UNIVERSITAS INDONESIA

PR 1 PEMROGRAMAN MPI

LAPORAN TUGAS PEMROGRAMAN PARALEL

KELOMPOK III
Muhammad Fathurachman 1506706276
Otniel Yosi Viktorisa 1506706295
Yohanes Gultom 1506706345

FAKULTAS ILMU KOMPUTER
PROGRAM STUDI MAGISTER ILMU KOMPUTER
DEPOK
APRIL 2016

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1
LINGKUNGAN PERCOBAAN

1.1 Lingkungan *Cluster*

Dalam eksperimen yang dilakukan, kelompok kami menggunakan empat buah lingkungan yaitu *cluster* Rocks University of California San Diego (UCSD), *cluster* Fasilkom Universitas Indonesia (UI) dan *cluster* Rocks pada *laptop*.

1.1.1 Cluster Rocks University of California San Diego (UCSD)

Cluster Rocks[1] milik University of California San Diego (UCSD) ini dapat diakses pada alamat *nbcr-233.ucsd.edu* menggunakan protokol SSH dari komputer yang telah didaftarkan *public key* nya. Berdasarkan informasi dari aplikasi *monitoring* Ganglia [2], cluster ini terdiri 10 *nodes* dengan total 80 prosesor.

Pada *cluster* ini sudah terpasang pustaka komputasi paralel OpenMPI[3] dan MPICH[4] serta paket dinamika molekular AMBER. Program paralel MPI dan eksperimen AMBER dijalankan mekanisme antrian *batch-jobs* untuk menjamin ketersediaan sumberdaya komputasi (*computing nodes*) saat program dieksekusi. Pengaturan eksekusi program paralel ini ditangani oleh *Sun Grid Engine*[5] yang juga tersedia dalam paket Rocks.

1.1.2 Cluster Fasilkom Universitas Indonesia (UI)

Cluster milik Fakultas Ilmu Komputer (Fasilkom) UI ini berada di jaringan lokal yang tidak bisa diakses langsung dari luar. *Cluster* ini berbasis Linux dan terdiri dari empat buah *nodes* dengan total 32 prosesor.

Pada *cluster* ini sudah tersedia pustaka OpenMPI untuk menjalankan program paralel. Program dijalankan langsung tanpa mekanisme *batch-jobs* seperti pada *cluster* UCSD dengan menjalankan program dari direktori khusus (supaya dapat direplikasi ke seluruh *nodes* pada *cluster*).

[1] www.rocksclusters.org
[2] http://nbcr-233.ucsd.edu/ganglia
[3] https://www.open-mpi.org/
[4] https://www.mpich.org/
[5] http://www.rocksclusters.org/roll-documentation/sge/5.4/

1.1.3 Cluster Rocks pada Laptop

Rocks merupakan distribusi Linux CentOS yang dikustomisasi untuk membangun *cluster high performance computing (HPC)* yang bersifat *opensource*. Rocks menyediakan berbagai macam paket (*rolls*) yang digunakan dalam berbagai *task* HPC.

Dalam eksperimen ini Rocks 6.2 Sidewinder dipakai untuk membangun *cluster* dengan dua buah *virtual machine (VM)* VirtualBox[6] pada *laptop* dengan sistem operasi Ubuntu (Prosesor Intel Core i7-3610QM dengan memori DDR3 8 GB serta penyimpanan HDD 1 TB). Konfigurasi *nodes* pada *cluster* ini adalah sbb:

1. *Front-end Node* (1 CPU, 1GB RAM, 30GB HDD)
   Node ini memiliki GUI (*Graphical User Interface*) berperan sebagai antarmuka pengguna dan manajer dari *compute node*. Program HPC idealnya tidak menggunakan sumberdaya dari *node* ini karena digunakan untuk menjalankan berbagai program antarmuka dan manajemen *cluster* .

2. *Compute Node*: 4 CPU, 1GB RAM, 30GB HDD
   Node ini tidak memiliki GUI dan berperan khusus sebagai sumberdaya komputasi program HPC.

1.2 Lingkungan Pengembangan

Program paralel yang digunakan di dalam eksperimen ini dibuat menggunakan bahasa C yang di-*compile* menggunakan pusaka OpenMPI pada sistem operasi Linux Ubuntu dan Mint.

Kode program-program dan laporan eksperimen ini kami simpan menggunakan layanan GitHub di alamat *https://github.com/yohanesgultom/parallel-programming-assignment*. Hal ini kami lakukan untuk mempermudah kolaborasi dalam pembuatan program dan laporan.

[6]https://www.virtualbox.org/

BAB 2
PERKALIAN MATRIKS & VEKTOR

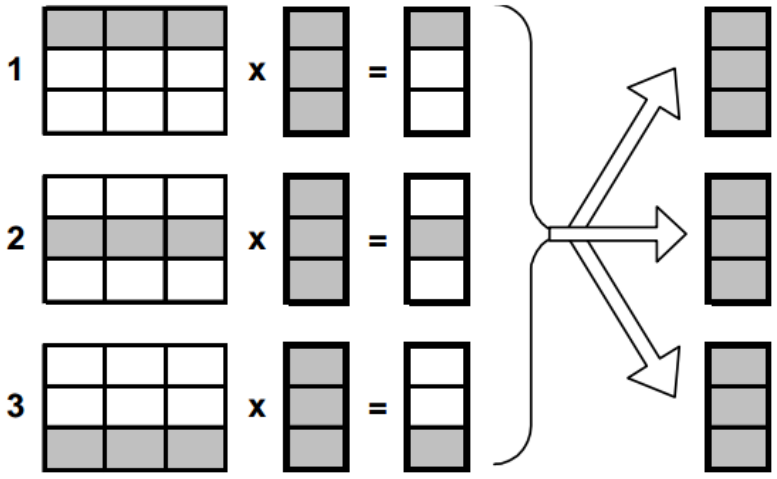
2.1 Pendahuluan

Topik eksperimen pertama adalah perkalian matriks-vektor dan perkalian matriks bujur sangkar dengan beberbagai algoritma paralel.

2.1.1 Perkalian Matriks-Vektor

2.1.1.1 Row-Wise Decomposition

Algoritma paralel perkalian matriks-vektor yang paling sederhana, yaitu memecah proses perkalian berdasarkan baris matriks (*row-wise*). Setiap prosesor akan bertanggung jawab untuk mengalikan sebuah baris matriks dan vektor pada satu waktu. Jika jumlah prosesor (np) lebih sedikit dari jumlah baris matriks (r) maka setiap prosesor bertugas mengalikan $n = \frac{r}{np}$ secara sekuensial.

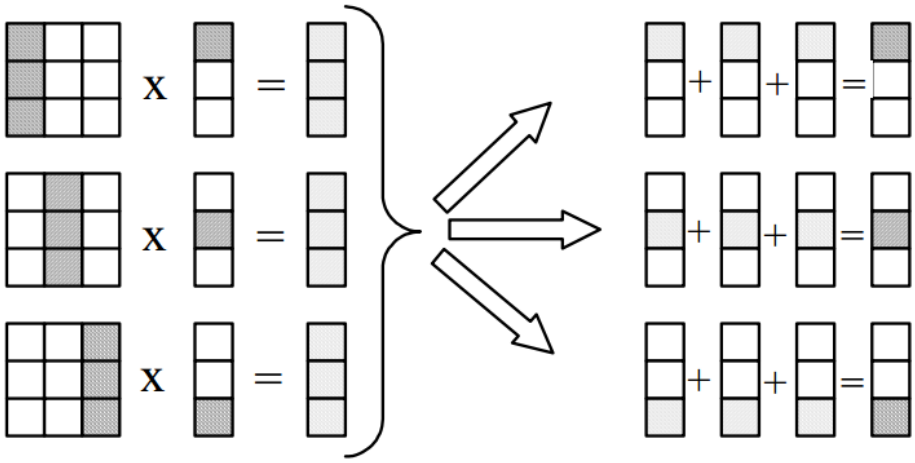


Gambar 2.1: Perkalian matriks-vektor Row-Wise Decomposition

2.1.1.2 Column-wise Decomposition

Algoritma perkalian matriks-vektor ini merupakan alternatif dari *row-wise decomposition* di mana pemecahan proses perkalian dilakukan berdasarkan kolom matriks. Setiap proses akan mengalikan sebuah kolom matriks dan sebuah elemen

vektor pada satu waktu. Mirip dengan algoritma *row-wise decomposition*, jika jumlah prosesor (np) lebih sedikit dari jumlah kolom matriks (c) maka setiap prosesor bertugas mengalikan $n = \frac{c}{np}$ secara sekuensial.



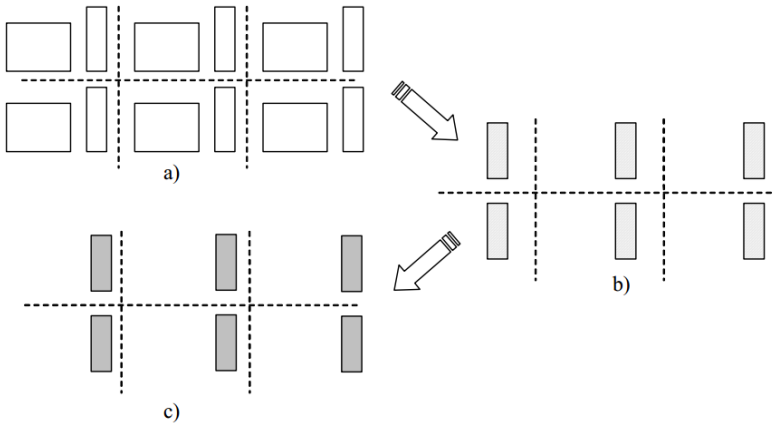
Gambar 2.2: Perkalian matriks-vektor Column-Wise Decomposition

2.1.1.3 Checkerboard Decomposition

Algoritma perkalian matriks-vektor *checkerboard decomposition* ini membagi matriks menjadi submatriks dengan ukuran yang sama dan mengalikannya dengan subvektor yang sesuai. Hasil perkalian tersebut kemudian akan dijumlahkan dan dipetakan ke vektor hasil.

Prekondisi dari algoritma ini adalah jumlah elemen matriks (n) harus bisa dibagi rata ke sejumlah prosesor (np) atau dengan kata lain memenuhi persamaan 2.1. Hasil pembagian ini (x) akan menjadi ukuran submatriks (dan subvektor) yang dikerjakan di tiap proses secara paralel.

$$x = \sqrt{\frac{n}{np}}, \text{ where } x \in \mathbb{Z} \tag{2.1}$$

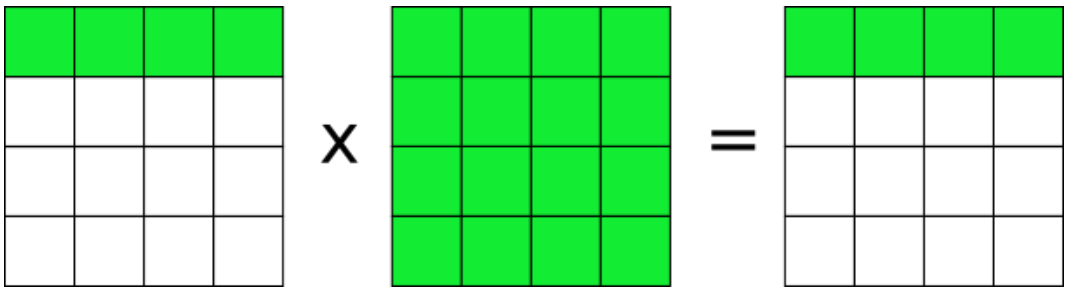


Gambar 2.3: Perkalian matriks-vektor Checkerboard Decomposition

2.1.2 Perkalian Matriks Bujursangkar

2.1.2.1 Row-Wise Decomposition

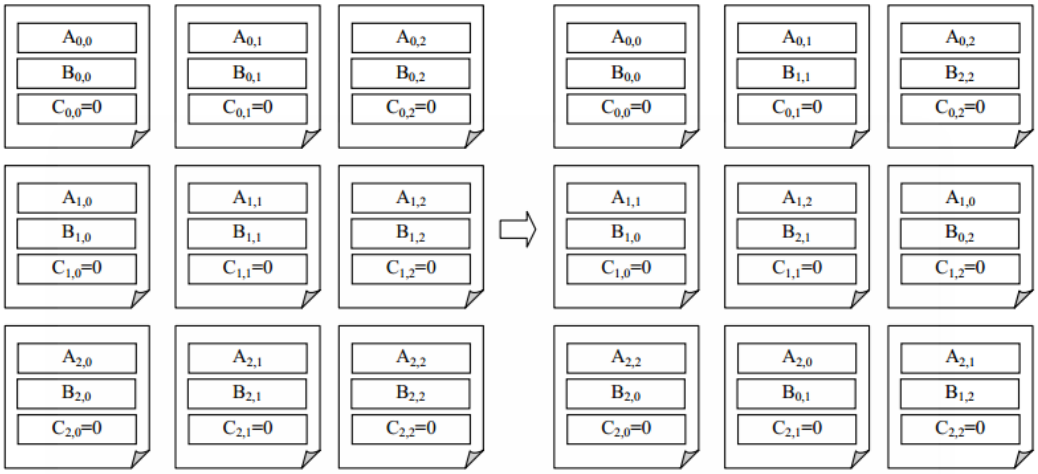
Mirip dengan algoritma perkalian matriks-vektor *row-wise decomposition*, algortima ini juga membagi pekerjaan berdasarkan baris dari matriks pertama seperti yang diilustrasikan pada gambar 2.4. Setiap prosesor akan mengalikan sebuah baris dari matriks pertama A dengan seluruh kolom dari matriks kedua B. Hasil dari seluruh prosesor kemudian dikonkatenasi menjadi matriks baru C.



Gambar 2.4: Perkalian matriks bujursangkar Row-Wise Decomposition

2.1.2.2 Cannon

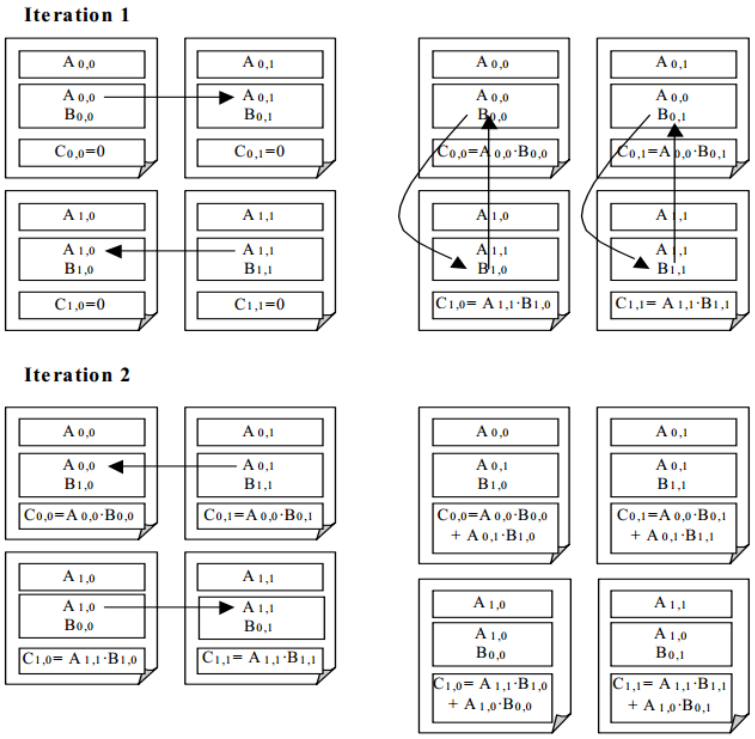
Algoritma Cannon menggunakan dekomposisi seperti algortima matriks-vektor *checkerboard decomposition* di mana matriks A dan B dibagi menjadi menjadi submatriks bujursangkar. Perbedaan pada algoritma Cannon adalah prekondisi di mana jumlah proses (np) harus merupakan bujursangkar sempurna (*perfect square*). Tujuan utama dari algoritma ini adalah untuk meningkatkan efisiensi penggunaan memori pada proses paralel.



Gambar 2.5: Perkalian matriks bujursangkar Cannon

2.1.2.3 Fox

Algoritma Fox memiliki kemiripan dengan algoritma Cannon dalam hal dekomposisi matriks menjadi submatriks bujursangkar dan pemetaan prosesor yang harus dapat membentuk bujursangkar sempurna (*perfect square*). Yang membedakan algoritma Fox dan Cannon adalah skema distribusi awal submatriks ke prosesor

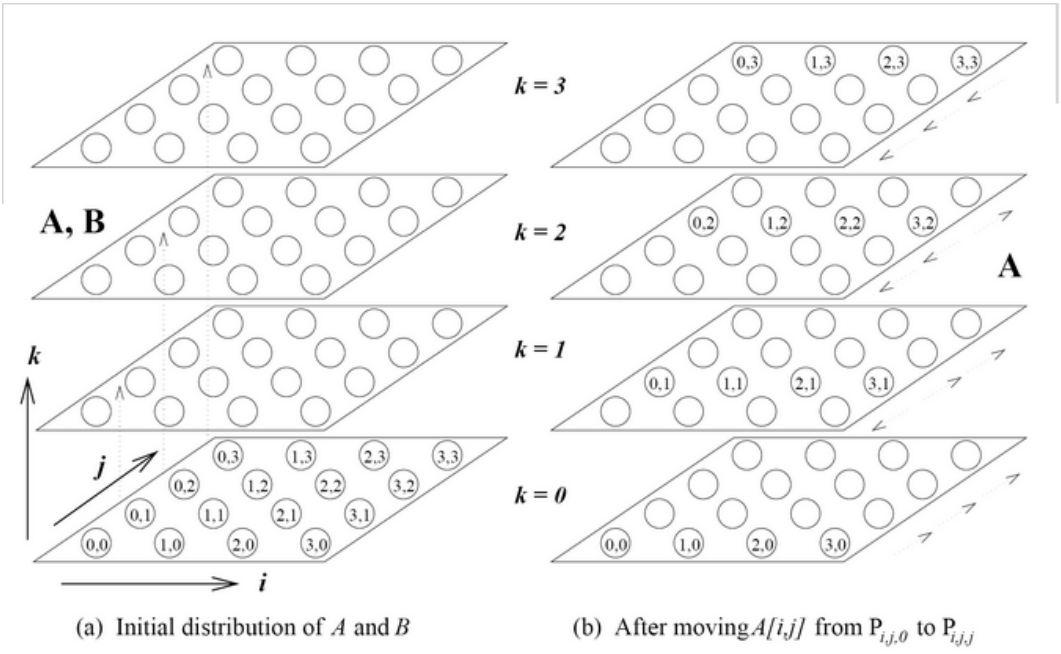


Gambar 2.6: Perkalian matriks bujursangkar Fox

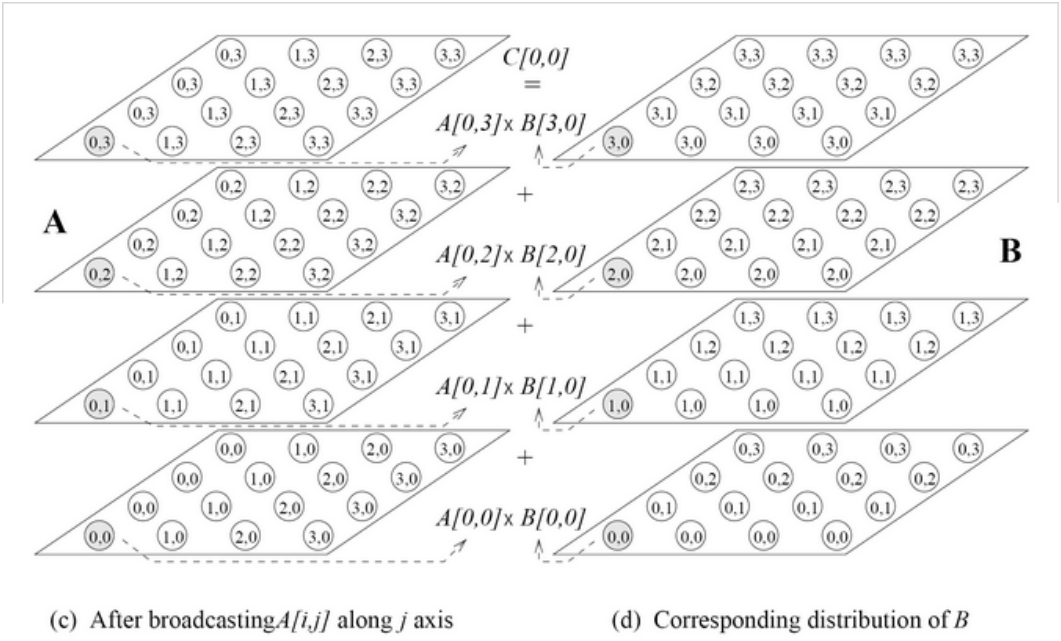
2.1.2.4 DNS

Algoritma yang diberi nama berdasarkan nama pembuatnya (Dekel, Nassimi and Aahni) ini, diajukan dalam rangka meningkatkan lagi efisiensi penggunaan memori pada perkalian matriks bujursangkar secara paralel. Karakteristik algoritma ini adalah:

- Berdasarkan partisi *intermediate data*
- Melakukan perkalian skalar n^3 sehingga membutuhkan proses sebanyak $n \times n \times n$
- Membutuhkan waktu komputasi $O(\log n)$ dengan menggunakan $O(\frac{n^3}{\log n})$



Gambar 2.7: Perkalian matriks bujursangkar DNS iterasi 1



Gambar 2.8: Perkalian matriks bujursangkar DNS iterasi 2

2.2 Eksperimen

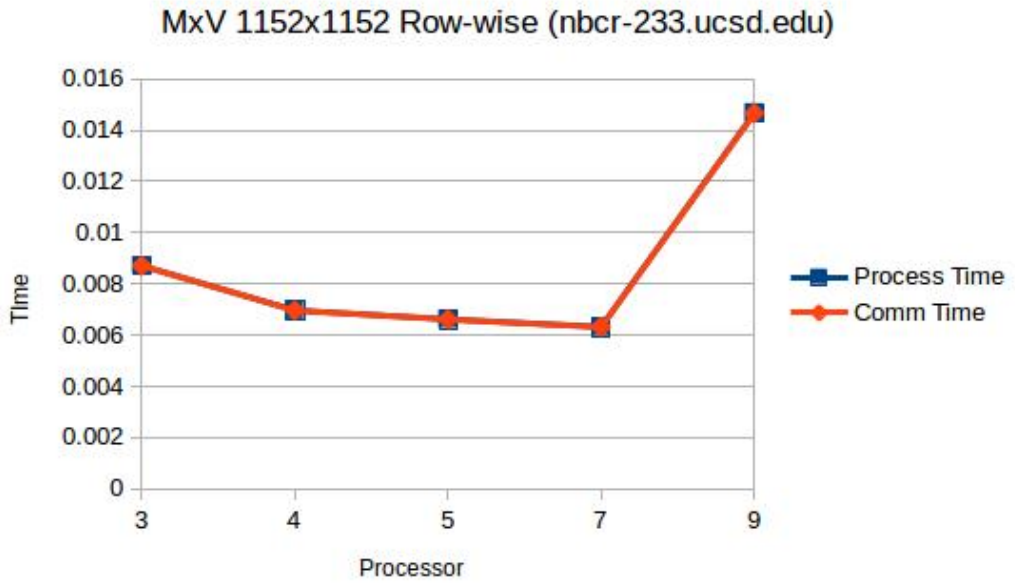
2.2.1 Perkalian Matriks-Vektor

2.2.1.1 Row-Wise Decomposition

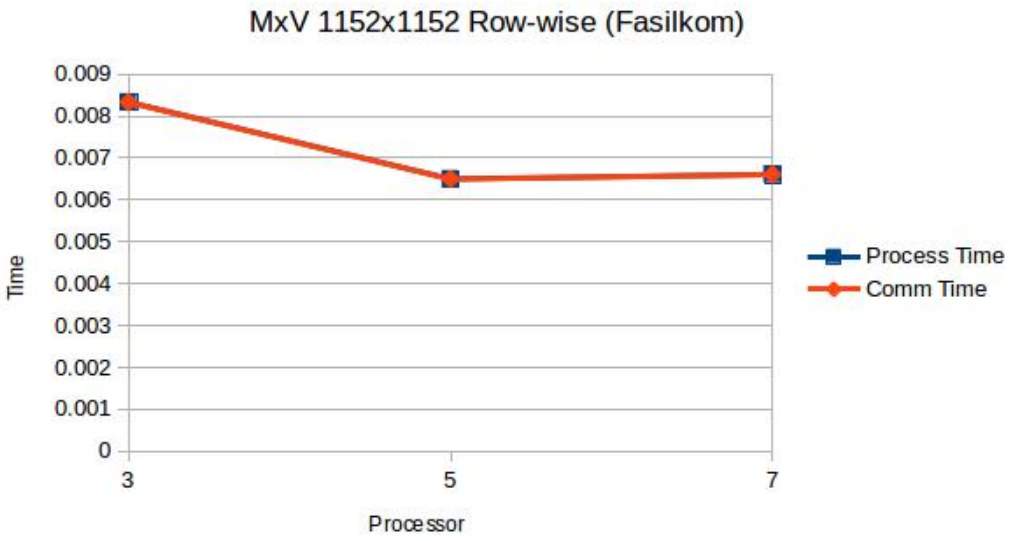
Deskripsi program yang digunakan:

- Sumber kode: `https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/problem1/mv_rowwise.c`
- Satu prosesor berlaku sebagai *manager* dan sisanya berperan sebagai *worker*
- Tugas *manager* adalah menginisialisasi matriks dan vektor, mendistribusikannya secara *row-wise decomposition* menggunakan `MPI_Send` dan `MPI_Bcast` dan mengumpulkan hasil dari tiap *worker* dengan `MPI_Recv`
- Waktu eksekusi dan komunikasi dihitung (dalam detik) menggunakan `MPI_Wtime`

Eksperimen dilakukan di *cluster* UCSD dan Fasilkom dengan variasi jumlah prosesor di mana waktu yang diukur adalah nilai rata-rata dari lima kali eksekusi.



Gambar 2.9: Grafik hasil eksperimen Row-Wise Decomposition Cluster UCSD



Gambar 2.10: Grafik hasil eksperimen Row-Wise Decomposition Cluster Fasilkom UI

Pada grafik gambar 2.9, terlihat bahwa pada *cluster* UCSD terjadi penurunan waktu eksekusi/komunikasi (*speed-up*) ketika digunakan 3, 4, 5, 7 prosesor (2, 3, 4, 6 *worker*). Tapi ketika digunakan 9 prosesor (8 *worker*), waktu yang dibutuhkan malah meningkat (tidak terjadi *speed-up*). Kami tidak mencoba lebih dari 9 prosesor karena sepertinya hasilnya akan sama (tidak ada *speed-up*).

Sedangkan pada *cluster* Fasilkom UI, kami hanya mencoba 3, 5, 7 prosesor karena masalah *permission* yang mengakibatkan hanya satu *node* (8 prosesor) yang

bisa digunakan. Seperti yang terlihat pada grafik gambar 2.10, *speed-up* hanya terjadi pada prosesor 3-5. Sedangkan dari 5-7 prosesor tidak terjadi *speed-up* (waktu ekskusi hampir sama).

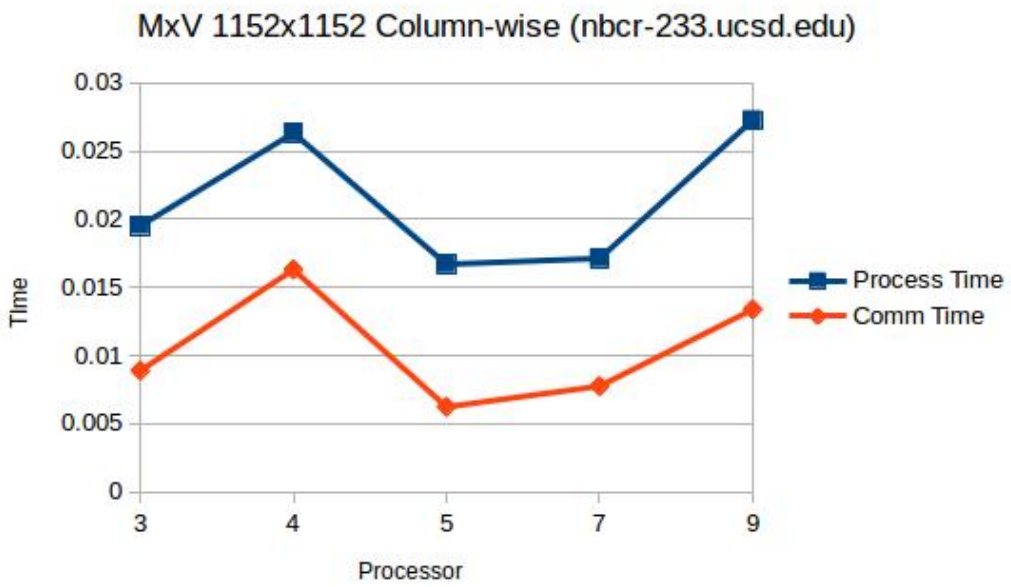
Kami tidak melakukan pengujian pada *cluster* Rocks karena keterbatasan jumlah prosesor pada *laptop* yang kami gunakan.

2.2.1.2 Column-wise Decomposition

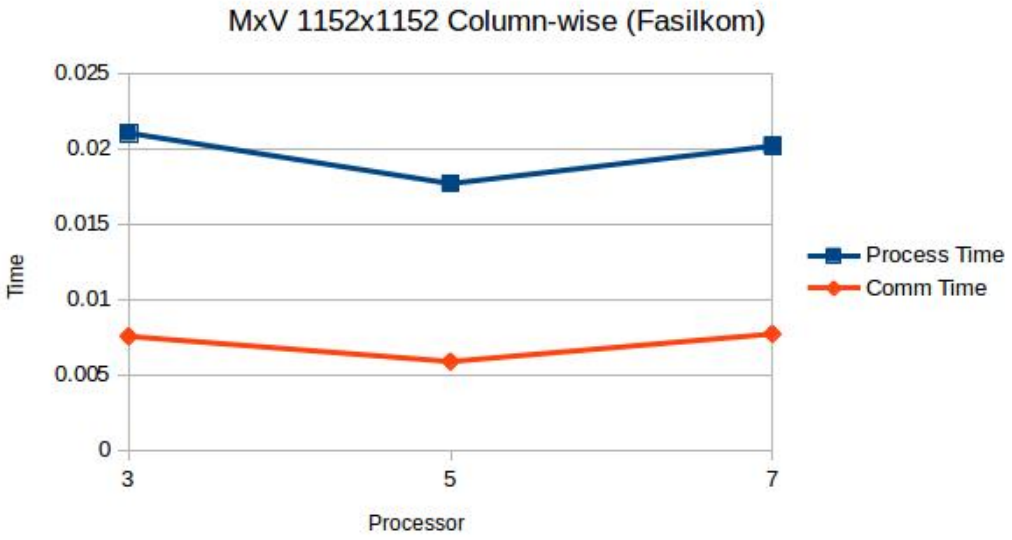
Deskripsi program yang digunakan:

- Sumber kode: `https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/problem1/mv_columnwise.c`
- Satu prosesor berlaku sebagai *manager* dan sisanya berperan sebagai *worker*
- Tugas *manager* adalah menginisialisasi matriks dan vektor, mendistribusikannya secara *column-wise decomposition* menggunakan `MPI_Send` dan `MPI_Bcast` dan mengumpulkan hasil dari tiap *worker* dengan `MPI_Recv`
- Waktu eksekusi dan komunikasi dihitung (dalam detik) menggunakan `MPI_Wtime`

Eksperimen dilakukan di *cluster* UCSD dan Fasilkom dengan variasi jumlah prosesor di mana waktu yang diukur adalah nilai rata-rata dari lima kali eksekusi.



Gambar 2.11: Grafik hasil eksperimen Column-Wise Decomposition Cluster UCSD



Gambar 2.12: Grafik hasil eksperimen Column-Wise Decomposition Cluster Fasilkom UI

Pada grafik gambar 2.11, terlihat bahwa pada *cluster* UCSD terjadi penurunan waktu eksekusi/komunikasi (*speed-up*) ketika digunakan 4-5 prosesor. Tapi ketika digunakan 3-4, 5-9 prosesor, waktu yang dibutuhkan malah meningkat (tidak terjadi *speed-up*).

Sedangkan pada *cluster* Fasilkom UI, kami hanya mencoba 3, 5, 7 prosesor karena masalah *permission* yang mengakibatkan hanya satu *node* (8 prosesor) yang bisa digunakan. Seperti yang terlihat pada grafik gambar 2.12, *speed-up* hanya terjadi pada prosesor 3-5. Sedangkan dari 5-7 prosesor tidak terjadi *speed-up* (waktu ekskusi meningkat).

Kami tidak melakukan pengujian pada *cluster* Rocks karena keterbatasan jumlah prosesor pada *laptop* yang kami gunakan.

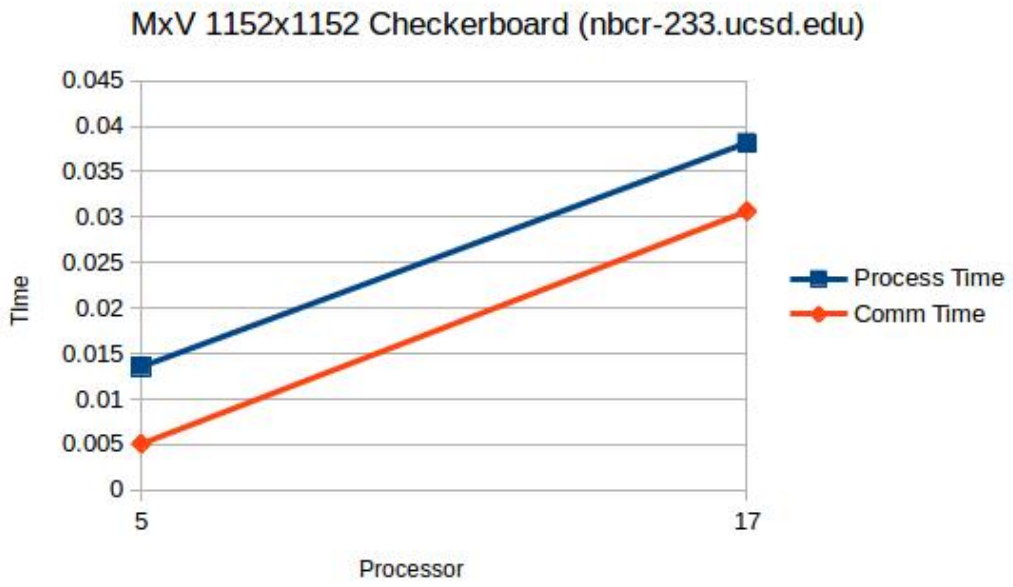
2.2.1.3 Checkerboard Decomposition

Deskripsi program yang digunakan:

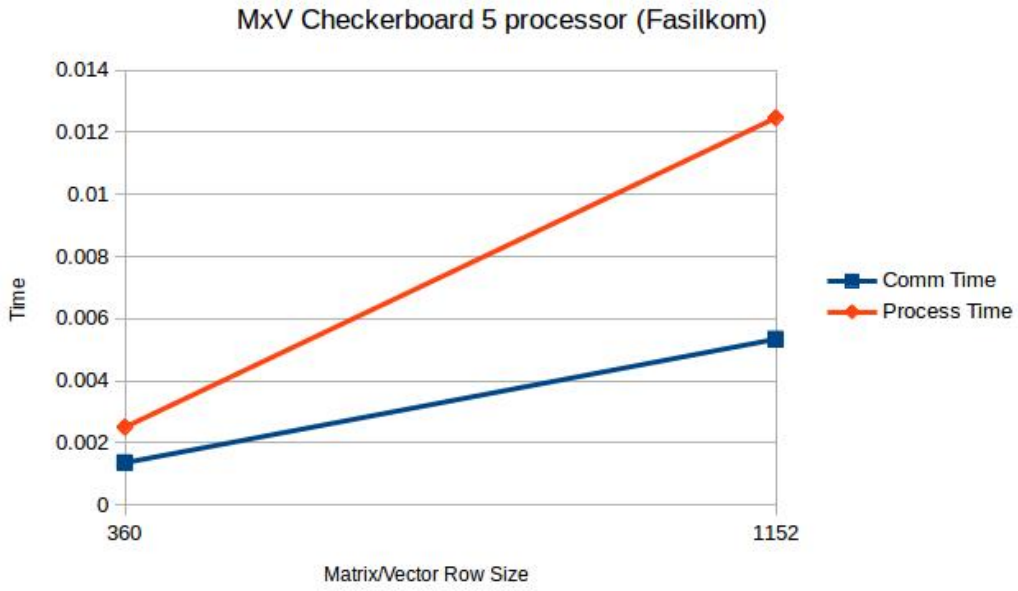
- Sumber kode: `https://github.com/yohanesgultom/parallel-programming-assignment/blob/master/problem1/mv_checkerboard.c`
- Satu prosesor berlaku sebagai *manager* dan sisanya berperan sebagai *worker*
- Tugas *manager* adalah menginisialisasi matriks dan vektor, mendistribusikannya secara *checkerboard decomposition* menggunakan `MPI_Send` dan `MPI_Bcast` dan mengumpulkan hasil dari tiap *worker* dengan `MPI_Recv`

- Waktu eksekusi dan komunikasi dihitung (dalam detik) menggunakan `MPI_Wtime`

Eksperimen dilakukan di *cluster* UCSD dan Fasilkom dengan variasi jumlah prosesor di mana waktu yang diukur adalah nilai rata-rata dari lima kali eksekusi.



Gambar 2.13: Grafik hasil eksperimen Checkerboard Cluster UCSD



Gambar 2.14: Grafik hasil eksperimen Checkerboard Cluster Fasilkom UI

Pada grafik gambar 2.13, terlihat bahwa waktu yang dibutuhkan malah

meningkat (tidak terjadi *speed-up*) ketika kami meningkatkan jumlah prosesor dari 5-7. Kami tidak mencoba prosesor yang lebih banyak karena melihat pola yang sama (tidak ada *speed-up*).

Sedangkan pada *cluster* Fasilkom UI, kami hanya mencoba 5 prosesor karena masalah *permission* yang mengakibatkan hanya satu *node* (8 prosesor) yang bisa digunakan. Seperti yang terlihat pada grafik gambar 2.14, kami hanya mencoba melakukan eksekusi dengan ukuran matriks/vektor yang berbeda (360x360 dan 1152x1152).

Kami tidak melakukan pengujian pada *cluster* Rocks karena keterbatasan jumlah prosesor pada *laptop* yang kami gunakan.

2.2.2 Perkalian Matriks Bujursangkar

2.2.2.1 Row-Wise Decomposition

2.2.2.2 Cannon

2.2.2.3 Fox

2.2.2.4 DNS

BAB 3
PROCESS TOPOLOGIES & DYNAMIC PROCESS GENERATION

@todo

tambahkan kata-kata pengantar bab 2 disini

3.1 LaTeX Secara Singkat

Berdasarkan [**?**]:

> LaTeX is a family of programs designed to produce publication-quality typeset documents. It is particularly strong when working with mathematical symbols.
> The history of LaTeX begins with a program called TEX. In 1978, a computer scientist by the name of Donald Knuth grew frustrated with the mistakes that his publishers made in typesetting his work. He decided to create a typesetting program that everyone could easily use to typeset documents, particularly those that include formulae, and made it freely available. The result is TEX. Knuth's product is an immensely powerful program, but one that does focus very much on small details. A mathematician and computer scientist by the name of Leslie Lamport wrote a variant of TEX called LaTeX that focuses on document structure rather than such details.

Dokumen LaTeX sangat mudah, seperti halnya membuat dokumen teks biasa. Ada beberapa perintah yang diawali dengan tanda '\'. Seperti perintah \\ yang digunakan untuk memberi baris baru. Perintah tersebut juga sama dengan perintah \newline. Pada bagian ini akan sedikit dijelaskan cara manipulasi teks dan perintah-perintah LaTeX yang mungkin akan sering digunakan. Jika ingin belajar hal-hal dasar mengenai LaTeX, silahkan kunjungi:

- http://frodo.elon.edu/tutorial/tutorial/, atau
- http://www.maths.tcd.ie/~dwilkins/LaTeXPrimer/

3.2 LaTeX Kompiler dan IDE

Agar dapat menggunakan LaTeX (pada konteks hanya sebagai pengguna), Anda tidak perlu banyak tahu mengenai hal-hal didalamnya. Seperti halnya pembuatan dokumen secara visual (contohnya Open Office (OO) Writer), Anda dapat menggunakan LaTeX dengan cara yang sama. Orang-orang yang menggunakan LaTeX relatif lebih teliti dan terstruktur mengenai cara penulisan yang dia gunakan, LaTeX memaksa Anda untuk seperti itu.

Kembali pada bahasan utama, untuk mencoba LaTeX Anda cukup mendownload kompiler dan IDE. Saya menyarankan menggunakan Texlive dan Texmaker. Texlive dapat didownload dari http://www.tug.org/texlive/. Sedangkan Texmaker dapat didownload dari http://www.xm1math.net/texmaker/. Untuk pertama kali, coba buka berkas thesis.tex dalam template yang Anda miliki pada Texmaker. Dokumen ini adalah dokumen utama. Tekan F6 (PDFLaTeX) dan Texmaker akan mengkompilasi berkas tersebut menjadi berkas PDF. Jika tidak bisa, pastikan Anda sudah menginstall Texlive. Buka berkas tersebut dengan menekan F7. Hasilnya adalah sebuah dokumen yang sama seperti dokumen yang Anda baca saat ini.

3.3 Bold, Italic, dan Underline

Hal pertama yang mungkin ditanyakan adalah bagaimana membuat huruf tercetak tebal, miring, atau memiliki garis bawah. Pada Texmaker, Anda bisa melakukan hal ini seperti halnya saat mengubah dokumen dengan OO Writer. Namun jika tetap masih tertarik dengan cara lain, ini dia:

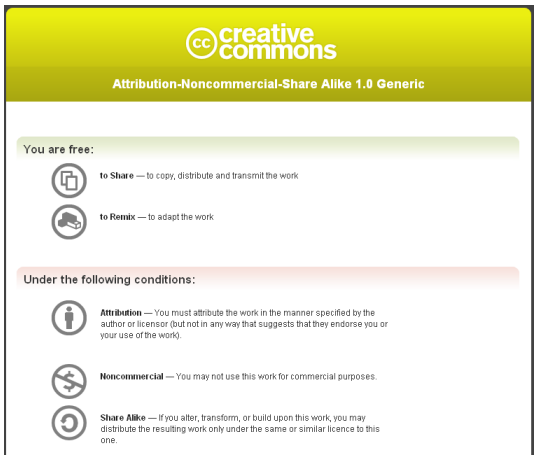
- **Bold**
  Gunakan perintah \textbf{} atau \bo{}.
- *Italic*
  Gunakan perintah \textit{} atau \f{}.
- Underline
  Gunakan perintah \underline{}.
- $\overline{Overline}$
  Gunakan perintah \overline.
- ^{superscript}
  Gunakan perintah \{}.

- *subscript*
  Gunakan perintah _{}.

Perintah \f dan \bo hanya dapat digunakan jika package uithesis digunakan.

3.4 Memasukan Gambar

Setiap gambar dapat diberikan caption dan diberikan label. Label dapat digunakan untuk menunjuk gambar tertentu. Jika posisi gambar berubah, maka nomor gambar juga akan diubah secara otomatis. Begitu juga dengan seluruh referensi yang menunjuk pada gambar tersebut. Contoh sederhana adalah Gambar 3.1. Silahkan lihat code LaTeX dengan nama bab2.tex untuk melihat kode lengkapnya. Harap diingat bahwa caption untuk gambar selalu terletak dibawah gambar.



Gambar 3.1: *Creative Common License 1.0 Generic.*

3.5 Membuat Tabel

Seperti pada gambar, tabel juga dapat diberi label dan caption. Caption pada tabel terletak pada bagian atas tabel. Contoh tabel sederhana dapat dilihat pada Tabel 3.1.

Tabel 3.1: Contoh Tabel

	kol 1	kol 2
baris 1	1	2
baris 2	3	4
baris 3	5	6
jumlah	9	12

Ada jenis tabel lain yang dapat dibuat dengan LaTeX berikut beberapa diantaranya. Contoh-contoh ini bersumber dari http://en.wikibooks.org/wiki/LaTeX/Tables

Tabel 3.2: An Example of Rows Spanning Multiple Columns

No	Name	Week 1			Week 2		
		A	B	C	A	B	C
1	Lala	1	2	3	4	5	6
2	Lili	1	2	3	4	5	6
3	Lulu	1	2	3	4	5	6

Tabel 3.3: An Example of Columns Spanning Multiple Rows

Percobaan	Iterasi	Waktu
Pertama	1	0.1 sec
Kedua	1	0.1 sec
	3	0.15 sec
Ketiga	1	0.09 sec
	2	0.16 sec
	3	0.21 sec

Tabel 3.4: An Example of Spanning in Both Directions Simultaneously

		Title			
		A	B	C	D
Type	X	1	2	3	4
	Y	0.5	1.0	1.5	2.0
Resource	I	10	20	30	40
	J	5	10	15	20

BAB 4
CONJUGATE GRADIENT

4.1 Pendahuluan

Conjugate gradient method digunakan untuk menyelesaikan persamaan linear $Ax = b$ di mana matriks koefisiennya bersifat simetris dan definit positif. Matriks A n x n dikatakan simetris jika $a_{ij} = a_{ji}$ untuk $i, j = 1, ..., n$. Matriks A dikatakan definit positif jika untuk setiap vektor x bukan nol, perkalian skalar $x \cdot Ax$ menghasilkan nilai lebih besar dari nol. Algoritma conjugate gradient method ditunjukkan pada Gambar 4.1. r_k merupakan sisa atau selisih antara b dengan Ax_k, sedangkan p_k merupakan *search direction*.

$k = 0$; $x_0 = 0$; $r_0 = b$
while $(\|r_k\|^2 > \text{tolerance})$ **and** $(k < \text{max\_iter})$
    $k++$
    if $k = 1$
        $p_1 = r_0$
    else
        $\beta_k = \frac{r_{k-1} \cdot r_{k-1}}{r_{k-2} \cdot r_{k-2}}$ // minimize $\|p_k - r_{k-1}\|$
        $p_k = r_{k-1} + \beta_k p_{k-1}$
    endif
    $s_k = Ap_k$
    $\alpha_k = \frac{r_{k-1} \cdot r_{k-1}}{p_k \cdot s_k}$ // minimize $q(x_{k-1} + \alpha p_k)$
    $x_k = x_{k-1} + \alpha_k p_k$
    $r_k = r_{k-1} - \alpha_k s_k$
endwhile
$x = x_k$

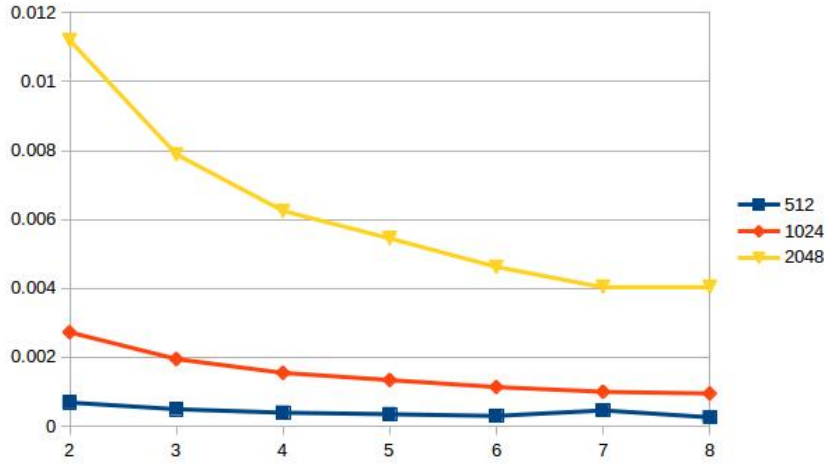
Gambar 4.1: Algoritma Conjugate Gradient Method.

Pada implementasi paralel CGM, matriks A akan didistribusikan menggunakan fungsi `MPI_scatter` kepada setiap proses dan masing-masing proses mendapat sebanyak n x n / p data. Algoritma tersebut dijalankan di setiap proses untuk setiap bagian distribusi matriks A.

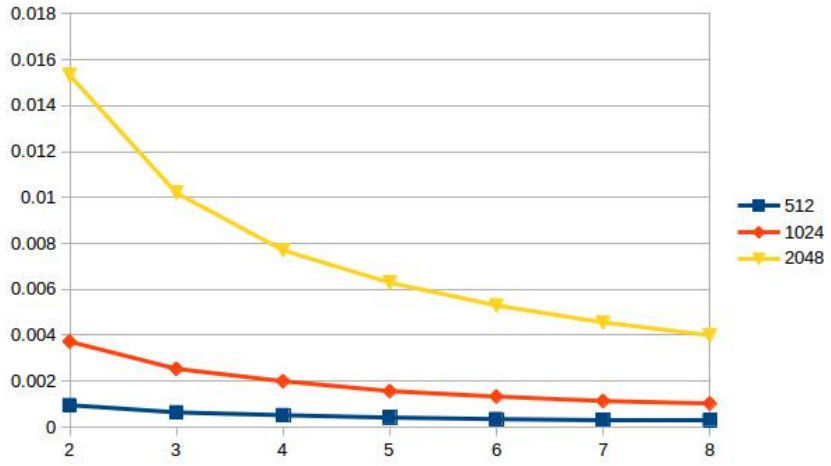
4.2 Eksperimen

Implementasi algoritma paralel CGM menggunakan kode sumber yang dibangun oleh Joseph Tanigawa yang dipublikasikan melalui Github. Eksperimen dilakukan

di lingkungan cluster Fasilkom dan nbcr-233.ucsd.edu. Percobaan dilakukan dengan variasi jumlah prosesor dan besar data. Banyaknya prosesor yang digunakan mulai dari 2 sampai dengan 8 sedangkan besar data yang digunakan adalah 512, 1024, dan 2048. Gambar 4.2 dan Gambar 4.3 berturut-turut menunjukkan hasil unjuk kerja implementasi algoritma paralel CGM di cluster Fasilkom dan nbcr-233.ucsd.edu. Pada grafik tersebut menunjukkan *execution time* per iterasi karena setiap eksperimen menghasilkan jumlah iterasi yang berbeda-beda. Pada eksperimen tersebut kami atur jumlah maksimum iterasi yang diijinkan adalah 1000 dengan toleransi (r) sebesar 1e-06.



Gambar 4.2: Hasil eksperimen paralel CG pada cluster Fasilkom.



Gambar 4.3: Hasil eksperimen paralel CG pada cluster nbcr-233.ucsd.edu.

Dari grafik tersebut menunjukkan bahwa paralelisasi algoritma CGM menghasilkan *speed up*. Meski terdapat penyimpangan saat melakukan eksperimen di cluster Fasilkom namun secara keseluruhan terdapat peningkatan performa.

BAB 5
MOLECULAR DYNAMICS: AMBER

@todo
tambahkan kata-kata pengantar bab 1 disini

5.1 thesis.tex

Berkas ini berisi seluruh berkas Latex yang dibaca, jadi bisa dikatakan sebagai berkas utama. Dari berkas ini kita dapat mengatur bab apa saja yang ingin kita tampilkan dalam dokumen.

5.2 laporan_setting.tex

Berkas ini berguna untuk mempermudah pembuatan beberapa template standar. Anda diminta untuk menuliskan judul laporan, nama, npm, dan hal-hal lain yang dibutuhkan untuk pembuatan template.

5.3 istilah.tex

Berkas istilah digunakan untuk mencatat istilah-istilah yang digunakan. Fungsinya hanya untuk memudahkan penulisan. Pada beberapa kasus, ada kata-kata yang harus selalu muncul dengan tercetak miring atau tercetak tebal. Dengan menjadikan kata-kata tersebut sebagai sebuah perintah LaTeX tentu akan mempercepat dan mempermudah pengerjaan laporan.

5.4 hype.indonesia.tex

Berkas ini berisi cara pemenggalan beberapa kata dalam bahasa Indonesia. LaTeX memiliki algoritma untuk memenggal kata-kata sendiri, namun untuk beberapa kasus algoritma ini memenggal dengan cara yang salah. Untuk memperbaiki pemenggalan yang salah inilah cara pemenggalan yang benar ditulis dalam berkas hype.indonesia.tex.

5.5 pustaka.tex

Berkas pustaka.tex berisi seluruh daftar referensi yang digunakan dalam laporan. Anda bisa membuat model daftar referensi lain dengan menggunakan bibtex. Untuk mempelajari bibtex lebih lanjut, silahkan buka http://www.bibtex.org/Format. Untuk merujuk pada salah satu referensi yang ada, gunakan perintah \cite, e.g. \cite{latex.intro} yang akan akan memunculkan [**?**]

5.6 bab[1 - 6].tex

Berkas ini berisi isi laporan yang Anda tulis. Setiap nama berkas e.g. bab1.tex merepresentasikan bab dimana tulisan tersebut akan muncul. Sebagai contoh, kode dimana tulisan ini dibaut berada dalam berkas dengan nama bab4.tex. Ada enam buah berkas yang telah disiapkan untuk mengakomodir enam bab dari laporan Anda, diluar bab kesimpulan dan saran. Jika Anda tidak membutuhkan sebanyak itu, silahkan hapus kode dalam berkas thesis.tex yang memasukan berkas LaTeX yang tidak dibutuhkan; contohnya perintah \include{bab6.tex} merupakan kode untuk memasukan berkas bab6.tex kedalam laporan.

BAB 6
KONTRIBUSI

@todo
Tambahkan kesimpulan dan saran terkait dengan perkerjaan yang dilakukan.

6.1 Kesimpulan

6.2 Saran

LAMPIRAN 1