BAB IV
PENGUJIAN DAN ANALISIS

4.1 Skenario Pengujian

Program aplikasi diimplementasikan pada sebuah PC *quadcore* dan *cluster* 4 PC *quadcore* untuk mendapatkan perbandingan kinerja antara algoritma paralel dengan MPICH2 dan Cilk++. Tujuan dari pengujian ini adalah mendapatkan waktu proses dari masing-masing algoritma paralel untuk aplikasi uji perkalian matriks dan *sorting*. Variabel analisis yang digunakan terhadap hasil pengujian adalah percepatan (*speed-up*) dan efisiensi.

4.1.1. Percepatan (*speed-up*)

Percepatan dalam pengujian komputasi paralel adalah nilai yang diperoleh dari perbandingan antara waktu proses komputasi serial dengan waktu proses komputasi paralel. Nilai ini diperoleh dari persamaan 4.1 [7]:

$$S_{(n)} = \frac{t_s}{t_{p(n)}} \tag{4.1}$$

Dengan :

S = percepatan

t_s = waktu proses komputasi serial

t_p = waktu proses komputasi paralel

n = nilai paralelisme

Nilai percepatan di atas 1 berarti ada peningkatan kecepatan proses sedangkan nilai di antara 0 dan 1 berarti ada penurunan kecepatan proses. Percepatan adalah ukuran paling umum yang digunakan untuk mengukur kinerja komputasi paralel.

4.1.2. Efisiensi

Efisiensi pada pengujian ini adalah nilai yang diperoleh dari perbandingan antara waktu proses komputasi serial dengan waktu proses komputasi paralel yang dikalikan besar paralelismenya, atau perbandingan antara percepatan dengan besar paralelismenya. Nilai ini diperoleh dari persamaan 4.2 [7]:

$$E_{(n)} = \frac{t_s}{t_{p(n)} \times n} = \frac{S_{(n)}}{n} \tag{4.2}$$

Dengan :

E = efisiensi

Nilai efisiensi di sini berarti besarnya peningkatan kecepatan setiap penambahan besaran paralelisme atau dengan kata lain porsi percepatan yang didapat ketika paralelisme ditingkatkan. Efisiensi merupakan ukuran umum untuk menilai harga (*cost*) dari penggunaan sumber daya komputasi.

4.1.3. Sampel dan Variabel Pengujian

Pada pengujian ini, penulis menggunakan sampel uji perkalian matriks dengan ukuran matriks 256 x 256, 512 x 512, dan 1024 x 1024. Sedangkan untuk sorting sampel ujinya adalah jumlah data 25.000.000, 50.000.000, 100.000.000, dan 200.000.000 data *integer*. Pengujian dilakukan pada dua arsitektur komputer paralel yang berbeda yaitu *cluster* PC *quadcore* dan PC *quadcore*. Adapun variabel-variabel pada algoritma paralel yang digunakan adalah sebagai berikut :

1) Jumlah proses untuk algoritma paralel dengan MPICH2

Jumlah proses adalah variabel yang digunakan dalam program paralel seperti pada MPICH2, karena jumlah proses menunjukkan kemampuan algoritma paralel melakukan *multiprocessing* dengan prosesor yang ada. Jumlah proses yang kurang dari jumlah prosesor bisa bermakna inefisiensi, namun jumlah proses yang *overhead* bisa tidak berpengaruh pada percepatan yang diharapkan. Pada pengujian ini jumlah proses yang digunakan adalah jumlah proses 4, 8, 12, dan 16 yang diimplementasikan pada *cluster* PC *quadcore* serta jumlah proses 1, 2, 3, dan 4 yang diimplementasikan pada PC *quadcore*.

2) Jumlah *thread* untuk algoritma paralel dengan Cilk++

Jumlah *thread* adalah variabel yang difasilitasi oleh Cilk++ sehingga pengguna bisa memanfaatkan sebagian atau seluruhnya sebagai *default*. Jumlah *thread* yang maksimal mungkin bermakna efisiensi, namun *overhead* juga bisa berpengaruh pada percepatan yang diharapkan. Adapun untuk pengujian ini jumlah *thread* yang digunakan adalah 1, 2, 3, dan 4 yang diimplementasikan pada PC *quadcore*.

Pada pengujian diambil data hasil implementasi masing-masing aplikasi dengan variabel dari setiap algoritma paralel sebanyak 10 kali untuk setiap operasinya untuk menghindari penyimpangan ekstrim dari satu pengujian.

4.2 Hasil yang Diperoleh

Setelah melakukan pengujian dengan skenario tersebut di atas, berikut ini adalah hasil-hasil yang diperoleh dari implementasi algoritma paralel dengan MPICH2 dan Cilk++ untuk aplikasi perkalian matriks dan *sorting*.

4.2.1 Hasil Pengujian Algoritma Paralel untuk Aplikasi Perkalian Matriks

Pengujian aplikasi perkalian matriks yang diimplementasikan mendapatkan hasil berupa waktu proses (dalam detik/*seconds*) sebagai berikut :

1) Algoritma paralel MPICH2 dengan jumlah proses 4, 8, 12, dan 16 pada *cluster* PC *quadcore* membutuhkan waktu proses seperti terlihat pada Tabel 4.1.

Tabel 4.1. Waktu Proses Rata-rata Perkalian Matriks Algoritma Paralel MPICH2 dengan Jumlah Proses 4, 8, 12, dan 16

Ukuran Matriks	Proses = 4	Proses = 8	Proses = 12	Proses = 16
256*256	0,111 s	0,054 s	0,060 s	0,068 s
512*512	0,621 s	0,421 s	0,422 s	0,381 s
1024*1024	10,581 s	9,768 s	7,224 s	6,795 s

2) Algoritma paralel MPICH2 dengan jumlah proses 1, 2, 3, dan 4 pada PC *quadcore* membutuhkan waktu proses seperti terlihat pada Tabel 4.2.

Tabel 4.2. Waktu Proses Rata-rata Perkalian Matriks Algoritma Paralel MPICH2 dengan Jumlah Proses 1, 2, 3, dan 4

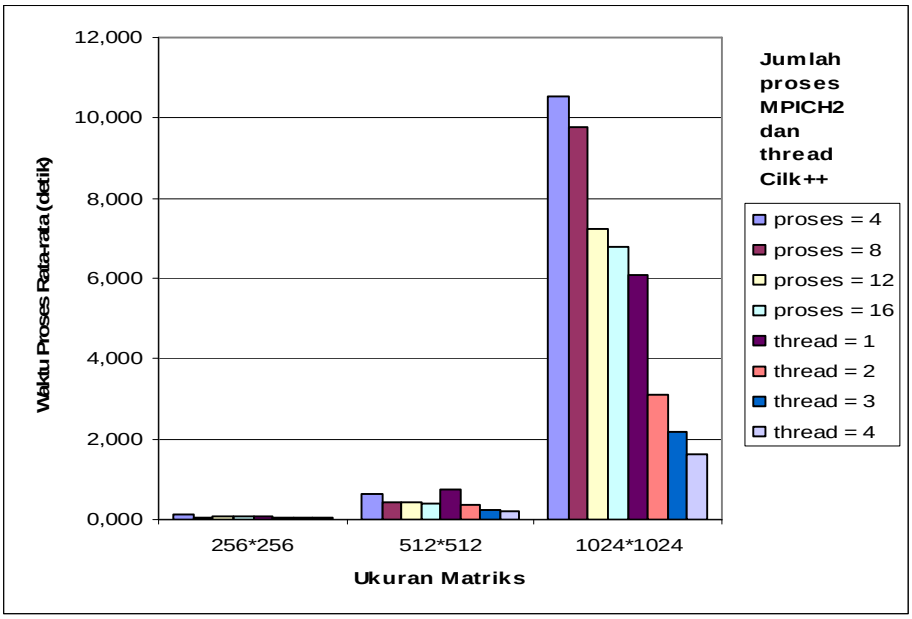
Ukuran Matriks	Proses = 1	Proses = 2	Proses = 3	Proses = 4
256*256	0,131 s	0,070 s	0,054 s	0,048 s
512*512	2,215 s	1,131 s	0,796 s	0,797 s
1024*1024	40,679 s	38,517 s	37,701 s	26,769 s

3) Algoritma paralel Cilk ++ dengan jumlah *thread* 1, 2, 3, dan 4 pada PC *quadcore* membutuhkan waktu proses seperti terlihat pada Tabel 4.3.

Tabel 4.3. Waktu Proses Rata-rata Perkalian Matriks Algoritma Paralel Cilk++ dengan Jumlah *Thread* 1, 2, 3, dan 4

Ukuran Matriks	*Thread* = 1	*Thread* = 2	*Thread* = 3	*Thread* = 4
256*256	0,094 s	0,047 s	0,032 s	0,028 s
512*512	0,743 s	0,373 s	0,255 s	0,202 s
1024*1024	6,097 s	3,107 s	2,174 s	1,619 s

4) Perbandingan waktu proses kedua algoritma paralel untuk aplikasi perkalian matriks terlihat pada Gambar 4.1.



Gambar 4.1. Grafik Perbandingan Waktu Proses Algoritma Paralel MPICH2 dan Cilk++ untuk Aplikasi Perkalian Matriks

4.2.2 Hasil Pengujian Algoritma Paralel untuk Aplikasi *Sorting*

Pengujian aplikasi *sorting* yang diimplementasikan mendapatkan hasil berupa waktu proses (dalam detik/*seconds*) sebagai berikut :

1) Algoritma paralel MPICH2 dengan jumlah proses 4, 8, 12, dan 16 pada *cluster* PC *quadcore* membutuhkan waktu proses seperti tampak pada Tabel 4.4.

Tabel 4.4. Waktu Proses Rata-rata *Sorting* Algoritma Paralel MPICH2 dengan Jumlah Proses 4, 8, 12, dan 16

Jumlah Data	Proses = 4	Proses = 8	Proses = 12	Proses = 16
25.000.000	4,621 s	3,797 s	3,878 s	3,250 s
50.000.000	10,469 s	7,845 s	7,800 s	6,562 s
100.000.000	21,325 s	15,953 s	15,640 s	13,528 s
200.000.000	43,316 s	36,675 s	31,463 s	35,714 s

2) Algoritma paralel MPICH2 dengan jumlah proses 1, 2, 3, dan 4 pada PC *quadcore* membutuhkan waktu proses seperti terlihat pada Tabel 4.5.

Tabel 4.5. Waktu Proses Rata-rata *Sorting* Algoritma Paralel MPICH2 dengan Jumlah Proses 1, 2, 3, dan 4

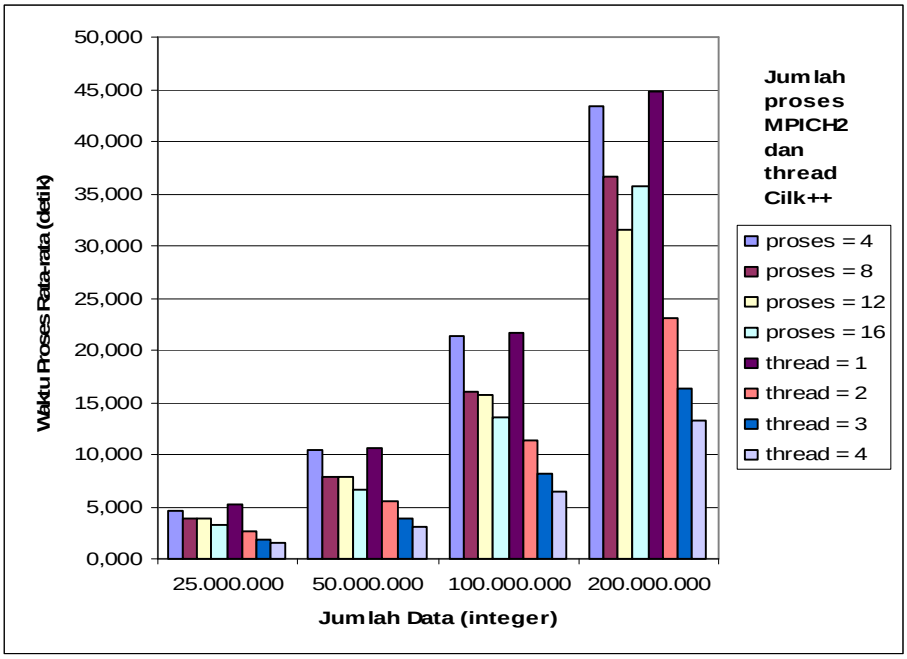
Jumlah Data	Proses = 1	Proses = 2	Proses = 3	Proses = 4
25.000.000	13,330 s	7,136 s	5,306 s	4,575 s
50.000.000	27,399 s	14,726 s	10,993 s	9,421 s
100.000.000	56,365 s	30,132 s	22,261 s	19,416 s
200.000.000	115,582 s	61,951 s	58,997 s	47,065 s

3) Algoritma paralel Cilk ++ dengan jumlah *core* atau *thread* 1, 2, 3, dan 4 membutuhkan waktu proses sebagaimana terlihat pada Tabel 4.6.

Tabel 4.6. Waktu Proses Rata-rata *Sorting* Algoritma Paralel Cilk++ dengan Jumlah *Thread* 1, 2, 3, dan 4

Jumlah Data	*Thread* = 1	*Thread* = 2	*Thread* = 3	*Thread* = 4
25.000.000	5,189 s	2,660 s	1,826 s	1,464 s
50.000.000	10,584 s	5,471 s	3,895 s	3,075 s
100.000.000	21,740 s	11,334 s	8,087 s	6,527 s
200.000.000	44,729 s	23,150 s	16,243 s	13,369 s

4) Perbandingan waktu proses kedua algoritma paralel untuk aplikasi *sorting* terlihat pada Gambar 4.2.



Gambar 4.2. Grafik Perbandingan Waktu Proses Algoritma Paralel MPICH2 dan Cilk++ untuk Aplikasi *Sorting*

4.3 Analisis Hasil Pengujian

Setelah mendapatkan hasil pengujian algoritma paralel untuk aplikasi perkalian dan *sorting* maka dapat dilakukan analisis terhadapnya. Adapun variabel yang digunakan dalam melakukan analisis sebagaimana tersebut pada Subbab 4.1 adalah percepatan (*speed-up*) dan efisiensi. Tujuannya adalah untuk melihat perbandingan performansi setiap algoritma paralel pada aplikasi uji dan lingkungan paralel yang berbeda.

Pengujian algoritma paralel MPICH2 pada *cluster* PC *quadcore* menggunakan variabel uji jumlah proses 4, 8, 12, dan 16. Variabel ini digunakan untuk mengetahui sejauh mana pengaruhnya terhadap penggunaan *core processor* pada setiap PC. Asumsinya, penambahan jumlah proses menurut kelipatan jumlah node akan menghasilkan percepatan mendekati jumlah node pada setiap variabel. Misal untuk jumlah proses 4, maka hasil yang diharapkan adalah percepatan mendekati 4 kali bila dibandingkan dengan operasi yang sama untuk variabel jumlah proses 1 pada PC *quadcore*.

Pengujian algoritma paralel MPICH2 pada PC *quadcore* menggunakan variabel uji jumlah proses 1, 2, 3, dan 4 sebagai parameter identitas baik untuk algoritma paralel MPICH2 yang diimplementasikan pada *cluster* PC *quadcore* maupun untuk algoritma paralel Cilk++ yang diimplementasikan pada PC *quadcore* yang menggunakan variabel uji jumlah *thread* 1, 2, 3, dan 4. Khusus yang terakhir, penambahan jumlah *thread* diharapkan akan meningkatkan percepatan sehingga mendekati jumlah *core*-nya.

4.3.1. Analisis Percepatan dan Efisiensi dari Algoritma Paralel untuk Aplikasi Perkalian Matriks

Implementasi algoritma paralel dengan MPICH2 pada *cluster* PC *quadcore* menghasilkan percepatan yang signifikan bila dibandingkan dengan algoritma serialnya (asumsi, sama dengan jumlah proses 1). Nilai terendah percepatannya 1,180 (untuk jumlah proses 4 dan ukuran matriks 256 x 256) sedangkan nilai tertingginya 5,987 (untuk jumlah proses 16 dan ukuran matriks 1024 x 1024). Kondisi ini tampak pada Tabel 4.7.

Tabel 4.7. Percepatan Perkalian Matriks Algoritma Paralel MPICH2 pada *cluster* PC *Quadcore*

Ukuran Matriks	Proses = 4	Proses = 8	Proses = 12	Proses = 16
256*256	1,180	2,426	2,183	1,926
512*512	3,567	5,261	5,249	5,814
1024*1024	3,868	4,165	5,631	5,987

Implementasi algoritma paralel pada PC *quadcore* baik dengan MPICH2 maupun Cilk++ masing-masing menghasilkan percepatan yang jauh berbeda. Bila nilai rata-rata percepatan pada MPICH2 adalah 1,767 maka nilai rata-rata

percepatan Cilk++ mencapai 8,761 saat dibandingkan dengan algoritma serialnya (asumsi, sama dengan jumlah proses 1). Kondisi ini diperoleh dari nilai percepatan yang tampak pada Tabel 4.8.

Tabel 4.8. Percepatan Perkalian Matriks Algoritma Paralel MPICH2 dan Cilk++ pada PC *Quadcore*

Ukuran Matriks	Jumlah Proses pada MPICH2				Jumlah *Thread* pada Cilk++			
	1	2	3	4	1	2	3	4
256*256	1,000	1,871	2,426	2,729	1,394	2,787	4,094	4,679
512*512	1,000	1,958	2,783	2,779	2,981	5,938	8,686	10,965
1024*1024	1,000	1,056	1,079	1,520	6,672	13,093	18,712	25,126

Sebagai pembanding obyektif, percepatan yang dihasilkan algoritma paralel Cilk++ dihitung dengan asumsi jumlah *thread* 1 sebagai parameter identitasnya. Maka akan diperoleh nilai percepatan sebagaimana tampak pada Tabel 4.9.

Tabel 4.9. Percepatan Perkalian Matriks Algoritma Paralel Cilk++ pada PC *Quadcore*

Ukuran Matriks	*Thread* = 1	*Thread* = 2	*Thread* = 3	*Thread* = 4
256*256	1,000	2,000	2,938	3,357
512*512	1,000	1,992	2,914	3,678
1024*1024	1,000	1,962	2,805	3,766

Dari analisis kecepatan di atas didapati bahwa implementasi algoritma paralel MPICH2 untuk aplikasi perkalian matriks pada *cluster* PC *quadcore* mampu meningkatkan performansi komputasi dengan memanfaatkan *core processor* yang dimiliki meskipun belum maksimal. Walaupun algoritma ini tidak secara eksplisit membagi pekerjaan pada *core* tertentu tetapi jumlah proses yang banyak menyebabkan sistem operasi memaksimalkan penggunaan *core* yang dimiliki. Hal ini dibuktikan dengan dicapainya percepatan melebihi jumlah *node* yang digunakan (4 PC dalam *cluster*).

Sedangkan implementasi algoritma paralel Cilk++ untuk aplikasi yang sama pada PC *quadcore* mampu menghasilkan percepatan maksimal 3,766, hampir mencapai 4 atau jumlah yang sama dengan *core processor* yang dimiliki PC. Secara umum peningkatan percepatannya berbanding lurus dengan penambahan jumlah *thread* yang digunakan. Artinya teknik *multithreading* dari Cilk++ mampu memaksimalkan penggunaan core yang dimiliki.

Analisis lebih lanjut terhadap percepatan masing-masing algoritma paralel menghasilkan nilai efisiensi sebagaimana tampak pada Tabel 4.10 dan Tabel 4.11, di mana bila dibandingkan dengan percepatan algoritma paralel MPICH2 dengan jumlah proses 1 maka nilai terendahnya adalah 0,120 (untuk algoritma paralel MPICH2 dengan jumlah proses 16 pada ukuran matriks 256 x 256) dan nilai tertingginya adalah 6,672 (untuk algoritma paralel Cilk++ dengan jumlah *thread* 1 pada ukuran matriks 1042 x 1024).

Tabel 4.10. Efisiensi Perkalian Matriks Algoritma Paralel MPICH2 pada *Cluster* PC *Quadcore*

Ukuran Matriks	Proses = 4	Proses = 8	Proses = 12	Proses = 16
256*256	0,295	0,303	0,182	0,120
512*512	0,892	0,658	0,437	0,363
1024*1024	0,967	0,521	0,469	0,374

Tabel 4.11. Efisiensi Perkalian Matriks Algoritma Paralel pada PC *Quadcore*

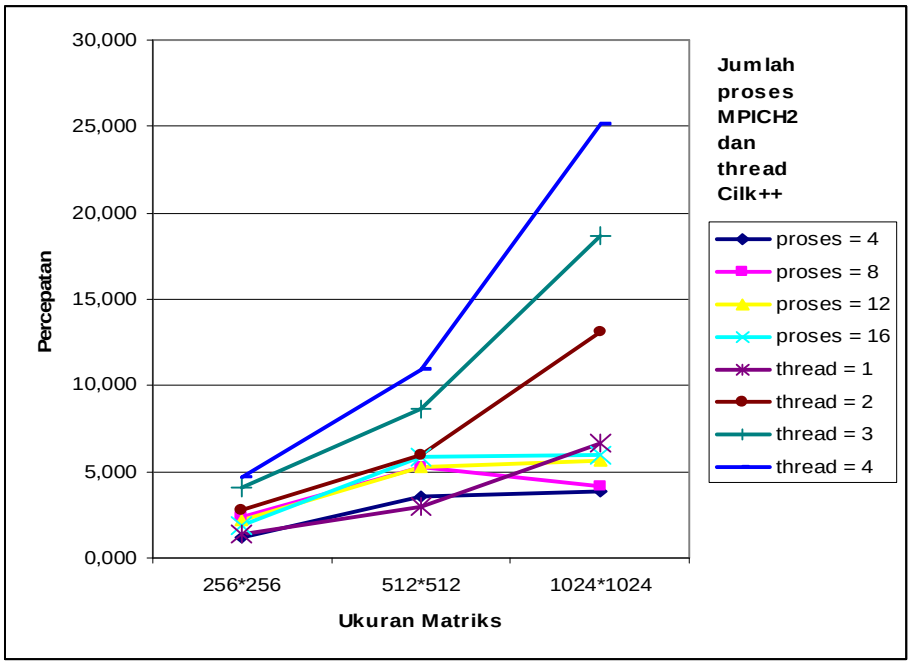
Ukuran Matriks	Jumlah Proses pada MPICH2				Jumlah *Thread* pada Cilk++			
	1	2	3	4	1	2	3	4
256*256	1,000	0,936	0,809	0,682	1,394	1,394	1,365	1,170
512*512	1,000	0,979	0,928	0,695	2,981	2,969	2,895	2,741
1024*1024	1,000	0,528	0,360	0,380	6,672	6,546	6,237	6,282

Efisiensi obyektif untuk algoritma paralel Cilk++ ditunjukkan pada Tabel 4.12, di mana nilai tertingginya adalah 0,996 (pada jumlah *thread* = 2 dan ukuran matriks 512 x 512) dan terendahnya adalah 0,839 (pada jumlah *thread* = 4 dan ukuran matriks 256 x 256).

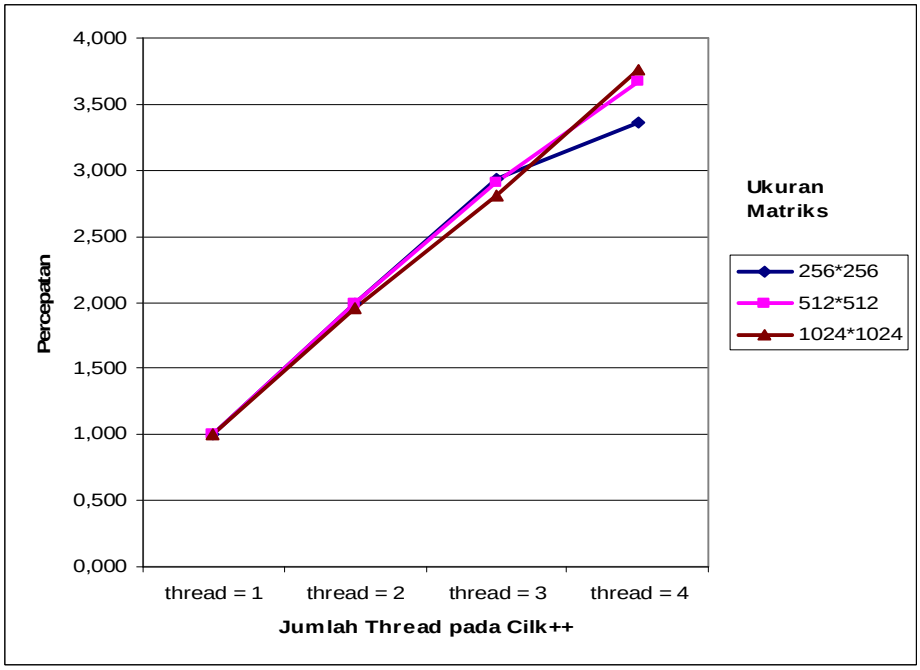
Tabel 4.12. Efisiensi Perkalian Matriks Algoritma Paralel Cilk++ pada PC *Quadcore*

Ukuran Matriks	*Thread* = 1	*Thread* = 2	*Thread* = 3	*Thread* = 4
256*256	1,000	1,000	0,979	0,839
512*512	1,000	0,996	0,971	0,920
1024*1024	1,000	0,981	0,935	0,941

Perbandingan percepatan algoritma paralel dengan MPICH2 dan Cilk++ baik yang diimplementasikan pada *cluster* PC *quadcore* maupun PC *quadcore* secara grafis ditunjukkan oleh Gambar 4.3.



Gambar 4.3. Grafik Percepatan Algoritma Paralel MPICH2 dan Cilk++ untuk Aplikasi Perkalian Matriks

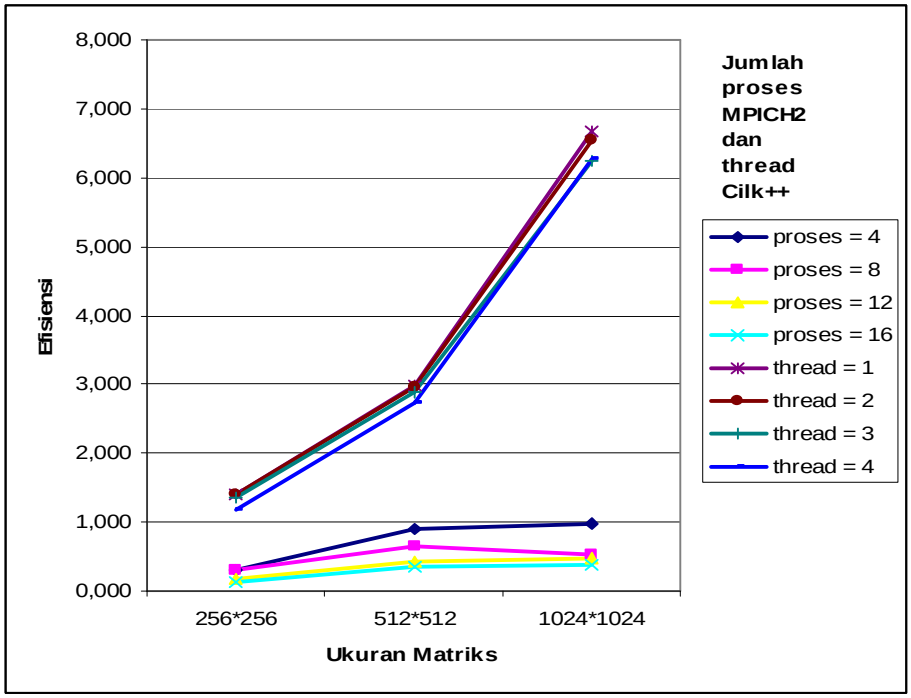


Gambar 4.4. Grafik Percepatan Algoritma Paralel Cilk++ untuk Aplikasi Perkalian Matriks

Dari grafik tersebut diperoleh kecenderungan percepatan dari implementasi algoritma paralel dengan MPICH2 mengalami perubahan yang hampir mendatar dengan pertambahan ukuran matriks, berbeda dengan implementasi algoritma

Cilk++ yang memiliki kecenderungan peningkatan percepatan sebanding dengan penambahan jumlah *thread* walaupun ukuran matriks bertambah. Kecenderungan percepatan Cilk++ mendekati jumlah *core* ditunjukkan secara grafis pada Gambar 4.4.

Perbandingan efisiensi dari implementasi algoritma paralel pada MPICH2 dan Cilk++ baik yang diimplementasikan pada *cluster* PC *quadcore* maupun PC *quadcore* secara grafis ditunjukkan Gambar 4.5.



Gambar 4.5. Grafik Efisiensi Algoritma Paralel MPICH2 dan Cilk++ untuk Aplikasi Perkalian Matriks

Dari grafik di atas terlihat bahwa nilai efisiensi yang dihasilkan oleh algoritma paralel dengan Cilk++ pada perkalian matriks jauh lebih besar bila dibandingkan dengan nilai efisiensi dari algoritma paralel dengan MPICH. Hal ini tentu menjadi catatan bahwa implementasi algoritma paralel dengan MPICH pada *cluster* PC *quadcore* belum bisa memaksimalkan paralelisme *core* yang dimilikinya, sedangkan algoritma paralel dengan Cilk++ mampu memaksimalkan paralelisme yang dimiliki PC *quadcore*.

4.3.2. Analisis Percepatan dan Efisiensi dari Algoritma Paralel untuk Aplikasi *Sorting*

Implementasi algoritma paralel dengan MPICH2 pada *cluster* PC *quadcore* menghasilkan percepatan yang signifikan bila dibandingkan dengan algoritma serialnya (asumsi, sama dengan jumlah proses 1). Nilai terendahnya 2,617 (untuk jumlah proses 4 dan jumlah data 50.000.000) sedangkan nilai tertingginya 4,175 (untuk jumlah proses 16 dan jumlah data 50.000.000). Kondisi ini tampak pada Tabel 4.13.

Tabel 4.13. Percepatan *Sorting* Algoritma Paralel MPICH2 pada *Cluster* PC *Quadcore*

Jumlah Data	Proses = 4	Proses = 8	Proses = 12	Proses = 16
25.000.000	2,885	3,511	3,437	4,102
50.000.000	2,617	3,493	3,513	4,175
100.000.000	2,643	3,533	3,604	4,167
200.000.000	2,668	3,152	3,674	3,236

Seperti halnya pada perkalian matriks, implementasi algoritma paralel untuk *sorting* pada PC *quadcore* baik dengan MPICH2 maupun Cilk++ masing-masing menghasilkan percepatan yang jauh berbeda. Bila nilai rata-rata percepatan pada MPICH2 adalah 2,009 maka nilai rata-rata percepatan Cilk++ mencapai 5,881 saat dibandingkan dengan algoritma serialnya (asumsi, sama dengan jumlah proses 1). Kondisi ini diperoleh dari nilai percepatan yang tampak pada Tabel 4.14.

Tabel 4.14. Percepatan *Sorting* Algoritma Paralel MPICH2 dan Cilk++ pada PC *Quadcore*

Jumlah Data	Jumlah Proses pada MPICH2				Jumlah *Thread* pada Cilk++			
	1	2	3	4	1	2	3	4
25.000.000	1,000	1,868	2,512	2,914	2,569	5,011	7,300	9,105
50.000.000	1,000	1,861	2,492	2,908	2,589	5,008	7,034	8,910
100.000.000	1,000	1,871	2,532	2,903	2,593	4,973	6,970	8,636
200.000.000	1,000	1,866	1,959	2,456	2,584	4,993	7,116	8,703

Sebagai pembanding obyektif, percepatan yang dihasilkan algoritma paralel Cilk++ dihitung dengan asumsi jumlah *thread* 1 sebagai parameter identitasnya. Maka akan diperoleh nilai percepatan sebagaimana tampak pada Tabel 4.15.

Tabel 4.15. Percepatan *Sorting* Algoritma Paralel Cilk++ pada PC *Quadcore*

Jumlah Data	*Thread* = 1	*Thread* = 2	*Thread* = 3	*Thread* = 4
25.000.000	1,000	1,925	2,681	3,313
50.000.000	1,000	1,936	2,693	3,354
100.000.000	1,000	1,919	2,632	3,248
200.000.000	1,000	1,952	2,706	3,270

Senada dengan analisis percepatan untuk aplikasi perkalian matriks, implementasi algoritma paralel MPICH2 untuk aplikasi perkalian matriks pada *cluster* PC *quadcore* mampu meningkatkan performansi komputasi dengan memanfaatkan *core processor* yang dimiliki meskipun belum maksimal. Walaupun algoritma ini tidak secara eksplisit membagi pekerjaan pada *core* tertentu tetapi jumlah proses yang banyak menyebabkan sistem operasi memaksimalkan penggunaan *core* yang dimiliki. Hal ini dibuktikan dengan dicapainya percepatan melebihi jumlah *node* yang digunakan (4 PC dalam *cluster*).

Sedangkan implementasi algoritma paralel Cilk++ untuk aplikasi yang sama pada PC *quadcore* mampu menghasilkan percepatan maksimal 3,354 atau hampir mencapai jumlah yang sama dengan *core processor* yang dimiliki PC *quadcore*. Secara umum peningkatan percepatannya berbanding lurus dengan penambahan jumlah *thread* yang digunakan.

Analisis lebih lanjut terhadap percepatan masing-masing algoritma paralel menghasilkan nilai efisiensi sebagaimana tampak pada Tabel 4.16 dan Tabel 4.17, di mana bila dibandingkan dengan percepatan algoritma paralel MPICH2 dengan jumlah proses 1 maka nilai terendahnya adalah 0,202 (untuk algoritma paralel MPICH2 dengan jumlah proses 16 dan jumlah data 200.000.000) dan nilai tertingginya adalah 2,593 (untuk algoritma paralel Cilk++ dengan jumlah *thread* 1 dan jumlah data 100.000.000).

Tabel 4.16. Efisiensi *Sorting* Algoritma Paralel MPICH2 pada PC *Quadcore*

Jumlah Data	Proses = 4	Proses = 8	Proses = 12	Proses = 16
25.000.000	0,721	0,439	0,286	0,256
50.000.000	0,654	0,437	0,293	0,261
100.000.000	0,661	0,442	0,300	0,260
200.000.000	0,667	0,394	0,306	0,202

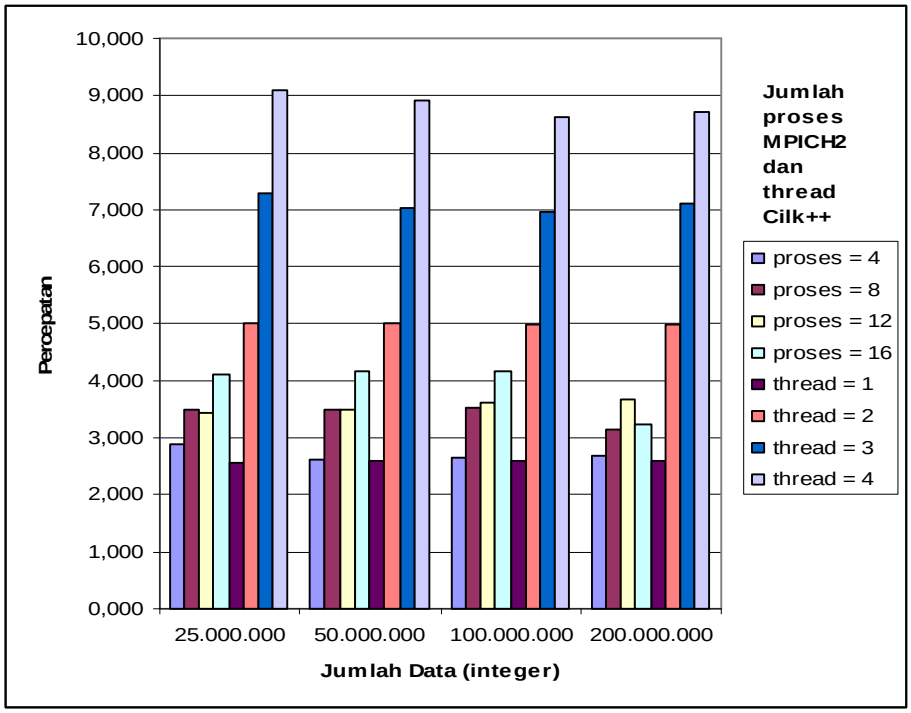
Tabel 4.17. Efisiensi *Sorting* Algoritma Paralel pada PC *Quadcore*

Jumlah Data	Jumlah Proses pada MPICH2				Jumlah *Thread* pada Cilk++			
	1	2	3	4	1	2	3	4
25.000.000	1,000	0,934	0,837	0,728	2,569	2,506	2,433	2,276
50.000.000	1,000	0,930	0,831	0,727	2,589	2,504	2,345	2,228
100.000.000	1,000	0,935	0,844	0,726	2,593	2,487	2,323	2,159
200.000.000	1,000	0,933	0,653	0,614	2,584	2,496	2,372	2,176

Efisiensi obyektif untuk algoritma paralel Cilk++ ditunjukkan pada Tabel 4.18, di mana nilai tertingginya adalah 0,975 dan terendahnya adalah 0,833.

Tabel 4.18. Efisiensi *Sorting* Algoritma Paralel Cilk++ pada PC *Quadcore*

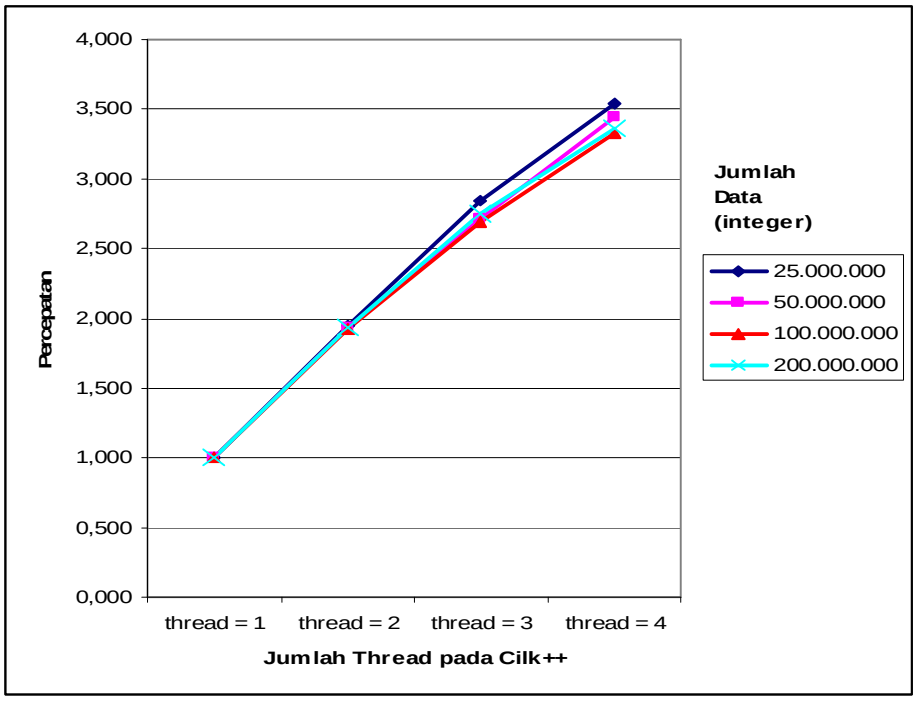
Jumlah Data	*Thread* = 1	*Thread* = 2	*Thread* = 3	*Thread* = 4
25.000.000	1,000	0,975	0,947	0,886
50.000.000	1,000	0,967	0,906	0,860
100.000.000	1,000	0,959	0,896	0,833
200.000.000	1,000	0,966	0,918	0,842



Gambar 4.6. Grafik Percepatan Algoritma Paralel MPICH2 dan Cilk++ untuk Aplikasi *Sorting*

Perbandingan percepatan algoritma paralel dengan MPICH2 dan Cilk++ baik yang diimplementasikan pada *cluster* PC *quadcore* maupun PC *quadcore* secara grafis ditunjukkan oleh Gambar 4.6. Dari grafik tersebut didapati bahwa percepatan kedua algoritma paralel mengalami peningkatan sejalan dengan bertambahnya jumlah proses atau *thread*. Berbeda dengan perkalian matriks, pada *sorting* perbedaan percepatan pada kedua algoritma paralel tidak terlalu jauh. Walaupun demikian bila melihat dari jumlah prosesor yang digunakan untuk implementasi keduanya maka algoritma paralel dengan Cilk++ lebih mendekati harapan karena percepatannya hampir berbanding lurus dengan jumlah *core*-nya.

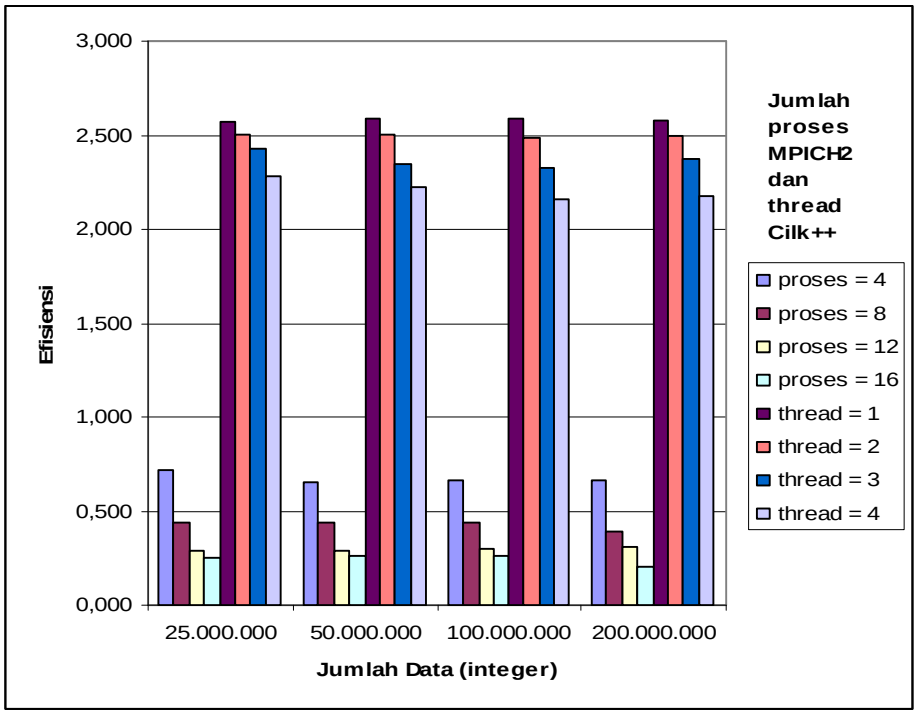
Pada Gambar 4.7. tampak bahwa kecenderungan percepatan algoritma paralel dengan Cilk++ mengikuti pola hampir linier. Hal ini menunjukkan bahwa algoritma paralel menggunakan setiap core processor lebih maksimal bila dibandingkan dengan algoritma paralel dengan MPICH2.



Gambar 4.7. Grafik Percepatan Algoritma Paralel Cilk++ untuk Aplikasi *Sorting*

Perbandingan efisiensi dari kedua algoritma paralel ditunjukkan pada Gambar 4.8. di mana tampak bahwa efisiensi algoritma paralel dengan MPICH tidak mencapai nilai 1 sedangkan efisiensi algoritma paralel dengan Cilk++ mencapai lebih dari nilai 2. Apabila diambil rata-rata efisiensi dari keduanya maka efisiensi algoritma

paralel dengan Cilk++ 5,872 kali lebih baik dibandingkan dengan algoritma paralel dengan MPICH2.



Gambar 4.8. Grafik Efisiensi Algoritma Paralel MPICH2 dan Cilk++ untuk Aplikasi *Sorting*