SIAM J. MATRIX ANAL. APPL.
Vol. 35, No. 4, pp. 1364–1406

COMMUNICATION-AVOIDING SYMMETRIC-INDEFINITE FACTORIZATION*

GREY BALLARD†, DULCENEIA BECKER‡, JAMES DEMMEL§, JACK DONGARRA‡¶‖, ALEX DRUINSKY#, INON PELED††, ODED SCHWARTZ§, SIVAN TOLEDO††, AND ICHITARO YAMAZAKI‡

Abstract. We describe and analyze a novel symmetric triangular factorization algorithm. The algorithm is essentially a block version of Aasen's triangular tridiagonalization. It factors a dense symmetric matrix A as the product $A = PLTL^TP^T$, where P is a permutation matrix, L is lower triangular, and T is block tridiagonal and banded. The algorithm is the first symmetric-indefinite *communication-avoiding* factorization: it performs an asymptotically optimal amount of communication in a two-level memory hierarchy for almost any cache-line size. Adaptations of the algorithm to parallel computers are likely to be communication efficient as well; one such adaptation has been recently published. The current paper describes the algorithm, proves that it is numerically stable, and proves that it is communication optimal.

Key words. symmetric-indefinite matrices, communication-avoiding algorithms, Aasen's factorization

AMS subject classifications. 65F05, 15A23, 65Y20

DOI. 10.1137/130929060

1. Introduction.

The running time of algorithms is mostly determined by the amount of arithmetic (or other primitive data transformations) and by the amount and types of data movements that are required. Early analyses of algorithms focused

*Received by the editors July 16, 2013; accepted for publication (in revised form) by I. C. F. Ipsen July 28, 2014; published electronically November 13, 2014. This research was supported in part by NSF CCF-1117062 and CNS-0905188, by Microsoft Corporation Research Project Description "Exploring Novel Approaches for Achieving Scalable Performance on Emerging Hybrid and Multi-Core Architectures for Linear Algebra Algorithms and Software," by grant 1045/09 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), by grant 2010231 from the U.S.-Israel Binational Science Foundation, by Microsoft (award 024263) and Intel (award 024894) funding, and by matching funding by U.C. Discovery (award DIG07-10227). Additional support comes from Par Lab affiliates Mathworks, National Instruments, Nokia, NVIDIA, Oracle, and Samsung. Research is also supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research award DE-SC0003959 and by contract DE-AC02-05CH11231; by the Applied Mathematics program awards DE-SC0004938 and DE-SC0010200; by the X-Stack program awards DE-SC0005136 and DE-SC0008700; and by DARPA grant HR0011-12-2-0016. Further support is provided by grant 1878/14 from the Israel Science Foundation and grant 3-10891 from the Ministry of Science and Technology, Israel, and by an appointment to the Sandia National Laboratories Truman Fellowship in National Security Science and Engineering, sponsored by Sandia Corporation (a wholly owned subsidiary of Lockheed Martin Corporation) as Operator of Sandia National Laboratories under its U.S. Department of Energy contract DE-AC04-94AL85000. This work was sponsored in part by the Ministry of Education and Science of the Russian Federation, Agreement N 14.607.21.0006 (unique identifier RFMEFI57714X0020).

http://www.siam.org/journals/simax/35-4/92906.html

†Sandia National Laboratories, Livermore, CA 94550 (gmballa@sandia.gov).

‡University of Tennessee, Knoxville, TN 37996 (dulceneia.becker@gmail.com, dongarra@eecs.utk.edu, iyamazak@eecs.utk.edu).

§University of California, Berkeley, CA 94720 (demmel@cs.berkeley.edu, odedsc@cs.berkeley.edu).

¶Oak Ridge National Laboratory, Oak Ridge, TN 37831 (dongarra@cs.utk.edu).

‖University of Manchester, M13 9PL, UK (jack.dongarra@manchester.ac.uk).

#Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (alex.druinsky@gmail.com).

††Tel Aviv University, Tel Aviv, Israel 69978 (inon.peled@gmail.com, sivan.toledo@gmail.com).

on the amount of arithmetic and early algorithmic optimizations focused on attempts to reduce this amount. These analyses are good predictors of actual running times only on computers with a flat fine-grained memory, in which bringing a word to an arithmetic unit costs about the same for all words.

Modern computers have multiple processors and memory systems that are far from flat and fine-grained. These architectural features have been used for decades now, but their effect on running times is becoming more and more significant [2]. In particular, communication between nodes in distributed-memory computers and communication between levels in memory hierarchies have become major determinants of performance.

The focus of this paper is a symmetric factorization algorithm that minimizes these communication costs. The algorithm is a block variant of Aasen's triangular tridiagonalization algorithm [1]. We designed the algorithm so that it can be implemented by a sequence of operations, each involving a constant number of b-by-b index-contiguous submatrices (blocks), where b is a tunable parameter. Most of these block operations perform $\Theta(b^3)$ arithmetic operations, which implies that the computation-to-communication ratio of the algorithm is $\Theta(b)$. Furthermore, since blocks are always contiguous in the row/column index space, they can be stored contiguously in memory, implying that each block operation requires only $O(1)$ data transfers that move $\Theta(b^2)$ words, which we refer to as *messages*.

For succinctness, in our description of the algorithm below, we coalesced individual b-by-b block operations into larger matrix-matrix operations. The individual block operations that correspond to each matrix operation will be clear from the description.

Matrix algorithms with such a structure usually perform well when implemented on sequential or shared-memory parallel computers. They can usually be adapted to distributed-memory parallel computers, but these adaptations are often intricate and far from trivial. The focus of this paper is on the block algorithm and its memory-hierarchy performance. A companion conference paper [5] described a shared-memory parallel implementation and its performance.[1] We do not discuss distributed-memory parallelization in this paper.

Models of computation and communication. Many computational models have been used in the literature for analyzing the communication efficiency of algorithms [10, 13, 19, 35, 36, 37]. In this paper we use a model that includes a processor connected to a fast memory containing M words that is too small to store all the algorithm's data structures. Data structures that do not fit within fast memory are stored in a slower, larger memory. Data transfers between the two memories occur in groups of n_{CL} contiguous aligned words (cache lines). Some algorithms are efficient only when cache lines are short (the so-called *tall-cache* assumption [13]). Because our algorithm transfers data using messages of $\Theta(b^2)$ contiguous words and because we can choose b, our algorithm does not rely on such assumptions. If we choose $b = \Theta(\sqrt{M})$, the algorithm is efficient even when n_{CL} is close to M; more formally, we assume only that $M = cn_{\mathrm{CL}}$, where $c \geq 4$. Algorithms that are asymptotically optimal in terms of the amount of data transferred between these memories are called *communication avoiding* [6] (there is also a generalization for distributed-memory computations that we do not use in this paper). Algorithms that are efficient in this model are also

[1] The main contributions of this paper relative to the companion conference paper [5] are complete numerical and complexity analyses of the algorithm; these did not appear in the conference paper.

efficient when cache lines are small (n_{CL} is small), and they do not depend on a fully associative cache; a four-way set-associative cache is sufficient.

Factorizations of symmetric indefinite matrices. Aasen's algorithm [1] factors

$$A = P^T L T L^T P \,,$$

where P is a permutation matrix selected for numerical stability, L is lower triangular (with ones on the diagonal and $|L_{ij}| \leq 1$), and T is symmetric and tridiagonal. The algorithm performs $n^3/3 + o(n^3)$ arithmetic operations, where n is the order of the matrices; it improves upon an earlier algorithm by Parlett and Reid that computes the same factorization in $2n^3/3 + o(n^3)$ operations [25]. The two algorithms were published in 1970 and 1971, respectively; neither is used extensively. A few years later Kaufman and Bunch discovered a similar factorization, one in which the tridiagonal T is replaced by a matrix that is block diagonal with 2-by-2 and 1-by-1 blocks [9].

Like other early factorizations, the algorithms of Aasen and of Parlett and Reid are not communication efficient even for very simple memory hierarchies. If $M < n^2/8$, both algorithms transfer $\Theta(n^3)$ words between fast and slow memory (even if cache lines are short). This is very inefficient. An implementation of the Bunch–Kaufman factorization that transfers only $O(\min(n^3, \frac{n^2}{M} \cdot n^2)) = O(n^4/M)$ words was later discovered,[2] and this implementation was included in LAPACK [3]. More recently, Rozložník, Shklarski, and Toledo discovered how to compute the factors produced by Aasen's algorithm (as well as Parlett and Reid's algorithm) with the same communication efficiency as LAPACK [29]. Reid and Scott also recently proposed a symmetric-indefinite factorization algorithm [28] whose design and communication requirements are similar to those of Rozložník et al. The pivot search in their algorithm can potentially increase communication to the same level of the original Aasen and Parlett–Reid algorithms, although they report in their paper that this is usually not the case. (This never happens in our algorithm.)

In this paper, we describe and analyze a stable symmetric factorization algorithm that is communication avoiding; it generates $O(n^3/\sqrt{M})$ cache misses even for $n_{\mathrm{CL}} = \Theta(M)$. In terms of communication, this is much more efficient than any existing symmetric-indefinite factorization. However, the algorithm produces a T that is banded rather than tridiagonal. To achieve this communication efficiency, the block size b and the half bandwidth of T must be $\Theta(\sqrt{M})$. We also show that the resulting T can be factored further in a way that is communication avoiding, and the resulting factorization allows linear systems of equations to be solved quickly.

Our algorithm is fundamentally a block version of Aasen's algorithm. While the methodology of producing block-matrix algorithms from element-by-element algorithms is well understood, applying it to this case proved to be challenging. The first block Aasen algorithm that we designed proved highly unstable. In Aasen's original algorithm, diagonal elements of T are computed by solving a scalar equation. In the block version, this scalar equation transforms into a linear system of equations whose solution is a diagonal block of T, which is symmetric. But the system itself is unsymmetric and the symmetry of the solution is implicit. When the system is

[2]The $O(n^4/M)$ bound is attained when $M \geq n$. In this regime, the algorithm factors panels of roughly M/n columns. Updating a trailing submatrix of dimension $\Theta(n)$ after the factorization of $\Theta(n/(M/n))$ such panels transfers $\Theta(n^4/M)$ words. When $M < n$, the algorithm transfers $O(n^3)$ words; in this regime the fast memory has no significant beneficial effect.

solved in floating-point arithmetic, the computed block of T can have a nonnegligible skew-symmetric component in addition to its symmetric part, and this excites an instability. To address this difficulty, we designed an algorithm that produces a symmetric T even in floating point. The algorithm reduces the computation of the diagonal blocks of T to the solution of symmetric two-sided triangular systems of linear equations; an algorithm for solving such systems is described in section 2.4.

The rest of the paper is organized as follows. We present the algorithm in section 2. Section 3 analyzes the stability of the algorithm, and section 4 its computation and communication complexity. We present a communication lower bound for the algorithm in section 5; this lower bound, along with the upper bound presented in section 4, establishes the asymptotic communication optimality of the algorithm. Numerical experiments presented in section 6 provide additional insights into the behavior of the algorithm. Section 7 presents our conclusions from this research.

2. The algorithm. To keep the notation simple, we initially ignore pivoting in the description of the algorithm. The algorithm factors the n-by-n matrix A into

$$A = LTL^T ,$$

where L is unit lower triangular and T is symmetric and banded with half bandwidth b (i.e., $T_{ij} = 0$ if $|i - j| > b$). The algorithm processes the matrices in aligned blocks of size b-by-b (except for the blocks that correspond to the trailing block row and column, which may have fewer than b rows and columns, respectively). The algorithm is a block version of Aasen's algorithm, so we view T as a block tridiagonal matrix with triangular blocks in the positions immediately adjacent to the main diagonal.

To describe the algorithm we must specify three auxiliary matrices. The first is a block upper-triangular matrix with symmetric diagonal blocks R that we require to satisfy

$$R^T + R = T .$$

The blocks above the diagonal in R are the same as the corresponding blocks in T, the diagonal blocks of R are scaled copies of those of T (with scaling $1/2$), and the blocks below the diagonal in R are zero (unlike in T, which is symmetric). The other two matrices are denoted by W and H and are required to satisfy

$$W = RL^T , \qquad H = TL^T .$$

Aasen's original algorithm also computes H (forming it was the key step that allowed Aasen to eliminate half the arithmetic operations from Parlett and Reid's algorithm), but it does not compute W.

We present the algorithm in the form of block-matrix equations, each of which defines one or two sets of blocks in these matrices. The blocks that are computed from each equation are underlined. We use capital I and capital J to denote block indices, and we denote the block dimension of all the matrices by $N = \lceil n/b \rceil$. We denote blocks of matrices using indexed notation with block indices. For example, the submatrix that is specified by $A_{1+(I-1)b:Ib,1+(J-1)b:Jb}$ in scalar-index colon notation is denoted $A_{I,J}$.

The initialization step of the algorithm assigns

$$\underline{L_{1:N,1}} = (\text{identity matrix})_{1:N,1} .$$

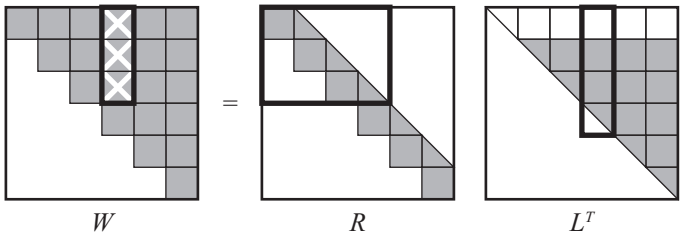

FIG. 1. *An illustration of computing superdiagonal blocks of W via matrix multiplication in* (AA1). *Here $N = 6$ and $J = 4$. The blocks that participate in the equation are enclosed in thick rectangles, and the blocks that are computed using this equation are crossed. The first block row of L^T is equal to the corresponding block row of the identity matrix. The same notation is used in other diagrams in this section.*

That is, the first b columns of L have ones on the diagonal and zeros everywhere else. After this initialization, the algorithm computes a block column of each of the matrices in every step. Step J computes column $J+1$ of L and columns J of T, H, and W (diagonal blocks of W are never needed so they are not computed) according to the following formulas, which we derive in section 2.1:

$$\text{(AA1)} \quad \underline{W_{1:J-1,J}} = R_{1:J-1,1:J}(L_{J,1:J})^T \,,$$

$$\text{(AA2)} \quad A_{J,J} = L_{J,1:J-1}W_{1:J-1,J} + (W_{1:J-1,J})^T(L_{J,1:J-1})^T + L_{J,J}\underline{T_{J,J}}(L_{J,J})^T \,,$$

$$\text{(AA3)} \quad \underline{H_{1:J,J}} = T_{1:J,1:J}(L_{J,1:J})^T \,,$$

$$\text{(AA4)} \quad A_{J+1:N,J} = L_{J+1:N,1:J}H_{1:J,J} + \underline{L_{J+1:N,J+1}}\,\underline{H_{J+1,J}} \,,$$

$$\text{(AA5)} \quad H_{J+1,J} = \underline{T_{J+1,J}}(L_{J,J})^T \,.$$

2.1. Correctness. We now show that the algorithm is correct. Verifying that the blocks that are computed in each equation depend only on blocks that are already known is trivial. Therefore, we focus on showing that $A = LTL^T$ whenever L and T are computed in exact arithmetic. The analysis also constitutes a more detailed presentation of the algorithm.

Equation (AA1) computes a column of W by multiplying two submatrices, using the equation $W = RL^T$, as shown in Figure 1. This guarantees that $W_{I,J} = (RL^T)_{I,J}$ for all $I < J$. The diagonal blocks of W are not computed, and we define for convenience $W_{J,J} = (RL^T)_{J,J}$ for all J. Because all other blocks of W and RL^T are zero, $W = RL^T$.

Equation (AA2) computes a diagonal block of T by solving a two-sided triangular linear system, as shown in Figure 2. This linear system can be solved by one of the existing solvers which we describe in section 2.4. The right-hand side matrix in this system,

$$A_{J,J} - L_{J,1:J-1}W_{1:J-1,J} - (L_{J,1:J-1}W_{1:J-1,J})^T \,,$$

must be computed symmetrically; this is done using the BLAS routine SYR2K [12]. The equation guarantees that

$$A_{J,J} = L_{J,1:J-1}W_{1:J-1,J} + (L_{J,1:J-1}W_{1:J-1,J})^T + L_{J,J}T_{J,J}(L_{J,J})^T \,.$$

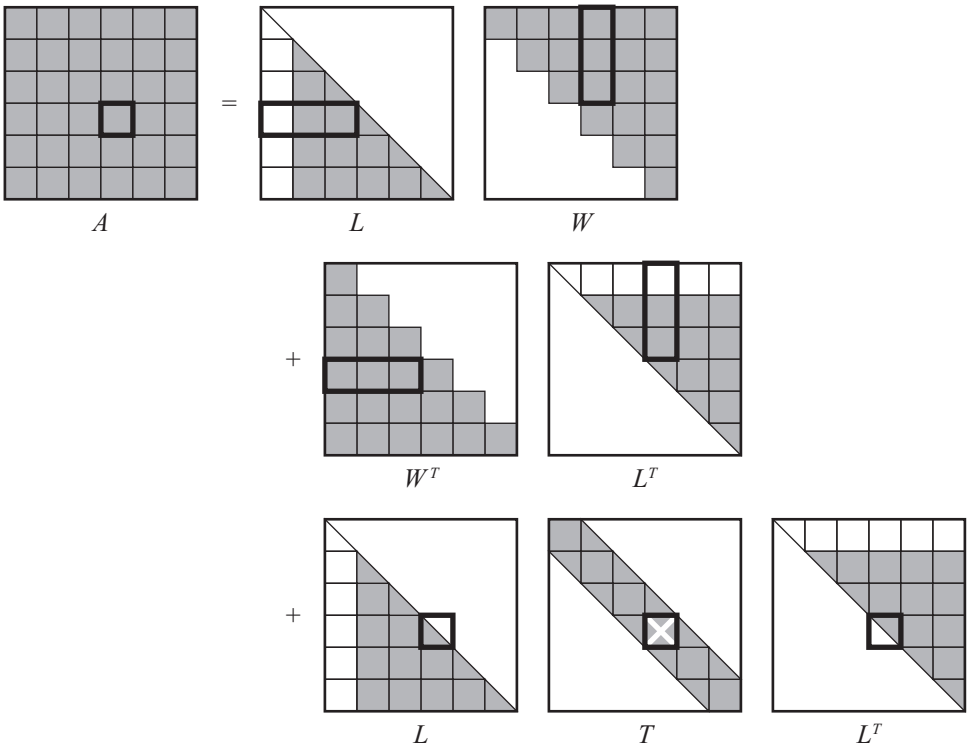

FIG. 2. *Computing a diagonal block of T in* (AA2) *by updating the corresponding block of A and solving a two-sided triangular system. The letters below each matrix describe only the matrices involved in the expression for $A_{J,J}$; they do not constitute a matrix equation.*

By noting that

$$\begin{aligned} L_{J,J}T_{J,J}(L_{J,J})^T &= L_{J,J}(R_{J,J} + (R_{J,J})^T)(L_{J,J})^T \\ &= L_{J,J}R_{J,J}(L_{J,J})^T + (L_{J,J}R_{J,J}(L_{J,J})^T)^T \end{aligned}$$

and that the diagonal blocks of $W = RL^T$ are $W_{J,J} = R_{J,J}(L_{J,J})^T$, as shown in Figure 3, we can transform (AA2) into

$$\begin{aligned} A_{J,J} &= L_{J,1:J-1}W_{1:J-1,J} + (L_{J,1:J-1}W_{1:J-1,J})^T + L_{J,J}W_{J,J} + (L_{J,J}W_{J,J})^T \\ &= L_{J,1:J}W_{1:J,J} + (L_{J,1:J}W_{1:J,J})^T \\ &= (LW + (LW)^T)_{J,J} \ . \end{aligned}$$

Substituting $W = RL^T$ we obtain

$$\begin{aligned} A_{J,J} &= (LRL^T + (LRL^T)^T)_{J,J} \\ &= (L(R + R^T)L^T)_{J,J} \\ &= (LTL^T)_{J,J} \ . \end{aligned}$$

Equation (AA3) computes a block column of H, except for the subdiagonal block, by multiplying matrices, as shown in Figure 4. Equation (AA4) multiplies blocks of

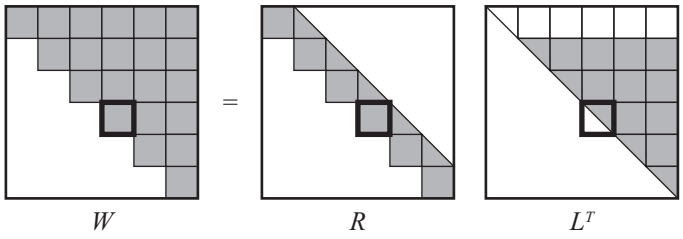

FIG. 3. *An expression for the diagonal blocks of* W.

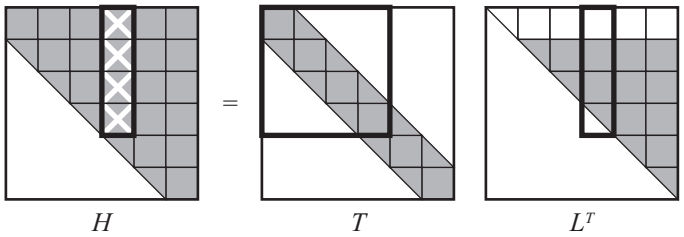

FIG. 4. *Computing blocks of* H *via matrix multiplication in* (AA3).

L and H, subtracts the product from a block of A, and factors the difference using an LU factorization, as shown in Figure 5. This equation represents a left-looking update operation followed by a panel factorization. Equation (AA5) solves a triangular linear system with a triangular right-hand side to compute a subdiagonal block of T, as shown in Figure 6.

Equations (AA3) and (AA5) guarantee that $H_{I,J} = (TL^T)_{I,J}$ for all $I \leq J$ and for all $I = J + 1$, respectively, and because all other blocks of H and TL^T are zero, $H = TL^T$. Equation (AA4) makes sure that $A_{I,J} = (LH)_{I,J}$ for all $I > J$, and substituting $H = TL^T$ shows that $A_{I,J} = (LTL^T)_{I,J}$ for all $I > J$. Because both A and LTL^T are symmetric, this holds for all $I < J$ as well, and thus $A = LTL^T$.

2.2. Pivoting. Just like the original elementwise Aasen algorithm, without pivoting, the new block algorithm can break down or become unstable. To stabilize the algorithm, in (AA4), the blocks $L_{J+1:N,J+1}$ and $H_{J+1,J}$ are computed using an LU factorization with row pivoting, meaning that step J factors

$$L_{J+1:N,J+1}H_{J+1,J} = P^{(J)}\left(A_{J+1:N,J} - L_{J+1:N,1:J}H_{1:J,J}\right) , \tag{2.1}$$

where $P^{(J)}$ is a permutation matrix. There are several ways of computing $P^{(J)}$ in order to ensure the communication optimality of the overall algorithm; we list and analyze them in section 4.2.2. We use the pivoting strategy of the LU factorization in (AA4) as a black box, and other than this we do not require any additional pivoting to ensure the numerical stability of our algorithm.

Obtaining a row reordering from the factorization in (AA4) means that instead of computing a factorization of A, we are computing the factorization of $P_J A \left(P_J\right)^T$, where

$$P_J = \begin{bmatrix} I & \\ & P^{(J)} \end{bmatrix}$$

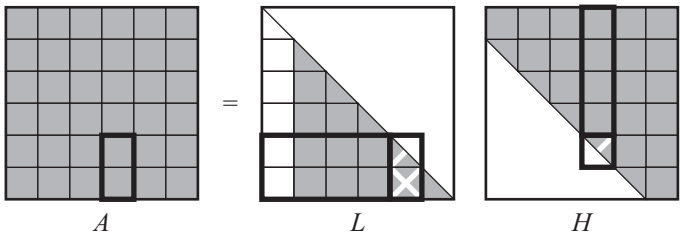

FIG. 5. *Computing a block column of L and a subdiagonal block of H in* (AA4) *via the LU factorization of an updated submatrix of A.*

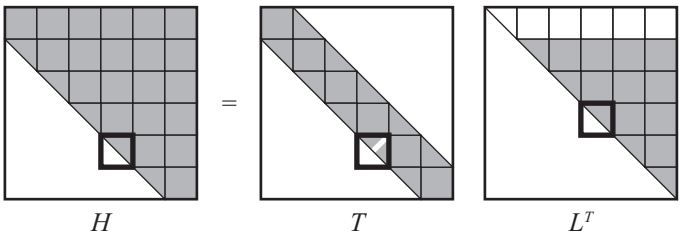

FIG. 6. *Computing a block of T by solving a triangular system in* (AA5).

and I is the identity matrix of order Jb. To make sure that subsequent steps of the algorithm take this change of plans into account, we must change the ordering of A to reflect the effect of P_J on its rows and columns. We accomplish this by replacing the trailing submatrix $A_{J+1:N,J+1:N}$ with its reordered version,

$$P^{(J)} A_{J+1:N,J+1:N} \left(P^{(J)}\right)^T .$$

Although P_J changes the ordering not just of the trailing submatrix but also of the first J block rows and columns of A, we do not need to explicitly reorder these rows and columns because they have already been processed and do not participate in the rest of the computation.

The introduction of pivoting in (AA4) and the consequent reordering of A in step J may have rendered some of the equations (AA1)–(AA5) invalid, and this has happened not just for the instances of these equations that correspond to step J, but also for those that correspond to steps K such that $K < J$. The already-computed blocks of the factorization must be adjusted to restore the validity of the equations.

Consider first the instance of (AA4) that corresponds to step K. Our postmultiplication of A by $(P_J)^T$ has no effect on the first J block columns of A, and therefore the effect that the reordering of A has on the equation amounts to premultiplying the equation's left-hand side with a permutation matrix that corresponds to $P^{(J)}$. To restore the validity of the equation, we must apply the same operation to its right-hand side, premultiplying the right-hand side by the same matrix and thereby reordering the rows of the already-computed part of L. This is accomplished by replacing the matrix $L_{J+1:N,1:J}$ with $P^{(J)} L_{J+1:N,1:J}$, which restores the validity of (AA4) for $K = 1, 2, \ldots, J-1$. The same operation also restores the validity for $K = J$; this follows from (2.1).

By reordering the already-computed part of L, we have restored the validity of (AA4), but what about the other equations? It turns out that introducing pivoting

has no effect on their validity. This is because our reordering applies only to block rows or columns $J+1, J+2, \ldots, N$ of A and L, whereas (AA1)–(AA3) and (AA5) only feature the first J diagonal blocks of A and the first J block rows of L, to which our reordering does not apply. Therefore, (AA1)–(AA5) are satisfied by the reordered matrices, and the arguments that we used in section 2.1 to show that $A = LTL^T$ can also be used now, showing that $PAP^T = LTL^T$, where $P = P_{N-1} \cdots P_2 P_1$.

Applying the permutation to L and to the trailing submatrix is not trivial to do in a communication-avoiding way, especially since only the upper or lower triangle of the trailing submatrix is stored. The details are explained in section 4.2.3.

2.3. Computing W and H. As we show in section 4, the arithmetic and communication costs of our algorithm are asymptotically dominated by the computation that corresponds to (AA4). Nevertheless, if optimizing the computation that corresponds to the other equations can yield any savings, then pursuing such optimizations would be desirable from a practical standpoint. It turns out that savings are possible in (AA1) and (AA3). These equations state that individual blocks of H and W are computed according to the formulas

$$\underline{H_{I,J}} = T_{I,I-1} \left(L_{J,I-1}\right)^T + T_{I,I} \left(L_{J,I}\right)^T + T_{I,I+1} \left(L_{J,I+1}\right)^T$$

and

$$\begin{aligned} \underline{W_{I,J}} &= R_{I,I} \left(L_{J,I}\right)^T + R_{I,I+1} \left(L_{J,I+1}\right)^T \\ &= 0.5\, T_{I,I} \left(L_{J,I}\right)^T + T_{I,I+1} \left(L_{J,I+1}\right)^T \end{aligned}$$

for all $I < J$. (We encourage the reader to review Figures 1 and 4 for a visualization of these relations.) The blocks $T_{I,I} \left(L_{J,I}\right)^T$ and $T_{I,I+1} \left(L_{J,I+1}\right)^T$ appear in these equations twice but need to be computed only once. By avoiding the recomputation of these blocks, the number of b-by-b matrix products required to compute $H_{I,J}$ and $W_{I,J}$ is reduced from five to three, thereby making the computation of W essentially free.

A pseudocode description of the algorithm that includes this optimization is given in Algorithm 1. Our pseudocode includes matrix update operations in lines 12 and 18 that involve indexing expressions where the starting index can be greater than the ending index. For example, line 12 contains the index set $2{:}J{-}1$, which for $J = 1$ is translated to 2:0. We interpret such indexing expressions as indicating empty index sets, which correspond to empty updates that return the updated matrix without change, similarly to how such expressions are interpreted in MATLAB.

Our implementation of the algorithm requires a workspace of 5 panels of size n-by-b; further details are given in [5, section IV.E].

2.4. Solving two-sided triangular linear systems. We now describe the procedure that solves the two-sided triangular linear system in (AA2). The method is a new recursive and communication-avoiding generalization of an existing algorithm called SYGST, which is used to reduce symmetric generalized eigenproblems to standard eigenproblems in LAPACK and ScaLAPACK [8, 31]. Other solvers that produce a symmetric solution would also be suitable for the task. Examples of such solvers are the subroutine REDUC in EISPACK [24, 32] and the algorithms implemented in the Elemental library [26, 27]. Because we apply the solver to block-sized problems, its arithmetic and communication costs do not have a substantial impact on the overall costs of the algorithm. The stability of the solver is important, and we analyze

Algorithm 1. Block Aasen factorization.

Input: the matrix A
Output: the factors L and T and the permutation matrix P
Initialize: $P = I$, $L_{1,1} = I$, $L_{2:N,1} = 0$

for $J \leftarrow 1, 2, \ldots, N$ **do**

  for $I \leftarrow 2, 3, \ldots, J-1$ **do** ▷ Eq. (AA1) and part of Eq. (AA3)
    $X \leftarrow T_{I,I-1} \left(L_{J,I-1}\right)^T$
    $Y \leftarrow T_{I,I} \left(L_{J,I}\right)^T$
    $Z \leftarrow T_{I,I+1} \left(L_{J,I+1}\right)^T$
    $W_{I,J} \leftarrow 0.5Y + Z$
    $H_{I,J} \leftarrow X + Y + Z$
  end for

  ▷ Eq. (AA2)
  $C \leftarrow A_{J,J} - L_{J,2:J-1} W_{2:J-1,J} - \left(L_{J,2:J-1} W_{2:J-1,J}\right)^T$ ▷ BLAS call SYR2K
  $T_{J,J} \leftarrow \left(L_{J,J}\right)^{-1} C \left(L_{J,J}\right)^{-T}$ ▷ LAPACK call SYGST

  if $J < N$ **then**

    if $J > 1$ **then** ▷ the rest of Eq. (AA3)
      $H_{J,J} \leftarrow T_{J,J-1} \left(L_{J,J-1}\right)^T + T_{J,J} \left(L_{J,J}\right)^T$
    end if

    $E \leftarrow A_{J+1:N,J} - L_{J+1:N,2:J} H_{2:J,J}$ ▷ Eq. (AA4)
    $\left[L_{J+1:N,J+1},\ H_{J+1,J},\ P^{(J)}\right] \leftarrow \mathrm{LU}(E)$ ▷ LU factorization

    ▷ Eq. (AA5)
    $T_{J+1,J} \leftarrow H_{J+1,J} \left(L_{J,J}\right)^{-T}$ ▷ LAPACK call TRSM
    $T_{J,J+1} \leftarrow \left(T_{J+1,J}\right)^T$

    $L_{J+1:N,2:J} \leftarrow P^{(J)} L_{J+1:N,2:J}$ ▷ back-pivoting
    $A_{J+1:N,J+1:N} \leftarrow P^{(J)} A_{J+1:N,J+1:N} (P^{(J)})^T$ ▷ forward-pivoting
    $P_{J+1:N,1:N} \leftarrow P^{(J)} P_{J+1:N,1:N}$
  end if
end for

the stability of SYGST and of our generalization in section 3.2. So long as the solver satisfies a bound similar to the one that we prove there, the impact on the overall algorithm is limited to the size of the constant in the backward stability bound.

Let us describe the algorithm. The equation that defines $T_{J,J}$ is of the form $KXK^T = B$, where K and B are known matrices of dimensions m-by-m, K is lower triangular, and B is symmetric. A trivial way to solve such systems is using a conventional triangular solver twice. That is, first solve for $K^{-1}B$ and then solve for $X = (K^{-1}B)K^{-T}$. This method produces a solution X that is not symmetric in floating-point arithmetic, and is thus not suitable for use in the block Aasen algorithm.

Another approach, which leads to a symmetric X and which performs only half the arithmetic, is an algorithm that we now describe. We partition the matrices that we introduce in this section such that they are all 2-by-2 block matrices with first diagonal blocks of dimensions c-by-c and second diagonal blocks of dimensions $(m-c)$-by-$(m-c)$. To describe the algorithm we must define an auxiliary matrix Y, which we require to be a block upper-triangular matrix with symmetric diagonal blocks that satisfies

$$X = Y^T + Y \ .$$

Such a matrix must have the form

$$\begin{bmatrix} Y_{11} & Y_{12} \\ Y_{21} & Y_{22} \end{bmatrix} = \begin{bmatrix} 0.5X_{11} & X_{12} \\ 0 & 0.5X_{22} \end{bmatrix} \ .$$

We also need two additional auxiliary matrices G and V, which we will require to satisfy

$$G = XK^T \ , \qquad V = YK^T \ .$$

The algorithm works by solving for the underlined blocks in the following equations:

$$\text{(ST1)} \qquad B_{11} = K_{11}\underline{X_{11}}(K_{11})^T \ ,$$

$$\text{(ST2)} \qquad B_{12} = K_{11}\underline{G_{12}} \ ,$$

$$\text{(ST3)} \qquad G_{12} = 0.5X_{11}(K_{21})^T + \underline{V_{12}} \ ,$$

$$\text{(ST4)} \qquad V_{12} = 0.5X_{11}(K_{21})^T + \underline{X_{12}}(K_{22})^T \ ,$$

$$\text{(ST5)} \qquad B_{22} = K_{21}V_{12} + (K_{21}V_{12})^T + K_{22}\underline{X_{22}}(K_{22})^T \ .$$

The key in this algorithm is to compute $B_{22} - K_{21}V_{12} - (K_{21}V_{12})^T$ in (ST5) symmetrically, which allows the algorithm to compute X_{22} symmetrically as well. Note that the block $0.5X_{11}(K_{21})^T$ is computed twice; the algorithm trades off additional computation for a reduction in workspace requirements, as we explain further at the end of this section.

We derived (ST1)–(ST5) by considering specific blocks of the equations

$$B = KXK^T \ , \quad B = KG \ , \quad B = KV + (KV)^T \ , \quad G = Y^TK^T + V \ , \quad V = YK^T \ .$$

The derivation is described by diagrams in Figures 7–11.

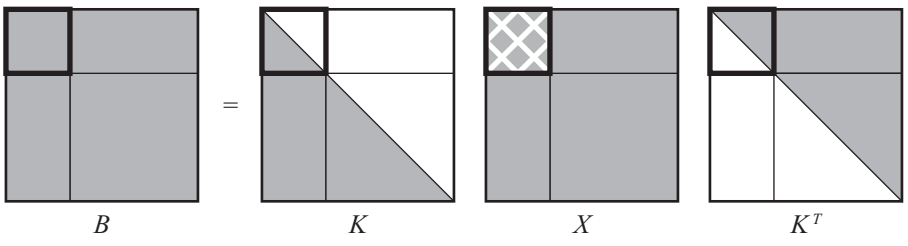

FIG. 7. *Computing the first diagonal block of X by recursively solving a smaller two-sided triangular system in step* (ST1).

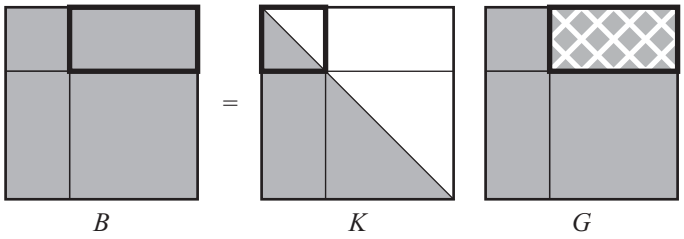

FIG. 8. *Solving a triangular system to compute the superdiagonal block of G in step* (ST2).

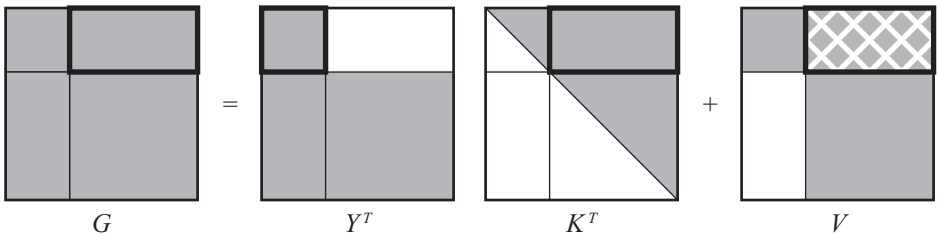

FIG. 9. *Computing the superdiagonal block of V in step* (ST3) *by updating the corresponding block of G.*

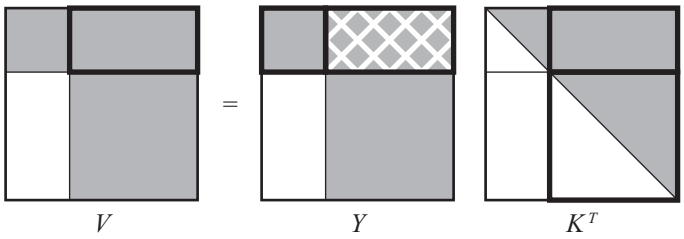

FIG. 10. *Computing the superdiagonal block of Y in step* (ST4) *by updating the corresponding block of V and solving a triangular system.*

We will now verify the correctness of the algorithm, meaning that $KXK^T = B$ whenever X is computed in exact arithmetic. The algorithm computes the diagonal and superdiagonal blocks of X and the superdiagonal blocks of G and V. The subdiagonal block of X is not computed because X is symmetric, and therefore that block is not needed. The diagonal and subdiagonal blocks of G and V are also not computed, and thus we are free to define them so that our notation is simplified. We define the uncomputed blocks of G and V such that the corresponding blocks of the equations $G = XK^T$ and $V = YK^T$ hold.

We start by verifying that $(KXK^T)_{12} = B_{12}$. Step (ST4) makes sure that $V_{12} = (YK^T)_{12}$ and thus $V = YK^T$, due to the way we defined the uncomputed blocks of V. Step (ST3) guarantees that $G_{12} = (Y^TK^T + V)_{12}$. Substituting $V = YK^T$ and noting that $Y^TK^T + YK^T = XK^T$ shows that $G_{12} = (XK^T)_{12}$ and thus $G = XK^T$, again due to our definition of the uncomputed blocks of G. Finally, step (ST2) makes sure that $B_{12} = (KG)_{12}$, and substituting $G = XK^T$ shows that $B_{12} = (KXK^T)_{12}$.

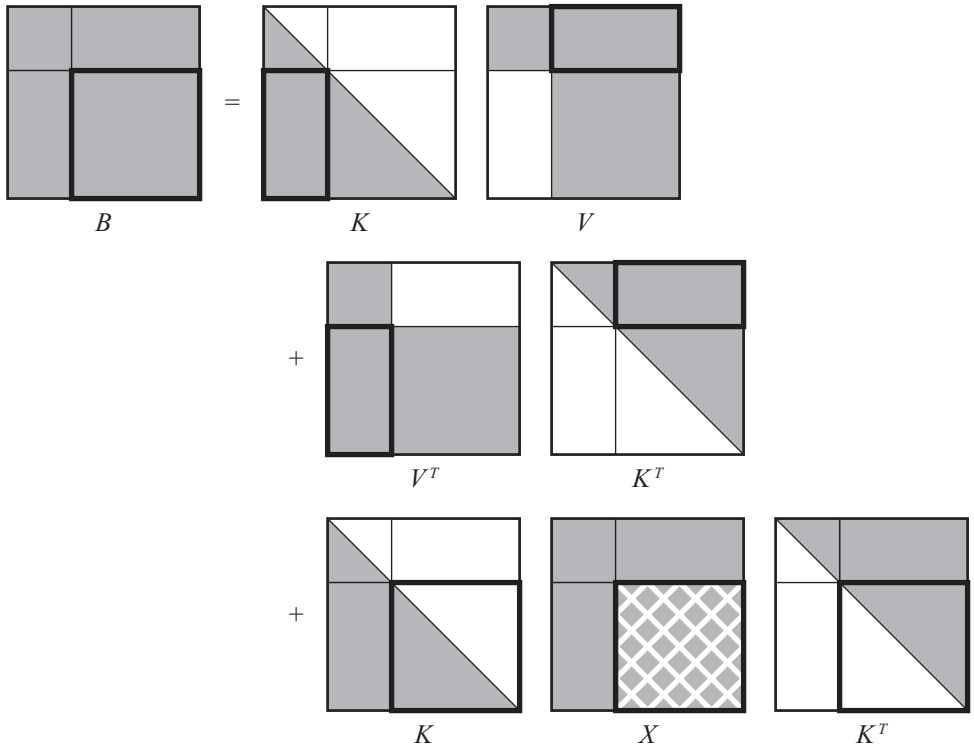

FIG. 11. *Computing the second diagonal block of X in step* (ST5) *by updating the corresponding block of B and solving a smaller two-sided triangular system.*

Next we verify that $(KXK^T)_{22} = B_{22}$ by transforming (ST5):

$$\begin{aligned}
B_{22} &= K_{21}V_{12} + (K_{21}V_{12})^T + K_{22}X_{22}(K_{22})^T \\
&= K_{21}V_{12} + (K_{21}V_{12})^T + K_{22}Y_{22}(K_{22})^T + (K_{22}Y_{22}(K_{22})^T)^T \\
&= K_{21}V_{12} + (K_{21}V_{12})^T + K_{22}V_{22} + (K_{22}V_{22})^T \\
&= (KV + (KV)^T)_{22} \\
&= (KYK^T + KY^TK^T)_{22} \\
&= (K(Y + Y^T)K^T)_{22} \\
&= (KXK^T)_{22} \, .
\end{aligned}$$

Finally, step (ST1) corresponds to recursively solving a smaller system. Using induction on the size of the system, we may conclude that $(KXK^T)_{11} = B_{11}$ and thus $KXK^T = B$.

We did not specify the dimensions of the blocks; different choices yield different algorithms. If we choose $c = 1$, we end up with an algorithm that computes the columns of X one at a time, in which (ST5) iterates over remaining columns. This version is called SYGS2 in LAPACK and ScaLAPACK. The costs in this partitioning are dominated by the triangular solve in (ST4) and the symmetric update in (ST5), which require $(m-i)^2$ and $2(m-i)^2$ flops, respectively. Thus, the leading term in

flop cost is given by

$$F_1(m) = \sum_{i=1}^{m-1} 3(m-i)^2 = 3\sum_{i=1}^{m-1} i^2 = m^3 + o(m^3) .$$

If instead of $c = 1$ we choose some fixed $c > 1$, we obtain SYGST, which works by computing a total of c columns of X at a time. Step (ST1) here corresponds to a call to SYGS2, and step (ST5) iterates over the remaining block columns. As long as $m \gg c$, the costs are again dominated by the triangular solve in (ST4) and the symmetric update in (ST5), which require $(m-ic)^2 c$ and $2(m-ic)^2 c$ flops, respectively (here i iterates over block columns). Thus, the leading term in the flop cost is given by

$$F_c(m) = \sum_{i=1}^{m/c-1} 3(m-ic)^2 c = 3c^3 \sum_{i=1}^{m/c-1} i^2 = m^3 + o(m^3) .$$

We can also formulate the algorithm recursively, with X_{11} being $\lfloor \frac{m}{2} \rfloor \times \lfloor \frac{m}{2} \rfloor$. This recursive version is new, and it is communication avoiding and cache oblivious even for large matrices (we use it only on blocks, so this is not useful for the block Aasen algorithm). Steps (ST1) and (ST5) are recursive calls. The triangular solves in steps (ST2) and (ST4) and the block multiplications in steps (ST3), (ST4), and (ST5) all contribute to the leading term in the flop cost. The product $X_{11}(K_{21})^T$ is a SYMM BLAS which costs $2(m/2)^3$. The triangular solves are TRSM calls that cost $(m/2)^3$ each, and the product $K_{21}V_{12}$ is a GEMM call that costs $2(m/2)^3$ (after computing and subtracting the product we also subtract its transpose). If we store the block $X_{11}(K_{21})^T$ in step (ST3) and reuse it in step (ST4), the recurrence is

$$F_R(m) = 2F_R\left(\frac{m}{2}\right) + 6\left(\frac{m}{2}\right)^3 ,$$

which again solves to

$$F_R(m) = \frac{3}{4}m^3 \sum_{i=0}^{\log m - 1} \left(\frac{1}{4}\right)^i = \frac{3}{4} \cdot \frac{4}{3}m^3 + o(m^3) = m^3 + o(m^3) .$$

Storing $X_{11}(K_{21})^T$ requires a workspace of $m^2/4$ words. If we choose to recompute that block in order to run the algorithm in place, the flop count is increased to $(4/3)m^3 + o(m^3)$.

2.5. The second phase of the algorithm: Factoring T. There are several single-pass algorithms that efficiently factor a banded symmetric matrix. All of these algorithms process $O(b)$ rows and columns at a time, so if we choose a small enough $b = \Theta(\sqrt{M})$, where M is the size of fast memory, the total number of cache misses that these algorithms generate is $O(bn/M)$, which makes them communication avoiding (because the size of input and output is $O(bn)$). Furthermore, given such a one-pass algorithm, the two phases can be fused together so that every new batch of $O(b)$ rows and columns of T is factorized by the second phase as soon as it is produced by the first one, before that batch is evicted from fast memory.

Single-pass algorithms for banded matrices include the unsymmetric banded LU factorization with partial pivoting, the banded QR factorization, Kaufman's retraction algorithm [22], and Irony and Toledo's snap-back algorithm [20]. The banded

LU and QR factorizations are well known and require no further comments. The two other methods produce a symmetric factorization. They minimize growth in the factorization while preserving the band structure using somewhat complex strategies to annihilate out-of-band fill. All four methods produce a factorization that is essentially banded with bandwidth $O(b)$, although the factors are represented in implicit economical forms and are not formed explicitly because their explicit forms are not banded. All of these factorizations can be used to solve linear systems of equations using $O(bn)$ arithmetic per right-hand side and $O(bn/M)$ cache misses (for up to $O(\sqrt{M})$ right-hand sides).

3. Numerical stability. We analyze the stability of the factorization of PAP^T, where P is the permutation matrix generated by the selection of pivots. We assume in the analysis that the matrix has been prepermuted so the algorithm is applied directly to PAP^T (rather than to A) and that it never pivots. The sequence of arithmetic operations in such a run of the algorithm is identical to that of the pivoting version applied to A, except perhaps for the order of summation in inner products. Our analysis does not depend on this ordering, so our results apply to the pivoting version.

We do not make any assumptions about the permutation P, and in fact our analysis applies even if there is no pivoting. As a consequence, Theorem 3.10, which we prove below, guarantees backward stability, but the guarantee is conditioned on having moderate growth in the factorization. The role of pivoting is to control growth, as in Aasen's original algorithm and in LU with partial pivoting. We discuss growth in more detail in section 3.5.

Our model of floating-point arithmetic is

$$\text{(FL1)} \qquad fl\left(x \operatorname{op} y\right) = \left(x \operatorname{op} y\right)\left(1+\delta\right) \ , \qquad |\delta| \le u \ , \qquad \operatorname{op} = +, -, \times, \div \ ,$$

where u is unit roundoff [18, section 2.2]. We also assume that 0.5 is a floating-point number and that

$$\text{(FL2)} \qquad fl\left(0.5x\right) = 0.5x \ .$$

In the following, we use the notation $|X|$, where X is a matrix, to refer to the elementwise absolute value, and inequalities of the form $|X| \le |Y|$, where X and Y are two matrices of the same dimension, to denote elementwise inequalities.

3.1. Known stability results. We begin by citing a few lemmas of floating-point error analysis, all of which are either well known or can be easily derived from well-known results.

We use the notation $\gamma_n = nu/\left(1-nu\right)$ for any positive n. The following lemma provides a rule for manipulating expressions involving γ_n or quantities bounded by it.

LEMMA 3.1 (see [18, Lemma 3.3]). *The bound*

$$\gamma_m + \gamma_n + \gamma_m\gamma_n \le \gamma_{m+n}$$

holds. Furthermore, if θ_m and θ_n are such that $|\theta_m| \le \gamma_m$ and $|\theta_n| \le \gamma_n$ and if θ_{m+n} is such that

$$\left(1+\theta_m\right)\left(1+\theta_n\right) = 1+\theta_{m+n} \ ,$$

then

$$|\theta_{m+n}| \le \gamma_{m+n} \ .$$

The following lemma provides a bound on the accuracy of matrix-matrix products. In our analysis we assume that matrices are multiplied using the conventional method, as opposed to Strassen's algorithm or any related scheme.

LEMMA 3.2 (see [18, section 3.5]). *Let A and B be m-by-p and p-by-n matrices, respectively. If the product $X = AB$ is formed in floating-point arithmetic, then*

$$X = AB + \Delta\,, \qquad |\Delta| \le \gamma_p\,|A|\,|B|\ .$$

The following two lemmas also deal with matrix-matrix multiplication. Their proofs are similar to the proof of Lemma 8.4 in [18], with the only difference stemming from the possible scaling by 0.5 in Lemma 3.3. The assumption (FL2) in our model guarantees that this scaling has no effect on the ultimate bound.

LEMMA 3.3. *Let A, B, and C be matrices of dimensions m-by-p, p-by-n, and m-by-n, respectively, and let α be one of the scalars 0.5 and 1. If the matrix $X = C - \alpha AB$ is formed in floating-point arithmetic, then*

$$C = \alpha AB + X + \Delta\,, \qquad |\Delta| \le \gamma_p\,(\alpha\,|A|\,|B| + |X|)\ .$$

LEMMA 3.4. *Let A, B, and C be matrices of dimensions m-by-p, p-by-m, and m-by-m, respectively. If the matrix $X = C - AB - (AB)^T$ is formed in floating-point arithmetic, then*

$$C = AB + (AB)^T + X + \Delta\,, \qquad |\Delta| \le \gamma_{2p}\left(|A|\,|B| + (|A|\,|B|)^T + |X|\right)\ .$$

Finally, the following two lemmas provide bounds on the accuracy of triangular solves and of the LU factorization.

LEMMA 3.5 (see [18, section 8.1]). *Let T and B be matrices of dimensions m-by-m and m-by-n, respectively, and assume that T is triangular. If the m-by-n matrix X is computed by solving the system $TX = B$ using substitution in floating-point arithmetic, then*

$$TX = B + \Delta\,, \qquad |\Delta| \le \gamma_m\,|T|\,|X|\ .$$

Furthermore, if the system being solved is $XT = B$ and the dimensions of X and B are n-by-m, then

$$XT = B + \Delta\,, \qquad |\Delta| \le \gamma_m\,|X|\,|T|\ .$$

LEMMA 3.6 (see [18, section 9.3]). *Let A be an m-by-n matrix and let $r = \min\{m, n\}$. If L and U are the LU factors of A, computed in floating-point arithmetic, then*

$$A = LU + \Delta\,, \qquad |\Delta| \le \gamma_r\,|L|\,|U|\ .$$

3.2. The stability of the two-sided triangular solver. Our notation in this section is the same as in section 2.4. The matrix X and the superdiagonal blocks of G and V represent the actually computed floating-point matrices. We use Y to denote the exact matrix

$$Y = \begin{bmatrix} 0.5X_{11} & X_{12} \\ 0 & 0.5X_{22} \end{bmatrix},$$

and we define the diagonal and subdiagonal blocks of G and V such that the corresponding blocks of $G = XK^T$ and $V = YK^T$ hold.

LEMMA 3.7. *If the two-sided triangular system $KXK^T = B$ is solved in floating-point arithmetic, then*

$$KXK^T = B + \Delta\,, \qquad |\Delta| \le \gamma_{3m-1}|K||X||K^T|\,.$$

Proof. We prove by induction on m that $|\Delta| \le \gamma_{f(m)}|K||X||K^T|$ with $f(m) = 3m - 1$. In the base case of the induction, we are solving 1-by-1 systems and the bound clearly holds. Next, assume that the inductive hypothesis is true when we solve systems of order up to $m-1$, and let us prove that it is also satisfied for systems of order m.

Let the matrices $\Delta^{(t)}$ for $t = 1, 2, \ldots, 5$ be such that

$$B = KG + \Delta^{(1)}\,, \qquad G = Y^TK^T + V + \Delta^{(2)}, \qquad V = YK^T + \Delta^{(3)},$$
$$B = KV + (KV)^T + \Delta^{(4)}\,, \qquad G = XK^T + \Delta^{(5)}\,.$$

Substituting the last of these expressions into the first one shows that

$$\begin{aligned} B &= K(XK^T + \Delta^{(5)}) + \Delta^{(1)} \\ &= KXK^T + K\Delta^{(5)} + \Delta^{(1)}, \end{aligned}$$

and therefore

$$\Delta = -\Delta^{(1)} - K\Delta^{(5)}\,. \tag{3.1}$$

Substituting the third expression into the fourth one, we see that

$$\begin{aligned} B &= K(YK^T + \Delta^{(3)}) + (K(YK^T + \Delta^{(3)}))^T + \Delta^{(4)} \\ &= KYK^T + KY^TK^T + K\Delta^{(3)} + (K\Delta^{(3)})^T + \Delta^{(4)} \\ &= K(Y + Y^T)K^T + K\Delta^{(3)} + (K\Delta^{(3)})^T + \Delta^{(4)} \\ &= KXK^T + K\Delta^{(3)} + (K\Delta^{(3)})^T + \Delta^{(4)}\,, \end{aligned}$$

and therefore

$$\Delta = -\Delta^{(4)} - K\Delta^{(3)} - (K\Delta^{(3)})^T\,. \tag{3.2}$$

Equations (3.1) and (3.2) provide us with two distinct expressions for Δ that we can use for bounding the magnitudes of that matrix's elements. Now we substitute the third expression into the second one, obtaining

$$\begin{aligned} G &= Y^TK^T + YK^T + \Delta^{(2)} + \Delta^{(3)} \\ &= (Y^T + Y)K^T + \Delta^{(2)} + \Delta^{(3)} \\ &= XK^T + \Delta^{(2)} + \Delta^{(3)}, \end{aligned}$$

and therefore

$$\Delta^{(5)} = \Delta^{(2)} + \Delta^{(3)}\,. \tag{3.3}$$

That formula will help us to bound $|\Delta^{(5)}|$.

Next we consider each of the steps (ST1)–(ST5) and list the bounds on the matrices $|\Delta^{(t)}|$ that we can derive directly by analyzing the computations in each of these steps. In step (ST1) we compute X_{11} by solving the system $K_{11}X_{11}(K_{11})^T = B_{11}$ recursively. Because $K_{11}X_{11}(K_{11})^T = (KXK^T)_{11}$, the errors committed in this computation correspond to the block Δ_{11}. To bound these errors we use the inductive hypothesis, which yields

$$\begin{aligned} |\Delta_{11}| &\le \gamma_{f(c)}|K_{11}||X_{11}||K_{11}|^T \\ &= \gamma_{f(c)}(|K||X||K^T|)_{11} \,. \end{aligned} \tag{3.4}$$

In step (ST2) we compute G_{12} by solving the triangular system $B_{12} = K_{11}G_{12}$. Because $K_{11}G_{12} = (KG)_{12}$, the errors committed in this computation correspond to the block $\Delta_{12}^{(1)}$. A bound on the accuracy of solving triangular systems is given in Lemma 3.5, which yields the bound

$$\begin{aligned} |\Delta^{(1)}|_{12} &\le \gamma_c|K_{11}||G_{12}| \\ &= \gamma_c(|K||G|)_{12} \,. \end{aligned} \tag{3.5}$$

In step (ST3) we compute V_{12} using the formula $V_{12} = G_{12} - 0.5X_{11}(K_{21})^T$. The errors committed in this computation correspond to the block $\Delta_{12}^{(2)}$, and they can be bounded using Lemma 3.3, which guarantees that

$$\begin{aligned} |\Delta^{(2)}|_{12} &\le \gamma_c(0.5|X_{11}||K_{21}|^T + |V_{12}|) \\ &= \gamma_c(|Y_{11}|^T|K_{21}|^T + |V_{12}|) \\ &= \gamma_c(|Y^T||K^T| + |V|)_{12} \,. \end{aligned} \tag{3.6}$$

In step (ST4) we compute X_{12} by forming the matrix $F_4 = V_{12} - 0.5X_{11}(K_{21})^T$ and then solving the triangular system $F_4 = X_{12}(K_{22})^T$. Let the matrices Φ_4 and Ψ_4 be such that

$$V_{12} = 0.5X_{11}(K_{21})^T + F_4 + \Phi_4 \,, \qquad F_4 = X_{12}(K_{22})^T + \Psi_4 \,.$$

Substituting the second of these expressions into the first one yields

$$\begin{aligned} V_{12} &= 0.5X_{11}(K_{21})^T + X_{12}(K_{22})^T + \Phi_4 + \Psi_4 \\ &= (YK^T)_{12} + \Phi_4 + \Psi_4 \,, \end{aligned}$$

and therefore

$$\Delta_{12}^{(3)} = \Phi_4 + \Psi_4 \,. \tag{3.7}$$

Hence, bounding $|\Phi_4|$ and $|\Psi_4|$ allows us to bound $|\Delta_{12}^{(3)}|$. Our bounds on $|\Phi_4|$ and $|\Psi_4|$ follow from Lemmas 3.3 and 3.5, respectively:

$$|\Phi_4| \le \gamma_c(0.5|X_{11}||K_{21}|^T + |F_4|) \,, \qquad |\Psi_4| \le \gamma_{m-c}|X_{12}||K_{22}|^T \,.$$

Applying these bounds to (3.7), substituting $F_4 = X_{12}(K_{22})^T + \Psi_4$, and bounding $|\Psi_4|$ again yields

$$\begin{aligned} |\Delta^{(3)}|_{12} &\le \gamma_c(0.5|X_{11}||K_{21}|^T + |F_4|) + \gamma_{m-c}|X_{12}||K_{22}|^T \\ &\le \gamma_c\, 0.5|X_{11}||K_{21}|^T + (\gamma_c + \gamma_{m-c} + \gamma_c\gamma_{m-c})|X_{12}||K_{22}|^T \,, \end{aligned}$$

and because $\gamma_c + \gamma_{m-c} + \gamma_c\gamma_{m-c} \leq \gamma_m$ by Lemma 3.1, we obtain

$$\begin{aligned} |\Delta^{(3)}|_{12} &\leq \gamma_c\, 0.5|X_{11}||K_{21}|^T + \gamma_m|X_{12}||K_{22}|^T \\ &\leq \gamma_m\, 0.5|X_{11}||K_{21}|^T + \gamma_m|X_{12}||K_{22}|^T \\ &= \gamma_m(|Y||K^T|)_{12}\ . \end{aligned} \tag{3.8}$$

Finally, in step (ST5) we compute X_{22} by forming the matrix $F_5 = B_{22} - K_{21}V_{12} - (K_{21}V_{12})^T$ and then recursively solving the system $F_5 = K_{22}X_{22}(K_{22})^T$. Let the matrices Φ_5 and Ψ_5 be such that

$$B_{22} = K_{21}V_{12} + (K_{21}V_{12})^T + F_5 + \Phi_5\ , \qquad F_5 = K_{22}X_{22}(K_{22})^T + \Psi_5\ .$$

Substituting the second of these expressions into the first one, we obtain

$$B_{22} = K_{21}V_{12} + (K_{21}V_{12})^T + K_{22}X_{22}(K_{22})^T + \Phi_5 + \Psi_5\ ,$$

and because

$$\begin{aligned} K_{22}X_{22}(K_{22})^T &= K_{22}(Y_{22} + (Y_{22})^T)(K_{22})^T \\ &= K_{22}Y_{22}(K_{22})^T + (K_{22}Y_{22}(K_{22})^T)^T \\ &= K_{22}V_{22} + (K_{22}V_{22})^T \end{aligned}$$

we may conclude that

$$\begin{aligned} B_{22} &= K_{21}V_{12} + (K_{21}V_{12})^T + K_{22}V_{22} + (K_{22}V_{22})^T + \Phi_5 + \Psi_5 \\ &= (KV + (KV)^T)_{22} + \Phi_5 + \Psi_5 \end{aligned}$$

and therefore

$$\Delta_{22}^{(4)} = \Phi_5 + \Psi_5\ . \tag{3.9}$$

This shows that bounding $|\Phi_5|$ and $|\Psi_5|$ allows us to bound $|\Delta_{22}^{(4)}|$. Our bounds on $|\Phi_5|$ and $|\Psi_5|$ follow from Lemma 3.4 and from the inductive hypothesis, respectively:

$$|\Phi_5| \leq \gamma_{2c}(|K_{21}||V_{12}| + (|K_{21}||V_{12}|)^T + |F_5|)\ , \quad |\Psi_5| \leq \gamma_{f(m-c)}|K_{22}||X_{22}||K_{22}|^T\ .$$

Applying these bounds to (3.9), substituting $F_5 = K_{22}X_{22}(K_{22})^T + \Psi_5$, and bounding $|\Psi_5|$ again yields

$$\begin{aligned} |\Delta^{(4)}|_{22} &\leq \gamma_{2c}(|K_{21}||V_{12}| + (|K_{21}||V_{12}|)^T + |F_5|) \\ &\quad + \gamma_{f(m-c)}|K_{22}||X_{22}||K_{22}|^T \\ &\leq \gamma_{2c}(|K_{21}||V_{12}| + (|K_{21}||V_{12}|)^T) \\ &\quad + (\gamma_{2c} + \gamma_{f(m-c)} + \gamma_{2c}\gamma_{f(m-c)})|K_{22}||X_{22}||K_{22}|^T \\ &\leq \gamma_{2c}(|K_{21}||V_{12}| + (|K_{21}||V_{12}|)^T) + \gamma_{f(m-c)+2c}|K_{22}||X_{22}||K_{22}|^T\ . \end{aligned} \tag{3.10}$$

Next we bound $|\Delta^{(5)}|$. Applying (3.6) to (3.3) and then substituting $V = YK^T + \Delta^{(3)}$, we obtain

$$\begin{aligned} |\Delta^{(5)}|_{12} &\leq \gamma_c(|Y^T||K^T| + |V|)_{12} + |\Delta_{12}^{(3)}| \\ &\leq \gamma_c(|Y^T||K^T| + |Y||K^T| + |\Delta^{(3)}|)_{12} + |\Delta_{12}^{(3)}| \\ &= \gamma_c((|Y^T| + |Y|)|K^T|)_{12} + (1+\gamma_c)|\Delta_{12}^{(3)}| \\ &= \gamma_c(|X||K^T|)_{12} + (1+\gamma_c)|\Delta_{12}^{(3)}|\ . \end{aligned}$$

Bounding $|\Delta_{12}^{(3)}|$ in that expression using (3.8) yields

$$\begin{aligned}|\Delta^{(5)}|_{12} &\le \gamma_c(|X||K^T|)_{12} + (1+\gamma_c)\gamma_m(|Y||K^T|)_{12} \\ &\le \gamma_c(|X||K^T|)_{12} + (1+\gamma_c)\gamma_m((|Y^T|+|Y|)|K^T|)_{12} \\ &= \gamma_c(|X||K^T|)_{12} + (1+\gamma_c)\gamma_m(|X||K^T|)_{12} \\ &\le (\gamma_c + \gamma_m + \gamma_c\gamma_m)(|X||K^T|)_{12} \\ &\le \gamma_{m+c}(|X||K^T|)_{12} \ . \end{aligned} \tag{3.11}$$

Next we bound $|\Delta_{12}|$. Applying (3.5) to (3.1) we obtain

$$\begin{aligned}|\Delta_{12}| &\le |\Delta^{(1)}|_{12} + (|K||\Delta^{(5)}|)_{12} \\ &\le \gamma_c(|K||G|)_{12} + (|K||\Delta^{(5)}|)_{12} \ . \end{aligned}$$

Further substituting $G = XK^T + \Delta^{(5)}$ yields

$$\begin{aligned}|\Delta_{12}| &\le \gamma_c(|K||X||K^T| + |K||\Delta^{(5)}|)_{12} + (|K||\Delta^{(5)}|)_{12} \\ &= \gamma_c(|K||X||K^T|)_{12} + (1+\gamma_c)(|K||\Delta^{(5)}|)_{12} \ . \end{aligned}$$

Bounding the block of $|\Delta^{(5)}|$ that appears in that expression using (3.11) yields

$$\begin{aligned}|\Delta_{12}| &\le \gamma_c(|K||X||K^T|)_{12} + (1+\gamma_c)\gamma_{m+c}(|K||X||K^T|)_{12} \\ &= (\gamma_c + \gamma_{m+c} + \gamma_c\gamma_{m+c})(|K||X||K^T|)_{12} \ , \end{aligned}$$

and because $\gamma_c + \gamma_{m+c} + \gamma_c\gamma_{m+c} \le \gamma_{m+2c}$, our bound on $|\Delta_{12}|$ is

$$|\Delta_{12}| \le \gamma_{m+2c}(|K||X||K^T|)_{12} \ . \tag{3.12}$$

Next we bound $|\Delta_{22}|$. We use the formula (3.2), starting with the terms $K\Delta^{(3)}$ and $(K\Delta^{(3)})^T$ in that formula. Because the diagonal and off-diagonal blocks of V are not computed by the algorithm, we defined them such that $V_{I,J} = (YK^T)_{I,J}$, and therefore $\Delta_{I,J}^{(3)} = 0$ for all $I \ge J$. Therefore

$$(K\Delta^{(3)} + (K\Delta^{(3)})^T)_{22} = K_{21}\Delta_{12}^{(3)} + (K_{21}\Delta_{12}^{(3)})^T \ .$$

Applying the bound (3.8) on $|\Delta^{(3)}|$ to that expression yields

$$|K_{21}||\Delta_{12}^{(3)}| + (|K_{21}||\Delta_{12}^{(3)}|)^T \le \gamma_m(|K_{21}|(|Y||K^T|)_{12} + (|K_{21}|(|Y||K^T|)_{12})^T) \ .$$

The expression on the right-hand side can be transformed into a more informative form:

$$\begin{aligned}&|K_{21}|(|Y||K^T|)_{12} + (|K_{21}|(|Y||K^T|)_{12})^T \\ &\quad= |K_{21}||Y_{11}||K_{21}|^T + |K_{21}||Y_{12}||K_{22}|^T \\ &\qquad + |K_{21}||Y_{11}|^T|K_{21}|^T + |K_{22}||Y_{12}|^T|K_{21}|^T \\ &\quad= |K_{21}|(|Y_{11}| + |Y_{11}|^T)|K_{21}|^T + |K_{21}||Y_{12}||K_{22}|^T + |K_{22}||Y_{12}|^T|K_{21}|^T \\ &\quad= |K_{21}||X_{11}||K_{21}|^T + |K_{21}||Y_{12}||K_{22}|^T + |K_{22}||Y_{12}|^T|K_{21}|^T \\ &\quad= |K_{21}||X_{11}||K_{21}|^T + |K_{21}||X_{12}||K_{22}|^T + |K_{22}||X_{21}||K_{21}|^T \\ &\quad= (|K||X||K^T|)_{22} - |K_{22}||X_{22}||K_{22}|^T \ , \end{aligned}$$

and therefore

$$|K\Delta^{(3)} + (K\Delta^{(3)})^T|_{22} \leq \gamma_m((|K||X||K^T|)_{22} - |K_{22}||X_{22}||K_{22}|^T) \,. \tag{3.13}$$

Next we bound $|\Delta^{(4)}|$ using (3.10). The expression $|K_{21}||V_{12}| + (|K_{21}||V_{12}|)^T$ in (3.10) can be bounded by substituting $V = YK^T + \Delta^{(3)}$ and proceeding along the same lines as in our derivation of (3.13). This produces the bound

$$|K_{21}||V_{12}| + (|K_{21}||V_{12}|)^T \leq (1+\gamma_m)((|K||X||K^T|)_{22} - |K_{22}||X_{22}||K_{22}|^T) \,. \tag{3.14}$$

Now we substitute (3.14) into (3.10) and then substitute the result together with (3.13) into (3.2), yielding

$$\begin{aligned}
|\Delta_{22}| &\leq \gamma_{2c}(1+\gamma_m)((|K||X||K^T|)_{22} - |K_{22}||X_{22}||K_{22}|^T) \\
&\quad + \gamma_{f(m-c)+2c}|K_{22}||X_{22}||K_{22}|^T \\
&\quad + \gamma_m((|K||X||K^T|)_{22} - |K_{22}||X_{22}||K_{22}|^T) \\
&= (\gamma_{2c} + \gamma_m + \gamma_{2c}\gamma_m)((|K||X||K^T|)_{22} - |K_{22}||X_{22}||K_{22}|^T) \\
&\quad + \gamma_{f(m-c)+2c}|K_{22}||X_{22}||K_{22}|^T \\
&\leq \gamma_{m+2c}((|K||X||K^T|)_{22} - |K_{22}||X_{22}||K_{22}|^T) \\
&\quad + \gamma_{f(m-c)+2c}|K_{22}||X_{22}||K_{22}|^T \\
&\leq \gamma_{\max\{m,f(m-c)\}+2c}((|K||X||K^T|)_{22} - |K_{22}||X_{22}||K_{22}|^T) \\
&\quad + \gamma_{\max\{m,f(m-c)\}+2c}|K_{22}||X_{22}||K_{22}|^T \\
&= \gamma_{\max\{m,f(m-c)\}+2c}(|K||X||K^T|)_{22} \,.
\end{aligned} \tag{3.15}$$

Gathering together the bounds (3.4), (3.12), and (3.15), we see that

$$|\Delta_{I,J}| \leq \gamma_{C(I,J)}(|K||X||K^T|)_{I,J}$$

for $1 \leq I, J \leq 2$, where

$$C = \begin{bmatrix} f(c) & m+2c \\ m+2c & \max\{m, f(m-c)\} + 2c \end{bmatrix} \,.$$

We bound the elements of C as follows:

$$\begin{aligned}
f(c) &= 3c - 1 \leq 3(m-1) - 1 = 3m - 4 \,, \\
m + 2c &\leq m + 2(m-1) = 3m - 2 \,, \\
\max\{m, f(m-c)\} + 2c &= \max\{m + 2c, f(m-c) + 2c\} \\
&\leq \max\{3m - 2, f(m-c) + 2c\} \\
&= \max\{3m - 2, 3(m-c) - 1 + 2c\} \\
&= \max\{3m - 2, 3m - c - 1\} \\
&\leq \max\{3m - 2, 3m - 2\} \\
&= 3m - 2 \,.
\end{aligned}$$

Therefore $C(I,J) \leq 3m - 2$ for $1 \leq I, J \leq 2$, and thus the inductive hypothesis is true for systems of order m. Although we showed that $C(I,J) \leq 3m-2$, the larger constant γ_{3m-1} in the statement of the lemma is required to account for the case $m = 1$. ∎

3.3. The stability of the overall algorithm. We now show that the block Aasen algorithm is backward stable. We use the symbols L, T, H, and W to denote the corresponding floating-point matrices and not their abstract exact equivalents. The exception to this is the diagonal blocks of W, which are not computed by the algorithm and which we define for convenience as being exactly

$$W_{J,J} = (RL^T)_{J,J} = R_{J,J}L_{J,J}^T \ .$$

Similarly, R is also not computed by the algorithm due to the optimization described in section 2.3. We define R as the block upper-triangular matrix with symmetric diagonal blocks that satisfies $R^T + R = T$. Its superdiagonal blocks are exactly those of the computed T and its diagonal blocks are obtained from those of the computed T by scaling them by 0.5.

LEMMA 3.8. *The computed factors satisfy* $A = LTL^T + \Delta$, *where*

$$|\Delta_{I,J}| \leq \gamma_{n+2b}(|L||T||L^T|)_{I,J}$$

whenever $I \neq J$.

Proof. Let the matrices $\Delta^{(1)}$ and $\Delta^{(2)}$ be such that

$$A = LH + \Delta^{(1)} \ , \qquad H = TL^T + \Delta^{(2)} \ .$$

Substituting the second expression into the first one yields

$$A = LTL^T + L\Delta^{(2)} + \Delta^{(1)} \ ,$$

and thus

$$\Delta = \Delta^{(1)} + L\Delta^{(2)} \ . \tag{3.16}$$

Bounding $|\Delta|$ requires that we obtain bounds on $|\Delta^{(1)}|$ and $|\Delta^{(2)}|$.

Let us bound the subdiagonal blocks of $|\Delta^{(1)}|$ by considering the computation that corresponds to (AA4). In that equation we form the matrix $F = A_{J+1:N,J} - L_{J+1:N,1:J}H_{1:J,J}$ and then compute its LU factorization $F = L_{J+1:N,J+1}H_{J+1,J}$. Let Φ and Ψ be such that

$$A_{J+1:N,J} = L_{J+1:N,1:J}H_{1:J,J} + F + \Phi \ , \tag{3.17}$$

$$F = L_{J+1:N,J+1}H_{J+1,J} + \Psi \ . \tag{3.18}$$

Substituting the second expression into the first one yields

$$\begin{aligned} A_{J+1:N,J} &= L_{J+1:N,1:J}H_{1:J,J} + L_{J+1:N,J+1}H_{J+1,J} + \Phi + \Psi \\ &= L_{J+1:N,1:J+1}H_{1:J+1,J} + \Phi + \Psi \ , \end{aligned}$$

because $H_{J+2:N,J}$ is zero,

$$A_{J+1:N,J} = (LH)_{J+1:N,J} + \Phi + \Psi \ ,$$

and therefore

$$\Delta^{(1)}_{J+1:N,J} = \Phi + \Psi \ . \tag{3.19}$$

We analyze the accuracy of forming F using Lemma 3.3, which yields the bound

$$|\Phi| \leq \gamma_{Jb}(|L_{J+1:N,1:J}||H_{1:J,J}| + |F|) \ .$$

However, because $L_{2:N,1}$ is zero, the inner dimension of the product $L_{J+1:N,1:J}H_{1:J,J}$ is effectively $(J-1)b$ instead of Jb, and therefore

$$|\Phi| \leq \gamma_{(J-1)b}(|L_{J+1:N,1:J}||H_{1:J,J}| + |F|) \,. \tag{3.20}$$

The accuracy of the LU factorization of F can be analyzed using Lemma 3.6, which yields

$$|\Psi| \leq \gamma_b |L_{J+1:N,J+1}||H_{J+1,J}| \,. \tag{3.21}$$

Substituting (3.20) and (3.21) into (3.19) yields

$$|\Delta^{(1)}_{J+1:N,J}| \leq \gamma_{(J-1)b}(|L_{J+1:N,1:J}||H_{1:J,J}| + |F|) + \gamma_b |L_{J+1:N,J+1}||H_{J+1,J}| \,,$$

and further substituting (3.18) and using (3.21) again yields

$$\begin{aligned} |\Delta^{(1)}_{J+1:N,J}| \leq{}& \gamma_{(J-1)b}|L_{J+1:N,1:J}||H_{1:J,J}| \\ &+ (\gamma_{(J-1)b} + \gamma_b + \gamma_{(J-1)b}\gamma_b)|L_{J+1:N,J+1}||H_{J+1,J}| \,. \end{aligned}$$

Bounding the constants in this expression according to

$$\begin{aligned} \gamma_{(J-1)b} &\leq \gamma_{Jb} \,, \\ \gamma_{(J-1)b} + \gamma_b + \gamma_{(J-1)b}\gamma_b &\leq \gamma_{Jb} \,, \end{aligned}$$

where the second bound is justified by Lemma 3.1, yields

$$\begin{aligned} |\Delta^{(1)}_{J+1:N,J}| &\leq \gamma_{Jb}|L_{J+1:N,1:J}||H_{1:J,J}| + \gamma_{Jb}|L_{J+1:N,J+1}||H_{J+1,J}| \\ &= \gamma_{Jb}(|L||H|)_{J+1:N,J} \,, \end{aligned}$$

and therefore

$$|\Delta^{(1)}_{I,J}| \leq \gamma_{Jb}(|L||H|)_{I,J} \tag{3.22}$$

for all $I > J$.

We bound the diagonal and superdiagonal blocks of $|\Delta^{(2)}|$ by considering the computation that corresponds to (AA3). In that equation we compute blocks of H by forming the corresponding blocks of TL^T, and as we discuss in section 2.3, these blocks are formed according to the formula

$$H_{I,J} = T_{I,I-1}(L_{J,I-1})^T + T_{I,I}(L_{J,I})^T + T_{I,I+1}(L_{J,I+1})^T \,.$$

This is equivalent to multiplying the b-by-$3b$ matrix $T_{I,I-1:I+1}$ by the $3b$-by-b matrix $(L_{J,I-1:I+1})^T$, and the accuracy of this computation is bounded in Lemma 3.2, which guarantees that

$$|\Delta^{(2)}_{I,J}| \leq \gamma_{3b}(|T||L^T|)_{I,J}$$

for all $I \leq J$. The blocks $\Delta^{(2)}_{J+1,J}$ correspond to (AA5), which solves the triangular system $H_{J+1,J} = T_{J+1,J}(L_{J,J})^T$. This is analyzed in Lemma 3.5, which guarantees that

$$|\Delta^{(2)}_{J+1,J}| \leq \gamma_b(|T||L^T|)_{J+1,J} \,.$$

All other blocks of $\Delta^{(2)}$ are zero, and thus

$$|\Delta^{(2)}| \leq \gamma_{3b}|T||L^T| \,. \tag{3.23}$$

Substituting (3.22) and (3.23) into (3.16) yields

$$|\Delta_{I,J}| \leq \gamma_{Jb}(|L||H|)_{I,J} + \gamma_{3b}(|L||T||L^T|)_{I,J}$$

for all $I > J$. Further substituting $H = TL^T + \Delta^{(2)}$ and using (3.23) once again yields

$$\begin{aligned} |\Delta_{I,J}| &\leq (\gamma_{Jb} + \gamma_{3b} + \gamma_{Jb}\gamma_{3b})(|L||T||L^T|)_{I,J} \\ &\leq \gamma_{Jb+3b}(|L||T||L^T|)_{I,J} \,. \end{aligned}$$

The constant γ_{Jb+3b} is maximized when $J = N - 1$, which yields the required bound for all $I > J$. As for $I < J$, the same bound holds because Δ is the difference of the two symmetric matrices A and LTL^T and is thus itself symmetric. ∎

LEMMA 3.9. *The computed factors satisfy $A = LTL^T + \Delta$, where*

$$|\Delta_{J,J}| \leq \gamma_{2n-b-1}(|L||T||L^T|)_{J,J}$$

for $J = 1, 2, \ldots, N$.

Proof. Let the matrices $\Delta^{(1)}$ and $\Delta^{(2)}$ be such that

$$A = LW + (LW)^T + \Delta^{(1)} \,, \qquad W = RL^T + \Delta^{(2)} \,.$$

Substituting the second expression into the first one yields

$$\begin{aligned} A &= LRL^T + (LRL^T)^T + L\Delta^{(2)} + (L\Delta^{(2)})^T + \Delta^{(1)} \\ &= L(R + R^T)L^T + L\Delta^{(2)} + (L\Delta^{(2)})^T + \Delta^{(1)} \\ &= LTL^T + L\Delta^{(2)} + (L\Delta^{(2)})^T + \Delta^{(1)} \end{aligned}$$

and therefore

$$\Delta = \Delta^{(1)} + L\Delta^{(2)} + (L\Delta^{(2)})^T \,. \tag{3.24}$$

Equation (AA2) computes $T_{J,J}$ by forming $F = A_{J,J} - L_{J,1:J-1}W_{1:J-1,J} - (L_{J,1:J-1}W_{1:J-1,J})^T$ and then solving $L_{J,J}T_{J,J}(L_{J,J})^T = F$. Let the matrices Φ and Ψ be such that

$$A_{J,J} = L_{J,1:J-1}W_{1:J-1,J} + (L_{J,1:J-1}W_{1:J-1,J})^T + F + \Phi \,, \tag{3.25}$$

$$F = L_{J,J}T_{J,J}(L_{J,J})^T + \Psi \,. \tag{3.26}$$

Substituting (3.26) into (3.25) yields

$$A_{J,J} = L_{J,1:J-1}W_{1:J-1,J} + (L_{J,1:J-1}W_{1:J-1,J})^T + L_{J,J}T_{J,J}(L_{J,J})^T + \Phi + \Psi \,.$$

Rewriting the term $L_{J,J}T_{J,J}(L_{J,J})^T$ according to

$$\begin{aligned} L_{J,J}T_{J,J}(L_{J,J})^T &= L_{J,J}(R_{J,J} + (R_{J,J})^T)(L_{J,J})^T \\ &= L_{J,J}R_{J,J}(L_{J,J})^T + (L_{J,J}R_{J,J}(L_{J,J})^T)^T \\ &= L_{J,J}W_{J,J} + (L_{J,J}W_{J,J})^T \end{aligned}$$

gives

$$\begin{aligned} A_{J,J} &= L_{J,1:J-1}W_{1:J-1,J} + (L_{J,1:J-1}W_{1:J-1,J})^T + L_{J,J}W_{J,J} + (L_{J,J}W_{J,J})^T + \Phi + \Psi \\ &= L_{J,1:J}W_{1:J,J} + (L_{J,1:J}W_{1:J,J})^T + \Phi + \Psi \\ &= (LW + (LW)^T)_{J,J} + \Phi + \Psi \end{aligned}$$

and thus

$$\Delta^{(1)}_{J,J} = \Phi + \Psi \ . \tag{3.27}$$

The accuracy of forming F and then solving for $T_{J,J}$ can be bounded using Lemmas 3.4 and 3.7, which guarantee that

$$\begin{aligned} |\Phi| &\le \gamma_{2(J-2)b}(|L_{J,1:J-1}||W_{1:J-1,J}| + (|L_{J,1:J-1}||W_{1:J-1,J}|)^T + |F|) \ , \\ |\Psi| &\le \gamma_{3b-1}|L_{J,J}||T_{J,J}||L_{J,J}|^T \ . \end{aligned}$$

Substituting these bounds into (3.27), further substituting (3.26), and bounding again yields

$$\begin{aligned} |\Delta^{(1)}_{J,J}| &\le \gamma_{2(J-2)b}(|L_{J,1:J-1}||W_{1:J-1,J}| + (|L_{J,1:J-1}||W_{1:J-1,J}|)^T) \\ &\quad + (\gamma_{2(J-2)b} + \gamma_{3b-1} + \gamma_{2(J-2)b}\gamma_{3b-1})|L_{J,J}||T_{J,J}||L_{J,J}|^T \\ &\le \gamma_{2(J-2)b}(|L_{J,1:J-1}||W_{1:J-1,J}| + (|L_{J,1:J-1}||W_{1:J-1,J}|)^T) \\ &\quad + \gamma_{2(J-2)b+3b-1}|L_{J,J}||T_{J,J}||L_{J,J}|^T \ , \end{aligned} \tag{3.28}$$

which is the bound we require for $|\Delta^{(1)}|$.

The superdiagonal blocks of $\Delta^{(2)}$ correspond to (AA1). That equation states that blocks of W are computed by forming the corresponding blocks of the product RL^T, which are formed according to the formula

$$W_{I,J} = 0.5(T_{I,I}(L_{J,I})^T) + T_{I,I+1}(L_{J,I+1})^T \ ,$$

as we explain in section 2.3. Because of the scaling by 0.5 we cannot apply Lemma 3.2 directly to this formula. Instead we must bound the errors resulting from forming the two single-block products separately, and then use assumptions (FL2) and (FL1) to account for the effects of scaling and summation, respectively. We skip the details; the resulting bound is

$$|\Delta^{(2)}_{I,J}| \le \gamma_{b+1}(|R||L^T|)_{I,J} \tag{3.29}$$

for all $I < J$.

Next we return to bounding (3.24), starting with the last two terms. The diagonal and subdiagonal blocks of W are defined such that the corresponding blocks of $\Delta^{(2)}$ are zero, and therefore

$$(|L||\Delta^{(2)}| + (|L||\Delta^{(2)}|))_{J,J} = |L_{J,1:J-1}||\Delta^{(2)}_{1:J-1,J}| + (|L_{J,1:J-1}||\Delta^{(2)}_{1:J-1,J}|)^T \ .$$

Substituting (3.29) yields

$$\begin{aligned} (|L||\Delta^{(2)}| + (|L||\Delta^{(2)}|))_{J,J} \le \gamma_{b+1}\big(&|L_{J,1:J-1}||R_{1:J-1,1:J}||L_{J,1:J}|^T \\ &+ (|L_{J,1:J-1}||R_{1:J-1,1:J}||L_{J,1:J}|^T)^T\big) \ , \end{aligned}$$

which can be further simplified according to

$$\begin{aligned}
&|L_{J,1:J-1}||R_{1:J-1,1:J}||L_{J,1:J}|^T + (|L_{J,1:J-1}||R_{1:J-1,1:J}||L_{J,1:J}|^T)^T \\
&\quad = \sum_{I=1}^{J-1} (2\,|L_{J,I}||R_{I,I}||L_{J,I}|^T + |L_{J,I}||R_{I,I+1}||L_{J,I+1}|^T + |L_{J,I+1}||R_{I,I+1}|^T|L_{J,I}|^T) \\
&\quad = \sum_{I=1}^{J-1} (|L_{J,I}||T_{I,I}||L_{J,I}|^T + |L_{J,I}||T_{I,I+1}||L_{J,I+1}|^T + |L_{J,I+1}||T_{I+1,I}||L_{J,I}|^T) \\
&\quad = (|L||T||L|^T)_{J,J} - |L_{J,J}||T_{J,J}||L_{J,J}|^T \,,
\end{aligned}$$

yielding

$$(|L||\Delta^{(2)}| + (|L||\Delta^{(2)}|)^T)_{J,J} \leq \gamma_{b+1}((|L||T||L^T|)_{J,J} - |L_{J,J}||T_{J,J}||L_{J,J}|^T) \,. \tag{3.30}$$

To bound the first term in (3.28) we substitute $W = RL^T + \Delta^{(2)}$ and apply the same arguments we used to produce (3.30), obtaining

$$\begin{aligned}
|L_{J,1:J-1}||W_{1:J-1,J}| + (|L_{J,1:J-1}||W_{1:J-1,J}|)^T \leq (1+\gamma_{b+1})((|L||T||L^T|)_{J,J} \\
- |L_{J,J}||T_{J,J}||L_{J,J}|^T) \,.
\end{aligned} \tag{3.31}$$

Finally, substituting (3.31) into (3.28) and then substituting the result together with (3.30) into (3.24) yields

$$\begin{aligned}
|\Delta_{J,J}| \leq (\gamma_{2(J-2)b} + \gamma_{b+1} + \gamma_{2(J-2)b}\gamma_{b+1})((|L||T||L^T|)_{J,J} - |L_{J,J}||T_{J,J}||L_{J,J}|^T) \\
+ \gamma_{2(J-2)b+3b-1}|L_{J,J}||T_{J,J}||L_{J,J}|^T \,.
\end{aligned}$$

Because $b \geq 1$, we can bound

$$\gamma_{2(J-2)b} + \gamma_{b+1} + \gamma_{2(J-2)b}\gamma_{b+1} \leq \gamma_{2(J-2)b+b+1} \leq \gamma_{2(J-2)b+3b-1} \,,$$

which allows us to cancel the two instances of $|L_{J,J}||T_{J,J}||L_{J,J}|^T$ and obtain

$$|\Delta_{J,J}| \leq \gamma_{2(J-2)b+3b-1}(|L||T||L^T|)_{J,J} \,.$$

The constant $\gamma_{2(J-2)b+3b-1}$ is maximized when $J = N$, which yields the required bound. ∎

THEOREM 3.10. *The computed factors satisfy $A = LTL^T + \Delta$, where*

$$|\Delta| \leq \gamma_{2n-b-1}|L||T||L^T|$$

if $n > 3b$ and

$$|\Delta| \leq \gamma_{n+2b}|L||T||L^T|$$

otherwise.

Proof. Lemmas 3.8 and 3.9 state that

$$|\Delta_{I,J}| \leq \gamma_{n+2b}(|L||T||L^T|)_{I,J}$$

whenever $I \neq J$, and

$$|\Delta_{I,J}| \leq \gamma_{2n-b-1}(|L||T||L^T|)_{I,J}$$

whenever $I = J$, and therefore the bound

$$|\Delta_{I,J}| \leq \max\{\gamma_{n+2b}, \gamma_{2n-b-1}\}\,(|L||T||L^T|)_{I,J}$$

holds for all I and J. The quantity γ_n increases monotonically with n (so long as $nu < 1$) and therefore $\gamma_{2n-b-1} \geq \gamma_{n+2b}$ whenever $2n - b - 1 \geq n + 2b$, which occurs whenever $n > 3b$. ∎

3.4. The stability of solving linear systems. Next we analyze the stability of using our factorization for solving systems of linear equations. In our analysis we assume that banded systems that involve the matrix T are solved using a specific algorithm, the LU factorization with partial pivoting. Using the other solvers that we mention in section 2.4 is also possible, but banded LU with partial pivoting is more suitable because it is implemented in LAPACK and has a straightforward analysis.

Let $Ax = f$ be the system that we are solving, and let $PAP^T = LTL^T$ be the factorization that we compute using our algorithm. We remind the reader that b denotes the block size of the factorization and the bandwidth of T. We compute x by using the following steps:

1. Solve the triangular system $Lu = Pf$.
2. Solve the banded system $Tv = u$.
3. Solve the triangular system $L^T w = v$.
4. Set $x = P^T w$.

The next lemmas can be proved by following the analyses of triangular solve and of the LU factorization, respectively, in [18, sections 8.1 and 9.3], while taking advantage of the special structure of the matrices to obtain the reduced constants that figure in the bounds.

LEMMA 3.11. *Let T be a unit lower-triangular matrix, and assume that it is equal to the identity matrix in the first b columns. If the system $Tu = v$ is solved in floating-point arithmetic using substitution, then*

$$(T + \Delta)\, u = v\ , \qquad |\Delta| \le \gamma_{n-b-1}\, |T|\ .$$

Alternatively, if T is a unit upper-triangular matrix that equals to the identity matrix in the first b rows, then the same bound holds.

LEMMA 3.12. *Let $Au = v$ be a banded linear system, and let b be its bandwidth. If the LU factorization with partial pivoting $PA = LU$ is used to solve the system in floating-point arithmetic, then*

$$\left(A + P^T \Delta\right) u = v\ , \qquad |\Delta| \le \gamma_{n+4b+1}\, |L|\, |U|\ .$$

Our bound is given in the following theorem.

THEOREM 3.13. *If the system $Ax = f$ is solved in floating-point arithmetic by using the factorization $PAP^T = LTL^T$, computed using our algorithm, and the LU factorization with partial pivoting $P_1 T = L_1 U_1$, then*

$$\left(A + P^T \Delta P\right) x = f\ ,$$

where

$$|\Delta| \le \gamma_{g(n,b)}\, |L|\, |T|\, |L|^T + \gamma_{h(n,b)}\, |L|\, P_1^T\, |L_1|\, |U_1|\, |L|^T + O(u^2)$$

for

$$g(n,b) = \begin{cases} 4n - 3b - 3 & \text{if } n > 3b\ , \\ 3n - 2 & \text{otherwise} \end{cases} \qquad \text{and} \qquad h(n,b) = n + 4b + 1\ .$$

Proof. Let the matrices Δ_t for $t = 1, 2, 3$ and Γ_1 and Γ_2 be such that

$$(L + \Delta_1)\, u = Pf\ , \qquad (T + \Delta_2)\, v = u\ , \qquad \left(L^T + \Delta_3\right) w = v$$

and

$$PAP^T + \Gamma_1 = LTL^T , \qquad (L + \Delta_1)(T + \Delta_2)\left(L^T + \Delta_3\right) = LTL^T + \Gamma_2 .$$

Combining the equations that correspond to the Δ_t matrices yields

$$(L + \Delta_1)(T + \Delta_2)\left(L^T + \Delta_3\right) w = Pf ,$$

and applying to this the equations that correspond to Γ_1 and Γ_2 yields

$$\left(PAP^T + \Gamma_1 + \Gamma_2\right) w = Pf .$$

Substituting $w = Px$ and premultiplying both sides of this equation by P^T yields

$$\left(A + P^T(\Gamma_1 + \Gamma_2)P\right) x = f ,$$

and therefore

$$\Delta = \Gamma_1 + \Gamma_2 . \tag{3.32}$$

Next we expand the left-hand side in the equation that defines Γ_2 and obtain

$$\begin{aligned} \Gamma_2 = {} & \Delta_1 T L^T + L T \Delta_3 + \Delta_1 T \Delta_3 \\ & + L \Delta_2 L^T + \Delta_1 \Delta_2 L^T + L \Delta_2 \Delta_3 + \Delta_1 \Delta_2 \Delta_3 . \end{aligned} \tag{3.33}$$

Our bounds on the Δ_t matrices, in absolute value, follow from Lemmas 3.11 and 3.12, which guarantee that

$$|\Delta_1| \le \gamma_{n-b-1} |L| \qquad |\Delta_2| \le \gamma_{n+4b+1} P_1^T |L_1| |U_1| \qquad |\Delta_3| \le \gamma_{n-b-1} |L|^T .$$

Substituting these bounds into (3.33) yields

$$|\Gamma_2| \le \gamma_{2n-2b-2} |L| |T| |L|^T + \gamma_{n+4b+1} (1 + \gamma_{2n-2b-2}) |L| P_1^T |L_1| |U_1| |L|^T .$$

Our bound on $|\Gamma_1|$ is given by Theorem 3.10, and substituting this bound together with the one on $|\Gamma_2|$ into (3.32) yields the required bound on $|\Delta|$. ∎

The theorem states that the norms of $|L| |T| |L|^T$ and $|L| P_1^T |L_1| |U_1| |L|^T$ determine the stability of the solution. The goal of pivoting is to make sure that these norms are as close as possible to $\|A\|$.

3.5. Growth. Stability depends on the magnitude of the factors relative to that of A. How large can the factors get? In the elementwise Aasen algorithm, the magnitude of elements of L is bounded by 1 (because the algorithm uses partial pivoting), and it is easy to show that $|T_{ij}| \le 4^{n-2} \max_{ij} |A_{ij}|$ [17]; the argument is essentially the same as the one that establishes the bound on $|U_{ij}|$ in Gaussian elimination with partial pivoting (GEPP). This bound is higher than the corresponding bounds of $2^{n-1} \max_{ij} |A_{ij}|$ for Gaussian elimination with partial pivoting and of $(2.57)^{n-1} \max_{ij} |A_{ij}|$ for the Bunch–Kaufman algorithm [9]. Furthermore, the bound is attained by a known matrix of order $n = 3$, although larger matrices that attain the bound are not known [18, p. 224].

It is important to interpret this bound correctly. The actual expression (4^{n-2}) is not important, because it does not indicate the growth that is normally attained. The same is true for LU with partial pivoting; it is stable *in spite* of the fact that the

growth factor bound is as large as 2^{n-1}, not because this bound is small (it is not; it is huge). Two other things are important. One is that the bound shows that growth is not related to the condition number of A. The second is that growth in practice is small. The reasons for this are complex and not completely understood even in LU with partial pivoting, but this is the reality; for deeper analyses and discussion, see [30, 34] and [18, section 9.4].

If we compute the factorization in (AA4) using LU with partial pivoting (GEPP), essentially the same bounds hold for our block algorithm. The block columns of the L factor are generated by GEPP, so the same two-sided doubling-up argument shows that the growth factor for T is bounded by 4^{n-b-1} (since the first columns of L are unit vectors and additions/subtractions start only in column $b+1$).

When the factorization in (AA4) is computed in a communication-avoiding way using the tall-and-skinny LU algorithm [14] (TSLU), L is still bounded, but the bound is 2^{bh}, where h is a parameter of TSLU that normally satisfies $h = O(\log n)$. This can obviously be much larger than 1, although experiments indicate that L is usually much smaller. This implies that growth in T is still bounded, but the bound is now 4^{nbh}. This is worse than with GEPP, but as we explained above, this theoretical bound is not what normally governs the stability of the algorithm. The recently developed panel rank-revealing LU factorization [23] may improve the growth bounds in our algorithm, but we have not fully explored this.

4. Complexity analyses. In this section we analyze the costs of the algorithm. We begin with an analysis of the computational complexity (asymptotic number of machine instructions, including arithmetic operations). We then analyze the communication costs of the algorithm. These costs consist of the number of words communicated and the number of messages communicated, which we refer to as the *bandwidth* and *latency* costs, respectively.

We use the word *bandwidth* in this section in two unrelated contexts. *Matrix bandwidth* refers to the location of the nonzero elements in a matrix and is distinct from the *bandwidth cost* of an algorithm. The reader is encouraged to keep this in mind so as not to confuse the two.

4.1. Computational cost. Our goal in this subsection is to show that the new block algorithm performs the same number of arithmetic operations as the elementwise one, up to low-order terms.

In order to determine the arithmetic complexity of the algorithm, we consider only (AA1)–(AA5) (the computational cost of pivoting is negligible). Letting J denote the index of the outermost loop of the algorithm, and letting b be the block size, the arithmetic cost of (AA1)–(AA3) is $O(Jb^3)$ flops. This is because each equation involves $O(J)$ block multiplications of b-by-b blocks (some of which are triangular). Note that in (AA2), the dominant cost is in computing the product of the block row of L with the block column of W; the arithmetic cost of the two-sided symmetric solve is $O(b^3)$. Similarly, (AA5) is a triangular solve involving one block and has an arithmetic cost of $O(b^3)$. The dominant arithmetic cost for the overall algorithm comes from (AA4), which consists of two subcomputations: a matrix multiplication involving L and H and an LU decomposition of a block column. The arithmetic cost of the LU decomposition is $O(Jb^3)$. The matrix multiplication step multiplies an $(N-J)b$-by-Jb submatrix of L by a Jb-by-b submatrix of H. At the Jth step of the algorithm, this arithmetic cost is $2(N-J)Jb^3$ flops, ignoring lower-order terms. Summing over the outermost loop and using the fact that $N = n/b$, we have a total

arithmetic cost of

$$\sum_{J=2}^{N} \left(2(N-J)Jb^3 + O(Jb^3)\right) = \frac{1}{3}n^3 + o(n^3).$$

4.2. Communication costs. To determine the communication complexity of the algorithm, we must consider (AA1)–(AA5) as well as the cost of applying symmetric permutations to the trailing matrix. We analyze the three parts of the algorithm separately: block operations (all of the computations described in (AA1)–(AA5) with the exception of the *LU* decomposition), *LU* decomposition of block columns, and application of the permutations to the trailing matrix. We assume the matrix is stored in block-contiguous format with block size b, the same as the algorithmic block size. In block-contiguous format, b-by-b blocks are stored contiguously in memory. The ordering of elements within blocks and the ordering of the blocks do not affect communication costs; we use column-major ordering. In the following analysis, we assume $b \leq \sqrt{M/3}$, where M is the size of fast memory, so that three blocks fit simultaneously in fast memory.

4.2.1. Block operations. By excluding the *LU* decomposition, all the other computations in (AA1)–(AA5) involve block operations—either block multiplication (sometimes involving triangular or symmetric matrices), block triangular solve, or block two-sided symmetric triangular solve. For example, in (AA3), we compute $H_{I,J}$ as $T_{I,I-1}(L_{J,I-1})^T + T_{I,I}(L_{J,I})^T + (T_{I+1,,I})^T(L_{I+1,J})^T$ (assuming only the lower halves of T and L are stored). Each of the three multiplications involve b-by-b blocks, so by the assumption that $b \leq \sqrt{M/3}$, the operations can be performed by reading contiguous input blocks of size b^2 words into fast memory, performing $O(b^3)$ floating-point operations, and then writing the output block back to slow memory. This implies that the number of messages is proportional to the number of block operations, which is $O((\text{computational cost})/b^3) = O(n^3/b^3)$ and the number of words moved is $O((\text{computational cost})/b) = O(n^3/b)$.

4.2.2. Panel decomposition. We now consider algorithms for the *LU* decomposition of the column panel. Note that the $O(N)$ *LU* factorizations, each involving $O(Nb^3)$ flops, contribute altogether only an $O(n^2b)$ term to the arithmetic complexity of the overall algorithm, a lower-order term. Thus, attaining the communication lower bound for the overall algorithm does not require minimizing communication within panel factorizations. For example, using a naive algorithm and achieving only constant reuse of data during the *LU* factorization translates to a total of $O(n^2b)$ words moved during *LU* factorizations, which is dominated by the communication complexity of the block operations, $O(n^3/b)$ words, in the case where $n \gg b^2$. However, to ensure that both bandwidth and latency costs of the *LU* factorizations do not asymptotically exceed the costs of the rest of the overall algorithm for all matrix dimensions, we need algorithms that achieve better than constant reuse (though the algorithms need not be asymptotically optimal). The bandwidth and latency costs of the naive algorithm are summarized in the first row of Table 1.

We choose to use the recursive algorithm (RLU) of [15, 33] for panel factorizations, updated slightly to match the block-contiguous data layout. The algorithm works by splitting the matrix into left and right halves, factoring the left half recursively, updating the right half, and then factoring the trailing matrix in the right half recursively. In order to match the block-contiguous layout, the update of the right half (consisting of a triangular solve and matrix multiplication) should be performed block

TABLE 1
Communication costs of LU decomposition algorithms applied to an n-by-b matrix stored in b-by-b block-contiguous storage, assuming $b \leq \sqrt{M/3}$.

Algorithm	Words	Messages
Naive	$O\left(nb^2\right)$	$O\left(n\right)$
RLU	$O\left(\frac{nb^2}{\sqrt{M}} + nb\log b\right)$	$O\left(\min\left(n, \frac{n^2}{M}\right)\right)$
SMLU [7]	$O\left(\frac{nb^2}{\sqrt{M}} + nb\log b\log\frac{nb}{M}\right)$	$O\left(\frac{nb^2}{M^{3/2}} + \frac{nb}{M}\log b\log\frac{nb}{M}\right)$
TSLU [14]	$O\left(\frac{nb^2}{\sqrt{M}}\right)$	$O\left(\frac{nb^2}{M^{3/2}}\right)$

by block. The bandwidth cost of this algorithm for n-by-b matrices is analyzed in [33], and the latency cost can be bounded by the recurrence $L(n,b) \leq 2L(n,b/2) + O(N)$. The $O(N)$ term comes from the update of the right half of the matrix, which involves reading contiguous chunks of each of the $O(N)$ blocks in the panel. The base case occurs either when the subpanel fits in memory ($nb < M$) or when $b = 1$. The cost of the recursive algorithm is dominated by its leaves, each of which requires $O(N)$ messages. Depending on the relative sizes of n and M, there are either nb/M or b leaves starting with an n-by-b matrix. The latency cost becomes the minimum of two terms: $O(n)$ or $O(n^2/M)$. The communication costs are given in the second row of Table 1.

In order to determine the contribution of LU factorizations to the costs of the overall algorithm, we must multiply the cost of the n-by-b factorization by N, the number of panel factorizations. Using the RLU algorithm, this yields a bandwidth cost of $O(n^2b/\sqrt{M} + n^2\log b)$ words and a latency cost of $O(\min(n^2/b, n^3/(bM)))$ messages. With the exception of the $O(n^2\log b)$ term in the bandwidth cost, these costs are always asymptotically dominated by the costs of the block operations.

While the RLU algorithm is sufficient for minimizing communication in the overall algorithm, there are algorithms which require fewer messages communicated. The shape-morphing LU algorithm (SMLU) [7] is an adaptation of RLU that changes the matrix layout on the fly to reduce latency cost. The algorithm and its analysis are provided in [7], and the communication costs are given in the third row of Table 1. SMLU uses partial pivoting and incurs a slight bandwidth cost overhead compared to RLU (an extra logarithmic factor). Another algorithm which reduces latency cost even further is the communication-avoiding TSLU [14]. The algorithm can be applied to general matrices, but the main innovation focuses on tall-skinny matrices. TSLU uses tournament pivoting, a different scheme than partial pivoting, which has slightly weaker theoretical numerical stability properties. The algorithm and analysis are provided in [14], and the communication costs are given in the bottom row of Table 1. The communication costs of TSLU are optimal with respect to each panel factorization.

4.2.3. Applying symmetric permutations. After each LU decomposition of a block column, we apply the internal permutation to the rest of the matrix. This permutation involves back-pivoting, or swapping rows of the already factored L matrix, and forward-pivoting of the trailing symmetric matrix. Applying the symmetric permutations to the trailing matrix includes swapping elements within a given set of rows and columns, as shown in Figure 12. For example, applying the transposition (k,l) implies that the L-shaped set of elements in the kth row and kth column (to the left and below the diagonal) is swapped with the L-shaped set of elements in the lth

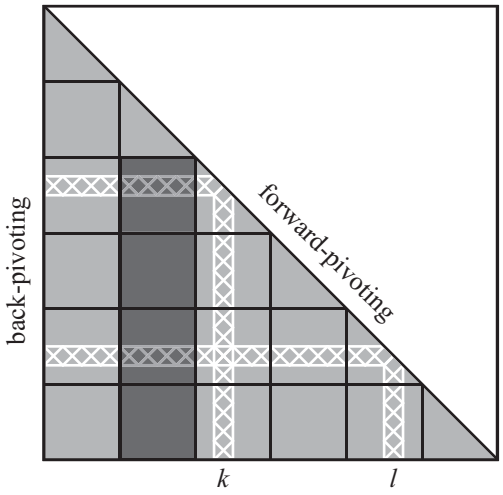

FIG. 12. *Exchanging rows and columns k and l. The second (dark) block column is the block column of the reduced matrix whose LU factorization was just computed. The first block column is a block column of L; the algorithm applies back-pivoting to it. Block columns* 3 *to* 6 *are part of the trailing submatrix; the algorithm applies forward-pivoting to them. The trailing submatrix is square and symmetric, but only its lower triangle is stored, so a row that needs to be exchanged is represented as a partial row (up to the diagonal) and a partial column, as shown here.*

row and lth column, such that element a_{kk} is swapped with element a_{ll} and element a_{kl} stays in place.

Since there are at most b swaps that must be performed for a given LU decomposition, and each swap consists of $O(n)$ data, the direct approach of swapping L-shaped sets of elements one at a time has a bandwidth cost of $O(nb)$ words. However, no matter how individual elements within blocks are stored, because the permutations involve accessing both rows and columns, at least half of the elements will be accessed noncontiguously, so the latency cost of the direct approach is also $O(nb)$ messages. Since there are $N = n/b$ symmetric permutations to be applied, these costs amount to a total of $O(n^2)$ words and $O(n^2)$ messages. While the bandwidth cost is a lower-order term with respect to the rest of the algorithm, the latency cost of the permutations exceeds the rest of the algorithm, except when $n \gg M^{3/2}$. This approach is the symmetric analogue of Variant 1 in [14], which is a direct method for applying row permutations to the trailing matrix in a nonsymmetric LU factorization.

In order to reduce the latency cost, we use a block approach which will require a greater bandwidth cost than the direct approach but will not increase the asymptotic bandwidth cost of the overall algorithm. The block approach accesses contiguous b-by-b blocks, but it may permute only a few rows or columns of the blocks. This approach is the symmetric analogue of Variant 2 in [14], which is a block method for applying row permutations to the trailing matrix in a nonsymmetric LU factorization.

The algorithm works as follows: for each block in the LU factorization panel that includes a permuted row, we update the N pairs of blocks shown in Figure 13. The updates include back-pivoting (updating parts of the L matrix that have already been computed) and forward-pivoting (updating the trailing matrix). Nearly all the updates involve pairs of blocks, which fit in fast memory simultaneously. Pairs of blocks involved in back-pivoting are not affected by column permutations and swap

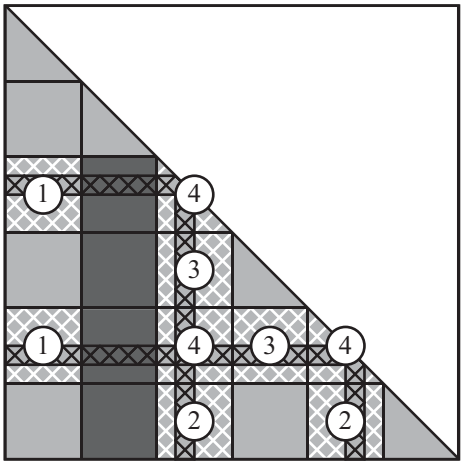

FIG. 13. *Exchanging a pair of rows in the block approach. Numbers indicate sets of blocks that are held simultaneously in fast memory: all the blocks marked* "1" *are held in fast memory simultaneously, later all the blocks marked* "2"*, and so on.*

TABLE 2
Communication costs of schemes for applying symmetric permutations.

Algorithm	Words	Messages
Direct	$O(nb)$	$O(nb)$
Block	$O(n^2)$	$O\left(\frac{n^2}{b^2}\right)$

only rows. Some pairs of blocks involved in forward-pivoting are not affected by row permutations and swap only columns. Because only half of the matrix is stored, some pairs of blocks in the trailing matrix will swap columns for rows. The more complicated updates involve blocks which are affected by both row and column permutations: the two diagonal blocks and the corresponding off-diagonal block, shown in Figure 13. In order to apply the two-sided permutation to these blocks, all three blocks are read into fast memory and updated at once. Since there are $O(N)$ blocks in each LU factorization panel, and each block with a permuted row requires accessing $O(N)$ blocks to apply the symmetric permutation, for a given LU factorization, the number of words moved in applying the associated permutation is $O(N^2b^2) = O(n^2)$, and the total number of messages moved is $O(N^2) = O(n^2/b^2)$.

The communication costs of the two approaches are summarized in Table 2.

5. A communication lower bound. To claim that our algorithm is communication optimal, we need to show a lower bound on the number of cache misses that any schedule for executing the algorithm must generate. We will appeal to the main results of [4, 6], which we repeat for completeness in Appendix A.

Inspecting the algorithm, we note that all the elements of L are computed using the following expressions:

$$\begin{aligned} l_{ii} &= 1 && \text{for } 1 \le i \le n\,, \\ l_{ij} &= 0 && \text{for } 1 \le j \le b,\ i > j\,, \\ l_{ij} &= \tfrac{1}{h_{j,j-b}}\left(a_{i,j-b} - \textstyle\sum_{k=b+1}^{j-b} l_{ik}h_{k,j-b}\right) && \text{for } b < j < i \le n\,. \end{aligned}$$

The lower bound assumes that elements of L are computed using these expressions, and that elements of L and H are computed only once. The bound does not depend on how elements of H and T are computed and it does not depend on the order of summations in the computation of l_{ij}. Similar assumptions are made in virtually all the communication lower bounds for matrix algorithms [6, 19, 21]. These assumptions admit a wide range of algorithms and schedules, but they do not admit completely different ways of computing the factorization, such as Strassen-like algorithms. We now state and prove this lower bound formally as a corollary of Theorem A.3.

THEOREM 5.1. *Any algorithm that computes the symmetric banded factorization $A = LTL^T$ while computing L using the expressions above and while computing elements of L and H at most once must transfer at least*

$$\frac{n^3}{48\sqrt{M}} - M - \frac{n^2}{2} - nb$$

words between slow and fast memory. The number of cache misses must be at least this bound divided by M. For large n, this lower bound is $\Omega\left(\frac{n^3}{\sqrt{M}}\right)$.

Proof. We first verify that computing the factorization LTL^T is a 3NL computation, as defined in Definition A.1, though with temporary operands. We follow the notation in that definition: let f_{ij} be the function defined above for l_{ij}, $b < j < i \le n$, and let g_{ijk} be the scalar multiplication of l_{ik} and $h_{k,j-b}$. Then we make the correspondences that l_{ij} is stored at location $\mathfrak{c}(i,j) = \mathfrak{a}(i,j)$ and $h_{i,j-b}$ is stored at location $\mathfrak{b}(i,j)$. Since L is an input stored in slow/global memory, the mappings $\mathfrak{a}$ and $\mathfrak{c}$ are one-to-one into slow/global memory. Further, functions f_{ij} (involving a summation of g_{ijk} outputs) and g_{ijk} (scalar multiplication) depend nontrivially on their arguments. Thus, the computation is 3NL, with the exception that $\mathfrak{b}$ is not necessarily a mapping into slow/global memory, and the number of temporary operands is the number of nonzeros in the matrix $H = TL^T$.

Since the factorization LTL^T is a 3NL computation with temporary operands, we can apply Theorem A.3. Thus, we need only determine the values of G and t in the statement of the theorem. In the case of an n-by-n matrix, the number of scalar multiplications is $G = n^3/6 + O(n^2 b)$ and the number of nonzeros in H is $n^2/2 + O(nb)$. When $n \ge \sqrt{M}$, the $n^3/\sqrt{M}$ term asymptotically dominates the (negative) M and $O(n^2)$ terms. ∎

6. Numerical experiments. Next we describe a set of numerical experiments that provide further insight into the numerical behavior of the algorithm.

We carried out three sets of experiments with both random matrices and matrices from the University of Florida Sparse Matrix Collection [11]. We factored each matrix twice, once using the block Aasen algorithm and once, for comparison, using the LAPACK subroutine `SYSV` that implements the Bunch–Kaufman algorithm. In our algorithm we used a block size $b = 256$ as in the companion paper [5] and factorized panels using GEPP (corresponding to panel factorization strategies RLU and SMLU in Table 1).

6.1. Experiments with random matrices. In the first set of experiments we generated a sequence of random square symmetric matrices of order n for 100 distinct values of n, linearly spaced in the interval $500 \le n \le 5{,}000$. The elements of these matrices are distributed normally and independently (preserving symmetry, of course) with mean 0 and standard deviation 1.

We used three parameters to evaluate the factorizations: the growth factor, the

error of the factorization, and the normwise backward error in the solution of a linear system of equations $Ax = f$. The right-hand side f in this system was generated by postmultiplying A with a random vector whose elements were produced in the same way as the elements of A. Solving $Ax = f$ using our algorithm requires a solver for banded systems involving the factor T; this role was played in our experiments by the LAPACK subroutine `GBTRF` that implements the banded LU factorization with partial pivoting. To solve $Ax = f$ using the Bunch–Kaufman factorization, we used `SYSV`, the LAPACK subroutine that we also used to compute the factorization. `SYSV` solves a linear system using the Bunch–Kaufman factors directly and does not require their further factorization. We computed the backward error using the formula

$$\frac{\|f - A\hat{x}\|_\infty}{\|A\|_\infty \|\hat{x}\|_\infty} , \tag{6.1}$$

where $\hat{x}$ is the computed solution. This formula produces the norm of the matrix Δ in Theorem 3.13 (see [18, Theorem 7.1]). We also computed the two growth factors

$$\rho_1 = \frac{\|T\|_M}{\|A\|_M} , \qquad \rho_2 = \frac{\|U_1\|_M}{\|A\|_M} , \tag{6.2}$$

where $\|X\|_M = \max_{i,j} |X_{i,j}|$ denotes the max-norm of a matrix X, and U_1 denotes the upper-triangular factor of the LU factorization with partial pivoting of T. Theorem 3.13 provides us with a componentwise bound on the backward error from which we can derive a normwise bound of the form

$$\|\Delta\|_M \leq u \cdot p\,(n, b)\,(\|T\|_M + \|U_1\|_M) , \tag{6.3}$$

where $p\,(n, b)$ is a polynomial. This means that ρ_1 and ρ_2 indicate the size of the backward error, and for this reason we consider them as the growth factors of the factorization. In our experiments we found that the two are always of the same order of magnitude, and therefore we only show ρ_1 in the figures in this section. For the Bunch–Kaufman factorization we computed growth using the analogous definition

$$\frac{\|D\|_M}{\|A\|_M} ,$$

where D is the block-diagonal factor computed by the algorithm. Here we do not require two growth factors as we do with our algorithm (compare Theorem 3.13 with [16, Theorem 4.1]). Finally, we define the factorization error as

$$\max_{i,j} \frac{\left|A - LTL^T\right|_{i,j}}{\left(|L|\,|T|\,|L|^T\right)_{i,j}} ,$$

with the convention that $0/0 = 0$. In this formula we divide the elements of the matrix on the left-hand side of the bound of Theorem 3.10 by the elements of the matrix on its right-hand side, which allows us to determine how sharp the analysis in that theorem is. The factorization error should not be considered as the *backward error* of the factorization (for that, the denominator in the definition would need to be $|A_{i,j}|$).

For the random matrices, the stability of solutions to linear systems and the growth factors are shown in Figure 14. The backward errors are moderate, varying

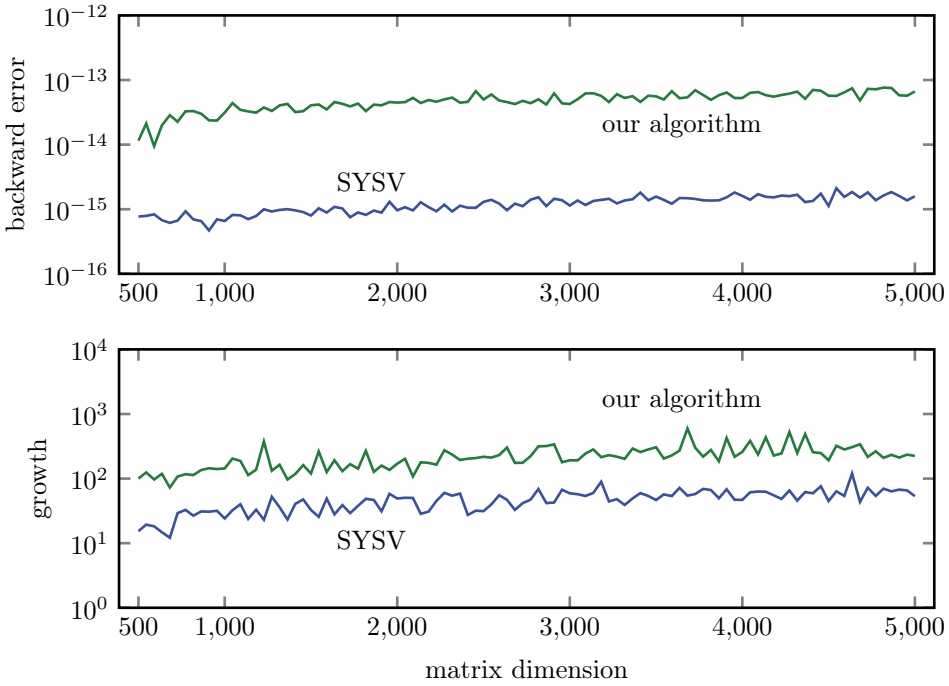

FIG. 14. *Results for random matrices. The top plot shows the backward error in the solution of $Ax = f$ as defined by* (6.1)*, and the bottom one shows the growth factor ρ_1 in the factorization of A as defined by* (6.2).

between 9.4×10^{-15} and 7.6×10^{-14} with a median of 4.9×10^{-14}. The backward error is increasing with n but at a rate that is slower than linear. The growth factor is strongly correlated with the error, which agrees with the bound in Theorem 3.13 and indicates that the polynomial $p(n, b)$ in (6.3) does not have a significant dependence on n. The factorization errors are small, between $2.3u$ and $4.9u$ with a median of $3.8u$, and do not seem to depend on n, in contrast with the bound in Theorem 3.10.

The solutions produced by `SYSV` are consistently more stable, with backward errors that are smaller by up to two orders of magnitude. In our experiments we found that this difference is largely explained by the block size in our algorithm. When we decrease the block size, backward errors and growth factors decrease accordingly until they ultimately reach the same level as that of Bunch–Kaufman. Section 6.3 explores this question in more detail.

6.2. Experiments with real-world matrices. The second set of experiments factored 143 matrices from the University of Florida Sparse Matrix Collection. We chose for this experiment *all* of the symmetric, real, nonpattern matrices of order $512 \leq n \leq 16{,}386$, with the exception of matrices with bandwidth b or less. Matrices with low bandwidth were omitted because they are their own T factors and therefore do not require factorization. This set of matrices is further described in Table 3; a list of the matrices is included as supplementary material to this paper. The experiment was conducted according to the same scheme as the experiment involving the random matrices.

For the 131 matrices on which the algorithm produced good results, the stability

TABLE 3

The University of Florida matrix set. The table shows the statistics of the dimension of the matrix n; of the average number of nonzero elements in each column `nnz`$/n$*, where* `nnz` *is the number of nonzero elements in the whole matrix; and of the 2-norm condition number κ.*

	n	`nnz`$/n$	κ
minimum	662.0	0.5	1.8×10^{1}
1st quartile	2,000.0	6.9	1.2×10^{4}
median	5,041.0	11.7	2.6×10^{7}
3rd quartile	9,899.3	23.2	7.5×10^{11}
maximum	16,146.0	1,260.0	`inf`

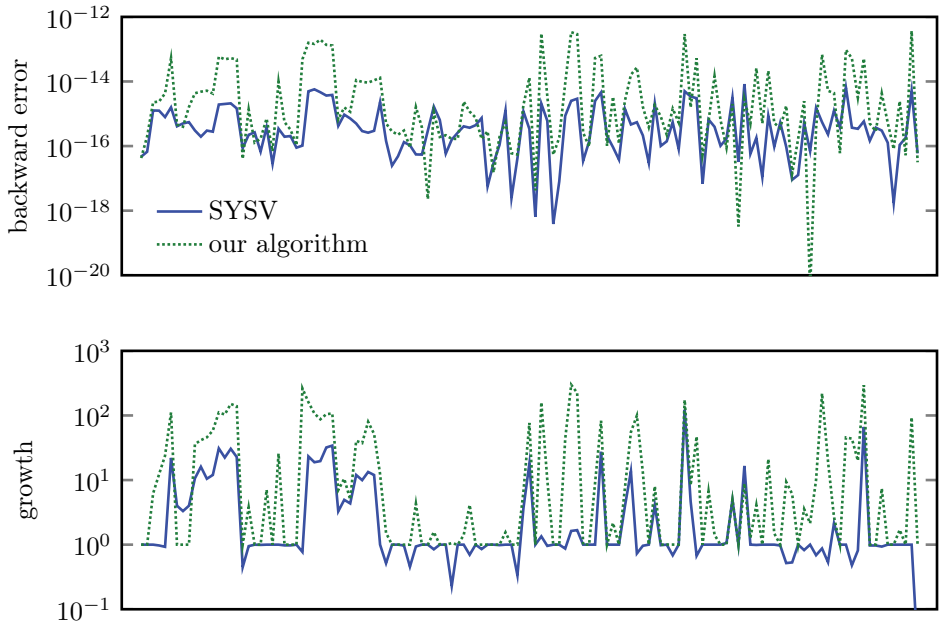

FIG. 15. *Results for matrices from the University of Florida Sparse Matrix Collection. The top plot shows the backward error in the solution of $Ax = f$ as defined by* (6.1)*, and the bottom one shows the growth factor ρ_1 in the factorization of A as defined by* (6.2)*. The matrices are shown in increasing order of n.*

of the linear solver and the growth factor are shown in Figure 15. The linear-solver backward errors are between 3.6×10^{-21} and 3.7×10^{-13}, with a median of 1.2×10^{-15}. The backward errors that we obtain from our algorithm are moderate and comparable to those of `SYSV`, although in a few cases the difference can be as high as three digits of accuracy. The factorization errors were small in all but 3 of the matrices, reaching at most a level of $34u$. In three matrices we found large factorization errors, which were caused by underflow and by subnormal numbers that were obtained during the factorization. This turned out to be harmless because the large relative errors that we obtained in individual arithmetic operations on tiny numbers translated to tiny normwise backward errors.

On 12 matrices, our linear solver failed to produce a solution. In these matrices, the LU factorization with partial pivoting of T produced a U factor, some of whose

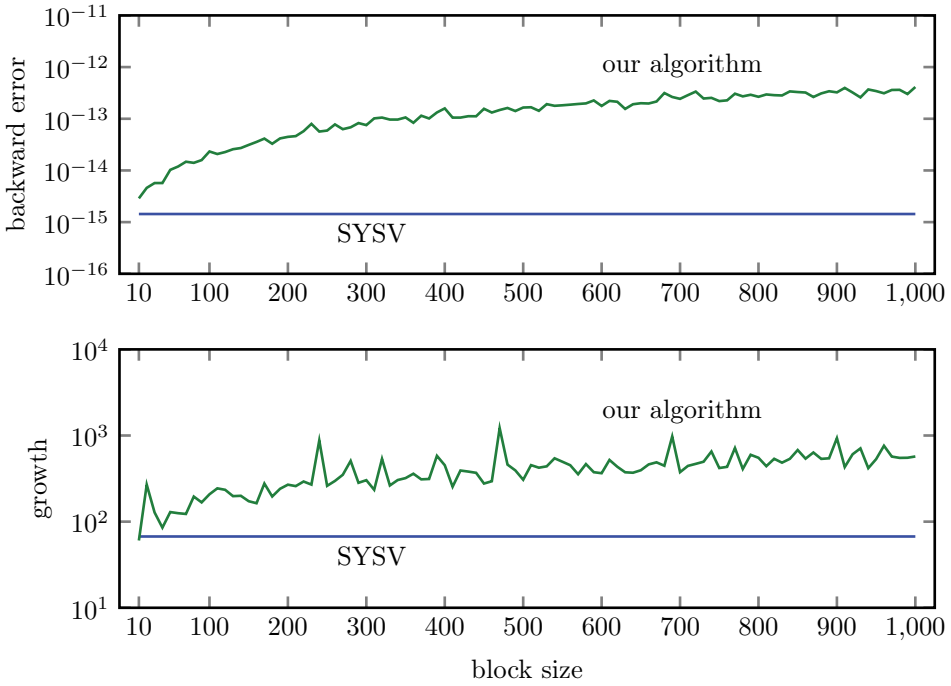

FIG. 16. *Results for random matrices, factorized using various values of the block size. The top plot shows the backward error in the solution of $Ax = f$ as defined by* (6.1)*, and the bottom one shows the growth factor ρ_1 in the factorization of A as defined by* (6.2)*. When plotted on linear-linear scales, our algorithm's backward error is close to linear. We computed only one factorization using* SYSV *because in that algorithm changing the block size does not affect the factors.*

diagonal elements were equal to zero. Because such a U factor cannot be used to solve linear systems by substitution, this caused the banded solver to indicate an error. In all 12 matrices the root cause was structural or numerical rank deficiency of A (MATLAB reported condition numbers larger than 10^{18}). Our factorization algorithm produced stable factorizations, with factorization errors of order u, well-conditioned L's, and typical growth, never greater than 100. We note that the Bunch–Kaufman algorithm also failed to produce a solution for these matrices.

6.3. The effect of the block size on numerical stability. All of the experiments we have described so far were conducted with a block size $b = 256$. Additional experimentation shows that the backward error depends on b. Figure 16 shows this phenomenon on a random matrix of dimension 5,000 that we factored with various values of b. The results show that the linear-solver backward error increases proportionally to the block size, growing by roughly two orders of magnitude as b is increased from 10 to 1,000.

The growth factor does not increase as quickly, growing by only one order of magnitude and failing to fully reflect the dependence on b. The reason for the linear dependence on b is the polynomial in (6.3), which also increases with b. The cause of this is the growth of the elements of the matrices $|L|\,|T|\,|L|^T$ and $|L|\,P_1^T\,|L_1|\,|U_1|\,|L|^T$, which feature in our bound on the backward error in Theorem 3.13. The elements of these matrices appear to increase proportionally with the number of nonzero elements in $|T|$ and $P_1^T\,|L_1|\,|U_1|$, and this number is typically proportional to b and equals $\Theta(nb)$

TABLE 4
The memory hierarchy structure of our machine. The machine consists of eight 6-core AMD Opteron 8439SE processors.

every core	
L1 data cache	64 KB 2-way associative
L1 instruction cache	64 KB 2-way associative
L2 cache	512 KB 16-way associative
every six cores	
L3 cache	6 MB 48-way associative
all cores	
main memory	126 GB

if T is full within its band and there is no numerical cancellation in its LU factorization with partial pivoting.

Nevertheless, it appears that this effect is not universal in all matrices. When we tried varying the block size in the factorization of sparse matrices, we found that the correlation between the backward errors and b is as likely to be negative as it is to be positive. The reason for this is that increasing the block size can make $|L|$ sparser, which compensates for the increase in the number of nonzero elements in $|T|$ and $P_1^T\,|L_1|\,|U_1|$.

Furthermore, we also found that the effect of increasing the block size can usually be reversed using working-precision iterative refinement. In all of the nonsingular matrices in our experiments, one step of iterative refinement was sufficient to reduce the backward error below a level of $10u$.

6.4. The effect of the block size on performance. Our analysis of communication costs in sections 4 and 5 shows that the algorithm is optimal if the block size is $\Theta(\sqrt{M})$, where M is the size of fast memory. This indicates that, similarly to other communication-avoiding algorithms, obtaining top performance from our algorithm requires that we tailor b to the machine on which the algorithm runs. Our experience indicates that using a larger b allows BLAS to take better advantage of the memory hierarchy, but making b smaller allows the algorithm to scale on a larger number of cores. To explore this, we computed the factorization of a matrix of order 42,000 on a machine of eight 6-core processors, described in detail in Table 4. We allowed the block size and the number of threads to vary and recorded the flop rates that we obtained. The results are shown in Table 5.

The optimal value of b that we found in the experiment was 200 for 24 threads and 300 for 48 threads. Choosing a different nearby value, such as $b = 300$ for 24 threads or $b = 200$ for 48 threads does not make a significant difference, decreasing performance by 2% and 5%, respectively. If b is far from the optimum, the difference becomes dramatic; for $b = 100$, performance is down by 49% and 63%, respectively.

The data confirm that b must be chosen from a specific range and indicate that the range is quite large. Although performance is not extremely sensitive within that range, this represents a constraint on the value of b that we can choose. If the machine has a large fast memory and a small number of cores, we may be forced to make b large, which can have a detrimental effect on numerical stability, although, as we mention in section 6.3, iterative refinement can often reverse this effect.

TABLE 5

The gigaflops per second rate in the factorization of a matrix of order 42,000. *Only the reduction of A to T (phase* 1 *of the algorithm) was computed. The rate is shown as a function of the block size and the number of threads.*

	block size					
threads	100	200	300	400	500	600
24	67.21	132.30	130.13	113.73	118.47	113.59
48	65.89	168.46	177.35	147.69	157.62	149.61

7. Conclusions. We have shown that a block variant of Aasen's factorization algorithm can reduce a symmetric matrix into a symmetric banded form in a communication-avoiding way. A companion conference paper [5] showed that the algorithm performs well in practice on a multicore machine; here we focused on complete analyses of the algorithm's communication costs, arithmetic costs, and numerical stability. No prior symmetric reduction algorithm achieves similar efficiency bounds.

Appendix A. General lower bounds. For completeness, we state here the communication lower bounds proved in [6] for a general set of linear algebraic computations. We use terminology introduced in [4, section 4], but the main content (including the proofs of the theorems) is nearly identical to [6].

We first define our model of computation formally and illustrate it on the case of matrix multiplication: $C = C + A \cdot B$. Let $S_a \subseteq \{1, 2, \ldots, n\} \times \{1, 2, \ldots, n\}$, corresponding in matrix multiplication to the subset of entries of the indices of the input matrix A that are accessed by the algorithm (e.g., the indices of the nonzero entries of a sparse matrix). Let $\mathcal{M}$ be the set of locations in slow/global memory (on a parallel machine $\mathcal{M}$ refers to a location in some processor's memory; the processor number is implicit). Let $\mathfrak{a} : S_a \mapsto \mathcal{M}$ be a mapping from the indices to memory, and similarly define S_b, S_c and $\mathfrak{b}(\cdot,\cdot), \mathfrak{c}(\cdot,\cdot)$, corresponding to the matrices B and C. The value of a memory location $l \in \mathcal{M}$ is denoted by $Mem(l)$. We assume that the values are independent—i.e., determining any value requires accessing the memory location.

DEFINITION A.1 (3NL computation). *A computation is considered to be three nested loops (3NL) if it includes computing, for all* $(i, j) \in S_c$ *with* $S_{ij} \subseteq \{1, 2, \ldots, n\}$,

$$Mem(\mathfrak{c}(i,j)) = f_{ij}\left(\{g_{ijk}(Mem(\mathfrak{a}(i,k)), Mem(\mathfrak{b}(k,j)))\}_{k \in S_{ij}}\right),$$

where

(a) *mappings* $\mathfrak{a}$, $\mathfrak{b}$, *and* $\mathfrak{c}$ *are all one-to-one into slow/global memory, and*

(b) *functions* f_{ij} *and* g_{ijk} *depend nontrivially on their arguments.*

Further, define a 3NL operation as an evaluation of a g_{ijk} *function, and let* G *be the number of unique 3NL operations performed:*

$$G = \sum_{(i,j) \in S_c} |S_{ij}|.$$

Note that while each mapping $\mathfrak{a}$, $\mathfrak{b}$, and $\mathfrak{c}$ must be one-to-one, the ranges are not required to be disjoint. By requiring that the functions f_{ij} and g_{ijk} depend "nontrivially" on their arguments, we mean the following: we need at least one word of space to compute f_{ij} (which may or may not be $Mem(\mathfrak{c}(i,j))$) to act as "accumulator" of the value of f_{ij}, and we need the values $Mem(\mathfrak{a}(i,k))$ and $Mem(\mathfrak{b}(k,j))$ to be in fast

or local memory before evaluating g_{ijk}. Note that f_{ij} and g_{ijk} may depend on other arguments, but we do not require that the functions depend nontrivially on them. Note also that we may not know until after the computation what S_c, f_{ij}, S_{ij}, or g_{ijk} were, since they may be determined on the fly.

We now state the communication lower bound for 3NL computation, also appearing as [6, Theorem 2.2] and [4, Theorem 4.2].

THEOREM A.2. *The bandwidth cost lower bound of a 3NL computation (Definition* A.1*) is*

$$W \geq \frac{G}{8\sqrt{M}} - M \ ,$$

where M is the size of the fast/local memory.

Many linear algebraic computations are nearly 3NL but fail to satisfy the assumption that $\mathfrak{a}$, $\mathfrak{b}$, and $\mathfrak{c}$ are one-to-one mappings into slow/global memory. The following result asserts that, under certain assumptions, we can still prove meaningful lower bounds.

First, we define a *temporary value* as any value involved in a computation that is not an original input or final output. In particular, a temporary value need not be mapped to a location in slow memory. Next, we distinguish a particular set of temporary values: we define the temporary inputs to g_{ijk} functions and temporary outputs of f_{ij} functions as *temporary operands*. While there may be other temporary values involved in the computation (e.g., outputs of g_{ijk} functions), we do not consider them temporary operands.[3] A temporary input $\mathfrak{a}(i,k)$ may be an input to multiple g_{ijk} functions (g_{ijk} and $g_{ij'k}$ for $j \neq j'$), but we consider it a single temporary operand. There may also be multiple accumulators for one output of an f_{ij} function, but we consider only the final computed output as a temporary operand.

We now state the result more formally, which also appears as [4, Theorem 4.10] and corresponds to [6, section 3.4].

THEOREM A.3. *Suppose a computation is 3NL except that some of its operands (i.e., inputs to g_{ijk} operations or outputs of f_{ij} functions) are temporary and are not necessarily mapped to slow/global memory. Then if the number of temporary operands is t, and if each (input or output) temporary operand is computed exactly once, then the bandwidth cost lower bound is given by*

$$W \geq \frac{G}{8\sqrt{M}} - M - t \ ,$$

where M is the size of the fast/local memory.

Acknowledgments. We thank Hua Xiang and Laura Grigori for helpful discussions. We also thank the associate editor Ilse Ipsen and the anonymous referees for bringing the paper by Reid and Scott [28] to our attention and for their valuable comments.

REFERENCES

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

[3] We ignore these other temporary values because, as in the case of true 3NL computations, they typically do not require any memory traffic.

[2] S. Amarasinghe, M. Hall, R. Lethin, K. Pingali, D. Quinlan, V. Sarkar, J. Shalf, R. Lucas, K. Yelick, P. Balaji, P. C. Diniz, A. Koniges, M. Snir, and S. R. Sachs, *Exascale Programming Challenges*, Report of the 2011 Workshop on Exascale Programming Challenges, Marina del Rey, 2011.

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

[4] G. Ballard, *Avoiding Communication in Dense Linear Algebra*, Ph.D. thesis, EECS Department, University of California, Berkeley, 2013.

[5] G. Ballard, D. Becker, J. Demmel, J. Dongarra, A. Druinsky, I. Peled, O. Schwartz, S. Toledo, and I. Yamazaki, *Implementing a blocked Aasen's algorithm with a dynamic scheduler on multicore architectures*, in Proceedings of the IEEE 27th International Parallel & Distributed Processing Symposium (IPDPS), 2013, pp. 895–907.

[6] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz, *Minimizing communication in numerical linear algebra*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 866–901.

[7] G. Ballard, J. Demmel, B. Lipshitz, O. Schwartz, and S. Toledo, *Communication efficient Gaussian elimination with partial pivoting using a shape morphing data layout*, in Proceedings of the 25th ACM Symposium on Parallelism in Algorithms and Architectures, 2013, pp. 232–240.

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

[9] J. R. Bunch and L. Kaufman, *Some stable methods for calculating inertia and solving symmetric linear systems*, Math. Comp., 31 (1977), pp. 163–179.

[10] D. E. Culler, R. M. Karp, D. Patterson, A. Sahay, E. E. Santos, K. E. Schauser, R. Subramonian, and T. von Eicken, *LogP: A practical model of parallel computation*, Comm. ACM, 39 (1996), pp. 78–85.

[11] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1–1:25.

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

[13] M. Frigo, C. E. Leiserson, H. Prokop, and S. Ramachandran, *Cache-oblivious algorithms*, in Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99), New York, 1999.

[14] L. Grigori, J. W. Demmel, and H. Xiang, *CALU: A communication optimal LU factorization algorithm*, SIAM J. Matrix Anal. Appl., 32 (2011), pp. 1317–1350.

[15] F. G. Gustavson, *Recursion leads to automatic variable blocking for dense linear-algebra algorithms*, IBM J. Res. Develop., 41 (1997), pp. 737–755.

[16] N. J. Higham, *Stability of the diagonal pivoting method with partial pivoting*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 52–65.

[17] N. J. Higham, *Notes on accuracy and stability of algorithms in numerical linear algebra*, in The Graduate Student's Guide to Numerical Analysis, M. Ainsworth, J. Levesley, and M. Marletta, eds., Springer, Berlin, Heidelberg, 1999, pp. 47–82.

[18] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, 2nd ed., SIAM, Philadelphia, 2002.

[19] J.-W. Hong and H. T. Kung, *I/O complexity: The red-blue pebble game*, in Proceedings of the 13th Annual ACM Symposium on Theory of Computing, 1981, pp. 326–333.

[20] D. Irony and S. Toledo, *The snap-back pivoting method for symmetric banded indefinite matrices*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 398–424.

[21] D. Irony, S. Toledo, and A. Tiskin, *Communication lower bounds for distributed-memory matrix multiplication*, J. Parallel Distrib. Comput., 64 (2004), pp. 1017–1026.

[22] L. Kaufman, *The retraction algorithm for factoring banded symmetric matrices*, Numer. Linear Algebra Appl., 14 (2007), pp. 237–254.

[23] A. Khabou, J. W. Demmel, L. Grigori, and M. Gu, *LU factorization with panel rank revealing pivoting and its communication avoiding version*, SIAM J. Matrix Anal. Appl., 34 (2013), pp. 1401–1429.

[24] R. S. Martin and J. H. Wilkinson, *Reduction of the symmetric eigenproblem* $Ax = \lambda Bx$ *and related problems to standard form*, Numer. Math., 11 (1968), pp. 99–110.

[25] 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 (1970), pp. 386–397.

[26] J. Poulson, B. Marker, R. A. van de Geijn, J. R. Hammond, and N. A. Romero, *Elemental: A new framework for distributed memory dense matrix computations*, ACM Trans. Math. Software, 39 (2013), pp. 13:1–13:24.

[27] J. Poulson, R. A. van de Geijn, and J. Bennighof, *(Parallel) Algorithms for Two-Sided Triangular Solves and Matrix Multiplication*, unpublished, 2012.
[28] J. K. Reid and J. A. Scott, *Partial factorization of a dense symmetric indefinite matrix*, ACM Trans. Math. Software, 38 (2012), pp. 10:1–10:19.
[29] M. Rozložník, G. Shklarski, and S. Toledo, *Partitioned triangular tridiagonalization*, ACM Trans. Math. Software, 37 (2011), pp. 38:1–38:16.
[30] A. Sankar, D. A. Spielman, and S.-H. Teng, *Smoothed analysis of the condition numbers and growth factors of matrices*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 446–476.
[31] M. P. Sears, K. Stanley, and G. Henry, *Application of a high performance parallel eigensolver to electronic structure calculations*, in Proceedings of the IEEE/ACM Conference on Supercomputing, 1998 (CD-ROM).
[32] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ikebe, V. C. Klema, and C. B. Moler, *Matrix Eigensystem Routines—EISPACK Guide*, 2nd ed., Springer-Verlag, 1976.
[33] S. Toledo, *Locality of reference in LU decomposition with partial pivoting*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 1065–1081.
[34] L. N. Trefethen and R. S. Schreiber, *Average-case stability of Gaussian elimination*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 335–360.
[35] L. G. Valiant, *A bridging model for parallel computation*, Comm. ACM, 33 (1990), pp. 103–111.
[36] J. S. Vitter and E. A. M. Shriver, *Optimal disk I/O with parallel block transfer*, in Proceedings of the 22nd Annual ACM Symposium on Theory of Computing (STOC), 1990, pp. 159–169.
[37] J. S. Vitter and E. A. M. Shriver, *Algorithms for parallel memory* I: *Two-level memories*, Algorithmica, 12 (1994), pp. 110–147.