

IMPLEMENTASI ALGORITME PARALLEL SVD DENGAN MENGUNAKAN METODE BLOCK-JACOBI DAN PENENTUAN SUBPROBLEM DINAMIS PADA LINGKUNGAN CLUSTER KOMPUTER

BASARUDDIN T¹, RIFKI SADIKIN²

¹Fakultas Ilmu Komputer Universitas Indonesia Jakarta, ²Jurusan Teknik Informatika Universitas Trisakti Jakarta
¹chan@cs.ui.ac.id, ²rsadikin@yahoo.com

ABSTRAK

Algoritme parallel untuk menghitung SVD sebuah matrik $A \in R^{m \times n}$ dengan menggunakan metode Jacobi dan penentuan subproblem dinamis diperkenalkan. Penentuan dinamis dilakukan berdasarkan informasi norm Frobernius pada subproblem. Algoritme SVD dengan metode Jacobi dan penjadwalan dinamik diimplementasikan dengan MPI dan dalam lingkungan kluster komputer. Percobaan dilakukan untuk memperoleh perbandingan unjuk kerja antara algoritme dinamis dan algoritme statis, dan speed up. Hasil percobaan menunjukkan bahwa kinerja algoritme parallel SVD dengan metode Jacobi dan penjadwalan dinamis lebih baik daripada algoritme statis, speed up pada matrik dense lebih baik daripada speed up pada matrik sparse (harwell-boeing).

1 PENDAHULUAN

Komputasi SVD merupakan komputasi yang penting dan sering dijumpai dalam persoalan aljabar linear. Algoritme serial untuk melakukan komputasi SVD memiliki kompleksitas $O(m \times n^2)$ untuk matrik dengan ukuran $m \times n$. Oleh karena itu, teknik parallel untuk menyelesaikan SVD diharapkan mengurangi waktu yang dibutuhkan. SVD sebuah matrik $A \in R^{m \times n}$ adalah

$$A = U \Sigma V^T \quad (1)$$

dengan $U \in R^{m \times m}$ dan $V \in R^{n \times n}$ adalah matrik orthogonal. Σ merupakan matrik diagonal. Elemen pada matrik diagonal Σ adalah nilai singular matrik [1].

Salah satu algoritme SVD adalah menggunakan metode Jacobi. Ide metode Jacobi adalah meminimalkan nilai elemen nondiagonal pada matrik A . Jika didefinisikan $F(A, I)$ sebagai

$$F(A, I) = \sum_{i=1}^n \sum_{j=1, j \neq i}^n A_{ij}^2 \quad (2)$$

Metode Jacobi secara terus menerus mengecilkan nilai $F(A, I)$ sehingga A menjadi diagonal.

2 ALGORITME PARALLEL DENGAN PEMILIHAN DINAMIS

Salah satu kelebihan metode Jacobi dalam melakukan perhitungan SVD adalah mudah diparallelkan. Apabila terdapat p prosesor maka sejumlah p iterasi dapat dilakukan secara bersamaan asalkan subproblem yang dilakukan tiap prosesor merupakan subproblem yang berbeda.

Algoritme SVD parallel dilakukan dengan strategi SIMD (Single Instruction Multiple Data). Algoritme yang sama dijalankan pada tiap prosesor yaitu algoritme yang melakukan perhitungan SVD untuk subproblem A_{ij} dan A_{ji} . Tiap prosesor melakukan perhitungan untuk matrik U, V dan memperharui nilai pada Matrik A untuk kolom ke- i dan kolom ke- j , kemudian mendistribusikan hasil yang diperoleh ke prosesor lain.

Seperti pada serial, di lingkungan parallel strategi dalam menetapkan pasangan indek dapat dilakukan secara statik atau secara Dinamis. Pemilihan Dinamis pada lingkungan parallel lebih sulit dibandingkan serial. Beberapa pasang indek harus ditetapkan sehingga pasangan-pasangan indek itu memiliki nilai paling optimal daripada kemungkinan pasangan indek lainnya. Permasalahan pemilihan beberapa pasanga indek optimal dapat dianalogikan dengan permasalahan *maximum-weight perfect matching problems* pada teori graf.

Permasalahan pemilihan pasangan indek dapat disederhanakan dari *Maximum-weight Perfect Matching* menjadi pencarian secara greedy. Algoritme greedy mengurangi biaya pencarian subset maksimum dengan harapan mendekati maksimum. Algoritme greedy dalam pencarian subset adalah dengan cara mengurutkan semua kemungkinan pasangan indek i dan j dengan $i < j$ berdasarkan nilai $\|A_{ij}\| F + \|A_{ji}\| F$ dalam urutan tidak menaik. Kemudian, dilakukan pindaian dari nilai yang paling tinggi ke paling rendah. Pasangan indek ditambahkan ke subset yang dicari apabila indek-indek dari pasangan indek yang sedang dipindai belum menjadi anggota subset. Algoritme greedy ini memiliki kompleksitas $O(p^2 \log(p))$ akibat pemindean dan pengurutan.

Berdasarkan pemilihan subproblem Dinamis, algoritme SVD paralel pada sejumlah p , dan tiap processor diberi label $me = 0, 1, \dots, p-1$ processor dapat dilakukan secara SIMD dengan menjalankan algoritme 1 pada masing-masing processor [2].

Algoritme 1 dijalankan pada tiap-tiap processor. Tiap processor mengerjakan 2 blok kolom yang berbeda. Blok kolom yang dikerjakan processor berbeda dengan blok kolom yang dikerjakan processor lain. Prosedur AllGather, dalam MPI adalah fungsi MPI Allgather [3], melakukan pengiriman hasil Matrik X (bagian dari Matrik $\cdot U$) yang dikerjakan ke processor ke semua processor lain. Tiap processor menerima matrik X dari semua processor lain disimpan di matrik sementara XX. Matrik XX digunakan untuk memperbaharui Matrik A ketika memperbaharui blok baris.

Algoritme 1

```

U = I_m, V = I_n
(i, j) = (2.me + 1, 2.me + 2)
while F(A, l) ≥ ε do
    if F(Sij, l) ≥ δ then
        SVD(Sij) → Xij, Yij
        for t = 1 to l do
            (Ait, Atj) = (Ait, Atj) · Yij
            (Uit, Utj) = (Uit, Utj) · Xij
            (Vit, Vtj) = (Vit, Vtj) · Yij
        end for
    else
        Xij = Im/p
    end if
    AllGather(Xij, i, j) →
    XX(t) = (Xrs, r, s), t = 0, 1, ..., p-1
    ▷ Perbaharui Blok Baris
    for t = 1 to p do
        ( Ari  Arj ) = Xrs,t}^T ( Ari  Arj )
        ( Asi  Asj ) = Xrs,t} ( Asi  Asj )
    end for
    update(W)
    ReOrderingComp(i, j, W, me) →
    dest1, dest2, tag1, tag2
    copy(Ai, Ui, Vi, i) → Ar, Ur, Vr, r
    copy(Aj, Uj, Vj, j) → As, Us, Vs, s
    send(Ar, Ur, Vr, r, dest1, tag1)
    send(As, Us, Vs, s, dest2, tag2)
    recv(Ai, Ui, Vi, i, 1)
    recv(Aj, Uj, Vj, j, 1)
end while
end
    
```

Norm tiap subproblem disimpan pada array W, setiap usai perhitungan SVD subproblem nilai W diperharui. Berdasarkan array W ditentukan subproblem yang akan dikerjakan oleh masing-masing processor. Setelan itu, blok kolom yang dikerjakan oleh processor dikirim ke processor lain sesuai dengan reordering dan menerima blok kolom dari processor lain yang merupakan blok kolom sweep berikutnya. Proses pengiriman dan penerimaan harus dilakukan secara nonbloking agar menghindari deadlock. Kompleksitas algoritme 1 dapat dihitung berdasarkan proses perkalian matrik, perhitungan SVD pada subproblem dan ongkos untuk reordering. Apabila matrik A merupakan matrik bujur sangkar berukuran n dan dibuat matrik blok berukuran $l \times l$ dengan $l = 2 \cdot p$ maka kompleksitas komputasi dengan greedy dynamic reordering adalah :

$$TIME = np^2 \left[c_1 \frac{n^3}{p^3} + 4c_2 \frac{n^2}{p^2} + c_3 \frac{n^2}{p} + p^4 \right] \quad (3)$$

3 PERCOBAAN DAN HASIL PERCOBAAN

Algoritme 1 diimplementasikan dengan MPI pada lingkungan parallel cluster dengan spesifikasi tiap komputer adalah : processor inter pentium 450 MHz, RAM sebesar 512 MB dan jaringan menggunakan Fast Ethernet yang dihubungkan dalam LAN. Prosedur dgemm di BLAS digunakan untuk perkalian Matrik. Sedangkan, untuk melakukan SVD pada subproblem digunakan prosedur dgesvd.

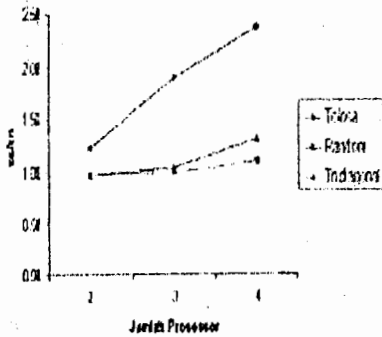
Percobaan dilakukan pada 3 jenis Matrik yaitu Matrik umum, Matrik sparse dan Matrik tridiagonal simetri. Matrik umum dibangkitkan dari prosedur yang ada di LAPACK yaitu dlatmr. Matrik sparse disebut matrik tolosa yang merupakan data dari industri yaitu Harwell-Boeing. Matrik tridiagonal simetri diperoleh dari dikritisasi persamaan differensial $d^2/dx^2 + x$.

Percobaan dilakukan dengan dimensi matrik 200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, dan 2000. Jumlah processor untuk percobaan algoritme paralel adalah 2, 3, dan 4. Toleransi yang digunakan adalah 10^{-10} untuk menetapkan ϵ dan δ . Seluruh komputasi dilakukan dengan akurasi presisi double real dengan presisi mesin $\epsilon_M = 1.11 \times 10^{-16}$.

3.1 Perbandingan antara Algoritme Statis dan Algoritme Dinamis

Perbandingan unjuk kerja antara algoritme dinamis dan statis dapat ditunjukkan dengan ratio antara waktu komputasi yang dibutuhkan oleh algoritme statis (ts) dan waktu komputasi yang dibutuhkan oleh algoritme dinamis (td). Gambar 1. menunjukkan trend kenaikan

rasio ts/td seiring dengan jumlah processor yang dipakai dalam parallel. Hal ini disebabkan jumlah kemungkinan pasangan indek sebanding dengan jumlah processor yang dipakai.

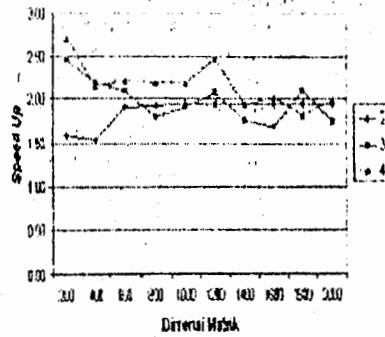


GAMBAR 1: RASIO ts/td BERBANDING DENGAN JUMLAH PROSESSOR

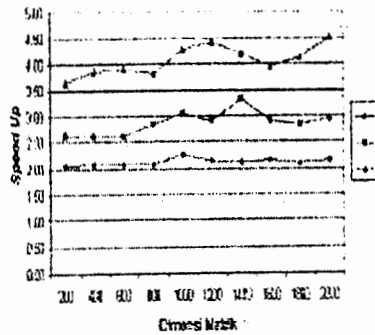
3.2 SPEED UP

Speed Up dihitung berdasarkan perbandingan antara waktu terbaik algoritme sekuensial dan waktu yang dibutuhkan dengan algoritme parallel. Algoritme SVD Blok-Jacobi 2 sisi dengan penjadwalan dinamis dijalankan pada processor tunggal dengan blocking factor : 4,6, dan 8. Waktu yang dibutuhkan untuk Blocking factor 4 digunakan untuk pembandingan dengan jumlah processor 2, 6 untuk jumlah processor 3 dan 8 untuk jumlah processor 4.

Gambar 2 menunjukkan speed up Matrik Tolosa tidak bertambah secara signifikan seiring dengan jumlah processor yang dipakai. Riciannya adalah sebagai berikut : Pada 2 processor rata-rata speed up adalah 1.87, dengan rentang speed up antara 1.54 sampai dengan 1.95. Speed up cenderung stabil untuk dimensi matrik ≥ 600 . Trend berbeda ditunjukkan oleh jumlah processor 3. Rata-rata speed up pada 3 processor adalah 1.99 dan dalam rentang antara 1.69 sampai dengan 2.45. Speed up pada 3 processor menurun seiring dengan dimensi matrik dan stabil pada kisaran 2.00. Hal yang sama terjadi pada 4 processor dengan rata-rata 2.11 dan dalam rentang 1.94 - 2.72. Speed up cenderung menurun seiring dengan dimensi matrik dan stabil untuk dimensi matrik ≥ 1400 . Speed up pada matrik Tolosa tidak bertambah secara signifikan dibandingkan jumlah processor disebabkan oleh nilai norm blok matrik Tolosa.



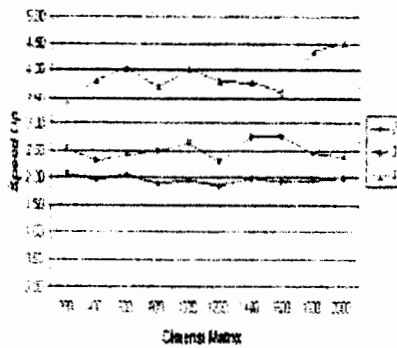
GAMBAR 2: SPEED UP ALGORITME BLOK SVD JACOBI PARALLEL DINAMIS untuk MATRIK TOLOSA



GAMBAR 3: SPEED UP ALGORITME BLOK SVD JACOBI PARALLEL DINAMIS UNTUK MATRIK RANDOM

Gambar 3 menunjukkan Speed Up untuk Matrik Random bertambah sebanding dengan pertambahan processor. Speed Up telah diperoleh pada jumlah processor 2,3 dan 4 dan pada dimensi matrik ≥ 200 . Rata-rata speed up dengan 2 processor adalah 2.14 dan dalam rentang 2.08-2.29. Pada jumlah processor 3 diperoleh speedup yang lebih tinggi daripada dengan 2 processor yaitu 2.88 dan dalam rentang 2.60-3.32. Rata-rata speed up pada jumlah processor 4 adalah tertinggi yaitu : 4.11 dan dalam rentang 3.65-4.50.

Gambar 4 menunjukkan Speed Up untuk Matrik Tridiagonal. Trend speed up matrik tridiagonal cenderung sama dengan trend speed up matrik random. Speed up bertambah seiring dengan jumlah processor yang digunakan. Pada jumlah processor 2 rata rata speed up adalah 1.97, dalam rentang 1.94-2.07. Pada jumlah processor 3 rata-rata speed up adalah 2.51, dalam rentang 2.32-2.77. Speed up tertinggi diperoleh pada jumlah processor 4 yaitu dengan rata-rata 3.97, dalam rentang 3.41-4.50.



GAMBAR 4: SPEED UP ALGORITME BLOK SVD JACOBI PARALLEL DINAMIS UNTUK MATRIK DIAGONAL.

4 KESIMPULAN

Algoritme SVD blok-Jacobi 2 sisi dengan penjadwalan dinamik dapat diimplementasikan dalam lingkungan komputer cluster paralel. Kinerja yang ditunjukkan oleh algoritme paralel lebih baik dibanding dengan sekuensial maupun algoritme paralel dengan penjadwalan statis.

Speed up dan perbandingan unjuk kerja algoritme dinamis dan algoritme statis tergantung dari kondisi norm blok matrik. Hal ini ditunjukkan oleh matrik tolosa yang memiliki norm blok matrik sangat tidak merata. Matrik Tolosa memiliki speed up yang tidak tergantung dengan jumlah prosessor dan kinerja algoritme dinamis jauh lebih baik daripada kinerja algoritme statis. Hal yang sebaliknya ditunjukkan oleh matrik Random dan matrik Tridiagonal.

PUSTAKA

- [1] Golub, G. H. and Loan, C. F. V. *Matrix Computation*. The Johns Hopkins University Press, Baltimore, Maryland 21211. 1989
- [2] Becka, M., Oksa, G., and Vajtersic, M.. Dynamic ordering for a parallel block-jacobi svd algorithm. *Journal of Parallel Computing*, 2002, 28:243-262.
- [3] PACS . *Introduction to MPI*. NCSA Access, Illinois, 2001