Parallel Computation of the SVD of a Matrix Product *

José M. Claver[1], Manuel Mollar[1] and Vicente Hernández[2]

[1] Dpto. de Informática, Univ. Jaume I, E-12080 Castellón, Spain.
[2] Dpto. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, E-46071 Valencia, Spain.

Abstract. In this paper we study a parallel algorithm for computing the singular value decomposition (SVD) of a product of two matrices on message passing multiprocessors. This algorithm is related to the classical Golub-Kahan method for computing the SVD of a single matrix and the recent work carried out by Golub *et al.* for computing the SVD of a general matrix product/quotient. The experimental results of our parallel algorithm, obtained on a network of PCs and a SUN Enterprise 4000, show high performances and scalability for large order matrices.

1 Introduction

The problem of computing the singular value decomposition (SVD) of a product of matrices occurs in a great variety of problems of control theory and signal processing (see [14, 15, 18]). The product singular value decomposition (PSVD) is defined as follows.

For any given matrices, $A \in \mathbb{R}^{m \times n}$, $B \in \mathbb{R}^{n \times p}$, there exist two matrices $U \in \mathbb{R}^{m \times n}$ and $V \in \mathbb{R}^{n \times p}$ with orthogonal columns, an orthogonal matrix $Q \in \mathbb{R}^{n \times n}$, and a full rank upper triangular matrix $R \in \mathbb{R}^{n \times n}$, such that

$$A = U \Sigma_A R Q^T , \quad B = Q R^{-1} \Sigma_B V^T \tag{1}$$

with

$$\Sigma_A = \mathrm{diag}(\alpha_1, \ldots, \alpha_n), \ \Sigma_B = \mathrm{diag}(\beta_1, \ldots, \beta_n). \tag{2}$$

The n pairs (α_i, β_i) are called the product singular values of (A, B) and the singular values of the product AB are the products $\alpha_i \beta_i$, for $i = 1, \ldots, n$.

The PSVD was introduced by Fernando and Hammarling [8] as a new generalization of the SVD, based on the product AB^T. It complements the generalized singular value decomposition (GSVD) introduced by Van Loan [20] and Paige and Sanders [19], now called quotient-SVD (QSVD), as proposed in [5]. The parallel computation of the GSVD has been treated in [2, 3]. A complete study of the properties of the PSVD can be found in [6]. The computation of the PSVD is typically carried out with an implicit Kogbetliantz algorithm, as proposed in [8]. Parallel implementations of this algorithm are presented in [16, 17].

* This research was partially supported by the spanish CICYT project under grant TIC96-1062-C03-01-03.

In this work we are interested in the computation of the SVD of the product AB. The method used is based on the algorithm designed by Golub *et al.* in [12]. Their algorithm is related to the Golub-Kahan procedure for computing the singular value decomposition of a single matrix in that a bidiagonal form of the sequence, as an intermediate result, is constructed [10]. The algorithm derived in [12] applies this method to two matrices as an alternative way of computing the product SVD. In this paper we describe a parallel algorithm for computing the singular values of the product of two matrices and study its implementation on two different message passing multiprocessors using ScaLAPACK.

The rest of the paper is organized as follows. Section 2 describes the implicit bidiagonalization as first step of this method. In section 3 we present a sequential version of this algorithm using LAPACK and the characteristics of our ScaLAPACK based parallel algorithm. In section 4 we analyze the performance and scalability of our parallel algorithm on both shared and distributed memory multiprocessors.

2 Implicit Bidiagonalization

Given two matrices A and B, we want to compute the SVD of the product AB without explicitly constructing of their product. This method involves two stages:

1. The implicit bidiagonalization of the product AB.

2. The computation of the SVD of the bidiagonal matrix.

For the first stage we need about $O(n^3)$ flops; for the second stage we need about $O(n^2)$ flops. Well-known Golub-Reinsch [11], Demmel-Kahan [7] or Fernando and Parlett [9] algorithms can be employed to perform the second stage. Thus, we have focused our efforts on the first stage.

For this purpose, we generate a sequence of matrix updates with the application of different Householder transformations [13]. In order to illustrate the procedure we show its evolution operating on a product of two 4x4 matrices A and B. Here, x stands for a nonzero element, and 0 a zero element.

First, we perform a Householder transformation Q_0 on the rows of B and the columns of A, chosen to annihilate all but the first component of the first column of B:

$$\begin{array}{l} AB = AQ_0^T Q_0 B, \\ A \leftarrow AQ_0^T, \\ B \leftarrow Q_0 B, \end{array} \qquad A = \begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix}, \quad B = \begin{bmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix}.$$

Then, we perform a Householder transformation Q_A on the rows of A, chosen to annihilate all but the first component of the first column of A:

$$Q_A A = \begin{bmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix}$$

Notice that the resulting matrix product $Q_A AB$ presents the same structure:

$$\begin{bmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} \begin{bmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} = \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix}.$$

We are interested in the first row of this product (indicated in boldface above). This row can be constructed as the product of the first row of A and the matrix B. Once it is constructed, we find a Householder transformation operating on the last $(n-1)$ elements of the row, which annihilates all but the two first components of of this row:

$$A(1,:)BQ_B = \begin{bmatrix} x & x & 0 & 0 \end{bmatrix}.$$

Thus, when this transformation is applied to B, the first step of the bidiagonalization is completed as

$$Q_A ABQ_B = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix}.$$

Next, we consider $A(2:n, 2:n)$ and $B(2:n, 2:n)$, and using the same procedure, we bidiagonalize the second column/row of the matrix product. The procedure is repeated until the full matrix product is bidiagonalized.

3 Parallel Algorithm

We have implemented a BLAS-2 routine (denoted by *BD*) for computing implicitly the bidiagonal of product of two $n \times n$ matrices using the LAPACK library [1].

Algorithm 1 *[BD]*

for $i = 1, 2, ..., n-1$

Compute the Householder reflector that nullify B(i+1:n,i):	DLARFG
Apply the Householder reflector to B from the left:	DLARF
Apply the reflector to A from the Right:	DLARF
Compute the reflector that nullify A(i+1:n,i):	DLARFG
Apply the reflector to A from the left:	DLARF
Compute de implicit product AB (A(i,i:n)B(i:n,i:n)):	DGEMV
Compute the reflector that nullify AB(i,min(i+2,n):n):	DLARFG
Store diagonal and off-diagonal elements of the matrix product	
Apply the reflector to B from Right:	DLARF

end for

Last diagonal element of the matrix product is A(n,n)B(n,n)

Algorithm 1 shows, in a detailed manner, the procedure followed by our *BD* routine, where the LAPACK routines used are described to the right hand of each step. The routine obtains two vectors corresponding to the diagonal and the upperdiagonal of the bidiagonalized matrix product.

In order to obtain the SVD of the resulting bidiagonal we can use either the xLASQ or the xBDSQR LAPACK routines, that implement the Fernando-Parlett[9] and the Demmel-Kahan [7] algorithms, respectively.

Our parallel implementation, see algorithm 2, that includes the corresponding ScaLAPACK routines used, (denoted by *PBD*), is implemented in ScaLAPACK [4]. In this parallel library, the matrices are block cyclically distributed among a $P = p \times q$ mesh of processors.

Algorithm 2 *[PBD]*
Input:
$A, B \in \mathbb{R}^{n \times n}$: distributed matrices
Output:
$D, E \in \mathbb{R}^{n}$: local vectors to store the fragments of the resulting bidiagonal

Create ScaLAPACK descriptors for D and E
for $i = 1, 2, ..., n-1$
Compute the Householder reflector that nullify B(i+1:n,i): PDLARFG
Apply the Householder reflector to B from the left: PDLARF
Apply the reflector to A from the Right: PDLARF
Compute the reflector that nullify A(i+1:n.i): PDLARFG
Apply the reflector to A from the left: PDLARF
Store diagonal ($D(i) \leftarrow B(i,i)$) element: PDELSET
Compute de implicit product AB (A(i,i:n)B(i:n,i:n)): PDGEMV
and part of D as a resulting distributed vector
Store off-diagonal ($E(i)$) element: PDELSET
Compute the reflector that nullify AB(i,min(i+2,n):n): PDLARFG
Store diagonal and off-diagonal elements of the matrix product
Apply the reflector to B from Right: PDLARF
end for
Last diagonal element of the matrix product is A(n,n)B(n,n)

4 Experimental Results

In this section we analyze the performance of our parallel algorithm obtained on two different architectures, a SUN Enterprise 4000 and a network of PCs(Personal Computers). We compare the performance of the serial and the parallel routines on the SUN and the PC cluster.

The SUN 4000 is a shared memory non bus-based multiprocessor (the main memory has 512 MBytes) with 8 UltraSparc processors at 167 MHz and a second

level cache of 512 KBytes. It has a 4×4 crossbar interconnection network between pairs of processors and main memory.

The network of PCs consists of 16 PCs interconnected by a Myricom Myrinet network (MMn). Each PC is a Pentium II at 300 MHz, with 512 KBytes of second level cache memory and 128 MBytes SDRAM of local memory per processor, under Linux operating system. The Miricom Myrinet is an 8×8 bidirectional crossbar network with a bandwidth of 1.28 Gbits/s per link.

All experiments were performed using Fortran 77 and IEEE double-precision arithmetic. We made use of the LAPACK library and the ScaLAPACK parallel linear algebra library [4]. The use of these libraries ensures the portability of the algorithms to other parallel architectures.

The communication in the ScaLAPACK library is carried out using the MPI communications library. We have used optimized MPI libraries on both the SUN (vendor supplied) and the MMn (GM version) machines.

Our first experiment is designed to evaluate the efficiency of our parallel algorithm *PDB*. In Figures 1 and 2 we report the efficiencies obtained for our algorithm, using $P = 2, 4, 6$ and 8 processors, on the MMn and the SUN platforms, respectively. Notice that high performances are obtained for medium-scale and large-scale problems on both machines. On the SUN, the efficiency of our parallel algorithm grows more quickly than in the MMn machine when the order of the matrices is increased. The reason is the better relative speed of communication on the SUN.

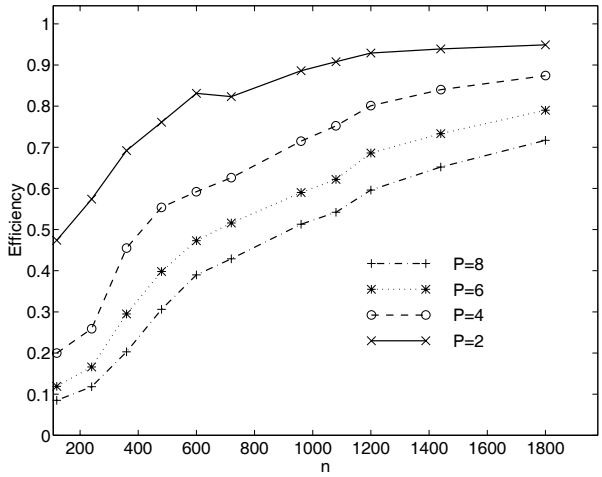

Fig. 1. Efficiencies obtained on the MMn using $P = 2$, 4, 6 and 8 processors for different problem orders.

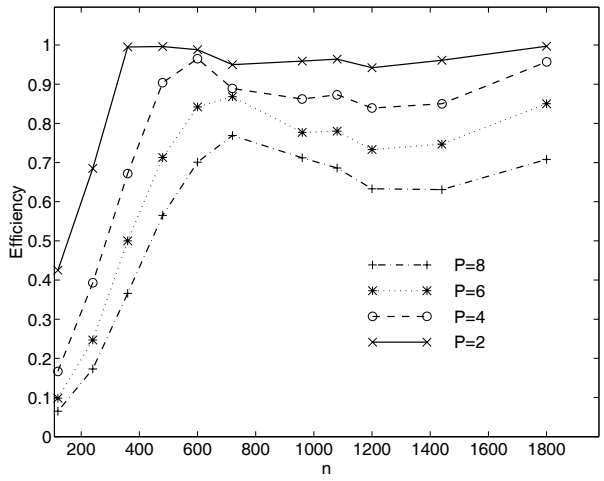

Fig. 2. Efficiencies obtained on the SUN using $P = 2$, 4, 6 and 8 processors for different problem orders.

In our following experiment we analyze the scalability of the parallel algorithm on the MMn platform. In Figure 3 we fix the memory requirements per node of the algorithm to $n/p = 500$, 1000, 1500 and 2000, and report the megaflops per node for $P = 1 \times p^2 = 1, 4, 9$, and 16 processors. Notice that the performance is slightly degraded as the number of processors gets larger, and this performance degradation is minimum as the memory requirements per node are increased.

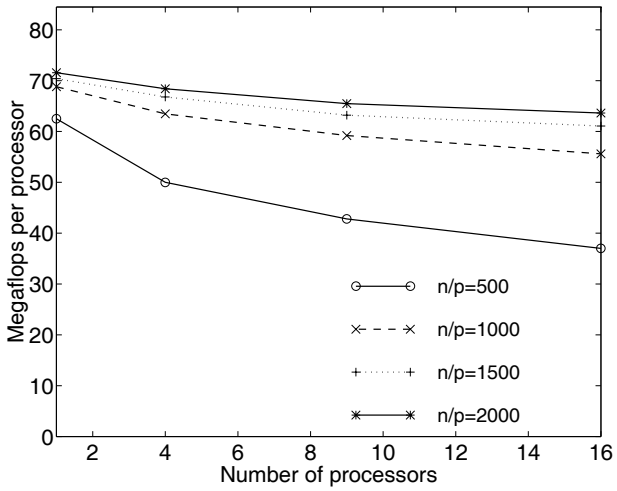

Fig. 3. Mflops per processor obtained on the MMn for constant data load n/p.

5 Concluding Remarks.

We have studied the parallelism of a new method for computing the SVD of a matrix product via the implicit bidiagonalization of this product as intermediate step.

The experimental results show the high efficiency of our algorithm for a wide range of scale problems on shared and distributed memory architectures. Moreover, this algorithm shows an excellent scalability on a PC cluster.

Our algorithm can be easily extended for computing the SVD of a general matrix product. Currently, we are working on a BLAS-3 routine in order to increase the performance of both serial and parallel algorithms.

References

1. Anderson, E., Bai, Z., Bischof, C., Demmel, J., Dongarra, J., Du Croz, J., Greenbaum, A., Hammarling, S., Mckenney, A., Ostrouchov, S., Sorensen, D.: LAPACK User's Guide, Release 1.0., SIAM, Philadelphia (1992).
2. Bai, Z: A parallel algorithm for computing the generalized singular value decomposition, Journal of Parallel and Distributed Computing **20** (1994) 280-288.
3. Brent, R., Luk, F. and van Loan, C.: Computation of the generalized singular value decomposition using mesh connected processors, Proc. SPIE Vol. 431, Real time signal processing VI (1983) 66-71.
4. Blackford, L., Choi, J., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, L., Hammarling, S., Henry, G., Petitet, A., Stanley, K., Walker, D., Whaley, R.: SCALAPACK User's Guide, SIAM (1997).
5. De Moor, B. and Golub, G. H.: Generalized singular value decompositions: A proposal for a standardized nomenclature, Num. Anal. Proj. Report 89-04, Comput. Sci. Dept., Stanford University (1989).
6. De Moor, B.: On the structure and geometry of the PSVD, Num. Anal. Project, NA-89-05, Comput. Sci. Dept., Stanford University (1989).
7. Demmel, J., Kahan, W.: Accurate singular values of bidiagonal matrices, SIAM J. Sci. Stat. Comput. **11** (1990) 873-912.
8. Fernando, K. and Hammarling, S.: A generalized singular value decomposition for a product of two matrices and balanced realization, NAG Technical Report TR1/87, Oxford (1987).
9. Fernando, K. and Parlett, B.: Accurate singular values and differential qd algorithms. Numerische Mathematik **67** (1994) 191-229.
10. Golub, G., W. Kahan, Calculation of the singular values and the pseudoinverse of a matrix, SIAM J. Numer. Anal. **2** (1965) 205-224.
11. Golub, G., Reinsch, W.: Singular value decomposition and the least square solution, Numer. Mathematik **14**, (1970) 403-420.
12. Golub, G., Sølna, K, and van Dooren, P.: Computing the SVD of a General Matrix Product/Quotient, submitted to SIAM J. on Matrix Anal. & Appl.,(1997).
13. Golub, G., Van Loan, C.: Matrix Computations, North Oxford Academic, Oxford (1983).
14. Heat, M., Laub, A., Paige, C., Ward, R.: Computing the singular value decomposition of a product of two matrices, SIAM J. Sci. Stat. Comput. **7** (1986) 1147-1159.

15. Laub, A., Heat, M., Paige, G., Ward, R.: Computation of system balancing transformations and other applications of simultaneous diagonalization algorithms, IEEE Trans. AC **32** (1987) 115-122.
16. Mollar, M., Hernández, V.: Computing the singular values of the product of two matrices in distributed memory multiprocessors, Proc. 4th Euromicro Workshop on Parallel and Distributed Computation, Braga (1996).
17. Mollar, M., Hernández, V.: A parallel implementation of the singular value decomposition of the product of triangular matrices, 1st NICONET Workshop, Valencia (1998)
18. Moore, B.: Principal component analysis in linear systems: Controlability, observability, and model reduction, IEEE Trans. AC **26** (1981) 100-105.
19. Paige, C., Sanders, M.: Towards a generalized singular value decomposition, SIAM J. Numer. Anal. **18** (1981) 398-405.
20. Van Loan, C.: Generalizing the singular value decomposition, SIAM J. Numer. Anal. **13** (1976) 76-83.