PARALEL BLOK FAKTORISASI QR DALAM SISTEM MEMORI TERSEBAR MULTIKOMPUTER BERBASIS MPI-LINUX

Abdul Rochman
Jurusan Teknik Informatika, Fakultas Teknologi Industri, Universitas Trisakti

ABSTRAK: Dalam tulisan ini akan dipaparkan implementasi dari paralel Blok Faktorisasi QR dengan bentuk *Compact* WY. Program paralel ditulis dalam model SPMD (*Single Program Multiple Data*) dan memanfaatkan pustaka MPI (*Message Passing Interface*) untuk komunikasi. Program ini sukses dijalankan dalam sistem memori tersebar, dengan empat komputer. Terjadi peningkatan kinerja (speedup) yang berarti seiring dengan penambahan jumlah prosesor dan penambahan ukuran matriks: 1.47 untuk dua prosesor, 1.84 untuk tiga prosesor dan 2.13 untuk empat prosesor.

Kata kunci: SPMD, blok faktorisasi QR bentuk *Compact* WY, sistem memori tersebar dan *speedup*

ABSTRACT: *This paper will present the implementation of parallel block factorization QR with Compact WY form. The parallel program has written in the SPMD (Single Program Multiple Data) style and use MPI (Message Passing Interface) library for communication. The program was successfully run in distributed memory system, with four computers. The Speedup was increase significantly long with increasing the number of processor and increasing the size of matrix*: *1.47 for two processors, 1.84 for three processors and* 2.13 *for four processors.*

Keywords: SPMD. *block factorization QR with compact WY form, distributed memory system, speedup.*

PENDAHULUAN

Faktorisasi QR merupakan algoritma yang stabil yang banyak dipakai dalam penerapan ilmiah seperti, program *solver* untuk menyelesaikan suatu sistem persamaan linear ($Ax = b$), penentuan semua nilai eigen dan semua vektor eigen dari suatu matriks.

Faktorisasi QR, mendekomposisi matriks A menjadi suatu matriks orthogonal Q dan suatu matriks segitiga atas R ($A=QR$), Algoritma konvensional untuk menghitung faktorisasi QR banyak melakukan operasi perkalian matriks-vektor ($C \leftarrow \tau A^T w$) dan operasi perbaharuan *rank*-1 ($A \leftarrow A\text{-}wC^T$). Algoritma ini kurang memanfaatkan secara optimal kemampuan terbaik yang dimiliki suatu komputer.

Schreiber dan Van Loan [2] memperbaharui algoritma faktorisasi QR, yang disebut algoritma blok faktorisasi QR menggunakan bentuk *Compact* WY. Algoritma ini banyak melakukan operasi perkalian matriks-matriks ($C \leftarrow A^TYT$) dan perbaharuan *rank-k* ($A \leftarrow A + YC^T$). Bila dibanding dengan algoritma konvensioal algoritma blok ini lebih memanfaatkan kemampuan terbaik dari suatu komputer.

Usaha berikutnya yang dapat dilakukan untuk meningkatkan kinerja dari algoritma blok faktorisasi QR adalah dengan menerapkannya dalam lingkungan komputasi paralel.

Penelitian ini mengimplementasikan algoritma blok faktorisasi QR menggunakan bentuk Compact WY dalam sistem memori tersebar multikomputer berbasis MPI-LINUX.

KOMPUTER PARALEL

Komputer paralel adalah komputer tunggal dengan beberapa internal prosesor atau beberapa komputer yang dihubungkan oleh suatu *interconnection network* (IN) Komputer paralel dapat dikelompokkan, berdasarkan pengorganisasian memorinya, ke dalam dua arsitektur dasar yaitu: sistem memori bersama (*shared memory*) dan sistem memori tersebar (*distributed memory*). [4]

Dalam sistem memori tersebar komunikasi antar proses menggunakan mekanisme pertukaran pesan (*message passing*). Program paralel dengan pertukaran pesan dapat ditulis menggunakan suatu bahasa pemrograman tingkat tinggi (misalnya: fortran, C/C++), dan menyertakan (atau pemanggilan) suatu pustaka pertukaran pesan MPI (*message passing interface*) atau PVM (*parallel virtual machine*). Pada penelitian ini digunakan digunakan pustaka MPI.

Terdapat dua struktur program parallel: *single program multiple data* (SPMD) dan *multiple program multiple data* (MPMD). Dalam struktur SPMD, hanya terdapat satu sumber program dan setiap prosessor akan mengeksekusi salinan dari program ini. Sedangkan dalam struktur MPMD, terdapat beberapa sumber program dan setiap prosessor megeksekusi program yang berbeda. [3].

Faktor *speedup*, *S*(*p*), adalah perbandingan antara waktu eksekusi sekuensial (t_s) terhadap waktu eksekusi program paralel(t_p): [1, 4]

$$S(p) = \frac{t_s}{t_p}.$$

Speedup maksimum adalah *p* dengan *p* prosesor. **Efisiensi**, *E*, yaitu efektivitas penggunaan prosesor, dan didefinisikan sebagai perbandingan faktor speedup dengan jumlah prosesor:

$$E = \frac{S(p)}{p}.$$

BLOK FAKTORISASI QR

Dalam Faktorisasi QR, suatu matriks *A* dikomputasi sedemikian hingga dihasilkan suatu matriks orthogonal *Q* dan matriks segitiga atas *R*, dan $A = Q \times R$. Faktorisasi QR dari suatu matriks *A* dapat dikomputasi menggunakan: **pencerminan Householder.**

Misalkan $w \in \Re^n$ dan $\|w\|_2 = 1$. Suatu matriks $H \in \Re^{n \times n}$ yang didapat dari persamaan berikut:

$$H = I - 2ww^T$$

adalah suatu pencerminan Householder (sering disebut: matriks Householder atau transformasi Householder). Vektor *w* disebut vektor Householder.

Fungsi Vektor Householder untuk mendapatkan τ (suatu skalar) dan vektor *w* dengan *w*(1) = 1 sedemikian hingga $(I - \tau\, w\, w^T)x$ bernilai nol semua kecuali komponen pertamanya, untuk $x \in \Re^n$. Fungsi **sign(*a*)** mengembalikan nilai 1 bila *a*>0, sebaliknya mengembalikan nilai -1 bila *a*<0.

Algoritma 1. Vektor Householder

```
Fungsi  [w, τ] = vektorHouse(x, n)
begin
   #-- n = panjang verktor x
   w ← x; τ ← 0;
   s ← ||x||2;
   β ← sign(x(1))s
   if s ≠ 0
         φ ← x(1) + β;
         τ ← φ/s;
         w(2:n) ← w(2:n)/φ;
   endif
   w(1) = 1
   return [w, τ]
end.
```

Algoritma 2 adalah algoritma konvensional untuk menghitung faktorisasi QR. Algoritma konvensional ini banyak melakukan operasi perkalian matriks-vektor ($C \leftarrow \tau\, A^T w$) dan operasi perbaharuan *rank*-1 ($A \leftarrow A - wC^T$).

Algoritma 2. Faktorisasi QR (Konvensional)

```
Fungsi qrfact(m, n, A, τ)
begin
      for j=1:n
[w(j:m), τ(j)]←vektorHouse(A(j:m,j),m-
j+1);
C(j:n)←τ(j)* A^T(j:m,j:n)*w(j:m)
A(j:m,j:n) ← A(j:m,j:n)- w(j:m)*C^T(j:n)
          if j<m
         A(j+1:m,j) ← w(j+1:m)
          endif
          endfor
end.
```

Untuk menggunakan blok faktorisasi QR, suatu matriks $A \in R^{m \times n}$ dipartisi menjadi *N* blok kolom:

$$A = [A_1, A_2, \Lambda\ , A_N],$$

setiap blok A_i memiliki *r* kolom. (Jika *r* tidak dapat membagi *n* maka A_N memiliki jumlah kolom lebih kecil dari *r*). Berikut algoritma blok faktorisasi QR dengan bentuk Compact WY.

Algoritma 3. Blok Faktorisasi QR dengan Bentuk *Compact* WY

```
Fungsi qrfactCWY(m, n, A, τ)
  begin
     for k=1:N
    s ←(k-1)r+1; nb ← min(r,n-s+1)
        gunakan algoritma 1. untuk
    medapatkan
    vektor-vekor   Householder   dan
    vektor τ
        if (s+nb)<n
    [Y,T] ← genYT(A(s:m,s:s+nb-1),τ)
    A(s:m,s+nb:n)←(I+YTY^T)^T
    A(s:m,s+nb:n)
        endif
     endfor
    end.
```

Algoritma 4. Membentuk Matrik Y dan T

```
Fungsi [Y,T] = genYT(V, Tau)
   begin
     Y ← V(:,1); T ← [-Tau(j)]
   for j=2:r
    Z ← -Tau(j)TY^TV(:,j)
    Y ←[Y V(:,j)]; T ←
                      [ T      Z      ]
                      [ 0   -Tau(j)   ]
     endfor
     return [W, Y]
    end.
```

IMPLEMENTASI PARALELISASI BLOK FAKTORISASI QR

Struktur data yang digunakan untuk menyimpan matrik $A_{m\times n}$ adalah suatu larik berukuran *mn*. Anggota-anggota dari matriks *A* disimpan berurut dari kolom pertama, kedua, hingga kolom ke-*n* ke dalam suatu larik *a*. Sebagai contoh penyimpanan matriks

$$A_{4\times 3}=\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix}$$

dalam larik *a* adalah:

a:	a_{11}	a_{21}	a_{31}	a_{41}	a_{12}	a_{22}	a_{32}	a_{42}	a_{13}	a_{23}	a_{33}	a_{43}

Dengan menggunakan model struktur data ini, proses pemisahan, pembagian, dan pengumpulan data dapat dilakukan dengan sederhana. Selain itu, pendekatan ini juga mendukung prinsip lokalitas. Pengorganisasian memori secara hierarki menerapkan prinsip lokalitas dalam proses perpindahan data antar dua tingkatan memori yang berdekatan

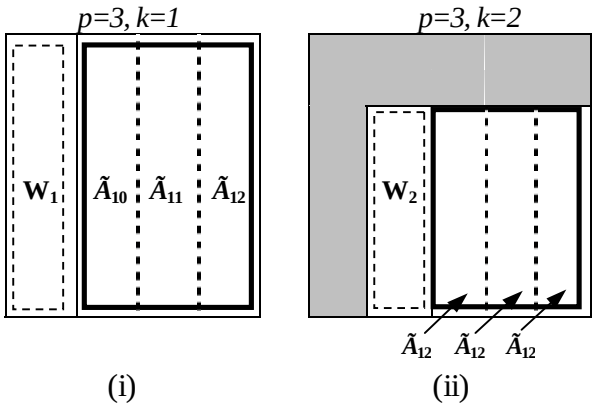
Algoritma Blok Faktorisasi QR melakukan tiga operasi: faktorisasi suatu blok kolom menggunakan algoritma QR konvensional, bangun suatu matriks *T* dan *update* suatu sisa matriks. Dari ketiga operasi ini, operasi *update* sisa matriks memiliki waktu komputasi terbesar. Oleh karena itu, operasi ini berpotensi untuk diparalelkan.

Algoritma 5. Paralel Blok Faktorisasi QR

```
if saya adalah prosesor 0
   for k=1:N
    s ← (k-1)r +1;   nb ← min(r,n-s+1)
    panggil qrfact (algoritma 2), untuk
mendapatkan
                vektor-vektor
          Householder dan vektor τ
    if(nb+s<n)
          kirim  vekor τ ke semua
slave
          kirim  vekor-vektor
          Householder ke semua slave
    kirim Ã_ki  ke prosesor i  untuk
i=1,2...p-1
          panggil GenYT
          update blok Ã_k0  dari A
dengan (I+YTY^T) Ã_k0
    terima dan letakkan blok update
Ã_ki dari slave
  endif
   endfor
else
      while masih ada blok Ã_ki
yang mau diupdate
      terima  tau, vektor-vektor
Householder, dan blok Ã_ki
    if semua data yang dibutuhkan
telah diterima
    panggil GenYT
          update blok Ã_ki  dengan
(I+YTY^T)  Ã_ki
    kirim blok Ã_ki yang telah
diupdate ke master
    endif
 endwhile
endif
```

Algoritma 5. merupakan versi paralel dari algoritma 3. Algoritma ini menerapkan paralelisasi pada operasi *update* suatu sisa matriks (lihat gambar 1), *r* menyatakan ukuran blok (atau banyak vektor kolom), *N* menyatakan banyak blok ($N=\frac{n}{r}$), dan *p* menyatakan banyak prosesor (setiap prosesor dikenali dengan nilai: 0, 1,.., *p*-1). $\tilde{A}_k$ adalah submatriks yang akan diperbaharui. Sedangkan $\tilde{A}_{ki}$ menyatakan bagian blok dari submatriks $\tilde{A}_k$ yang akan diperbaharui oleh prosesor *i*.

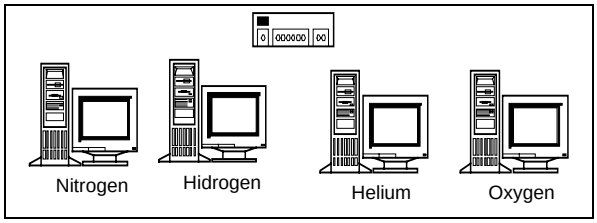
Gambar 2 (i) memperlihatkan partisi data yang terjadi untuk *p*=3, dan *k*=1. Matriks *W*1 berisi vektor-vektor Householder. Submatriks yang akan diperbaharui ($\tilde{A}_1$) berada dalam segi empat bergaris tebal. Perbaharuan untuk submatriks $\tilde{A}_{10}$ dilakukan oleh prosesor 0, submatriks $\tilde{A}_{11}$ dilakukan oleh prosesor 1, dan submatriks $\tilde{A}_{12}$ dilakukan oleh prosesor 2. Gambar 2 (ii) memperlihatkan partisi data yang terjadi untuk *k*=2.



Gambar 1. Partisi Sisa Matriks

Sistem memori tersebar multikomputer yang digunakan dalam penelitian dapat terlihat pada gambar

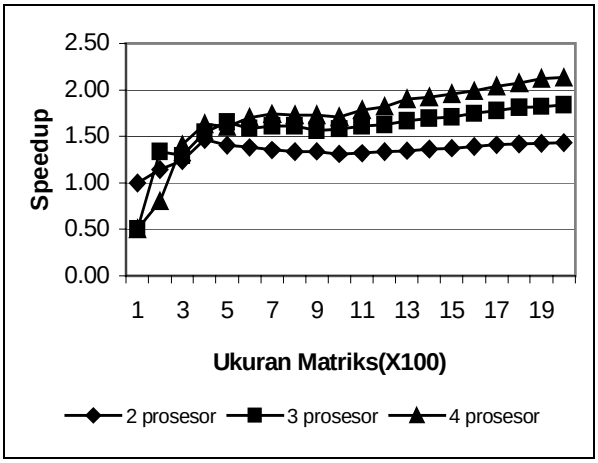
2. Setiap komputer terhubung dalam jaringan lokal (Ethernet, 10/100 Mbps) dan memiliki spesifikasi sama, yaitu prosesor Intel dengan *clock speed* 450 MHz, memori utama 512 Mb dan memori *cache* 512 Kb. Komputer Oxygen bertindak sebagai *network file server* (NFS). Sistem operasi jaringan yang digunakan adalah Linux Mandrake 6.0. Pustaka (*library*) pertukaran pesan yang digunakan adalah *Local Area Multicomputer*(LAM) MPI.



Gambar 2. Sistem Memori Tersebar Multikomputer

EVALUASI KINERJA

Waktu eksekusi dan speedup(S) dari fungsi PQRFACT (implementasi algoritma 5) untuk dua prosesor sampai empat prosesor atas matriks-matriks padat bujur sangkar dapat dilihat dalam Tabel 1 dan Gambar 3. Waktu eksekusi yang dipakai adalah waktu ekseksi terbaik dari hasil uji coba atas ukuran blok 10 s/d 200. Speedup tertinggi untuk dua prosesor 1.47, untuk tiga prosesor 1.84, dan untuk empat prosesor 2.13.

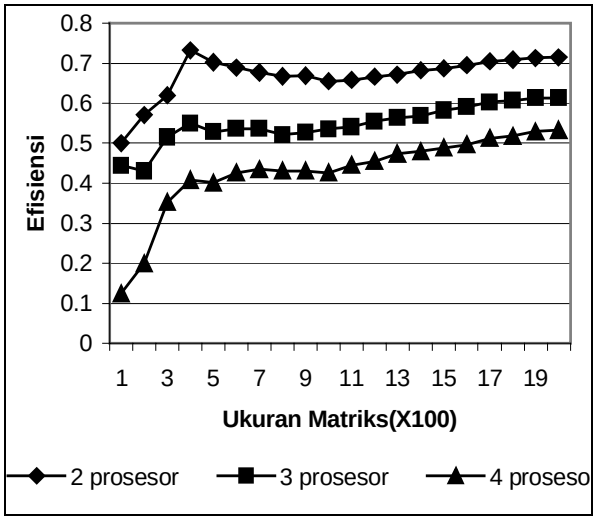


Gambar 3. Speedup

Gambar 4 memperlihatkan efisiensi untuk dua, tiga dan empat prosesor. Efisiensi terbesar untuk dua prosesor 70% , untuk tiga prosesor 61%, untuk empat prosesor 53%.

Tabel. 1. Waktu eksekusi dan speedup fungsi PQRFACT

m = n	**Satu prosesor**	**Dua prosesor**		**Tiga prosesor**		**Empat prosesor**	
	(detik)	(detik)	S	(detik)	S	(detik)	S
100	0.01	0.01	1.00	0.02	0.50	0.02	0.50
200	0.08	0.07	1.14	0.06	1.33	0.10	0.80
300	0.31	0.25	1.24	0.24	1.29	0.22	1.41
400	0.85	0.58	1.47	0.55	1.55	0.52	1.63
500	1.7	1.21	1.40	1.03	1.65	1.06	1.60
600	2.98	2.16	1.38	1.88	1.59	1.75	1.70
700	4.78	3.53	1.35	2.97	1.61	2.75	1.74
800	7.15	5.36	1.33	4.44	1.61	4.14	1.73
900	10.25	7.66	1.34	6.55	1.56	5.94	1.73
1000	14.14	10.79	1.31	8.95	1.58	8.29	1.71
1100	19.28	14.64	1.32	11.99	1.61	10.82	1.78
1200	25.98	19.48	1.33	16.00	1.62	14.27	1.82
1300	34.18	25.44	1.34	20.52	1.67	18.03	1.90
1400	44.01	32.26	1.36	26.00	1.69	22.91	1.92
1500	55.08	40.11	1.37	32.28	1.71	28.19	1.95
1600	67.82	48.80	1.39	38.81	1.75	34.13	1.99
1700	83.83	59.53	1.41	47.22	1.78	41.17	2.04
1800	101.09	71.34	1.42	55.89	1.81	48.76	2.07
1900	119.93	84.01	1.43	65.84	1.82	56.61	2.12
2000	140.47	98.24	1.43	76.37	1.84	65.91	2.13



Gambar 4. Efisiensi

KESIMPULAN

Algoritma blok Faktorisasi QR banyak melakukan operasi perkalian matriks-matriks. Bagian dari algoritma blok faktorisasi QR yang paling banyak memakan waktu adalah operasi *update* sisa matrik. Oleh karena itu, strategi paralelisasi sangat tepat dilakukan pada operasi *update* sisa matriks.

Uji coba program paralel blok faktorisasi QR terhadap matriks-matriks padat bujur sangkar berukuran n=m=100 s/d n=m=2000 dengan ukuran blok dari 10 s/d 200 pada sistem memori tersebar multikomputer dalam sistem operasi LINUX memperlihatkan peningkatan kinerja yang cukup baik. Speedup tertinggi untuk dua prosesor 1.47, untuk tiga prosesor 1.84, dan untuk empat prosesor 2.13. Efisiensi terbesar untuk dua prosesor 70%, untuk tiga prosesor 61%, dan untuk empat prosesor 53%.

DAFTAR PUSTAKA

1. Foster, I., *Designing and Building Parallel Programs*, Addison-Wesley Publishing Company, Inc., 1995
2. Golub, G. H., and Van Loan, C. F., *Matrix Computations*, The Johns Hopkins University Press, Sudbury, MA, USA, 1996.
3. Pacheco, P. S., *Parallel Programming With MPI*, Morgan Kaufmann Publishers, Inc., San Francisco, California, 1997
4. Wilkinson, B., Allen, M., *Parallel Programming Techniques and Applications Using Networked Workstations and Parallel Computer*, Prentice Hall, Upper Saddle River, N J, 2004.