Blocked Algorithms for the Reduction to Hessenberg-Triangular Form Revisited

LAPACK Working Note 198

Bo Kågström* Daniel Kressner† Enrique S. Quintana-Ortí‡

Gregorio Quintana-Ortí‡

February 1, 2008

Abstract

We present two variants of Moler and Stewart's algorithm for reducing a matrix pair to Hessenberg-triangular (HT) form with increased data locality in the access to the matrices. In one of these variants, a careful reoganization and accumulation of Givens rotations enables the use of efficient level 3 BLAS. Experimental results on four different architectures, representative of current high performance processors, compare the performances of the new variants with those of the implementation of Moler and Stewart's algorithm in subroutine DGGHRD from LAPACK, Dackland and Kågström's two-stage algorithm for the HT form, and a modified version of the latter which requires considerably less flops.

Keywords: Generalized eigenvalue problems, Hessenberg-triangular form, QZ algorithm, orthogonal transformations, high-performance computing, level 3 BLAS, blocked algorithms.

1 Introduction

Given a matrix pair (A, B) where $A, B \in \mathbb{R}^{n \times n}$, a preprocessing step of the QZ algorithm [18] for solving the regular generalized eigenvalue problem $(A - \lambda B)x = 0$ consists in computing orthogonal matrices $Q, Z \in \mathbb{R}^{n \times n}$ such that $Q^T AZ$ is upper Hessenberg while $Q^T BZ$ is upper triangular. This so-called *Hessenberg-triangular (HT) form* of the matrix pair (A, B) yields a significant reduction in the computational cost during the iterative part of the QZ algorithm.

The HT reduction originally proposed by Moler and Stewart [18] first computes a QR decomposition $B = Q_0 B_0$, where Q_0 is orthogonal and B_0 is upper triangular. The matrices A and B are then overwritten by $Q_0^T A$ and $Q_0^T B = B_0$, respectively. All algorithms under consideration require this initial step, which can be performed via efficient routines in LAPACK (DGEQRF, DORGQR/DORMQR) [1]; we therefore assume for the rest of this paper that the

*Department of Computing Science and HPC2N, Umeå University, S-901 Umeå, Sweden; bokg@cs.umu.se.

†Seminar für angewandte Mathematik, ETH Zürich, Switzerland; kressner@math.ethz.ch.

‡Depto. de Ingeniería y Ciencia de Computadores, Universidad Jaume I, 12.071–Castellón, Spain; {quintana,gquintan}@icc.uji.es.

matrix B in the pair (A, B) is already in upper triangular form. In Moler and Stewart's algorithm, matrix A is then reduced to Hessenberg form by applying a sequence of Givens rotations; see Algorithm 1 and Figure 1. As illustrated there, the reduction is solely based on level 1 BLAS operations, half of those applied to A and B having a large stride access to memory. Therefore, this algorithm can be expected to perform poorly on current processors with deep memory hierarchies. Applying a single Givens rotation to a pair of vectors of length m requires $6m$ floating-point arithmetic operations (*flops*), yielding a total cost for Algorithm 1 of $14n^3$ flops. (Hereafter we neglect lower order terms in the flop counts.) A variant of Algorithm 1 based on fast Givens rotations with a lower computational cost was proposed in [16].

Algorithm 1 Moler and Stewart's HT reduction [18]

Input: A general matrix $A \in \mathbb{R}^{n\times n}$ and an upper triangular matrix $B \in \mathbb{R}^{n\times n}$.
Output: Orthogonal matrices $Q, Z \in \mathbb{R}^{n\times n}$ such that $(H, T) = (Q^T AZ, Q^T BZ)$ is in HT form. The matrices A and B are overwritten by H and T, respectively.
Remark: $G_{i-1,i} \in \mathbb{R}^{n\times n}$ denotes a Givens rotation [8, Sec. 5.1.8] acting on rows/columns $i-1$ and i. I_n stands for the identity matrix of order n.

Set $Q \leftarrow I_n,\ Z \leftarrow I_n$.
for $j \leftarrow 1, 2, \ldots, n-2$ **do**
  for $i \leftarrow n, n-1, \ldots, j+2$ **do**
    Construct $G_{i-1,i}$ such that the (i, j) entry of $G_{i-1,i}^T A$ is zero.
    Update $A \leftarrow G_{i-1,i}^T A,\ B \leftarrow G_{i-1,i}^T B,\ Q \leftarrow QG_{i-1,i}$.
    Construct $G_{i,i-1}$ such that the fill-in $(i, i-1)$ entry of $BG_{i,i-1}$ is zero.
    Update $A \leftarrow AG_{i,i-1},\ B \leftarrow BG_{i,i-1},\ Z \leftarrow ZG_{i,i-1}$.
  end for
end for

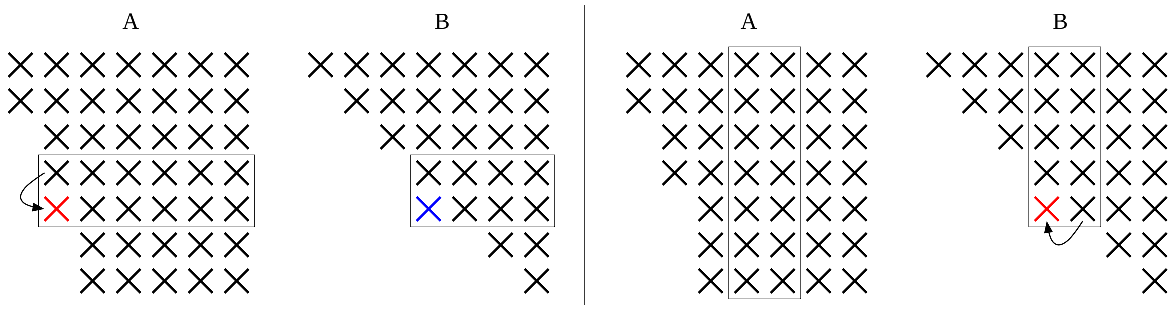


Figure 1: Illustration of one iteration of the innermost loop of Algorithm 1 for $n = 7$, $j = 2$, $i = 5$. From left to right: $A \leftarrow G_{i-1,i}^T A$, $B \leftarrow G_{i-1,i}^T B$, $A \leftarrow AG_{i,i-1}$, and $B \leftarrow BG_{i,i-1}$.

Most LAPACK subroutines achieve portable high performance by casting the computationally most intensive parts of the underlying linear algebra algorithms in terms of matrix-matrix multiplications. This strategy benefits from the use of a highly optimized level 3 BLAS implementation, which itself is often entirely based upon the general matrix multiply and add (GEMM) operation [10, 11]. For example, the LAPACK subroutine `DGEHRD`[1], which

[1]All LAPACK subroutines mentioned in this paper refer to version 3.1.0 released in November 2006.

reduces a matrix to Hessenberg form, performs asymptotically 80% of its operations via calls to level 3 BLAS. The underlying algorithm is based on (compact) WY representations of aggregated products of Householder reflectors [7, 19]. A similar reformulation of Algorithm 1 for reducing a matrix pair to HT form seems to be impossible, since the need to preserve the upper triangular shape of B effectively forces to use either Givens rotations or Householder reflectors of small order.

Two-stage approaches represent an alternative strategy to invoke the matrix-matrix multiplication in the reduction of a matrix to Hessenberg form, as already noted by Bischof and Van Loan in [3]. Considered to be inferior to the LAPACK subroutine `DGEHRD`, because of the large increase in the number of flops [14], they have not attracted much attention for standard eigenvalue problems ($B = I_n$). However, this approach becomes a viable option for generalized eigenproblems. Indeed, on current computer architectures, the two-stage reduction to HT form developed by Dackland and Kågström [6] outperforms Algorithm 1 significantly. In the first stage of this algorithm, A is reduced to a block Hessenberg form with $n_\mathrm{b} > 1$ nonzero subdiagonals, while B is maintained in triangular form. The second stage consists of annihilating all nonzero entries below the first subdiagonal of A by chasing them off at the bottom right corner, much in the spirit of the Rutishauser-Schwarz algorithm [20, 22] for reducing the bandwidth of a symmetric matrix; see also [2]. Unlike in the symmetric standard case, the two stages taken together require considerably more flops than Algorithm 1. For example, when $n_\mathrm{b} = 32$ (and n is sufficiently large) the number of flops of the two-stage algorithm is increased by roughly 50% with respect to that of Algorithm 1. Still, high performance can be attained from the intensive use of level 3 BLAS during the first stage and the application of several techniques to increase the data locality during the second stage.

The main contribution of this paper is a variant of Algorithm 1 that performs at least asymptotically 60% of its operations via calls to level 3 BLAS while asymptotically performing the same number of flops as Algorithm 1. This is achieved by regrouping and accumulating Givens rotations so that they can be applied as matrix-matrix multiplications when generating Q, Z and updating parts of A, B. Experimental results on several platforms indicate that our new variant is significantly faster than subroutine `DGGHRD` from LAPACK, which implements Algorithm 1, and a prototype implementation of the two-stage approach. Note that the same technique has been applied by Lang [15] to generate the orthogonal transformation matrix in the symmetric QR algorithm. Also, parallel algorithms for computing the QR decomposition of a matrix employ similarly grouped sequences of Givens rotations; see, e.g., [5, 17]. Two other contributions of this paper include a modified variant of the two-stage approach with a more reduced computational cost, and a second variant of Algorithm 1 which still applies Givens rotations in terms of level 1 BLAS but exhibits better data locality.

The rest of this paper is organized as follows. In Section 2, we briefly review the two-stage approach and propose a modification that increases the performance of the first stage by allowing for larger matrix-matrix multiplications. Section 3 investigates several possibilities to increase the data locality of Algorithm 1 by regrouping its operations. In particular, it is described how a certain regrouping enables the efficient use of matrix-matrix multiplications in Algorithm 1, leading to our new variant mentioned above. Numerical experiments, reported in Section 4, compare four algorithms/variants for the HT reduction. A few concluding remarks follow in Section 5.

2 Two-stage algorithms

In the following, we briefly describe the two stages of the blocked algorithm for HT reduction presented in [6] and propose a modification of the first stage.

2.1 Stage 1

The first stage consists in reducing A to block Hessenberg form, with $n_\mathrm{b} \times n_\mathrm{b}$ upper triangular subdiagonal blocks for a user-defined block size $n_\mathrm{b} > 1$, while preserving B in upper triangular form. For simplicity, consider that n is an integer multiple of n_b. This allows us to partition A and B into square blocks A_{ij} and B_{ij} of order n_b. Let us illustrate the block partitioning for $n = 6n_\mathrm{b}$:

$$\left(\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} \\ A_{41} & A_{42} & A_{43} & A_{44} & A_{45} & A_{46} \\ A_{51} & A_{52} & A_{53} & A_{54} & A_{55} & A_{56} \\ A_{61} & A_{62} & A_{63} & A_{64} & A_{65} & A_{66} \end{bmatrix}, \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} & B_{15} & B_{16} \\ 0 & B_{22} & B_{23} & B_{24} & B_{25} & B_{26} \\ 0 & 0 & B_{33} & B_{34} & B_{35} & B_{36} \\ 0 & 0 & 0 & B_{44} & B_{45} & B_{46} \\ 0 & 0 & 0 & 0 & B_{55} & B_{56} \\ 0 & 0 & 0 & 0 & 0 & B_{66} \end{bmatrix} \right).$$

Here, as B is upper triangular, each of its diagonal blocks B_{ii} is also upper triangular. Our goal is to reduce the matrix pair (A, B) to block upper Hessenberg-triangular form

$$\left(\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} \\ 0 & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} \\ 0 & 0 & A_{43} & A_{44} & A_{45} & A_{46} \\ 0 & 0 & 0 & A_{54} & A_{55} & A_{56} \\ 0 & 0 & 0 & 0 & A_{65} & A_{66} \end{bmatrix}, \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} & B_{15} & B_{16} \\ 0 & B_{22} & B_{23} & B_{24} & B_{25} & B_{26} \\ 0 & 0 & B_{33} & B_{34} & B_{35} & B_{36} \\ 0 & 0 & 0 & B_{44} & B_{45} & B_{46} \\ 0 & 0 & 0 & 0 & B_{55} & B_{56} \\ 0 & 0 & 0 & 0 & 0 & B_{66} \end{bmatrix} \right), \tag{1}$$

where the subdiagonal blocks $A_{i+1,i}$ and the diagonal blocks B_{ii} are all upper triangular.

For this purpose, we choose some small integer $p \geq 2$ and start by reducing the bottom $p-1$ blocks in the first block column of A to upper triangular form. For example, if $p = 3$, this amounts to computing the QR decomposition

$$\begin{bmatrix} A_{41} \\ A_{51} \\ A_{61} \end{bmatrix} = U_1 R = (I + V_1 T_1 V_1^T) \begin{bmatrix} \hat{A}_{41} \\ 0 \\ 0 \end{bmatrix},$$

where $\hat{A}_{41}$ is upper triangular and $I + V_1 T_1 V_1^T$, with $T_1 \in \mathbb{R}^{n_\mathrm{b} \times n_\mathrm{b}}$ and $V_1 \in \mathbb{R}^{pn_\mathrm{b} \times n_\mathrm{b}}$, is the compact WY representation [21] of the orthogonal factor U_1. Applying $(I + V_1 T_1 V_1^T)^T$ to the bottom three block rows of A and B yields

$$\left(\begin{bmatrix} A_{11} & A_{12} & A_{13} & A_{14} & A_{15} & A_{16} \\ A_{21} & A_{22} & A_{23} & A_{24} & A_{25} & A_{26} \\ A_{31} & A_{32} & A_{33} & A_{34} & A_{35} & A_{36} \\ \hat{A}_{41} & \hat{A}_{42} & \hat{A}_{43} & \hat{A}_{44} & \hat{A}_{45} & \hat{A}_{46} \\ \hat{0} & \hat{A}_{52} & \hat{A}_{53} & \hat{A}_{54} & \hat{A}_{55} & \hat{A}_{56} \\ \hat{0} & \hat{A}_{62} & \hat{A}_{63} & \hat{A}_{64} & \hat{A}_{65} & \hat{A}_{66} \end{bmatrix}, \begin{bmatrix} B_{11} & B_{12} & B_{13} & B_{14} & B_{15} & B_{16} \\ 0 & B_{22} & B_{23} & B_{24} & B_{25} & B_{26} \\ 0 & 0 & B_{33} & B_{34} & B_{35} & B_{36} \\ 0 & 0 & 0 & \hat{B}_{44} & \hat{B}_{45} & \hat{B}_{46} \\ 0 & 0 & 0 & \hat{B}_{54} & \hat{B}_{55} & \hat{B}_{56} \\ 0 & 0 & 0 & \hat{B}_{64} & \hat{B}_{65} & \hat{B}_{66} \end{bmatrix} \right). \tag{2}$$

Now, in order to annihilate the fill-in in B, we compute the RQ decomposition

$$\begin{bmatrix} \hat{B}_{54} & \hat{B}_{55} & \hat{B}_{56} \\ \hat{B}_{64} & \hat{B}_{65} & \hat{B}_{66} \end{bmatrix} = \begin{bmatrix} 0 & \check{B}_{55} & \check{B}_{56} \\ 0 & 0 & \check{B}_{66} \end{bmatrix} U_{\mathrm{r}}, \tag{3}$$

where both $\check{B}_{55}$ and $\check{B}_{66}$ are upper triangular and U_{r} is orthogonal. For a general value of p, this reduction involves $(p-1)n_{\mathrm{b}}$ Householder reflectors; a compact WY representation of U_{r} would therefore take the form $U_{\mathrm{r}} = I + V_{\mathrm{r}} T_{\mathrm{r}} V_{\mathrm{r}}^T$, with $T_{\mathrm{r}} \in \mathbb{R}^{(p-1)n_{\mathrm{b}} \times (p-1)n_{\mathrm{b}}}$ and $V_{\mathrm{r}} \in \mathbb{R}^{pn_{\mathrm{b}} \times (p-1)n_{\mathrm{b}}}$. For $p > 2$, such a WY representation becomes too costly and is replaced by a product of $(p-1)$ WY representations, each of which corresponds to a group of n_{b} Householder reflectors.

Next, blocks A_{21}, A_{31} and $\hat{A}_{41}$ in (2) are reduced to upper Hessenberg form similarly, by orthogonal transformations of block rows/columns 2, 3, and 4 of A, B, while B is maintained in upper triangular form. As a result, the first block column of A takes the desired form (1). To reduce the rest of A, we subsequently process block columns 2, 3, 4, and 5 in an analogous manner; see [6] for more details. The following table contains the flop counts of the overall procedure.

Update of A	Update of B	Update of Q	Update of Z
$\frac{3p^2+10p-7}{3(p-1)} n^3$	$\frac{2p^2+6p-5}{3(p-1)} n^3$	$\frac{2p+1}{p-1} n^3$	$(p+3)n^3$

These figures reflect that the costs for updating A, B, and Z increase linearly with p. This increase is partially compensated by the larger matrix-matrix multiplications resulting from larger values of p. In practice, for typical values of n_{b}, the choices $p = 2$ or $p = 3$ often yield the lowest execution time [6].

2.2 Modified Stage 1

The significant increase of flops for large values of p in Stage 1 is due to the $(p-1)n_{\mathrm{b}}$ Householder reflectors required by the RQ decomposition in (3). In the following, we show how this number can effectively be reduced to n_{b} during the application of the orthogonal transformations. For this purpose, let U_{r} be defined as in (3), and compute a QR decomposition of its first block row:

$$U_{\mathrm{r}}^T \begin{bmatrix} I_{n_{\mathrm{b}}} \\ 0 \\ 0 \end{bmatrix} = \tilde{U}_{\mathrm{r}} \begin{bmatrix} D_1 \\ 0 \\ 0 \end{bmatrix}, \tag{4}$$

where $\tilde{U}_{\mathrm{r}} \in \mathbb{R}^{pn_{\mathrm{b}} \times pn_{\mathrm{b}}}$ is orthogonal and $D_1 \in \mathbb{R}^{n_{\mathrm{b}} \times n_{\mathrm{b}}}$ is a diagonal matrix with diagonal entries ± 1 . Moreover, $\tilde{U}_{\mathrm{r}}$ is a product of only n_{b} Householder reflectors (instead of $(p-1)n_{\mathrm{b}}$ for U_{r}) and consequently admits a compact WY representation $\tilde{U}_{\mathrm{r}} = I + \tilde{V}_{\mathrm{r}} \tilde{T}_{\mathrm{r}} \tilde{V}_{\mathrm{r}}^T$, with $\tilde{T}_{\mathrm{r}} \in \mathbb{R}^{n_{\mathrm{b}} \times n_{\mathrm{b}}}$ and $\tilde{V}_{\mathrm{r}} \in \mathbb{R}^{pn_{\mathrm{b}} \times n_{\mathrm{b}}}$. Then, it follows from (3) and (4) that

$$\begin{aligned} \begin{bmatrix} \hat{B}_{54} & \hat{B}_{55} & \hat{B}_{56} \\ \hat{B}_{64} & \hat{B}_{65} & \hat{B}_{66} \end{bmatrix} \tilde{U}_{\mathrm{r}} \begin{bmatrix} I_{n_{\mathrm{b}}} \\ 0 \\ 0 \end{bmatrix} &= \begin{bmatrix} \hat{B}_{54} & \hat{B}_{55} & \hat{B}_{56} \\ \hat{B}_{64} & \hat{B}_{65} & \hat{B}_{66} \end{bmatrix} U_{\mathrm{r}}^T \begin{bmatrix} D_1^{-1} \\ 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 0 & \check{B}_{55} & \check{B}_{56} \\ 0 & 0 & \check{B}_{66} \end{bmatrix} \begin{bmatrix} D_1^{-1} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \end{aligned}$$

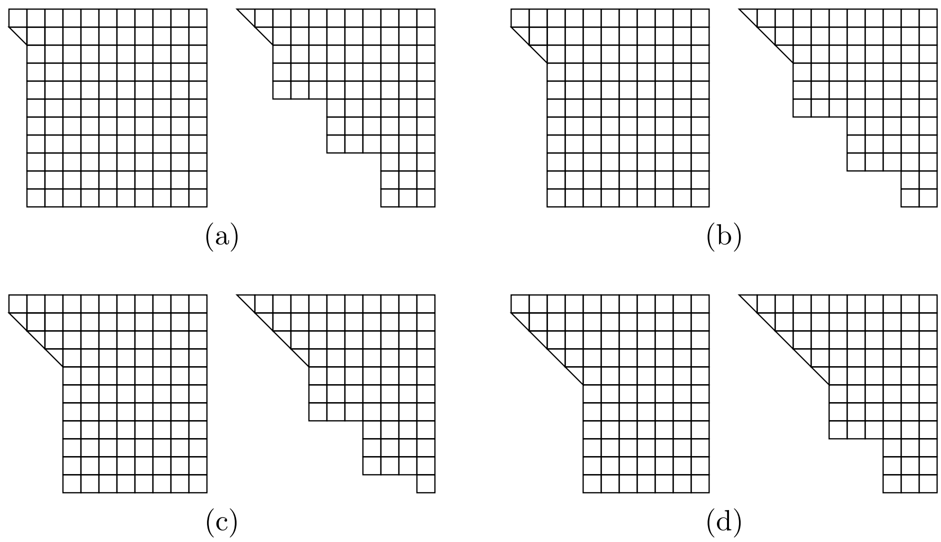


Figure 2: Shapes of A and B after reduction of the (a) 1st, (b) 2nd, (c) 3rd, and (d) 4th block column of A during the modified Stage 1.

This implies that applying $\tilde{U}_{\mathrm{r}}$ to the last three block columns of B in (2) annihilates the blocks $\hat{B}_{54}$ and $\hat{B}_{64}$. However, $\hat{B}_{65}$ is not annihilated, resulting in a chain of $n_{\mathrm{b}} \times n_{\mathrm{b}}$ bulges on the diagonal of B after the complete reduction of the first block column of A with the modified method; see Figure 2 (a) which illustrates the case $p = 4$. It can also be seen from Figure 2 (b)–(d) that these bulges do not interfer with subsequent reductions; they are merely chased along the diagonal until they disappear at the bottom right corner. This allows us to replace the orthogonal matrix U_{r} by the compact WY representation of $\tilde{U}_{\mathrm{r}}^T$. The following table reveals that this approach considerably reduces the number of flops needed for updating A, B, and Z in Stage 1.

Update of A	Update of B	Update of Q	Update of Z
$\frac{10p+5}{3(p-1)}n^3$	$\frac{2p+1}{p-1}n^3$	$\frac{2p+1}{p-1}n^3$	$\frac{2p+1}{p-1}n^3$

These numbers actually *decrease* as p increases, misleadingly suggesting that p should be chosen as large as possible. However, terms of order $O(n^2)$, neglected in the previous expressions, grow proportionally with p^3 and become dominant for large values of p. In practice, for typical values of n_{b} and sufficiently large n, we found that choosing p between 5 and 12 often minimizes the execution time; see Section 4.

2.3 Stage 2

In Stage 2, the nonzero lower $n_{\mathrm{b}} - 1$ subdiagonals of the matrix A in the block HT form (1) are annihilated. The basic algorithm that applies here is a variant of Algorithm 1 that limits the fill-in in A.

The first column of A is reduced by applying a sequence of Givens rotations G_{n_{b},n_b+1}, $G_{n_b-1,n_{\mathrm{b}}}$,..., $G_{2,3}$ from the left. The corresponding update of B creates nonzero subdiagonal entries at positions $(n_{\mathrm{b}} + 1, n_{\mathrm{b}})$, $(n_{\mathrm{b}}, n_{\mathrm{b}} - 1)$,...,$(3, 2)$; see Figure 3 (a). A sequence of

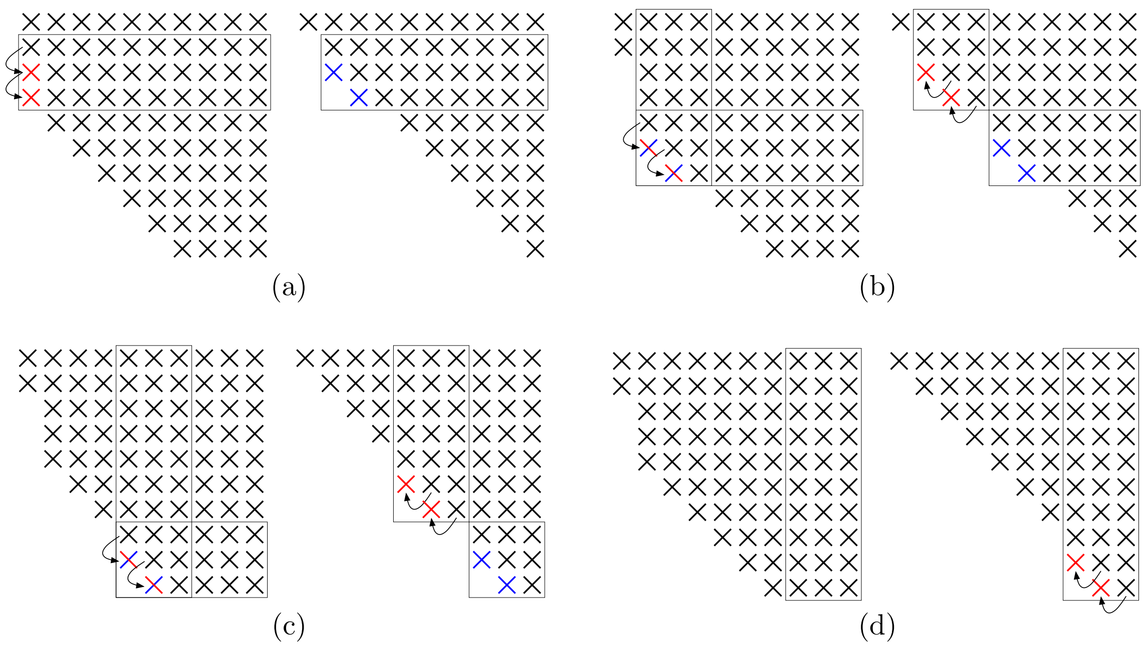


Figure 3: Illustration of the reduction of the first column of A in Stage 2 for $n = 10$, $n_\mathrm{b} = 3$.

Givens rotations G_{n_b+1,n_b}, G_{n_b,n_b-1}, $\dots$, $G_{3,2}$ from the right is next used to annihilate these entries. If we were to apply this sequence directly to update A, then a fully dense $n_\mathrm{b} \times n_\mathrm{b}$ block would be created below the subdiagonal of A. To avoid this effect, we initially only apply G_{n_b+1,n_b}, creating a nonzero $(2n_\mathrm{b} + 1, n_\mathrm{b})$ entry in A. This nonzero entry is immediately annihilated by applying a Givens rotation $G_{2n_\mathrm{b},2n_\mathrm{b}+1}$ from the left. If we now apply G_{n_b,n_b-1}, only one nonzero entry at position $(2n_\mathrm{b}, n_\mathrm{b} - 1)$ is created in A. Again this entry is immediately annihilated, by applying a Givens rotation $G_{2n_\mathrm{b}-1,2n_\mathrm{b}}$ from the left. After all Givens rotations G_{n_b+1,n_b}, G_{n_b,n_b-1},$\dots$, $G_{3,2}$ have been processed in the described manner, we obtain a new sequence of Givens rotations $G_{2n_\mathrm{b},2n_\mathrm{b}+1}$, $G_{2n_\mathrm{b}-1,2n_\mathrm{b}}$, $\dots$, $G_{n_\mathrm{b}+2,n_\mathrm{b}+3}$. To complete the transformation we need to apply this new sequence from the left to B. It can be seen in Figure 3 (b) that proceeding as described effectively pushes the nonzero subdiagonal entries of B by n_b entries downwards. By repeating the process the nonzero subdiagonal entries are pushed further down, until they eventually reach the bottom right corner of B; see Figure 3 (c) and (d). In the last step, all subdiagonal entries can be annihilated without causing any fill-in in A. Columns 2, $3, \dots, n-2$ are reduced in an analogous manner.

Assuming $n_\mathrm{b} \ll n$, the number of flops needed for performing Stage 2 can be found in the following table.

Update of A	Update of B	Update of Q	Update of Z
$\frac{3(n_\mathrm{b}-1)}{n_\mathrm{b}}n^3$	$\frac{3(n_\mathrm{b}-1)}{n_\mathrm{b}}n^3$	$\frac{3(n_\mathrm{b}-1)}{n_\mathrm{b}}n^3$	$\frac{3(n_\mathrm{b}-1)}{n_\mathrm{b}}n^3$

For comparison, the following table contains the costs in flops needed by Algorithm 1.

Update of A	Update of B	Update of Q	Update of Z
$5n^3$	$3n^3$	$3n^3$	$3n^3$

Stage 2 requires less operations than Algorithm 1 for updating A because part of the reduction was already performed in Stage 1. The modified Stage 1 and Stage 2 together require approximately a total amount of

$$\frac{10p+5}{3(p-1)}n^3 + 3\frac{2p+1}{p-1}n^3 + 4\frac{3(n_\mathrm{b}-1)}{n_\mathrm{b}}n^3 \approx \frac{64}{3}n^3$$

flops. Compared with the $14n^3$ flops needed by Algorithm 1, the cost is therefore increased by $32/21 \approx 52\%$.

On the other hand, this increase can be compensated by the use of level 3 BLAS in Stage 1. Also, two strategies are proposed in [6] to increase data locality in Stage 2:

1. Right after reducing the first column and pushing the corresponding $n_\mathrm{b}-1$ nonzero subdiagonal elements in B by n_b steps, the fully updated second column is reduced. This creates another set of $n_\mathrm{b}-1$ nonzero subdiagonal elements in B. In total we have now a chain of $2n_\mathrm{b}-2$ such elements, which are chased simultaneously to the bottom right corner.

   It is straightforward to generalize this technique to reduce more than two columns simultaneously, but in the experiments in [6], the application of two concurrent reductions was found to be (nearly) optimal.

2. The updates of A and B are restricted to a few, say $n_\mathrm{c} = 32$, consecutive columns at a time. The sines and cosines of previously applied Givens rotations are stored to enable delayed updates of the other columns.

More details on the implementation of these two strategies can be found in [6]. Instead of Givens rotations, one could use tiny Householder reflectors of order, say, 3 or 4. Although this reduces the number of flops needed by Stage 2, the actual impact on the execution time was found to be too marginal to justify the increased complexity of the implementation [13].

3 One-stage algorithms based on Givens rotations

Although the two-stage algorithm utilizes the memory hierarchy much better than Algorithm 1 does, with asymptotically 44% of the operations performed via calls to level 3 BLAS, its significant increase of flops is dissatisfying and tempts us to reconsider Algorithm 1 in the form of the following two variants.

3.1 Level 1 BLAS variant with increased data locality

In the second part of the two-stage algorithm, we reduce all the nonzero entries in a column of A by a sequence of Givens rotations before updating the rest of A. The same technique can be applied to rearrange the computations in Algorithm 1, leading to Algorithm 2. The LAPACK subroutine `DLASR` is a straightforward implementation of the functions `row_givens` and `col_givens` used in Algorithm 2. In practice, this subroutine needs to be specialized in order to handle the upper triangular structure of B.

We next propose to use two strategies with the goal of increasing data locality in the accesses performed in Algorithm 2.

Algorithm 2 HT reduction using sequences of Givens rotations

Input: A general matrix $A \in \mathbb{R}^{n\times n}$ and an upper triangular matrix $B \in \mathbb{R}^{n\times n}$.
Output: Orthogonal matrices $Q, Z \in \mathbb{R}^{n\times n}$ such that $(H, T) = (Q^T AZ, Q^T BZ)$ is in HT form. The matrices A and B are overwritten by H and T, respectively.
Remark: `row_givens` and `col_givens` apply a sequence of Givens rotations to the rows and columns of a matrix, respectively.

Set $Q \leftarrow I_n,\ Z \leftarrow I_n$.
for $j \leftarrow 1, 2, \ldots, n-2$ **do**
  Construct a sequence of Givens rotations $G_{n-1,n}, \ldots, G_{j+1,j+2}$ to reduce $A(j+2:n, j)$.
  `row_givens`$(G_{n-1,n}, \ldots, G_{j+1,j+2}, A)$.
  `row_givens`$(G_{n-1,n}, \ldots, G_{j+1,j+2}, B)$.
  `col_givens`$(G_{n-1,n}, \ldots, G_{j+1,j+2}, Q)$.
  Construct a sequence of Givens rotations $G_{n,n-1}, \ldots, G_{j+2,j+1}$ to annihilate the subdiagonal fill-in in B.
  `col_givens`$(G_{n,n-1}, \ldots, G_{j+2,j+1}, B)$.
  `col_givens`$(G_{n,n-1}, \ldots, G_{j+2,j+1}, A)$.
  `col_givens`$(G_{n,n-1}, \ldots, G_{j+2,j+1}, Z)$.
end for

3.1.1 Increasing data register reuse

The application of a Givens rotation to a pair of vectors of length m (e.g., rows or columns of a matrix) is an operation that requires $6m$ flops for a total of $4m$ accesses to memory (each entry of both vectors needs to be read and written back to memory). This yields a ratio of flops to memory accesses (hereafter, *memops*) of $6/4 = 1.5$, which explains the low performance of this operation on current architectures, with a large gap between the floating-point performance of the processor and the latency of memory.

Now, given that we are applying the Givens rotations in the sequence $G_{n-1,n}, \ldots, G_{j+1,j+2}$ to consecutive rows of a matrix, we can improve the ratio of flops to memops by applying several of these rotations simultaneously. For example, consider the application of two rotations defined by the sine and cosine arguments c_1, s_1, c_2, s_2 to three consecutive rows of a matrix, $a_1^T, a_2^T, a_3^T \in \mathbb{R}^{1\times m}$, as follows:

$$\left[\begin{array}{cc|c} c_2 & s_2 & \\ -s_2 & c_2 & \\ \hline & & 1 \end{array}\right] \left[\begin{array}{c|cc} 1 & & \\ \hline & c_1 & s_1 \\ & -s_1 & c_1 \end{array}\right] \left[\begin{array}{c} a_1^T \\ a_2^T \\ a_3^T \end{array}\right].$$

Then, provided the sine and cosine arguments remain in four registers of the processor and two more registers are available for temporary variables, the update of the three rows requires $12m$ flops for a total of $6m$ memops yielding the more favourable ratio of 2 flops to memop.

The technique can be easily generalized to the simultaneous application of ρ Givens rotations to a block of $\rho + 1$ vectors, delivering a ratio of $6\rho/2(\rho + 1)$ flops to memops. Note that the improvement of this ratio quickly flattens as ρ increases. On architectures with a small number of registers (e.g., Intel x86), this limits the practical values to $\rho = 2$ or $\rho = 3$. However, on other architectures with more registers (e.g., Intel Itanium2), larger values of ρ attain higher performance; see Section 4.

$$\begin{array}{l}
G^{(1)}_{12,13} \to G^{(1)}_{11,12} \to G^{(1)}_{10,11} \mapsto G^{(1)}_{9,10} \to G^{(1)}_{8,9} \to G^{(1)}_{7,8} \to G^{(1)}_{6,7} \mapsto G^{(1)}_{5,6} \to G^{(1)}_{4,5} \to G^{(1)}_{3,4} \to G^{(1)}_{2,3} \mapsto \\
G^{(2)}_{12,13} \to G^{(2)}_{11,12} \mapsto G^{(2)}_{10,11} \to G^{(2)}_{9,10} \to G^{(2)}_{8,9} \to G^{(2)}_{7,8} \mapsto G^{(2)}_{6,7} \to G^{(2)}_{5,6} \to G^{(2)}_{4,5} \to G^{(2)}_{3,4} \mapsto \\
G^{(3)}_{12,13} \mapsto G^{(3)}_{11,12} \to G^{(3)}_{10,11} \to G^{(3)}_{9,10} \to G^{(3)}_{8,9} \mapsto G^{(3)}_{7,8} \to G^{(3)}_{6,7} \to G^{(3)}_{5,6} \to G^{(3)}_{4,5} \mapsto \\
G^{(4)}_{12,13} \to G^{(4)}_{11,12} \to G^{(4)}_{10,11} \to G^{(4)}_{9,10} \mapsto G^{(4)}_{8,9} \to G^{(4)}_{7,8} \to G^{(4)}_{6,7} \to G^{(4)}_{5,6}
\end{array}$$

Figure 4: Sequence of Givens rotations used to reduce the first 4 columns of A for $n = 13$.

3.1.2 Reducing cache misses

For matrices stored columnwise (with a leading dimension larger than the size of the cache lines), accessing the entries of a matrix by rows is disadvantageous in that it preempts hardware prefetch. In particular, consider the application of the sequence of Givens rotations $G_{n-1,n}, \dots, G_{j+1,j+2}$ performed in function `row_givens` and, for simplicity, assume that the technique described previously to increase data register reuse is not employed. Then, just after having applied $G_{n-1,n}$ to the last two rows of A, the cache is likely filled with elements from the last columns of the matrix. Unfortunately, these elements are useless during the earlier stages of the application of the next rotation $G_{n-2,n-1}$. Furthermore, by the time the last columns are needed, they have been overwritten in the cache by the first columns of the matrix leading to *cache trashing*.

We can reduce the effects of row accesses by updating the matrices panelwise by blocks of columns as follows. Assume the sequence of Givens rotations $G_{n-1,n},\ G_{n-2,n-1}, \dots, G_{j+1,j+2}$ is applied to A (or B) so that the entries in a *panel* (column block) of width $n_{\rm c}$ are completely updated before proceeding with the next column block. Then, if $n_{\rm c}$ is chosen small relative to the cache size, by the time we are done with $G_{n-1,n}$ those elements in the first columns of the current panel still lie in the cache and can be rapidly brought into the processor registers during the application of $G_{n-2,n-1},\ G_{n-3,n-2}, \dots, G_{j+1,j+2}$.

3.2 Level 3 BLAS variant with increased data locality

The data locality of Algorithm 2 can be further increased by regrouping and accumulating Givens rotations. The technique described in the following has been proposed by Lang [15] in the context of the QR algorithm for symmetric tridiagonal matrices.

To illustrate the basic idea of regrouping, let $G^{(j)}_{i-1,i}$ denote the Givens rotation used to annihilate the (i, j) element of A. Figure 4 illustrates the sequence of Givens rotations used in Algorithm 2 to annihilate the elements below the first subdiagonals of the first $n_{\rm b} = 4$ columns of a 13×13 matrix A. A trivial but far-reaching observation is that two non-intersecting rotations $G^{(j_1)}_{i_1-1,i_1}$ and $G^{(j_2)}_{i_2-1,i_2}$ commute if $i_2 > i_1 - 1$. For example, in the given sequence, this allows to commute $G^{(2)}_{12,13}$, $G^{(2)}_{11,12}$, and $G^{(3)}_{12,13}$ subsequently with the complete sequence $G^{(1)}_{2,3} \to G^{(1)}_{3,4} \to \cdots \to G^{(1)}_{9,10}$. The resulting sequence

$$G^{(1)}_{12,13} \to G^{(1)}_{11,12} \to G^{(2)}_{12,13} \to G^{(2)}_{11,12} \to G^{(3)}_{12,13} \tag{5}$$

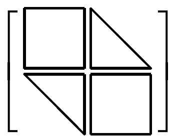
is more localized than the original start sequence. In a similar fashion we can regroup the rest of the transformations in stripes of $n_{\rm b} = 4$ antidiagonals, as indicated by the dashed lines in Figure 4, leading to the three sequences illustrated in Figure 5.

$$
\begin{array}{l}
G^{(1)}_{12,13} \rightarrow G^{(1)}_{11,12} \rightarrow G^{(1)}_{10,11} \rightarrow G^{(2)}_{12,13} \rightarrow G^{(2)}_{11,12} \rightarrow G^{(3)}_{12,13} \rightarrow \\
\hdashline
G^{(1)}_{9,10} \rightarrow G^{(1)}_{8,9} \rightarrow G^{(1)}_{7,8} \rightarrow G^{(1)}_{6,7} \rightarrow G^{(2)}_{10,11} \rightarrow G^{(2)}_{9,10} \rightarrow G^{(2)}_{8,9} \rightarrow G^{(2)}_{7,8} \rightarrow \\
G^{(3)}_{11,12} \rightarrow G^{(3)}_{10,11} \rightarrow G^{(3)}_{9,10} \rightarrow G^{(3)}_{8,9} \rightarrow G^{(4)}_{12,13} \rightarrow G^{(4)}_{11,12} \rightarrow G^{(4)}_{10,11} \rightarrow G^{(4)}_{9,10} \rightarrow \\
\hdashline
G^{(1)}_{5,6} \rightarrow G^{(1)}_{4,5} \rightarrow G^{(1)}_{3,4} \rightarrow G^{(1)}_{2,3} \rightarrow G^{(2)}_{6,7} \rightarrow G^{(2)}_{5,6} \rightarrow G^{(2)}_{4,5} \rightarrow G^{(2)}_{3,4} \rightarrow \\
G^{(3)}_{7,8} \rightarrow G^{(3)}_{6,7} \rightarrow G^{(3)}_{5,6} \rightarrow G^{(3)}_{4,5} \rightarrow G^{(4)}_{8,9} \rightarrow G^{(4)}_{7,8} \rightarrow G^{(4)}_{6,7} \rightarrow G^{(4)}_{5,6} \\
\hdashline
\end{array}
$$

Figure 5: Regrouped sequence of Givens rotations.

An additional benefit of regrouping is that we can efficiently apply each regrouped sequence in terms of a matrix-matrix multiplication. For example, the product of all rotations in the second sequence from Figure 5, $G^{(1)}_{9,10} \rightarrow G^{(1)}_{8,9} \rightarrow \cdots \rightarrow G^{(4)}_{9,10}$, takes the form $\begin{bmatrix} I_5 & 0 \\ 0 & U \end{bmatrix}$, where

$$U = \begin{bmatrix} U_{11} & U_{12} \\ U_{21} & U_{22} \end{bmatrix} = \quad (6)$$



that is, $U_{21} \in \mathbb{R}^{4\times 4}$ and $U_{12} \in \mathbb{R}^{4\times 4}$ are triangular. For general n_b, the matrix U is $2n_\mathrm{b} \times 2n_\mathrm{b}$ with upper/lower off-diagonal triangular blocks $U_{21}, U_{12} \in \mathbb{R}^{n_\mathrm{b}\times n_\mathrm{b}}$. (An exception is the first regrouped sequence (5), for which U is smaller and U_{21} is a full matrix.) Exploiting the structure of the intermediate matrices, the overall accumulation of U requires $6n_\mathrm{b}^3 + O(n_\mathrm{b}^2)$ flops; see also [15].

Storing the sines and cosines of the Givens rotations $G^{(j)}_{i-1,i}$ allows us to update the matrix Q in Algorithm 2 every n_b outer loops by means of regrouped sequences of Givens rotations or matrix-matrix multiplications. The following table compares the number of flops needed for applying one regrouped sequence.

Givens rotations	Full matrix multiply	Triangular matrix multiply
$6n_\mathrm{b}^2 n$	$8n_\mathrm{b}^2 n$	$(6n_\mathrm{b}^2 + 2n_\mathrm{b})n$

The last column refers to the case when the triangular block structure of U in (6) is exploited during the matrix-matrix multiplication. Whether the corresponding reduction in the flop count leads to an actual reduction of execution time strongly depends on the implementation of the BLAS `DGEMM` (general matrix multiply and add) and `DTRMM` (triangular matrix multiply). Typically, n_b is not much larger than 64 for which the performance of `DGEMM` and `DTRMM` is often suboptimal or erratical; see, e.g., [9]. Note that a similar issue appears in multishift variants of the QR and QZ algorithms [4, 12, 14]. Our implementation offers both variants of multiplying U.

The results from this section apply likewise to the sequences of Givens rotations used for annihilating the fill-in subdiagonal elements of B. In particular, the orthogonal matrix Z can be updated via calls to level 3 BLAS.

3.3 Panel-wise reduction

In the following, we describe a panel-wise reduction that allows to perform the updates of A and B partly by means of matrix-matrix multiplications. For this purpose, let us assume that j_c outer loops of Algorithm 1 have been performed and partition the partially reduced matrix A and the matrix B as

$$A = \begin{array}{c} \\ j_c{+}1 \\ n{-}j_c{-}1 \end{array} \begin{array}{c} \begin{array}{ccc} j_c & n_b & n{-}j_c{-}n_b \end{array} \\ \left[\begin{array}{ccc} A_{11} & A_{21} & A_{31} \\ 0 & A_{22} & A_{23} \end{array}\right] \end{array}, \quad B = \begin{array}{c} \\ j_c{+}1 \\ n{-}j_c{-}1 \end{array} \begin{array}{c} \begin{array}{cc} j_c{+}1 & n{-}j_c{-}1 \end{array} \\ \left[\begin{array}{cc} B_{11} & B_{12} \\ 0 & B_{22} \end{array}\right] \end{array}.$$

During the subsequent reduction of the n_b columns of the *panel* (or block) A_{22}, the updates of A and B are restricted to the minimum. That is, only the blocks A_{22} and B_{22} (but *not* A_{23}) are immediately updated by transformations from the left and only the blocks $[A_{22}, A_{23}]$ and B_{22} (but *not* $[A_{21}, A_{31}]$ and B_{12}) are immediately updated by transformations from the right. After the complete panel has been reduced, all employed Givens rotations are accumulated into $2n_b \times 2n_b$ orthogonal matrices as explained in Section 3.2. This allows to perform the remaining updates of A and B by matrix-matrix multiplications; see also Figure 6. The

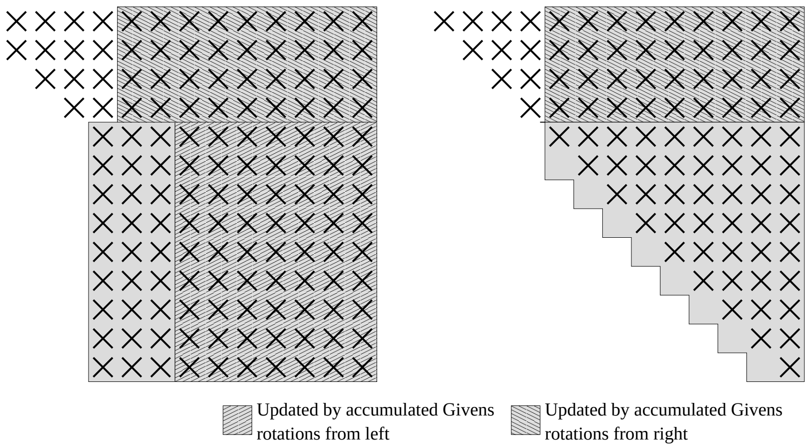


Figure 6: Illustration of panel-wise factorization for $j_c = 3$, $n_b = 3$.

complete procedure can be found in Algorithm 3. If the block triangular structure (6) is fully exploited during the accumulation and application of the matrices U, the flops required by Algorithm 3 are as follows.

Update of A	Update of B	Update of Q	Update of Z
$5n^3 + O(n_b n^2)$	$3n^3 + O(n_b n^2)$	$3n^3 + O(n_b n^2)$	$3n^3 + O(n_b n^2)$

Hence as $n_b/n \to 0$, Algorithms 1 and 3 have the same computational cost. However, Algorithm 3 performs asymptotically 50% of the operations for updating A, B and 100% of the operations for generating Q, Z via calls to level 3 BLAS.

3.4 Accumulation of orthogonal transformation matrices

When the generalized eigenvalue problem $(A - \lambda B)x = 0$ is to be solved and the orthogonal factors are desired, the Givens rotations applied during Algorithm 1 are typically accumulated into an orthogonal matrix Q_0 resulting from an initial QR decomposition of B and into $Z_0 = I_n$. By carefully exploiting the structure of Z while reducing the pair (A, B) to HT

Algorithm 3 Panel-wise HT reduction

Input: A general matrix $A \in \mathbb{R}^{n\times n}$ and an upper triangular matrix $B \in \mathbb{R}^{n\times n}$. Block size n_b.
Output: Orthogonal matrices $Q,\ Z \in \mathbb{R}^{n\times n}$ such that $(H,T) = (Q^TAZ, Q^TBZ)$ is in HT form. The matrices A and B are overwritten by H and T, respectively.

Set $Q \leftarrow I_n,\ Z \leftarrow I_n$.
for $j_\mathrm{c} \leftarrow 0, n_\mathrm{b}, 2n_\mathrm{b}, \ldots$ **do**
  $n_\mathrm{nb} \leftarrow \lfloor (n - j_\mathrm{c})/n_\mathrm{b} \rfloor - 1, \quad n_\mathrm{bt} = n - j_\mathrm{c} - n_\mathrm{b}n_\mathrm{nb} + 1.$
  Initialize $U_1 \leftarrow I_{2n_\mathrm{b}}, \ldots, U_{n_\mathrm{nb}} \leftarrow I_{2n_\mathrm{b}}, U_{n_\mathrm{nb}+1} \leftarrow I_{n_\mathrm{bt}}$.
  for $j \leftarrow j_\mathrm{c}+1, j_\mathrm{c}+2, \ldots, j_\mathrm{c}+n_\mathrm{b}$ **do**
    Reduce $A(j+1:n, j)$ by Givens rotations sequence $G_{n-1,n}\cdots G_{j+1,j+2}$.
    Accumulate $\begin{cases} G_{n-1,n}\cdots G_{n-n_\mathrm{bt}+j-j_\mathrm{c},n-n_\mathrm{bt}+j-j_\mathrm{c}+1} \text{ into } U_{n_\mathrm{nb}+1}; \\ G_{j_\mathrm{c}+kn_\mathrm{nb}-1,j_\mathrm{c}+kn_\mathrm{nb}+1}\cdots G_{j_\mathrm{c}+(k-1)n_\mathrm{nb},j_\mathrm{c}+(k-1)n_\mathrm{nb}+1} \text{ into } U_k \text{ for } k = n_\mathrm{nb},\ldots,2,1. \end{cases}$
    $G_{n-1,n}\cdots G_{j+1,j+2}$ into $U_{n_\mathrm{nb}+1}, U_{n_\mathrm{nb}}, \ldots, U_1$.
    `row_givens`$(G_{n-1,n}\cdots G_{j+1,j+2}, B)$.
    Construct Givens rotations sequence $G^{(j)}_{n,n-1}\cdots G^{(j)}_{j+2,j+1}$ to annihilate subdiagonal fill-in in B.
    `col_givens`$(G^{(j)}_{n,n-1}\cdots G^{(j)}_{j+2,j+1}, A(j_\mathrm{c}+2:n,:))$.
    `col_givens`$(G^{(j)}_{n,n-1}\cdots G^{(j)}_{j+2,j+1}, B(j_\mathrm{c}+2:n,:))$.
    if $j < j_\mathrm{c}+n_\mathrm{b}$ **then**
      Multiply $U_{n_\mathrm{nb}+1}, U_{n_\mathrm{nb}}, \ldots, U_1$ from the left to update $A(j_\mathrm{c}+2:n, j+1)$.
    end if
  end for
  Multiply $U_{n_\mathrm{nb}+1}, U_{n_\mathrm{nb}}, \ldots, U_1$ from the left to update $A(j_\mathrm{c}+2:n, j_\mathrm{c}+n_\mathrm{b}+1:n)$.
  Multiply $U_{n_\mathrm{nb}+1}, U_{n_\mathrm{nb}}, \ldots, U_1$ from the right to update Q.
  Initialize $U_1 \leftarrow I_{2n_\mathrm{b}}, \ldots, U_{n_\mathrm{nb}} \leftarrow I_{2n_\mathrm{b}}, U_{n_\mathrm{nb}+1} \leftarrow I_{n_\mathrm{bt}}$.
  for $j \leftarrow j_\mathrm{c}+1, j_\mathrm{c}+2, \ldots, j_\mathrm{c}+n_\mathrm{b}$ **do**
    Accumulate $\begin{cases} G^{(j)}_{n,n-1}\cdots G^{(j)}_{n-n_\mathrm{bt}+j-j_\mathrm{c},n-n_\mathrm{bt}+j-j_\mathrm{c}+1} \text{ into } U_{n_\mathrm{nb}+1}; \\ G^{(j)}_{j_\mathrm{c}+kn_\mathrm{nb}+1,j_\mathrm{c}+kn_\mathrm{nb}-1}\cdots G^{(j)}_{j_\mathrm{c}+(k-1)n_\mathrm{nb}+1,j_\mathrm{c}+(k-1)n_\mathrm{nb}} \text{ into } U_k \text{ for } k = n_\mathrm{nb},\ldots,2,1. \end{cases}$
  end for
  Multiply $U_{n_\mathrm{nb}+1}, U_{n_\mathrm{nb}}, \ldots, U_1$ from the right to update $A(1:j_\mathrm{c},:)$.
  Multiply $U_{n_\mathrm{nb}+1}, U_{n_\mathrm{nb}}, \ldots, U_1$ from the right to update $B(1:j_\mathrm{c},:)$.
  Multiply $U_{n_\mathrm{nb}+1}, U_{n_\mathrm{nb}}, \ldots, U_1$ from the right to update Z.
end for

form via Algorithm 1 (and the two variants that have been proposed), the cost of updating this matrix can be reduced from $3n^3$ to just n^3 flops, yielding a reduction of the overall algorithm from $14n^3$ to $12n^3$ flops. This is a trivial observation which is exploited, e.g., in the construction of the orthogonal factor resulting from a QR decomposition in subroutine `DORGQR` from LAPACK. However, the current implementation of the LAPACK subroutine `DGGHRD` does not exploit this structure leading to unnecessary overhead.

Note that in the two-stage algorithm described in Section 2, similar savings can only be attained in the first stage. After the first stage, the factor Z is almost fully populated so that there is little opportunity to exploit this in the second stage.

4 Numerical experiments

In this section, we show the performance benefits attained by the following algorithms/variants for HT reduction, compared to the LAPACK implementation `DGGHRD`:

Platform	Architecture	Frequency (GHz)	L2 cache (KBytes)	L3 cache (MBytes)	RAM (GBytes)
ATHLON	AMD Athlon	1.66	256	–	1
ITANIUM	Intel Itanium2	1.5	256	4096	4
OPTERON	AMD Opteron	2.2	1024	–	8
PENTIUM	Intel Pentium4	3.2	2048	–	1

Table 1: Architectures employed in the experimental evaluation.

Platform	Compiler	Optimization flags	BLAS	Operating System
ATHLON	`Portland F90` 6.0	`-O4 -fast`	ATLAS BLAS 3.5.9	Debian/Linux 3.0
ITANIUM	`icc` 9.0	`-O3`	GotoBLAS 1.06	Linux 2.4.21
OPTERON	`Portland F90` 6.0	`-O4 -fastsse`	GotoBLAS 0.94	Debian/Linux 3.1
PENTIUM	`icc` 9.0	`-O3`	GotoBLAS 1.06	Linux 2.6.13

Table 2: Software employed in the experimental evaluation.

– `DK`: Prototype implementation of Dackland and Kågström's two-stage blocked algorithm from [6].

– `MDK`: Modified prototype implementation of Dackland and Kågström's two-stage blocked algorithm from [6] with Stage 1 replaced with the new procedure described in Section 2.2.

– `GB1`: New level 1 BLAS variant of Algorithm 1 that incorporates the techniques introduced in Section 3.1 to both increase data register reuse and reduce cache misses.

– `GB3`: New level 3 BLAS variant of Algorithm 1 with regrouped and accumulated Givens rotations presented in Section 3.2 and described in Algorithm 3.

Four different platforms, ATHLON, ITANIUM, OPTERON, and PENTIUM are employed in the experimental evaluation of the algorithms, representative of current desktop platforms. For details on the hardware and software of these platforms, see Tables 1 and 2, respectively. All experiments on these processors were carried out using double-precision floating-point arithmetic.

4.1 Performance dependence on block parameters

The performance of the considered algorithms/variants can be tuned for a particular architecture by various parameters, see Table 3. In the following experiments, no exhaustive search was made to tune these values. Typical values that provided high performance were $n_\mathrm{b}, n_\mathrm{c} \in \{16, 24, 32, 48, 64\}$, depending on algorithm/variant and the problem size, $p \in \{2, 3\}$, and $\rho = 3$.

4.2 Performance comparison

In the following, we report the speed-ups obtained for algorithms/variants `DK`, `MDK`, `GB1`, and `GB3` with respect to the LAPACK subroutine `DGGHRD` which implements Algorithm 1.

Algorithm/ variant	Parameter	Purpose
DK/MDK	n_b	Order of the subdiagonal blocks in the block HT form
	p	#Blocks simultaneously reduced
	n_c	#Columns the application of the updates is restricted to
GB1	ρ	#Givens rotations simultaneously applied
	n_c	#Columns the application of the row updates is restricted to
GB3	n_b	#Givens rotations regrouped and accumulated
	n_c	#Columns the application of the row updates is restricted to

Table 3: Parameters which affect the performance of the presented algorithms/variants.

Figures 7–10 summarize the observed speed-ups on the four platforms for the HT reduction with and without accumulation of the orthogonal factors. Note that the initial zero structure of Z (see Section 3.4) is exploited in `GB1` and `GB3` but not in `DGGHRD`, `DK`, and `MDK`.

The results show that variant `GB3` outperforms all other algorithms/variants on ATHLON and OPTERON except for the smallest problem sizes. On these two platforms, `GB3` also outperforms subroutine `DGGHRD` (speed-up higher than 1) for problems of dimension starting at 150–250.

The results are completely different on ITANIUM. There, the clear winner is `GB1` in all except one case: orthogonal factors computed for problem size n=1400. It is necessary to note here that the architecture of the ITANIUM processor is quite different from those of the remaining three processors. While the ITANIUM is a VLIW processor with a large number of registers (compared with the x86 platforms) and three levels of cache memory, ATHLON, OPTERON, and PENTIUM are superscalar architectures with a more reduced number of registers and only two levels of cache. In particular, the larger number of registers on ITANIUM allowed us to improve data register reuse during the application of Givens rotations in variant `GB1` using larger values for ρ. The performance improvements over subroutine `DGGHRD` are remarkable on this architecture.

On the PENTIUM, `GB1` is the best option for problem sizes up to n=500 when the orthogonal factors are not needed. In all other cases, `GB3` is the the variant to choose.

5 Conclusions

We have presented two new variants of Moler and Stewart's algorithm for Hessenberg-triangular reduction which exhibit better locality in the data access. One of the variants also performs a significant part of the computations via efficient calls to level 3 BLAS.

Experimental results on four different processor architectures show that the new variants are significantly faster than subroutine `DGGHRD` in LAPACK, a prototype implementation of Dackland and Kågström's two-stage approach, and a modified version of Dackland and Kågström's algorithm with a more reduced cost contributed in this paper. Supported by these results, we suggest to replace the current implementation of subroutine `DGGHRD` in LAPACK by a combined implementation of the newly developed algorithms.

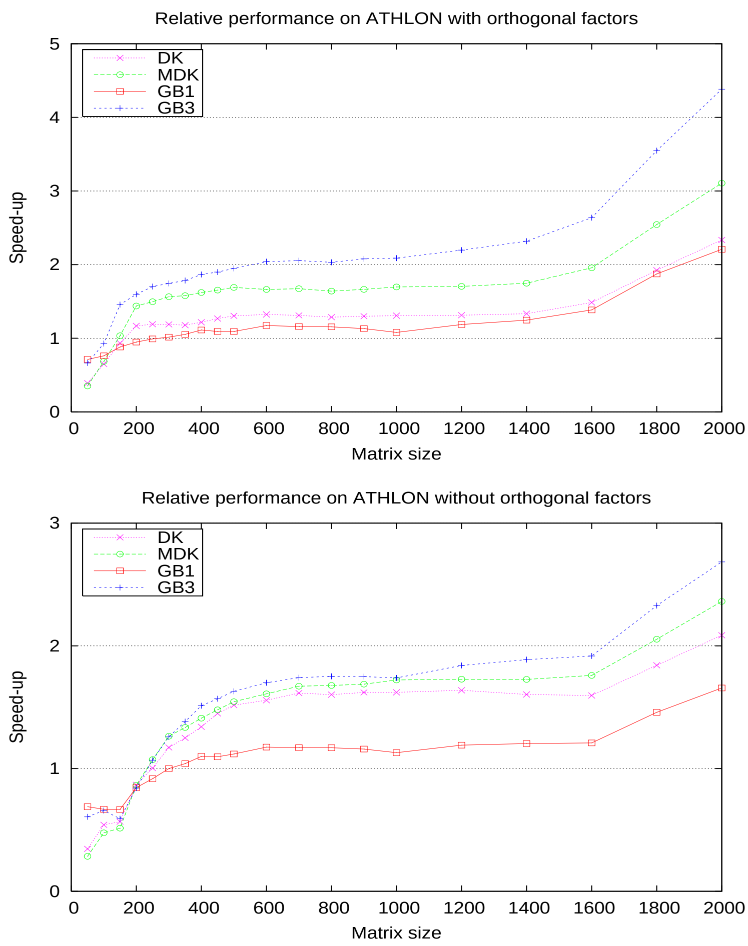


Figure 7: Performance benefits attained by the algorithms/variants for the HT reduction on ATHLON.

Acknowledgments

Bo Kågström and Daniel Kressner were supported by the *Swedish Research Council* under grant VR 621-2001-3284 and by the *Swedish Foundation for Strategic Research* under grant SSF A3 02:128. Additionally, Daniel Kressner was supported by the DFG Emmy Noether fellowship KR 2950/1-1.

Enrique S. Quintana-Ortí and Gregorio Quintana-Ortí were supported by the CICYT project TIN2005-09037-C02-02 and FEDER.

References

[1] E. Anderson, Z. Bai, C. H. Bischof, S. Blackford, J. W. Demmel, J. J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. C. Sorensen. *LAPACK Users' Guide*. SIAM, Philadelphia, PA, third edition, 1999.

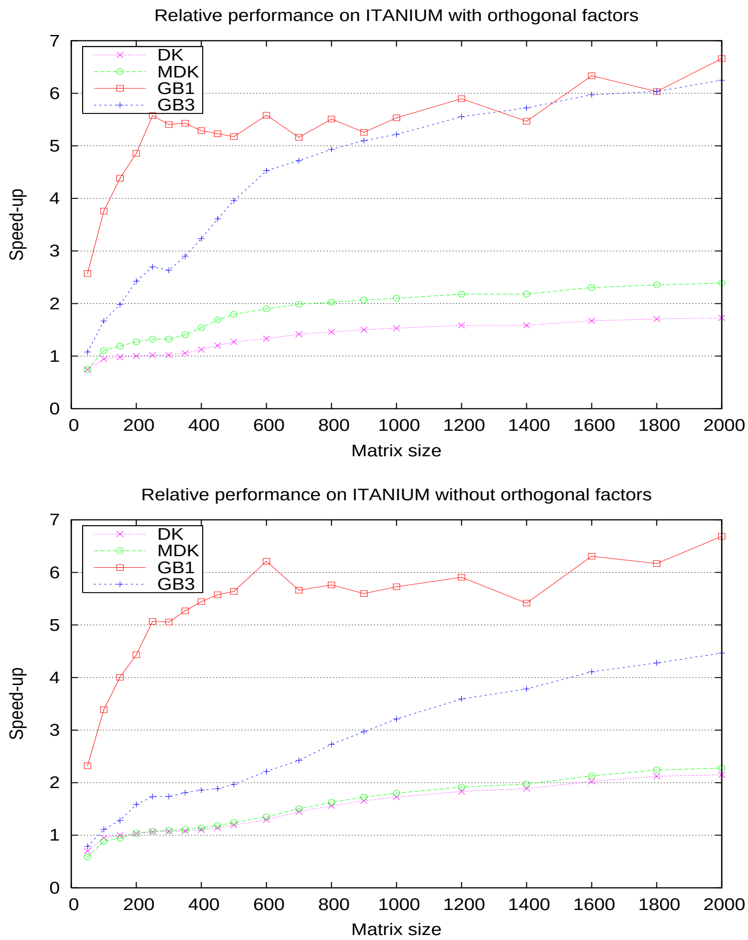


Figure 8: Performance benefits attained by the algorithms/variants for the HT reduction on ITANIUM.

[2] C. H. Bischof, B. Lang, and X. Sun. A framework for symmetric band reduction. *ACM Trans. Math. Software*, 26(4):581–601, 2000.

[3] C. H. Bischof and C. F. Van Loan. The WY representation for products of Householder matrices. *SIAM J. Sci. Statist. Comput.*, 8(1):S2–S13, 1987.

[4] K. Braman, R. Byers, and R. Mathias. The multishift QR algorithm. I. Maintaining well-focused shifts and level 3 performance. *SIAM J. Matrix Anal. Appl.*, 23(4):929–947, 2002.

[5] M. Cosnard, J.-M. Muller, and Y. Robert. Parallel QR decomposition of a rectangular matrix. *Numer. Math.*, 48(2):239–249, 1986.

[6] K. Dackland and B. Kågström. Blocked algorithms and software for reduction of a regular matrix pair to generalized Schur form. *ACM Trans. Math. Software*, 25(4):425–454, 1999.

[7] J. J. Dongarra, D. C. Sorensen, and S. J. Hammarling. Block reduction of matrices to condensed forms for eigenvalue computations. *J. Comput. Appl. Math.*, 27(1-2):215–227,

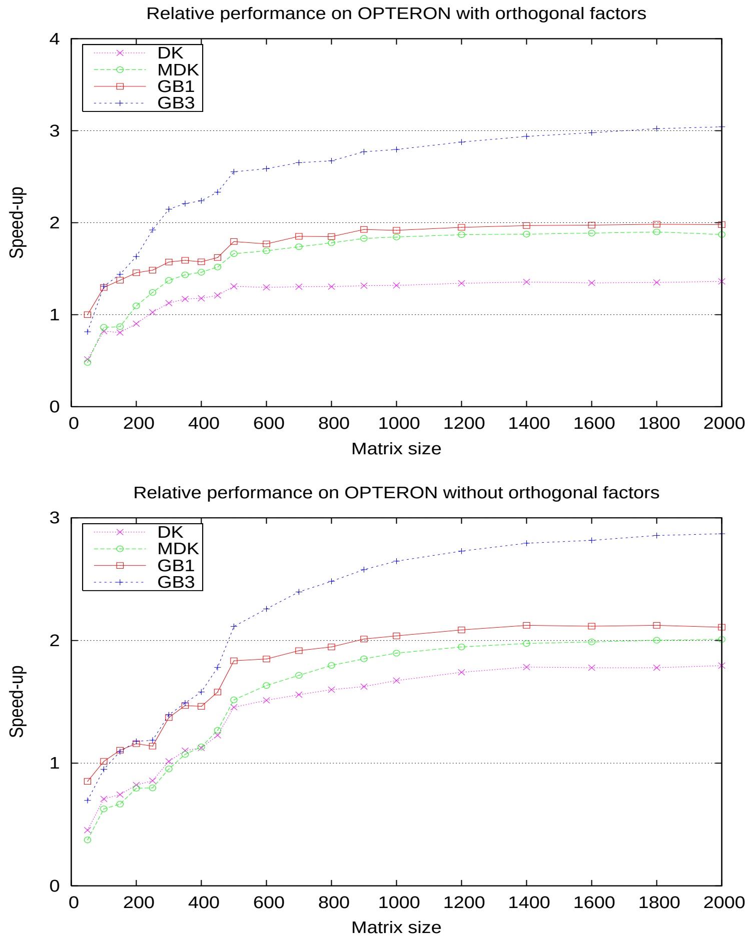


Figure 9: Performance benefits attained by the algorithms/variants for the HT reduction on OPTERON.

1989. Reprinted in *Parallel algorithms for numerical linear algebra*, 215–227, North-Holland, Amsterdam, 1990.

[8] G. H. Golub and C. F. Van Loan. *Matrix Computations.* Johns Hopkins University Press, Baltimore, MD, third edition, 1996.

[9] K. Goto and R. van de Geijn. High-performance implementation of the level-3 BLAS. Technical Report TR-2006-23, The University of Texas at Austin, Department of Computer Sciences, 2006.

[10] B. Kågström, P. Ling, and C. F. Van Loan. GEMM-based level 3 BLAS: Algorithms for the model implementations. *ACM Trans. Math. Software*, 24(3):268–302, 1999.

[11] B. Kågström, P. Ling, and C. F. Van Loan. GEMM-based level 3 BLAS: High-performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Software*, 24(3):303–316, 1999.

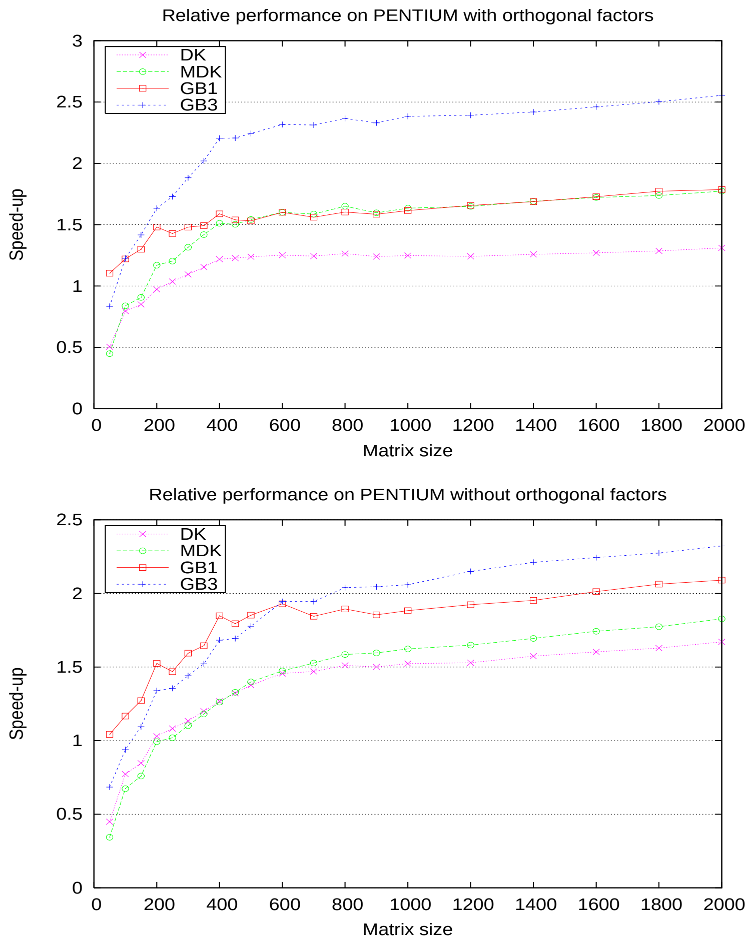


Figure 10: Performance benefits attained by the algorithms/variants for the HT reduction on PENTIUM.

[12] B. Kågström and D. Kressner. Multishift variants of the QZ algorithm with aggressive early deflation. *SIAM J. Matrix Anal. Appl.*, 29(1):199–227, 2006. Also appeared as LAPACK working note 173.

[13] D. Kressner. *Numerical Methods and Software for General and Structured Eigenvalue Problems*. PhD thesis, TU Berlin, Institut für Mathematik, Berlin, Germany, 2004.

[14] B. Lang. Effiziente Orthogonaltransformationen bei der Eigen- und Singulärwertzerlegung. Habilitationsschrift, 1997.

[15] B. Lang. Using level 3 BLAS in rotation-based algorithms. *SIAM J. Sci. Comput.*, 19(2):626–634, 1998.

[16] H. P. Märchy. On a modification of the QZ algorithm with fast Givens rotations. *Computing*, 38(3):247–259, 1987.

[17] J. J. Modi and M. R. B. Clarke. An alternative Givens ordering. *Numer. Math.*, 43(1):83–90, 1984.

[18] C. B. Moler and G. W. Stewart. An algorithm for generalized matrix eigenvalue problems. *SIAM J. Numer. Anal.*, 10:241–256, 1973.

[19] G. Quintana-Ortí and R. A. van de Geijn. Improving the performance of reduction to Hessenberg form. *ACM Trans. Math. Software*, 32(2):180–194, 2006.

[20] H. Rutishauser. On Jacobi rotation patterns. In *Proc. Sympos. Appl. Math., Vol. XV*, pages 219–239. Amer. Math. Soc., Providence, R.I., 1963.

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

[22] H.-R. Schwarz. Die Reduktion einer symmetrischen Bandmatrix auf tridiagonale Form. *Z. Angew. Math. Mech.*, 45:T75–T77, 1965.