SIAM J. SCI. COMPUT.
Vol. 41, No. 1, pp. C52–C72

EFFICIENT REDUCTION OF BANDED HERMITIAN POSITIVE DEFINITE GENERALIZED EIGENVALUE PROBLEMS TO BANDED STANDARD EIGENVALUE PROBLEMS*

BRUNO LANG†

Abstract. We present a method for reducing the generalized eigenvalue problem $Ax = Bx\lambda$ with banded hermitian matrices A, B, and B positive definite to an equivalent standard eigenvalue problem $Cy = y\lambda$, such that C again is banded. Our method combines ideas of an algorithm proposed by Crawford in 1973 and LAPACK's reduction routines _{SY,HE}GST and retains their respective advantages, namely, being able to rely on matrix–matrix operations (Crawford) and to handle split factorizations and different bandwidths b_A and b_B (LAPACK). In addition, it includes two algorithmic parameters (block size, n_b, and width of blocked orthogonal transformations, w) that can be adjusted to optimize performance. We also present a heuristic for automatically determining suitable values for these parameters. Numerical experiments confirm the efficiency of our new method.

Key words. banded positive definite generalized eigenvalue problem, reduction to standard eigenvalue problem, bulge chasing, split factorization, blocked algorithm, performance model

AMS subject classifications. 65F15, 65-04

DOI. 10.1137/18M1167322

1. Introduction.

Generalized eigenvalue problems (GEPs)

$$AX = BX\Lambda \tag{1.1}$$

with hermitian matrices $A, B \in \mathbb{C}^{n\times n}$ and B positive definite arise in many areas, e.g., in electronic structure computations based on Hartree–Fock [12, 20] or Kohn–Sham theory [15] and in dynamic response analysis with a consistent mass matrix [2]. Depending on the application, all or only a subset of k eigenvalues may be required. These are contained in the diagonal matrix $\Lambda \in \mathbb{R}^{k\times k}$, and the columns of $X \in \mathbb{C}^{n\times k}$ are associated B-orthogonal eigenvectors.

If A and B are almost dense then an established procedure for solving (1.1) transforms the GEP into a standard eigenvalue problem (SEP). To this end, one first determines a Cholesky decomposition $B = LL^H$ with a lower triangular matrix L. Here, L^H denotes the conjugate transpose of L, that is, $(L^H)_{i,j} = \overline{L_{j,i}}$ for $i, j = 1, \ldots, n$. For reasons explained toward the end of this section, in this work we will instead consider a decomposition $B = L^H L$, which is obtained by starting Cholesky's method at B's bottom-right corner. Then multiplying (1.1) with L^{-H} gives

$$L^{-H}AL^{-1} \cdot LX = L^{-H}BL^{-1} \cdot LX \cdot \Lambda = LX \cdot \Lambda,$$

and therefore solving the hermitian SEP

$$CY = Y\Lambda \qquad \text{with} \quad C := L^{-H}AL^{-1} \text{ and } Y := LX \tag{1.2}$$

*Submitted to the journal's Software and High-Performance Computing section January 29, 2018; accepted for publication (in revised form) November 29, 2018; published electronically February 19, 2019.

http://www.siam.org/journals/sisc/41-1/M116732.html

Funding: This work was supported by the Federal Ministry of Education and Research within the project "Eigenvalue Solvers for Petaflop Applications - Algorithmic Extensions and Optimizations" under grant 01IH15001.

†Mathematics and Natural Sciences, University of Wuppertal, Wuppertal, D-42119, Germany (lang@math.uni-wuppertal.de).

and transforming back to the needed eigenvectors via $X := L^{-1}Y$ yields the solution of (1.1). (With straightforward modifications this approach also works with other decompositions, e.g., $B = LL^H$, and for real symmetric matrices.)

Depending on the couplings described by the matrices A and B and the numbering of the variables, the matrices may in addition be banded with (semi)bandwidths b_A and b_B, respectively. Then the matrix L is also (lower) banded with bandwidth b_B, but the triangular matrix L^{-1}—and therefore also C—is full, in general, and, therefore, a full hermitian standard eigensolver must be used for (1.2), leading to $\mathcal{O}(n^3)$ overall arithmetic operations and $\mathcal{O}(n^2)$ memory requirements. By contrast, $\mathcal{O}(n^2 b_C)$ operations and $\mathcal{O}(nb_C)$ memory would be sufficient for computing just the eigenvalues *if C were banded*. For computing eigenvalues *and* eigenvectors the gain from preserving the banded structure would be somewhat lower.

In 1973, Crawford [6] proposed a new method for the real symmetric case with $b_B = b_A$. It reduces the matrix A to a matrix $\widehat{C}$ that is orthogonally similar to C—and in particular has the same eigenvalues as C and (A, B)—but is again banded with (semi)bandwidth b_A. The method is based on a decomposition of the matrices into $b_A \times b_A$ blocks and on a "bulge chasing" scheme that removes intermediate fill-in before it can spread over the whole matrix. Accordingly, the back transformation $y \mapsto x$ of $\widehat{C}$'s eigenvectors to those of (A, B) now includes the orthogonal transformations from the bulge chasing, together with the inversion of L. The extension of the method to the complex hermitian case is straightforward. In section 2 we will present Crawford's approach in a more general setting.

Starting with Release 2.0, the LAPACK library [1] includes routines for the (complex hermitian and real symmetric in single and double precision) reduction of the banded GEP, named {C,Z}HBGST and {S,D}SBGST, respectively. They differ from Crawford's original method in three main points. First, they do not rely on a block decomposition, but focus on single matrix entries. The chasing is done with rotations, and a reordering of these allows efficient vectorization, in particular, for small bandwidths; see [13]. (Later, a hybrid scheme was proposed [14] that increases the vector length also for larger bandwidths; see also our discussion in subsection 2.4 and section 4. However, this scheme has not yet been adopted in LAPACK.) Second, the bandwidths of the matrices may differ, provided that $b_B \leq b_A$, and the operation count is roughly proportional to b_B instead of b_A. Finally, the Cholesky decomposition is replaced with a "split factorization" $B = S^H S$ [22], where for some split position p, the leading $p \times p$ submatrix is upper triangular with bandwidth b_B and the lower $n-p$ rows have nonzeros only in the first b_B subdiagonals; see Figure 1.1 for an example. In section 2 we will see that this roughly halves the arithmetic complexity by allowing one to chase each bulge to the nearer end of the matrix. Split factorizations have been known for some time; in numerical libraries, their use can be traced back at least to the LINPACK routines _PTSL [8, section 7].

In this work we present a reduction algorithm that combines features of these two approaches, retaining their respective advantages. Like Crawford's original algorithm, it proceeds by blocks, thus allowing one to do almost all operations as matrix–matrix operations. Similarly to the LAPACK routines ___GST, it can also handle (and benefit from) different bandwidths $b_B < b_A$ and uses a split factorization $B = S^H S$. (This factorization can be obtained with LAPACK's _PBSTF, which we have patched to allow any split position $p \in \{0, \ldots, n\}$. In particular, $p = 0$ yields the factorization $L^H L$ that is used to present the simpler nonsplit case.) Our algorithm includes two parameters, the block size n_b and the width w of the blocked orthogonal transformations in the

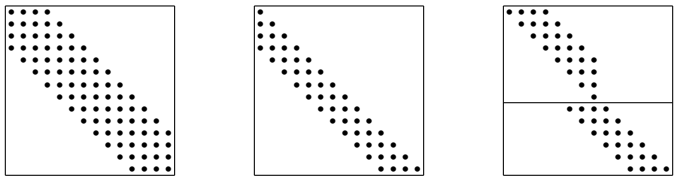

FIG. 1.1. *Nonzero patterns of the matrix B (left), the Cholesky factor L (middle), and the "split" factor S (right) for $n = 14$, $b_B = 3$, $p = 8$.*

bulge chasing, that can be chosen independently from the bandwidths and can be used to optimize the performance.

The remainder of the article is organized as follows. In section 2 we present the reduction algorithm, starting with a nonsplit factorization and a high-level description. Following this, we discuss the algorithm at a close-to-implementation level, including its extension to split factorizations and modifications for optimizing its performance and memory requirements. In section 3 we present a heuristic for automatically selecting suitable values for the algorithmic parameters n_b and w. The results of numerical experiments are reported in section 4, and section 5 summarizes our findings.

Throughout the article we will assume that the split factorization $B = S^H S$ has been computed beforehand, e.g., with the LAPACK routine {D,S,Z,C}PBSTF. We also assume $b_B > 0$ because otherwise $S^{-1} = \mathrm{diag}(1/\sqrt{b_{11}}, \ldots, 1/\sqrt{b_{nn}})$ is a real diagonal matrix and the computation $C = S^{-H} A S^{-1}$ is just a two-sided scaling.

2. The reduction algorithm. To simplify the presentation we will first consider the "nonsplit" case $p = 0$, i.e., S is a lower triangular banded matrix L that has been obtained by running Cholesky's algorithm from the lower end upward.

2.1. The reduction with a decomposition $B = L^H L$. Given an arbitrary block size $n_b > 0$, the algorithm partitions L into $N = \lceil n/n_b \rceil$ block rows of height n_b. To simplify the index computations we first assume that all block rows have equal size, $n = N \cdot n_b$ (otherwise the first or last block row will be smaller). Then

$$L = L_1 \cdots L_N,$$

where L_ℓ is the identity matrix, except for the $n_b \times n_b$ diagonal block $L_{\ell\ell}$ and the $n_b \times b_B$ subdiagonal block $L_{\ell,\ell-1}$ from L; see Figure 2.1. Therefore

$$L^{-1} = L_N^{-1} \cdots L_1^{-1},$$

where L_ℓ^{-1} has a similar structure, with $n_b \times n_b$ diagonal block $D_\ell := L_{\ell\ell}^{-1}$ (lower triangular) and $n_b \times b_B$ subdiagonal block $E_\ell := -D_\ell \cdot L_{\ell,\ell-1}$ (full). Thus the matrix C in (1.2) might be obtained in N *inversion steps*

$$A =: A_N \;\mapsto\; L_N^{-H} A_N L_N^{-1} =: A_{N-1} \;\mapsto\; \cdots \;\mapsto\; L_1^{-H} A_1 L_1^{-1} = C.$$

Letting

$$\begin{cases} J \equiv j_1 : j_2 := (\ell - 1) n_b + 1 : \ell n_b, \\ I \equiv i_1 : i_2 := j_1 - b_B : j_1 - 1 \end{cases} \tag{2.1}$$

denote the index ranges of the diagonal and subdiagonal block in L_ℓ, right multiplication with L_ℓ^{-1} and left multiplication with L_ℓ^{-H} affect only the $b_B + n_b$ columns and

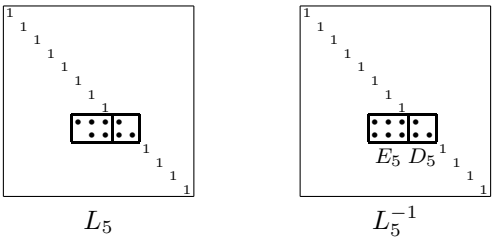

FIG. 2.1. *Factor L_5 containing the fifth block row of L (left) and its inverse L_5^{-1} (right) for $n = 14$, $b_B = 3$, $n_B = 2$.*

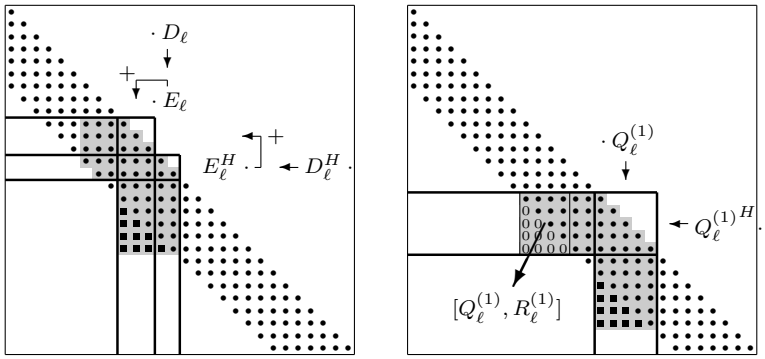

FIG. 2.2. *Inversion step $A \mapsto L_\ell^{-H} A L_\ell^{-1}$ according to* (2.2) *(left) and first step of bulge chasing (right) for $n = 28$, $b_A = 6$, $b_B = 3$, $n_b = 2$, and $\ell = 7$. Only the entries with grey background shading are changed, filled squares denote fill-in, and the "0" entries are zeroed.*

rows $i_1 : j_2$ of the matrix:

$$\begin{cases} A(:,I) := A(:,I) + A(:,J) \cdot E_\ell, \\ A(:,J) := A(:,J) \cdot D_\ell, \\ A(I,:) := A(I,:) + E_\ell^H \cdot A(J,:), \\ A(J,:) := D_\ell^H \cdot A(J,:). \end{cases} \tag{2.2}$$

The left picture in Figure 2.2 shows the situation for $\ell = 7$, $b_B = 3$, $n_b = 2$, and a matrix of size $n = 28$, $b_A = 6$. Since A's symmetry is preserved throughout the reduction, only one triangle of the matrix (here the lower one) is updated.

The inversion step typically generates an $(n_b + b_B - 1) \times (n_b + b_B - 1)$ triangle of new nonzero elements outside the band, the so-called "*bulge*." If this bulge is not removed then its size will grow by n_b with each of the following inversion steps, leading to an almost full matrix. (In fact, Figure 2.2 assumes that the bulges from the preceding steps $N, \ldots, \ell+1$ already have been removed. Otherwise the lower triangle would have filled up completely in columns n through $i_1 = (\ell - 1)n_b + 1 - b_B$.)

The bulge can be removed with a unitary transformation $A \mapsto {Q_\ell^{(1)}}^H A Q_\ell^{(1)}$, where $Q_\ell^{(1)}$ is obtained from a QR decomposition of the $(n_b + b_B) \times (n_b + b_B - 1)$ submatrix containing the bulge and one additional row above it; see the right picture in Figure 2.2. Applying this transformation leads to a new $(n_b+b_B-1)\times(n_b+b_B-1)$ bulge, b_A columns to the right of the original one. Again, this bulge can be chased by another b_A columns to the right with a transformation $Q_\ell^{(2)}$, and so on, until the lower

end of the band is reached and no more fill-in is created. Then the banded structure of A has been restored and the next inversion step with $L_{\ell-1}^{-1}$ can be carried out.

The final matrix obtained by this procedure is given by

$$\widehat{C} = \widehat{L}^{-H} \cdot A \cdot \widehat{L}^{-1},$$

where

$$\widehat{L}^{-1} = L_N^{-1} Q_N^{(1)} \cdots Q_N^{(\nu_N)} L_{N-1}^{-1} Q_{N-1}^{(1)} \cdots Q_{N-1}^{(\nu_{N-1})} \cdots L_1^{-1} Q_1^{(1)} \cdots Q_1^{(\nu_1)},$$

and ν_ℓ is the number of chasing steps following inversion step ℓ. (For the situation depicted in Figure 2.2, $b_A = 6$, $b_B = 3$, $n_b = 2$, steps N and $N-1$ do not require bulge chasing. More precisely, $\nu_N > 0$ iff $n_b + b_B > b_A + 1$.)

In contrast to L^{-1} from (1.2), $\widehat{L}^{-1}$ is not lower triangular. Since we are always chasing downward whereas the later inversion steps $L_{\ell-1}^{-1}, \ldots, L_1^{-1}$ progress upward, the columns affected by the $Q_\ell^{(\cdot)}$ from the ℓth chasing are disjoint from those affected by $L_{\ell-1}^{-1}, \ldots, L_1^{-1}$, and therefore these steps can be interchanged, and the procedure is equivalent to first doing all inversion steps and then applying all the $Q_\cdot^{(\cdot)}$,

$$\widehat{L}^{-1} = L_N^{-1} \cdots L_1^{-1} \cdot \underbrace{Q_N^{(1)} \cdots Q_N^{(\nu_N)} \cdots Q_1^{(1)} \cdots Q_1^{(\nu_1)}}_{=: \widehat{Q}} = L^{-1} \widehat{Q}.$$

This implies that the obtained matrix $\widehat{C}$ is unitarily similar to the matrix C in (1.2) and therefore has the same eigenvalues as (1.1).

If the eigenvectors of (1.1) are required as well then they can be obtained from those of $\widehat{C}$ via $X := \widehat{L}^{-1}\widehat{Y}$. There are at least three strategies to do this.

(I) Explicitly build $Z := \widehat{L}^{-1}$ by also applying the transformations from all the inversion and chasing steps to the columns of the identity matrix. Such "on-the-fly" accumulation is done in several LAPACK band reduction routines, e.g., when calling `___GST` with `VECT='V'`. Then either
  (Ia) continue to accumulate the transformations that yield the eigenvectors $\widehat{Y}$ in this matrix (this is typical, e.g., with QR-type eigensolvers) or
  (Ib) compute the eigenvectors $\widehat{Y}$ and do a matrix–matrix multiplication.

  Alternatively, one can

(II) store the transformations L_ℓ^{-1} and $Q_\ell^{(\cdot)}$, compute the eigenvectors $\widehat{Y}$, and then apply the transformations to $\widehat{Y}$ (from the left, *in reverse order*).

The latter approach (back-transformation of the eigenvectors) is taken in the ELPA library [18]; it tends to be superior if only a subset of the eigenvectors is needed because it avoids building the $n \times n$ matrix Z. In this work we will follow strategy (I) and build the matrix Z.

So far we have given an overview of the reduction algorithm based on a nonsplit factorization, and the pictures are reflecting the situation $n_b \le b_B$ and $n_b + b_B \le b_A$. In the following two subsections we will give more details on the computations and discuss the impact of removing these constraints.

2.2. More details on the inversion and chasing steps. An implementation of the reduction algorithm will not operate on complete block rows and block columns of the banded matrix A but on subblocks. This enables updating only one triangle of the matrix and using a more compact storage scheme. While the above restrictions $n_b \le b_B$ and $n_b + b_B \le b_A$ are not necessary for the correctness of the reduction

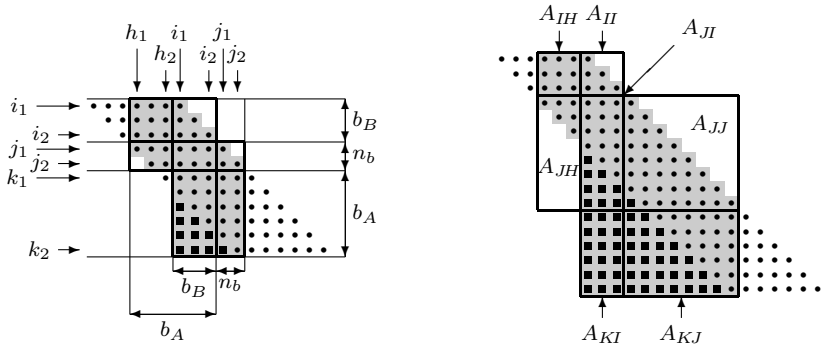

FIG. 2.3. *The seven blocks that must be updated in each inversion step and their index range for $b_A = 6$, $b_B = 3$. The left picture introduces the lowest and highest index of each block (on the left and top) and shows the default block sizes (on the right and bottom) for $n_b = 2$. The right picture shows the blocks in a different situation ($n_b = 8$), using the shorthands $A_{IJ} \equiv A(i_1 : i_2, j_1 : j_2)$, etc.*

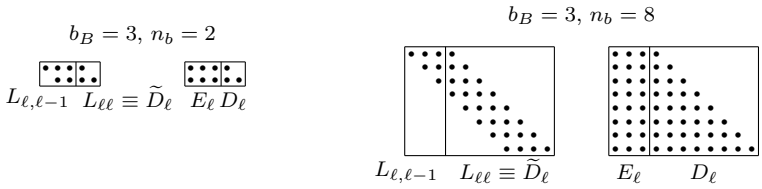

FIG. 2.4. *Structure of the blocks $L_{\ell,\ell-1}$, $L_{\ell\ell}$, D_ℓ, and E_ℓ for the cases $n_b < b_B$ (left) and $n_b \geq b_B$ (right).*

algorithm, they do influence the size and structure of the subblocks. In the following we discuss this by looking more closely at individual inversion and chasing steps.

As pointed out in subsection 2.1, the inversion step $A \mapsto L_\ell^{-H} A L_\ell^{-1}$ affects only rows and columns i_1 to j_2 of the matrix A (cf. (2.1)), and only the lower triangle is actually used and updated. Some of the entries in these rows and columns are affected by both transformations (L_ℓ^{-H} from the left, L_ℓ^{-1} from the right), others undergo only one transformation, and some remain unchanged. Figure 2.3 gives a more detailed view on these entries and introduces the indices for the relevant subblocks of the matrix; note the shorthand $A_{IH} \equiv A(I, H) \equiv A(i_1 : i_2, h_1 : h_2)$ and similarly for the other subblocks, and that the shape of the subblocks depends on the values of b_A, b_B, and n_b. Close to the upper end of the band, the H and I blocks may become smaller than their default sizes indicated in the left picture of Figure 2.3 (or even become empty), and at the lower end this holds for the K blocks. In addition, there are no H blocks if $b_B = b_A$. If the matrix size is not a multiple of n_b then the J blocks in the first or last inversion step will be smaller, but J is the only set that cannot be empty.

The pseudocode given in Algorithm 2.1 reflects these cases. A few comments are in order.

Lines 6 to 9 (together with the current j_2) set up the blocks depicted in Figure 2.3 for the current inversion step. Line 6 ensures that all except for the first inversion steps operate with J blocks of maximum size $|J| = n_b$.

Lines 10 and 11: The blocks $\widetilde{D} \equiv L_{\ell\ell}$ are lower triangular, and if $n_b > b_B$ then they are also banded with bandwidth b_B (see the right picture in Figure 2.4), whereas their inverses $D_\ell = L_{\ell\ell}^{-1}$ from (2.2) would be full in any case. Instead of forming these explicitly, multiplication with D_ℓ is done via triangular solves with $\widetilde{D}$, for example,

Algorithm 2.1 Transformation $A \mapsto \widehat{C}$ for the case $p = 0$ (i.e., S is a lower banded matrix L).

1. **if** the transformation matrix $Z = \widehat{L}^{-1}$ is needed
2. $\quad Z := I_n$
3. $\mathrm{row}_{\min} := 1$; $\quad \mathrm{col}_{\min} := 1$
4. $j_2 := n$
5. **while** $j_2 \geq \mathrm{row}_{\min}$

{ *Determine block ranges for inversion step; typical sizes are* $|H| = b_A - b_B$, $|I| = b_B$, $|J| = n_b$, $|K| = b_A$ }

6. $\quad j_1 := \mathrm{row}_{\min} + \lfloor (j_2 - \mathrm{row}_{\min})/n_b \rfloor \cdot n_b$; $\quad J := j_1 : j_2$
7. $\quad i_1 := \max\{j_1 - b_B, \mathrm{col}_{\min}\}$; $\quad i_2 := j_1 - 1$; $\quad I := i_1 : i_2$
8. $\quad h_1 := \max\{j_1 - b_A, 1\}$; $\quad h_2 := i_1 - 1$; $\quad H := h_1 : h_2$
9. $\quad k_1 := j_2 + 1$; $\quad k_2 := \min\{j_2 + b_A, n\}$; $\quad K = k_1 : k_2$

{ *Set up nontrivial blocks of* L_ℓ^{-1} }

10. $\quad$ Copy lower $|J| \times |J|$ triangle L_{JJ} into buffer $\widetilde{D}$
11. $\quad E := -D \cdot L_{JI}$ { $D \equiv \widetilde{D}^{-1}$: _TRSM $|J| \times |J| \times |I|$ *with* $\widetilde{D}$ }

{ *Update the blocks of* A *according to* (2.2) *and Figure* 2.3 }

12. $\quad A_{JI,\mathrm{old}} := A_{JI}$ { $|J| \times |I|$ *buffer* }
13. $\quad A_{IH} := A_{IH} + E^H \cdot A_{JH}$ { _GEMM $|I| \times \min\{|J|, b_A - |I|\} \times |H|$ }
14. $\quad A_{JI} := A_{JI} + A_{JJ} \cdot E$ { _{SY,HE}MM $|J| \times |J| \times |I|$ }
15. $\quad A_{KI} := A_{KI} + A_{KJ} \cdot E$ { _GEMM $|K| \times |J| \times |I|$ }
16. $\quad M := A_{JI,\mathrm{old}}^H \cdot E + E^H \cdot A_{JI}$ { $2 \times$ _GEMM $|I| \times |J| \times |I|$ }
17. $\quad A_{II} := A_{II} + M$ { *update only lower triangle* }
18. $\quad [A_{JH}, A_{JI}] := D^H \cdot [A_{JH}, A_{JI}]$ { _TRSM $|J| \times |J| \times (|H| + |I|)$ *with* $\widetilde{D}$ }
19. $\quad A_{KJ} := A_{KJ} \cdot D$ { _TRSM $|K| \times |J| \times |J|$ *with* $\widetilde{D}$ }
20. $\quad M := A_{JJ}$ { *hermitian* $|J| \times |J|$ *matrix; upper triangle via transposition* }
21. $\quad M := D^H \cdot M$; $\quad M := M \cdot D$ { $2 \times$ _TRSM $|J| \times |J| \times |J|$ *with* $\widetilde{D}$ }
22. $\quad A_{JJ} := M$ { *copy back only the lower triangle* }

{ *Accumulate transformation matrix* $Z = \widehat{L}^{-1}$ }

23. $\quad$ **if** Z is needed
24. $\quad\quad Z(:, I) := Z(:, I) + Z(:, J) \cdot E$ { _GEMM $n \times |J| \times |I|$ }
25. $\quad\quad Z(:, J) := Z(:, J) \cdot D$ { _TRSM $n \times |J| \times |J|$ *with* $\widetilde{D}$ }

{ *Bulge chasing* }

26. $\quad$ Call Algorithm 2.2

{ *Go to next inversion step* }

27. $\quad j_2 := j_1 - 1$

in line 11. We always use _TRSM even if $\widetilde{D}$ is banded. We also do not make use of the zeros in the subdiagonal block $L_{\ell,\ell-1} \equiv L_{JI}$, which is nonzero only in its upper $b_B \times b_B$ triangle if $n_b \geq b_B$, and upper trapezoidal if $n_b < b_B$; see Figure 2.4.

In line 13 we make use of the fact that only the first $b_A - |I|$ rows of A_{JH} can contain nonzero entries. However, we do not exploit all the zeros in A_{JH}. Additional savings would be possible if an optimized _GEMM/_TRMM-like routine for adding the product of a general and a triangular matrix to another matrix were available.

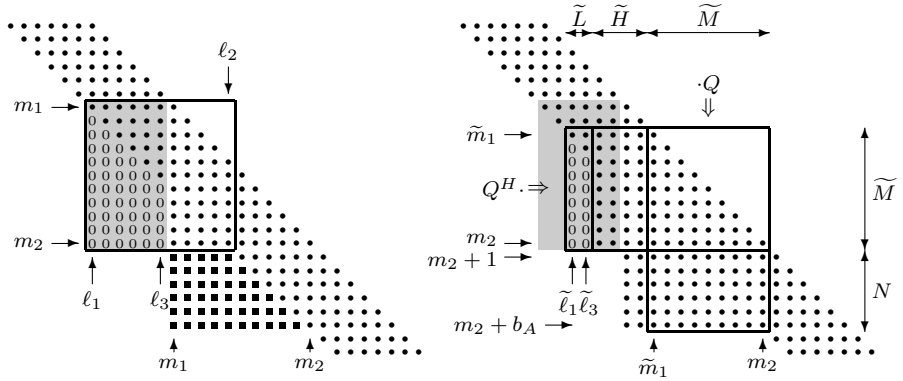

FIG. 2.5. *(Left) Data affected by one chasing step.* • *indicates nonzero entries, the "0" entries are made zero, and the* ■ *entries are fill-in created by this step. The submatrix used for the QR decomposition is shaded gray. (Right) Index ranges of the blocks that must be transformed for the second chunk ($c = 1$) if the chasing step is done in $n_{\mathrm{QR}} = 3$ chunks of width $w = 2$. Both pictures assume $b_A = 6$, $b_B = 3$, and $n_b = 8$.*

Lines 16 and 17: According to (2.2), the diagonal block A_{II} is updated twice, $A_{II} := A_{II} + A_{IJ} \cdot E_\ell$ and $A_{II} := A_{II} + E_\ell^H \cdot A_{JI}$. The first update is done with the original $A_{IJ} \equiv A_{JI}^H$ (saved in line 12), whereas the A_{JI} in the second update already has been updated by the second line in (2.2), $A_{JI} := A_{JI} + A_{JJ} \cdot E_\ell$ (the current block from line 14). This overall rank-$2 \cdot \min\{n_b, b_B\}$ update of A_{II} might be implemented without an auxiliary $n_b \times n_b$ buffer for a matrix M, but not with the available level-3 BLAS routines [9, 10].

Lines 20 to 22: The update $A_{JJ} := D_\ell^H \cdot A_{JJ} \cdot D_\ell$ again might be implemented such that it uses only the lower triangle of A_{JJ} and the triangular matrix $\widetilde{D}_\ell$, similarly to the LAPACK routines _{SY,HE}GST. We cannot use the latter because our $\widetilde{D}_\ell$ comes from a bottom-up Cholesky factorization. Instead, we use two triangular solves and a full $n_b \times n_b$ auxiliary matrix M.

Only nonempty blocks are updated; e.g., lines 11 to 17 are skipped if $I = \emptyset$. These checks have been omitted in the pseudocode.

A detailed description of the bulge chasing is given in Algorithm 2.2. It requires some explanations.

The $(n_b + b_B) \times (n_b + b_B)$ submatrix $A(m_1 : m_2, \ell_1 : \ell_2)$ contains the current bulge in its strict lower triangle; see Figure 2.5 for all indices and subblocks discussed in the following. The initial position of the bulge is set in lines 1 and 2, and after the chasing step the position is updated in line 22, until the bulge disappears (cf. **while** loop in line 3). Close to the lower end of the band the bulge will become smaller; this is accounted for in lines 4 and 5.

Essentially, one bulge chasing step consists of (i) determining a QR factorization of the block $A(m_1 : m_2, \ell_1 : \ell_2) \mapsto (Q^{(k)}, R^{(k)})$ and overwriting that block with $R^{(k)}$, (ii) applying $Q^{(k)}$ from the left to $A(m_1 : m_2, \ell_2+1 : m_1-1)$, from both sides to (the lower triangle of) $A(m_1 : m_2, m_1 : m_2)$ and from the right to $A(m_2+1 : m_2+b_A, m_1 : m_2)$, and, optionally, (iii) updating columns $m_1 : m_2$ of the transformation matrix Z. Since most of the overall work is spent in bulge chasing, our implementation comprises a few modifications of this scheme to make it more efficient.

First, the last column of the square block needs no treatment, and if $n_b + b_B > b_A$ then it makes no sense to eliminate more than b_A columns because the bulge moves only b_A positions and some of the newly introduced zeros would be destroyed

Algorithm 2.2 Bulge chasing; i_1 and j_2 are taken from Algorithm 2.1.

1. $\ell_1 := i_1$; $\ell_2 := j_2$
2. $m_1 := \ell_1 + b_A$; $m_2 := \ell_2 + b_A$
3. **while** $m_1 < n$
   { *Clip indices at the matrix boundaries* }
4. **if** $m_2 > n$
5. $\ell_2 := \ell_2 - (m_2 - n)$; $m_2 := n$
6. **if** $m_2 > m_1$
7. $\ell_3 := \min\{\ell_2 - 1, \ell_1 + b_A - 1\}$
   { *Do a QR decomposition of* $A(m_1 : m_2, \ell_1 : \ell_3)$ *by chunks of width* w *and apply resulting unitary transformations* Q }
8. $\widetilde{\ell}_1 := \ell_1$; $\widetilde{m}_1 := m_1$
9. **while** $\widetilde{\ell}_1 \le \ell_3$
   { *Typical block sizes for chunk* $c = 0, 1, \ldots$ *are* $|\widetilde{L}| = w$, $|\widetilde{M}| = n_b + b_B - c \cdot w$, $|\widetilde{H}| = b_A - w$, $|N| = b_A$ }
10. $\widetilde{\ell}_3 := \min\{\widetilde{\ell}_1 + w - 1, \ell_3\}$; $\widetilde{L} := \widetilde{\ell}_1 : \widetilde{\ell}_3$
11. $\widetilde{M} := \widetilde{m}_1 : m_2$
12. $\widetilde{H} := \widetilde{l}_3 + 1 : \widetilde{m}_1 - 1$
13. $N := m_2 + 1 : \min\{m_2 + b_A, n\}$
14. $[Q, R] :=$ QR factorization of $A_{\widetilde{ML}}$ { *_GEQRF* $|\widetilde{M}| \times |\widetilde{L}|$; *overwrite* $A_{\widetilde{ML}}$ *with* R }
15. Build W and Y factors for Q { *_GEWYG* $|\widetilde{M}| \times |\widetilde{L}|$ }
16. $A_{\widetilde{MH}} := Q^H \cdot A_{\widetilde{MH}}$ { *_GEWY or _ORMQR* $|\widetilde{L}| \times |\widetilde{M}| \times |\widetilde{H}|$ }
17. $A_{\widetilde{MM}} := Q^H \cdot A_{\widetilde{MM}} \cdot Q$ { *_{SY,HE}WY* $|\widetilde{M}| \times |\widetilde{M}| \times |\widetilde{L}|$ }
18. $A_{N\widetilde{M}} := A_{N\widetilde{M}} \cdot Q$ { *_GEWY or _ORMQR* $|N| \times |\widetilde{M}| \times |\widetilde{L}|$ }
19. **if** Z is needed
20. $Z(:, \widetilde{M}) := Z(:, \widetilde{M}) \cdot Q$ { *_GEWY or _ORMQR* $n \times |\widetilde{M}| \times |\widetilde{L}|$ }
    { *Go to next QR chunk* }
21. $\widetilde{\ell}_1 := \widetilde{\ell}_1 + w$; $\widetilde{m}_1 := \widetilde{m}_1 + w$
    { *Go to next chasing step,* b_A *positions down the band* }
22. $\ell_1 := \ell_1 + b_A$; $\ell_2 := \ell_2 + b_A$; $m_1 := m_1 + b_A$; $m_2 := m_2 + b_A$

immediately by the application of $Q^{(k)}$ from the right; cf. the left picture in Figure 2.5. Thus the QR decomposition is determined only for the block $A(m_1 : m_2, \ell_1 : \ell_3)$, where $\ell_3 = \min\{l_2 - 1, \ell_1 + b_A - 1\}$, and the remaining columns are included in the left update, $A(m_1 : m_2, \ell_3 + 1 : m_1 - 1) := {Q^{(k)}}^H \cdot A(m_1 : m_2, \ell_3 + 1 : m_1 - 1)$.

Second, the application of $Q^{(k)}$ could be done with the LAPACK routine _ORMQR, but this approach has two drawbacks. This routine achieves BLAS 3 performance via the so-called *compact WY representation* [21] for products of Householder transformations. Since the triangular factors of these representations are not returned by the routine and thus cannot be saved, the same factors would be built at least three times for the updates of the different blocks. In addition, there is no _ORMQR-type routine for the two-sided update of the diagonal block; instead, two consecutive one-sided transformations might be performed, similarly to lines 20–22 in Algorithm 2.1. The

symmetry of the diagonal block cannot be fully utilized this way, leading to a higher operations count. To circumvent these problems, we also provide the option to perform the transformations with routines from the SBR toolbox [4, 5]: _GEWYG explicitly builds the W factor for the "original" *WY representation* [3], $Q = I - WY^H$, and then this factor is used in the one-sided and two-sided updates with _GEWY and _{SY,HE}WY, respectively. The latter makes full use of symmetry and does not require an additional buffer for the block. However, while the matrix Y just contains the Householder vectors from the QR decomposition and therefore is lower trapezoidal, W is full, and thus the number of operations in the WY updates can also be substantially higher than for unblocked application of the Householder transformations, even if the zeros in Y are fully exploited. To alleviate this problem, the QR decomposition and the ensuing block updates are done in n_{QR} chunks of a prescribed width w; cf. the right picture in Figure 2.5 and lines 8 to 21 in Algorithm 2.2. The block boundaries and ranges that depend on the current chunk are marked with a tilde. With small values of w, the overhead caused by the filled-up Ws becomes negligible. On the other hand, w is the smallest dimension in the matrix–matrix products occurring in the multiplication with $Q = I - WY^H$, and therefore the _GEMM performance deteriorates if w is chosen too small. We will come back to suitable choices of w in section 3.

Again, only nonempty blocks are transformed. That is, line 16 (line 18) of Algorithm 2.2 is executed only if $\widetilde{H} \neq \emptyset$ ($N \neq \emptyset$).

To close this discussion we note that in the case $w = n_b = b_B = b_A$ the procedure given in Algorithms 2.1 and 2.2 essentially corresponds to Crawford's original algorithm.

2.3. Using a split factorization $B = S^H S$. So far we have only considered the nonsplit case $p = 0$, i.e., working with a "reverse" Cholesky factorization $B = L^H L$. However, the method described in Algorithms 2.1 and 2.2 is easily extended to a split factorization $B = S^H S$ with $1 \leq p < n$, which can be obtained with the LAPACK routine _PBSTF. (Actually, we use a slightly modified version of this routine that allows freely choosing the split position p.)

One way would be to duplicate the code, reversing the direction of the loops for the upper "half" of S (progressing downward with the inversion steps, bulge chasing toward the upper end).

Since the code is already somewhat involved, we have adopted another approach. For working with the lower part of S, we simply set $\mathrm{row}_{\min} := p+1$ and $\mathrm{col}_{\min} := 1$ in line 3 of Algorithm 2.1, and use S instead of L. The upper part of S can be handled by running the same code on *flipped* matrices. That is, we replace

$$\begin{aligned} &A := \overline{A}(n:-1:1, n:-1:1), \\ &S := \overline{S}(n:-1:1, n:-1:1), \\ &Z := Z(n:-1:1, n:-1:1) \qquad \{ \textit{ no complex conjugation with } Z \}, \end{aligned}$$

before the second call to Algorithm 2.1, where we set $\mathrm{row}_{\min} := n - p + 1$ and $\mathrm{col}_{\min} := \mathrm{row}_{\min}$ and drop the initialization of Z in lines 1 and 2. Note that the matrices must be flipped back at the end.

Code duplication would avoid these data movements, but they are not overly costly. Since A and B are symmetric and banded, only roughly $n(b_A + 1)$, $n(b_B + 1)$, and n^2 entries must be moved for flipping A, S, and Z, respectively, whereas the operation count is $\mathcal{O}(n^2 b_A)$ for the work on A and $\mathcal{O}(n^3)$ for building Z. Actual timings revealed that the flips indeed take only a rather low percentage of the overall time.

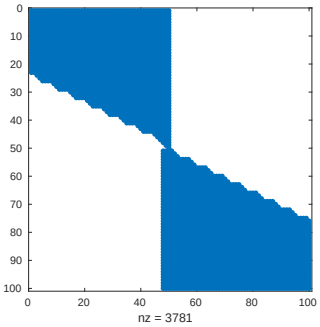

FIG. 2.6. *Structure of the accumulated transformation matrix* $Z = \widehat{L}^{-1}$ *for* $n = 100$, $p = 50$, $b_A = 6$, $b_B = 3$, *and* $n_b = 8$.

2.4. Improving the accumulation. The reduction $(A, B) \mapsto \widehat{C}$ is the first step in the solution of the GEP, and therefore the overall transformation matrix Z is initially set to the identity. Thus, even if the accumulation of the transformations during Algorithms 2.1 and 2.2 formally affects whole block columns of Z, major parts of these are zero. This tends to hold throughout the computations, in particular, when using a split factorization with a split position p close to the middle; see Figure 2.6 showing the final structure of such a matrix Z.

To make use of this fact, we keep track of the nonzeros in Z with two size-n integer arrays $i_{\min}$ and $i_{\max}$ indicating the positions of the topmost and bottommost nonzero entry in each column. Initially, $i_{\min} = i_{\max} = 1 : n$ due to $Z = I_n$.

Then each transformation is applied only to its "active range." Considering line 25 in Algorithm 2.1 as an example, the product can be restricted to

$$Z(\underline{i} : \overline{i}, J) := Z(\underline{i} : \overline{i}, J) \cdot D,$$

where $\underline{i} = \min i_{\min}(J)$ and $\overline{i} = \max i_{\max}(J)$. Following this computation, the index vectors must be updated. For each $j \in J$, we let $i_{\min}(j)$ be the position of the first nonzero in column j, where the search is started at position $\underline{i}$ ($\leq$ old $i_{\min}(j)$). Note that simply setting $i_{\min}(j) := \underline{i}$ would also provide a (cheaper, but not tight) "upper envelope" for the nonzeros in the new Z. The entries $i_{\max}(j)$ of the "lower envelope" are updated similarly, and line 20 in Algorithm 2.2 is adapted in the same manner.

Line 24 in Algorithm 2.1 is slightly more complicated. Again, $Z(:, I) := Z(:, I) + Z(:, J) \cdot E$ can be restricted to rows $\underline{i} : \overline{i}$, where $\underline{i}$ and $\overline{i}$ are determined as above (from columns J of Z), but then the entries $j \in I$ of $i_{\min}$ and $i_{\max}$ must be updated. Since these columns j may have contained nonzeros above position $i_{\min}$, the search now must start at position $\min\{\underline{i}, \text{old } i_{\min}(j)\}$.

Note that the integer arrays must be flipped together with Z, i.e., $i_{\min} := n + 1 - i_{\max}(n : -1 : 1)$ and $i_{\max} := n + 1 - \text{old } i_{\min}(n : -1 : 1)$.

Monitoring the "active lengths" of the updates revealed that, in connection with split factorizations, their average is reduced to approximately $n/3$.

It should be mentioned that in the LAPACK routines `_{SB,HB}GST` some of the rotations also are applied only to portions of the columns of Z, of length approximately $n/2$. Using $i_{\min}$ and $i_{\max}$ arrays would lead to significantly lower vector lengths and might be even slightly more effective than in our method because for each rotation the bounds $\underline{i}$ and $\overline{i}$ of the active range would be determined as the minimum (maximum) of just two numbers, leading to very few unnecessary operations with zeros. In [14],

Kaufman proposes a similar scheme; she derives formulas for predicting the active range $\underline{i} : \overline{i}$ rather than tracing it with arrays $i_{\min}$ and $i_{\max}$.

2.5. Storage considerations. Similarly to the reduction routine _{SB,HB}GST from LAPACK, we use the (lower) LAPACK storage scheme for symmetric banded matrices. That is, an entry a_{ij} from the lower triangle of A is stored at position A(1+i-j,j) of a two-dimensional array A, leading to columns (diagonals) of A's lower triangle being stored in A's columns (rows). Thus, an LDA $\times\, n$ array is required to hold A, where LDA $\geq 1 + b_A$.

The storage scheme for B and S is similar. B again is provided in lower symmetric banded storage in an LDB $\times\, n$ array, where LDB $\geq 1 + b_B$. The LAPACK factorization routine _PBSTF overwrites B with the split factor S, such that S's lower part is again stored in lower banded storage at the end of the array, and the remaining positions at the beginning are filled with the *transpose* of the upper part. Note that flipping puts this upper part at the end of the array, in lower banded storage.

Running Algorithms 2.1 and 2.2 on the array A would require LDA $\geq b_A + b_B + n_b$, much more than the $1 + b_A$ rows in the initial matrix A and the resulting matrix $\widetilde{C}$. However, only a small part of the additional rows is needed at any given time to hold the moving $(b_B + n_b - 1) \times (b_B + n_b - 1)$ bulge. Therefore we copy a "sliding window" (the blocks shown in Figures 2.3 and 2.5) into a $(b_A + b_B + n_b) \times (b_A + b_B + n_b)$ buffer, perform all operations on A in this buffer, and then copy the updated columns back to the array A. More precisely, since each chasing step drives the bulge by b_A positions toward the bottom of the band, the first b_A columns of the buffer are written back to A and the next b_A columns are loaded from A. As a result, both banded matrices can be packed tightly (LDA $\geq 1 + b_A$). Note that the whole "trailing" part of the band must be loaded and written back for each inversion step (and ensuing bulge chasing). These data movements are not shown in the algorithms, and again timings reveal that they account for only a small percentage of the overall time.

The matrix Z does not have a banded structure, and therefore standard two-dimensional storage in an LDZ $\times\, n$ array with LDZ $\geq n$ is used for it. There is, however, potential for saving workspace in the accumulation. While the work on Z in Algorithm 2.1 is done completely in place, the update in line 20 of Algorithm 2.2 requires $\mathcal{O}(w \cdot n)$ additional workspace. More precisely, _GEWY (_ORMQR with default settings) needs $w \cdot n$ ($\min\{w, 64\} \cdot n$, resp.) elements to hold an intermediate result. To avoid this, the update is done by segments, i.e., the active row range $\underline{i} : \overline{i}$ (cf. subsection 2.4) is split into chunks of a prescribed height h_Z, leading to updates of size $h_Z \times |\widetilde{M}| \times |\widetilde{L}|$ and $\mathcal{O}(w \cdot h_Z)$ workspace requirements. In practice, this approach did not significantly impact the performance of the updates, provided that h_Z was large enough (e.g., $h_Z \geq 256$) and that the block transforms could be reused (i.e., with _GEWY).

Together with, e.g., the buffers $\widetilde{D}$, etc., in Algorithm 2.1, our implementation requires workspace of overall size $\mathcal{O}((b_A + b_B + n_b)^2 + h_Z \cdot b_A)$, which, *asymptotically*, is even less than the size-n workspace needed by the reduction routine _{SB,HB}GST from LAPACK. For realistic matrix sizes and bandwidths, however, or if we use the $i_{\min}$ and $i_{\max}$ arrays, the LAPACK routine wins in terms of workspace requirements.

3. Selecting the algorithmic parameters. In this section we present a simple performance model that allows us to select the algorithmic parameters n_b and w in an automated way, at runtime. A couple of simplifications are made to speed up this selection process. Our model essentially follows the approach taken in [16, 17].

First, we observe that most of the overall work is done in the bulge chasing. Therefore we focus on this part of the algorithm, completely neglecting the inversion

steps. We consider only the variant that relies on the routines from the SBR toolbox (based on standard WY transformations).

To simplify things further, we analyze only a single bulge chasing step (lines 8 to 21 of Algorithm 2.2) with the bulge far enough from the end of the band, such that all blocks have the "typical sizes" indicated in the algorithm.

Dropping lower-order terms, lines 14 and 15 of the algorithm take $2|\widetilde{M}||\widetilde{L}|^2 - \frac{2}{3}|\widetilde{L}|^3$ and $2|\widetilde{M}||\widetilde{L}|^2$ operations [16], respectively, and most of these cannot be done with level-3 BLAS. By contrast, the $4|\widetilde{L}||\widetilde{M}||\widetilde{H}|$, $4|\widetilde{M}|^2|\widetilde{L}|$, $4|N||\widetilde{M}||\widetilde{L}|$, and, optionally, $4\frac{n}{3}|\widetilde{M}||\widetilde{L}|$ operations in lines 16, 17, 18, and 20, respectively, are done almost exclusively in level-3 BLAS routines, mainly with `_GEMM`. (Recall that the average active length for the Z updates in line 20 is $n/3$ instead of n.)

The next simplification concerns the performance that is achieved in these six computations. We assume that the performance of each call is determined *only by the smallest dimension* of the matrices involved, that is, w, and that it is given by

$$P^{\text{op}}(w) = \widehat{P}^{\text{op}} \cdot \frac{w}{w + w_{1/2}^{\text{op}}}, \tag{3.1}$$

where $\widehat{P}^{\text{op}}$ denotes the peak performance for the operation and $w_{1/2}^{\text{op}}$ is the value of w for which half of the peak performance is achieved. (This ansatz is well known from modeling the performance of vector operations, where the operands feature only one dimension, namely, the vector length [11].) Furthermore, we assume that these parameters are identical among all the BLAS3 operations and among the non-BLAS3 operations, resulting in just four values $\widehat{P}^{\text{BLAS3}}$, $w_{1/2}^{\text{BLAS3}}$, $\widehat{P}^{\text{nonBLAS3}}$, and $w_{1/2}^{\text{nonBLAS3}}$.

To determine these, we *measure* the BLAS3 performance for some number of w values, e.g., $w_{1:9} = (1, 2, 4, 8, 16, 32, 64, 128, 256)^T$, and fit the model (3.1) by solving the 9×2 weighted least squares problem

$$\min_x \|D \cdot (Mx - c)\|_2 ,$$

where

$$M = \left(w_{1:9} \mid -P^{\text{measured}}(w_{1:9}) \right), \quad x = \begin{pmatrix} \widehat{P}^{\text{BLAS3}} \\ w_{1/2}^{\text{BLAS3}} \end{pmatrix}, \quad c = P^{\text{measured}}(w_{1:9}) * w_{1:9},$$

the scaling with $D = \text{diag}(1/\log_2(w_{1:9} + 1))$ increases the influence of small w values, and "$*$" denotes componentwise multiplication. More precisely, for each w value, we measure the performance of multiplications $X \cdot Y$ ($X \in \mathbb{R}^{m \times n}$, $Y \in \mathbb{R}^{n \times w}$) and $X \cdot Y^T$ ($X \in \mathbb{R}^{m \times w}$, $Y \in \mathbb{R}^{n \times w}$), for all combinations of $m, n \in \{50, 100, 200, 400\}$, and take $P^{\text{measured}}(w)$ as the average of these 32 values. To minimize the effect of timer resolution, each multiplication is repeated until the overall time exceeds 10^4 times the resolution of the timer (including the overhead for calling it). The values for $\widehat{P}^{\text{nonBLAS3}}$ and $w_{1/2}^{\text{nonBLAS3}}$ are obtained similarly by timing `_GEQRF` and `_GEWYG`.

Figure 3.1 shows the averaged $P^{\text{measured}}(w_{1:9})$ performance ("$\times$" marks) and the fitted model (3.1) (solid line) for single-thread runs on the "Broadwell" machine described further in section 4. We obtained $\widehat{P}^{\text{BLAS3}} \approx 30.4$ GFlop/s (dashed horizontal line), $w_{1/2}^{\text{BLAS3}} \approx 5.7$, $\widehat{P}^{\text{nonBLAS3}} \approx 12.5$ GFlop/s, and $w_{1/2}^{\text{nonBLAS3}} \approx 23.7$. This calibration has to be done only *once* for each configuration (machine, number of threads, BLAS library). In our case, with a 1 μs resolution timer, it took less than five seconds.

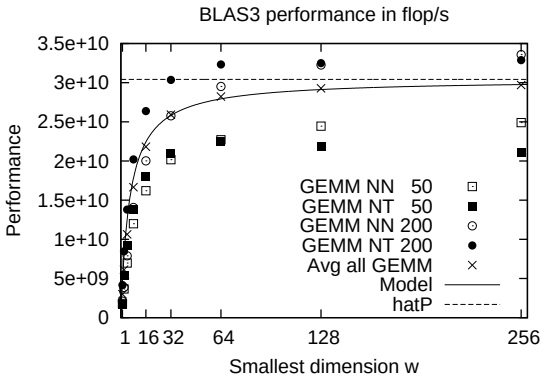

FIG. 3.1. *Measured average* `DGEMM` *performance ("×"), performance predicted by the fitted model (solid line), peak performance* $\widehat{P}^{\mathrm{BLAS3}}$ *(dashed horizontal line), and detailed measurements for* $m = n \in \{50, 200\}$ *with a single thread on the Broadwell machine; cf. section* 4.

Figure 3.1 also includes some of the more detailed data contributing to the averaged P^{measured} values, namely, for the multiplications $X \cdot Y$ ("GEMM NN") and $X \cdot Y^T$ ("GEMM NT") with $m = n \in \{50, 200\}$. These indicate that our assumptions may be oversimplifying things because the performance also depends on the other dimensions of the matrix, as well as on their shapes. We will comment on this issue at the end of the section.

Taking the above operation counts for the six calls occurring in the bulge chasing, plugging in the performance model, and summing up over the chunks gives us an estimate for the overall time of a single bulge chasing step,

$$T_{\mathrm{step}} = k\frac{4h^{\mathrm{av}}w^2 - \frac{2}{3}w^3}{P^{\mathrm{nonBLAS3}}(w)} + k\frac{4h^{\mathrm{av}}w(2b_A + z) + 4w(\overline{h}^2 - (k-1)\overline{h}w + \frac{(k-1)(2k-1)}{6}w^2)}{P^{\mathrm{BLAS3}}(w)}$$
$$+ \frac{4h_0w_0^2 - \frac{2}{3}w_0^3}{P^{\mathrm{nonBLAS3}}(w_0)} + \frac{4h_0w_0(2b_A + z) + 4h_0^2w_0}{P^{\mathrm{BLAS3}}(w_0)},$$

where $k = \lfloor(\ell_3 - \ell_1 + 1)/w\rfloor$ is the number of full-width chunks, $\overline{h} = b_B + n_b$ and $h^{\mathrm{av}} = \overline{h} - \frac{k-1}{2}w$ are their maximum and average height, respectively, $w_0 = \ell_3 - \ell_1 + 1 - kw$ $(< w)$ and $h_0 = \overline{h} - kw$ are the width and height, respectively, of the final chunk, and z is the average active length of the Z updates, i.e., $z = n/3$ if Z is needed and $z = 0$ otherwise.

Noting that a split decomposition with split position close to the middle of the matrix leads to two major phases, each requiring roughly $(n/2)/n_b$ inversion steps and bulge chasings, and that on average the bulge is chased over a distance of $(n/2)/2$, progressing by b_A positions with each chasing step, the overall time is given by

$$2 \cdot \frac{n/2}{n_b} \cdot \frac{n/4}{b_A} \cdot T_{\mathrm{step}} = \frac{n^2}{4b_A} \cdot \frac{T_{\mathrm{step}}}{n_b}.$$

Therefore n_b and w should be chosen such that the quantity $\tau := T_{\mathrm{step}}/n_b$ is minimized.

Our `auto` heuristic does this by evaluating and minimizing τ over the block sizes $n_b = 1 : n_b^{\max}$ with $n_b^{\max} = \min\{512, n/2\}$, each with $n_{\mathrm{QR}} = 1, 2, \ldots$ chunks (of width

$w = \lceil \min\{b_B + n_b - 1, b_A\}/n_{\mathrm{QR}} \rceil$). Multiple chunks ($n_{\mathrm{QR}} > 1$) are considered only as long as $w \geq 8$ in order to achieve satisfactory BLAS performance and to reduce the number of possible n_{QR} values and thus the search space.

For the single-threaded runs on the Broadwell system, `auto` selected $(n_b, n_{\mathrm{QR}}, w) = (171, 1, 20)$ for matrix size $n = 4000$, $b_A = b_B = 20$ with updating Z, and $(39, 2, 10)$ without Z. Note that n_b is significantly larger than the bandwidths, which would not be possible in Crawford's original formulation of the reduction scheme.

With the performance model described in this section, τ can be evaluated very quickly, and this allows searching in a rather large space of (n_b, w) combinations. On the machines used in our tests, the `auto` scheme relying on this very simple model was able to propose suitable parameter settings; see section 4. However, this may not always be the case. Recall that `auto` completely neglects data movements, computations for the inversion steps, and only distinguishes between BLAS3 and non-BLAS3 computations. If the need arises then the model might be extended to include different $\widehat{P}^{\mathrm{op}}$ and $w_{1/2}^{\mathrm{op}}$ parameters for each BLAS routine—and possibly even specific to the cases "matrix times block column," "block column times transposed block column," etc.—and/or to take the other dimensions of the matrices into account and/or to include inversion steps and other activities.

Finally we note that $\widehat{P}^{\mathrm{BLAS3}} = \widehat{P}^{\mathrm{nonBLAS3}} = 1$, $w_{1/2}^{\mathrm{BLAS3}} = w_{1/2}^{\mathrm{nonBLAS3}} = 0$ might be used to let `auto` minimize the number of arithmetic operations.

4. Numerical experiments. The results presented in this section were obtained with symmetric pseudorandom matrices of various dimensions. They were generated columnwise as follows. Starting with $k = 2016$, for $j = 1 : n$ and $d = 0 : \min\{b_A, n - j\}$, we let $a_{j+d,j} = \sin k + \cos k$ and incremented k. We continued increasing k to set B's entries in the same manner and, finally, we applied a shift $B := B + \sigma I_n$ such that $\mathrm{cond}(B) = 10$. If overflow/underflow occurs only rarely, then the performance of the routines considered in the following is almost independent of the actual values in the matrices.

Unless explicitly stated otherwise, the split position was set to the middle of the matrix, $p = n/2$, and all timings for our new implementation `DSBTCR` were made with "standard settings," i.e., nonsegmented ($n_{\mathrm{QR}} = 1$) WY transformations in the bulge chasing (cf. end of subsection 2.2), segmented updates of the active range $\underline{i} : \overline{i}$ of Z (cf. subsections 2.4 and 2.5, $h_Z = 256$), tight storage for the banded matrices (`LDA` $= b_A + 1$, `LDB` $= b_B + 1$), and they include the time for flipping the matrices and copying the sliding window (cf. subsection 2.5) and—in `auto` mode—for automatically selecting appropriate n_b and n_{QR} values. Note that the `auto` heuristic may choose segmented transformations, $n_{\mathrm{QR}} > 1$.

We first report *single-thread* results obtained on a machine with two 2.20 GHz 22-core Intel Xeon E5-2699v4 Broadwell CPUs. The test program was compiled with GNU `gfortran` 4.8.5, linked to the system's `/usr/lib64/liblapack.so.3` and the OpenBLAS v0.2.20 available from http://www.openblas.net, and run with the setting `OPENBLAS_NUM_THREADS=1`.

Figure 4.1 reveals that the time for LAPACK's double precision reduction routine `DSBGST` is roughly proportional to n^2 and b_B if Z is not required, and to n^3 and b_B/b_A if Z is accumulated. Our new implementation `DSBTCR` also profits from $b_B < b_A$, although to a much lesser degree. The `DSBTCR` times are "best" in the sense that, for each (n, b_A, b_B) combination, a set of n_b values was tested, and the fastest of these runs is reported. For $n \leq 4000$, we tried all $n_b = 8 : 256$, whereas only the values $n_b \in \{8, 16, 32, 64, 128, 256\}$ were used for the largest matrix size $n = 8000$.

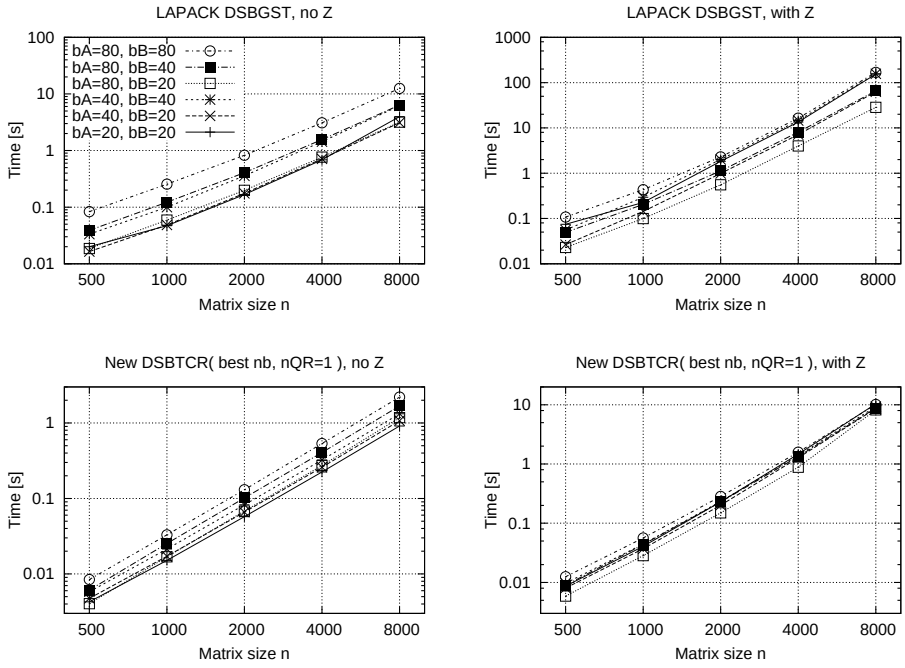

FIG. 4.1. *Single-thread timings for the LAPACK routine* `DSBGST` *(top) and our implementation* `DSBTCR` *(bottom). Right: including the accumulation of* Z*; left: without* Z*.*

The speedup plots in the top row of Figure 4.2 are based on the same timings. They show that, with best n_b (and $n_{\mathrm{QR}} = 1$), `DSBTCR` tends to be faster. The speedups in the bottom row of the figure were obtained with the n_b and w values determined by the `auto` scheme described in section 3. In comparison to the top row we see that `auto` is able to achieve rather satisfactory, if not optimum, performance. (Manual selection also might yield higher performance if the restriction $n_{\mathrm{QR}} = 1$ were dropped, but this would also increase the size of the search space for optimum parameters.)

As pointed out by one of the referees, Kaufman had proposed an alternative strategy [14] for applying the rotations in the reduction. Compared to the LAPACK implementation `DSBGST`, it leads to higher vector lengths for larger bandwidths b_A and allows reliance on an optimized `DROT` routine from the BLAS. We implemented this approach for *Phase* 1 of the reduction, dealing with the L part of the split factorization, and a speedup of almost 3 (up to 1.8 for matrices of size ≥ 2000) over `DSBGST` could be obtained for the reduction alone. (If Z is accumulated as well, then the difference is much smaller because the accumulation is done in the same way in both implementations and takes the vast majority of the arithmetic operations.) A higher speedup might be achieved by increasing the data locality, e.g., similiarly to techniques applied to the QZ algorithm in [7]. But this would require a complete redesign of `DSBGST`, which is out of scope for the present work. As no improved version of the `DSBGST` routine seems to be available publicly, we will continue comparison to the LAPACK implementation in the following.

Table 4.1 indicates what might happen if we deviate from the standard settings described above and used in Figures 4.1 and 4.2. (i) Relying on the LAPACK routine `DORMQR` instead of `DGEWY` wherever possible in Algorithm 2.2 will slightly slow down

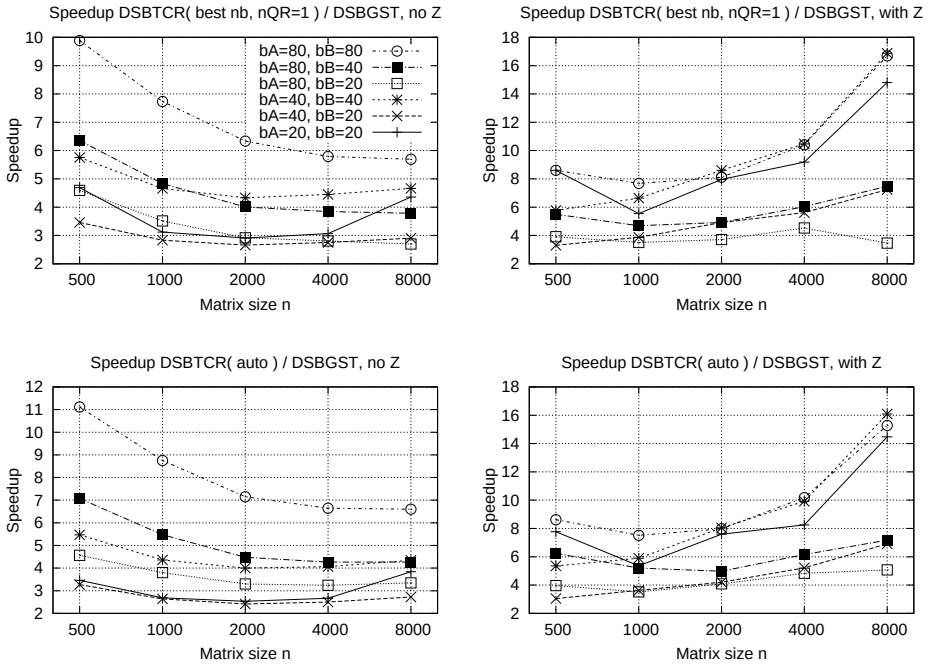

FIG. 4.2. *Single-thread speedups of* `DSBTCR` *over* `DSBGST` *with the best n_b values (top) and the parameters selected by the* `auto` *scheme (bottom). The timings were made on an Intel Xeon E5-2699v4 (Broadwell) CPU.*

TABLE 4.1

Effects of deviations from the standard settings (i) *using standard WY transforms* (ii) *without segmentation, $n_{\mathrm{QR}} = 1$, for the bulge chasing and* (iii) *height-($h_Z = 256$)-segmented accumulation of the* (iv) *active range of Z for matrices of size $n = 4000$, $b_A = b_B = 40$ with* (v) *split position in the middle, $p = n/2$. Times are given in seconds.*

			$n_b = 32$		$n_b = 128$	
			no Z	with Z	no Z	with Z
	Standard settings		0.436	2.287	0.329	1.433
(i)	Using `DORMQR` instead of `DGEWY`		0.529	3.255	0.374	1.889
(ii)	Segmented WY transforms	$n_{\mathrm{QR}} = 3$	0.403	2.728	0.359	1.902
(iii)	Nonsegmented update	$h_Z = n$		2.233		1.400
(iv)	Update of Z without active range			6.055		3.550
(v)	Unbalanced split	$p = n/4$	0.556	3.908	0.421	2.364
		$p = 0$	0.903	8.428	0.674	4.899
	Computations turned off		0.021	0.157	0.023	0.155

the bulge chasing and the accumulation of Z, due to the repeated generation of the triangular factor in the compact WY representation. (ii) Doing the WY transformations in several chunks may speed them up or slow them down, depending mainly on the values of b_A, b_B, and n_b. If the height of the overall QR decomposition, $m_2 - m_1$ ($= b_B + n_b$ in general), is much larger than its width (at most b_A) then little can be gained by using multiple chunks to reduce the fill in W. If, in addition, b_A is small then low width W and Y will not yield near-to-peak performance in the `DGEMM`s inside `DGEWY`. In our example, using multiple chunks paid for $n_b = 32$ (as it did when the `auto` speedups were higher than the "best" ones in the plots), whereas it did not pay for $n_b = 128$. (iii) The update of Z can be sped up only slightly by not doing it

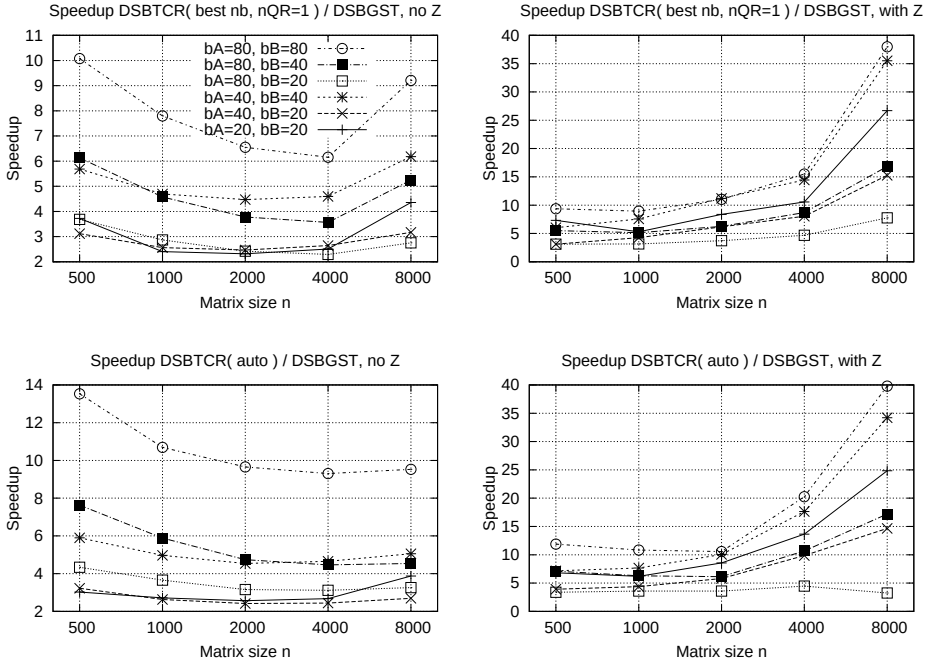

FIG. 4.3. *Speedups as in Figure* 4.2 *(same Broadwell CPU with OpenBLAS, four threads).*

in height-h_Z segments, at the cost of significantly more work space. (iv) The active range update is highly effective and should not be turned off; it reduces the time for updating Z by roughly two thirds. (v) Placing the split position far from the middle of the matrix cannot be recommended since this increases the average length of the bulge chases (up to a factor of two, if a nonsplit factorization is used) and leads to significantly more nonzeros in Z, thus impairing the effectiveness of the active range technique. The bottom row in the table gives the time for the overhead (setting indices, copying, etc.), which is on the order of 10% of overall time. In `auto` mode, an additional 0.0001 seconds are needed for selecting n_b and w.

Figures 4.3 and 4.4 show that the performance of our implementation and the effectiveness of the `auto` scheme are not limited to the particular machine configuration considered so far. The timings for Figure 4.3 have been obtained on the same Broadwell-based machine described above, but with four threads (`OPENBLAS_NUM_THREADS=4`), whereas those for Figure 4.4 have been made on a laptop with a 2-core 3.0 GHz Intel Core i7-4610M CPU and LAPACK routines and the BLAS taken from the Intel MKL. We did not restrict thread parallelism (`MKL_NUM_THREADS=`), leading to two threads being used. The speedups tend to be higher than in Figure 4.1 because `DSBTCR` benefits more from thread parallelism than `DSBGST`.

So far we have presented results only for double precision computations with real symmetric matrices. However, our reduction routine is available in four versions, {`D,S`}`SBTCR` for double and single precision real and {`Z,C`}`HBTCR` for double and single precision complex problems. Some results are given in Table 4.2, indicating that the new algorithm is competitive in all precisions. (For complex matrices, the real part of the entries was initialized as described above, and the imaginary part was chosen constant along the diagonals: $a_{j+d,j} = d/b_A$ and $b_{j+d,j} = d/b_B$.) We do not yet have

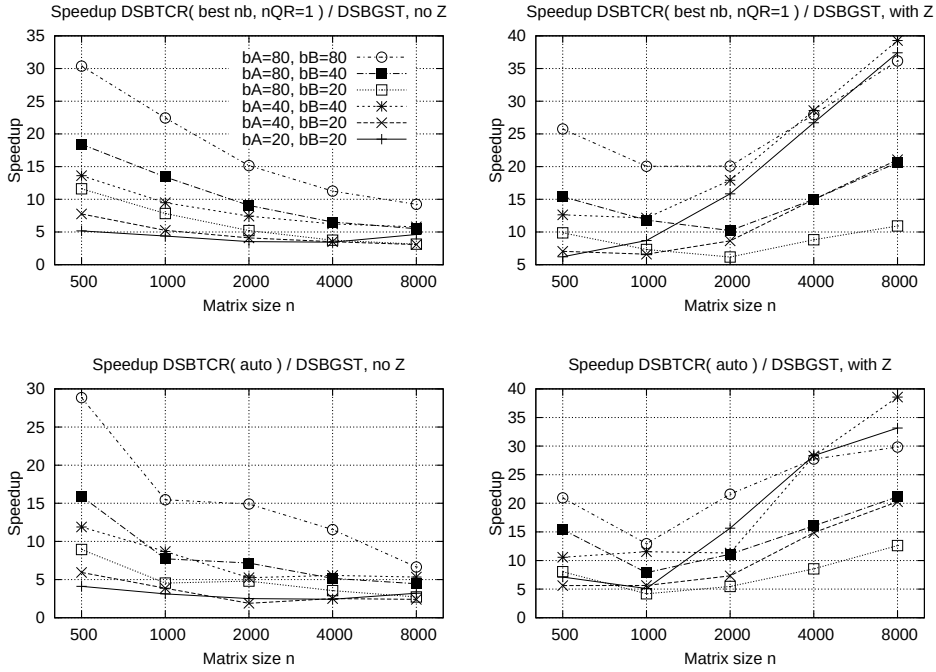

FIG. 4.4. *Speedups as in Figure* 4.2*, timings from an Intel Core i*7 − 4610*M with Intel MKL, two threads.*

TABLE 4.2

Timings (in seconds) of the LAPACK reduction routines for double and single precision real symmetric (`DSBGST` *and* `SSBGST`*, resp.) and double and single precision complex hermitian (*`ZHBGST`*,* `CHBGST`*) problems with the respective new* {`DSB,SSB,ZHB,CHB`}`TCR` *routines (*`auto` *mode and standard settings with fixed block sizes* $n_b \in \{32, 128\}$*) for matrices of size* $n = 4000$*,* $b_A = b_B = 40$ *on the Broadwell system with one thread.*

	___GST		___TCR auto		___TCR $n_b = 32$		___TCR $n_b = 128$	
	no Z	with Z	no Z	with Z	no Z	with Z	no Z	with Z
DSB___	1.512	15.532	0.363	1.484	0.458	2.377	0.351	1.489
SSB___	1.284	14.306	0.239	1.191	0.327	2.874	0.209	1.483
ZHB___	4.454	71.456	0.813	4.613	1.123	7.409	0.902	4.684
CHB___	3.740	206.722	0.590	3.420	0.856	14.050	0.564	9.008

a clear explanation for the (reproducibly) high times of some `CHB___` runs; they might be caused by a large number of underflows in the accumulation of Z.

In all our tests, reducing the generalized eigenproblem to a standard one with either the LAPACK routines `___GST` or our `___TCR` led to eigenpairs featuring residuals and B-orthogonality of comparable magnitude, without a clear winner.

5. Concluding remarks. We have proposed an algorithm for the reduction of the banded positive definite GEP $Ax = Bx\lambda$ to an equivalent SEP, such that the banded structure is preserved. Our algorithm combines and extends features from the original bulge chasing algorithm proposed by Crawford (block decomposition, working with submatrices) and the reduction routines `_{SY,HE}GST` available in LAPACK (bandwidths may be different, and faster execution, if $b_B < b_A$, halved operations count by using a split factorization of B instead of a Cholesky decomposition).

We have described additional techniques for increasing the performance, such as subdividing the transformations in the bulge chasing into chunks and monitoring the active range in the accumulation of the transformations, as well as for reducing the memory requirements, such as working with a sliding buffer to allow tight storage for the banded matrices and segmentation of the accumulation. Overall, our algorithm includes two additional parameters n_b (block size determining the size and progress of the inversion steps) and w (width of the chunks in the bulge chasing) that can be adjusted to optimize performance. We have proposed a very simple performance model and a heuristic `auto` for automatically selecting suitable n_b and w values.

While some further optimizations seem possible (in particular, to reduce data movements and to fully exploit the structure of some blocks in the matrices), numerical experiments confirmed that our current code is already highly competitive to the existing LAPACK implementation in terms of computing time, with comparable accuracy. In particular, the `auto` scheme led to adequate performance on different platforms (processor, BLAS library, number of threads).

It is planned to include this method in an upcoming extension of the SBR toolbox [4]. An efficient implementation of the method for distributed memory parallel computers is currently being completed [19].

Acknowledgment. The author wants to thank the unknown referees for their helpful comments, in particular, for pointing him to reference [14], which had escaped his attendance.

REFERENCES

[1] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide*, 2nd ed., SIAM, Philadelphia, 1995.

[2] K.-J. Bathe and E. L. Wilson, *Solution methods for eigenvalue problems in structural mechanics*, Internat. J. Numer. Methods Engng., 6 (1973), pp. 213–226.

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

[4] C. H. Bischof, B. Lang, and X. Sun, *Algorithm* 807: *The SBR Toolbox—Software for successive band reduction*, ACM Trans. Math. Software, 26 (2000), pp. 602–616.

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

[6] C. R. Crawford, *Reduction of a band-symmetric generalized eigenvalue problem*, Comm. ACM, 16 (1973), pp. 41–44.

[7] 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 (1999), pp. 425–454.

[8] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, 1979.

[9] J. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff, *Algorithm* 679—*A set of level* 3 *basic linear algebra subprograms: Model implementation and test programs*, ACM Trans. Math. Software, 16 (1990), pp. 18–28.

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

[11] J. J. Dongarra, I. S. Duff, D. C. Sorensen, and H. A. van der Vorst, *Numerical Linear Algebra for High-Performance Computers*, Software, Environments, and Tools, SIAM, Philadelphia, 1998.

[12] V. Fock, *Näherungsmethode zur Lösung des quantenmechanischen Mehrkörperproblems*, Z. Phys., 61 (1930), pp. 126–148.

[13] L. Kaufman, *Banded eigenvalue solvers on vector machines*, ACM Trans. Math. Software, 10 (1984), pp. 73–86.

[14] L. Kaufman, *Band reduction algorithms revisited*, ACM Trans. Math. Software, 26 (2000), pp. 551–567.
[15] W. Kohn and L. J. Sham, *Self-consistent equations including exchange and correlation effects*, Phys. Rev. (2), 140 (1965), pp. A1133–A1138.
[16] B. Lang, *Effiziente Orthogonaltransformationen bei der Eigen- und Singulärwertberechnung*, Habilitationsschrift, Fachbereich Mathematik, Bergische Universität GH Wuppertal, Germany, 1997 (in German).
[17] B. Lang, *Efficient eigenvalue and singular value computations on shared memory machines*, Parallel Comput., 25 (1999), pp. 845–860.
[18] A. Marek, V. Blum, R. Johanni, V. Havu, B. Lang, T. Auckenthaler, A. Heinecke, H.-J. Bungartz, and H. Lederer, *The ELPA library: Scalable parallel eigenvalue solutions for electronic structure theory and computational science*, J. Phys., 26 (2014), 213201.
[19] M. Rippl, *Efficient Transformation of the General Eigenproblem with Symmetric Banded Matrices to a Banded Standard Eigenproblem*, PASC17, Platform for Advanced Scientific Computing (PASC) Conference, 2017, Lugano, Switzerland.
[20] C. C. J. Roothaan, *Modern developments in molecular orbital theory*, Rev. Modern Phys., 23 (1951), pp. 69–89.
[21] R. Schreiber and C. Van Loan, *A storage-efficient WY representation for products of Householder transformations*, SIAM J. Sci. Stat. Comput., 10 (1989), pp. 53–57.
[22] J. H. Wilkinson, *Some recent advances in numerical linear algebra*, in Proceedings of the IMA Conference on The State of the Art in Numerical Analysis, York, D. Jacobs, ed., Academic Press, London, 1977, pp. 3–53.