SIAM J. MATRIX ANAL. APPL.
Vol. 29, No. 3, pp. 826–837

PARALLEL BIDIAGONALIZATION OF A DENSE MATRIX*

CARLOS CAMPOS†, DAVID GUERRERO‡, VICENTE HERNÁNDEZ‡, AND RUI RALHA§

Abstract. A new stable method for the reduction of rectangular dense matrices to bidiagonal form has been proposed recently. This is a one-sided method since it can be entirely expressed in terms of operations with (full) columns of the matrix under transformation. The algorithm is well suited to parallel computing and, in order to make it even more attractive for distributed memory systems, we introduce a modification which halves the number of communication instances. In this paper we present such a modification. A block organization of the algorithm to use level 3 BLAS routines seems difficult and, at least for the moment, it relies upon level 2 BLAS routines. Nevertheless, we found that our sequential code is competitive with the LAPACK DGEBRD routine. We also compare the time taken by our parallel codes and the ScaLAPACK PDGEBRD routine. We investigated the best data distribution schemes for the different codes and we can state that our parallel codes are also competitive with the ScaLAPACK routine.

Key words. bidiagonal reduction, parallel algorithms

AMS subject classifications. 15A18, 65F30, 68W10

DOI. 10.1137/05062809X

1. Introduction.

The problem of computing the singular value decomposition (SVD) of a matrix is one of the most important operations in numerical linear algebra and is employed in a variety of applications. The SVD is defined as follows.

For any rectangular matrix $A \in \mathbb{R}^{m\times n}$ (we will assume that $m \geq n$), there exist two orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ and a matrix $\Sigma = \begin{bmatrix}\Sigma_A \\ 0\end{bmatrix} \in \mathbb{R}^{m\times n}$, where $\Sigma_A = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$ is a diagonal matrix, such that $A = U\Sigma V^t$. The values $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$ are called the singular values of A.

To compute the SVD of a dense matrix, an important class of methods starts with *Householder bidiagonalization* [11], [12], [14], [15], [19], [20], [21], reducing A to upper bidiagonal form $B \in \mathbb{R}^{n\times n}$, from which the singular values are computed in an iterative manner [2], [16], [18], [22].

The Householder bidiagonalization computes $U_B \in \mathbb{R}^{m\times m}$ and $V_B \in \mathbb{R}^{n\times n}$, as products of Householder matrices, such that

$$A = U_B \begin{bmatrix} B \\ 0 \end{bmatrix} V_B^t, \qquad \text{where} \qquad B = \begin{bmatrix} \alpha_1 & \beta_2 & & \\ & \alpha_2 & \ddots & \\ & & \ddots & \beta_n \\ & & & \alpha_n \end{bmatrix}. \tag{1}$$

The classical method for producing this decomposition is a two-sided algorithm which employs both premultiplication and postmultiplication by Householder matri-

*Received by the editors March 30, 2005; accepted for publication (in revised form) by J. L. Barlow February 1, 2007; published electronically July 25, 2007.

http://www.siam.org/journals/simax/29-3/62809.html

†Departamento de Matemática, Escola Superior de Tecnologia e Gestão (Leiria), Campus 2, Morro do Lena, Alto do Vieiro, 2401-951 Leiria, Portugal (ccampos@estg.ipleiria.pt).

‡Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n, E-46022 Valencia, Spain (dguerrer@dsic.upv.es, vhernand@dsic.upv.es).

§Departamento de Matemática, Universidade do Minho, 4710-057 Braga, Portugal (r_ralha@math.uminho.pt). The research of this author was supported by the Portuguese Foundation for Science and Technology through the research program POCI 2010.

ces. In order to establish an algorithm which is better suited to parallel implementation than the standard bidiagonalization method, Ralha [25], [26], and Ralha and Mackiewicz [27] proposed a new technique that uses only multiplication on the right side of A by Householder matrices. Inspired by Ralha's method, Barlow, Bosner, and Drmač [4] proposed a new stable method for the reduction of rectangular dense matrices to the bidiagonal form.

In this paper we present both methods and propose a modification to the Barlow method which halves the number of communication instances in the parallel implementation, making the algorithm even more attractive for distributed memory systems.

This paper is organized as follows. In section 2 we describe the new bidiagonalization methods. Section 3 deals with the sequential and parallel implementations, using LAPACK [1] and ScaLAPACK [8] routines. In section 4 we analyze the experimental results of our numerical tests. Section 5 summarizes our conclusions and future work.

2. New bidiagonalization methods.

2.1. Ralha bidiagonal reduction. Given a rectangular dense matrix $A \in \mathbb{R}^{m\times n}$, the bidiagonalization method proposed by Ralha is comprised of two stages. The first stage consists of a sequence of $n-2$ Householder transformations

$$A_r = A_{r-1} \cdot \operatorname{diag}(I_r, H_r) \qquad (r = 1, \ldots, n-2), \tag{2}$$

where I_r is the identity matrix of order r, $A_0 = A$, and the columns a_i and a_j of the final matrix A_{n-2} satisfy

$$a_i^t a_j = 0 \qquad \text{for} \qquad |i-j| > 1. \tag{3}$$

This can be understood as an implicit reduction of the symmetric semidefinite positive matrix A^tA to tridiagonal form. In the rth step, the construction of the Householder vector v_r in

$$H_r = I_{n-r} - \frac{2}{v_r^t v_r} v_r v_r^t \tag{4}$$

requires the computation of $n-r$ dot products involving the appropriate columns of A_{r-1}.

Having produced A_{n-2}, the second stage is a variant of the Gram–Schmidt orthogonalization method that produces the factorization $A_{n-2} = QB$, where B is the required upper bidiagonal matrix.

Representing by a_i and q_i the columns of A_{n-2} and Q, respectively, we have

$$\begin{bmatrix} a_1 \cdots & a_i \cdots & a_n \end{bmatrix} = \begin{bmatrix} q_1 \cdots & q_i \cdots & q_n \end{bmatrix} \begin{bmatrix} \alpha_1 & \beta_2 & & & & \\ & \alpha_2 & \ddots & & & \\ & & \ddots & \beta_i & & \\ & & & \alpha_i & \ddots & \\ & & & & \ddots & \beta_n \\ & & & & & \alpha_n \end{bmatrix} \tag{5}$$

with

$$q_1 = \frac{a_1}{\alpha_1}, \qquad q_i = \frac{a_i - \beta_i q_{i-1}}{\alpha_i} \quad (i = 2, \ldots, n). \tag{6}$$

Each β_i is chosen to make q_i orthogonal to q_{i-1}, and each α_i is such that $\|q_i\|_2 = 1$; with these conditions, we get from (6)

$$\alpha_1 = \|a_1\|_2 ,$$
$$\beta_i = a_i^t q_{i-1} \quad (i = 2, \ldots, n) , \tag{7}$$
$$\alpha_i = \|a_i - \beta_i q_{i-1}\|_2 \quad (i = 2, \ldots, n) .$$

The first stage of this method is perfectly stable in the sense that the computed $\widetilde{A}_{n-2}$ satisfies

$$\widetilde{A}_{n-2} = (A + E)P, \tag{8}$$

where P is exactly orthogonal and

$$\|E\|_2 \leq g(m, n)\varepsilon_M \|A\|_2 \tag{9}$$

for some modestly growing function $g(m, n)$ and machine epsilon ε_M [24, pp. 94–96]. Furthermore, if $A = DX$, where D is diagonal and $cond(X) \ll cond(A)$, then these one-sided orthogonal transformations preserve the small singular values better than two-sided transformations [10].

It may happen that some nonadjacent columns of $\widetilde{A}_{n-2}$ are not orthogonal to working precision[1] and, even when all those columns are numerically orthogonal, the process of producing a bidiagonal B from $\widetilde{A}_{n-2}$ may bring trouble. To give an insight into the problem, consider the following triangular matrix:

$$R = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 10^{-9} & 1 & 10^{-9} & 10^{-7} \\ 0 & 0 & 10^{-3} & 10^{-6} & -10^{-4} \\ 0 & 0 & 0 & 1 & 10^{-11} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} .$$

We have

$$R^t R = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 1 + 10^{-18} & 10^{-9} & 10^{-18} & 10^{-16} \\ 0 & 10^{-9} & 1 + 10^{-6} & 2 \times 10^{-9} & 0 \\ 0 & 10^{-18} & 2 \times 10^{-9} & 1 + O(10^{-12}) & O(-10^{-10}) \\ 0 & 10^{-16} & 0 & O(-10^{-10}) & 1 + O(10^{-8}) \end{bmatrix} ,$$

which differs from a tridiagonal matrix by quantities not larger than 10^{-16}. If $A = QR$, with Q orthogonal, then $A^t A = R^t R$ and the nonadjacent columns of A are orthogonal to working precision; however, from the uniqueness of the QR decomposition, it follows that there is no bidiagonal B satisfying $A = QB$; in other words, small perturbations outside the tridiagonal band of $A^t A$ cause much larger (as large as $O(10^{-4})$ in this case) perturbations in B; i.e., the problem is ill-conditioned.

[1] See [26] for an example: the Lauchli matrix $L(n, \mu)$ with $n = 7$ and $\mu = \varepsilon_M$.

2.2. Barlow bidiagonal reduction. Barlow, Bosner, and Drmač [4] recently proposed a stable algorithm which consists, essentially, in interleaving the two stages of Ralha's method. More precisely,

1. the vector q_r is computed immediately after applying the Householder transformation H_r; and
2. the Householder vector used in the transformation H_{r+1} is computed from $x_r = A(:, r+1:n)^t q_r$.

Those authors also noted that the computation of the dot product $\beta_r = q_{r-1}^t A(:, r)$ may be avoided (this is also the case with Ralha's algorithm); in [4], an extensive error analysis of the proposed algorithm is carried out which shows that the method is always able to compute an upper bidiagonal matrix B in a backward stable manner. That is, we have, for each $k = 1, \ldots, n$,

$$|\sigma_k(B) - \sigma_k(A)| \leq f(m,n)\varepsilon_M \|A\|_2 + O\left(\varepsilon_M^2\right) \tag{10}$$

for some modestly growing function $f(m,n)$. As was already the case with Ralha's bidiagonalization, there are ill-conditioned matrices where this algorithm obtains small relative errors for all the singular values [26], [10]. Furthermore, the orthogonality of the columns of U is similar to that of the matrix Q in the QR factorization by the modified Gram–Schmidt method [6], [7]; that is, U may be far from orthogonal (see [4, Example 3.1]). Nevertheless, the leading left singular vectors of A can be recovered with good orthogonality (see [4, Corollary 3.20]).

The algorithm may be stated simply as follows (for a more complete statement see [4]).

ALGORITHM 1 (Barlow bidiagonal reduction).

for $r = 1 : n-2$
  $\alpha_r = \|A(:, r)\|_2$
  $q_r = \frac{A(:,r)}{\alpha_r}$
  $x_r = A(:, r+1:n)^t q_r$
  compute H_r *such that* $H_r^t x_r = \beta_{r+1} e_1$
  $A(:, r+1:n) = A(:, r+1:n) H_r$
  $A(:, r+1) = A(:, r+1) - \beta_{r+1} q_r$
end
$\alpha_{n-1} = \|A(:, n-1)\|_2$
$q_{n-1} = \frac{A(:,n-1)}{\alpha_{n-1}}$
$\beta_n = q_{n-1}^t A(:, n)$
$A(:, n) = A(:, n) - \beta_n q_{n-1}$
$\alpha_n = \|A(:, n)\|_2$
$q_n = \frac{A(:,n)}{\alpha_n}$

2.3. Modified Barlow bidiagonal reduction. The advantages of one-sided transformations for parallel bidiagonalization on a multiprocessor system with distributed memory were first discussed in [25]. The best data decomposition consists in assigning to each one of p processors a number of rows of the matrix under transformation, that is, a segment of length m/p of each column (we are assuming, for simplicity, that p is a divisor of m; if this is not the case, then each processor should get either $floor(m/p)$ or $floor(m/p) + 1$ rows). According to the ideas proposed in [25], in the rth step, each processor gets a copy of the entire vector x_r and computes its own copy of the corresponding Householder vector. So, there is some redundancy

in the computation, but its negative effect in the overall efficiency is not dramatic when m/p is large (see [25]). The reward for this approach is two-fold: the load balancing is optimal and interprocessor communication is required only for the $n-r+1$ dot products involved in the computation of the norm α_r and the vector x_r.

Following this strategy, in the parallel implementation of Barlow's method, a communication event, involving all processors, is required to compute α_r alone. Then, a normalization is carried out to produce q_r, and this will be followed by a second communication event involving the processors in the computation of the $n-r$ dot products $x_r = A(:, r+1:n)^t q_r$.

These two communication events may be reduced to only one if we postpone the normalization of the rth column of A_r; that is, the processors cooperate in the global task of computing the $n-r+1$ dot products $x_r = A(:, r:n)^t A(:, r)$ and, from these, each processor will compute locally $\alpha_r = \sqrt{x_r(1)}$, where $x_r(1) = A(:, r)^t A(:, r)$, and the local segment of $q_r = A(:, r)/\alpha_r$. Observe that $x_r(2 : n-r+1)$ differs from the vector x_r computed in Barlow's method by a factor equal to α_r, but there is no need to perform a scaling of $x_r(2 : n-r+1)$ since the resulting Householder reflector in (4) is invariant under such a scaling. However, in the computation of the off-diagonal element we must take into account that the relation

$$\beta_{r+1} = \frac{\|x_r(2 : n-r+1)\|_2}{\alpha_r} \tag{11}$$

holds in each step.

An essential ingredient in the proof presented in [4] for the error bound given in (10) is the fact that, in each step, the computed A_r is the exact product $(A_{r-1} + E_r) \cdot \mathrm{diag}(I_r, H_r)$, where H_r is the exact Householder reflector corresponding to the vector x_r of the inner products and $\|E_r\|_2 \leq O(\varepsilon_M)\|A_{r-1}\|_2$. This matrix E_r encapsulates the errors in the approximation $\hat{v}_r$ computed for the exact vector v_r and also the errors produced in the update $A_{r-1} \cdot \mathrm{diag}(I_r, \hat{H}_r)$, with $\hat{H}_r = I_{n-r} - \frac{2}{\hat{v}_r^t \hat{v}_r} \hat{v}_r \hat{v}_r^t$. In the modified method, a slightly different approximation $\tilde{v}_r$ will be produced (for a detailed error analysis in the computation of the Householder vector see [30, pp. 152–157]), but, similarly to $\hat{v}_r$, $\tilde{v}_r$ defines a Householder reflector, say, $\tilde{H}_r$, that is very close to the exact one; i.e., we have $\|\tilde{H}_r - H_r\|_2 = O(\varepsilon_M)$. We therefore claim that the error analysis given in [4] also applies to our modified method.

Our proposal does not change the arithmetic complexity of Barlow's method and does not reduce the volume of data to be transferred but halves the number of communication events, therefore reducing the overhead caused by the latency in the communications. The total cost of communication depends upon the parallel computation of the inner products only; in [25] it is shown that, on a simple chain of processors, this cost is approximately given by

$$p\frac{n^2}{2}\left(t_{flop} + 2t_{com}\right), \tag{12}$$

where t_{flop} represents the time taken by one floating point operation and t_{com} stands for the time required to pass a floating point number from one processor to another. The factor p in (12) essentially reflects the diameter of the network and may be replaced by $\sqrt{p}$ in the case of a square grid.

The computation of α_{n-1} and β_n may also be arranged in a way that saves one communication event in the parallel implementation. As in the previous steps, we may use communication to get $x_{n-1} = A(:, n-1:n)^t A(:, n-1)$ in each processor;

then, we have

$$\alpha_{n-1} = \sqrt{x_{n-1}(1)} \qquad \text{and} \qquad \beta_n = x_{n-1}(2)/\alpha_{n-1}. \tag{13}$$

In practical implementations of these algorithms, q_r may overwrite $A(:,r)$ to reduce the volume of the storage required. In the next section we do so.

3. Sequential and parallel implementations. In this section we describe the methodology used to develop our sequential and parallel implementations. The same methodology was applied to all implementations, but from now on we will refer only to the sequential and parallel implementations of the modified Barlow method.

In order to obtain high portability and efficiency, all our implementations use, as much as possible, LAPACK and ScaLAPACK routines. It must be stressed that our implementations rely on level 2 BLAS routines [17]. A block organization of the Barlow method has been under development by Bosner and Barlow (see [9]), who have reported significant reduction in the execution time of the sequential algorithm, depending upon the size of the matrices. However, those authors also found that for parallel processing the nonblocked algorithm is preferred due to large overheads in the block version.

From our sequential codes we obtained the corresponding parallel codes by translating the BLAS and the LAPACK routines into calls of the equivalent parallel routines of PBLAS [13] and ScaLAPACK. This translation process takes into account the data distribution and the corresponding rules to convert sequential LAPACK-based programs into parallel ScaLAPACK-based programs.

Our parallel implementation of the modified Barlow method, including the corresponding PBLAS and ScaLAPACK routines, is stated as follows.

ALGORITHM 2 (parallel implementation).

Get the context of ScaLAPACK's process grid	*BLACS_GRIDINFO*
Create ScaLAPACK's descriptor for x_r	*DESCINIT*
for $r = 1 : n-2$	
$x_r = A(:, r:n)^t A(:, r)$	*PxGEMV*
$\alpha_r = \sqrt{x_r(1)}$	
$A(:, r) = \frac{A(:,r)}{\alpha_r}$	*PxSCAL*
compute H_r *such that* $H_r^t x_r(2 : n-r+1) = \phi_r e_1$	*PxLARFG*
$A(:, r+1:n) = A(:, r+1:n) H_r$	*PxLARF*
$\beta_{r+1} = \frac{\phi_r}{\alpha_r}$	
$A(:, r+1) = A(:, r+1) - \beta_{r+1} A(:, r)$	*PxAXPY*
end	
$x_{n-1} = A(:, n-1:n)^t A(:, n-1)$	*PxGEMV*
$\alpha_{n-1} = \sqrt{x_{n-1}(1)}$	
$\beta_n = x_{n-1}(2)/\alpha_{n-1}$	
$A(:, n-1) = \frac{A(:,n-1)}{\alpha_{n-1}}$	*PxSCAL*
$A(:, n) = A(:, n) - \beta_n A(:, n-1)$	*PxAXPY*
$\alpha_n = \|A(:, n)\|_2$	*PxNRM2*
$A(:, n) = \frac{A(:,n)}{\alpha_n}$	*PxSCAL*

With the ScaLAPACK data distribution, which follows a two-dimensional block cyclic scheme, we manage to assign m/p rows (not contiguous) to each processor by reducing the grid to a single column of processors. We emphasize that, as a direct consequence of the use of the routines from ScaLAPACK, we have not fully implemented the parallel algorithm as presented in the previous section. In our implementation,

the computation of the inner products is carried out with PxGEMV and, as follows from the array descriptor that we have used, the resulting vector x_r is stored on a single processor (processor 0, say). As a consequence of this distribution, during the execution of PxLARFG, the computation of ϕ_r is carried out on processor 0 only and no communication is required. The application of the Householder reflectors (with PxLARF) requires communication to make the value ϕ_r available on each processor.

Finally, we note that, in applications where it is not necessary to produce a matrix Q with normalized columns, we may change Algorithm 2 in a way that reduces the number of floating point divisions. This consists of removing the scaling operations $A(:,r) = A(:,r)/\alpha_r$ (PxSCAL) for $r = 1, \ldots, n$ and rewriting the PxAXPY operations as $A(:,r+1) = A(:,r+1) - \frac{\beta_{r+1}}{\alpha_r} A(:,r)$ for $r = 1, \ldots, n-1$, with a total savings of $mn - (n-1)$ divisions.

4. Experimental results.

4.1. Introduction. In this section we analyze the execution times of our implementations, obtained on a cluster with 20 biprocessor nodes where each node is a Pentium Xeon at 2GHz, 1GB of RAM, and Redhat Linux operating system. The nodes are connected through a SCI network, organized in a 4×5 2D torus grid. Each node has been treated as a single processor machine and the biprocessor feature has not been exploited. Unfortunately, only 10 nodes of the cluster were available for our computational tests.

All experiments were performed using Fortran 90 and IEEE standard double precision floating point arithmetic [23]. As already said, we made use of LAPACK and ScaLAPACK routines in order to ensure a high level of portability and efficiency of our implementations. The communications in ScaLAPACK were carried out using Scali MPI [28], which is an optimized implementation of the standard MPI communication library [29] for SCI networks.

In all experiments, the execution times were measured in seconds, and the test matrices (rectangular matrices with sizes ranging from 10000×1000 to 10000×4500 and square matrices with sizes ranging from 1000×1000 to 4500×4500) were generated randomly.

The execution times that will be reported are strictly for the process of producing the bidiagonal; i.e., no accumulation of the orthogonal transformations was carried out.

4.2. Sequential codes. In Figure 1 we compare the execution times of the LAPACK routine DGEBRD and our sequential codes for the case of rectangular matrices. If m is much larger than n, it is more efficient to carry out an initial QR decomposition [11]. In our tests we have not done this, mainly because PDGEBRD (from ScaLAPACK) also does not perform such a decomposition. As can be seen, the new bidiagonalization methods have similar execution times, and, in general, our sequential codes are competitive with DGEBRD.

The number of flops involved in the new methods is approximately equal to $3mn^2$ flops and the operation count for DGEBRD is $4mn^2 - 4/3n^3$; therefore, the new methods require fewer flops whenever $m > \frac{4}{3}n$ [4], [25]. For m fixed, the new methods are less competitive as n grows. For $m = 10000$ and $n = 4000$, DGEBRD uses about $\frac{4}{3}$ the number of flops required by the new method. However, looking at Figure 1, we see that the execution times are almost equal. This is because DGEBRD applies block updates of the form $A - UX^t - YV^t$ using two calls to the level 3 BLAS routine DGEMM; these calls account for about half the work [1] and make the code more

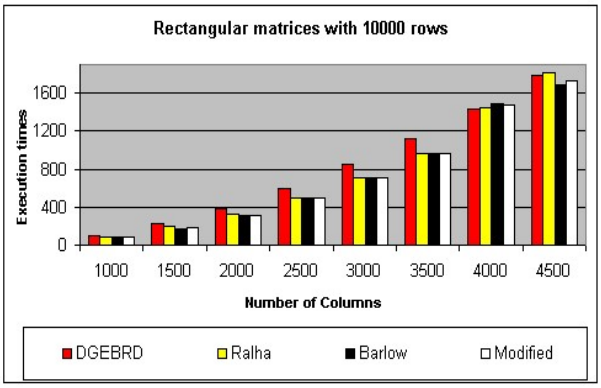

FIG. 1. *DGEBRD versus new methods (rectangular matrices).*

efficient. The new methods are based upon level 2 BLAS routines; i.e., the ratio of floating point operations to memory references is lower.

4.3. Parallel codes. In this section we compare, in terms of the execution times measured, the ScaLAPACK routine PDGEBRD and our parallel codes. We have observed that there is a nonnegligible influence of the sizes of the rectangular grid used for the configuration of the processors. Unlike the ScaLAPACK routine, our parallel implementations perform better on a grid with a single column. In the following comparisons we will always use the best execution time.

In Figure 2 we report the execution times of the Barlow and the modified Barlow parallel codes running on 2, 4, 6, 8, and 10 processors for rectangular matrices. For each n (number of columns), there are five pairs of consecutive bars, one pair for each value of p (number of processors). On each pair, the dark bar corresponds to the Barlow method, and the white bar corresponds to the modified method. The gain that can be observed for the parallel implementation of the modified method is not impressive. This is not surprising because our computational platform has efficient communication, as one can conclude from the high efficiency obtained for all the parallel algorithms. Since the modified method reduces the communication overheads, we expect the gain to be much more significant on a system where the cost of the communications is heavier (a loosely coupled network of personal computers, for example).

Figure 3 allows a comparison of the execution times of PDGEBRD and the modified method on 2, 4, 6, 8, and 10 processors. Again, for each n, there are five pairs of consecutive bars, dark and white, the dark bar corresponding to PDGEBRD, and the white bar to the modified method. As can be seen, on two processors our code is slower than PDGEBRD, but the situation is reversed as we increase the number of processors. At this point, in our experiments, we were very sorry to not have the opportunity to use many more processors since we do believe that the new method has better scalability than the ScaLAPACK routine. In section 5 we justify this conviction with some arguments.

In Figure 4, for each n, we give the efficiency obtained for PDGEBRD (dark) and for the parallel code of the modified Barlow algorithm (white), running on 2, 4, 6, 8,

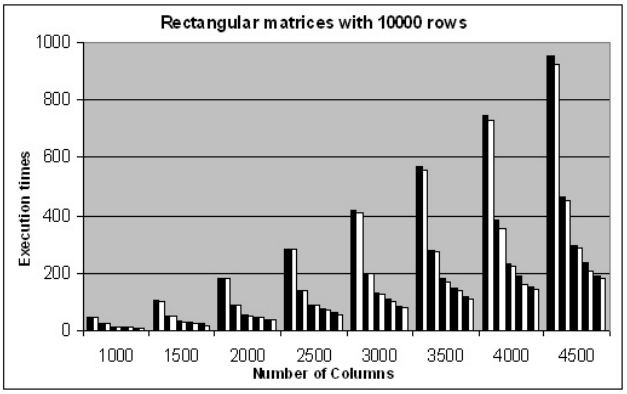

FIG. 2. *Barlow's versus modified (rectangular matrices).*

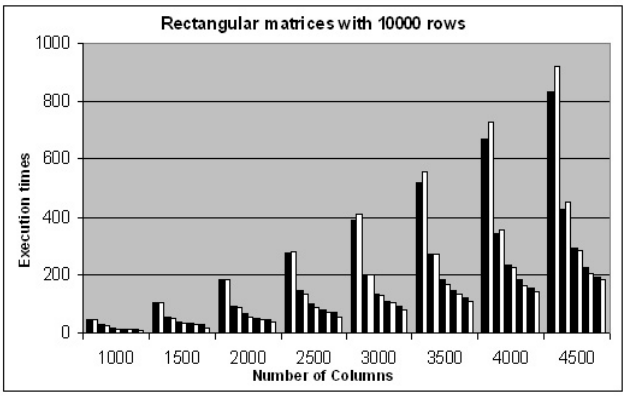

FIG. 3. *PDGEBRD versus modified (rectangular matrices).*

and 10 processors. The efficiency is computed according to the usual formula:

$$\text{Efficiency} = \frac{\text{Execution time on a single processor}}{p \times (\text{Execution time on } p \text{ processors})}. \tag{14}$$

Finally, to illustrate the influence of the processors grid, we ran PDGEBRD on a linear array of processors. This causes a significant degradation of the efficiency of PDGEBRD (compare the dark bars for efficiency in Figures 4 and 5), and, in this case, the new code is clearly more efficient.

5. Conclusions and future work. Inspired by an algorithm proposed by one of us, Barlow, Bosner, and Drmač recently presented a backward stable method for the reduction of a matrix to bidiagonal form.

We have presented parallel implementations of the method as proposed by those authors and also of a modified method that halves the number of communication events.

The advantages of one-sided transformations over two-sided methods for parallel bidiagonalization have been explained in detail in [25]. The parallel code for the one-sided algorithm is essentially the sequential code with a procedure to compute the dot

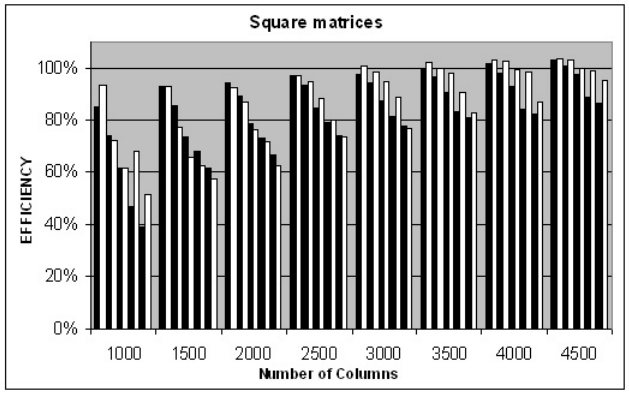

FIG. 4. *PDGEBRD versus modified (efficiency).*

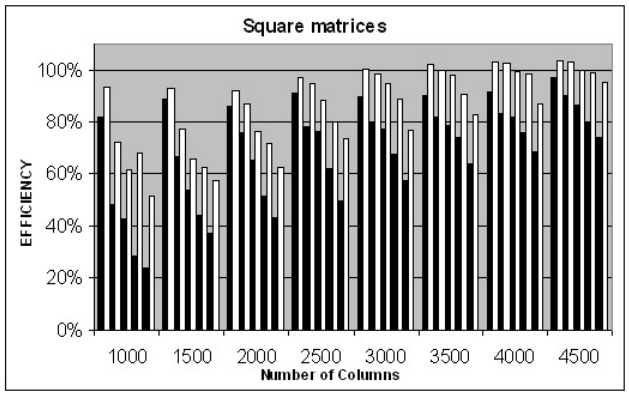

FIG. 5. *PDGEBRD versus modified (efficiency on a linear array of processors).*

products in parallel. This procedure (dubbed GLOBAL.SDOT in [25]) encapsulates all the communication that is required in the parallel algorithm, provided that each processor gets full rows of the matrix. For this reason we do not use a two-dimensional block cyclic distribution. Note that the ScaLAPACK routine PDGEBRD does require communication not only to compute the dot products but also to compute and apply the Householder reflectors. Our one-dimensional distribution, together with the acceptance of some redundancy in the arithmetic, due to the computation of the Householder vectors, produces a parallel algorithm which is well load-balanced and reduces significantly the communication needs, as compared to the ScaLAPACK code. The communication overhead expressed in (12) allows us to conclude (see [25]) that our parallel algorithm is efficient provided that m/p is large enough.

We have described the methodology employed to develop our sequential and parallel codes which intends to use, as much as possible, calls of LAPACK and ScaLAPACK routines, in order to obtain high levels of portability and efficiency.

Our results show that the sequential code for the new method is competitive with the LAPACK routine DGEBRD; although for square matrices we found DGEBRD to be faster. This is not surprising since DGEBRD requires in this case about $\frac{8}{3}n^3$ flops

and the new method uses $\frac{1}{3}n^3$ additional flops. Furthermore, DGEBRD has a better ratio of floating point operations to memory references, because it uses level 2 and level 3 BLAS, whereas the new algorithm does not use any level 3 BLAS routine. Even so, without an initial QR decomposition, the new method is faster than DGEBRD if m is much larger than n (in our tests, this happened with $m = 10000$ and $n = 3500$).

Our experimental results on a multiprocessor system do not show as clearly as we expected initially the superiority of the new method. There is a very good reason for this: the ScaLAPACK routine PDGEBRD proved to be very efficient in our tests; this is due to the fact that the communications in our machine are fast and also because we used a maximum of 10 processors only. Since the new algorithm reduces the communication overheads, its virtues will emerge whenever the cost of communication becomes higher comparatively to computation time (a larger number of processors and/or slower communications). Nevertheless, for rectangular matrices our parallel code was marginally faster than PDGEBRD. We expect to be able to use a larger number of processors in the very near future in order to be able to support our claim that the new method has better scalability than the ScaLAPACK routine.

The modification proposed in this paper for the new algorithm reduces to half the number of communication events. In our tests, the gain has not been dramatic, but it may be much more significant on systems with a larger number of processors and/or larger latency in the communications.

To conclude, let us express our view that the new algorithm is very promising for parallel processing. There is still scope to optimize our code and to make it even more competitive with the highly optimized code of the ScaLAPACK routine.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK Users' Guide*, 3rd ed., Software Environ. Tools 9, SIAM, Philadelphia, 1999.

[2] P. Arbenz, *Divide-and-conquer algorithms for the computation of the SVD of bidiagonal matrices*, in Vector and Parallel Computing, Ellis Horwood Ser. Comput. Appl., Horwood, Chichester, UK, 1989, pp. 1–10.

[3] J. Barlow, *More accurate bidiagonal reduction for computing the singular value decomposition*, SIAM J. Matrix Anal. Appl., 23 (2002), pp. 761–798.

[4] J. Barlow, N. Bosner, and Z. Drmač, *A new stable bidiagonal reduction algorithm*, Linear Algebra Appl., 397 (2005), pp. 35–84.

[5] J. Barlow and J. Demmel, *Computing accurate eigensystems of scaled diagonally dominant matrices*, SIAM J. Numer. Anal., 27 (1990), pp. 762–791.

[6] A. Björck, *Solving linear least squares problems by Gram-Schmidt orthogonalization*, BIT, 7 (1967), pp. 1–21.

[7] A. Björck, *Numerics of Gram-Schmidt orthogonalization*, Linear Algebra Appl., 197/198 (1994), pp. 297–316.

[8] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. Whaley, *ScaLAPACK Users' Guide*, Software Environ. Tools 4, SIAM, Philadelphia, 1997.

[9] N. Bosner and J. Barlow, *Block and Parallel Versions of One-Sided Bidiagonalization*, Tech. report, University of Zagreb, Zagreb, Croatia, 2005; poster available on http://osijek.fernuni-hagen.de/∼luka/Presentations/Nela.pdf.

[10] N. Bosner and Z. Drmač, *On accuracy properties of one-sided bidiagonalization algorithm and its applications*, in Proceedings of the Conference on Applied Mathematics and Scientific Computing, Z. Drmač, M. Marušić, and Z. Tutek, eds., Springer, Dordrecht, 2005, pp. 141–150.

[11] T. F. Chan, *An improved algorithm for computing the singular value decomposition*, ACM Trans. Math. Software, 8 (1982), pp. 72–83.

[12] T. F. Chan, *Rank revealing QR factorizations*, Linear Algebra Appl., 88/89 (1987), pp. 67–82.

[13] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. Whaley, *A proposal for a set of parallel basic linear algebra subprograms*, LAPACK Working Note 100, University of Tennessee, Knoxville, TN, 1995.
[14] J. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.
[15] J. Demmel, M. Gu, S. Eisenstat, I. Slapničar, K. Veselić, and Z. Drmač, *Computing the singular value decomposition with high relative accuracy*, LAPACK Working Note 119, University of Tennessee, Knoxville, TN, 1997.
[16] J. Demmel and W. Kahan, *Accurate singular values of bidiagonal matrices*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 873–912.
[17] J. Dongarra, J. Croz, S. Hammarling, and R. Hanson, *An extended set of FORTRAN basic linear algebra subprograms*, ACM Trans. Math. Software, 14 (1988), pp. 1–17.
[18] K. V. Fernando and B. N. Parlett, *Accurate singular values and differential QD algorithms*, Numer. Math., 67 (1994), pp. 191–229.
[19] G. Golub and W. Kahan, *Calculating the singular values and pseudo-inverse of a matrix*, SIAM J. Numer. Anal., 2 (1965), pp. 205–224.
[20] G. H. Golub and C. Reinsch, *Singular value decomposition and least squares solution*, Numer. Math., 14 (1970), pp. 403–420.
[21] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 3rd ed., The Johns Hopkins University Press, Baltimore, MD, 1996.
[22] M. Gu and S. C. Eisenstat, *A divide-and-conquer algorithm for the bidiagonal SVD*, SIAM J. Matrix Anal. Appl., 16 (1995), pp. 79–92.
[23] IEEE, *IEEE Standard for Binary Floating Point Arithmetic*, ANSI/IEEE std 754/1985, IEEE Computer Society, Los Alamitos, CA, 1985.
[24] B. N. Parlett, *The Symmetric Eigenvalue Problem*, Prentice–Hall, Englewood Cliffs, NJ, 1980.
[25] R. Ralha, *A new algorithm for singular value decompositions*, in Proceedings of the Second Euromicro Workshop on Parallel and Distributed Processing, IEEE Computer Society, Los Alamitos, CA, 1994, pp. 240–244.
[26] R. Ralha, *One-sided reduction to bidiagonal form*, Linear Algebra Appl., 358 (2003), pp. 219–238.
[27] R. Ralha and A. Mackiewicz, *An efficient algorithm for the computation of singular values*, in Proceedings of the Third International Congress of Numerical Methods in Engineering (Zaragoza, Spain), M. Doblaré, J. M. Correas, E. Alarcón, L. Gavete, and M. Pastor, eds., Spanish Society of Numerical Methods in Engineering, Barcelona, Spain, pp. 1371–1380, 1996.
[28] Scali AS, *Scali System Guide*, http://www.scali.com, 2002.
[29] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, and J. Dongarra, *MPI: The Complete Reference*, MIT Press, Cambridge, MA, 1996.
[30] J. H.Wilkinson, *The Algebraic Eigenvalue Problem*, Oxford University Press, Oxford, UK, 1965.