Jurnal Integrasi
vol. 4, no. 1, 2012, 47-52
ISSN: 2085-3858 (print version)

Article History
Received 13 February 2012
Accepted 16 March 2012

Eksperimen Komputasi Parallel dalam Perhitungan Matrik Invers Menggunakan Metoda Eliminasi Gauss Jordan

Riwinoto

Teknik Informatika, Politeknik Batam
Email: riwi@polibatam.ac.id

ABSTRACT

Komputasi parallel menjanjikan perfomansi yang baik dalam menyelesaikan perhitungan dengan jumlah data yang besar. Mencari invers dari suatu matrik merupakan salah satu operasi matrik yang paling sering dilakukan dalam mencari solusi persamaan linier. Komputasi parallel pada cluster grid dengan menggunakan MPI sebagai middleware untuk menyelesaikan perhitungan matrik invers yang besar dapat diperkecil waktu komputasinya dengan membesarkan jumlah prosessor yang terlibat pada komputasi parallel. Eksperimen menunjukkan komputasi matrik invers dengan jumlah data kecil atau besar dengan menggunakan prosessor berjumlah 2,4,8, dan 16 dalam perhitungan matrik inverse dengan algoritma pivot menghasilkan perfomansi yang lebih buruk dibandingkan komputasi dengan prosessor tunggal berdasarkan speed up, overhead, efisensi cost dan algoritmik pembagian kerja dari komputasi parallel.

Kata Kunci: Komputasi paralel,grid,, matrik inverse, MPI

PENDAHULUAN

Kebutuhan pengaksesan informasi kapanpun dan dimanapun sekarang ini mendorong penyediaan sumber daya kom-putasi yang lebih powerful. Dunia science dan industri semakin hari juga semakin membutuhkan sumber daya komputasi yang sangat besar. Simulasi gempa bumi, pemodelan struktur molekul kimia, dan simulasi per-ubahan iklim adalah beberapa contoh aplikasi yang melibatkan data yang sangat besar mencapai satuan petabyte (1015 byte). Hal ini tentu saja membutuhkan sumber daya komputasi yang besar pula. Dengan semakin banyaknya sumber daya komputasi yang saling terhubung jaringan, komputasi parallel memberikan harapan untuk mengatasi permasalahan itu[1].

DETAIL PERCOBAAN

Komputasi Paralel

Komputasi paralel merupakan suatu teknologi yang dapat mendayagunakan seluruh kemampuan multiprosesor dalam suatu aplikasi. Walaupun sudah cukup banyak aplikasi yang dapat mengeksploitasi kemampuan multiprosesor, namun komputasi paralel ini belum diadaptasi secara luas untuk aplikasi-aplikasi umum. Hal ini cukup penting mengingat di tahun-tahun mendatang, penggunaan multiprosesor pada suatu sistem akan menjadi hal yang umum.

MPI (Message Parsing Interface)

Message Passing Interface (MPI) merupakan implementasi standar dari model "message passing" komputasi paralel. Sebuah komputasi paralel terdiri dari sejumlah proses, dimana masing masing bekerja pada beberapa data local [6]. Setiap prsoses mempunyai variabel lokal, dan tidak ada mekanisme suatu proses yang bisa mengakses secara langsung memori yang lain. MPI perlu digunakan ketika kita menginginkan kode paralel yang portable dan mendapatkan performa yang tinggi dalam pemrograman parallel[1].

Sharing data antar proses-proses dilakukan dengan message passing, yaitu dengan mengirim dan menerima message antar proses-proses. Alasan menggunakan model ini sudah sangat umum. Dan tipe komputasi parallel dapat dibuat dalam bentuk message passing. Model ini juga dapat diimplementasikan pada bermacam-macam platform, seperti shared-memory multi-prosessors maupun single prosessor.

1. Fitur dasar dari program-program Message Passing

Program message passing terdiri dari multiple instan dari program serial yang berkomunikasi dengan pemanggilan library. Pemanggilan ini dapat dibagi menjadi empat kelas:

- Pemanggilan untuk inisialisasi, meng-atur, dan memutuskan komunikasi.
- Pemanggilan untuk mengkomunikasi-kan beberapa pasang prosesor.
- Pemanggilan yang mewujudkan operasi-operasi komunikasi diantara kelompok-kelompok prosesor.
- Pemanggilan untuk membuat tipe data acak (arbitrary).

Kelas pertama dari pemanggilan terdiri dari pemanggilan-pemanggilan untuk memulai komunikasi, mengidentifikasi jumlah prosesor yang sedang digunakan, membuat subgrup dari prosesor-prosesor dan mengidentifikasi prosesor mana yang sedang menjalankan instan khusus dari program.

Kelas kedua dari pemanggilan, dinamakan komunikasi point-to-point, terdiri atas tipe pengiriman dan penerimaan yang berbeda.

Kelas ketiga dari pemanggilan adalah operasi kolektif yang memberikan sinkronisasi antar grup proses dan pemanggilan yang melakukan operasi komunikasi/kalkulasi.

Kelas terakhir dari pemanggilan memberikan eksibilitas dalam berurusan dengan struktur data yang rumit.

2. Struktur Program MPI

Program paralel dengan MPI mempunyai struktur sebagai berikut:

- include MPI header file
- variable declarations
- initialize the MPI environment
- ...do computation and MPI communication calls...
- close MPI communications

Berikut adalah contoh dalam bahasa C,

```
#include
#include
void main (int argc, char *argv[])
{
    int err;
    err = MPI_Init(&argc, &argv);
    printf("Hello world!\n");
    err = MPI_Finalize();
}
```

Matrix inverse

Jika A dan B adalah matrik yang berorde sama dengan memenuhi persamaan AB = I dan BA = I maka A adalah matrik invers dari B atau B adalah matrik invers dari A.

1. Metoda Eliminasi Gauss Jordan

Pada penyelesaian sistem persamaan linier, Metoda eliminasi gauss Jordan digunakan untuk membentuk matrik baru dengan elemen-elemen di bawah dan diatas elemen diagonal utama bernilai 0 dari sebuah matrik [2].

Eliminasi Gauss Jordan untuk mencari invers dari matrik dilakukan dengan menambahkan matrik identitas disebelah kanan matrik A dengan dimensi yang sama dan dilakukan operasi-operasi matrik sebagai berikut:

$$[AI] \rightarrow A^{-1}[AI] \rightarrow [IA^{-1}]$$

Sehingga pada akhir operasi didapatkan pasangan matrik identitas disebelah kiri dan matrik invers A disebelah kanan.

Prosedur umum dari metoda eliminasi gauss Jordan adalah:

a. Tulis matrik gabungan [A|b] untuk system persamaan linear B=AX
b. Gunakan operasi baris elementer pada matrik gabungan [A|b] untuk mentransformasikan A ke bentuk diagonal. Jika terdapat nilai 0 pada garis diagonal . Tukar dengan baris sampai pada tempat tersebut bernilai tidak sama dengan 0. Jika tidak bisa maka berhenti. Sistem infinite atau tidak punya solusi.
c. Dengan membagi elemen diagonal dan elemen kanan sisi dalam pada setiap baris dengan elemen diagonal pada baris tersebut, buat supaya setiap elemen diagonal sama dengan 1

Berikut adalah contoh penyelesaian masalah pada persamaan linear dengan menggunakan eliminasi gauss:

Untuk sistem persamaan linier:

$2y + z = 4$

$X + y + 2x = 6$

$2x + y + z = 7$

a. Tulis matrik gabungan [A|b] untuk system persamaan linear AX=B

Diubah menjadi bentuk matrik sebagai berikut:

$$\left[\begin{array}{ccc|c} 0 & 2 & 1 & 4 \\ 1 & 1 & 2 & 6 \\ 2 & 1 & 1 & 7 \end{array}\right]$$

b. Lakukan pencarian matrik diagonal

- Baris 2 ditukar dengan baris 1 hasilnya sebagai berikut:

$$\left[\begin{array}{ccc|c} 1 & 1 & 2 & 6 \\ 0 & 2 & 1 & 4 \\ 2 & 1 & 1 & 7 \end{array}\right]$$

-Update baris 3 dengan

r 3= r 3+ (-2 x r1)

hasilnya sebagai berikut:

$$\left[\begin{array}{ccc|c} 1 & 1 & 2 & 6 \\ 0 & 2 & 1 & 4 \\ 0 & -1 & -3 & -5 \end{array}\right]$$

-Update baris 1 dan baris 3

r1=r1+ (-0.5x r2)

r3=r3+(0.5 * r2)

hasilnya sebagai berikut:

$$\left[\begin{array}{ccc|c} 1 & 0 & 3/2 & 4 \\ 0 & 2 & 1 & 4 \\ 0 & 0 & -5/2 & -3 \end{array}\right]$$

- update baris 1 dan baris 2

r1=r1+ (3/5 x r3)

r2=r2+ (2/5 x r3)

hasilnya sebagai berikut

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 11/5 \\ 0 & 2 & 1 & -14/3 \\ 0 & 0 & -5/2 & -3 \end{array}\right]$$

c. Pembagian dengan sebuah nilai untuk mendapatkan matrik diagonal bernilai 1 dan solusi pada sisi kanan matrik.

Dilakukan dengan mengupdate baris 2 dan baris 3

r2=r2* (-0.5)

r3=r3 * (-2/5)

hasilnya sebagai berikut:

$$\left[\begin{array}{ccc|c} 1 & 0 & 0 & 7/5 \\ 0 & 1 & 1 & 11/5 \\ 0 & 0 & 1 & 6/5 \end{array}\right]$$

Jadi didapatkan

X = 7 / 5

Y = 11 / 5

Z = 6 / 5

Pada komputasi matrik, salah satu pendekatan yang digunakan untuk mencari invers dari sebuah matrik adalah eliminasi Gauss Jordan.

Algoritma yang dibandingkan adalah sekuensial dan parallel [3]. Implementasi algoritma sekuensial dilakukan pada satu server. Sedangkan implementasi algoritma parallel pada pencarian matrik inverse membutuhkan dua pihak yaitu server master dan worker dengan teknik pivot . Berikut adalah algoritma yang digunakan dalam penelitian ini.

Algoritma sekuensial pada matrik invers

```
For  col = 1..n:
    Choose pivot row from matrix
    Swap rows between pivot and column
    Scale pivot row for each column
    For row = 1..n:
        If (row != col)
            Add row multiple
    End for
End for
```

Algoritma parallel pada matrik invers dengan pivot

For i = 1..m:

Workers: Send best pivot row (index i) to master.

Worker who owns row i: Send row i to master.

Master:

Choose best pivot row out of all pivot rows received.

Divide the pivot row by its leading coefficient.

Send row i, the rescaled pivot row, and the pivot row index to all workers.

Worker who owns row i: Replace row i with pivot row.

Worker who owns pivot row: Replace pivot row with row i.

All workers: Set all non-diagonal entries in column i to zero by subtracting the appropriate multiple of the pivot row from all rows above and below row i.

End For

Lingkungan Percobaan

Eksperimen dilakukan menggunakan cluster Hastinapura melalui portal InGrid UI. Cluster Hastinapura mempunyai 16 node dengan masing-masing node menggunakan Dual-Core prosessor, sehingga secara lojik jumlah prosessornya adalah 32 buah. Pada percobaan yang dilakukan maximum memory ditentukan sebesar 400 MB dengan maximum time 60 menit.

Analisis Hasil Eksperimen

Aspek yang dianalisa terhadap hasil eksperimen adalah *speed up*, *overhead*, efisiensi, *cost* dan pembagian beban setiap node [4]:

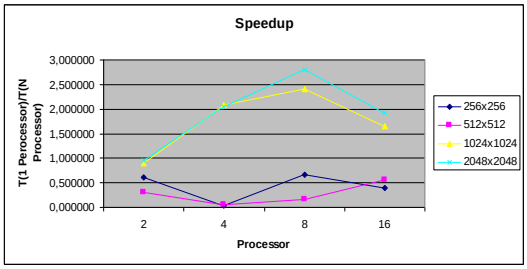
1. Speed Up

Speed Up merupakan perbandingan antara kecepatan eksekusi pada single prosessor dengan pada N prosessor. Speed Up digunakan mengecek reduksi waktu seiring dengan penambahan jumlah prosessor. Berikut rumus dari speed up.

$$S(n) = \frac{ts}{tn} \quad (1)$$

ts=waktu eksekusi menggunakan 1 prosessor

tn= waktu eksekusi menggunakan 1 prosessor



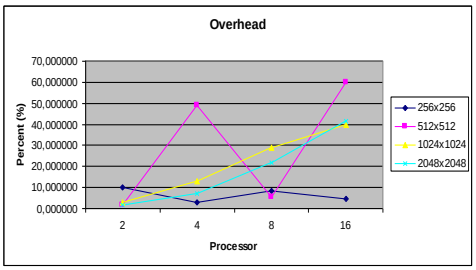
Gambar 1: Perbandingan Speed Up

Dari Gambar 1 terlihat bahwa:

- Speed pada penggunaan beberapa prosessor dengan ukuran matrik bervariasi tidak stabil.
- Speed up terbaik cenderung terjadi pada penggunaan prosessor sebanyak 8 buah.
- Perfomasi speed up terburuk cenderung pada penggunaan prosessor sebanyak 4 buah pada penggunaan data kecil (256 dan 512).

2. Overhead

Overhead merupakan perbandingan antara waktu komunikasi master dengan slave dengan waktu total eksekusi. Overhead digunakan untuk mengetahui seberapa besar pengaruh waktu komunikasi terhadap perfomansi komputasi. Semakin kecil overhead semakin bagus perfomansi komputasi parallel.



Gambar 2: Overhead

Dari Gambar 2 terlihat bahwa *overhead* cenderung membesar seiring dengan membesarnya jumlah prosessor. Ini terlihat jelas pada data berukuran besar. Pada data kecil (256) *overhead* menunjukkan kecenderungan menurun seiring dengan penambahan prosessor.

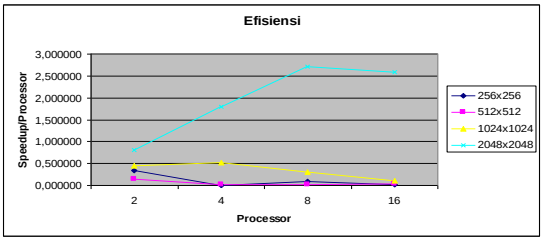
3. Efisiensi

Efisiensi meliputi efisiensi N prosessor speed up terhadap jumlah prosessor. Besarnya efisiensi menunjukkan prosessor seluruhnya digunakan untuk komputasi. Rumus sebagai berikut,

$$E(n) = \frac{S(n)}{N} \quad (2)$$

S(n) : speed up untuk N prosessor

N: Jumlah total prosessor



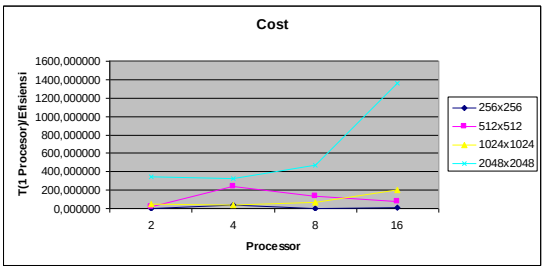
Gambar 3: Efisiensi

Dari grafik perbandingan efisiensi terlihat bahwa Efisiensi pada matrik berukuran kecil (256-1024) cenderung menurun jika jumlah prosessor dinaikan namun untuk matrik besar (2048) maka efisiensi naik dengan perfomansi terbaik di jumlah prosessor 8 namun kemudian cenderung menurun.

4. *Cost*

Cost N prosessor merupakan beban waktu yang terjadi terhadap efisiensi tersebut, rumus sebagai berikut,

$$tp = \frac{tn}{S(n)} \qquad (3)$$



Gambar 4: Cost

Dari grafik cost terlihat pada matrik berukuran besar (2048), cost menjadi besar (mendekati fungsi kuadratik) sedangkan pada matrik berukuran yang lebih kecil, cost juga cenderung naik namun tidak tajam.

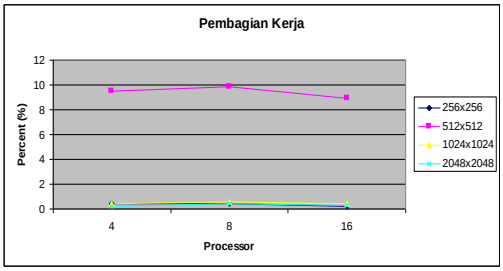
5. Pembagian beban kerja setiap node

Pembagian beban kerja yang seimbang mengindikasikan setiap node bekerja dengan beban yang sama. Spesifikasi hardware pada setiap node dan algoritma parallel yang digunakan menentukan beban kerja masing-masing node.

Beban kerja yang seimbang diindikasikan dengan nilai dari koefisien keseragaman (jika spesifikasi hardware node sama). Koefisien keseragaman adalah ukuran penyebaran nisbi relative dari suatu data dan dirumuskan sebagai berikut:

$$KK = (S/ \bar{Y}.) \times 100\% \qquad (4)$$

disini S adalah simpangan baku dan Y adalah rata-rata. Nilai keragaman biasanya dianggap baik jika $\leq 30\%$ [5].



Gambar 5: Pembagian kerja node slave

Dari grafik terlihat bahwa pembagian beban kerja masing-masing node slave untuk setiap ukuran matrik dan jumlah prosessor dianggap baik karena di bawah 30%.

Dari data perbandingan nilai rata-rata, minimum dan maksimum waktu proses slave menunjukkan nilai rata-rata juga hampir berada di sekitar pertengahan antara nilai min dan max. Hal berarti ada Kecenderungan linear pertambahan waktu proses antara node slave 1, node slave 2 sampai node terakhir. Hal ini dikarenakan eliminasi gauss jordan mempunyai ciri semakin sedikit komputasi ketika scaling dilakukan karena jumlah elemen yang terkena scaling semakin sedikit. Scaling dilakukan sesuai elemen diagonal.. Akibatnya setiap node mempunyai beban yang berbeda untuk setiap kali scaling. Dari penjelasan eliminasi gauss, secara teoritis node yang mendapat porsi potongan matrik terakhir akan mendapatkan beban kerja terbesar karena setiap langkah scaling selalu harus melakukan komputasi mulai dari kolom 1 sampai terakhir. Pada keterangan node terlihat pada node slave terakhir mempunyai waktu proses terbesar. Hal ini berarti pengujian sesuai dengan teoritis.

KESIMPULAN

Dari tersebut di atas, dapat diambil kesimpulan sebagai berikut:

1. Speed up yang terjadi tidak sesuai dengan keadaan ideal seperti yang diharapkan. Seharusnya speed berbanding linier dengan jumlah prosessor
2. Overhead yang terjadi membesar ketika jumlah prosessor yang terlibat semakin besar.
3. Efisiensi pada data besar menjadi optimal pada jumlah prosessor 8 namun ketika jumlah prosessor dinaikan maka efisiensi semakin menurun.Hal ini terjadi juga pada data yang kecil.
4. Cost waktu komputasi semakin besar jika jumlah prosessor semakin besar.
5. Pembagian beban kerja setiap node slave menunjukkan beban waktu proses yang tidak sama untuk setiap node. Meskipun demikian koefisien keragaman beban kerja menunjukkan peningkatan beban kerja untuk node yang mengerjakan potongan matrik dari atas ke bawah masih cukup baik pemerataanya dengan nilai koefisien < 30.
6. Dari kelima aspek tersebut terlihat secara bahwa penggunaan parallel computing dengan jumlah apapun tidak membuat perfomansi semakin baik dibandingkan jika dilakukan dengan single prosessor.

ACKNOWLEDGMENTS

Ucapan terima kasih penulis kepada Prof. Heru Suhartanto dan Dr. Boby Nazief dari Universitas Indonesia, Fakultas Ilmu Komputer yang memberikan akses ke server IngGrid UI.

REFERENSI

[1] Bern Mohr (2006), Introduction to Parallel Computing,, Computational Nanoscience: Do it your Self, J. Grotendorst, S. Bl ¨ugel, D. Marx (Eds.), John von Neumann Institute for Computing, Julich, NIC Series, Vol. 31, ISBN 3-00-017350-1, pp. 491-505, 2006

[2] Gauss Jordan Elimination , http://ceee.rice.edu/Books/CS/chapter2/linear44.html, diakses pada tanggal 6 oktober 2009

[3] Vancea.C, Vancea.F,Parallel Algorithm for Computing Matrix Inverse by Gauss-Jordan Method ,http://electroinf.uoradea.ro/reviste%20CSCS/documente/JCSCS_2008/JCSCS_2008_22_VanceaC_1.pdf, diakses tanggal 1 november 2009

[4] Parallel Computing, http://www.cfd-online.com/Wiki/Parallel_computing, diakses tanggal 31 oktober 2010

[5] Statistika-Deskriptif,I Putu Sampurna, http://staff.unud.ac.id/~sampurna/wp-content/uploads/2007/12, diakses tanggal 1 oktober 2010

[6] Message Passing Interface (MPI), Blaise Barney, Lawrence Livermore National Laboratory, https://computing.llnl.gov/tutorials/mpi/ diakses tanggal 11 April 2011