On Speeding-up Parallel Jacobi Iterations for SVDs

Soumitra Pal Sudipta Pathak Sanguthevar Rajasekaran
Computer Science and Engineering, University of Connecticut
371 Fairfield Road, Storrs, CT 06269, USA
{mitra@,sudipta.pathak@,rajasek@engr.}uconn.edu

***Abstract*—We live in an era of big data and the analysis of these data is becoming a bottleneck in many domains including biology and the internet. To make these analyses feasible in practice, we need efficient data reduction algorithms. The Singular Value Decomposition (SVD) is a data reduction technique that has been used in many different applications. For example, SVDs have been extensively used in text analysis. Several sequential algorithms have been developed for the computation of SVDs. The best known sequential algorithms take cubic time which may not be acceptable in practice. As a result, many parallel algorithms have been proposed in the literature. There are two kinds of algorithms for SVD, namely, QR decomposition and Jacobi iterations. Researchers have found out that even though QR is sequentially faster than Jacobi iterations, QR is difficult to parallelize. As a result, most of the parallel algorithms in the literature are based on Jacobi iterations. JRS is an algorithm that has been shown to be very effective in parallel. JRS is a relaxation of the classical Jacobi algorithm. In this paper we propose a novel variant of the classical Jacobi algorithm that is more efficient than the JRS algorithm. Our experimental results confirm this assertion. We also provide a convergence proof for our new algorithm. We show how to efficiently implement our algorithm on such parallel models as the PRAM and the mesh.**

***Keywords*-SVD; Jacobi iterations; JRS; parallel algorithms**

I. Introduction

Singular Value Decomposition (SVD) is a fundamental computational problem in linear algebra and it has application in various computational science and engineering areas. For example, it is widely used in areas such as statistics where it is directly related to principal component analysis, in signal processing and pattern recognition as an essential filtering tool, and in control systems. Recently, it is used as one of the fundamental steps in many machine learning applications such as least square regressions, information retrieval and so on. With the advent of BigData, it has become essential to process data matrices with thousands of rows and columns in real time. The SVD is one of the data reduction techniques. Hence, there is a strong need for efficient sequential and parallel algorithms for the SVD.

SVD takes as input a matrix $A \in \mathbb{F}^{m\times n}$ where $\mathbb{F}$ could be the field of real ($\mathbb{R}$) or complex ($\mathbb{C}$) numbers and outputs three matrices U, S, V such that $A = USV^T$, where $U \in \mathbb{F}^{m\times m}$, $V \in \mathbb{F}^{n\times n}$ are orthogonal matrices (i.e. $U^TU = I_m$, $V^TV = I_n$) and $S \in \mathbb{R}^{m\times n}$ is a diagonal matrix. If $S = diag(\sigma_1, \sigma_2, ...\sigma_{\min\{m,n\}})$, then the diagonal elements σ_is are called the singular values of A. The columns of U and V are referred to as the left and right singular vectors respectively. Without loss of generality, we assume that $m \geq n$.

There are various methods of computing the SVD [1]. The most commonly used algorithms for dense matrices, which we consider in this paper, can be classified as *QR-based* and *Jacobi-based*.

A. Two-sided Jacobi Algorithm

Among all algorithms for finding SVD Jacobi iteration based algorithms are most widely used in parallel settings since they are easier to parallelize than others. There are two types of Jacobi algorithms for SVD, namely Two-sided Jacobi SVD algorithm and One-sided Jacobi SVD algorithm. Two-sided Jacobi SVD algorithm is applicable to symmetric matrices only. It applies a series of *Jacobi Rotations* on input matrix A to diagonalize it.

$$A_{k+1} = U_k A_k U_k^T, k = 1, 2, ...$$

where $U_k = U_k(i, j, \phi_{ij}^k)$ represents a rotation of (i, j)-plane.

$$u_{ii}^k = u_{jj}^k = c^k = \cos(\phi_{ij}^k) \tag{1}$$

$$u_{ij}^k = -u_{ji}^k = s^k = \sin(\phi_{ij}^k) \tag{2}$$

ϕ^k is chosen such that

$$b_{ij}^{k+1} = b_{ji}^{k+1} = 0, \tag{3}$$

$$\text{or}\tan(2\phi_{ij}^k) = \frac{2b_{ij}^k}{b_{ii}^k - b_{jj}^k} \tag{4}$$

$$\text{and where}|\phi_{ij}^k| \leq \frac{1}{4}\pi \tag{5}$$

c^k is computed by

$$c^k = \frac{1}{\sqrt{1+t_k^2}} \text{and} s^k = c^k t^k \tag{6}$$

where$t^k$is the smaller root (in magnitude) of the quadratic equation give

(7)

$$t^{k^2} + 2\alpha^k t^k - 1 = 0, \alpha^k = \cot(2\phi_{ij}^k) \tag{8}$$

$$\text{hence}, t^k = \frac{sign(\alpha^k)}{|\alpha^k| + \sqrt{1+\alpha^{k^2}}} \tag{9}$$

B_{k+1} is identical to B_k other than the rows and columns i and j and the modified elements are given by

$$b_{ii}^{k+1} = b_{ii}^k + t^k b_{ij}^k \tag{10}$$
$$b_{jj}^{k+1} = b_j j^k - t^k b_{ij}^k \tag{11}$$
$$b_{ix}^{k+1} = c^k b_{ix}^k + s_{jr}^k \tag{12}$$
$$b_{jx}^{k+1} = -s_k b_{ix}^k + c_k b_{jx}^k \tag{13}$$
$$x \neq i, j \tag{14}$$

The above algorithm ensures that with every rotation A_k approaches the diagonal matrix S = diag$(\sigma_1, \sigma_2, ...\sigma_n)$ and $(U_k.....U_2U_1)^T$ approaches a matrix whose $j-$th column is the eigenvector corresponding to σ_j. [2] shows that each sweepcauses the off-diagonal norm to decrase and evetually the algorithm converges. According to the original Jacobi scheme one should search for the largest off-diagonal element to eliminate at every rotation. However, this scheme is extremely time consuming in practice. Hence, a common scheme used is to choose elements (i, j) to be annihilated in cyclic order $(1,2),(1,3),(1,4),.....(1,n),(2,3),.....(2,n),.....,(n-1,n)$. This is known as cyclic jacobi algorithm. [3]–[5] show that cyclic Jacobi algorithm has a quadratic convergence rate and argues that convergence ususally occurs within 6 to 10 sweeps or $3n^2$ to $5n^2$ Jacobi rotations.

B. One-sided Jacobi Algorithm

A more general and more widely used algorithm is One-Sided Jacobi SVD algorithm that deals with any matrix. One-Side Jacobi SVD algorithm finds an orthogonal matrix $V \in \mathbb{R}^{n\times n}$, such that $B = (b_1, b_2, ..., b_n) = AV$ where $B \in \mathbb{R}^{m\times n}$ is an orthogonal matrix. The above statement implies that:

$$b_i^T b_j = \sigma_i^2 \delta_{ij}$$

where

$$\delta_{ij} = \begin{cases} 0, & \text{for } i \neq j \\ 1, & \text{for } i = j \end{cases}$$

The quantities σ_i are called *Singular Values* of matrix A. B can be written as $B = US$ where $U = (u_1, u_2, ..., u_k)$such that $u_j = \frac{b_j}{\sigma_j}$ for $\sigma_j \neq 0$ and $U^TU = I \in \mathbb{R}^{n\times n}$. Hence we can now write $B = AV$ as $US = AV$ or using properties of orthogonality $A = USV^T$. If some of the σ_js are zeros singular values, the corresponding columns of U are set to null vectors. The first k singular values are always chosen to be non-zero causing the matrix U^TU to be unit matrix of order k augmented to order n with zeros. This is written as follows :

$$S = \begin{pmatrix} \mathbf{1}_k & \\ & \mathbf{0}_{n-k} \end{pmatrix} \tag{15}$$

The matrix V used for orthogonalization of A is a product of matrices indexed by $V^{(k)}$.

$$V = \prod_{k=1}^{z} V^{(k)}$$

V is constructed in such a way that

$$(a_i, a_j) \begin{bmatrix} c & -s \\ s & c \end{bmatrix} = (a_i^{k+1}, a_j^{k+1}) \tag{16}$$

, $i < j$ and $||a_i^{k+1}||_2 > ||a_j^{k+1}||_2$, here a_i is the $i-$th column of A. This is achieved by choosing
$c = [\frac{\beta+\gamma}{2\gamma}]^{\frac{1}{2}}$ and $s = [\frac{\alpha}{2\gamma c}]$, $s = [\frac{\gamma-\beta}{2\gamma}]^{\frac{1}{2}}$ and $c = [\frac{\alpha}{2\gamma s}]$
where $\alpha = 2a_i^T a_j$, $\beta = ||a_j||_2^2$ and $\gamma = (\alpha^2 + \beta^2)^{\frac{1}{2}}$
The matrices given by $V^{(k)}$ are called rotation matrices and z is not dependent of dimension of A.

Describe one sided Jacobi with math notations from Dr Rajs paper:

C. Block Householder Jacobi

For $A \in \mathbb{R}^{m\times n}$ where $m >> n$, a common practice is to reduce the problem complexity by applying initial orthogonal factorization on A followed by one-sided Jacobi method. Given A we apply the following orthogonal facotirzation given by $A = QR$, where $Q \in \mathbb{R}^{m\times n}$ is an orthogonal matrix and $R \in \mathbb{R}^{n\times n}$ is an upper triangular matrix. Next, One-sided Jacobi SVD algorithm is applied on R. The Block Householder- Jacobi SVD algorithm is presented below:
Step 1: Factorize A by $A = QR$ where $Q \in \mathbb{R}^{m\times n}$ with orthonormal columns and $R \in \mathbb{R}^{n\times n}$ is upper triangular.
Step 2: Determine SVD of the upper triangular matrix using One-sided Jacobi algorithm.

$$R = \hat{U} \begin{bmatrix} \hat{S} \\ 0 \end{bmatrix} V^T \tag{17}$$

where $\hat{U} \in \mathbb{R}n \times p$ and $V \in \mathbb{R}n \times p$ orthogonal matrices and $\hat{S} = \text{diag}(\sigma_i)$ is a diagonal matrix containing p non-zero singular values of A.
Step 3: Compute $U = Q\hat{U}$. Left singular vectors of A are given by u_i where u_i are columns of U.

[6]–[8] introduced and implemented the idea of parallel one-sided block jacobi algorithm with dynamic ordering and variable blocking. Their approach, known as OSBJ method, accomplishes the parallel SVD in three stages namely, pre-processing, iteration and post processing. There are two types of pre-processing depending on the dimension of the matrix, namely QR-pre-processing and LQ pre-processing. During QR-pre-processing OSBJ decomposes input matrix $A = Q_1R$ where $A \in \mathbb{R}^{m\times n}$, orthonormal matrix $Q_1 \in \mathbb{R}^{m\times m}$ and upper triangular matrix $R \in \mathbb{R}^{n\times n}$. On the contrary, LQ-pre-processing decomposes input matrix $A = LQ_2$ where $L \in \mathbb{R}^{n\times n}$ is a lower triangular matrix and $Q_2 \in \mathbb{R}^{\kappa\times\kappa}$ is an orthogonal matrix. The L or R

matrix in pre-processing stage is considered as input A^0 for the iteration stage. A^0 is partitioned into column blocks. Let, $A_i^{(r)}$ and $A_j^{(r)}$ are one such column block pair where $1 \leq i < j \leq l$ assuming we have l such column blocks. Weight $\hat{w}_{ij}^r = \frac{||A_i^{(r)^T} A_j^{(r)} \mathbf{e}||_2}{||\mathbf{e}||_2}$, where $\mathbf{e} = (1, 1, ...1)^T \in \mathbb{R}^{\frac{n}{l}}$. $\hat{w}_{ij}^r$ serves as an approximate measure of inclination between $Im(A_i^r)$ and $Im(A_j^r)$. The column block pairs are ordered based on their mutual inclination and the most mutually inclined pair is chosen to be orthogonalized and those columns are excluded from the set for next iteration. The process is repeated until all the column blocks are used up. For each column block pair a 2×2 Gram matrix $G_s \in \mathbb{R}^{\frac{2n}{l} \times \frac{2n}{l}}$ is constructed as is given by $G_s = [A_{i_{r,s}}^{(r)}, A_{j_{r,s}}^{(r)}]^T [A_{i_{r,s}}^{(r)}, A_{j_{r,s}}^{(r)}]$ where $1 \leq s \leq \frac{l}{2}$. This gram matrix is them diagonalized as $G_s = V_s D_s V_s^T$ where V_s is an orthogonal matrix and D_s is a diagonal matrix. The column blocks for next iteration $A^{(r+1)}$ is computed by $[A_{i_{r,s}}^{(r+1)}, A_{j_{r,s}}^{(r+1)}] = [A_{i_{r,s}}^{(r)}, A_{j_{r,s}}^{(r)}] V_s$. Iteration process continues until all the columns are mutually orthogonal.

The Jacobi Relaxation Scheme (JRS) algorithm by Rajasekaran and Song introduced the idea of improving parallelism in SVD computation by multiplying the off-diagonal element in each iteration by a very small number ϵ such that $0 < \epsilon < 1$ instead of setting the off-diagonal element to zero. [9] compares number of iterations taken by JRS algorithm to converge with that of Strumpen's Independent Jacobi algorithm [**?**] and shows that JRS takes much less number of iterations to converge than Independent Jacobi.

This paper has done same parallel implementation: [10].

This paper seems to do some kind of sorting for [11].

D. Contributions

Subsection text here.

II. Quick Pivoting

Wherever Times is specified, Times Roman or Times New Roman may be used. If neither is available on your system, please use the font closest in appearance to Times. Avoid using bit-mapped fonts if possible. True-Type 1 or Open Type fonts are preferred. Please embed symbol fonts, as well, for math, etc.

III. Parallel Implementation

Wherever Times is specified, Times Roman or Times New Roman may be used. If neither is available on your system, please use the font closest in appearance to Times. Avoid using bit-mapped fonts if possible. True-Type 1 or Open Type fonts are preferred. Please embed symbol fonts, as well, for math, etc.

IV. Experimental Results

V. Conclusion

The conclusion goes here. this is more of the conclusion

Acknowledgment

The authors would like to thank... more thanks here

References

[1] G. H. Golub and C. F. Van Loan, *Matrix computations*. Johns Hopkins University Press, 2012, vol. 3.

[2] F. T. Luk and H. Park, "A proof of convergence for two parallel jacobi svd algorithms," *Computers, IEEE Transactions on*, vol. 38, no. 6, pp. 806–811, 1989.

[3] G. E. Forsythe and P. Henrici, "The cyclic jacobi method for computing the principal values of a complex matrix," *Transactions of the American Mathematical Society*, vol. 94, no. 1, pp. 1–23, 1960.

[4] A. Schönhage, "Zur konvergenz des jacobi-verfahrens," *Numerische Mathematik*, vol. 3, no. 1, pp. 374–380, 1961.

[5] J. H. Wilkinson, "Note on the quadratic convergence of the cyclic jacobi process," *Numerische Mathematik*, vol. 4, no. 1, pp. 296–300, 1962.

[6] M. Bečka and G. Okša, "Parallel one–sided jacobi svd algorithm with variable blocking factor," in *Parallel Processing and Applied Mathematics*. Springer, 2013, pp. 57–66.

[7] S. Kudo, Y. Yamamoto, M. Becka, and M. Vajteršic, "Parallel one-sided block jacobi svd algorithm with dynamic ordering and variable blocking: Performance analysis and optimization," 2016.

[8] M. Becka, G. Okša, and M. Vajteršic, "Parallel Code for One-sided Jacobi-Method," 2015.

[9] S. Rajasekaran and M. Song, "A relaxation scheme for increasing the parallelism in Jacobi-SVD," *Journal of Parallel and Distributed Computing*, vol. 68, no. 6, pp. 769–777, 2008.

[10] M. I. Soliman, "Memory Hierarchy Exploration for Accelerating the Parallel Computation of SVDs," *Neural, Parallel Sci. Comput.*, vol. 16, no. 4, pp. 543–561, Dec. 2008.

[11] B. B. Zhou and R. P. Brent, "On parallel implementation of the one-sided jacobi algorithm for singular value decompositions," in *Parallel and Distributed Processing, 1995. Proceedings. Euromicro Workshop on*. IEEE, 1995, pp. 401–408.