BSH

DEPARTEMEN MATEMATIKA
FMIPA - INSTITUT PERTANIAN BOGOR

ISSN:
1412-677X

Journal of Mathematics and Its Applications

J M A

Jurnal Matematika dan Aplikasinya

Volume 11, No. 2
Desember 2012

Alamat Redaksi :
Departemen Matematika
FMIPA –Institut Pertanian Bogor
Jln. Meranti, Kampus IPB
Dramaga - Bogor

Phone/Fax:(0251) 8625276
E-mail: math@ipb.ac.id

Vol. 11, No. 2, Desember 2012 **ISSN : 1412-677X**

Journal of Mathematics and Its Applications

JMA

Jurnal Matematika dan Aplikasinya

ALAMAT REDAKSI:
Departemen Matematika
FMIPA – Institut Pertanian Bogor
Jln. Meranti, Kampus IPB Dramaga
Bogor
Phone./Fax: (0251) 625276
Email:math@ipb.ac.id

JMA merupakan media yang memuat informasi hasil penelitian matematika baik murni maupun terapan, bagi para matematikawan atau para pengguna matematika. JMA diterbitkan dua kali (dua nomor) setiap tahun (periode Juli dan Desember).

Harga langganan per volume, termasuk biaya pos, Vol.9, No.1 dan 2:
Institusi/Perpustakaan Rp. 350.000,- (dalam IPB), Rp. 500.000,- (luar IPB)
Staf/Perorangan Rp. 200.000,- (dalam IPB), Rp.250.000,- (luar IPB)
Mahasiswa Rp. 75.000,-
Penulis makalah yang diterima dikenai biaya administrasi Rp.25.000,- per lembar

Semua pembayaran biaya dapat ditransfer melalui:

Retno Budiarti
BNI Cabang Bogor
No. Rek. 000291007-5

TATA CARA PENULISAN MAKALAH

JMA menerima makalah dalam bahasa Indonesia atau bahasa Inggris. Makalah dapat dikirim melalui pos (berupa 2 hard copy beserta disketnya) atau lewat email ke alamat berikut:

Redaksi JMA
Departemen Matematika
FMIPA – Institut Pertanian Bogor
Jln. Meranti, Kampus IPB Dramaga
Bogor
Phone/Fax: (0251) 625276
Email:math@ipb.ac.id

Makalah yang orisinil berupa hasil penelitian matematika murni atau terapan mendapat prioritas utama untuk diterima. Tulisan yang bersifat review juga bisa diterima. Makalah akan diseleksi oleh redaksi, dan hasil seleksi akan diinformasikan.

Makalah ditulis dengan LATEX/TEX/MS-WORD dengan kualitas baik, format A4, tidak boleh bolak balik, spasi satu, font 12, margin kiri 4 cm, margin kanan 3 cm. Margin atas dan bawah 4 cm. Maksimum jumlah halaman 20 yang didalamnya termasuk tabel, illustrasi, dan gambar.

Judul makalah dibuat singkat, jelas, dan merepresentasikan isi makalah. Nama penulis diletakkan di bawah judul, diikuti nama instansi (bila ada), dan alamat (termasuk email jika ada).

Abstrak diletakkan di bawah nama dan alamat penulis, ditulis tidak melebihi 250 kata, meringkas hasil yang diperoleh dan metode yang digunakan. Di bawah abstrak boleh diletakkan kata kunci. Kata kunci terdiri atas satu kata atau lebih yang merupakan istilah yang paling dominan digunakan dan merupakan istilah yang paling menentukan isi tulisan.

Acknowledgment/Ungkapan Terima Kasih ditulis pada akhir tulisan sebelum referensi.

Referensi/Daftar Pustaka diletakkan pada akhir tulisan setelah Ungkapan Terima Kasih, penulisan mengikuti pola pada contoh berikut ini dengan pengurutan naik didasarkan abjad huruf pertama pada nama belakang penulis pertama.

[1] S. Guritman, F. Hooweg, and J. Simonis, "The Degree of Functions and Weights in Linear Codes, "*Discrete Applied Mathematics*, vol.111, no. 1, pp. 87-102, 2001
[2] J.H. van Lint, *Introduction to Coding Theory*, 2nd ed. Berlin, Germany, Springer-Verlag, 1992.

Illustrasi atau *Gambar* sedapat mungkin ditempatkan pada badan tulisan mengikuti apa yang diillustrasikan atau yang digambarkan. Jika itu tidak mungkin, boleh juga ditempatkan setelah referensi. Tidak ada illustrasi atau gambar yang ditulis tangan.

Journal of Mathematics and Its Applications

JMA

Jurnal Matematika dan Aplikasinya

Vol. 9 No.2, Desember 2012

Journal of Mathematics and Its Applications

JMA

Jurnal Matematika dan Aplikasinya

DAFTAR ISI

TEKNIK REKONSTRUKSI ALJABAR UNTUK MENYELESAIKAN SISTEM PERSAMAAN LINEAR DENGAN SCILAB

RUHIYAT[1], M. ILYAS[1], A.D. GARNADI[1], S.NURDIATI[1]

[1] *Departemen Matematika,*
Institut Pertanian Bogor, Bogor, 16680

ABSTRAK

Dalam artikel ini diuraikan program art.sci, sebuah program Scilab untuk menyelesaikan sistem persamaan linear berukuran sembarang. Program ini merupakan implementasi dari metode teknik rekonstruksi aljabar (TRA). Uji coba numerik TRA menggunakan sistem persamaan linear yang berasal dari diskretisasi persamaan integral jenis pertama. Salah satu obyektif tulisan ini, adalah menyediakan alat untuk menyelesaikan sistem persamaan linear sembarang di Scilab sebagai sebuah lingkungan komputasional yang bebas (free).

1. Pendahuluan

Teknik Rekonstruksi Aljabar (TRA) atau ***Algebraic Reconstruction Technique*** (ART) merupakan salah satu metode yang pertama kali digunakan diimplementasikan dalam komputer. Metode ini berupa cara iteratif yang diperkenalkan oleh S. Kaczmarz pada tahun 1937 dalam hasil karyanya yang berjudul "*Angenäherte auflösung von systemen linearer gleichungen*" (Censor, 1983).

Tujuan penulisan ini adalah menyediakan metode penyelesaian system persamaan linear berukuran bebas dalam Scilab, sebuah perangkat lunak ilmiah (Scientific software) yang bersifat bebas. Tulisan ini secara tak langsung mendukung upaya IGOS (Indonesia Goes Open Source). Tulisan ini dibagi atas 2 bagian, pertama adalah deskripsi dari metode TRA, kemudian dilanjutkan dengan uji numerik dari metode TRA untuk menyelesaikan system persamaan linear yang diambil dari diskretisasi persamaan integral jenis pertama. Pada lampiran, diberikan pustaka numerik TRA beserta fungsi ujinya.

2. Algoritme Teknik Rekonstruksi Aljabar untuk menyelesaikan Sistem Persamaan Linear

TRA merupakan salah satu cara untuk menyelesaikan sistem persamaan linear (SPL). Misalkan SPL yang terdiri atas M persamaan dengan N faktor yang tidak diketahui adalah:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1N}x_N = b_1,$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2N}x_N = b_2,$$
$$\vdots$$
$$a_{M1}x_1 + a_{M2}x_2 + \cdots + a_{MN}x_N = b_M.$$

Persamaan linear simultan tersebut dapat dituliskan sebagai

$$\mathbf{a}_i^T\mathbf{x} = b_i, \qquad i = 1, 2, \ldots, M$$

dengan vektor kolom $\mathbf{a}_i$ dan $\mathbf{x}$ secara berturutan adalah

$$\mathbf{a}_i = \begin{bmatrix} a_{i1} \\ a_{i2} \\ \vdots \\ a_{iN} \end{bmatrix}, \qquad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}, \qquad i = 1, 2, \ldots, M.$$

Perhatikan bahwa M persamaan ini adalah **hiperbidang** (***hyperplane***) pada ruang Euclidean berdimensi N, $\mathbb{R}^N$. Selanjutnya, setiap M persamaan ini disebut garis dan bidang secara berturut-turut untuk kasus di $\mathbb{R}^2$ dan $\mathbb{R}^3$.

SPL tidak akan mempunyai sebuah penyelesaian eksak jika M hiperbidangnya tidak memiliki perpotongan yang sama. Oleh karena itu, mencari suatu titik di dalam $\mathbb{R}^N$ yang relatif "dekat" dengan seluruh hiperbidang tersebut menjadi pilihan lain. Titik semacam ini akan menjadi sebuah penyelesaian hampiran atas SPL.

Proses iterasi pada algoritme berikut (lihat [1]) akan menghasilkan siklus rangkaian proyeksi ortogonal yang berurutan pada M hiperbidang yang dimulai dari sebarang titik awal di $\mathbb{R}^N$. Sebelumnya, diperkenalkan terlebih dahulu notasi untuk iterasi yang berurutan ini. Misalkan $\mathbf{x}_k^{(p)}$ adalah titik yang terletak pada hiperbidang ke-k yang dihasilkan saat siklus iterasi ke-p. Algoritme untuk hal ini disebut algoritme TRA sebagai berikut ini.

Algoritme TRA

1. Pilihlah titik sebarang di $\mathbb{R}^N$ dan tandai dengan $\mathbf{x}_0$.
2. Untuk siklus iterasi pertama, tetapkan $p = 1$.
3. Untuk $k = 1, 2, \ldots, M$, hitunglah
$$\mathbf{x}_k^{(p)} = \mathbf{x}_{k-1}^{(p)} + \frac{\left(b_k - \mathbf{a}_k^T \mathbf{x}_{k-1}^{(p)}\right)}{\mathbf{a}_k^T \mathbf{a}_k} \mathbf{a}_k.$$
4. Tetapkan $\mathbf{x}_0^{(p+1)} = \mathbf{x}_M^{(p)}$.
5. Naikkan jumlah siklus p sebanyak satu dan kembali ke Langkah 3.

Titik $\mathbf{x}_k^{(p)}$ pada langkah 3 disebut **proyeksi ortogonal** (***orthogonal projection***) dari titik $\mathbf{x}_{k-1}^{(p)}$ pada hiperbidang $\mathbf{a}_k^T \mathbf{x} = b_k$. Algoritme ini menentukan rangkaian proyeksi ortogonal yang berurutan dari satu hiperbidang ke hiperbidang berikutnya. Proyeksi akan kembali pada hiperbidang pertama setelah proyeksi pada hiperbidang terakhir dilakukan. Rumus proyeksi ortogonal disajikan pada teorema berikut.

Teorema (Rumus Proyeksi Ortogonal)

Misalkan L adalah sebuah hiperbidang di dalam $\mathbb{R}^N$ dengan persamaan $\mathbf{a}^T \mathbf{x} = b$ dan misalkan $\mathbf{x}^*$ adalah sebarang titik di dalam $\mathbb{R}^N$, maka proyeksi ortogonal dari $\mathbf{x}^*$ terhadap L, yaitu $\mathbf{x}_p$, dinyatakan dengan

$$\mathbf{x}_p = \mathbf{x}^* + \frac{(b - \mathbf{a}^T \mathbf{x}^*)}{\mathbf{a}^T \mathbf{a}} \mathbf{a}.$$

Titik-titik $\mathbf{x}_M^{(1)}$, $\mathbf{x}_M^{(2)}$, $\mathbf{x}_M^{(3)}$, … yang terletak pada hiperbidang ke-M akan konvergen menuju sebuah titik $\mathbf{x}_M^*$ pada hiperbidang tersebut (tidak tergantung pada pilihan titik awal $\mathbf{x}_0$) jika vektor-vektor $\mathbf{a}_1$, $\mathbf{a}_2$, …, $\mathbf{a}_M$ merentang $\mathbb{R}^N$. Salah satu dari titik-titik $\mathbf{x}_M^{(p)}$ untuk p yang cukup besar akan diambil sebagai penyelesaian hampiran dari SPL.

Program SCILAB dari algoritme TRA disajikan pada Lampiran 1 dalam bentuk suatu fungsi. Nama fungsinya adalah ‘tra’. Fungsi dibuat di dalam *sci-file*. Masukannya adalah matriks $A = [\mathbf{a}_1 \quad \mathbf{a}_2 \quad \cdots \quad \mathbf{a}_M]^T$ yang berukuran $M \times N$, vektor kolom $\mathbf{b} = [b_1 \quad b_2 \quad \cdots \quad b_M]^T$

(disebut sebagai **ruas kanan** SPL) yang panjangnya M, dan vektor kolom $\mathbf{x}_0$ yang panjangnya N sebagai penyelesaian hampiran awal. Keluarannya adalah titik $\mathbf{x}_M^{(m)}$ (titik yang terletak pada hiperbidang ke-M/terakhir) yang disajikan dalam suatu vektor kolom yang panjangnya N serta bilangan m yang menyatakan banyaknya iterasi yang diperlukan. Penyelesaian hampiran ini didapatkan dengan menetapkan ukuran penghentian sebagai berikut:

$$\left\| A\mathbf{x}_M^{(m)} - \mathbf{b} \right\|_2 \leq 10^{-3}$$

dengan $\|\cdot\|_2$ menyatakan norm vektor Euclidean.

3. Implementasi dan Hasil Komputasi

Algoritme TRA ini akan diimplementasikan untuk menyelesaikan SPL yang dibangun langsung maupun tidak langsung (dengan memodifikasi) dari matriks A dan vektor kolom **b** yang dihasilkan dari fungsi 'phillips' yang terdapat pada Lampiran 2. Matriks A yang dihasilkan merupakan matriks takdefinit negatif dan simetris. Penyelesaian yang dihasilkan dari fungsi 'phillips' akan dijadikan sebagai penyelesaian hampiran awal.

Fungsi 'phillips' merupakan fungsi yang membangkitkan matriks A, vektor kolom **b**, serta vektor kolom **x** dari suatu SPL. Ketiganya dihasilkan dengan mendiskretisasi persamaan integral Fredholm jenis pertama [(Hansen, 2007)]

$$\int_{-6}^{6} \kappa(\tau,\sigma)x(\sigma)d\sigma = b(\tau), \qquad -6 \leq \tau \leq 6,$$

dengan Metode Galerkin. Penyelesaian x, kernel κ, dan ruas kanan b diberikan oleh

$$x(\sigma) = \begin{cases} 1 + \cos\left(\frac{\pi}{3}\sigma\right), & |\sigma| < 3 \\ 0, & |\sigma| \geq 3 \end{cases}$$

$$\kappa(\tau,\sigma) = x(\tau - \sigma)$$

$$b(\tau) = (6 - |\tau|)\left(1 + \frac{1}{2}\cos\left(\frac{\pi}{3}\tau\right)\right) + \frac{9}{2\pi}\sin\left(\frac{\pi}{3}|\tau|\right).$$

Hal ini ditemukan oleh D. L. Phillips dan dapat dilihat pada [(Calvetti dan Reichel, 2002),(Hansen,2007)]. Masukan dari fungsi 'phillips' ini adalah bilangan bulat positif n dengan n harus kelipatan 4.

Banyaknya SPL yang akan diselesaikan ada 14 SPL. Matriks A dan vektor kolom $\mathbf{b}$ dari SPL ke-1 sampai SPL ke-10 murni dihasilkan dari fungsi 'phillips'. SPL ke-k dari sepuluh SPL pertama tersebut terdiri atas $4k$ persamaan dan $4k$ faktor yang tidak diketahui. Sebagai contoh, berikut merupakan SPL ke-1 beserta penyelesaiannya yang dibangkitkan dari fungsi 'phillips'.

$$4.2159x_1 + 0.8921x_2 = 0.4922$$

$$0.8921x_1 + 4.2159x_2 + 0.8921x_3 = 9.9001$$

$$0.8921x_2 + 4.2159x_3 + 0.8921x_4 = 9.9001$$

$$0.8921x_3 + 4.2159x_4 = 0.4922$$

$$\mathbf{x}_{phillips} = (0 \quad 1.7321 \quad 1.7321 \quad 0)^T.$$

Peyelesaian hampiran yang dihasilkan dari fungsi 'tra' ($\mathbf{x}_{tra}$) selanjutnya akan dibandingkan dengan penyelesaian yang dihasilkan dari fungsi 'phillips' ($\mathbf{x}_{phillips}$). Kedua penyelesaian dari sepuluh SPL pertama ini terdapat pada Lampiran 3. Banyaknya iterasi, *running time*, dan tingkat kesalahan dari penyelesaian sepuluh SPL pertama terdapat pada Tabel 1. Hal ini didasarkan pada ukuran penghentian proses algoritme yang telah dibahas sebelumnya.

Tabel 1. Banyaknya iterasi, *running time*, dan tingkat kesalahan untuk penyelesaian SPL ke-1 sampai SPL ke-10

SPL ke-	Banyaknya iterasi	*Running time* (detik)	$\left\|A\mathbf{x}_{phillips} - b\right\|_2$	$\|A\mathbf{x}_{tra} - b\|_2$
1	8	0.013950	2.1059	5.7613×10^{-4}
2	116	0.248873	0.8337	9.9849×10^{-4}
3	1 127	3.351584	0.3979	9.9962×10^{-4}
4	385	1.395177	0.2298	1.0000×10^{-3}
5	139	0.630695	0.1489	9.9956×10^{-4}
6	65	0.561271	0.1041	9.9243×10^{-4}
7	47	0.370987	0.0768	9.8985×10^{-4}
8	37	0.370060	0.0590	9.8878×10^{-4}
9	30	0.259787	0.0467	9.7668×10^{-4}
10	22	0.302518	0.0379	9.9003×10^{-4}

Banyaknya iterasi dan *running time* yang diperlukan untuk mendapatkan penyelesaian hampiran pada awalnya meningkat dari SPL ke-1 ke SPL ke-3, kemudian menurun dari SPL ke-3 ke SPL ke-10. Hal ini dapat dikarenakan penyelesaian hampiran awal yang diambil dari fungsi 'phillips'. Penyelesaian hampiran SPL yang dihasilkan dari fungsi 'tra' lebih akurat daripada penyelesaian yang dihasilkan dari fungsi 'phillips'. Hal ini dapat dilihat dari tingkat

kesalahannya. Akan tetapi, dibutuhkan waktu lebih untuk menghasilkan penyelesaian hampiran SPL yang dihasilkan dari fungsi ‘tra’ ini. Sebagai contoh, berikut merupakan penyelesaian hampiran dari SPL ke-1 yang dihasilkan dari fungsi ‘tra’.

$\mathbf{x}_{tra} = (-0.3048 \quad 1.9915 \quad 1.9915 \quad -0.3048)^T$.

Penyelesaian hampiran ini terlihat sangat berbeda dengan penyelesaian yang dihasilkan dari fungsi ‘phillips’. Walaupun begitu, dapat dilihat bahwa penyelesaian yang dihasilkan dari fungsi ‘phillips’ semakin akurat dengan bertambah besarnya ukuran SPL.

Matriks A dari SPL ke-11 sampai SPL ke-14 murni dihasilkan dari fungsi ‘phillips’, sedangkan ruas kanannya dimodifikasi dari vektor kolom $\mathbf{b}$ yang dihasilkan dari fungsi ‘phillips’. Ruas kanan yang digunakan adalah $\hat{\mathbf{b}} = \mathbf{b} + \mathbf{e}$, dengan $\mathbf{e}$ (disebut juga sebagai ‘gangguan’) merupakan vektor kolom berukuran $M \times 1$, elemen-elemennya merupakan bilangan-bilangan acak dari sebaran normal baku, dan $\|\mathbf{e}\|_2 \cong 10^{-3} \times \|\mathbf{b}\|_2$. Tujuan dari modifikasi ini adalah ingin dilihat seberapa signifikan perubahan penyelesaian hampiran apabila diberikan ‘gangguan’.

SPL ke-$(10 + k)$ terdiri atas $4k$ persamaan dan $4k$ faktor yang tidak diketahui, sehingga SPL ke-$(10 + k)$ ini merupakan modifikasi dari SPL ke-k. Sebagai contoh, SPL ke-11 merupakan modifikasi dari SPL ke-1. ‘Gangguan’ yang diberikan yaitu vektor kolom

$$\mathbf{e} = (0.0140 \quad -0.0012 \quad 0.0138 \quad -0.0040)^T,$$

yang dibangkitkan secara acak dengan bantuan SCILAB, sehingga SPL ke-11 adalah sebagai berikut:

$4.2159x_1 + 0.8921x_2 = 0.5062$

$0.8921x_1 + 4.2159x_2 + 0.8921x_3 = 9.8989$

$0.8921x_2 + 4.2159x_3 + 0.8921x_4 = 9.9140$

$0.8921x_3 + 4.2159x_4 = 0.4882.$

Peyelesaian hampiran yang dihasilkan dari fungsi ‘tra’ ($\mathbf{x}_{tra}$) untuk empat SPL terakhir ini terdapat pada Lampiran 4. Banyaknya iterasi, *running time*, dan tingkat kesalahan dari penyelesaian empat SPL terakhir tersebut terdapat pada Tabel 2. Hal ini juga didasarkan pada ukuran penghentian proses algoritme yang telah dibahas sebelumnya.

Tabel 2. Banyaknya iterasi, *running time*, dan tingkat kesalahan untuk penyelesaian SPL ke-11 sampai SPL ke-14

SPL ke-	Banyaknya iterasi	*Running time* (detik)	$\|A\mathbf{x}_{tra} - b\|_2$
11	8	0.014091	0.0197
12	481	0.780851	0.0309
13	7 481	19.620956	0.0211
14	469 743	1531.209505	0.0306

Banyaknya iterasi dan *running time* yang diperlukan untuk mendapatkan penyelesaian hampiran SPL ke-12 sampai SPL ke-14 (setelah diberikan 'gangguan') jauh lebih besar dibandingkan SPL ke-2 sampai SPL ke-4 (sebelum diberikan 'gangguan'). Hal ini dapat dikarenakan penyelesaian hampiran awal yang diambil dari fungsi 'phillips' menjadi tidak efisien lagi. Sebagai contoh, berikut merupakan penyelesaian hampiran dari SPL ke-11 yang dihasilkan dari fungsi 'tra'.

$\mathbf{x}_{tra} = (-0.3010 \quad 1.9896 \quad 1.9954 \quad -0.3064)^T$.

Penyelesaian hampiran ini tidak terlalu berbeda dengan penyelesaian hampiran dari SPL ke-1 yang dihasilkan dari fungsi 'tra'.

4. Kesimpulan

Penyelesaian hampiran atas SPL yang dihasilkan dengan menggunakan TRA mempunyai tingkat kesalahan yang sangat kecil. Walaupun begitu, khusus untuk SPL-SPL tertentu, penyelesaiannya membutuhkan iterasi yang banyak dan waktu yang lama. Proses mendapatkan penyelesaian hampiran atas SPL juga dipengaruhi oleh penentuan penyelesaian hampiran awalnya.

Ucapan Terima Kasih

Pekerjaan in didanai oleh Kementrian Pendidkan dan Kebudayaan, Republik Indonesia, melalui "Penelitian Strategis Unggulan", hibah DIPA-IPB, 0558/023-04.2.01/12/2012, dengan kontrak **no : 44/I3.24.4/SPK-PUS/IPB/2012.**

Daftar Pustaka

[1] Anton, H. dan C. Rorres. 2005. *Aljabar Linear Elementer* Edisi Kedelapan Jilid 2. Jakarta: Erlangga.

[2] Calvetti, D. dan L. Reichel. 2002. *Tikhonov Regularization of Large Linear Problems*. BIT: 43(2): 15-16.

[3] Censor, Y., 1983, *Finite series-expansion reconstruction methods*, Proceedings of the IEEE, 71(3), 409-419.

[4] Hansen, P.C., 2007, *Regularization tools version 4.0 for Matlab 7.3*, Numerical Algorithms, V 46 (2), 189-194, 1017-1398

Lampiran

Lampiran 1. Program SCILAB untuk algoritme TRA (untuk menyelesaikan sistem persamaan linear)

Fungsi tra

```
function [x,m] = tra(A,b,x0)
%-------------------------------------------------------------------------------
% Fungsi tra menghasilkan penyelesaian hampiran dari sistem linear Ax=b :
% a11*x1 + a12*x2 + ... + a1N*xN = b1
% a21*x1 + a22*x2 + ... + a2N*xN = b2
%                  :
% aM1*x1 + aM2*x2 + ... + aMN*xN = b3
% menggunakan Teknik Rekonstruksi Aljabar.
% Penyelesaian hampiran x yang dipilih adalah x yang terletak pada
% hiperbidang ke-M.
%-------------------------------------------------------------------------------
% Masukan:
%   A  = matriks berukuran MxN
%        a11 a12 ... a1N
%        a21 a22 ... a2N
%                :
%        aM1 aM2 ... aMN
%   b  = vektor kolom berukuran Mx1
%        b1
%        b2
%        :
%        bM
%   x0 = vektor kolom berukuran Nx1 (penyelesaian hampiran awal)
%
% Keluaran:
%   x  = vektor kolom berukuran Nx1
%        x1
%        x2
%        :
%        xN
%   m  = banyaknya iterasi
%-------------------------------------------------------------------------------
tic
M = size(A,1);
m = 0;
p = norm(A*x0-b);
if p <= 0.001
    x = x0;
else
    while p > 0.001
        m = m+1;
        for k = 1:M
            x = x0+(b(k)-A(k,:)*x0)*A(k,:)'/(A(k,:)*A(k,:)');
            x0 = x;
        end
        p = norm(A*x-b);
    end
end
toc
end
```

Lampiran 2. Program SCILAB untuk membangkitkan matriks *A*, vektor kolom **b**, dan vektor kolom **x** dari suatu SPL

Fungsi phillips

```
function [A,b,x] = phillips(n)
% PHILLIPS Test problem: Phillips "famous" problem.
%
% [A,b,x] = phillips(n)
%
% Discretization of the 'famous' first-kind Fredholm integral
% equation deviced by D. L. Phillips.  Define the function
%    phi(x) = | 1 + cos(x*pi/3) ,  |x| <  3 .
%             | 0               ,  |x| >= 3
% Then the kernel K, the solution f, and the right-hand side
% g are given by:
%    K(s,t) = phi(s-t) ,
%    f(t)   = phi(t) ,
%    g(s)   = (6-|s|)*(1+.5*cos(s*pi/3)) + 9/(2*pi)*sin(|s|*pi/3) .
% Both integration intervals are [-6,6].
%
% The order n must be a multiple of 4.

% Reference: D. L. Phillips, "A technique for the numerical solution
% of certain integral equations of the first kind", J. ACM 9
% (1962), 84-97.

% Discretized by Galerkin method with orthonormal box functions.

% Per Christian Hansen, IMM, 09/17/92.

% Check input.
if (rem(n,4)~=0), error('The order n must be a multiple of 4'), end

% Compute the matrix A.
h = 12/n; n4 = n/4; r1 = zeros(1,n);
c = cos((-1:n4)*4*pi/n);
r1(1:n4) = h + 9/(h*pi^2)*(2*c(2:n4+1) - c(1:n4) - c(3:n4+2));
r1(n4+1) = h/2 + 9/(h*pi^2)*(cos(4*pi/n)-1);
A = toeplitz(r1);

% Compute the right-hand side b.
if (nargout>1),
  b = zeros(n,1); c = pi/3;
  for i=n/2+1:n
    t1 = -6 + i*h; t2 = t1 - h;
    b(i) =   t1*(6-abs(t1)/2) ...
           + ((3-abs(t1)/2)*sin(c*t1) - 2/c*(cos(c*t1) - 1))/c ...
           - t2*(6-abs(t2)/2) ...
           - ((3-abs(t2)/2)*sin(c*t2) - 2/c*(cos(c*t2) - 1))/c;
    b(n-i+1) = b(i);
  end
  b = b/sqrt(h);
end

% Compute the solution x.
if (nargout==3)
  x = zeros(n,1);
  x(2*n4+1:3*n4) = (h + diff(sin((0:h:(3+10*eps))'*c))/c)/sqrt(h);
  x(n4+1:2*n4) = x(3*n4:-1:2*n4+1);
end
```

METODE *CONJUGATE GRADIENT* PARALEL UNTUK MENYELESAIKAN SISTEM PERSAMAAN LINEAR DALAM SCILAB

F. AYATULLAH[1], M.T. JULIANTO[1], A.D. GARNADI[1], S.NURDIATI[1]

[1] *Departemen Matematika,*
Institut Pertanian Bogor, Bogor, 16680

ABSTRAK. Komputasi paralel merupakan salah satu alternatif untuk meningkatkan kinerja komputasi. Komputasi paralel bertujuan menyelesaikan masalah komputasi yang besar dan mempercepat waktu eksekusinya. Komputasi paralel yang dilakukan dalam percobaan menggunakan beberapa komputer dalam satu jaringan. *Software* yang digunakan dalam bercobaan adalah SCILAB dan *Parallel Virtual Machine* (PVM). Masalah komputasi yang akan diselesaikan adalah penyelesaian sistem persamaan linear dengan menggunakan metode *Conjugate Gradient* (CG). Algoritma paralel dari metode *Conjugate Gradient* dibuat agar metode ini dapat diterapkan secara paralel. Waktu eksekusi metode *Conjugate Gradient* baik secara sekunsial maupuan paralel untuk menyelesaikan sistem persamaan linear yang sama dalam percobaan diamati. Percobaan dilakukan terhadap 3 buah sistem persamaan linear dengan matriks koefisien *A* yang berbeda. Percobaan metode *Conjugate Gradient* paralel untuk sistem persamaan linear dengan matriks nos3 dan matriks ex13 berhasil mencapai *speedup* yang dicapai pada percobaan paralel untuk matriks ex13_30_30 sangat kecil, artinya waktu eksukisanya cenderung sama atau lebih lambat daripada waktu eksekusinya. *Speedup* yang dicapai pada setiap percobaan paralel selalu bertambah seiring bertambahnya jumlah komputer yang digunakan. *Speedup* yang dicapai pada setiap percobaan metode *Conjugate Gradient* paralel untuk nilai toleransi 10^{-10} lebih besar dibandingkan pada percobaan untuk toleransi 10^{-5}.

ABSTRACT Parallel computing is an alternative way to increase computation performance. The objective of parallel computing is to solve very large computation problem and speed up the execution time. The experiments for parallel computing use computer networks, SCILAB software and Parallel Virtual Machine software running in LINUX machine. The computation problems for the experiment are solving systemof linear equationsusing Conjugate Gradient method. A parallel algorithm for Conjugate Gradient method is made for parallel computing experiment. Conjugate Gradient method is used to solve three systems of linear equations with three different coefficient matrices. The execution time for sequential and parallel Conjugate Gradient method was observed. The parallel experiment for nos3 and ex13 matrices gained large speedup, it means that the parallelexecution time was much faster tah the sequential execution time. The parallel experiment for matrix gr_30_30 gained very small speedup, which mean that the parallel execution time wasn't faster than the sequential execution time. The parallel experiment using more computer gained larger speedup than using less computer. The parallel experiment for tolerance value 10^{-10} gained larger speedup than for tolerance value 10^{-5}.

PENDAHULUAN

Latar Belakang

Saat ini komputer banyak digunakan untuk membantu menyelesaikan permasalahan manusia. Penggunaan komputer menjadi lebih mudah dan cepat. Namun kinerja komputer saat ini masih belum mampu untuk memenuhi semua tuntunan penggunanya. Beberapa masalah komputasi yang sangat besar masih belum dapat diselesaikan atau waktu eksekusinya masih lambat.

Salah satu solusi untuk meningkatkan kinerja komputasi adalah dengan melakukan komputasi paralel. Komputasi paralel membuat masalah komputasi yang sangat besar dibagi menjadi masalah-masalah yang lebih kecil dan diselesaikan oleh banyak prosesor secara bersamaan.

Komputasi paralel dapat dilakukan dengan menggunakan banyak komputer dalam satu jaringan. Komputer multiprosesor saat ini masih jarang dan sangat mahal, sehingga komutasi paralel lebih memungkinkan untuk dilakukan pada suatu jaringan komputer.

Komputasi paralel dengan menggunakan suatu jaringan komputer membutuhkan bantuan *software* untuk komunikasi antar prosesor. Salah satu *software* yang dapat digunakan adalah Parallel Virtual Machine (PVM). PVM dapat membuat suatu jaringan komputer seakan-akan menjadi sebuah komputer multiprosesor

Dahulu komputasi paralel dengan PVM hanya dapat dilakukan dengan menggunakan program dalam bahasa C atau FORTRAN. Namun saat ini komputasi paralel dengan PVM dapat dilakukan dengan lebih mudah dengan menggunakan SCILAB. SCILAB adalah sebuah *software* matematika dan sains yang sejenis dengan MATLAB. SCILAB dapat diperoleh dan digunakan secara gratis, sedangkan MATLAB merupakan *software* komersial dan berlisensi.

Komputasi paralel banyak diterapkan pada masalah komputasi yang berkaitan dengan matriks berukuran besar. Salah satu masalah komputasi yang berkaitan dengan matriks adalah penyelesaian sistem persamaan linear secara numerik.

Masalah komputasi yang dibahas dalam tulisan ini adalah penyelesaian sistem persamaan linear berukuran besar secara numerik dengan menggunakan SCILAB dan PVM. Tulisan ini, didahului dengan sejumlah pengertian dasar dan istilah, kemudian diikuti dengan deskripsi metode *Conjugate Gradient* dan dilanjutkan dengan uji numerik metode atas sejumlah kasus. Pada bagian lampiran, disertakan script SCILAB yang digunakan.

PENGERTIAN DASAR DAN ISTILAH

Komputasi paralel Komputasi paralel adalah melakukan suatu proses komputasi dengan menggunakan banyak prosesor secara bersamaan. Komputer yang digunakan untuk komputasi paralel disebut Komputer paralel. Komputer paralel dapat berupa sebuah komputer multiprosesor atau beberapa Komputer dalam satu jaringan.

Berdasarkan sistem komunikasi antar prosesor yang digunakan, komputer paralel terbagi menjadi dua, yaitu: [Grama *et al*, 2003]

1. *Shared-Address-Space*.

Setiap prosesor dalam Komputer paralel dengan sistem *Shared-Address-Space* dapat mengakses sebuah ruang dalam *memory*. Seluruh data yang digunakan dalam proses komputasi disimpan di ruang tersebut.

2. *Message-Passing*.

Komunikasi antar prosesor dalam Komputer paralel dengan sistem *Message-Passing* dilakukan dengan saling mengirim pesan. Komputer paralel ini tidak memiliki ruang *memory* yang dapat diakses oleh semua prosesor, sehingga seluruh data yang digunakan dalam proses komputasi harus dikirim dari satu prosesor lainnya.

Algoritma Paralel[Grama *et al*, 2003] Algoritma paralel adalah langkah-langkah untuk menyelesaikan suatu masalah yang dapat dijalankan pada beberapa prosesor secara bersama-sama.

Paralelisasi Algoritma Sekuensial [Grama *et al*, 2003] Algoritma paralel dapat berupa Algoritma yang berdiri sendiri atau merupakan paralelisasi dari suatu algoritma sekuensial. Paralelisasi suatu algoritma sekuensial dapat dilakukan dengan menjalankansebagian atau semua langkah-langkah di bawah ini:

1. Melakukan identifikasi terhadap bagian-bagian Algoritma yang dapat dijalankan secara bersamaan.
2. Memetakan bagian-bagian yang akan dijalankan secara bersamaan kepada proses-proses yang paralel.
3. Mendistribusikan input, output, dan data lain yang berhubungan dengan program.
4. Mengatur akses setiap prosesor kepada data dalam *Shared-Address-Space*.
5. Melakukan sinkronisasi terhadap setiap prosesor pada beberapa tahapan pada saat eksekusi program.

Speedup adalah tingkat keuntungan yang diperoleh dengan melakukan komputasi paralel. *Speedup* merupakan perbandingan antara waktu eksekusi algoritma sekuensial dengan waktu eksekusi algoritma paralelnya. [Grama *et al*, 2003] Kita tuliskan:

$$S_p = \frac{TS}{TP},$$

Dengan P menyatakan jumlah prosesor yang digunakan, S_P merupakan *Speedup* untuk *p* buah proses, T_S menyatakan waktu eksekusi algoritma sekuensial, dan T_P menunjukkan waktu eksekusi algoritma paralel pada *p* buah prosesor.

Parallel Virtual Machine [Geist *et al*, 1994]

Paralel Virtual Machine (PVM) adalah *software* yang memungkinan berbagai jenis komputer dalam satu jaringan melakukan proses komputasi paralel. PVM membuat beberapa komputer dalam satu jaringan seakan-akan menjadi sebuah komputer mutiprosesor.

Software ini dikembangkan oleh *Oak Ridge National Laboratory* dan *University of Tennssee*. *Software* ini telah banyak digunakan di bidang sains dan aplikasi komputasi di bidang lainnya di seluruh dunia.

Parallel Virtual Machine didistribusikan secara gratis dan dapat diperoleh di www.netlib.org.

Sistem Persamaan Linear

Sistem persamaan linear adalah kumpulan *m* persamaan dengan *n* variabel dalam bentuk

$$a_{11}x_1+a_{12}x_2+\ldots+a_{1n}x_n=b_1,$$
$$a_{21}x_1+a_{22}x_2+\ldots+a_{2n}ax_n=b_2,$$
$$\vdots$$
$$a_{m1}x_1+a_{m2}x_2+\ldots+a_{mn}x_n=b_m.$$

Dengan x_j merupakan variable ke-*j*, sedangkan a_{ij} dan b_i adalah konstanta untuk koefisien matriks ke-ij dan vektor ruas kanan, dengan indeks $1 \le i \le m$, $1 \le j \le n$.

Sistem persamaan linear di atas dapat ditulis dalam bentuk

$$Ax = b .$$

Matriks *A* berukuran *m* x *n* dan disebut dengan matriks koefisien dari SPL di atas, vektor *b* berukuran *m* dan disebut dengan konstanta, sedangkan vektor x berukuran *n* dan disebut dengan vektor solusi. Selanjutnya, sebuah matriks *A* disebut simetrik jika dipenuhi sifat $A^T = A$, yaitu $a_{ij} = a_{ji}$. Kita katakan juga bahwa sebuah matriks *A* disebut definit positif jika untuk semua vektor $x \neq 0$, berlaku $\mathbf{x}^T A x > 0$. Dua vektor ***x*** dan ***y*** disebut *conjugate* jika $x^T A\mathbf{y} = 0$, $x \neq \mathbf{y}$. Misalkan **x** dan **y** merupakan vektor berukuran *m*, *inner product* dari vektor ***x*** dan ***y*** adalah $\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=0}^{m} x_i y_i$. Perhatikan bahwa karena vektor juga merupakan sebuah matriks dengan satu kolom. Oleh karena itu, inner product di atas dapat ditulis menjadi $x^T \mathbf{y} = x_1 y_1 + x_2 y_2 + \cdots + x_m y_m$.

Perkalian Matriks-Vektor. Misalkan *A* adalah sebuah matriks berukuran *m* x *n* dan A_i adalah baris ke-i dari matriks A serta v adalah sebuah vektor berukuran n, perkalian antara A dan v adalah $A\mathbf{v} = (A_1\mathbf{v}, A_2v, \ldots, A_mv)^T$, dengan $A_i\mathbf{v}$ adalah *inner product* antara baris ke-i dari matriks A dengan vektor **v**.

Norm. Norm dari suatu vektor **x** berukuran m adalah $\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \cdots + xm^2}$.

ALGORITMA CONJUGATE GRADIENT.

Metode Conjugate Gradient

Metode *Conjugate Gradient* adalah salah satu metode iteratif untuk menyelesaikan sistem persamaan linear. Metode ini efektif untuk sistem persamaan linear dengan matriks koefisien yang simetrik definit positif. Secara umum, motode ini membangkitkan vektor-vektor yang *conjugate* dan juga merupakan *gradient* dari fungsi kuadrat. Metode ini menyelesaikan sistem persamaan linear dengan cara menemukan titik minmum dari fungsi kuadrat. [Barret et al, 1994]

Metode *conjugate gradient* (CG) adalah sebuah metode iteratif untuk menyelesaikan sistem persamaan linear (SPL) berbentuk

$$A\mathbf{x} = \mathbf{b} .$$

dengan matriks koefisien A bersifat simetrik denitif positif. Metode ini banyak digunakan untuk menyelesaikan SPL berukuran besar.

Ide Metode CG untuk menyelesaikan suatu SPL adalah dengan mencari titik minimum dari suatu fungsi kuadrat dari suatu vektor yang berbentuk

$$f(x)=\frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x} + c .$$

Gradient dari fungsi kuadrat di atas adalah

$$f'(x)=\frac{1}{2}A^T\mathbf{x} + \frac{1}{2}A\mathbf{x} - \mathbf{b} .$$

Jika A simetrik maka *gradient* dari fungsi kuadrat di atas adalah

$$f'(x) = A\mathbf{x} - \mathbf{b} .$$

Titik kritis dari fungsi kuadrat di atas merupakan solusi dari SPL (1).

$$f'(\mathbf{x}) = 0,$$
$$A\mathbf{x} - \mathbf{b} = 0,$$
$$A\mathbf{x} = \mathbf{b}.$$

Jika A definit positif, maka **x** merupakan titik minimum global dari fungsi kuadrat f(**x**), sehingga **x** merupakan solusi yang unik dari SPL (1).

Untuk menemukan titik minimum atau vektor solusi **x**, dilakukan langkah-langkah untuk menghitung nilai perkiraan vektor solusi **x** pada setiap langkah atau iterasi ke-i dilambangkan dengan $\mathbf{x}_{(i)}$. indeks untuk langkah atau iterasi dilambangkan dalam tanda kurang. Jika pada iterasi ke-i toleransi sudah terpenuhi, maka $x_{(i)}$ adalah solusi numerik dari SPL (1).

Pada setiap iterasi didefinisikan dua buah vektor sebagai berikut:

1. Vektor *error* $\mathbf{e}_{(i)}$, yaitu selisih antara vektor perkiraan pada iterasi ke-i dengan vektor solusi.

   $$\mathbf{e}_{(i)} = \mathbf{x}_{(i)} - \mathbf{x} . \qquad (2)$$

2. Vektor *residual* $r_{(i)}$, yaitu selisih yang masih terjadi pada SPL dengan menggunakan perkiraan solusi yang telah diperoleh pada iterasi ke-i.

   $$\mathbf{r}_{(i)} = \mathbf{b} - A\mathbf{x} . \qquad (3)$$

   Dari persamaan di atas diketahui bahwa vektor *residual* $\mathbf{r}_{(i)}$ juga merupakan negatif dari *gradient* $f'(\mathbf{x})$ pada titik $x_{(i)}$.

   $$\mathbf{r}_{(i)} = -f'(\mathbf{x}_{(i)}) .$$

   Langkah pertama pada metode CG adalah dengan memilih tebakan awal $x_{(0)}$.

   Selanjutnya vektor $\mathbf{x}_{(i)}$ diperoleh dengan persamaan

   $$\mathbf{x}_{(i+1)} = \mathbf{x}_{(i)} + a_{(i)}\,\mathbf{d}_{(i)} . \qquad (4)$$

   Vektor $\mathbf{d}_{(i)}$ adalah vektor arah (*search direction*) untuk mencari titik minimum x pada iterasi ke-i.

   Nilai vektor arah pada langkah pertama $\mathbf{d}_{(0)}$ ditetapkan sebagai $\mathbf{r}_{(0)}$, sedangkan pada langkah selanjutnya digunakan persamaan

   $$\mathbf{d}_{(i+1)} = \mathbf{r}_{(i+1)} - ß_{(i+1)}\,\mathbf{d}_{(i)} . \qquad (5)$$

   Konstanta $a_{(i)}$ dan $ß_{(i+1)}$ merupakan skalar dan diperoleh dengan persamaan-persamaan di bawah ini.

   $$a_{(i)} = \frac{rT(i)r(i)}{dT(i)Ad(i)} \qquad (6)$$

   dan

   $$ß_{(i+1)} = \frac{rT(i+1)r(i+1).}{rT(i)r(i)} \qquad (7)$$

   Pada persamaan (3) dan (6) masing-masing terdapat perkalian matriks-vektor. Untuk meningkatkan efisiensi komputasi pada persamaan (3) dan (6) , maka dilakukan langkah-langkah berikut ini:

   1. Subtitusikan persamaan (1) dan (2) pada persamaan (3) sehingga diperoleh persamaan

$\mathbf{r}_{(i)} = A\mathbf{e}_{(i)}$. (8)

2. Subtitusikan persamaan (2) pada persamaan (4) sehingga diperoleh persamaan
   $\mathbf{e}_{(i+1)} = \mathbf{e}_{(i)} + a_{(i)}\mathbf{d}_{(i)}$ (9)
3. Subtitusikan persamaan (9) pada persamaan (8) sehingga diperoleh persamaan
   $\mathbf{r}_{(i+1)} = \mathbf{r}_{(i)} - a_{(i)}A\mathbf{d}_{(i)}$. (10)
4. Perkalian matriks-vektor $A\mathbf{d}_{(i)}$ pada persamaan (6) dan (10) dapat dinyatakan sebagai **q**, sehingga $A\mathbf{d}_{(i)}$ hanya dihitung satu kali dengan menambahkan persamaan
   $\mathbf{q} = A\mathbf{d}_{(i)}$. (11)

Secara lengkap metode *Conjugate Gradient* diberikan pada Algoritma 1.

Paralelisasi Metode Conjugate Gradient

Langkah-langkah yang dilakukan dalam paralelisasi suatu algoritma sekuensial tidak baku. Paraelisasi dapat dilakukan dengan mempertimbangkan karakteristik *hardware* dan *software* yang akan digunakan. Percobaan yang dilakukan dalam tulisan ini menggunakan suatu jaringan komputer dengan sistem *Message-Passing*. Komputasi paralel dalam percobaan dilakukan dengan menggunakan *software* SCILAB dan PVM.

Diberikan SPL Ax = b
Pilih tebakan awal $x_{(0)}$
r(0)=b-Ax(0)
d(0)=r(0)
i=0
While i < *imaks* and $\frac{\|r(i)\|}{\|r(0)\|} \geq tol$
    q = Ad(i)
    $a(i) = \frac{rT(i)r(i)}{dT(i)q}$
    x(i+1) = x(i) + a(i)d(i)
    r(i+1) = r(i) - a(i)q
    β (i+1) = $\frac{rT(i+1)r(i+1)}{rT(i)r(i)}$
    d(i+1) = r(i+1) - β (i+1) d(i)
    i = i + 1

Algoritma 1. Metode *Conjugate Gradient*

Metode CG merupakan metode iteratif yang umumnya sulit untuk diparalelisasi. Hal ini disebabkan setiap langkah pada metode iterative membutuhkan hasil dari langkah sebelumnya. Berikut ini adalah langkah-langkah yang dilakukan dalam paralelisasi metode CG:

Melakukan identifikasi terhadap bagian-bagian algoritma yang dapat dijalankan secara bersamaan

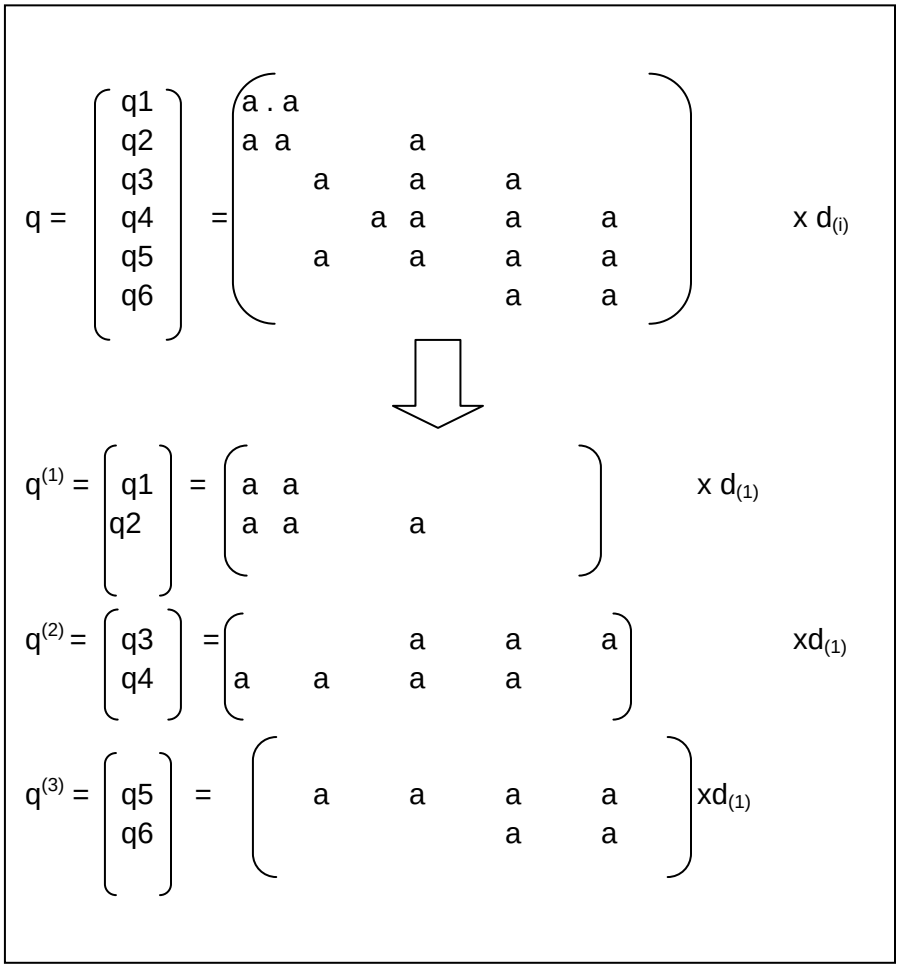
Metode CG memiliki dua tahapan, yaitu tahapan inisialisasi dan tahapan yang berulang-ulang. Paralelisasi dan tahapan yang berulang-ulang karena beban komputasi pada tahapan inisialisasi jauh lebih kecil jika dibandingkan pada tahapan yang berulang-ulang.

Pada tahapan yang berulang-ulang terlihat bahwa hanya penghitungan **x**(i+1) dan **r**(i+1) yang dapat dijalankan secara bersamaan. Namun beban komputasi pada perhitungan **x**(i+1) dan **r**(i+1) sangat kecil, sehingga tidak efisien jika dijalankan secara bersamaan.

Beban komputasi yang terbesar pada metode CG adalah perkalian matriks-vektor pada perhitungan vektor **q**, oleh karena itu, lebih efisien jika membagi perhitungan vektor **q** menjadi beberapa bagian untuk dijalankan secara bersamaan.

Misalkan vektor **q** dengan panjang *n* dan prosesor yang digunakan sebanyak *p* bagian. Setiap bagian adalah sebuah vektor berukuran (*n* / *p*) x *n* dengan vektor $\mathbf{d}_{(j)}$.
Misalkan $\mathbf{q}^{(k)}$ adalah bagian ke-k dari vektor **q** dan *a* adalah elemen tol dari matriks A.

Berikut ini adalah ilustrasi pembagian vektor **q**.



Gambar 1. Ilustrasi pembagian vector **q** berukuran 6.

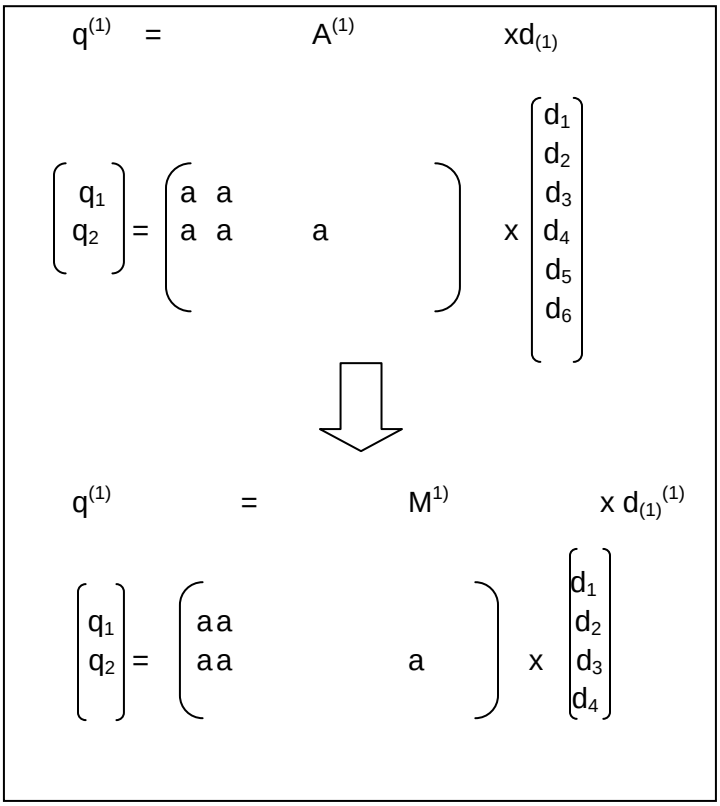
Setelah dilakukan pembagian seperti yang diilistrasikan pada Gambar 1, masih dapat dilakukan efisiensi pada setiap bagian $\mathbf{q}^{(k)}$. Efisiensi dilakukan dengan menghilangkan kolom-kolom yang bernilai nol pada setiap partisi matriks A. Kolom-kolom yang dihilangkan adalah kolom-kolom yang terletak sebelum kolom tak nol

terakhir. Kemudian vektor $\mathbf{d}^{(i)}$ pada setiap bagian disesuaikan dengan partisi matriks A pada bagian tersebut.

Misalkan partisi matriks A pada bagian ke-k dilambangkan dengan $A^{(k)}$ yang telah dihilangkan kolom-kolom nol-nya dilambangkan dengan $M^{(k)}$ dilambangkan dengan $\mathbf{d}_{(i)}{}^{(k)}$. Berikut ini adalah ilustrasi penghilangan kolom-kolom partisi matriks A pada $\mathbf{q}^{(1)}$yang terdapat dalam Gambar 1.

Jadi, bagian-bagian dari metode CG yang akan dijalankan secara bersamaan adalah bagian-bagian pada perhitungan vektor **q** pada setiap iterasi. Jumlah dari bagian-bagian ini adalah sesuai dengan jumlah prosesor yang akan digunakan

$$\mathbf{q}^{(k)} = M^{(k)}\, \mathbf{d}_{(i)}{}^{(k)}, \qquad 1 \le k \le p\ .$$



Gambar 2. Ilustrasi penghilangan kolom-kolom yang bernilai nol pada partisi matriks A.

Memetakan bagian-bagian yang akan dijalankan secara bersamaan kepada proses-proses yang paralel.

Proses yang mengontrol seluruh langkah pada metode CG disebut dengan master. Di lain pihak, setiap proses yang hanya menjalankan satu bagian yang dijalankan secara bersamaan disebut dengan *slave*. Setiap *slave* dijalankan pada satu prosesor.

Pada langkah pertama telah diputuskan bahwa bagian-bagian yang dijalankan secara bersamaan adalah p buah perhitungan $\mathbf{q}^{(k)}$. jadi, setiap bagian

$$\mathbf{q}^{(k)} = M^{(k)}\, \mathbf{d}_{(i)}{}^{(k)}$$

dikerjakan oleh proses *slave* pada prosesor ke-k

1. Mendistribusikan input, output, dan data lain yang berhubungan dengan program.

Setiap proses slave pada prosesor ke-k membutuhkan input $M^{(k)}$ dan $\mathbf{d}_{(i)}^{(k)}$. Input $M^{(k)}$ dikirim oleh master hanya pada iterasi pertama, sedangkan $\mathbf{d}_{(i)}^{(k)}$ dikirim pada setiap iterasi. Output yang dihasilkan oleh setiap slave ($\mathbf{q}^{(k)}$) pada setiap iterasi dikirim kepada master.

Seluruh data yang akan didistibusikan harus memiliki tanda atau identitas. Hal ini dilakukan agar master dapat mengirim input yang tepat untuk setiap slave dapat disatukan kembali oleh master dengan benar.

2. Mengatur akses setiap prosesor kepada data dalam Shared-Address-Space.

Langkah ini tidak dilakukan karena sistem yang digunakan pada percobaan adalah sistem Message-Passing.

3. Melakukan sinkronasi terhadap setiap prosesor pada beberapa tahapan pada saat eksekusi program.

Pada setiap iterasi, proses *slave* pada prosesor ke-k melakukan perhitungan $\mathbf{q}^{(k)}$. Hasil dari setiap proses *slave* yang dikirim kepada proses master. Kemudian proses master harus menyatukan kembali setiap bagian $\mathbf{q}^{(k)}$ menjadi vektor $\mathbf{q}$ yang utuh, yaitu

$\mathbf{q} = (q^{(1)}, q^{(2)}, \ldots, q^{(p)})^T$

Secara lengkap algoritma paralel metode *Conjugate Gradient* diberikan di Algoritma 2.

Diberikan SPL $Ax = b$
Pilih tebakan awal $x_{(0)}$
$r_{(0)} = b - Ax_{(0)}$
$d_{(0)} = r_{(0)}$
$i = 0$
Menentukan $M_{(k)}$ untuk setiap k,
$1 \le k \le p$
While I < *imaks* and and $\frac{\|r(i)\|}{\|r(0)\|} \ge tol$
Menentukan $d_{(i)}^{(k)}$ untuk setiap k,
$1 \le k \le p$

Pada setiap prosesor ke-k
$1 \le k \le p$
$q^{(k)} = M^{(k)} d_{(i)}^{(k)}$

$\mathbf{q} = (q^{(1)}, q^{(2)}, \ldots, q^{(p)})^T$

$a_{(i)} = \frac{rT(i)\, r(i)}{dT(i)q}$
$x_{(i+1)} = x_{(i)} + a_{(i)}\, d_{(i)}$
$r_{(i+1)}\, r_{(i)} - a_{(l)}\, q$
$\beta_{(i+1)} = \frac{rT(i+1)r(i+1)}{rT(i)r(i)}$
$d_{(i+1)} = r_{(i+1)} - \beta_{(i+1)}\, d_{(i)}$
$i = i + 1$

Algoritma 2. Metode CG yang telah diparalelisasi

Hasil Percobaan Komputasi

Seluruh percobaan komputasi dilakukan di laboratorium computer Departemen Matematika, Fakultas Matematika dan Ilmu Pengetahuan Alam, Institut Pertanian Bogor. Semua computer yang digunakan adalah computer satu prosesor yang saling terhubunga dalam satu jaringan. Prosesor yang digunakan pada setiap komputer adalah Intel Pentium 4 3.00 GHz dengan RAM 512 MB. Percobaan dilakukan dalam sistem operasi FEDORA CORE 5. Seluruh komputasi menggunakan *software* SCILAB 4.0, sedangkan komputasi paralel dilakukan dengan PVM 3.4.5.

Percobaan komputasi dilakukan dengan menggunakan matriks-matriks di bawah ini sebagai matriks A. seluruh matriks diperoleh dari koleksi matriks diperoleh dari lokeksi *University of Florida* (Davis 2006).

Nama	Ukuran	Elemen tak nol
gr_30_30	900x900	7744
nos3	960x960	15844
ex13	2568x2568	75628

Tabel 1. Matriks-matriks yang digunakan dalam percobaan.

Berikut ini hasil yang diperoleh dari percobaan yang telah dilakukan:

1. Hasil percobaan komputasi dengan menggunakan matriks gr_30_30.

   Matriks gr_30_30 merupakan koleksi dari Harwell-Boeing (HB). Elemen-elemen matriks ini adalah bilangan real. Matriks ini bersifat simetrik denifit positif.

   Vektor konstanta **b** yang digunakan dalam percobaan ini adalah hasil dari perkalian matriks gr_30_30 dengan vektor yang semua elemennya bernilai satu dan berukuran 900,

   b = gr_30_30 **. 1;** **1** = $(1,1,\ldots,1)^T$.

   Tebakan awal $x_{(0)}$ yang digunakan adalah vektor nol berukuran 900, $x_{(0)} = (0,0,\ldots,0)^T$.

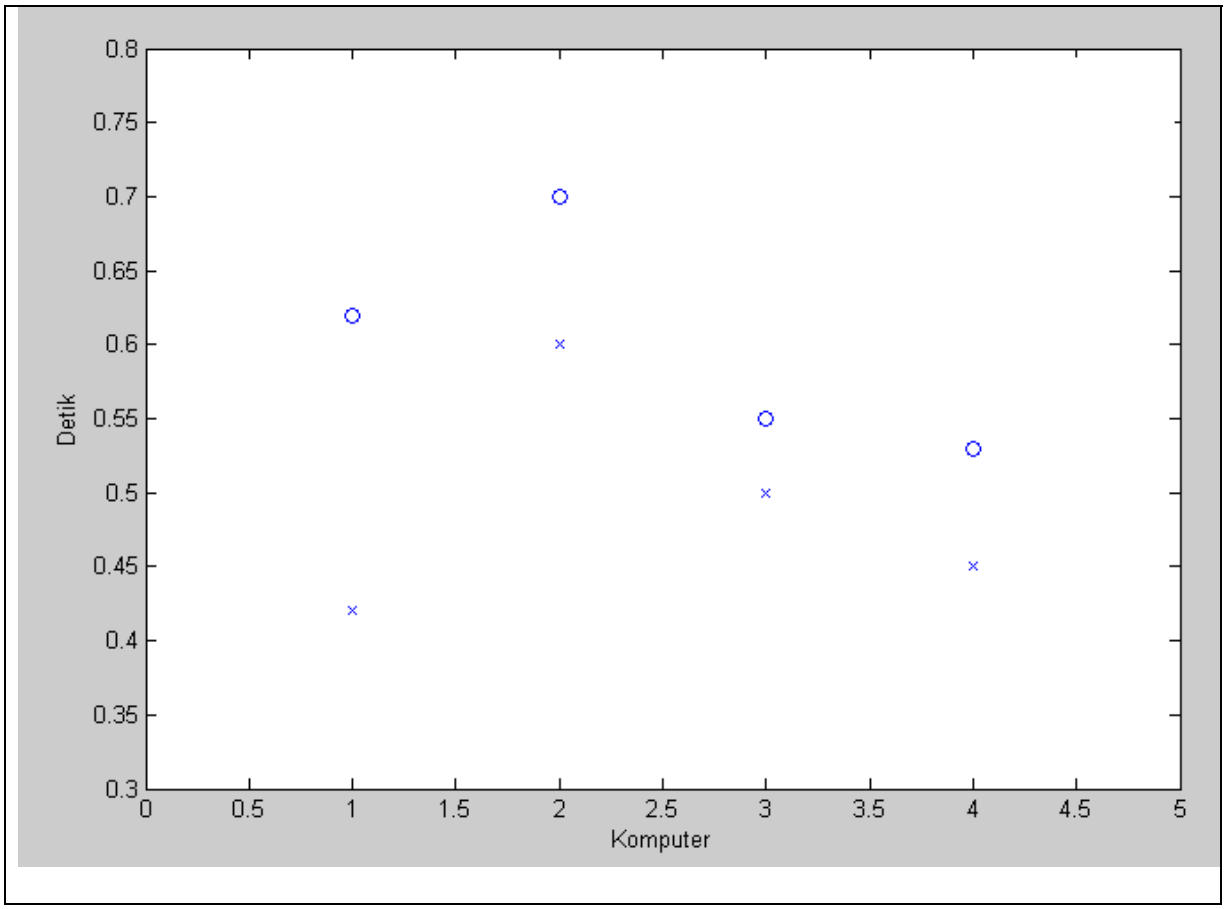
   Percobaan dilakukan untuk dua buah nilai toleransi *tol*, yaitu 10^{-5} dan 10^{-10}. Di sisi lain, iterasi maksimal *imaks* yang digunakan sama dengan ukuran vektor solusi ingin dicapai, yaitu 900.

   Percobaan metode CG secara parelel menggunakan 2, 3, dan 4 komputer. Proses master pada percobaan metode CG secara paralel dilakukan pada komputer yang digunakan untuk percobaan metode CG secara sekuensial. Pada tabel 2, diberikan hasil waktu eksekusi metode CG matriks gr_30_30.

Toleransi	Iterasi	Jumlah Komputer			
		1	2	3	4
10^{-5}	33	0.42	0.6	0.5	0.45
10^{-10}	46	0.62	0.7	0.55	0.53

Tabel 2. Waktu eksekusi (detik) metode CG untuk matriks gr_30_30.

Percobaan metode CG untul nilai toleransi 10^{-5} konvergen pada iterasi ke-33, sedangkan untuk nilai toleransi 10^{-10} konvergen pada iterasi ke-46. Tabel 2 di atas memeperlihatkan bahwa waktu eksekusi metode CG secara paralel masih lebih lambat daripada waktu eksekusi sekuensialnya kecuali pada percobaan paralel untuk nilai toleransi 10^{-10} dengan menggunakan 3 dan 4 komputer. Data pada Tabel 2, disajikan dalam bentuk grafik di gambar 3.



Gambar 3. Grafik waktu eksekusi (detik) metode CG untuk matriks gr_30_30

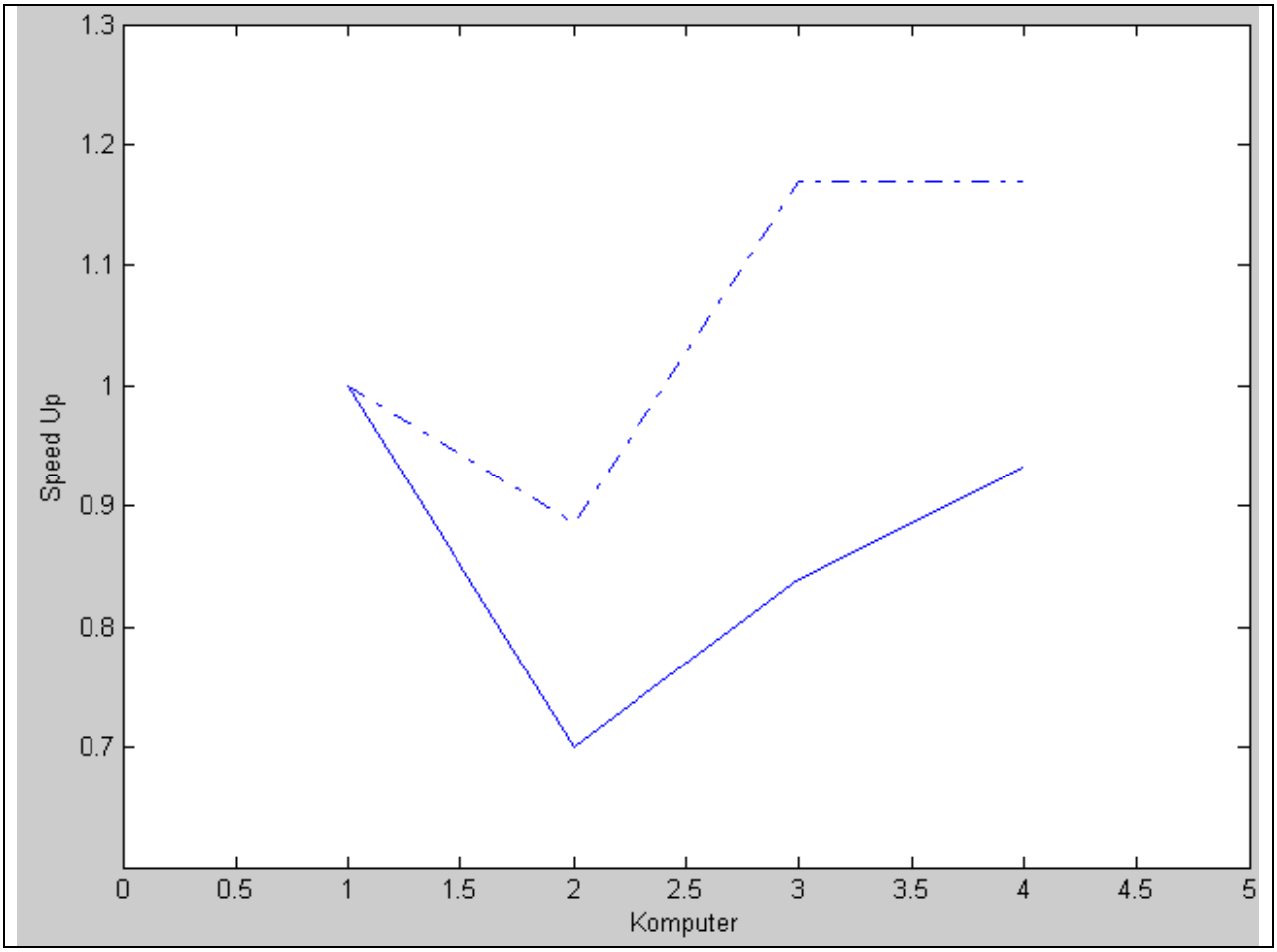
Walaupun beberapa waktu eksekusi pada percobaan paralel masih lebih lambat daripada waktu eksekusi sekuensialnya, waktu eksekusi pada percobaan paralel menurun, baik untuk nilai toleransi 10^{-5} maupun untuk nilai toleransi 10^{-10}. Hal ini terlihat jelas pada grafik di atas.

Dari data yang terdapat pada Tabel 2, dapat diperoleh nilai *speedup* yang dicapai. Berikut ini adalah tabel *speedup* yang dicapai pada percobaan ini.

Toleransi	Iterasi	Jumlah Komputer			
		1	2	3	4
10^{-5}	33	1	0.7	0.84	0.933
10^{-10}	46	1	0.886	1.17	1.17

Tabel 3. *Speedup* metode CG untuk matriks gr_30_30

Speedup yang berhasil dicapai pada percobaan ini sangat kecil, *speedup* terbesar hanya 1.17. Tabel di atas memperlihatkan bahwa waktu eksekusi pada percobaan paralel masih lebih lambat atau cenderung sama dengan waktu eksekusi sekuensialnya. Hal ini disebabkan beban komputasi dan jumlah iterasinya terlalu kecil.



Gambar 4. Grafik speedup metode CG untuk matriks gr_30_30

Dari grafik di atas, terlihat bahwa *speedup* yang dicapai pada percobaan paralel untuk nilai toleransi 10^{-10} lebih besar daripada *speedup* yang dicapai pada percobaan untuk nilai toleransi 10^{-5}. Hal ini disebabkan pralelisasi metode CG dilakukan pada tahapan yang berulang-ulang, sedangkan jumlah iterasi pada percobaan paralel untuk nilai toleransi 10^{-10}.

2. Hasil percobaan komputasi dengan menggunakan matriks nos3.

Matriks nos3 merupakan koleksi dari Harwell-Boeing (HB) sama dengan matriks gr_30_30. Elemen-elemen matriks ini bersifat definit positif.

Vektor konstanta **b** yang digunakan dalam percobaan ini adalah hasil dari perkalian matriks nos3 dengan vektor yang semua elemennya bernilai satu dan berukuran 960, yaitu **b** = nos3 **. 1;** **1** = $(1,1,\ldots,1)^T$. Tebakan awal $x_{(0)}$ yang digunakan adalah vektor nol berukuran 960, $\mathbf{x}_{(0)} = (0,0,\ldots,0)^T$.

Percobaan dilakukan untuk dua buah nilai toleransi *tol*, yaitu 10^{-5} dan 10^{-10} dengan iterasi maksimal *imaks* 960. Pada tabel 4 berikut, diberikan hasil waktu eksekusi metode CG untuk matriks nos3.

Toleransi	Iterasi	Jumlah Komputer			
		1	2	3	4
10^{-5}	219	3.28	2	1.5	1.27
10^{-10}	284	4.25	02.5	1.8	1.55

Tabel 4. Waktu eksekusi (detik) metode CG untuk matriks nos3

Metode CG konvergen untuk nilai toleransi 10^{-5} pada iterasi ke-219, sedangkan untuk nilai toleransi 10^{-10} pada iterasi ke-284. Jumlah iterasi pada percobaan ini jauh lebih besar daripada jumlah iterasi pada percobaan metode CG untuk matriks gr_30_30, sehingga waktu eksekusinya lebih lama. Namun waktu eksekusi pada seluruh percobaan paralel lebih cepat daripada waktu eksekusi sekuensialnya.

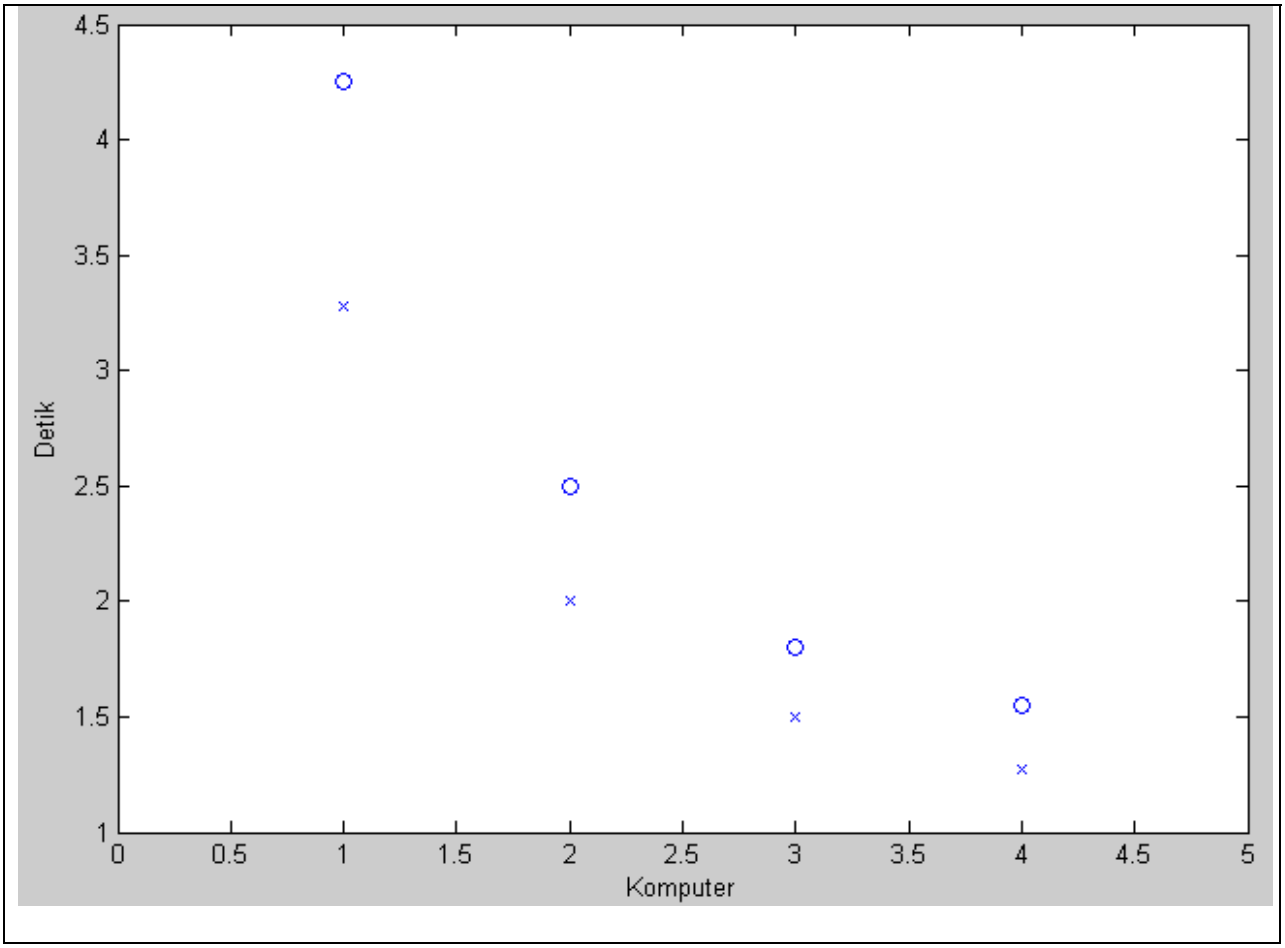
Tabel 4 memperlihatkan bahwa semakin banyak komputer yang digunakan dalam percobaan, waktu eksekusi semakin kecil. Hal ini terjadi baik pada percobaan untuk toleransi 10^{-5} maupun untuk toleransi 10^{-10}. Data waktu eksekusi hasil percobaan pada Tabel 4 disajikan dalam bentuk grafik pada Gambar 5. *Speedup* yang telah dicapai pada percobaan metode CG untuk matriks nos3 terdapat dalam tabel 5.

Toleransi	Iterasi	Jumlah Komputer			
		1	2	3	4
10^{-5}	219	1	1.64	2.187	2.583
10^{-10}	284	1	1.7	2.361	2.742

Tabel 5. Speedup metode CG untuk matriks nos3

Speedup yang dicapai pada percobaan paralel untuk matriks nos3 cukup besar. *Speedup* yang dicapai dua kali lebih besar daripada *speedup* yang dicapai pada percobaan metode CG secara paralel untuk matriks gr_30_30. *Speedup* terbesar mencapai 2.74, yaitu pada percobaan paralel untuk toleransi 10^{-10} dengan menggunakan 4 komputer, artinya waktu eksekusinya lebih cepat 2.47 kali dari waktu eksekusi sekuensialnya.

Dari tabel di atas diketahui bahwa *speedup* pada percobaan paralel untuk kedua nilai toleransi semakin besar seiring bertambahnya jumlah komputer yang digunakan. Tabel 5 juga memperlihatkan bahwa *seepdup* yang dicapai pada percobaan paralel untuk nilai toleransi 10^{-10} lebih besar daripada *speedup* yang dicapai pada percobaan untuk nilai toleransi 10^{-5} dengan menggunakan jumlah komputer yang sama. Kedua hal terlihat jelas dalam grafik *speedup* pada Gambar 6.



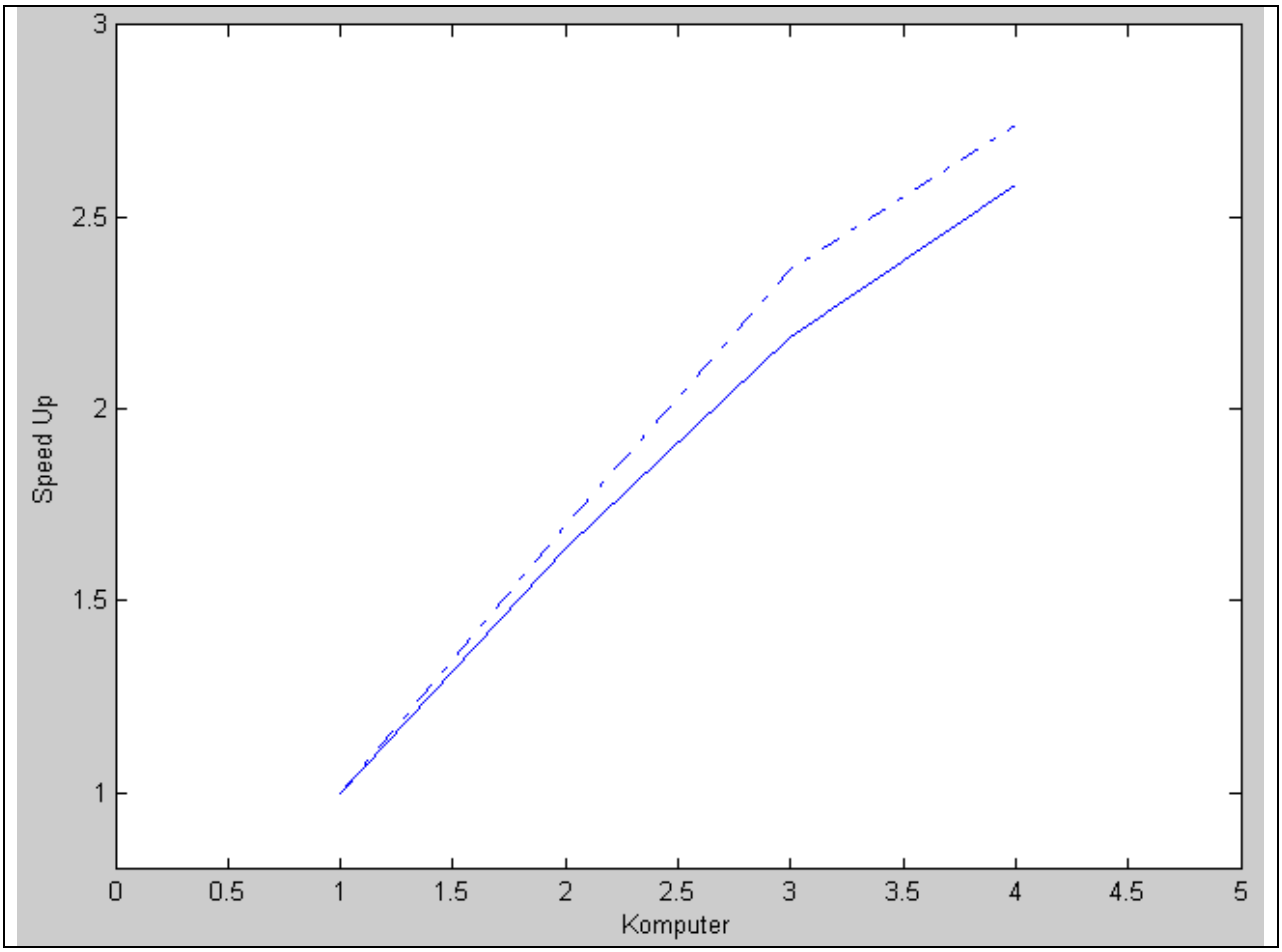
Gambar 5. Grafik waktu eksekusinya (detik) metode CG untuk matriks nos3

3. Hasil percobaan komputasi dengan menggunakan matriks ex13.

Matriks ketiga yang digunakan pada percobaan sebagai matriks A adalalah matriks ex13. Matriks ex13 merupakan koleksi dari FIDAP (*Fluid Dynamics Analysis Package*) berbeda dengan matriks gr_30_30 dan nos3. Elemen-elemen matriks ini adalah bilangan real. Matriks ini berbentuk persegi dan berukuran 2568x2568. Matriks ini bersifat simetrik definit positif.

Vektor konstanta **b** yang digunakan dalam percobaan ini adalah hasil dari perkalian matriks ex13 dengan vektor yang semua elemennya bernilai satu dan berukuran 2568,

b = ex13 **. 1;** **1** = $(1,1,\ldots,1)^T$. Tebakan awal $\mathbf{x}_{(0)}$ yang digunakan adalah vektor nol dg panjang 2568, $\mathbf{x}_{(0)} = (0,0,\ldots,0)^T$.



Gambar 6. Grafik speedup metode CG untuk matriks nos3.

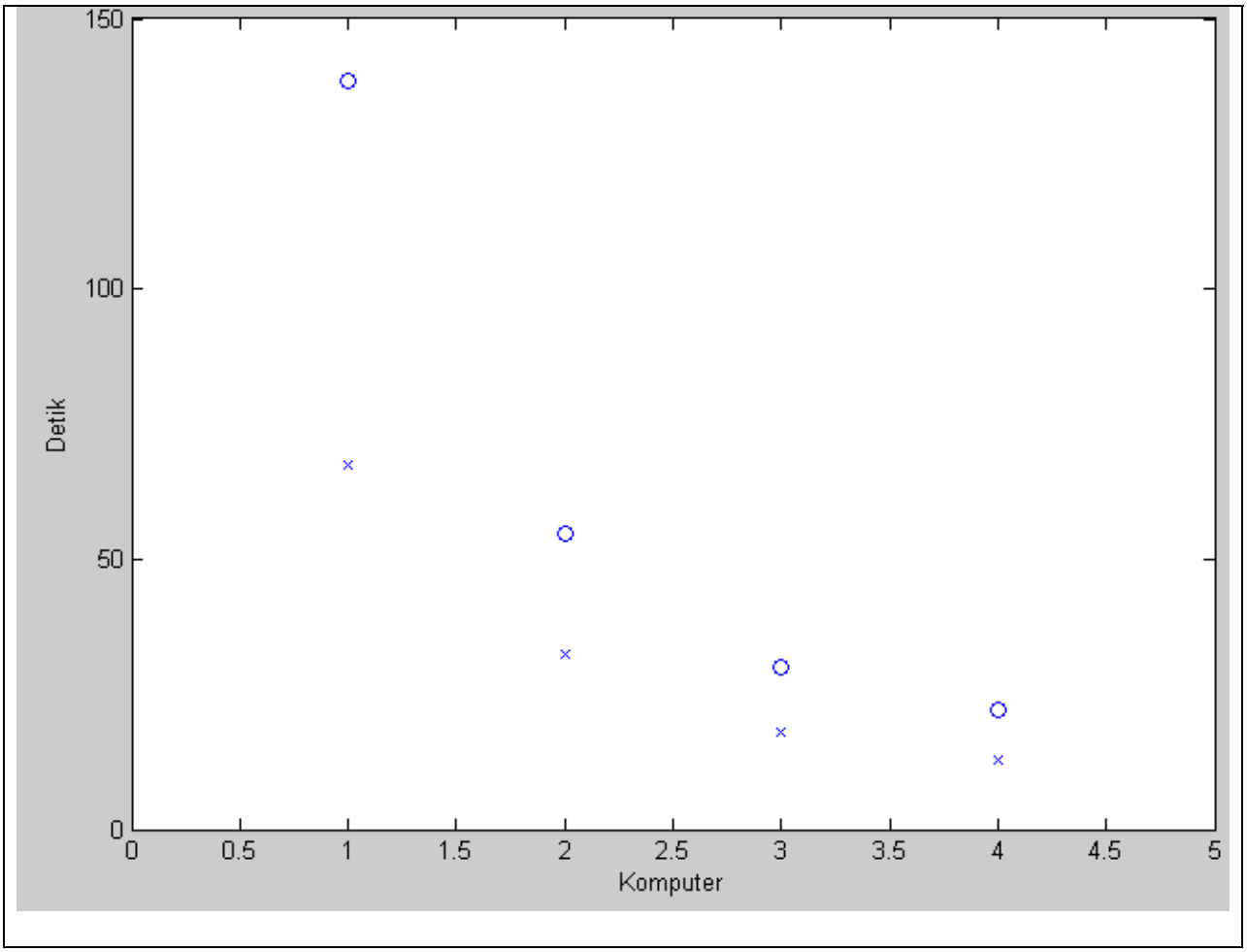
Percobaan dilakukan untuk dua buah nilai toleransi *tol*, yaitu 10^{-5} dan 10^{-10} dengan iterasi maksimal *imaks* 2568. Pada tabel 6, diberikan hasil waktu eksekusi metode CG untuk matriks ex13.

Toleransi	Iterasi	Jumlah Komputer			
		1	2	3	4
10^{-5}	665	67.52	32.5	18	13
10^{-10}	1342	138.6	54.6	30	22

Tabel 6. Waktu eksekusi (detik) metode CG untuk matriks

Waktu eksekusi percobaan metode CG untuk matriks ex13 sangat lambat bila dibandingkan dengan dua percobaan sebelumnya. Selain ukuran matriks ex13 jauh lebih besar daripada matriks gr_30_30 dan matriks nos3, iterasi yang dibutuhkan untuk masing-masing nilai toleransi juga lebih besar.

Dua waktu eksekusi lain percobaan pada Tabel 6 disajikan dalam bentuk grafik pada Gambar 7.



Gambar 7. Grafik waktu eksekusi (detik) metode CG untuk matriks es13

Waktu eksekusi pada seluruh percobaan metode CG untuk matriks ex13 secara paralel lebih cepat daripada waktu eksekusi pada percobaan sekuensialnya. Grafik di atas memperlihatkan bahwa semakin banyak komputer yang digunakan dalam percobaan, waktu eksekusi semakin kecil. Pada tabel 7, diberikan *speedup* metode CG untuk matriks ex13.

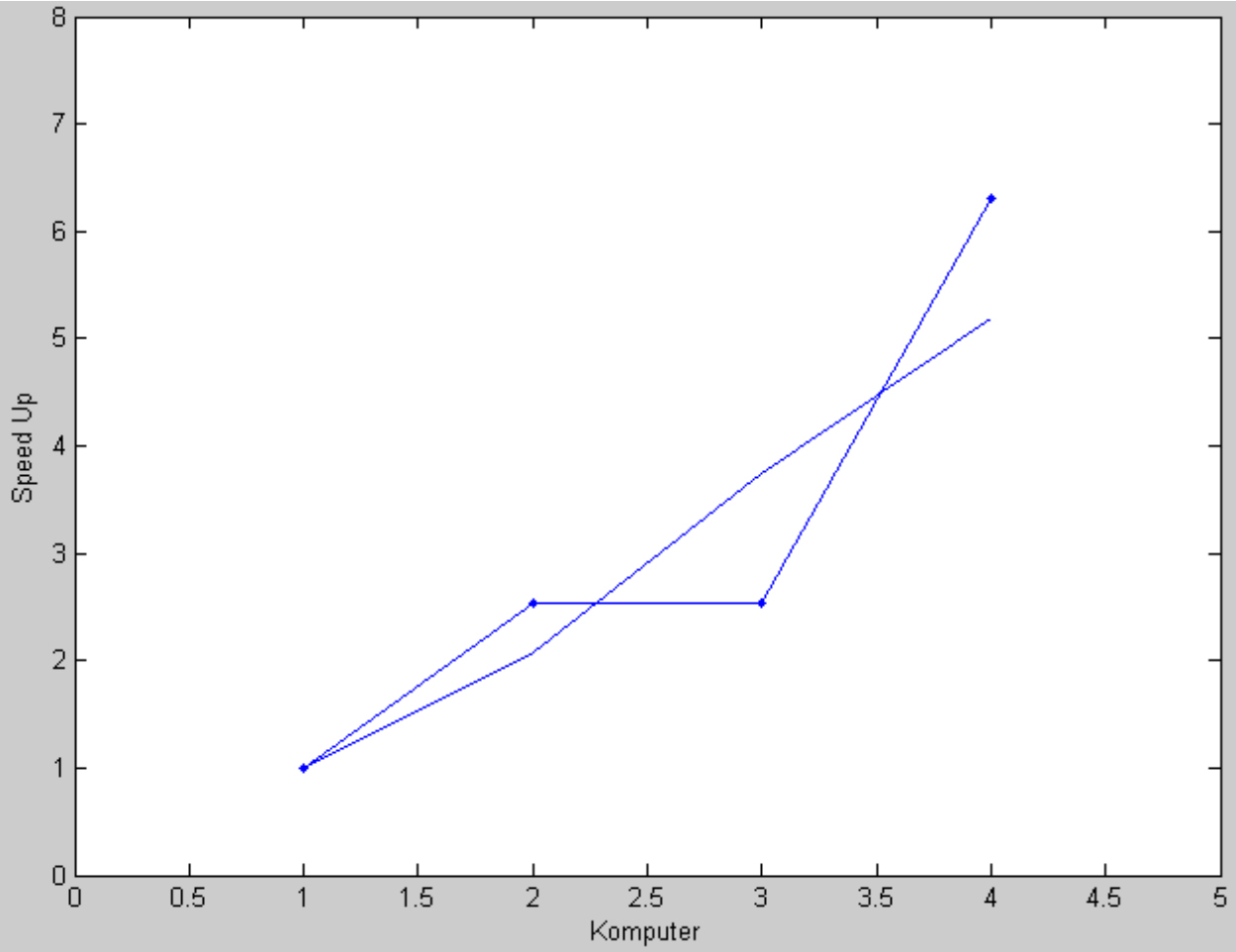
Toleransi	Iterasi	Jumlah Komputer			
		1	2	3	4
10^{-5}	665	1	2.078	3.751	5.194
10^{-10}	1342	1	2.538	2.538	6.299

Tabel 7. *Speedup* metode CG untuk ex13.

Walaupun waktu eksekusi pada percobaan ini sangat lembat, namun *speedup* yang dicapai sangat besar. *Speedup* terbesar yang dicapai pada percobaan ini adalah 6.3, artinya waktu eksekusi paralel 6.3 kali lebih cepat daripada waktu eksekusi sekuensialnya.

Speedup yang dicapai pada percobaan paralel untuk matriks ex13 lebih besar daripada *speedup* pada dua percobaan sebelumnya. Nilai *speedup* yang besar menujukkan bahwa waktu

eksekusi pada percobaan paralel untuk matriks ex13 jauh lebih cepat daripada waktu eksekusi sekuensialnya.Data pada Tabel 7 disajikan dalam bentuk grafik di gambar 8.



Gambar 8. Grafik *speedup* metode CG untuk matriks ex13.

Grafik di atas memperlihatkan bahwa *speedup* pada percobaan paralel untuk kedua nilai toleransi semakin besar seiring bertambahnya jumlah komputer yang digunakan. Grafik pada Gambar 11 di atas juga memperlihatkan bahwa *speedup* yang dicapai pada percbobaan paralel untuk nilai toleransi 10^{-10} lebih besar daripada *speedup* yang dicapai pada percobaan untuk niai toleransi 10^{-5} dengan menggunakan jumlah komputer yang sama.

SIMPULAN

Sebuah algoritma paralel untuk metode *Conjugate Gradient* telah berhasil dibuat dalam tulisan ini. Paralelisasi metode *Conjugate Gradient* dilakukan dengan matriks-vektor yang terdapat pada setiap iterasi metode *Conjugate Gradient*. Metode *Conjugate Gradient* paralel yang telah dibuat berhasil diterapkan dengan menggunakan SCILAB dan PVM dalam suatu jaringan komputer.

Percobaan metode *Conjugate Gradient* baik secara sekuensial maupun paralel dilakukan untuk menyelesaikan 3 buah sistem persamaan linear dengan matriks A yang berbeda-beda. Percobaan pada setiap sistem persamaan linear dilakukan untuk niai toleransi 10^{-5} dan 10^{-10}. Percobaan metode *Conjugate Gardient* paralel dilakukan dengan meggunakan 2, 3, dan 4 komputer.

Percoban metode *Conjugate Garadient* paralel untuk sistem persamaan linear dengan matriks nos3 dan matriks ex13 berhasil mencapai *speedup* yang cukup besar, artinya waktu eksekusinya lebih cepat dari pada waktu eksekusi sekuensialnya. *Speedup* yang di capai pada percobaan paralel untuk matriks gr_30_30 sangat kecil, artinya waktu eksekusinya cenderung sama atau lebih lambat dari pada waktu eksekusi sekuensialnya. Hal ini di sebabkan beban komputasi dan jumlah iterasi terlalu kecil.

Dari percobaan-percobaan yang telah dilakukan, diketahui bahwa nilai *speedup* yang di capai pada setiap percobaan paralel sellu bertambah seiring bertambahnya jumlah komputer yang di gunakan. *Speedup* yang dicapai pada setiap percobaan metode *Conjugate Gradient* paralel untuk nilai toleransi 10^{-10} lebih besar dibandingkan pada percobaan untuk toleransi 10^{-5}.

Ucapan Terima Kasih. Pekerjaan in didanai oleh Kementrian Pendidkan dan Kebudayaan, Republik Indonesia, melalui "Penelitian Strategis Unggulan", hibah DIPA-IPB, 0558/023-04.2.01/12/2012, dengan kontrak **no : 44/I3.24.4/SPK-PUS/IPB/2012.**

DAFTAR PUSTAKA

Barret, R, M Berry, T F Chan, J Demmel, J Donato, J Dongarra, V Eijkhout, R Pozo, C

Romine, H Van Der Vorst. 1994. Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods. Philadelphia: SIAM.

Basuki, S. A. 2001. Komputasi Paralel Sebagai Altrnatif Solusi Peningkatan Kinerja Komputasi. INTEGRAL vol.6, no.2.

Beezer, R. A. 2006. A First Course in Linear Algebra. Departement of Mathematics and Computer Science: University of Puget Sound.

Davis, T. 2006. University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices/. [22 November 2006]

Fisher, B, S Toupet, R Vuduc. 1998. Assignment 4: Parallel Conjugate Gradient. http://www.cs.berkeley.edu/~richie/cs267/cg/results/. [22 November 2006]

Geist, A, A Beguelin, J Dongarra, W Jiang, R Manchek, V Sunderam. 1994. PVM: Parallel Virtual Machine, A Users' Guide and Tutorial for Networked Parallel Computing. Massachusetts: The MIT Press. http://www.netlib.org/pvm3/book/pvm-book.ps

Grama, A, A Gupta, G Karypis, V Kumar. 2003. Introduction to Prallel Computing. London: Addison Wesley.

Shewchuk, J. R. 1994. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf.
[22 November 2006]

Lampiran 1. Program SCILAB untuk metode CG.

```
Function [x, i] = CG(A, b, tol)

    x = zeros(b);
    r = b – A * x;
    norm_r0 = norm(r);
    p = r;
    rho = r’ * r;
    for I = 1: length(b)
        q = A * p;
        alp = rho / (p’ * q);
        x = x + alp * p;
        r = r – alp * q;
        if norm(r) / norm_r0 < tol
            break;
        end
        new_rho = r’ * r;
        bet = new_rho / rho;
        p = r + bet * p;
        rho = new_rho;
    end

endfunction
```

Lampiran 2. Program SCILAB untuk metode CG yang telah diparalelisasi.

```
Function [x, i] = ParCG(A, b, tol, N)

    Sa = sparse(A);
    [m,n] = size(A);
    N = min(m,N);
    bl = round(m/N);
    for k = 1:N-1
            xadj= sp2adj(Sa((k-1)*bl+1:k*bl,:));
            for I = 1:n+1
                if xadj(i) > 1
                        coll(k) = i-1;
                        break;
                end

            end

            for I = n+1:-1:1
                if xadj(i) > xadj(i-1)
                        col2(k) = i-1:
                        break:
                end

            end

    end
    xadj= sp2adj(sA((N-1)*bl+1:$,:));
    for I = 1:n+1
            if xadj(i) > 1
                coll(N) = i-1;
                break;


            end

    end
    for i = n+1:-1:1
            if xadj(i) > xadj(i-1)
            col2(N) = i-1;
                break;
            end

    end
    [tid,nt] = pvm_spawn('~/script1.sce',N,'nw);
    for k = 1:N-1
            pvm_send(tid(k),A((k-1)*bl+1:k*bl,coll(k):col2(k)),1);
            pvm_send(tid(k),'AA=%var',1);



    end
```

```
pvm_send(tid(N),A((N-1)*bl+1:$,coll(N):col2(N)),1);
pvm_send(tid(N),'AA=%var',1);


x = zeros(b);
r = b- A * x;
norm_r0 = norm(r);
p = r;
q = zeros(p);
rho = r' * r;
for I = 1: length(b)
        for k = 1:N

                pvm_send(tid(k), p(coll(k):col2(k)), 1);
                pvm_send(tid(k), 'pvm_send(parent, AA *%var, 1)', 1);
        end

                for k =1: N - 1
                        q((k-1) * bl + 1: k * bl) = pvm_recv(tid(k),1);

                end
                q((N - 1) * bl + 1 :n) = pvm_recv(tid(N), 1);
                alp = rho / (p' * q);
                x = x + alp * p;
                r = r - alp * q;
                if norm(r) / norm_r0 < tol
                        break;
                end
                new_rho = r' * r;
                bet = new_rho / rho;
                p = r + bet * p;
                rho = new_rho;
        end

endfunction
```

Script1.sce

```
// Copyright INRIA
Parent=pvm_parent(),if parent<0 then return,end
while%t                                 //Infinite loop
  [buf,into] = pvm_recv(parent, -1) //get new variable or instruction

  if info<0 then break, end
  if type(buf)==10 then                                 //an instruction
  if execstr(buf,'errcatch') then break, end//execute it
//   write(%io(2),'Instruction: '+buf+' done')
else                                                    //a variable
  %var=buf //preserve it in %var
//   disp('Variable received')
  end
end
```