Using Pentangular Factorizations for the Reduction to Banded Form

B. Großer[1] and B. Lang[2]

[1] Department of Mathematics, University of Wuppertal, D-42097 Wuppertal, Germany
[2] Computing Center, Aachen University of Technology, D-52074 Aachen, Germany

Abstract. Most methods for computing the singular value decomposition (SVD) first bidiagonalize the matrix. The ScaLAPACK implementation of the blocked reduction of a general dense matrix to bidiagonal form performs about one half of the operations with BLAS3. If we subdivide the task into two stages *dense* → *banded* and *banded* → *bidiagonal*, we can increase the portion of matrix-matrix operations and expect higher performance. We give an overview of different techniques for the first stage.
This note summarizes the results of [9, 10].

Keywords: Linear algebra; Singular value decomposition; Bidiagonal reduction; Parallel BLAS

1 Introduction

Many algorithms for computing the singular value decomposition (SVD) of a general matrix $A \in \mathbb{R}^{m\times n}$ start with the reduction to bidiagonal form. That is, $A \to B = U^T AV$, where B is upper or lower bidiagonal, and $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal. We assume $m \geq n$ and consider only reduction to upper bidiagonal form. We can obtain B by alternately pre- and postmultiplying A with Householder transformations in order to introduce zeros in the columns and rows of the matrix. In [6], a block formulation of the bidiagonalization algorithm is given, which allows one half of the operations to be performed in matrix-matrix products (BLAS3 [7]) as well as a straightforward parallelization. We will call this reduction technique the *direct method*, because it reduces a dense matrix directly to bidiagonal form.

From Table 1 we see that subdividing the reduction to bidiagonal form into two stages *dense* → *banded* (cf. [10]) and *banded* → *bidiagonal* (cf. [11]) allows the design of *two-stage* bidiagonalization algorithms that do the vast majority of the calculations within matrix-matrix products and therefore can make full use of an optimized BLAS3 implementation. Due to better communication management this gain even increases if the algorithms are run on distributed memory parallel machines. The two-stage algorithms offer an attractive alternative, if only the singular values are required. If the orthogonal transformations must be accumulated explicitly (e.g., to compute all the singular vectors), then the *direct* method is superior.

P. Amestoy et al. (Eds.): Euro-Par'99, LNCS 1685, pp. 1088–1095, 1999.
© Springer-Verlag Berlin Heidelberg 1999

	reduction of A	update U	update V
direct			
overall flop	$4mn^2 - \frac{4}{3}n^3$	$2mn(2m-n)$	$2n^3$
BLAS3 portion	$2mn^2 - \frac{2}{3}n^3$	$2mn(2m-n)$	$2n^3$
two-stage			
first stage			
overall flop	$4mn^2 - \frac{4}{3}n^3 + \mathcal{O}(mnb)$	$2mn(2m-n)$	$2(n-b)^3$
BLAS3 portion	$4mn^2 - \frac{4}{3}n^3 + \mathcal{O}(mnb)$	$2mn(2m-n)$	$2(n-b)^3$
second stage			
overall flop	$8n^2b$	$2mn^2 + \mathcal{O}(n^2b)$	$2n^3 + \mathcal{O}(n^2b)$
BLAS3 portion	0	$2mn^2$	$2n^3$

Table 1. Approximate flop counts for the bidiagonalization methods.

In this note we briefly describe three implementations of the first stage, i.e., the reduction to banded form. Additional details can be found in [9] and [10].

2 Reduction to Banded Form

2.1 The Standard Algorithm

Our methods were designed to make extensive use of the ScaLAPACK [3], PBLAS [4], and BLACS [8] libraries. We assume that A is distributed over a process grid in a two-dimensional block cyclic data layout with block size $n_b \times n_b$. Algorithm 1 reduces A to b upper diagonals. A snapshot of the algorithm is given in Figure 1.

Algorithm 1 : standard (reduction to b upper diagonals)

$i = 0$
while $i < n$ **do**
  $s = i/b + 1$
  $\{\, A_s^{(QRfact)} \equiv A(i{+}1{:}m, i{+}1{:}i{+}b),\ A_s^{(QRupd)} \equiv A(i{+}1{:}m, i{+}b{+}1{:}n)\,\}$
  $A_s^{(QRfact)} = Q_s R_s$ {QR decomposition}
  $A_s^{(QRupd)} \leftarrow Q_s^T A_s^{(QRupd)}$ {update}
  if $i + b < n$
    $\{\, A_s^{(LQfact)} \equiv A(i{+}1{:}i{+}b, i{+}b{+}1{:}n),\ A_s^{(LQupd)} \equiv A(i{+}b{+}1{:}m, i{+}b{+}1{:}n)\,\}$
    $A_s^{(LQfact)} = L_s P_s^T$ {LQ decomposition}
    $A_s^{(LQupd)} \leftarrow A_s^{(QRupd)} P_s$ {update}
  endif
  $i = i + b$
enddo

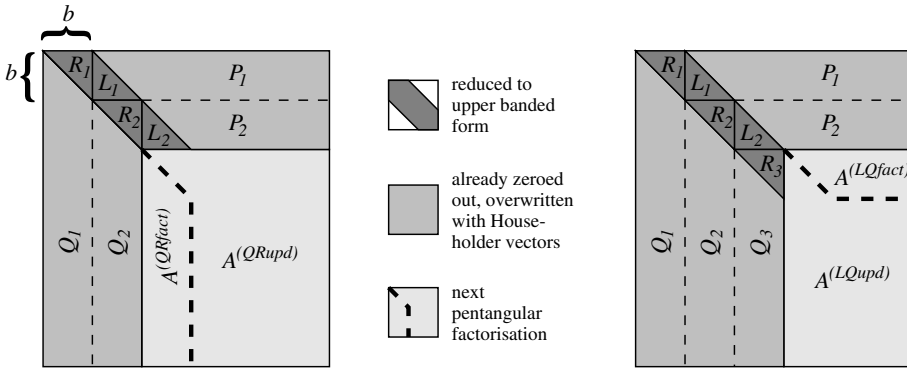


Fig. 1. Third step of Algorithm 1 (**standard**). The transformations Q_s and P_s are represented by the corresponding Householder pairs.

A typical subtask of the algorithm may be regarded as *pentangular factorization*: A given matrix $\tilde{A}$ is partitioned as $\tilde{A} = [A^{(QRfact)}, A^{(QRupd)}]$, where $A^{(QRfact)}$ contains the first b columns of $\tilde{A}$. A QR decomposition $A^{(QRfact)} = Q \cdot R \in \mathbb{R}^{\tilde{m} \times b}$ gives $\tilde{A} = Q \cdot [R, Q^T \cdot A^{(QRupd)}]$ where $[R, Q^T \cdot A^{(QRupd)}]$ has (upper) pentangular shape.

We inspect different methods to compute such factorizations (and their counterparts based on row-partitioning and LQ decompositions).

Our first approach is to modify the ScaLAPACK routine **PDGEQRF** [5], which is designed to compute a QR decomposition of a matrix, in a way that the complete upper pentangular factorization (i.e., not only the QR decomposition of $A^{(QRfact)}$) is computed. **PDGEQRF** partitions the matrix $A^{(QRfact)} \in \mathbb{R}^{\tilde{m} \times b}$ into panels of width $n_b \leq b$. For the k-th panel, the corresponding n_b Householder transformations are combined (cf. [12]):

$$H_{v_1}^{(k)} H_{v_2}^{(k)} \cdots H_{v_{n_b}}^{(k)} = I + V^{(k)} T^{(k)} {V^{(k)}}^T \ , \tag{1}$$

and then applied to the trailing submatrix of $A^{(QRfact)}$. Our modification is based on two observations: First, note that these Householder transformations have also to be applied to $A^{(QRupd)}$. Second, $A^{(QRfact)}$ and $A^{(QRupd)}$ are stored consecutively. Thus we can slightly modify **PDGEQRF** such that $A^{(QRupd)}$ is updated with $I + V^{(k)} T^{(k)} {V^{(k)}}^T$ *immediately*.

A naive implementation of Algorithm 1 would call the ScaLAPACK routines **PDGEQRF** for the blocked QR factorization of $A^{(QRfact)}$ and **PDORMQR** for the blocked update of $A^{(QRupd)}$. If we assume that both routines are called with identical block sizes n_b, the routine **PDORMQR** would compute the *block factors* $T^{(k)} \in \mathbb{R}^{n_b \times n_b}$ once again before they are applied to $A^{(QRupd)}$. It is exactly the computation of $T^{(k)}$ by **PDLARFT** which represents a major bottleneck of the whole algorithm. Thus the modification of **PDGEQRF** to compute the complete factorization results in significant savings.

2.2 Splitting the Factorizations

Another approach to compute the pentangular factorizations was inspired by [1], where reduction to block upper Hessenberg form was considered. In this approach, each QR and LQ decomposition is further subdivided into a "local" and a "global" phase.

For the QR decomposition, these two phases are best explained by taking a "row view" and "column view", resp., of the data distribution, see Figure 2.

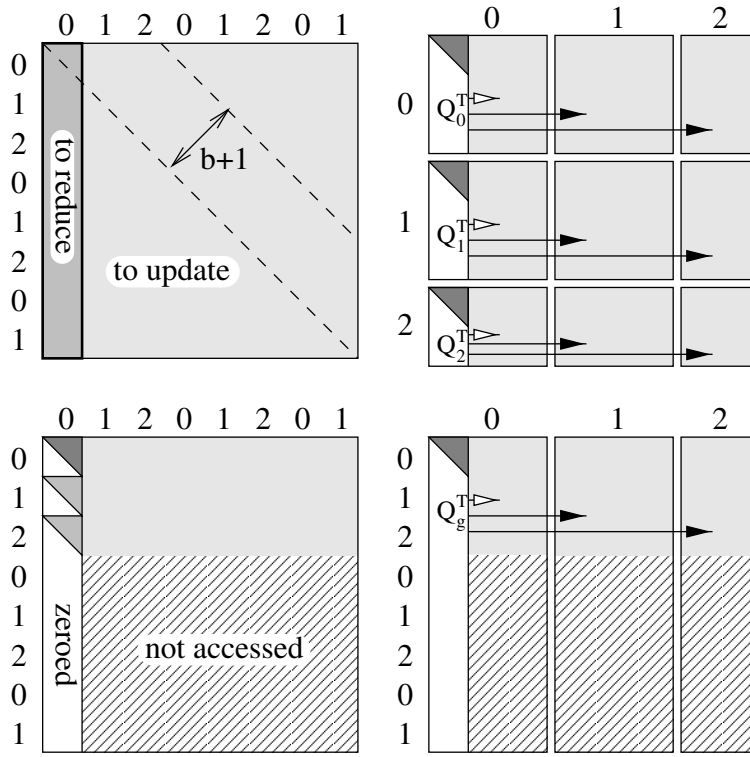


Fig. 2. Local (top) and global (bottom) phases in the "split" QR decomposition.

In the "local" phase, pentangular factorizations are computed for the matrix elements which belong to identical process rows. Here we assume $b = n_b$. This means that the QR decompositions can be carried out locally. Communication is only needed for the update of the trailing submatrices. The "global" phase computes a pentangular factorization for the first $nprows \cdot n_b$ rows of the matrix, where $nprows$ is the number of process rows. The "split factorizations" technique is summarized in Algorithm 2.

Algorithm 2 : splitfac (reduction to b upper diagonals)

$i = 0$
while $i < n$ **do**
  $s = i/n_b + 1$
  { $A_s^{(QRfact)} \equiv A(i{+}1{:}m, i{+}1{:}i{+}n_b)$ }
  { $A_s^{(QRupd)} \equiv A(i{+}1{:}m, i{+}n_b{+}1{:}n)$ }
  local QR decompositions for $A_s^{(QRfact)}$ and local updates for $A_s^{(QRupd)}$
  global QR decomposition for $A_s^{(QRupd)}$'s first *nprows* blocks
    and global update of $A_s^{(QRupd)}$'s first *nprows* block rows
  if $i + b < n$
    { $A_s^{(LQfact)} \equiv A(i{+}1{:}i{+}n_b, i{+}b{+}1{:}n)$ }
    { $A_s^{(LQupd)} \equiv A(i{+}n_b{+}1{:}m, i{+}b{+}1{:}n)$ }
    local LQ decompositions for $A_s^{(LQfact)}$ and local updates for $A_s^{(LQupd)}$
    global LQ decomposition for $A_s^{(LQfact)}$'s first *npcols* blocks
      and global update of $A_s^{(LQupd)}$'s first *npcols* block columns
  endif
  $i = i + n_b$
enddo

Besides other differences, the fact that the reduction to banded form involves equivalence transformations rather than similarity transformations greatly simplifies the communication patterns as compared to the algorithm described in [1].

2.3 Rank–2b Updates

The update phases of the upper and lower pentangular factorizations incorporate rank-b updates, which are performed as BLAS3 operations. Typically, the performance of these operations increases with the blocking factor, i.e., the smallest dimension of the matrices involved in the matrix-matrix products. In the following we will describe one way to double this dimension by replacing the two rank-b updates of Algorithm 1 with one rank-2b update.

Algorithm 3 reduces the matrix A to banded form with b upper *and* b lower diagonals. Here both updates with Q_s^T and P_s are carried out on the same submatrix $A_s^{(upd)} = A_s^{(QRupd)} = A_s^{(LQupd)} \in \mathbb{R}^{q \times p}$, where $q = m - i - b$ and $p = n - i - b$.

Using the original WY representations $Q_s = I + W_L Y_L^T$ and $P_s = I + W_R Y_R^T$ (cf. [2]), the two updates $A^{(upd)} \leftarrow Q_s^T A^{(upd)} P_s$ in Algorithm 3 may either be done separately with about $8qpb + 2qp$ flop. Alternatively they may be combined to a single rank-2b update

$$\begin{aligned} A^{(upd)} &\leftarrow Q_s^T A^{(upd)} P_s \\ &= A^{(upd)} + Y_L W_L^T A^{(upd)} + A^{(upd)} W_R Y_R^T + Y_L W_L^T A^{(upd)} W_R Y_R^T \\ &= A^{(upd)} + [X, Y_L][Y_R, Z_L]^T \ , \end{aligned} \tag{2}$$

where $Z_L = {A^{(upd)}}^T W_L \in \mathbb{R}^{p\times b}$, $Z_R = A^{(upd)} W_R \in \mathbb{R}^{q\times b}$, and $X = Z_R + Y_L(Z_L^T W_R) \in \mathbb{R}^{q\times b}$. The computation of Z_L, Z_R, and X requires $4qpb + 2qb^2 + 2pb^2 + qb$ flop, while the rank-$2b$ update in (2) requires $4qpb + qp$ flop.

Formula (2) is based on the same idea of remodeling two-sided matrix updates as presented in [6].

Algorithm 3 : rk2b (reduction to b upper and b lower diagonals)

$i = 0$
while $i + b < n$ **do**
  $s = i/b + 1$
  $A(i+b+1:m, i+1:i+b) \equiv A_s^{(QRfact)} = Q_s R_s$ {QR decomposition}
  $A(i+1:i+b, i+b+1:n) \equiv A_s^{(LQfact)} = L_s P_s^T$ {LQ decomposition}
  $\{\ A_s^{(upd)} \equiv A(i+b+1:m, i+b+1:n) \equiv A_s^{(QRupd)} \equiv A_s^{(LQupd)}\ \}$
  $A_s^{(upd)} \leftarrow Q_s^T A_s^{(upd)} P_s$ {update}
  $i = i + b$
enddo
cleanup: reduce $A(i + 1 : m, i + 1 : n)$ to banded form.

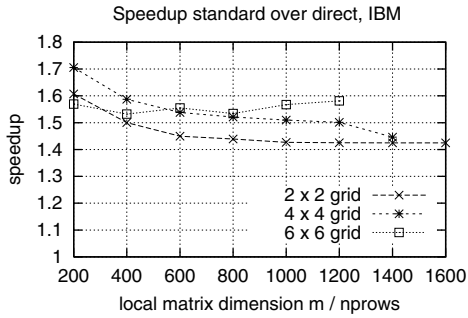
3 Numerical Results

In this section we compare our two-stage reduction techniques to the direct bidiagonalization routine **PDGEBRD** from ScaLAPACK. We present results from the IBM SP/1 located at the High Performance Computing Research Facility, Mathematics and Computer Science Division, Argonne National Laboratory, USA. All programs are coded in Fortran77 and use the portable BLACS library for doing the communication. Calls to the BLAS were directed to the assembly-coded optimized `essl`-library. The experiments were performed in double precision on random matrices with $m = n$.

The overall execution time of the two-stage methods includes the times for the stages *dense* $\rightarrow$ *banded* and *banded* $\rightarrow$ *bidiagonal* and for a re-distribution of the data between the two stages (cf. [9]). A parallel implementation of the second stage *banded* $\rightarrow$ *bidiagonal* is described in [11]. Performance of the algorithms strongly depends on the block size n_b. Thus, the numerical experiments were preceded by a calibration phase to determine optimum parameters. Depending on the size of the process grid optimal block factors $n_b = 12$ or 24 were found. Optimal choices for the intermediate bandwidth in the two-stage approaches lay between $b = 24$ and 36 with b being a multiple of n_b.

Our first approach, using the modified **PDGEQRF** routine (cf. Section 2.1), performed best. Using this algorithm for the reduction to banded form, the two-stage bidiagonalization proved to be superior to the direct method for almost all matrix dimensions and for all grid sizes considered, with speedups up to 1.6 (see Figure 3, left). The improved performance is due to the fact that—roughly

speaking—the matrix-vector products of the direct method have been replaced by matrix-matrix products without significantly increasing the overall complexity. Note that the *speedup increases with the number of processors* because the matrix-matrix products can save on communication startup, too.



IBM, 4×4 grid				
m	t_{direct}	$t_{standard}$	$t_{splitfac}$	t_{rk2b}
800	12.7	6.4	9.6	16.5
1600	34.4	22.2	27.2	44.3
2400	80.7	53.0	67.0	95.2
3200	163.1	103.6	141.9	185.9
4000	273.7	179.8	244.5	312.0
4800	438.0	288.0	398.6	486.2
5600	653.8	452.1	608.0	712.6

Fig. 3. Left: Speedup of the two-stage reduction technique (**standard** variant) over the direct bidiagonalization algorithm **PDGEBRD**. Right: Overall execution times for the direct method vs. variants **standard**, **splitfac** and **rk2b**. The two-stage timings include the part *banded* $\rightarrow$ *bidiagonal*.

The two alternative approaches, **splitfac** and **rk2b**, do not reach the performance of the **standard** variant. The **splitfac** method can still outperform the routine **PDGEBRD** (see Figure 3, right). We will briefly discuss the most important reasons.

In the **splitfac** variant we can identify the global phases as a bottleneck. Here a large amount of communication is delaying a rather moderate number of floating-point operations. The global phases consume about 10 to 20 percent of the overall execution time. Asynchronous communication schemes may help to reduce this percentage, but at the cost of abandoning the BLACS library and therefore disabling fair timing comparisons with ScaLAPACK.

Our approach to maximize the dimension of the matrices involved in the update in order to squeeze some more speed out of the PBLAS3 routine **PDGEMM** as in Algorithm 3 (**rk2b**) was not successful for several reasons. First, detailed measurements show that the rank-$2b$ update according to (2) performs only marginally better than a rank-b update. But this small gain is more than neutralized by the costly extra computations, as setting up W_L or W_R requires substantially more time than building T via **PDLARFT**. In addition, the second stage must now bidiagonalize a banded matrix that has double bandwidth as compared to the one resulting from Algorithm 1.

We have already mentioned that the two-stage approach is not competitive if explicit updates of the transformation matrices U and V are needed. Tests with updates are still of interest to monitor rounding errors by computing deviation

from orthogonality and residual. The errors from both the direct and the two-stage algorithms were always of the same magnitude.

4 Conclusions

Subdividing the bidiagonalization algorithm into two stages allows us to increase the portion of matrix-matrix operations. If only the reduction is needed, arithmetic costs are comparable to the direct method. Of the three techniques to reduce a dense matrix to banded form using pentangular factorizations, the method described in Section 2.1 is the easiest one to implement and yields the best performance. Combined with an effective algorithm for stage *banded* $\rightarrow$ *bidiagonal*, the two stage algorithm significantly outperforms the direct method.

References

[1] M. W. Berry, J. J. Dongarra, and Y. Kim. A parallel algorithm for the reduction of a nonsymmetric matrix to block upper-Hessenberg form. *Parallel Comput.*, 21(8):1184–1200, August 1995.

[2] C. Bischof and C. Van Loan. The WY representation for products of Householder matrices. *SIAM J. Sci. Stat. Comput.*, 8(1):s2–s13, January 1987.

[3] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: A portable linear algebra library for distributed memory computers—design issues and performance. *Computer Phys. Comm.*, 97:1–15, 1996.

[4] J. Choi, J. Dongarra, S. Ostrouchov, A. Petitet, D. Walker, and R. C. Whaley. A proposal for a set of parallel basic linear algebra subprograms. In J. Dongarra, K. Masden, and J. Waśniewski, editors, *Applied Parallel Computing*, pages 107–114. Springer Verlag, 1995.

[5] J. Choi, J. J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley. The design and implementation of the ScaLAPACK LU, QR, and Cholesky factorization routines. *Scientific Programming*, 5:173–184, 1996.

[6] J. Choi, J. J. Dongarra, and D. W. Walker. The design of a parallel dense linear algebra software library: Reduction to Hessenberg, tridiagonal, and bidiagonal form. *Numer. Alg.*, 10:379–399, 1995.

[7] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.*, 16(1):1–17, March 1990.

[8] J. J. Dongarra and R. C. Whaley. LAPACK Working Note 94: A user's guide to the BLACS v1.0. Technical Report CS-95-281, University of Tennessee at Knoxville, March 1995.

[9] B. Großer. Parallele zweistufige Verfahren zur Reduktion auf Bidiagonalgestalt. Diplomarbeit, Fachbereich Mathematik, Bergische Universität GH Wuppertal, 1997.

[10] B. Großer and B. Lang. Efficient parallel reduction to bidiagonal form. *submitted to Parallel Comput.*

[11] B. Lang. Parallel reduction of banded matrices to bidiagonal form. *Parallel Comput.*, 22:1–18, January 1996.

[12] R. Schreiber and C. Van Loan. A storage-efficient WY representation for products of Householder transformations. *SIAM J. Sci. Stat. Comput.*, 10(1):53–57, January 1989.