社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－MPS－60 (12)
2006／6／26

Preliminary Result of Parallel double Divide and Conquer

Taro Konda†, Hiroaki Tsuboi†‡, Masami Takata*, Masashi Iwasaki†‡
and Yoshimasa Nakamura†‡
† Department of Applied Mathematics and Physics,
Graduate School of Informatics,
Kyoto University
Yoshida Honmachi, Sakyo-ku, Kyoto, JAPAN
‡ SORST, JST
⋆ Graduate School of Humanity and Science,
Nara Women's University

Abstract *This paper shows a concept for parallelization of double Divide and Conquer and its preliminary result. For singular value decomposition, double Divide and Conquer was recently proposed. It first computes singular values by a compact version of Divide and Conquer. The corresponding singular vectors are then computed by twisted factorization. The speed and accuracy of double Divide and Conquer are as well or even better than standard algorithms such as QR and Divide and Conquer. In addition, it is expected that double Divide and Conquer has great parallelism because each step is theoretically parallel and heavy communication is no required. However, any parallel model of double Divide and Conquer has not been studied yet. In this paper, policy of the parallelization is discussed. Then, a parallel implementation with MPI is tested on a distributed memory parallel computer. It successfully shows a high parallelism.*

1 Introduction

A new framework of Singular value decomposition (SVD) was recently developed. It first computes singular values and the corresponding singular vectors are then computed by twisted factorization. MR^3[1, 2, 3] and I-SVD[4, 5, 6] are algorithms in this manner. They achieve successful speed-up compared to past standard algorithm such as QR, although further study about numerical accuracy of twisted factorization is desired.

Interest in parallelism of numerical algorithm is growing due to increased access to parallel computers coupled by the necessity to process growing data size. The parallelism of these new algorithms seems to be excellent because each piece of twisted factorization is parallel executable. However, the total parallelism is practically limited by seriality in the section of singular value computation[7].

A new algorithm, double Divide and Conquer(dDC)[7, 8], improves the parallelism of algorithm with the help of twisted factorization. It adopts Divide and Conquer(D&C) to parallelize the section of singular value computation. It is as fast and accurate as I-SVD and high parallelism is expected for any type of matrix. The focus of this paper is to evaluate preliminarily a parallel version of dDC to see a high potential of parallelism.

2 double Divide and Conquer

Here, we introduce double Divide and Conquer(dDC) for singular value decomposition. It first computes singular values by a "compact" Divide and Conquer described in section 3.1. We here call this compact version *Singular Value oriented Divide and Conquer (SVDC)*. Then, it computes the corresponding singular vectors by twisted factorization described in section 3.2. dDC has the following features.

1. *Speed.* Complexity of D&C is ranged from $O(n^2)$ to $O(n^3)$ due to the frequency of deflations. In contrast, SVDC is dramatically down-sized and achieves stability because it skips heavy update of vectors in D&C. Also, twisted factorization costs $O(n^2)$, too. Thus complexity of dDC is $O(n^2)$.

2. *Accuracy.* Singular values are computed with high accuracy particularly, because dDC inherits good performance of D&C. However, dDC has a theoretical issue to be discussed. Although dDC computes singular values in *absolutely* high precision, twisted factorization assumes that singular values are computed in *relatively* high precision. It may cause a problem when matrix has a tiny singular value. Further investigation should be taken.

3. *Parallelism.* Both SVDC and twisted factorization have essentially good parallelism.

4. *Memory space.* D&C requires $O(n^2)$ memory space to hold SVD of submatrices temporarily. In contrast, dDC needs $O(n)$ memory space because of a few vectors to be at hand. Therefore, we can use dDC to solve various problems with huge matrix.

5. *Partial computation of singular values and singular vectors.* Twisted factorization computes selected vectors according to need because the computations are independent each other. However, mdLVs of I-SVD has to finish the computation of all singular values to find the targeted values. In contrast, SVDC of dDC can specially compute the targets, selecting secular equation to be solved.

2.1 Divide and Conquer to Compute Singular Values

Given an $n \times (n+1)$ upper bidiagonal matrix

$$B = \begin{pmatrix} b_1 & b_2 & & & \\ & b_3 & \ddots & & \\ & & \ddots & b_{2n-2} & \\ & & & b_{2n-1} & b_{2n} \end{pmatrix}, \qquad (1)$$

its SVD is

$$B = U \begin{pmatrix} \Sigma & 0 \end{pmatrix} V^T, \qquad (2)$$

where U is an $n \times n$ orthogonal matrix whose columns are left singular vectors. V is an $(n+1) \times (n+1)$ orthogonal matrix whose columns are right singular vectors. Σ is an $n \times n$ nonnegative definite diagonal matrix and 0 is a column of zero elements.

The $n \times (n+1)$ upper bidiagonal matrix B is partitioned into two submatrices as

$$B = \begin{pmatrix} B_1 & 0 \\ b_{2k-1}\mathrm{e}_k{}^T & b_{2k}\mathrm{e}_1{}^T \\ 0 & B_2 \end{pmatrix} \qquad (3)$$

for a fixed k such that $1 < k < n$, where B_1 is a $(k-1) \times k$ lower bidiagonal matrix, B_2 is an $(n-k) \times (n-k+1)$ upper bidiagonal matrix and e_j is the j-th unit vector of appropriate dimension. The parameter k is usually taken to be $\lfloor n/2 \rfloor$.

Now, suppose that SVD of the B_i is given by

$$B_i = U_i \, (D_i \; 0) \, (V_i \; \mathrm{v}_i)^T. \qquad (4)$$

Let l_1 be the last row of V_1, ψ_1 be the last element of v_1, f_2 be the first row of V_2 and ϕ_2 be the first element of v_2. By substituting (4) into (3), we then obtain

$$B = \begin{pmatrix} 0 & U_1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & U_2 \end{pmatrix} \begin{pmatrix} b_{2k-1}\psi_1 & b_{2k-1}\mathrm{l}_1 & b_{2k}\mathrm{f}_2 & b_{2k}\phi_2 \\ 0 & D_1 & 0 & 0 \\ 0 & 0 & D_2 & 0 \end{pmatrix} \times \begin{pmatrix} \mathrm{v}_1 & V_1 & 0 & 0 \\ 0 & 0 & V_2 & \mathrm{v}_2 \end{pmatrix}^T. \qquad (5)$$

If Givens rotation is applied to make $b_{2k}\phi_2$ zero, then we get

$$B = \tilde{U} \begin{pmatrix} M & 0 \end{pmatrix} \begin{pmatrix} \tilde{V} & \tilde{\mathrm{v}} \end{pmatrix}^T, \qquad (6)$$

where

$$\begin{aligned} \tilde{U} &\equiv \begin{pmatrix} 0 & U_1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & U_2 \end{pmatrix}, \\ M &\equiv \begin{pmatrix} r_0 & b_{2k-1}\mathrm{l}_1 & b_{2k}\mathrm{f}_2 \\ 0 & D_1 & 0 \\ 0 & 0 & D_2 \end{pmatrix}, \\ \tilde{V} &\equiv \begin{pmatrix} c_0\mathrm{v}_1 & V_1 & 0 \\ s_0\mathrm{v}_2 & 0 & V_2 \end{pmatrix}, \quad \tilde{\mathrm{v}} \equiv \begin{pmatrix} -s_0\mathrm{v}_1 \\ c_0\mathrm{v}_2 \end{pmatrix}, \\ r_0 &= \sqrt{(b_{2k-1}\psi_1)^2 + (b_{2k}\phi_2)^2}, \\ c_0 &= \frac{b_{2k-1}\psi_1}{r_0}, \quad s_0 = \frac{b_{2k}\phi_2}{r_0}. \end{aligned} \qquad (7)$$

Thus the matrix B is reduced to $\begin{pmatrix} M & 0 \end{pmatrix}$ by the orthogonal transformations $\tilde{U}$ and $\begin{pmatrix} \tilde{V} & \tilde{\mathrm{v}} \end{pmatrix}$.

The above D&C process can be simplified when only singular values are desired. From (6) and (7), B is written as

$$\begin{aligned} B &= \tilde{U} \begin{pmatrix} M & 0 \end{pmatrix} \begin{pmatrix} \tilde{V} & \tilde{\mathrm{v}} \end{pmatrix}^T \\ &= \tilde{U} \begin{pmatrix} U_M \Sigma V_M^T & 0 \end{pmatrix} \begin{pmatrix} \tilde{V} & \tilde{\mathrm{v}} \end{pmatrix}^T \\ &= U\Sigma \begin{pmatrix} \tilde{V}V_M & \tilde{\mathrm{v}} \end{pmatrix}^T \\ &= U\Sigma \left(\begin{pmatrix} c_0\mathrm{v}_1 & V_1 & 0 \\ s_0\mathrm{v}_2 & 0 & V_2 \end{pmatrix} V_M \quad \begin{pmatrix} -s_0\mathrm{v}_1 \\ c_0\mathrm{v}_2 \end{pmatrix} \right)^T, \end{aligned} \qquad (8)$$

thus

$$\begin{aligned} f &= \begin{pmatrix} c_o\phi_1 & f_1 & 0 \end{pmatrix} V_M, \\ l &= \begin{pmatrix} s_o\psi_2 & 0 & l_2 \end{pmatrix} V_M, \\ \phi &= -s_0\phi_1, \quad \psi = c_0\psi_2, \end{aligned} \qquad (9)$$

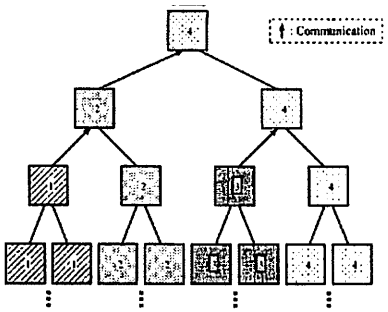

Figure 1: Parallel model with less communication ($P = 4$).

where $\mathbf{f}_1$ is the first row of V_1, ϕ_1 is the first element of $\mathbf{v}_1$, $\mathbf{l}_2$ is the last row of V_2, ψ_2 is the last element of $\mathbf{v}_2$, $\mathbf{l}$ is the last row of V, ψ is the last element of $\mathbf{v}$, $\mathbf{f}$ is the first row of V and ϕ is the first element of $\mathbf{v}$.

Because most of the running time of D&C is consumed for vector update during singular vector computation, SVDC is wholly faster than the normal one.

3 Preliminary result of Parallel double Divide and Conquer

Figure 1 is a pictorial model of SVDC executed by 4 processors. It is the tree built by the dividing process. A leaf of the tree represents a submatrix. A node is a merge process of D&C. A number on leaves or nodes shows processor number to address the process, which is computing SVD of submatrix at leaves and merge of submatrices at nodes. A line is process transition. In particular, an arrowed line between boxes (leaves and nodes) means communication between processors.

First, submatrices at the lowest level (represented as leaves of the tree) are evenly assigned to all processors in order. Second, each processor computes SVDs of assigned submatrix independently. Then the merge processes are begun at each processor. For each depth of the tree from the bottom, the submatrices are merged in order. The process is paused when the depth becomes more than P, the number of processors. It is time to start communication. We call a processor which is a left son of the parent *Slave* and a son *Master*. A slave sends data Σ_M, $\mathbf{f}$, $\mathbf{l}$, ϕ, ψ, to be needed for parent's merge process to master and finishes tasks at SVDC process. A master receives the data and merge two submatrices. This process is proceeded until the root node is computed. When it is over, the computed singular values are broadcasted to every processor to compute singular vectors parallel. Finally, twisted factorization is invoked parallel. Vectors to be computed are assigned to every processor evenly in order, and these results are gather to one processor.

Table 1: Specification of test bed.

	Appro HyperBlade PC Cluster
CPU	AMD Opteron 1.6GHz (SMP with 2CPUs × 8)
Network	Gigabit Ethernet
OS	SuSE Linux 8.0
(kernel)	(Linux 2.4.19-SMP)
Compiler	pgf77 5.1-3
(Option)	-O3
BLAS	Optimized by ATLAS

This parallel model is a straightforward approach. Although it calls a few communication to order, practical efficiency is not high. After the communication is begun, the number of idle processors is increased step by step. What is worse, the tree is top–heavy, merge process costs more at upper level. Parallelization of each merge makes improvement. The detail will be discussed in future papers.

3.1 Numerical experiments

In this section, we evaluate the parallel dDC with respect to parallelism. We compare it with the following two algorithms on a bidiagonal SVD.

- Parallel I–SVD: I–SVD algorithm whose discretized interval of mdLVs is $\delta^{(t)} = 1.0$. Inverse Iteration updates computed vectors. Twisted factorization and Inverse Iteration is parallel.
- QR: QR algorithm with shifts (PDBDSQR in ScaLAPACK[9]).

Here, dDC uses QR to solve submatrices and inverse iteration to update computed vectors.

Table 1 shows specification of test beds. We compute SVD of a bidiagonal matrix whose diagonal elements are 2.001 and subdiagonal elements are 2.0. All singular values are separated to each other. The deflation of D&C and dDC seldom occurs.

The parallel dDC is implemented with MPI[10]. The number of processors is ranged from 1 to 16. Table 2 shows the best times of 10 executions of parallel dDC. Dimensions of the matrices are $n = 3,000$, $n = 5,000$ and $n = 7,000$.

Table 2: Timing of Parallel dDC.

	The number of processors				
	1	2	4	8	16
n=3,000	5.78	2.93	1.50	0.79	0.48
n=5,000	18.17	9.38	4.76	2.83	1.66

in second: [s]

Table 3: Comparison of parallel dDC to parallel I-SVD and QR ($n = 3,000$).

	The number of processors				
	1	$2_{(2\times1)}$	$4_{(2\times2)}$	$8_{(2\times4)}$	$16_{(2\times8)}$
dDC	5.78	2.93	1.50	0.79	0.48
I-SVD	5.53	3.44	2.36	1.80	1.52
QR	867.98	433.61	173.44	66.78	34.71

in second: [s]

In the case of $n = 7,000$, parallel dDC acquires 0.84 of parallel efficiency and speed-up is 13.47 by 16 processors. For the smaller case, it also shows good performance. We can say that parallel dDC has high scalability in this case.

Then, parallel dDC is compared to parallel versions of I-SVD and QR. Table 3 shows the best times of 10 executions. The dimension is $n = 3,000$. Parallel dDC is more than tree times faster than Parallel I-SVD by 16 processors. It is much faster than Parallel QR, although it shows super linear efficiency due to good use of cache memory. Figure 2 illustrates the execution times of parallel dDC and parallel I-SVD.

4 Conclusions

A new framework of SVD was recently proposed that computes singular values first, then the corresponding vectors are computed by twisted factorization later. double Divide and Conquer (dDC) is one of the algorithms built on this framework. The speed and accuracy of dDC are as well or even better than standard algorithms such as QR and normal Divide and Conquer. In addition, it is expected that dDC has great parallelism because each step is theoretically parallel and heavy communication is not required. This paper shows a basic idea to parallelize dDC and its parallel implementation with MPI is tested on distributed memory parallel computer. It shows high scalability and much faster than parallel I-SVD and QR on a matrix. As a future work, study on an affinity of precision between the part of singular computation in dDC and twisted factorization should be proceed. Moreover, comprehensive test on practical and huge applications will be taken.

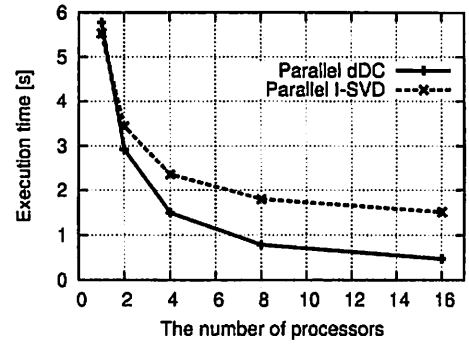

Figure 2: Execution times of parallel dDC and parallel I-SVD.

References

[1] I. Dhillon and B. Parlett. Orthogonal eigenvectors and relative gaps. *SIAM J. Matrix Anal. Appl.*, 25(3):858–899, 2004.

[2] K. Fernando. On computing an eigenvector of a tridiagonal matrix. part 1: basic results. *SIAM J. Matrix. Anal. Appl.*, 18(4):1013–1034, 1997.

[3] B. Parlett and I. Dhillon. Fernando's solution to wilkinson's problem: An application of double factorization. *Lin. Alg. Appl.*, 267:247–279, 1997.

[4] M. Iwasaki, S. Sakano, and Y. Nakamura. Accurate twisted factorization of real symmetric tridiagonal matrices and its application to singular value decomposition. *Trans. Japan. Soc. Indust. Appl. Math.*, 15(3):461–481, 2005. (in Japanese).

[5] M. Takata, M. Iwasaki, K. Kimura, and Y. Nakamura. An evaluation of singular value computation by the discrete lotka-volterra system. In *Proc. International Conf. on Parallel and Distributed Processing Techniques and Applications*, volume 2, pages 410–416, 2005. Las Vegas, USA.

[6] M. Iwasaki and Y. Nakamura. Accurate computation of singular values in terms of the shifted integrable scheme. preprint, 2005.

[7] T. Konda, M. Takata, M. Iwasaki, and Y. Nakamura. A new singular value decomposition algorithm suited to parallelization and preliminary results. *Proceedings of the IASTED International Conference on Advances in Computer Science and Technology*, pages 79–84, 2006.

[8] T. Konda, M. Takata, M. Iwasaki, and Y. Nakamura. A new svd algorithm by divide and conquer and twisted factorizations. *IPSJ Symposium Series*, 2006(1):105–112, 2006. (in Japanese).

[9] L. Blackford, J. Choi, and A. Cleary et al. *ScaLAPACK Users' Guide*. SIAM, 1997.

[10] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI Second Edition*. The MIT Press, 1999.