Communication-Avoiding Symmetric-Indefinite Factorization

Grey Ballard
Dulceneia Becker
James Demmel
Jack Dongarra
Alex Druinsky
Inon Peled
Oded Schwartz
Sivan Toledo
Ichitaro Yamazaki

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2013-127
http://www.eecs.berkeley.edu/Pubs/TechRpts/2013/EECS-2013-127.html

July 5, 2013

Copyright © 2013, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

Supported by NSF CCF-1117062 and CNS-0905188, 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” grant 1045/09 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), and grant 2010231 from the US-Israel Bi-National Science Foundation, and by Microsoft (Award #024263) and Intel (Award #024894) funding, and matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung. Also supported by DE-SC0004938, DE-SC0005136, DESC0003959, DE-SC0008700, and AC02-

05CH11231, and DARPA grant HR0011-12-2-0016.

Communication-Avoiding Symmetric-Indefinite Factorization

GREY BALLARD[1], DULCENEIA BECKER[2], JAMES DEMMEL[1], JACK DONGARRA[2,3,4], ALEX DRUINSKY[5], INON PELED[5], ODED SCHWARTZ[1], SIVAN TOLEDO[5], AND ICHITARO YAMAZAKI[2]

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.

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 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 only requires $O(1)$ data transfers that move $\Theta(b^2)$ words, which we refer to as *messages*.

[1]University of California, Berkeley
[2]University of Tennessee, Knoxville
[3]Oak Ridge National Laboratory
[4]University of Manchester
[5]Tel Aviv University
Date: May 2013.

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 [4] 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 [9, 12, 17, 31, 32, 33]. 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 [12]). Because our algorithm transfers data using messages that each transfers $\Theta(b^2)$ 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* [5] (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 efficient when cache lines are small (n_{CL} is small), and they do not depend on a fully-associative cache; a 4-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; it improves upon an earlier algorithm by Parlett and Reid that computes the same factorization in $2n^3/3 + o(n^3)$ operations [22]. Neither algorithm is used extensively; a few years later Kaufman and Bunch discovered a similar factorization that proved to be more popular, one in which the tridiagonal T is replaced by a matrix that is block diagonal with 2-by-2 and 1-by-1 blocks [8].

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 Parlett-Reid-Aasen factorization with the same communication efficiency [25].

[1] The main contributions of this paper relative to the companion conference paper [4] are complete numerical and complexity analyses of the algorithm; these did not appear in the conference paper.

[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.

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 half bandwidth of T is $\Theta(\sqrt{M})$. 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 solved in floating-point arithmetic, the computed block of T can have a non-negligible 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 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 trailing blocks which might be smaller). 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 subdiagonal blocks 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

$$W = RL^T$$
$$H = TL^T .$$

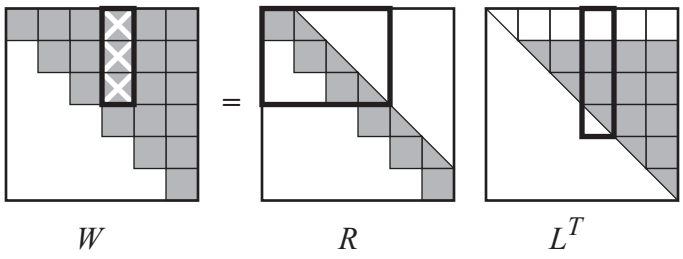

FIGURE 2.1. An illustration of computing superdiagonal blocks of W via matrix multiplication in Equation (AA1). Here $N = 6$ and $J = 4$. The blocks that participate in the equation are enclosed in thick rectangles, and the blocks that are computed using this equation are crossed. The same notation is used in other diagrams in this section.

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

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

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 formulas:

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

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

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

$$A_{J+1:N,J} = L_{J+1:N,1:J}H_{1:J,J} + \underline{L_{J+1:N,J+1}}\,\underline{H_{J+1,J}} \tag{AA4}$$

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

2.1. **Correctness.** We now show that the algorithm is correct. Verifying that the blocks that are computed in each equation only depend 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 2.1. 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, $W = RL^T$.

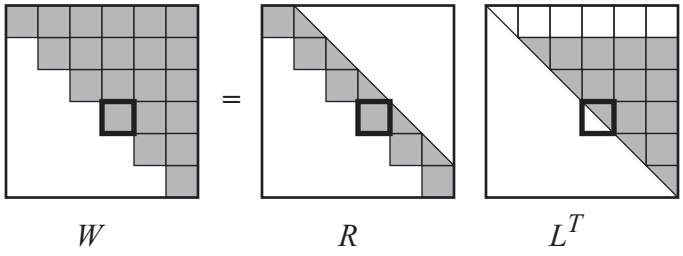

FIGURE 2.2. An expression for the diagonal blocks of W.

Equation (AA2) computes a diagonal block of T by solving a two-sided triangular linear system. This linear system can be solved by one of the existing solvers which we describe below. 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 [11]. The equation guarantees that

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

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 2.2, 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 2.4. Equation (AA4) multiplies blocks of L and H, subtracts the product from a block of A, and factors the difference using an LU factorization. Equation (AA5) solves a triangular linear system with a triangular right-hand side to compute a subdiagonal block of T.

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 $A = LTL^T$.

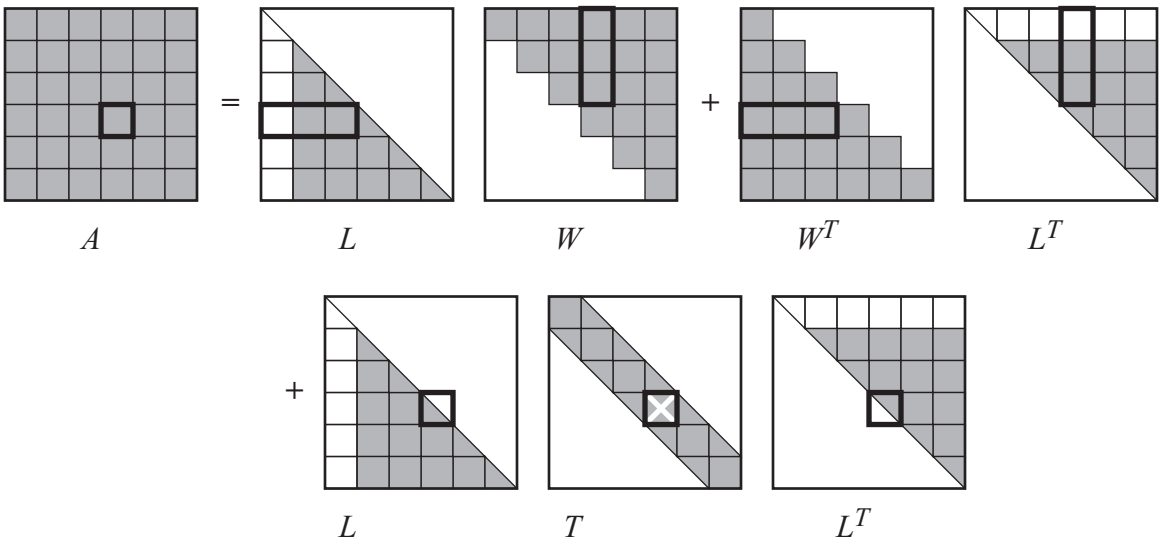

FIGURE 2.3. Computing a diagonal block of T in Equation (AA2) by updating the corresponding block of A and solving a two-sided triangular system. The letters below each matrix describe only the matrices involved in the expression for $A_{J,J}$; they do not constitute a matrix equation.

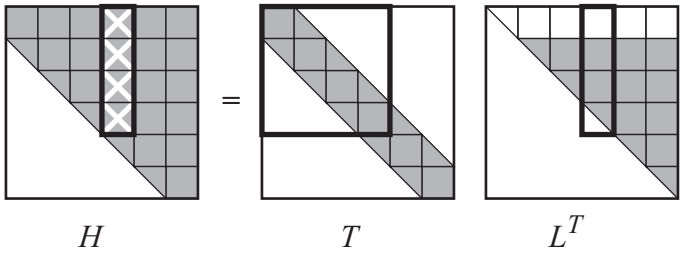

FIGURE 2.4. Computing blocks of H via matrix multiplication in Equation (AA3).

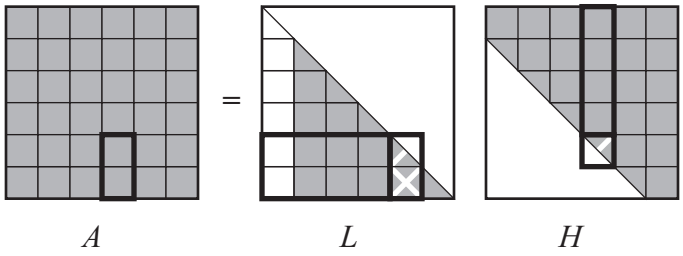

FIGURE 2.5. Computing a block column of L and a subdiagonal block of H in Equation (AA4) via the LU factorization of an updated submatrix of A.

2.2. **Solving Two-Sided Triangular Linear Systems.** We now describe the procedure that solves the two-sided triangular linear system in Equation (AA2). The method is not new; it is used to reduce symmetric generalized eigenproblems to standard eigenproblems and is available in LAPACK and ScaLAPACK under the name SYGST [7, 27]. Even though we are focused on SYGST in this paper, other solvers that produce a symmetric solution would also be suitable for the task. Examples of such solvers are the subroutine REDUC

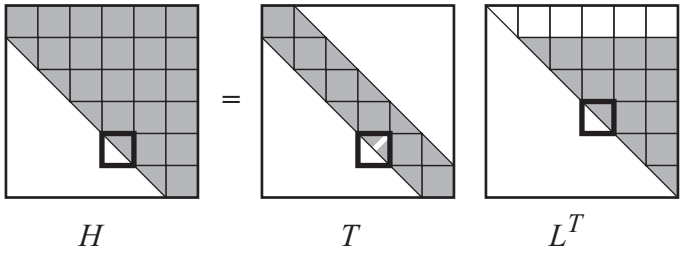

FIGURE 2.6. Computing a block of T by solving a triangular system in Equation (AA5).

in EISPACK [21, 28] and the algorithms implemented in the Elemental library [23, 24]. Because we apply the solver to block-sized problems, its flops and communications costs do not have a substantial impact on the overall costs of the algorithm. The stability of the solver is important, but as long as it satisfies a bound similar to the one we prove for SYGST in Section 3, the impact on the overall algorithm is limited to the size of the constant in the backward stability bound.

To the best of our knowledge the stability of SYGST has not been previously analyzed. In order to analyze the algorithm we will now describe the relevant details of how it works. The equation that defines $T_{J,J}$ is of the form $LXL^T = B$ with a symmetric right-hand side B.[3] A trivial way to solve such systems is using a conventional triangular solver twice. That is, to first solve for $L^{-1}B$ and to then solve for $X = (L^{-1}B)L^{-T}$. This method produces a solution X that is not exactly symmetric, and is thus not suitable for use in the block Aasen algorithm. Another approach, which leads to an exactly symmetric X and which performs only half the arithmetic, is an algorithm that we now describe. We partition all of the matrices that we introduce in this section such that they are all 2-by-2 block matrices with first diagonal blocks of dimensions b-by-b and second diagonal blocks of dimensions $(n-b)$-by-$(n-b)$. 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 H and W, which we will require to satisfy

$$H = XL^T \ , \qquad\qquad W = YL^T \ .$$

[3] This section uses self-contained notation, for simplicity. The matrix that we call L here is a diagonal block of the lower-triangular factor in the overall block Aasen factorization. In addition, the auxiliary matrices H and W, the dimension of the problem n and the block size b, all of which we define later in this section, are also distinct.

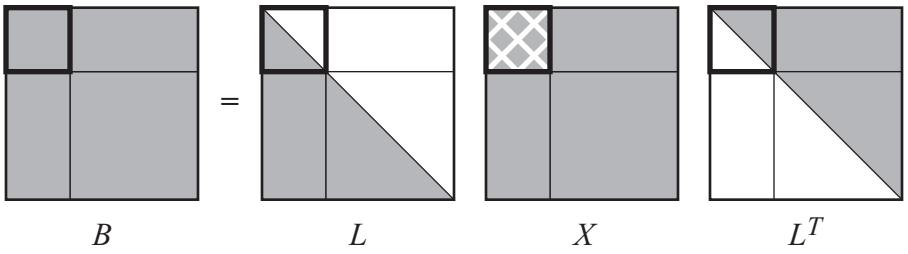

FIGURE 2.7. Computing the first diagonal block of X by solving a smaller two-sided triangular system in Step (ST1).

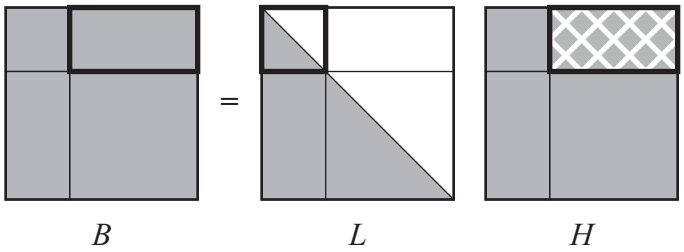

FIGURE 2.8. Solving a triangular system to compute the superdiagonal block of H in Step (ST2).

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

(ST1) $$B_{11} = L_{11}\underline{X_{11}}(L_{11})^T$$

(ST2) $$B_{12} = L_{11}\underline{H_{12}}$$

(ST3) $$H_{12} = 0.5X_{11}(L_{21})^T + \underline{W_{12}}$$

(ST4) $$W_{12} = 0.5X_{11}(L_{21})^T + \underline{X_{12}}(L_{22})^T$$

(ST5) $$B_{22} = L_{21}W_{12} + (L_{21}W_{12})^T + L_{22}\underline{X_{22}}(L_{22})^T \ .$$

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

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

$$B = LXL^T \ , \quad B = LH \ , \quad B = LW + (LW)^T \ , \quad H = Y^TL^T + W \ , \quad W = YL^T \ .$$

The derivation is described by diagrams in Figures 2.7–2.11.

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

We start by verifying that $(LXL^T)_{12} = B_{12}$. Step (ST4) makes sure that $W_{12} = (YL^T)_{12}$ and thus $W = YL^T$, due to the way we defined the uncomputed blocks of W. Step (ST3)

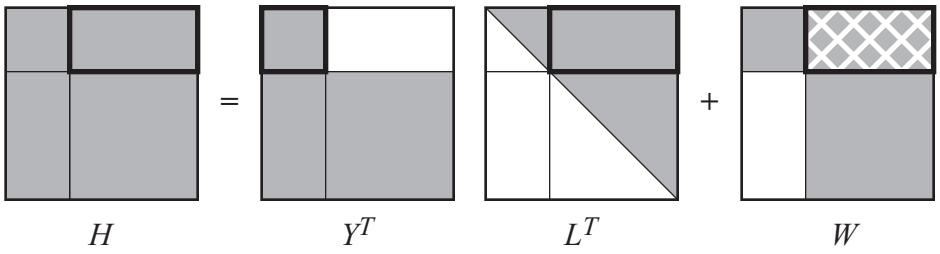

FIGURE 2.9. Computing the superdiagonal block of W in Step (ST3) by updating the corresponding block of H.

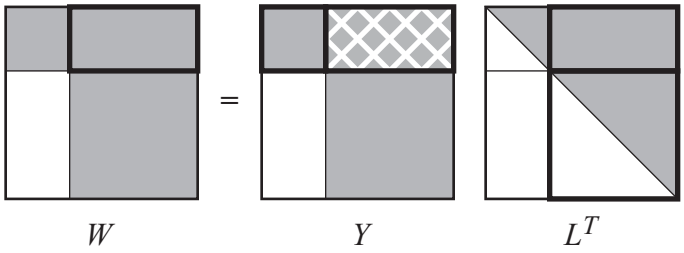

FIGURE 2.10. Computing the superdiagonal block of Y in Step (ST4) by updating the corresponding block of W and solving a triangular system.

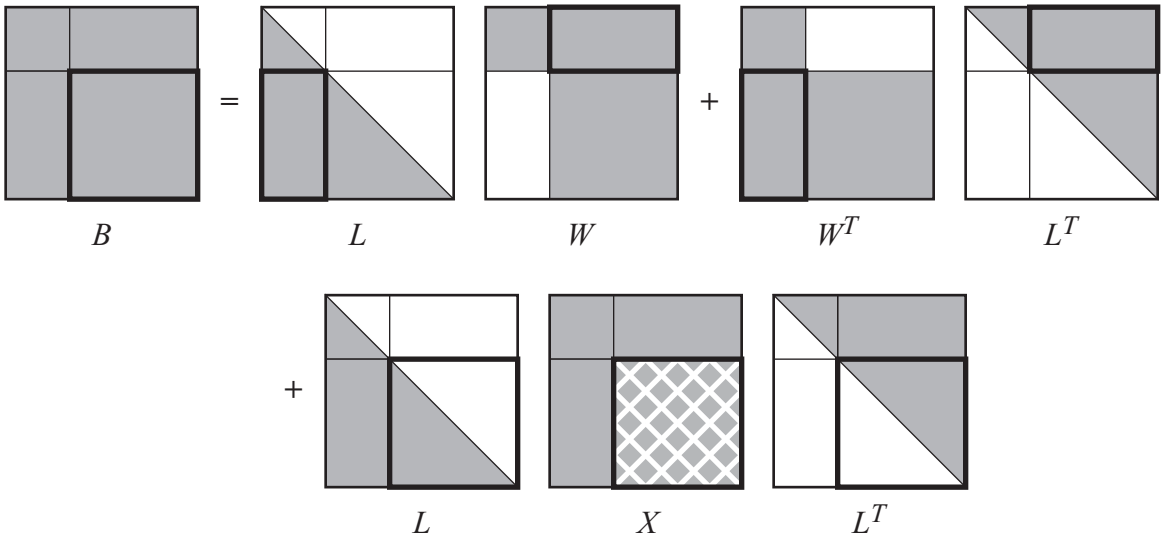

FIGURE 2.11. Computing the second diagonal block of X in Step (ST5) by updating the corresponding block of B and solving a smaller two-sided triangular system.

guarantees that $H_{12} = (Y^T L^T + W)_{12}$. Substituting $W = YL^T$ and noting that $Y^T L^T + YL^T = XL^T$ shows that $H_{12} = (XL^T)_{12}$ and thus $H = XL^T$, again due to our definition of the uncomputed blocks of H. Finally, Step (ST2) makes sure that $B_{12} = (LH)_{12}$, and substituting $H = XL^T$ shows that $B_{12} = (LXL^T)_{12}$.

Next we verify that $(LXL^T)_{22} = B_{22}$ by transforming the equation in Step (ST5):

$$\begin{aligned}
B_{22} &= L_{21}W_{12} + (L_{21}W_{12})^T + L_{22}X_{22}(L_{22})^T \\
&= L_{21}W_{12} + (L_{21}W_{12})^T + L_{22}Y_{22}(L_{22})^T + (L_{22}Y_{22}(L_{22})^T)^T \\
&= L_{21}W_{12} + (L_{21}W_{12})^T + L_{22}W_{22} + (L_{22}W_{22})^T \\
&= (LW + (LW)^T)_{22} \\
&= (LYL^T + LY^TL^T)_{22} \\
&= (L(Y + Y^T)L^T)_{22} \\
&= (LXL^T)_{22} \ .
\end{aligned}$$

Finally, Step (ST1) explicitly makes sure that $(LXL^T)_{11} = B_{11}$ and thus $LXL^T = B$.

We did not specify the dimensions of the blocks; different choices yield different algorithms. If we choose $b = 1$, 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 $(n-i)^2$ and $2(n-i)^2$ flops, respectively. Thus, the leading term in flop cost is given by

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

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

$$F_b(n) = \sum_{i=1}^{n/b-1} 3(n-ib)^2b = 3b^3\sum_{i=1}^{n/b-1} i^2 = n^3 + o(n^3) \ .$$

We can also formulate the algorithm recursively, with X_{11} being $\left\lfloor \frac{n}{2} \right\rfloor \times \left\lfloor \frac{n}{2} \right\rfloor$. 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 blocked 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 $X_{11}(L_{21})^T$ is a SYMM BLAS which costs $2(n/2)^3$. The triangular solves are TRSM calls that cost $(n/2)^3$ each, and the product $L_{21}W_{12}$ is a GEMM call that costs $2(n/2)^3$ (can subtract the transpose after computing and subtracting the product). If we store the $X_{11}(L_{21})^T$ and reuse it in both step (ST3) and step (ST4), the recurrence is

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

which again solves to

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

If we chose to recompute $X_{11}(L_{21})^T$ in order to run the algorithm in place, the flop count increases but is still $O(n^3)$.

2.3. **Pivoting.** Without pivoting, the algorithm can break down or become unstable, just like the classical elementwise Aasen algorithm. In the new algorithm, blocks $L_{J+1:N,J+1}$ and $H_{J+1,J}$ are computed using an LU factorization, and without pivoting, the factorization may fail to exist or may be unstable. Clearly, we need to pivot in Equation (AA4). It turns out that this stabilizes the algorithm, as in elementwise Aasen.

We use row pivoting, meaning that step J factors

$$L_{J+1:N,J+1}H_{J+1,J} = P_J \left(A_{J+1:N,J} - L_{J+1:N,1:J}H_{1:J,J}\right) ,$$

where P_J is a permutation matrix. From the numerical-stability point of view, we can use partial pivoting. There are several ways to compute this LU factorization in a way that ensures that the overall algorithm is communication avoiding; we list and analyze them in Section 4.2.2 below.

Once the LU factorization is computed, we also apply P_J to $L_{J+1:N,1:J}$ and to the trailing submatrix $A_{J+1:N,J+1:N}$. 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 below, in Section 4.2.3.

2.4. **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 Equation (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 Equations (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 2.1 and 2.4 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. Avoiding the recomputation of these blocks reduces the number of b-by-b matrix products required to compute W and H by a ratio of 5:3, thereby making the computation of W essentially free.

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 $b = \Theta(\sqrt{M})$, 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 $O(bn)$).

Algorithms with these properties include the unsymmetric banded LU factorization with partial pivoting, Kaufman's retraction algorithm [19], and Irony and Toledo's snap-back algorithm [18]. All three produce a factorization that is essentially banded with bandwidth $O(b)$. 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 pre-permuted 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.

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 [16, Section 2.2]. We also assume that 0.5 is a floating-point number and that

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

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 1 ([16, 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 $|\theta_n| \le \gamma_n$ then

$$\left(1+\theta_m\right)\left(1+\theta_n\right) = 1+\theta_{m+n} \ , \qquad |\theta_{m+n}| \le \gamma_{m+n} \ .$$

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 2 ([16, 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*

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

The following two lemmas also deal with matrix-matrix multiplication. Their proofs are similar to the proof of Lemma 8.4 in [16], with the only difference stemming from the possible scaling by 0.5 in Lemma 3. The assumption (FL2) in our model guarantees that this scaling has no effect on the ultimate bound.

Lemma 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*

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

Lemma 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*

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

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

Lemma 5 ([16, 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*

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

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

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

Lemma 6 ([16, 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*

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

3.2. **The Stability of the Overall Algorithm.** We now show that the block Aasen algorithm is backward stable. The analysis relies on lemmas from Section 3.1 above and on Lemma 10 which we prove in the next section (without relying on any result from this section). The reader is invited to review the statement of Lemma 10 before proceeding.

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 2.4. 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.

Theorem 7. *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 8 and 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 $n > 3b$. $\square$

Lemma 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

$$\Delta = \Delta^{(1)} + L\Delta^{(2)} . \tag{3.1}$$

Bounding Δ 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 Equation (AA4). In that equation we form the matrix $X = A_{J+1:N,J} - L_{J+1:N,1:J}H_{1:J,J}$ and then compute its LU factorization $X = L_{J+1:N,J+1}H_{J+1,J}$. Let $\Gamma^{(1)}$ and $\Gamma^{(2)}$ be such that

$$A_{J+1:N,J} = L_{J+1:N,1:J}H_{1:J,J} + X + \Gamma^{(1)} \tag{3.2}$$

$$X = L_{J+1:N,J+1}H_{J+1,J} + \Gamma^{(2)} . \tag{3.3}$$

Substituting the second expression into the first one yields

$$\begin{aligned} A_{J+1:N,J} &= L_{J+1:N,1:J}H_{1:J,J} + L_{J+1:N,J+1}H_{J+1,J} + \Gamma^{(1)} + \Gamma^{(2)} \\ &= L_{J+1:N1:J+1}H_{1:J+1,J} + \Gamma^{(1)} + \Gamma^{(2)} , \end{aligned}$$

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

$$A_{J+1:N,J} = (LH)_{J+1:N,J} + \Gamma^{(1)} + \Gamma^{(2)} ,$$

and therefore

$$\Delta^{(1)}_{J+1:N,J} = \Gamma^{(1)} + \Gamma^{(2)} . \tag{3.4}$$

We analyze the accuracy of forming X using Lemma 3, which yields the bound

$$|\Gamma^{(1)}| \leq \gamma_{Jb}(|L_{J+1:N,1:J}||H_{1:J,J}| + |X|) .$$

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

$$|\Gamma^{(1)}| \leq \gamma_{(J-1)b}(|L_{J+1:N,1:J}||H_{1:J,J}| + |X|) . \tag{3.5}$$

The accuracy of the LU factorization of X can be analyzed using Lemma 6, which yields

$$|\Gamma^{(2)}| \leq \gamma_b|L_{J+1:N,J+1}||H_{J+1,J}| . \tag{3.6}$$

Substituting (3.5) and (3.6) into (3.4) yields

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

and further substituting (3.3) and using (3.6) again yields

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

Bounding the constants in this expression according to

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

where the second bound is justified by Lemma 1, yields

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

and therefore

$$|\Delta^{(1)}_{I,J}| \le \gamma_{Jb}(|L||H|)_{I,J} \tag{3.7}$$

for all $I > J$.

We bound the diagonal and superdiagonal blocks of $\Delta^{(2)}$ by considering the computation that corresponds to Equation (AA3). In that equation we compute blocks of H by forming the corresponding blocks of TL^T, and as we discuss in Section 2.4, 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 2, which guarantees that

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

for all $I \le J$. The blocks $\Delta^{(2)}_{J+1,J}$ correspond to Equation (AA5), which solves the triangular system $H_{J+1,J} = T_{J+1,J}(L_{J,J})^T$. This is analyzed in Lemma 5, which guarantees that

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

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

$$|\Delta^{(2)}| \le \gamma_{3b}|T||L^T| \,. \tag{3.8}$$

Substituting (3.7) and (3.8) into (3.1) yields

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

for all $I > J$. Further substituting $H = TL^T + \Delta^{(2)}$ and using (3.8) once again yields

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

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

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

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

$$\Delta = \Delta^{(1)} + L\Delta^{(2)} + (L\Delta^{(2)})^T . \tag{3.9}$$

Equation (AA2) computes $T_{J,J}$ by forming $X = A_{J,J} - L_{J,1:J-1}W_{1:J-1,J} - (L_{J,1:J-1}W_{1:J-1,J})^T$ and then solving $L_{J,J}T_{J,J}(L_{J,J})^T = X$. Let $\Gamma^{(1)}$ and $\Gamma^{(2)}$ be such that

$$A_{J,J} = L_{J,1:J-1}W_{1:J-1,J} + (L_{J,1:J-1}W_{1:J-1,J})^T + X + \Gamma^{(1)} \tag{3.10}$$

$$X = L_{J,J}T_{J,J}(L_{J,J})^T + \Gamma^{(2)} . \tag{3.11}$$

Substituting (3.11) into (3.10) yields

$$A_{J,J} = L_{J,1:J-1}W_{1:J-1,J} + (L_{J,1:J-1}W_{1:J-1,J})^T + L_{J,J}T_{J,J}(L_{J,J})^T + \Gamma^{(1)} + \Gamma^{(2)} .$$

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

$$\begin{aligned} A_{J,J} &= L_{J,1:J-1}W_{1:J-1,J} + (L_{J,1:J-1}W_{1:J-1,J})^T + L_{J,J}W_{J,J} + (L_{J,J}W_{J,J})^T + \Gamma^{(1)} + \Gamma^{(2)} \\ &= L_{J,1:J}W_{1:J,J} + (L_{J,1:J}W_{1:J,J})^T + \Gamma^{(1)} + \Gamma^{(2)} \\ &= (LW + (LW)^T)_{J,J} + \Gamma^{(1)} + \Gamma^{(2)} \end{aligned}$$

and thus

$$\Delta^{(1)}_{J,J} = \Gamma^{(1)} + \Gamma^{(2)} . \tag{3.12}$$

The accuracy of forming X and then solving for $T_{J,J}$ can be bounded using Lemmas 4 and 10, which guarantee that

$$\begin{aligned} |\Gamma^{(1)}| &\leq \gamma_{2(J-2)b}(|L_{J,1:J-1}||W_{1:J-1,J}| + (|L_{J,1:J-1}||W_{1:J-1,J}|)^T + |X|) \\ |\Gamma^{(2)}| &\leq \gamma_{3b-1}|L_{J,J}||T_{J,J}||L_{J,J}|^T . \end{aligned}$$

Substituting these bounds into (3.12), further substituting (3.11) and bounding again yields

$$\begin{aligned} |\Delta^{(1)}_{J,J}| &\leq \gamma_{2(J-2)b}(|L_{J,1:J-1}||W_{1:J-1,J}| + (|L_{J,1:J-1}||W_{1:J-1,J}|)^T) \\ &\quad + (\gamma_{2(J-2)b} + \gamma_{3b-1} + \gamma_{2(J-2)b}\gamma_{3b-1})|L_{J,J}||T_{J,J}||L_{J,J}|^T \\ &\leq \gamma_{2(J-2)b}(|L_{J,1:J-1}||W_{1:J-1,J}| + (|L_{J,1:J-1}||W_{1:J-1,J}|)^T) \\ &\quad + \gamma_{2(J-2)b+3b-1}|L_{J,J}||T_{J,J}||L_{J,J}|^T , \end{aligned} \tag{3.13}$$

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

The superdiagonal blocks of $\Delta^{(2)}$ correspond to Equation (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 2.4. Because of the scaling by 0.5 we cannot apply Lemma 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

$$|\Delta^{(2)}_{I,J}| \le \gamma_{b+1}(|R||L^T|)_{I,J} \tag{3.14}$$

for all $I < J$.

Next we return to bounding (3.9), 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 (3.14) yields

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

which can be further simplified according to

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

yielding

$$(|L||\Delta^{(2)}| + (|L||\Delta^{(2)}|)^T)_{J,J} \le \gamma_{b+1}((|L||T||L^T|)_{J,J} - |L_{J,J}||T_{J,J}||L_{J,J}|^T) \ . \tag{3.15}$$

To bound the first term in (3.13) we substitute $W = RL^T + \Delta^{(2)}$ and apply the same arguments we used to produce (3.15), obtaining

$$\begin{aligned}|L_{J,1:J-1}||W_{1:J-1,J}| + (|L_{J,1:J-1}||W_{1:J-1,J}|)^T \le (1+\gamma_{b+1})((&|L||T||L^T|)_{J,J} \\ &- |L_{J,J}||T_{J,J}||L_{J,J}|^T) \ .\end{aligned} \tag{3.16}$$

Finally, substituting (3.16) into (3.13) and then substituting the result together with (3.15) into (3.9) yields

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

Because $b \ge 1$, we can bound

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

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. □

3.3. **The Stability of the Two-Sided Triangular Solver.** Our notation in this section is the same as in Section 2.2. The matrix X and the superdiagonal blocks of H and W 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 H and W such that the corresponding blocks of $H = XL^T$ and $W = YL^T$ hold.

Lemma 10. *If the two-sided triangular system $LXL^T = B$ is solved in floating-point arithmetic then*

$$LXL^T = B + \Delta \ , \qquad |\Delta| \leq \gamma_{3n-1}|L||X||L^T| \ .$$

Proof. The proof is by induction on n. In the base case we are solving a 1-by-1 system and the bound clearly holds. To make the inductive step we let the matrices $\Delta^{(t)}$ for $t = 1, 2, \ldots, 5$ be such that

$$B = LH + \Delta^{(1)} \ , \quad H = Y^TL^T + W + \Delta^{(2)} \ , \quad W = YL^T + \Delta^{(3)} \ , \quad B = LW + (LW)^T + \Delta^{(4)}$$

and

$$H = XL^T + \Delta^{(5)} \ .$$

Substituting the third formula into the second one, and then substituting the result into the first formula yields

$$\begin{aligned} H &= XL^T + \Delta^{(2)} + \Delta^{(3)} \\ B &= LXL^T + \Delta^{(1)} + L(\Delta^{(2)} + \Delta^{(3)}) \ , \end{aligned}$$

and substituting the third formula into the fourth one yields

$$B = LXL^T + L\Delta^{(3)} + (L\Delta^{(3)})^T + \Delta^{(4)} \ ,$$

and thus

$$\Delta^{(5)} = \Delta^{(2)} + \Delta^{(3)} \tag{3.17}$$

$$\Delta = \Delta^{(1)} + L\Delta^{(5)} \tag{3.18}$$

$$\Delta = L\Delta^{(3)} + (L\Delta^{(3)})^T + \Delta^{(4)} \ . \tag{3.19}$$

Applying the lemmas of rounding-error analysis and the inductive hypothesis to equations (ST1)–(ST5) allows us to derive the bounds

$$|\Delta_{11}| \leq \gamma_{3b-1}(|L||X||L|^T)_{11} \tag{3.20}$$

$$|\Delta_{12}^{(1)}| \leq \gamma_b(|L||H|)_{12} \tag{3.21}$$

$$|\Delta^{(2)}| \leq \gamma_b(|Y^T||L^T| + |W|) \tag{3.22}$$

$$|\Delta^{(3)}| \leq \gamma_n|Y||L^T| \tag{3.23}$$

$$|\Delta_{22}^{(4)}| \leq \gamma_{2b}(|L_{21}||W_{12}| + (|L_{21}||W_{12}|)^T) + \gamma_{2b+3(n-b)-1}|L_{22}||X_{22}||L_{22}|^T \ . \tag{3.24}$$

We omit the derivation because it is based on the same arguments as those we have used in the proofs of the previous lemmas.

Applying (3.22) and (3.23) to (3.17), substituting $W = YL^T + \Delta^{(3)}$ and then bounding again yields

$$|\Delta^{(5)}| \le \gamma_b |Y^T||L^T| + (\gamma_b + \gamma_n + \gamma_b\gamma_n)|Y||L^T|$$

and bounding $\gamma_b \le \gamma_{n+b}$ and $\gamma_b + \gamma_n + \gamma_b\gamma_n \le \gamma_{n+b}$ yields

$$|\Delta^{(5)}| \le \gamma_{n+b}|X||L^T| \ . \tag{3.25}$$

Substituting (3.21) and (3.25) into (3.18), further substituting $H = XL^T + \Delta^{(5)}$ and then applying (3.25) again yields

$$\begin{aligned} |\Delta_{12}| &\le (\gamma_b + \gamma_{n+b} + \gamma_b\gamma_{n+b})(|L||X||L^T|)_{12} \\ &\le \gamma_{n+2b}(|L||X||L^T|)_{12} \ . \end{aligned} \tag{3.26}$$

Next we bound Δ_{22} using (3.19), starting with the first two terms in that equation. We defined W such that $\Delta_{22}^{(3)}$ is zero and thus

$$(|L||\Delta^{(3)}| + (|L||\Delta^{(3)}|)^T)_{22} = |L_{21}||\Delta_{12}^{(3)}| + (|L_{21}||\Delta_{12}^{(3)}|)^T \ .$$

Further applying (3.23) yields

$$\begin{aligned} (|L||\Delta^{(3)}| + (|L||\Delta^{(3)}|)^T)_{22} &\le \gamma_n |L_{21}||Y_{1,:}||L_{2,:}|^T + \gamma_n(|L_{21}||Y_{1,:}||L_{2,:}|^T)^T \\ &= \gamma_n((|L||X||L^T|)_{22} - |L_{22}||X_{22}||L_{22}|^T) \ . \end{aligned} \tag{3.27}$$

Substituting $W = YL^T + \Delta^{(3)}$ in the first two terms of (3.24) and using the same argument that we used to produce (3.27) yields

$$|L_{21}||W_{12}| + (|L_{21}||W_{12}|)^T \le (1 + \gamma_n)((|L||X||L^T|)_{22} - |L_{22}||X_{22}||L_{22}|^T) \ . \tag{3.28}$$

Substituting (3.28) into (3.24), and then substituting the result together with (3.27) into (3.19) yields

$$\begin{aligned} |\Delta_{22}| \le\ & (\gamma_n + \gamma_{2b} + \gamma_{2b}\gamma_n)((|L||X||L^T|)_{22} - |L_{22}||X_{22}||L_{22}|^T) \\ & + \gamma_{2b+3(n-b)-1}|L_{22}||X_{22}||L_{22}|^T \ . \end{aligned}$$

Because $1 \le b \le n-1$,

$$\begin{aligned} \gamma_n + \gamma_{2b} + \gamma_{2b}\gamma_n \le \gamma_{n+2b} \le \gamma_{3n-2} \\ \gamma_{2b+3(n-b)-1} = \gamma_{3n-b-1} \le \gamma_{3n-2} \ , \end{aligned}$$

and thus

$$|\Delta_{22}| \le \gamma_{3n-2}(|L||X||L^T|)_{22} \ . \tag{3.29}$$

Combining bounds (3.20), (3.26) and (3.29) we see that $|\Delta_{I,J}| \le C_{I,J}(|L||X||L^T|)_{I,J}$, where

$$C = \begin{bmatrix} \gamma_{3b-1} & \gamma_{n+2b} \\ \gamma_{n+2b} & \gamma_{3n-2} \end{bmatrix} \ ,$$

and because $C_{I,J} \le \gamma_{3n-2}$ for all I and J, we conclude that $|\Delta| \le \gamma_{3n-2}(|L||X||L^T|)$. The exception to this is the case $n = 1$, which requires the larger constant γ_{3n-1} in the statement of the lemma. □

3.4. **Growth.** The stability of the factorization algorithm depends on the magnitude of L and T relative to that of A. How large can L and T 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 $|T_{ij}| \leq 4^{n-2} \max_{ij} |A_{ij}|$ [15]; the argument is essentially the same as the one that establishes the bound on $|U_{ij}|$ in Gaussian elimination with partial pivoting. Furthermore, this bound is attained by a known matrix of order $n = 3$, although larger matrices that attain the bound are not known [16, 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 [26, 30] and [16, Section 9.4].

If we compute the factorization in Equation (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 Gaussian elimination with partial pivoting, 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 Equation (AA4) is computed in a communication-avoiding way using the tall-and-skinny LU factorization [13] (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 [20] 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 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.

4.1. **Computational Cost.** Our goal in this subsection is to show that the new blocked 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 Equations (AA1)–(AA5) (the computational cost of pivoting is negligible). Letting J denote the index of the outermost loop of the algorithm and b be the block size, the arithmetic cost of Equations (AA1)–(AA3) is $O(Jb^3)$ flops. This is because each equation involves $O(J)$ block multiplications of $b \times b$ blocks (some of which are triangular). Note that in Equation (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, Equation (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 Equation (AA4), which consists of two sub-computations: 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 $(N-J)b \times Jb$ submatrix of L by a $Jb \times b$ 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 Equations (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 Equations (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, $b \times b$ 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 $b \leq \sqrt{M/3}$ so that three blocks fit simultaneously in fast memory.

4.2.1. *Block Operations.* By excluding the LU decomposition, all the other computations in Equations (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 Equation (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 $b \times b$ 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 re-use 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 re-use (though the algorithms need not be asymptotically optimal).

We choose to use the recursive algorithm (RLU) of [14, 29] for panel factorizations, updated slightly to match the block-contiguous data layout. The algorithm works by splitting the

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

Algorithm	Words	Messages
RLU	$O\left(\frac{nb^2}{\sqrt{M}} + nb \log b\right)$	$O\left(\min\left(n, \frac{n^2}{M}\right)\right)$
SMLU [6]	$O\left(\frac{nb^2}{\sqrt{M}} + nb \log b \log \frac{nb}{M}\right)$	$O\left(\frac{nb^2}{M^{3/2}} + \frac{nb}{M} \log b \log \frac{nb}{M}\right)$
TSLU [13]	$O\left(\frac{nb^2}{\sqrt{M}}\right)$	$O\left(\frac{nb^2}{M^{3/2}}\right)$

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 by block. The bandwidth cost of this algorithm for $n \times b$ matrices is analyzed in [29], 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 sub-panel 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 $n \times b$ matrix. The latency cost becomes the minimum of two terms: $O(n)$ or $O(n^2/M)$. The bandwidth and latency costs are summarized in the first 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 $n \times b$ factorization by N, the number of panel factorizations. Using the RLU algorithm, this yields a bandwidth cost of $O(n^2 b/\sqrt{M} + n^2 \log b)$ 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) [6] 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 [6], and the communication costs are given in the second 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 tall-skinny LU algorithm (TSLU) [13]. 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 [13], and the communication costs are given in the third 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

TABLE 2. Communication costs of symmetric pivoting schemes

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

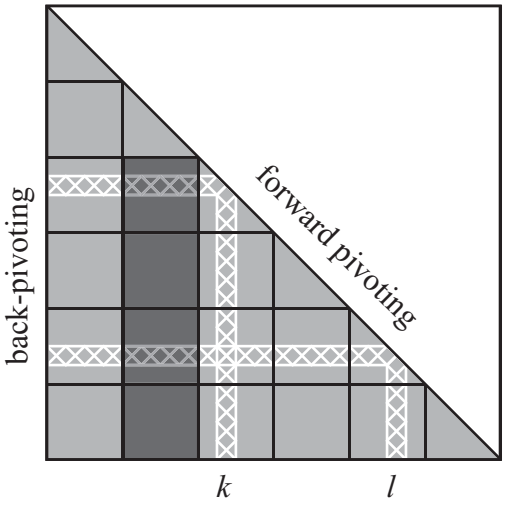

FIGURE 4.1. Exchanging rows and columns k and l. The second (dark) block column is the block column of the reduced matrix whose LU factorization was just computed. The first block column is a block column of L; the algorithm applies back pivoting to it. Block columns 3 to 6 are part of the trailing submatrix; 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.

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 4.1. For example, applying the transposition (k, l) implies that the L-shaped set of elements in the k^{th} row and k^{th} column (to the left and below the diagonal) is swapped with the L-shaped set of elements in the l^{th} row and l^{th} 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 non-contiguously, 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

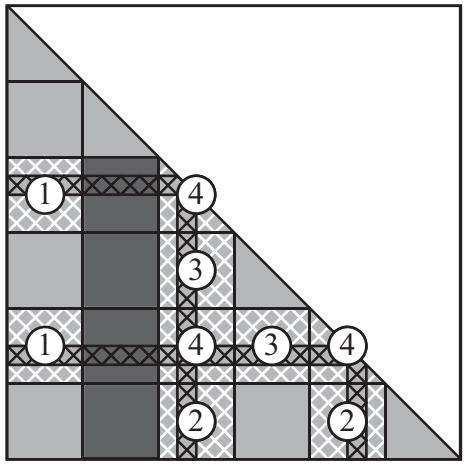

FIGURE 4.2. Exchanging a pair of rows in the blocked approach. Numbers indicate sets of blocks that are held simultaneously in fast memory: all the blocks marked "1" are held in fast memory simultaneously, later all the blocks marked "2", and so on.

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 [13].

In order to reduce the latency cost, we use a blocked approach which will require greater bandwidth cost than the direct approach but will not increase the asymptotic bandwidth cost of the overall algorithm. The blocked approach accesses contiguous $b \times b$ blocks, but it may permute only a few rows or columns of the blocks. This approach is the symmetric analogue of Variant 2 in [13].

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 4.2. 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 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 4.2. 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.

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 [5]. 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 this lower bound formally:

Theorem 11. *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: To derive the lower bound, we use Theorem 2.2 in Ballard et al. [5]. The products $l_{ik}h_{k,j-b}$ constitute the g_{ijk} operations in that theorem, and the summations that are subtracted from a_{ij} constitute the f_{ij} functions. To use the theorem, we need to count the overall number of g_{ijk} multiplications, which is

$$\sum_{i=b+1}^{n} \sum_{j=b+1}^{i-1} \sum_{k=b+1}^{j-b} 1 = \frac{1}{6} n^3 + O(n^2 b) \ .$$

To apply the theorem, none of the quantities is allowed to be computed and discarded without being written to slow memory. The output of our algorithm does not need to output H and therefore it may compute and discard elements of H, but for the sake of the proof, we will assume that the algorithm also outputs H; we will later argue that this assumption does not change the asymptotic lower bound. In the language of the theorem, this means that none of the quantities is R2/D2.

These assumptions, together with Theorem 2.2 in Ballard et al., imply that the algorithm must transfer

$$\frac{G}{8\sqrt{M}} - M \ge \frac{n^3}{48\sqrt{M}} - M$$

words between slow and fast memory. Therefore, the number of cache misses must be at least $n^3/48\sqrt{M} - 1$. An algorithm that does not output H can perform less communication, but the difference is at most $\frac{1}{2}n^2 + nb$ words, an upper bound on the number of elements in H.

6. Numerical Experiments

Next we describe a set of numerical experiments that provide further insight into the numerical behavior of the algorithm. We used a block size $b = 16$ in all the experiments.

We carried out two sets of experiments: one set involving random matrices and another involving matrices from the University of Florida Sparse Matrix Collection [10]. 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 $100 \leq n \leq 5{,}000$. The elements of these matrices are distributed normally and independently (preserving symmetry, of course) with mean 0 and standard deviation 1. In all of our experiments we used GEPP for panel factorizations. Experiments using the tall-and-skinny communication-avoiding LU (TSLU) algorithm will be described in the journal version of this report. For each matrix we measured three parameters: the growth factor, the backward error of the factorization, and the backward error in the solution of a linear system of equations $Ax = b$, where b is the sum of the columns of A (so x is the vector of all ones).

We define the growth factor as the number

$$\frac{\left\| |L|\,|T|\,|L|^T \right\|_\infty}{\|A\|_\infty},$$

a definition that is justified by Theorem 7. The factorization error is defined as

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

using the convention $0/0 = 0$. We compute the backward error of the floating-point solution $\hat{x}$ to the system $Ax = b$ according to

$$\frac{\|A\hat{x} - b\|_\infty}{\|A\|_\infty \|\hat{x}\|_\infty + \|b\|_\infty}.$$

The factorizations of random matrices were completely backward stable, with backward errors between $1.1u$ and $2.4u$, with a median of $1.9u$. The stability of solutions to linear systems and the growth factors are shown in Figure 6.1. The backward errors are moderate, varying between 1.7×10^{-15} and 1.7×10^{-14}. The backward error is increasing with n but at a rate that is clearly slower than linear. The growth factor is strongly correlated with the error, which is consistent with the bound in Theorem 7. The error does not seem to depend on n outside of the implicit dependence through the growth factor, in contrast with the bound in Theorem 7.

The second set of experiments factored 180 matrices from the University of Florida Sparse Matrix Collection. We chose for this experiment *all* of the symmetric, real, non-pattern matrices of order $64 \leq n \leq 8{,}192$, 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 180 matrices is further described in Table 3. The experiment was conducted according to the same scheme as the experiment involving the random matrices.

The algorithm experienced difficulties on 14 of the matrices; they are discussed later in this section.

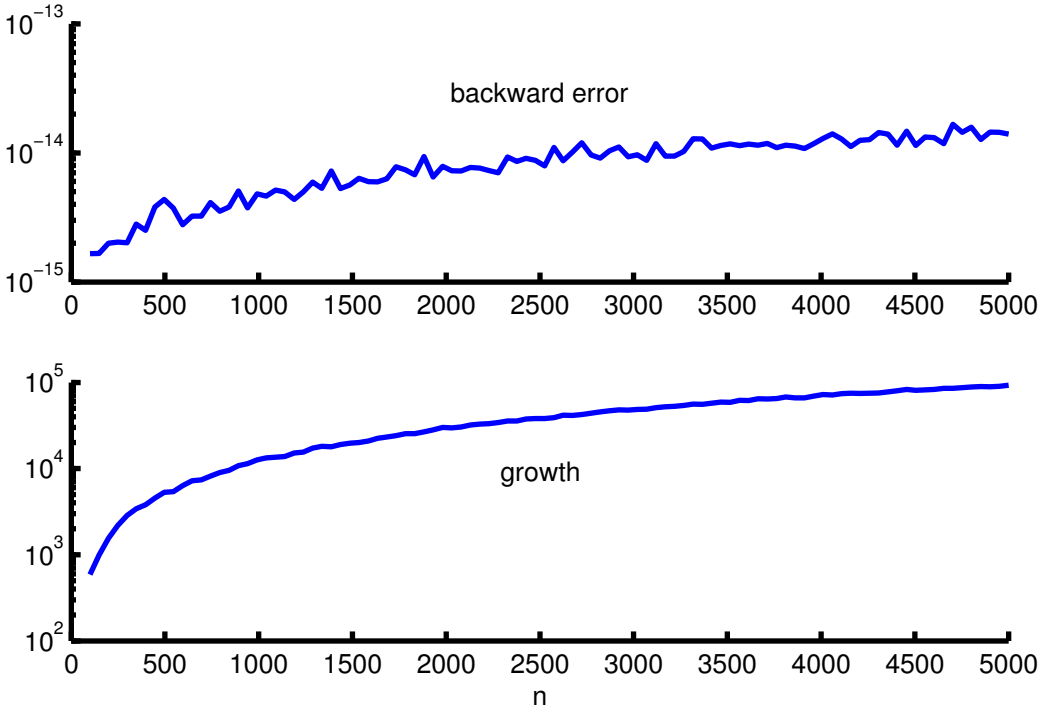

FIGURE 6.1. Backward errors in the solution of $Ax = b$ and the growth factors in the factorization of random matrices.

TABLE 3. Statistics of the University of Florida matrix set.

	n	κ	nnz/n
minimum	64	5.0×10^{0}	0.46
1st quartile	800	8.4×10^{3}	6.81
median	2,000	2.5×10^{6}	12.94
3rd quartile	4,581	3.2×10^{10}	23.93
maximum	8,140	`inf`	378.19

On the 166 matrices on which the algorithm produced good results, we obtained stable factorizations with backward errors of less than $11u$. The stability of the linear solver and the growth are shown in Figure 6.2. The linear-solver backward errors are in the interval $[1.8 \times 10^{-18},\ 2.0 \times 10^{-13}]$ with a median of 2.1×10^{-15}.

On 14 matrices, the linear solver that we used to solve banded systems involving T failed to produce a solution. For solving such systems we use the LAPACK subroutine `GBSV`, which is a banded implementation of GEPP. The source of the problem is that when T is rank deficient, `GBSV` produces a U factor with zeros on the diagonal, and this factor cannot be used to solve linear systems. In all 14 matrices the root cause was structural or numerical rank deficiency of A (Matlab reported condition numbers larger than 10^{20}). Our factorization

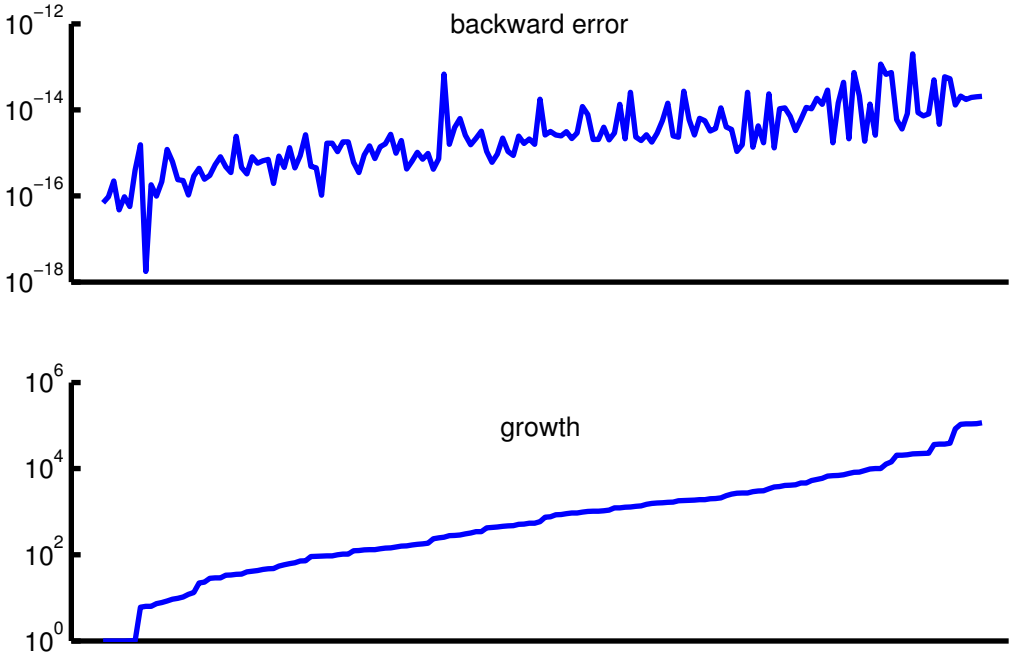

FIGURE 6.2. Backward errors and growth factors on matrices from the University of Florida Collection. The matrices are ordered by growth.

algorithm produced stable factorizations, with backward errors of order u, well conditioned L's, and mild growth (up to 1.2×10^6).

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 showed that the algorithm performs well in practice on a multi-core 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.

Acknowledgements. This research was supported in part by NSF CCF-1117062 and CNS-0905188, 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," grant 1045/09 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities), and grant 2010231 from the US-Israel Bi-National Science Foundation, and by Microsoft (Award #024263) and Intel (Award #024894) funding, and matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung. Research is also supported by DE-SC0004938, DE-SC0005136, DE-SC0003959, DE-SC0008700, and AC02- 05CH11231, and DARPA grant HR0011-12-2-0016.

References

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

[2] Saman Amarasinghe, Mary Hall, Richard Lethin, Keshav Pingali, Dan Quinlan, Vivek Sarkar, John Shalf, Robert Lucas, Katherine Yelick, Pavan Balaji, Pedro C. Diniz, Alice Koniges, Marc Snir, and Sonia R. Sachs. Exascale programming challenges. Report of the 2011 Workshop on Exascale Programming Challenges, Marina del Rey, July 27–29, 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*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

[4] Grey Ballard, Dulceneia Becker, James Demmel, Jack Dongarra, Alex Druinsky, Inon Peled, Oded Schwartz, Sivan Toledo, and Ichitaro Yamazaki. Implementing a blocked Aasen's algorithm with a dynamic scheduler on multicore architectures. In *Parallel Distributed Processing Symposium (IPDPS), 2013 IEEE 27th International*, May.

[5] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32:866–901, 2011.

[6] Grey Ballard, James Demmel, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. Communication efficient Gaussian elimination with partial pivoting using a shape morphing data layout. In *25th ACM Symposium on Parallelism in Algorithms and Architectures*, 2013.

[7] 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, 1997.

[8] James R. Bunch and Linda Kaufman. Some stable methods for calculating inertia and solving symmetric linear systems. *Mathematics of Computation*, 31:163–179, January 1977.

[9] David E. Culler, Richard M. Karp, David Patterson, Abhijit Sahay, Eunice E. Santos, Klaus Erik Schauser, Ramesh Subramonian, and Thorsten von Eicken. LogP: a practical model of parallel computation. *Communications of the ACM*, 39:78–85, November 1996.

[10] Timothy A. Davis and Yifan Hu. The University of Florida sparse matrix collection. *ACM Transactions on Mathematical Software*, 38:1:1–1:25, 2011.

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

[12] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science (FOCS '99)*, New York, October 1999.

[13] Laura Grigori, James Demmel, and Hua Xiang. CALU: A communication optimal LU factorization algorithm. *SIAM Journal on Matrix Analysis and Applications*, 32:1317–1350, 2011.

[14] F. G. Gustavson. Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM Journal of Research and Development*, 41:737–755, 1997.

[15] Nicholas J. Higham. Notes on accuracy and stability of algorithms in numerical linear algebra. In Mark Ainsworth, Jeremy Levesley, and Marco Marletta, editors, *The Graduate Student's Guide to Numerical Analysis*, pages 47–82. Springer Berlin Heidelberg, 1999.

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

[17] Jia-Wei 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*, pages 326–333, 1981.

[18] Dror Irony and Sivan Toledo. The snap-back pivoting method for symmetric indefinite matrices. *SIAM Journal on Matrix Analysis and Applications*, 28:398–424, 2006.

[19] Linda Kaufman. The retraction algorithm for factoring banded symmetric matrices. *Numerical Linear Algebra with Applications*, 14:237–254, 2007.

[20] Amal Khabou, James W. Demmel, Laura Grigori, and Ming Gu. LU factorization with panel rank revealing pivoting and its communication avoiding version. arXiv:1208.2451 [cs.NA], August 2012.

[21] R. S. Martin and J. H. Wilkinson. Reduction of the symmetric eigenproblem $Ax = \lambda Bx$ and related problems to standard form. *Numerische Mathematik*, 11:99–110, 1968.

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

[23] Jack Poulson, Bryan Marker, Robert A. van de Geijn, Jeff R. Hammond, and Nichols A. Romero. Elemental: A new framework for distributed memory dense matrix computations. *ACM Transactions on Mathematical Software*, 39:13:1–13:24, February 2013.

[24] Jack Poulson, Robert A. van de Geijn, and Jeffrey Benninghof. (Parallel) algorithms for two-sided triangular solves and matrix multiplication. Unpublished, 2012.

[25] Miroslav Rozloznik, Gil Shklarski, and Sivan Toledo. Partitioned triangular tridiagonalization. *ACM Transactions on Mathematical Software*, 37:38:1–38:16, February 2011.

[26] Arvind Sankar, Daniel A. Spielman, and Shang-Hua Teng. Smoothed analysis of the condition numbers and growth factors of matrices. *SIAM Journal on Matrix Analysis and Applications*, 28:446–476, 2006.

[27] Mark P. Sears, Ken Stanley, and Greg Henry. Application of a high performance parallel eigensolver to electronic structure calculations. In *Proceedings of the IEEE/ACM Conference on Supercomputing*, nov 1998. on CD-ROM.

[28] 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*. Springer-Verlag, 2nd edition, 1976.

[29] Sivan Toledo. Locality of reference in LU decomposition with partial pivoting. *SIAM Journal on Matrix Analysis and Applications*, 18:1065–1081, 1997.

[30] Lloyd N. Trefethen and Robert S. Schreiber. Average-case stability of Gaussian elimination. *SIAM Journal on Matrix Analysis and Applications*, 11:335–360, 1990.

[31] Leslie G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33:103–111, August 1990.

[32] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Optimal disk I/O with parallel block transfer. In *Proceedings of the twenty-second annual ACM symposium on theory of computing (STOC)*, pages 159–169, 1990.

[33] Jeffrey Scott Vitter and Elizabeth A. M. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12:110–147, 1994.