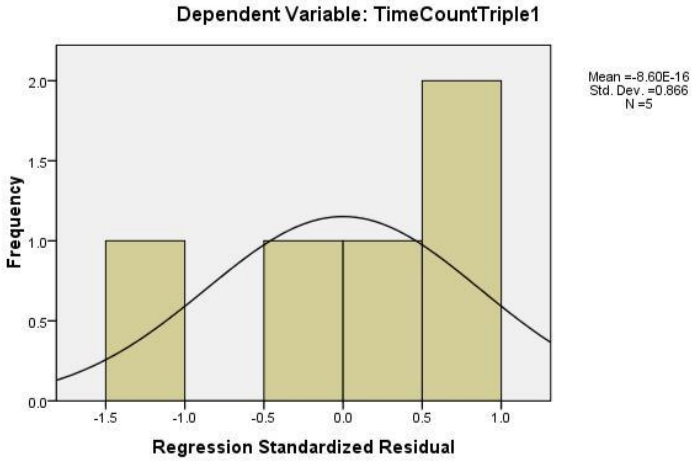
Gambar 9. Hasil Analisis Pertama Model Program Pertama

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-21.071	3.605		-5.845	.010
	Verteks	.095	.007	.992	13.675	.001

a. Dependent Variable: TimeCountTriple1

Gambar 10. Hasil Analisis Kedua Model Program Pertama



Gambar 11. Hasil Analisis Ketiga Model Program Pertama

Pada analisa pertama dilakukan pada hasil percobaan model program 1 dimana jumlah verteks sebagai variabel bebas (independent) dan waktu yang dibutuhkan untuk melakukan penghitungan jumlah triple-nya sebagai variabel yang dipengaruhi/terikat (dependent). Pada Gambar 9 terlihat nilai koefisien korelasinya sebesar 0,992 mendekati 1 itu bearti hubungan antara kedua variabel tersebut kuat dan nilai R Squared-nya sebesar 98.4% bahwa jumlah verteks memiliki kontribusi mempengaruhi jumalh waktu eksekusi, sedangkan variabel lainnya hanya sebesar 1,6%. Standard Error of The Estimate = 2,19 yang menunjukkan ukuran tingkat kesalahan dalam melakukan prediksi terhadap variabel terikat. Pada Gambar 10 digunakan untuk membentuk taksiran pada garis regresinya, didapatkan hasil sebagai berikut :

$Y_1 = a + bX_1$

Dimana :

Y_1 = TimeCountTriple

a = intercept

b = slope

X_1 = Jumlah verteks yang diinputkan

Hasilnya taksiran garis regresinya adalah $Y_1 = -21{,}071 + 0{,}095\ X_1$

Dari hasil taksiran garis regresi tersebut bisa digunakan untuk memprediksi waktu yang dibutuhkan untuk menghitung jumlah *triple* apabila diperkirakan jumlah verteksnya dengan nilai tertentu. Misal dilakukan percobaan jumlah inputan verteksnya 2000 maka waktu penghitungan jumlah *triple*-nya adalah $Y_1 = -21{,}071 + 0{,}095 \times 2000 = -21{,}071 + 180 = 158{,}92$ *second*.

Untuk menguji koefisen garisnya dapat dilihat pada kolom t dan sig. Hasil pengujian ditemukan nilai t hitung sebesar -5,845 dengan sig. = 0,010 (bandingkan dengan nilai sig. F). Oleh karena nilai sig. < 0,05 maka Ho ($\beta = 0$) ditolak yang artinya jumlah verteks yang diinputkan berpengaruh positif terhadap waktu yang dibutuhkan untuk menghitung jumlah *triple*. Positif disitu bearti semakin banyak jumlah verteks yang diinputkan maka semakin lama waktu yang dibutuhkan untuk menghitung jumlah *triple*-nya. Pada Gambar 11 untuk memperlihatkan bagaimana pendistribusian data yang diperoleh dari hasil percobaan pada program pertama.

2. Keterangan Hasil Analisa Program Model Kedua

Pada analisa kedua dilakukan pada hasil percobaan model program kedua dimana jumlah verteks sebagai variabel bebas (*independent*) dan waktu yang dibutuhkan untuk melakukan penghitungan jumlah *triple*-nya sebagai variabel yang dipengaruhi atau terikat (*dependent*). Pada Gambar 12 terlihat nilai koefisien korelasinya sebesar 0,994 mendekati 1 itu bearti hubungan antara kedua variabel tersebut kuat dan nilai R Squared-nya sebesar 98.8% bahwa jumlah verteks memiliki kontribusi mempengaruhi jumalh waktu eksekusi, sedangkan variabel lainnya hanya sebesar 1,2%. *Standard Error of The Estimate* = 1,61 yang menunjukkan ukuran tingkat kesalahan dalam melakukan prediksi terhadap variabel terikat. Pada Gambar 13 digunakan untuk membentuk taksiran pada garis regresinya.

$Y_2 = a + bX_2$

Dimana :

Y_2 = TimeCountTriple

a = intercept

b = slope

X_2 = Jumlah verteks yang diinputkan

Hasilnya taksiran garis regresinya adalah $Y_2 = -16{,}118 + 0{,}079\ X_2$

Dari hasil taksiran garis regresi tersebut bisa digunakan untuk memprediksi waktu yang dibutuhkan untuk menghitung jumlah triple apabila diperkirakan jumlah verteksnya dengan nilai tertentu. Misal dilakukan percobaan jumlah inputan verteksnya 2000 maka waktu penghitungan jumlag triplenya adalah Y_1 = -16,118 + 0,079 x 2000= -16,118 + 158 = 141,882 *second.*

Untuk menguji koefisen garisnya dapat dilihat pada kolom t dan sig. Hasil pengujian ditemukan nilai t hitung sebesar -6,108 dengan sig. = 0,009 (bandingkan dengan nilai sig. F). Oleh karena nilai sig. < 0,05 maka Ho (β = 0) ditolak yang artinya jumlah verteks yang diinputkan berpengaruh positif terhadap waktu yang dibutuhkan untuk menghitung jumlah *triple*. Positif disitu bearti semakin banyak jumlah verteks yang diinputkan maka semakin lama waktu yang dibutuhkan untuk menghitung jumlah *triple*-nya.

Model Summary[b]

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.999[a]	.998	.996	.01062

a. Predictors: (Constant), CountTriple2, Verteks

b. Dependent Variable: ExecutionTime

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate
1	.999[a]	.997	.994	1.12321

a. Predictors: (Constant), Verteks, CountTriple1

Gambar 12. Hasil Analisis Pertama Model Program Kedua

Coefficients[a]

Model		Unstandardized Coefficients		Standardized Coefficients		
		B	Std. Error	Beta	t	Sig.
1	(Constant)	-.072	.048		-1.497	.273
	Verteks	.000	.000	.228	1.621	.246
	CountTriple2	5.000E-8	.000	.775	5.504	.031

a. Dependent Variable: ExecutionTime

Coefficients[a]

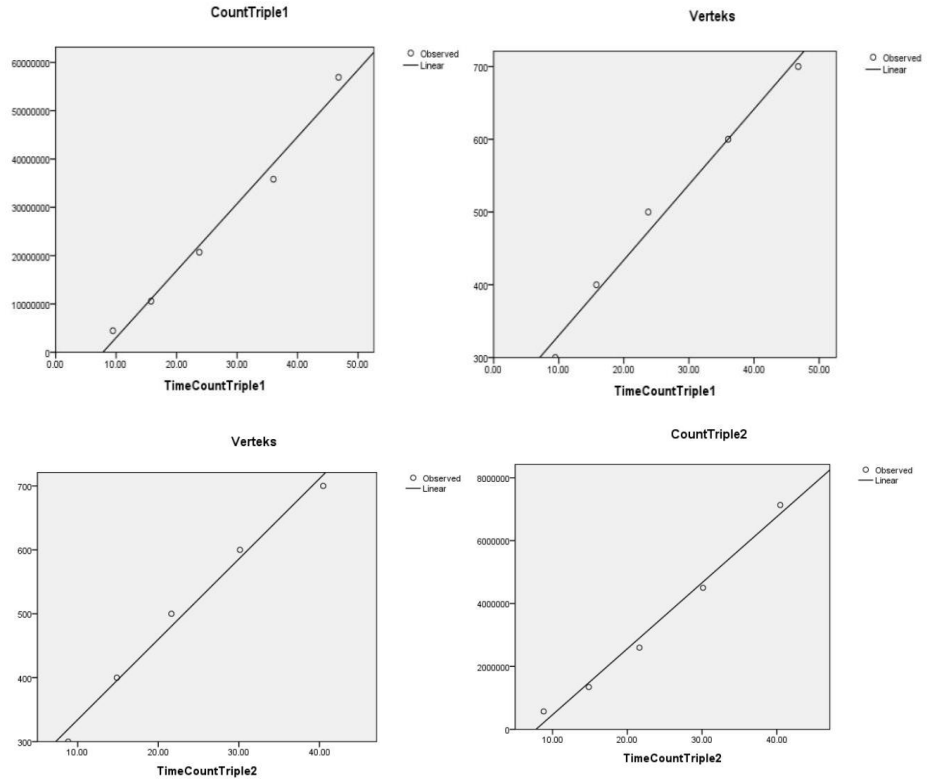
Model		Unstandardized Coefficients		Standardized Coefficients		
		B	Std. Error	Beta	t	Sig.
1	(Constant)	-6.541	5.075		-1.289	.326
	CountTriple1	3.689E-7	.000	.515	3.073	.092
	Verteks	.047	.016	.490	2.924	.100

a. Dependent Variable: TimeCountTriple1

Gambar 13. Hasil Analisis Kedua Model Program Kedua

3. Keterangan Hasil Analisa Program Secara Keseluruhan

Pada analisa keseluruhan, variabel bebas (*independent*) adalah *CountTriple1* dan *CountTriple2* jumlah verteksnya dan waktu yang dibutuhkan untuk melakukan pengerjaan instruksi pada program sebagai variabel yang dipengaruhi atau terikat (*dependent*) (Gambar 14). Didapatkan hasil bahwa hubungan kedua variabel terikat dan bebasnya adalah positif, artinya waktu eksekusi sangat dipengaruhi oleh jumlah *triple* yang diproses dan jumlah verteks yang diinputkan. Jadi bisa dikatakan positif atau hubungan searah di sini adalah apabila nilai variabel bebasnya bertambah maka variabel terikatnya ikut bertambah.



Gambar 14. Hasil Analisis Program Keseluruhan

4. Kesimpulan

Berdasarkan uji coba yang dilakukan dapat disimpulkan bahwa kecepatan pemrosesan suatu sistem paralel dipengaruhi oleh jumlah proses dan jumlah verteks. Pada komputer tunggal semakin besar proses maka akan menambahkan sedikit waktu pada pengerjaannya. Begitu pula dengan bertambahnya jumlah verteks mengakibatkan waktu yang diperlukan semakin besar.

Ucapan Terimakasih

Penulis mengucapkan terima kasih kepada Sekolah Tinggi Teknologi Adisutjipto Yogyakarta yang telah memberi dukungan finansial terhadap penelitian ini. Juga kepada bagian P3M STTA yang telah membantu tugas dosen untuk melaksanakan salah satu Tridharma perguruan tinggi.

Daftar Pustaka

[1] Ayuningtyas, A. (2016, November). Pemrosesan Paralel pada *Low Pass Filtering* Menggunakan *Transform Cosinus* di MPI (*Message Passing Interface*). In *Conference SENATIK STT Adisutjipto Yogyakarta* (Vol. 2, pp. 115-120).

[2] Gebali, F. (2011). Algorithms and Parallel Computing (Vol. 84). John Wiley & Sons.

[3] Matloff, N. (2011). Programming on Parallel Machines. Retrieved August 23, 2016, from http://heather.cs.ucdavis.edu/~matloff/ParProcBook.pdf.

[4] Quinn, M. J. (2003). Parallel Programming in C with MPI and OpenMP. McGraw-Hill.

[5] Wilkinson, B., & Allen, M. (2005). Parallel Programming (2nd ed.). Pearson Educational International.

[6] Adji, T. B., & Nugroho, H. A. (2016, November). Pengaruh Load Balancing Pada Pemrosesan Paralel untuk Kompresi Video. In Conference SENATIK STT Adisutjipto Yogyakarta (Vol. 2, pp. 121-128).

[7] Indrianingsih, Y., Wintolo, H., & Sari, I. K. (2013, December). Penerapan Grid Computing Untuk Mengkompilasi Program Berbahasa C/C++. In Conference SENATIK STT Adisutjipto Yogyakarta (Vol. 1, p. 67).

[8] Kelen, W. W., & Nugraheny, D. (2015). Analisa Pemrosesan Paralel untuk Kompresi dan Dekompresi Data. Compiler, 4(1)

[9] Ngadiyono, N., & Wintolo, H. (2015). Penerapan Pemrosesan Paralel untuk Menguji Waktu Rendering Design Web dengan Framework terhadap Processor melalui Lan, Router dan Ekstranet. Compiler, 4(1).