PARTITIONED TRIANGULAR TRIDIAGONALIZATION

MIROSLAV ROZLOŽNÍK, GIL SHKLARSKI, AND SIVAN TOLEDO

Abstract. We present a partitioned algorithm for reducing a symmetric matrix to a tridiagonal form, with partial pivoting. That is, the algorithm computes a factorization $PAP^T = LTL^T$ where P is a permutation matrix, L is lower triangular with a unit diagonal and entries' magnitudes bounded by 1, and T is symmetric and tridiagonal. The algorithm is based on the basic (non partitioned) methods of Parlett and Reid and of Aasen. We show that our factorization algorithm is component-wise backward stable (provided that the growth factor is not too large), with a similar behavior to that of Aasen's basic algorithm. Our implementation also computes the QR factorization of T and solves linear systems of equations using the computed factorization. The partitioning allows our algorithm to exploit modern computer architectures (in particular, cache memories and high-performance BLAS libraries). Experimental results demonstrate that our algorithms achieve approximately the same level of performance as the partitioned Bunch-Kaufman factor and solve routines in LAPACK.

- Categories and Subject Descriptors:
  - G.1.3 [Numerical Analysis]: Numerical Linear Algebra—linear systems;
  - G.4 [Mathematics of Computing]: Mathematical Software—algorithm analysis; efficiency
- General Terms: Algorithms, Performance
- Additional Key Words and Phrases: symmetric indefinite matrices, tridiagonalization, Aasen's tridagonalization, Parlett-Reid tridagonalization, partitioned factorizations, recursive factorizations

1. Introduction

This paper presents a partitioned algorithm for reducing a square symmetric matrix to a tridiagonal form. This algorithm is essentially a partitioned formulation of the algorithm of Aasen [1] combined with an update step from the algorithm of Parlett and Reid [14]. The new formulation effectively exploits modern computer architectures through the use of the level-3 BLAS [8].

The Parlett-Reid and Aasen algorithms factor a symmetric matrix A into a product of a unit lower triangular matrix L, a symmetric tridiagonal matrix T, and the transpose of L:

$$A = LTL^T .$$

When partial pivoting is employed, these algorithms compute a factorization $PAP^T = LTL^T$ where P is a permutation matrix. Aasen's algorithm is component-wise backward stable when partial pivoting is used [12]. Asymptotically, the Parlett-Reid algorithm performs $2n^3/3 + O(n^2)$ arithmetic operations. Aasen's method performs only $n^3/3 + O(n^2)$ arithmetic operations, so it is much more efficient.

This factorization is one of the two main methods for factoring dense symmetric matrices. The other method, due to Bunch and Kaufman [7], factors the matrix into

Date: 5 June 2009.

a product $PAP^T = LDL^T$ where L is unit lower triangular and D is block diagonal with 1-by-1 and 2-by-2 blocks. The method of Bunch and Kaufman also performs $n^3/3 + O(n^2)$ operations. The performance of straightforward implementations of Bunch and Kaufman's method is similar to that of straightforward implementations of Aasen (see [5] for old experimental evidence; there is no reason that the results should be different on a more modern machine).

The performance of dense matrix algorithms improves dramatically when the ordering of operations is altered so as to cluster together operations on sub-blocks of the matrix. This is usually called *partitioning* or *blocking* (e.g., see [6] and [10]). The blocked Bunch-Kaufman algorithm is described in [9]. A study by Anderson and Dongarra compared the performance of blocked versions of the Bunch-Kaufman and Aasen methods on the Cray 2 and found that Aasen's method did not benefit from blocking as much as the Bunch-Kaufman method [3]. This finding appears to be the main reason that the Bunch-Kaufman method is used in the symmetric linear solver in LAPACK [2] rather than Aasen's method. The details of the blocked algorithms are not given in the report by Anderson and Dongarra; the code appears to have been lost [4].

The Bunch-Kaufman method is conditionally normwise backward stable (conditioned on element growth in the factors). The entries in its triangular factors can grow and this can lead to accuracy problems [4]. Ashcraft, Grimes, and Lewis, who observed this phenomenon, developed a variant called Bounded Bunch-Kaufman to address this problem. They have not implemented a partitioned version of the algorithm[1]. This issue has motivated us to develop a partitioned variant of the tridiagonalization algorithms, in which the magnitudes of the entries in the triangular factors are bounded by 1.

The implementation of our partitioned algorithm achieves similar performance to that of the blocked Bunch-Kaufman code in LAPACK. Our algorithm performs partial pivoting and its storage requirements are similar to those of LAPACK's implementation of Bunch-Kaufman. Our findings show that there is no performance-related reason to prefer the Bunch-Kaufman method over that of Aasen. We cannot determine whether the opposite conclusion of Anderson and Dongarra resulted from the particular way they blocked the algorithm or whether it is specific to the Cray 2 architecture.

We show that our partitioned algorithm is as stable as the basic (non partitioned) original algorithm, with only a slight change in the backward stability constant; this is a stronger result than the normwise backward stability property of the Bunch-Kaufman algorithm. We also show that our solution routine, which is based on QR factorization of T is also conditionally backward stable.

This paper is organized as follows. Section 2 presents a variant of Aasen's algorithm. Section 3 shows our partitioned formulation. Section 4 discusses how to efficiently implement our partitioned formulation. Section 5 presents numerical stability analysis of both the factorization and solution routines. Section 6 presents experimental results. We present our conclusions in Section 7.

[1]Higham recently wrote to us that he, Hammarling, and Lucas just developed such a code.

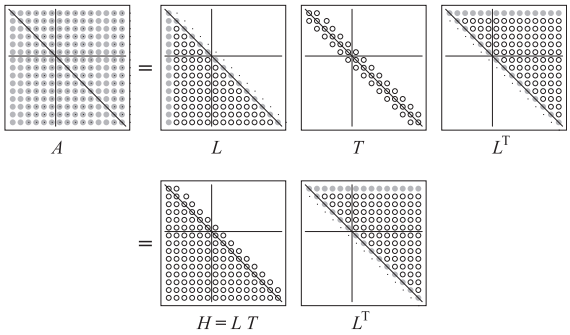

FIGURE 2.1. An illustration of the factorization. Filled gray circles represent known values and empty circles represent values that are computed during the factorization. The lines show the partitioning of the matrices into blocks.

2. A Variant of Aasen's Method

We start by describing a variant of Aasen's method. This variant, which we use as a building block in our partitioned algorithm, is slightly different from the original algorithm in [1] and the algorithm described in [12, 13].

Let A be an n-by-n real matrix and let

$$PAP^T = LTL^T$$

be its factorization, where L is a unit lower triangular matrix, T is a symmetric tridiagonal matrix, and P is a permutation matrix.

Let $H = LT$. Clearly, H is a lower-Hessenberg matrix and $PAP^T = HL^T$. See Figure 2.1 for an illustration of these matrices. The matrix H plays a central role in the formulation of Aasen's method. In our variant we also emit H as the output of the factorization. This makes it inefficient in terms of space, but this will not be a problem in the specific way that we use it in our partitioned algorithm.

For clarity, we ignore pivoting initially; we show later how to pivot. The factorization advances column by column. At step i we have already computed the first i columns of L, the first $i-1$ columns of T, and the first $i-1$ columns of H. The output of this step is the ith column of H and T, and the $(i+1)$st column of L.

For convenience of presentation, for a matrix M and indices i, j, k and ℓ we use $M^T_{i:j,k:\ell}$ to denote $(M_{i:j,k:\ell})^T$. Assume that we completed step $i-1$, we now show how to advance in a single step. We first compute $H_{i:n,i}$. From the ith column in the equation $A = HL^T$, we obtain $A_{i:n,i} = H_{i:n,1:n}L^T_{i,1:n}$. Since L is unit triangular, we have $A_{i:n,i} = H_{i:n,1:i-1}L^T_{i,1:i-1} + H_{i:n,i}$. Therefore, we can set $H_{i:n,i} = A_{i:n,i} - H_{i:n,1:i-1}L^T_{i,1:i-1}$ (or simply $H_{i:n,i} = A_{i:n,i}$ if $i = 1$).

Next we recover $T_{i-1,i}$ and $H_{i-1,i}$. Since T is symmetric we can set $T_{i-1,i} = T_{i,i-1}$. If $i > 1$, using the $(i-1, i)$ element in the system $H = LT$, and since $L_{i-1,i-1} = 1$, we get $H_{i-1,i} = T_{i-1,i}$. We now use the same equation $H = LT$ to compute the rest of $T_{:,i}$ and $L_{:,i+1}$. From elements i to n in the ith column of this equation we get

$$H_{i:n,i} = L_{i:n,i-1}T_{i-1,i} + L_{i:n,i:n}T_{i:n,i} \; .$$

We define $w = L_{i:n,i:n}T_{i:n,i}$, and obtain

$$w = H_{i:n,i} - L_{i:n,i-1}T_{i-1,i} \ .$$

The vector w can be computed since all the parts of the last equation are known (again, if $i = 1$, w is simply $H_{i:n,i}$). Moreover, since $L_{i,i} = 1$, $T_{i,i} = w_1$. In order to complete step i, we need to compute $T_{i+1,i}$, and $L_{(i+1):n,i+1}$ (if $i < n$). They can both be extracted from w. By definition,

$$\begin{aligned} w_{2:\text{end}} &= L_{(i+1):n,i:n}T_{i:n,i} \\ &= L_{(i+1):n,i}T_{i,i} + L_{(i+1):n,i+1}T_{i+1,i} \ . \end{aligned}$$

We define $v = L_{(i+1):n,i+1}T_{i+1,i}$. By the last equation, $v = w_{2:\text{end}} - L_{(i+1):n,i}T_{i,i}$, so it can be easily computed. Still ignoring pivoting for now, we can set $T_{i+1,i} = v_1$, since $L_{i+1,i+1} = 1$. Now, $L_{(i+2):n,i+1} = v_{2:\text{end}}/v_1$. This complete the formulation of this variant without pivoting.

Without pivoting, this construction may fail in one specific point. When computing $L_{(i+2):n,i+1}$, v_1 might be zero, or a very small v_1 might cause numerical problems. This can be solved easily, by permuting v, such that v_1 is the element with the highest magnitude. (If v is all zeros, $L_{(i+2):n,i+1}$ can be set to zero.) If such a permutation is performed, the appropriate preceding rows of L and H should be permuted accordingly, and also the trailing submatrix of A should be symmetrically permuted. Algorithm 1 summarizes our variant of Aasen's method.

3. Our Partitioned Method

We now derive our partitioned algorithm. We again start our description with pivoting ignored. Let $1 \leq k < n$. We assume that the first k columns of H, T and the first $(k+1)$ columns of L are computed using our Aasen variant from the previous section. The core idea is that instead of advancing by one more column, we update the trailing submatrix of A and continue to work on the trailing submatrix as if it were a new matrix. This is similar to the concept of the Parlett-Reid algorithm [14], in which the trailing submatrix is always symmetric and ready to be factored as if it was a new matrix. Moreover, when setting a partition size of 1, our algorithm is identical to the Parlett-Reid algorithm.

For convenience of presentation, we denote

$$\begin{aligned} A^{[11]} &= A_{1:k,1:k} \ , \\ A^{[12]} &= A_{1:k,(k+1):n} \ , \\ A^{[21]} &= A_{(k+1):n,1:k} \ , \text{ and} \\ A^{[22]} &= A_{(k+1):n,(k+1):n} \ , \end{aligned}$$

and similarly for L, T, and H. We examine the trailing submatrix part in the equation $A = HL^T$; we have

$$\begin{aligned} A^{[22]} &= H_{(k+1):n,1:n}L^T_{(k+1):n,1:n} \\ &= H^{[21]}\ {L^{[21]}}^T + H^{[22]}\ {L^{[22]}}^T \ . \end{aligned}$$

Figure 3.1 illustrates this equation. We have $A^{[22]}$ and we have already computed

Algorithm 1 Aasen factorization of the symmetric matrix A. This routine can also receive the first column of L as input, and also the number k of columns to handle. To get the standard factorization $PAP^T = LTL^T$, it should be called with AASEN(A,e_1,n), where n is the order of A and e_1 is a unit vector of order n. These extra parameters are useful when used as a subroutine for the partitioned factorization.
The routine also emits the permutation matrix P in two ways: as a matrix P and as a list p representing up to n index exchanges.

$[L,T,P,H,p] \leftarrow$AASEN(A,ℓ,k)
$n \leftarrow$ order of A.
$L \leftarrow I_n$ ▷ n-by-n identity matrix
$H \leftarrow O_n$ ▷ n-by-n zero matrix, stored as a whole for clarity only
$T \leftarrow O_n$ ▷ n-by-n zero matrix, stored as a whole for clarity only
$p \leftarrow \{\}_n$▷ Empty array of order n
$L_{1:n,1} \leftarrow \ell$
For $i = 1 \ldots k$ do
  $H_{i:n,i} \leftarrow A_{i:n,i} - H_{i:n,1:(i-1)} L^T_{i,1:(i-1)}$
  $w \leftarrow H_{i:n,i}$
  If $(i > 1)$
    $w \leftarrow w - L_{i:n,(i-1)} T_{(i-1),i}$
    $T_{(i-1),i} \leftarrow T_{i,(i-1)}$
    $H_{(i-1),i} \leftarrow T_{i,(i-1)}$ ▷This value is not really needed in practice
  End If
  $T_{i,i} \leftarrow w_1$
  If $(i < n)$
    $v \leftarrow w_{2:\text{end}} - T_{i,i} L_{(i+1):n,i}$
    If $(\max(|v|) \neq 0)$ and $(\text{argmax}(|v|) \neq 1)$
      Switch rows and columns $(i+1)$ and $(i + \text{argmax}(|v|))$ in $A_{(i+1):n,(i+1):n}$
      Switch rows $(i+1)$ and $(i + \text{argmax}(|v|))$ in $H_{:,1:i}$ and $L_{:,1:i}$
      Switch values in indecies 1 and $\text{argmax}(|v|)$ in v
      $p(i+1) \leftarrow i + \text{argmax}(|v|)$
    End If
    $T_{(i+1),i} \leftarrow v_1$
    If $(v_1 \neq 0)$
      $L_{(i+1):n,(i+1)} \leftarrow v / T_{(i+1),i}$
    End If
  End If
End For
▷ Creating an explicit P
$P \leftarrow I_n$ ▷ n-by-n identity matrix
For $i = 1 \ldots n$
  Switch rows $p(i)$ and i in P
End For

$H^{[21]}$, $L^{[21]}$, so we can compute the product $H^{[22]}\, {L^{[22]}}^T$. We now explore its

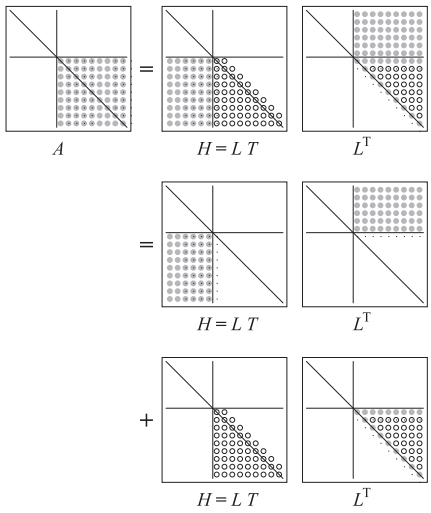

FIGURE 3.1. Expansion of $A^{[k,22]}$ in the partitioned algorithm.

structure; we first focus on $H^{[22]}$. We use the equation $H = LT$ and get

$$\begin{aligned} H^{[22]} &= L_{(k+1):n,1:n} T_{1:n,(k+1):n} \\ &= L^{[21]} T^{[12]} + L^{[22]} T^{[22]} . \end{aligned}$$

We multiply it by ${L^{[22]}}^T$ and get

$$\begin{aligned} H^{[22]} \, {L^{[22]}}^T &= L^{[21]} T^{[12]} \, {L^{[22]}}^T + L^{[22]} T^{[22]} \, {L^{[22]}}^T \\ &= L^{[21]}_{:,k} T^{[12]}_{k,:} \, {L^{[22]}}^T + L^{[22]} T^{[22]} \, {L^{[22]}}^T \\ &= L^{[21]}_{:,k} T^{[12]}_{k,1} \, {L^{[22]}_{1,:}}^T + L^{[22]} T^{[22]} \, {L^{[22]}}^T . \end{aligned}$$

The transition from the first to the second line is based on an expansion of $L^{[21]}T^{[12]}$ into a sum of column-times-row outer products, all of which except the first are zero. The transition from the second to the third line is based on a similar trick: $L^{[21]}_{:,k} T^{[12]}_{k,:}$ is a square matrix in which only the first column, $L^{[21]}_{:,k} T^{[12]}_{k,1}$, is not zero. Figure 3.2 illustrates these steps. We can compute $L^{[21]}_{:,k} T^{[12]}_{k,:} \, {L^{[22]}}^T$ simply by multiplying $L^{[21]}_{:,k} T^{[12]}_{k,1}$ by the first row of ${L^{[22]}}^T$. The first row of ${L^{[22]}}^T$ is exactly column $(k+1)$ in L which we already have. Therefore, we can compute the following expression

$$\begin{aligned} L^{[22]} T^{[22]} \, {L^{[22]}}^T &= H^{[22]} \, {L^{[22]}}^T - L^{[21]}_{:,k} T^{[12]}_{k,1} \, {L^{[22]}_{1,:}}^T \\ &= A^{[22]} - H^{[21]} \, {L^{[21]}}^T - L^{[21]}_{:,k} T^{[12]}_{k,1} \, {L^{[22]}_{1,:}}^T . \end{aligned}$$

This computation comprises of a rank-k and a rank-1 updates. The matrix

$$L^{[22]} T^{[22]} \, {L^{[22]}}^T$$

is an $(n-k)$-by-$(n-k)$ symmetric matrix, so it can be tridiagonalized with first column set to the (already computed) $L_{(k+1):n,k+1}$. The resulting factors are the last $(n-k)$ columns of T and the last $(n-k-1)$ columns of L.

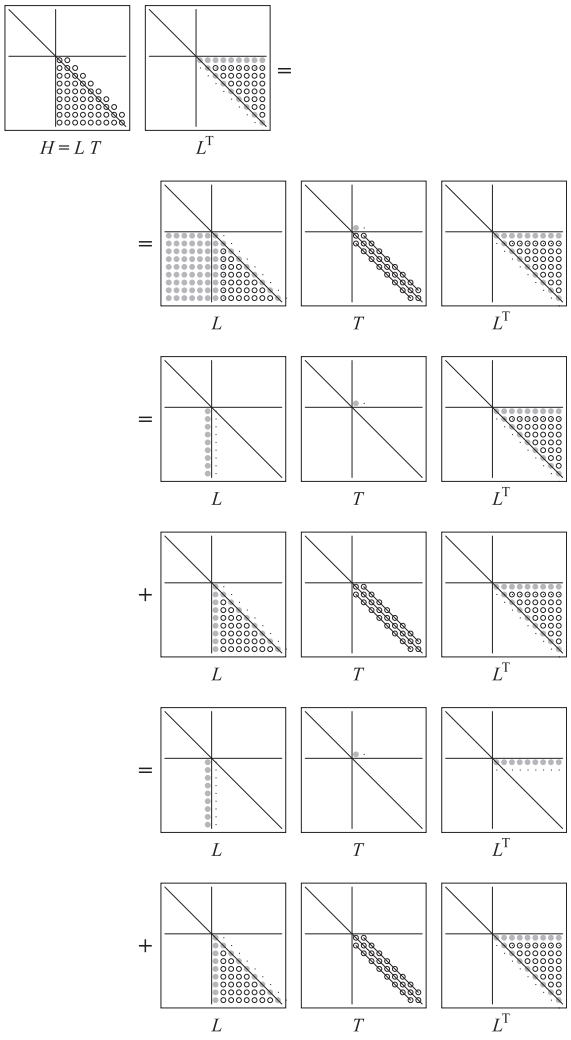

FIGURE 3.2. Expansion of $H^{[22]}\ {L^{[22]}}^T$ in the partitioned algorithm.

This reduction step can now be applied every k columns until no columns are left. At each step we use our Aasen variant to generate the next k columns. Then, we use them to update the trailing submatrix and continue to work on the trailing submatrix.

We now address the pivoting issue. The output of the Aasen factorization at each step is not only the next k columns of H, T and L, but also a permutation matrix P representing the row and column exchanges that we performed. We perform the exact row and column switches on A before updating it. This ensures that the row/column ordering is consistent and we can continue. We also store those updates to generate the final matrix P and update the previously computed columns of L. Algorithm 2 summarizes the entire partitioned process with pivoting.

Algorithm 2 Our partitioned algorithm top-level routine. k is the prescribed partition size.

[L,T,P] $\leftarrow$ PartitionedTridiagonalReduction(A,k)
$n \leftarrow$ order of A
$L \leftarrow I_n$ ▷ n-by-n identity matrix
$T \leftarrow O_n$ ▷ n-by-n zero matrix, stored as a whole for clarity only
$p \leftarrow [1, \ldots, n]$ ▷ Array of order n, used to keep track of interchanges
$\ell \leftarrow L_{:,1}$ ▷ The prescribed next column of L
$i \leftarrow 0$ ▷Keeps the number of columns of A that were already processed
While $(i < n)$
  $k \leftarrow \min(\text{order of } A, k)$ ▷Block size is k except (possibly) the last block
  $[L^{(k)},T^{(k)},P^{(k)},H^{(k)}, q] \leftarrow$ Aasen(A,ℓ,k) ▷ Aasen on the first k columns.
  ▷Place the results into L and T
  $L_{(i+1):n,(i+1):(i+k)} \leftarrow L^{(k)}_{:,1:k}$
  $T_{(i+1):(i+k),(i+1):(i+k)} \leftarrow T^{(k)}_{1:k,1:k}$
  If $((i + k) < n)$
    $\ell \leftarrow L^{(k)}_{(k+1):\text{end},(k+1)}$
    $T_{(i+k),(i+k+1)} \leftarrow T^{(k)}_{(k+1),k}$
    $T_{(i+k+1),(i+k)} \leftarrow T_{(i+k),(i+k+1)}$
  End If
  ▷Make the appropriate row and column exchanges dictated by the Aasen
  For $j \leftarrow 2 : \min(k + 1, \text{order of } A)$
    If $(j \neq q(j))$
      Switch rows and columns j and $q(j)$ in A
      Switch rows j and $q(j)$ in $L_{:,1:i}$
      Switch array cells $(j + i)$ and $(q(j) + i)$ in p
    End If
  End For
  $i \leftarrow i + k$
  If $((i + 1) \leq n)$ ▷ Update the trailing submatrix

$$A_{(k+1):\text{end},(k+1):\text{end}} \leftarrow A_{(k+1):\text{end},(k+1):\text{end}} - H^{(k)}_{(k+1):\text{end},1:k} \left(L^{(k)}_{(k+1):\text{end},1:k}\right)^T - L^{(k)}_{(k+1):\text{end},k} T_{(i+1),i} \left(L^{(k)}_{(k+1):\text{end},(k+1)}\right)^T$$

  End If
  $A \leftarrow A_{(k+1):\text{end},(k+1):\text{end}}$ ▷ Replace A with its updated the trailing submatrix
End While
▷ Form the global P
$P \leftarrow O_n$
For $i = 1 \ldots n$
  $P(i, p(i)) \leftarrow 1$
End For

4. Implementation

In the algorithm description so far, we have updated the entire trailing submatrix, which is inefficient. There is no need to update the entire trailing submatrix because it is symmetric; the tridiagonalization algorithms only use the lower triangular part of its input matrix A. With only half a matrix represented, the row and column interchanges should be carefully performed. The key idea is that row j in a symmetric matrix A with only its lower half is stored in two parts: $A_{j,1:j}$ and $A^T_{(j+1):n,j}$. Using this idea, our implementation only stores and updates half a matrix.

In order to perform the majority of the update with BLAS-3, we perform the following changes in the algorithm that we described so far. First, we fuse the rank-k update with $H^{[k,21]}\ {L^{[k,21]}}^T$ with the rank-1 update $L^{[k,21]}_{:,k} T^{[k,12]}_{k,1}\ {L^{[k,22]}_{1,:}}^T$, to create a single rank-$(k+1)$ update. Let UV^T be the fused update to the trailing m-by-m submatrix B, where U is m-by-$(k+1)$ and V is m-by-$(k+1)$. The second change is the specific way we update only half a matrix. We update the trailing submatrix in panels of width k. In the first step we subtract $UV^T_{1:k,1:(k+1)}$ from $B_{1:m,1:k}$, in the second step we subtract $U_{(k+1):m,1:(k+1)}V^T_{(k+1):2k,1:(k+1)}$ from $B_{(k+1)::m,(k+1):2k}$ and so on. This means that we update k-by-k upper triangular blocks that we never use. Nevertheless, it keeps most of the update process computation in BLAS-3.

In practice, in order to avoid some memory copies, the update process is not implemented entirely using BLAS-3. We perform the following slight change. We update the first column of B using a BLAS-2 update $UV^T_{1,1:(k+1)}$, and then update the rest of B using panels of width k. This is due to some implementation choices and is not an inherent problem.

We are now ready to compute the time complexity of our algorithm. The Aasen factorization of k columns of order n takes $O(nk^2)$ operations so the total n/k such calls cost $O(n^2k)$ operations. There are at most $O(n)$ exchange operations per column that is factored, so the total is $O(n^2)$. Therefore, the algorithm performs $O(n^2k)$ operations aside from the updates. Updating a trailing submatrix of size m costs $(m^2+mk)(k+1)+2m(k+1)$, out of which $2m(k+1)$ is outside BLAS-3. Careful analysis reveals that the total running time is

$$\frac{1}{3}\left(1+\frac{1}{k}\right)n^3+O(n^2k)\ ,$$

out of which only $O(n^2k)$ are not performed using BLAS-3 calls.

The next aspect of implementation that we need to examine is storage requirements. In order to perform the Aasen factorization for k columns and to store the part of H that is needed for the update, we store an extra buffer of size kn. The rest of the computation can be performed in-place. The computed columns of L can be stored in the array that stores A and the unit diagonal of L need not be stored. We do not store T, but factor it into its QR factors using Givens rotations as we go. The factors are stored in unused diagonals in the upper part of A. We also use 2 additional temporary arrays of size n. The total extra storage needed is therefore $(k+3)n$ which is similar to the working buffer needed for the blocked Bunch-Kaufman routine in LAPACK.

Our choice of QR and the fact that we fused the factorization of T with that of A are not essential. We could have also used LU with partial pivoting for T; both

preserve the sparsity of a tridiagonal matrix almost completely (in both cases the upper triangular matrix has 3 nonzero diagonals; in LU the unit lower triangular matrix has only two nonzero diagonals, and in QR we need to store $n-1$ rotations). We chose to fuse the factorization in order to demonstrate that solving with T, which is slightly harder than solving with the block diagonal D in Bunch-Kaufman factorization, does not have a significant effect on performance.

We end this section with a comment about row exchanges in L. There is no real need to exchange rows in preceding columns of L after a batch of new k columns of L is created. We do that in Algorithm 2 for clarity, in order to generate the complete L. In practice, as long as the solve routine is implemented accordingly, there is no need for exchanging elements of L in memory.

5. Rounding Error Analysis

Our partitioned algorithm (Algorithm 2) is thus the basic Aasen's algorithm in which some operations has been grouped and reordered into BLAS-3 matrix-matrix multiplications. In the rest of the computation we perform scalar (point) operations in the standard floating point arithmetic with the unit roundoff u (for details we refer to [13, Chapter 2]). Therefore, the rounding error analysis in [12] of the basic Aasen's algorithm (Algorithm 1 in this paper) should apply if the BLAS-3 operations are computed in a conventional way. However, if fast algorithms for matrix multiplication such as Strassen's method are eventually used, the standard results are not applicable and must be reformulated accordingly. We will make the assumption that the computed result $\hat{Z}$ from the BLAS-3 multiplication $Z = XY$ of matrices $X \in \mathbb{R}^{m\times k}$ and $Y \in \mathbb{R}^{k\times n}$ satisfies the component-wise error bound

$$|\hat{Z} - Z| \leq c_1(k)u|X||Y|, \tag{5.1}$$

where $c_1(k)$ is a constant that depends only on the dimension k. If we consider the conventional matrix-matrix multiplication, then for well-implemented BLAS-3 routines we have $c_1(k) \equiv \frac{k}{1-ku}$, see e.g. [13]. For implementations based on Strassen's method, the component-wise bound (5.1) does not hold and we have only norm-wise result $\|\hat{Z} - Z\| \leq c_2(m,n,k)u\|X\|\|Y\|$, where $c_2(m,n,k)$ is a rather complicated function of dimensions m, n and k and of the threshold that determines the level of recursion in the algorithm. The results for the corresponding partitioned Aasen's method would be significantly weaker than those obtained in Theorem 5.1 below; but we do not treat this issue here any further.

In the following we assume that the BLAS-3 operations satisfy (5.1) and formulate our component-wise results for the factorization and norm-wise results for the solution of linear systems. The analysis must answer the question whether Algorithm 2 is influenced in some significant way by the BLAS-3 formulation and whether the constants $c_1(k)$ coming from (5.1) are propagated stably into the final error bound (we want to prevent the exponential growth of the constants $c_3(n,k)$ and $c_5(n,k)$ that will appear in our bounds). First, we wish to bound the residual $\hat{P}A\hat{P}^T - \hat{L}\hat{T}\hat{L}^T$ of the computed factors $\hat{P}$, $\hat{L}$ and $\hat{T}$. For simplicity, we will ignore the permutation matrix $\hat{P}$ and we look at the matrix A as pre-pivoted. We have the following backward error bound for the factorization $A = LTL^T$.

Theorem 5.1. *If the partitioned Aasen's algorithm (Algorithm 2) with the partition size k applied to a symmetric $A \in \mathbb{R}^{n\times n}$ runs to completion, then the computed*

factors $\hat{L}$ and $\hat{T}$ satisfy the factorization

$$A + \Delta A = \hat{L}\hat{T}\hat{L}^T, |\Delta A| \leq c_3(n,k)u|\hat{L}||\hat{T}||\hat{L}|^T, \tag{5.2}$$

where the constant $c_3(n,k)$ depends on the dimension n and the partition size k ($n \geq k > 1$) and

$$c_3(n,k) = c_1(n + \left\lfloor \frac{n}{k} \right\rfloor + 2), \quad c_3(n,1) = c_1(n+3). \tag{5.3}$$

Proof. At the first step of the algorithm, we apply Algorithm 1 and tridiagonalize the first k columns of A to get $A^{[11]} = L^{[11]}T^{[11]}(L^{[11]})^T$ together with $L^{[21]}$ and the first column of $L^{[22]}$. It follows from Theorem 3.5 in [12] that the computed factors $\hat{L}^{[11]}$ and $\hat{T}^{[11]}$ satisfy

$$A^{[11]} + \Delta A^{[11]} = \hat{L}^{[11]}\hat{T}^{[11]}(\hat{L}^{[11]})^T, |\Delta A^{[11]}| \leq c_3(k,1)u|\hat{L}^{[11]}||\hat{T}^{[11]}||\hat{L}^{[11]}|^T, \tag{5.4}$$

where $c_3(k,1) = c_1(k+3)$. We note that Algorithm 1 is slightly different than the algorithm analyzed in [12]. Nevertheless, the same proof technique and the same bound can be derived for our variant.

The columns of the $H^{[21]}$ are computed using multiple triangular solves as $H_{:,i} = A_{:,i} - H_{:,1:(i-1)}(L_{i,1:(i-1)})^T$ for $i = 1, \ldots, k$ (Actually we compute only the coordinates i to n in the ith column of H). Using Lemma 2.2 from [12] for computed quantities we have $|A^{[21]}_{:,i} - \hat{H}^{[21]}_{:,1:i}(\hat{L}^{[11]}_{i,1:i})^T| \leq c_1(i)u|\hat{H}^{[21]}_{:,1:i}||\hat{L}^{[11]}_{i,1:i}|^T$ and collecting these identities for $i = 1, \ldots, k$ we get the identity for computed $\hat{H}^{[21]}$

$$A^{[21]} + \Delta B^{[21]} = \hat{H}^{[21]}(\hat{L}^{[11]})^T, |\Delta B^{[21]}| \leq c_1(k)u|\hat{H}^{[21]}||\hat{L}^{[11]}|^T \ . \tag{5.5}$$

Similar analysis applied to three-term recurrences $H_{i:n,i} = L_{i:n,i-1}T_{i-1,i} + L_{i:n,i}T_{i,i} + L_{i:n,i+1}T_{i+1,i}$ for $i = 1, \ldots, k$ leads to the matrix equation for computed $\hat{L}^{[21]}_{:,2:k}$ (note that $\hat{L}^{[21]}_{:,1}$ is given), $\hat{L}^{[22]}_{:,1}$, $\hat{T}^{[11]}$ and $\hat{T}^{[21]}_{1,k}$ in the form

$$\hat{H}^{[21]} + \Delta H^{[21]} = \hat{L}^{[21]}\hat{T}^{[11]} + \hat{L}^{[22]}_{:,1}\hat{T}^{[21]}_{1,k} \ , \tag{5.6}$$

$$|\Delta H^{[21]}| \leq c_1(3)u\left(|\hat{L}^{[21]}||\hat{T}^{[11]}| + |\hat{L}^{[22]}_{:,1}||\hat{T}^{[21]}_{1,k}|\right) \ . \tag{5.7}$$

Next we analyze the reduction step which assumes the tridiagonalization of the $(n-k)$-by-$(n-k)$ matrix $C^{[22]} = A^{[22]} - H^{[21]}(L^{[21]})^T - L^{[21]}_{:,k}T^{[21]}_{1,k}(L^{[22]}_{:,1})^T = L^{[22]}T^{[22]}(L^{[22]})^T$ and which is performed recursively k columns by k columns until no columns are left. Indeed, this computation comprises of a rank-k and a rank-1 update of $A^{[22]}$. In Algorithm 2 we actually combine these two updates, create a single $(k+1)$-rank update and work only on the half of a matrix as it is described in Section 4. We apply Lemma 2.2 of [12] on each $(k+1)$-rank update of the $[j*k : (n-k), (j-1)*k+1 : j*k]$-submatrix of $C^{[22]}$, where $j = 1, \ldots, [(n-k)/k]-1$. The corresponding computed matrix $\hat{C}^{[22]}$ satisfies (actually we have the result only for the lower-block half of the matrix)

$$\hat{C}^{[22]} - \Delta C^{[22]} = A^{[22]} - \hat{H}^{[21]}(\hat{L}^{[21]})^T - \hat{L}^{[21]}_{:,k}\hat{T}^{[21]}_{1,k}(\hat{L}^{[22]}_{:,1})^T, \tag{5.8}$$

$$|\Delta C^{[22]}| \leq c_1(k+1)u\left(|\hat{H}^{[21]}||\hat{L}^{[21]}|^T + |\hat{L}^{[21]}_{:,k}||\hat{T}^{[21]}_{1,k}||\hat{L}^{[22]}_{:,1}|^T\right). \tag{5.9}$$

Since we apply our partitioned algorithm recursively on the top of the $(n-k)$-by-$(n-k)$ matrix $\hat{C}^{[22]}$ the computed factors $\hat{L}^{[22]}$ and $\hat{T}^{[22]}$ satisfy

$$\hat{C}^{[22]} + \Delta D^{[22]} = \hat{L}^{[22]}\hat{T}^{[22]}(\hat{L}^{[22]})^T , \tag{5.10}$$

$$|\Delta D^{[22]}| \leq c_3(n-k,k)u|\hat{L}^{[22]}||\hat{T}^{[22]}||\hat{L}^{[22]}|^T . \tag{5.11}$$

Substituting (5.6) into (5.5) and introducing $\Delta C^{[21]} = \Delta H^{[21]}(\hat{L}^{[11]})^T$ we get

$$A^{[21]} + \Delta B^{[21]} + \Delta C^{[21]} = \hat{L}^{[21]}\hat{T}^{[11]}(\hat{L}^{[11]})^T + \hat{L}^{[22]}\hat{T}^{[21]}(\hat{L}^{[11]})^T.$$

Similarly we substitute (5.6) into (5.8) to obtain

$$\begin{aligned} A^{[22]} + \Delta B^{[22]} + \Delta C^{[22]} + \Delta D^{[22]} &= (\hat{L}^{[21]}\hat{T}^{[11]} + \hat{L}^{[22]}\hat{T}^{[21]})(\hat{L}^{[21]})^T \\ &+ (\hat{L}^{[21]}\hat{T}^{[21]} + \hat{L}^{[22]}\hat{T}^{[22]})(\hat{L}^{[22]})^T, \end{aligned} \tag{5.12}$$

where $\Delta B^{[22]} = \Delta H^{[21]}(\hat{L}^{[21]})^T$. Taking these two equations together with (5.4) we get the statement $A + \Delta A = \hat{L}\hat{T}\hat{L}^T$, where $\Delta A^{[21]} = \Delta B^{[21]} + \Delta C^{[21]}$ and $\Delta A^{[22]} = \Delta B^{[212]} + \Delta C^{[22]} + \Delta D^{[22]}$. The perturbation matrices $\Delta A^{[21]}$ and $\Delta A^{[22]}$ can be after some manipulation bounded as follows

$$|\Delta A^{[21]}| \leq [c_1(k) + c_1(3)]\, u \left(|\hat{L}^{[21]}||\hat{T}^{[11]}| + |\hat{L}^{[22]}||\hat{T}^{[21]}|\right) |\hat{L}^{[11]}|^T ,$$

$$\begin{aligned} |\Delta A^{[22]}| &\leq [c_1(k) + c_1(3)]\, u \left(|\hat{L}^{[21]}||\hat{T}^{[11]}| + |\hat{L}^{[22]}||\hat{T}^{[21]}|\right) |\hat{L}^{[21]}|^T , \\ &+ [c_1(k+1) + c_3(n-k,k)]\, u \left(|\hat{L}^{[21]}||\hat{T}^{[21]}| + u|\hat{L}^{[22]}||\hat{T}^{[22]}|\right) |\hat{L}^{[22]}|^T . \end{aligned} \tag{5.13}$$

The error constant $c_3(n,k)$ in the final bound depends on the constants $c_3(k,1)$, $c_1(k+3)$ and $c_1(k+1) + c_3(n-k,k)$ and the block size k. Since $c_1(k) + c_1(3) \leq c_1(k+3)$, we set $c_3(n,k)$ to

$$c_3(n,k) = \max\left\{c_1(k+3), c_3(k,1), c_1(k+1) + c_3(n-k,k)\right\} .$$

□

The bound (5.1) for the BLAS-3 matrix-matrix multiplication used in the rank-$(k+1)$ updates of $A^{[22]}$ actually appears in the presence of the term $c_1(k+1)$ in (5.8), which is then reflected in the bound (5.13) for the matrix $\Delta A^{[22]}$. It seems that the error constant $c_3(n,k)$ grows additively at worst. The difference of constants $c_3(n,k) - c_3(n,1)$ in bounds for the partitioned and basic algorithm is negligible and is given by the number of extra rank-1 updates occurring after reduction of each partition, which do not appear in the basic Aasen's method.

The result of Theorem 5.1 essentially justifies the use of conventional BLAS-3 routines in the partitioned Aasen's scheme. To solve a symmetric linear system $Ax = b$ using the factorization $A = LTL^T$ we need to consider the systems $Ly = b$, $Tz = y$ and $L^Tx = z$. The tridiagonal system $Tz = y$ is solved by QR factorization using Givens rotations. Given a QR factorization of T, the system $Tz = y$ can be solved by forming Q^Ty and then solving $Rz = Q^Ty$. As already noted, instead of storing T in our implementation we keep only the rotations for Q and the upper triangular matrix R which has 3 nonzero diagonals. For the approximate solution $\hat{x}$ computed in finite precision arithmetic we can formulate the following statement.

Theorem 5.2. *Let A be symmetric and $\hat{L}$ and $\hat{T}$ be the computed factors from the partitioned Aasen's algorithm (Algorithm 2). Assuming $c_4(2n-2)n^{1/2}u\kappa_\infty(\hat{T}) <$*

1, *the computed solution* $\hat{x}$ *to* $Ax = b$ *computed with the aid of the Givens QR factorization of* $\hat{T}$ *satisfies*

$$(A + \widehat{\Delta A})\hat{x} = b + \widehat{\Delta b}, \tag{5.14}$$

$$\|\widehat{\Delta A}\|_\infty \le c_5(n,k)u\|\hat{T}\|_\infty, \quad \|\widehat{\Delta b}\|_\infty \le c_5(n,k)u\|\hat{T}\|_\infty\|\hat{x}\|_\infty \tag{5.15}$$

where $c_4(k)$ *stands for a small multiple of* $c_1(k)$ *(that is,* $c_4(k) = c_0c_1(k)$ *for some small* c_0*) and the constant* $c_5(n,k)$ *depends on the dimension* n *and the partition size* k*.*

Proof. Since $\hat{L}$ is unit lower-diagonal, for the computed vector $\hat{z}$ we have $(\hat{L} + \Delta L_1)\hat{z} = b$ with $|\Delta L_1| \le c_1(n-1)u|\hat{L}|$. Based on Theorem 19.10 in [13] the computed triangular factor $\hat{R}$ obtained via the Givens QR decomposition of the matrix $\hat{T}$ satisfy

$$\hat{T} + \Delta T = \tilde{Q}\hat{R}, \|\Delta T\|_\infty \le c_4(2n-2)u\|\hat{T}\|_\infty, j = 1, \dots n, \tag{5.16}$$

where $\tilde{Q}$ is exactly orthonormal matrix with $\tilde{Q}^T\tilde{Q} = I$. By Lemma 19.9 in [13], the computed right-hand side satisfies $\hat{c} = \tilde{Q}^T(\hat{z} + \Delta z)$, where $\|\Delta z\| \le c_4(2n-2)u\|\hat{z}\|$. It is clear from (5.16) that the computed $\hat{R}$ is guaranteed to be nonsingular if $c_4(2n-2)n^{1/2}u\kappa_\infty(\hat{T}) < 1$. The computed solution $\hat{y}$ to the triangular system $\hat{R}y = \hat{c}$ satisfies $(\hat{R} + \Delta R)\hat{y} = \hat{c}$, where $|\Delta R| \le c_1(n)u|\hat{R}|$. Multiplying this identity from the left by $\tilde{Q}$ together with (5.16) we obtain

$$(\hat{T} + \Delta T + \tilde{Q}\Delta R)\hat{y} = \hat{z} + \Delta z. \tag{5.17}$$

The computed approximate solution $\hat{x}$ comes from the triangular solve involving unit upper triangular matrix $\hat{L}^T$ and thus we have $(\hat{L} + \Delta L_2)^T\hat{x} = \hat{y}$ with $|\Delta L_2| \le c_1(n-1)u|\hat{L}_2|$. Substituting for $\hat{y}$ and multiplying (5.17) by $(\hat{L} + \Delta L_1)$ we get

$$(\hat{L} + \Delta L_1)(\hat{T} + \Delta T + \tilde{Q}\Delta R)(\hat{L} + \Delta L_2)^T\hat{x} = (\hat{L} + \Delta L_1)(\hat{z} + \Delta z).$$

Since due to Theorem 5.1 $A + \Delta A = \hat{L}\hat{T}\hat{L}^T$ we introduce the perturbations $\widehat{\Delta A} = \Delta A + \Delta L_1(\hat{T} + \Delta T + \tilde{Q}\Delta R)(\hat{L} + \Delta L_2)^T + \hat{L}(\Delta T + \tilde{Q}\Delta R)(\hat{L} + \Delta L_2)^T + \hat{L}\hat{T}(\Delta L_2)^T$ and $\widehat{\Delta b} = (\hat{L} + \Delta L_1)\Delta z$ to obtain the desired identity $(A + \widehat{\Delta A})\hat{x} = b + \widehat{\Delta b}$. The bounds for $\|\widehat{\Delta A}\|$ and $\|\widehat{\Delta b}\|$ follow accordingly from bounds $\|\Delta A\|_\infty \le c_3(n,k)u\|\hat{L}\|_\infty\|\hat{T}\|_\infty\|\hat{L}^T\|_\infty$, $\|\Delta T\|$, $\|\Delta R\|_\infty \le c_1(n)u\|\hat{R}\|_\infty$, $\|\Delta L_1\|_\infty \le c_1(n-1)u\|\hat{L}\|_\infty$ and $\|\Delta L_2\|_\infty \le c_1(n-1)u\|\hat{L}\|_\infty$. Due to pivoting, every element of $\hat{L}$ is bounded by 1 and since its first column is a unit vector, we have that $\|\hat{L}\| \le \|\hat{L}\|_\infty \le n-1$. Consequently, for the constant $c_5(n,k)$ we have $c_5(n,k) = (n-1)^2[c_3(n,k) + c_4(2n-2) + 3c_1(n)]$. Neglecting higher order terms (that lead only to somewhat larger constant $c_5(n,k)$) we obtain the statement of our Theorem. □

Theorem 5.2 shows that solving systems with the partitioned Aasen's algorithm is a backward stable method provided that $\|\hat{T}\|_\infty/\|A\|_\infty$ is not too large. As in [12], we can define the growth factor $\rho_n = \frac{\max_{i,j}|\hat{T}_{i,j}|}{\max_{i,j}|A_{i,j}|}$ to obtain the bounds

$$\|\widehat{\Delta A}\|_\infty \le c_5(n,k)nu\rho_n\|A\|_\infty, \quad \|\widehat{\Delta b}\|_\infty \le c_5(n,k)nu\rho_n\|A\|_\infty\|\hat{x}\|_\infty.$$

The growth factor ρ_n can be easily monitored and plays a similar role as in Gaussian elimination with pivoting. For a discussion on this topic we refer to Subsection 3.2.2 of [12]. The influence of partitioning on the accuracy of the computed approximate solution $\hat{x}$ is again marginal and it appears only in the constant $c_5(n,k)$

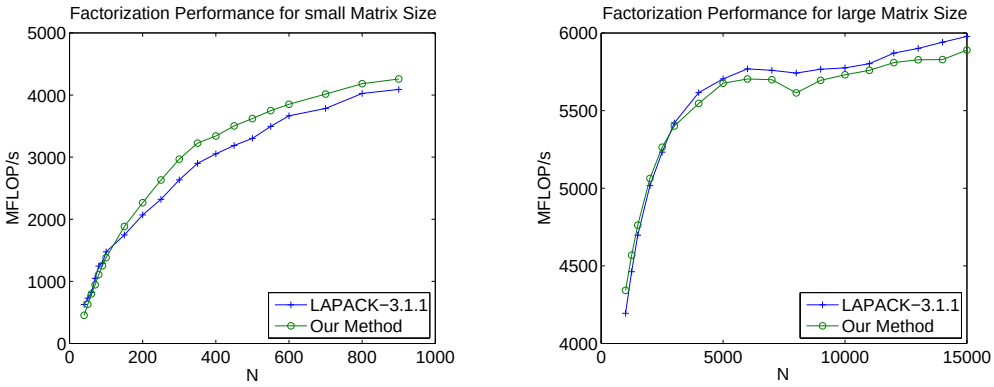

FIGURE 6.1. The performance of the methods. The vertical axis shows the effective computational rate.

which depends additively on $c_3(n,k)$. Therefore, the difference between the numerical behaviors of partitioned and point Aasen's algorithm will be hardly visible in practical computations.

6. Experimental Results

We have compared the performance of our partitioned tridiagonalization method to the performance of the partitioned Bunch-Kaufman algorithm in LAPACK version 3.1.1 [2]. The results that we report below show that the performance of the partitioned tridiagonalization method is similar to that of the Bunch-Kaufman algorithm. The performance of the solver phases is also similar.

We linked all the codes (ours and LAPACK) to the Goto BLAS version 1.12 [11]. The codes were compiled using GCC version 4.1.2 on an Intel x86_64 machine running Linux. The computer on which we ran the experiments had an Intel Core 2 model 6400 running at 2.13 GHz. We restricted the BLAS to use only one core of the CPU using an environment variable. The computer had 4 GB of main memory; we did not detect any significant paging activity during the experiments.

We used a block size $k = 64$ both in our method and in LAPACK. This value is the default of LAPACK 3.1.1. In experiments not reported here, we verified that this value is indeed the optimal block size for this machine (and for all the methods).

The experiments were conducted using random symmetric matrices whose elements are uniformly distributed in $(-1, 1)$. The relative residuals were always below 10^{-12} and the residuals produced by the different methods were of similar magnitude.

Figure 6.1 compares the performance of the methods. The vertical axis shows effective computational rates in millions of floating-point operations per second,

$$\frac{1}{3}\frac{n^3}{T} \times 10^{-6} \,,$$

where T is the running time in seconds. Note that this metric compares actual running times and ignores differences in operation counts. Therefore, a higher value always indicates a faster factorization.

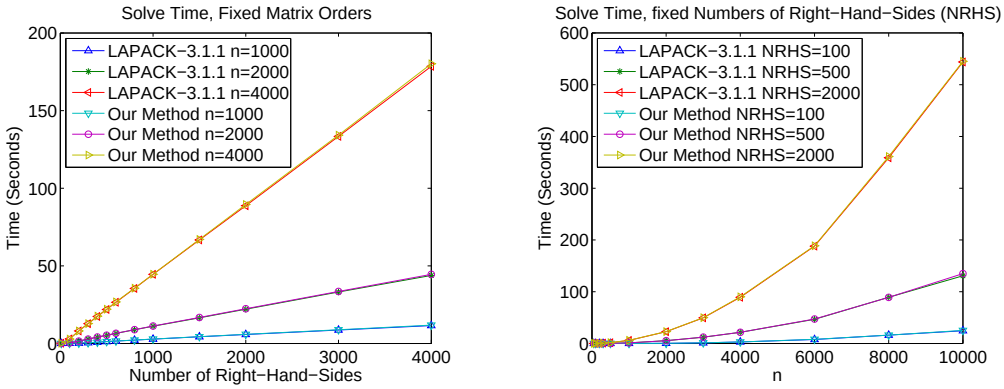

FIGURE 6.2. The performance of the solve phase. The graphs of our method coincide with the graphs of the Bunch-Kaufman method in LAPACK.

On matrices of order up to about 5000, our tridiagonalization method is slightly faster or run at the same speed as the Bunch-Kaufman algorithm. On larger matrices, the Bunch-Kaufman algorithm is sometimes faster. In either case, the relative performance differences are not large. We do not know why the performance of the tridiagonalization algorithm drops at $n = 8000$.

The graphs in Figure 6.2 show that the performance of the solve phase is essentially the same for our new methods and for LAPACK's implementation of the Bunch-Kaufman algorithm.

7. Concluding Remarks

We have presented a symmetric tridiagonalization algorithm. The algorithm is partitioned, so it effectively exploits high-performance level-3 BLAS routines. The algorithm performs almost the same number of operations asymptotically as the basic Aasen algorithm [1]. We have analyzed this issue theoretically and verified the results experimentally.

A detailed stability analysis shows that our factorization is conditionally componentwise backward stable as long as the BLAS-3 routines satisfy a componentwise error bound. This error bound is satisfied by partitioned (blocked) conventional matrix multiplication codes but not by Strassen-like algorithms. With a conventional (non-Strassen) BLAS-3, the only noticeable difference between the analysis of Aasen's original algorithm and that of our algorithm is due to extra rank-1 update per partition. Therefore, in this sense, the analysis is quite similar; numerically, our algorithm perform almost the same as the original one.

Our implementation of the algorithm performs similarly to LAPACK's symmetric indefinite solver, in both the factor and the solve phases. This shows that the findings of Anderson and Dongarra, who found that a partitioned variant of Aasen's algorithm performed more poorly than a partitioned Bunch-Kaufman [3], are not valid for our new algorithm. Our experiments were conducted on a very different machine than the one used by Anderson and Dongarra, but we do not see why our algorithm would perform poorer than LAPACK even on a machine like the Cray 2;

our methods and LAPACK's implementation of Bunch-Kaufman perform essentially the same sequence of level-3 BLAS calls.

More generally, our results show that symmetric tridiagonalization methods can be as efficient as symmetric block-diagonalization (when both utilize non-unitary transformations). The extra cost of solving tridiagonal matrices is negligible. The triangular factors that our methods produce always have elements bounded in magnitude by 1, whereas Bunch-Kaufman methods sometimes produce triangular factors with large elements [4].

In terms of stability, the tridiagonal factor computed by the Aasen method may also have large elements. Roughly speaking, the numerical behavior of both approaches is similar (although of a somewhat different nature) and provided that the corresponding growth factors are not too large, both their partitioned versions are backward stable.

Acknowledgement. Shlarski and Toledo were supported by an IBM Faculty Partnership Award, by grant 848/04 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), and by grant 2002261 from the United-States-Israel Binational Science Foundation. Rozložník was supported by the Grant Agency of the Czech Academy of Sciences under the project IAA100300802. We thank the anonymous referees and Nick Higham for helpful suggestions and comments.

References

[1] J. O. Aasen. On the reduction of a symmetric matrix to tridiagonal form. *BIT*, 11:233–242, 1971.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[3] Edward Anderson and Jack Dongarra. Evaluating block algorithm variants in LAPACK. In Jack Dongarra, Paul Messina, Danny C. Sorensen, and Robert G. Voigt, editors, *Proceedings of the 4th Conference on Parallel Processing for Scientific Computing*, pages 3–8. SIAM, 1989.

[4] Cleve Ashcraft, Roger G. Grimes, and John G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM J. Matrix Anal. Appl.*, 20(2):513–561, 1998.

[5] Barwell, V. and George, A. A comparison of algorithms for solving symmetric indefinite systems of linear equations. *ACM Trans. Math. Software*, 2:242–251, 1976.

[6] Paolo Bientinesi, John A. Gunnels, Margaret E. Myers, Enrique Quintana-Orti, and Robert van de Geijn. The science of deriving dense linear algebra algorithms. *ACM Transactions on Mathematical Software*, 31:1–26, 2005.

[7] J. Bunch and L. Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31(137):163–179, January 1977.

[8] Jack J. Dongarra, Jeremy Du Cruz, Sven Hammarling, and Ian Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, 1990.

[9] Jack J. Dongarra, Iain S. Duff, Danny C. Sorensen, and Henk A. van der Vorst. *Numerical Linear Algebra for High-Performance Computers*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1998.

[10] Erik Elmroth, Fred Gustavson, Isak Jonsson, and Bo Kågström. Recursive blocked algorithms and hybrid data structures for dense matrix library software. *SIAM Review*, 46(1):3–45, March 2004.

[11] Kazushige Goto and Robert van de Geijn. High-performance implementation of the level-3 BLAS. Technical Report CS-TR-06-23, The University of Texas at Austin, Department of Computer Sciences, May 5 2006.

[12] Nicholas J. Higham. Notes on accuracy and stability of algorithms in numerical linear algebra. In Mark Ainsworth, Jeremy Levesley, and Marco Marletta, editors, *The Graduate Student's Guide to Numerical Analysis '*98, pages 48–82. Springer-Verlag , Berlin, 1999.

[13] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[14] B. N. Parlett and J. K. Reid. On the solution of a system of linear equations whose matrix is symmetric but not definite. *BIT*, 10:386–397, 1970.