SIAM J. MATRIX ANAL. APPL.
Vol. 32, No. 4, pp. 1317–1350

CALU: A COMMUNICATION OPTIMAL LU FACTORIZATION ALGORITHM*

LAURA GRIGORI†, JAMES W. DEMMEL‡, AND HUA XIANG§

Abstract. Since the cost of communication (moving data) greatly exceeds the cost of doing arithmetic on current and future computing platforms, we are motivated to devise algorithms that communicate as little as possible, even if they do slightly more arithmetic, and as long as they still get the right answer. This paper is about getting the right answer for such an algorithm. It discusses CALU, a communication avoiding LU factorization algorithm based on a new pivoting strategy, that we refer to as tournament pivoting. The reason to consider CALU is that it does an optimal amount of communication, and asymptotically less than Gaussian elimination with partial pivoting (GEPP), and so will be much faster on platforms where communication is expensive, as shown in previous work. We show that the Schur complement obtained after each step of performing CALU on a matrix A is the same as the Schur complement obtained after performing GEPP on a larger matrix whose entries are the same as the entries of A (sometimes slightly perturbed) and zeros. More generally, the entire CALU process is equivalent to GEPP on a large, but very sparse matrix, formed by entries of A and zeros. Hence we expect that CALU will behave as GEPP and it will also be very stable in practice. In addition, extensive experiments on random matrices and a set of special matrices show that CALU is stable in practice. The upper bound on the growth factor of CALU is worse than that of GEPP. However, there are Wilkinson-like matrices for which GEPP has exponential growth factor, but not CALU, and vice-versa.

Key words. LU factorization, communication optimal algorithm, numerical stability

AMS subject classifications. 65F50, 65F05, 68R10

DOI. 10.1137/100788926

1. Introduction.

In this paper we discuss CALU, a communication avoiding LU factorization algorithm. The main part of the paper focuses on showing that CALU is stable in practice. We also show that CALU minimizes communication. For this, we use lower bounds on communication for dense LU factorization that were introduced in [6]. These bounds were obtained by showing through reduction that lower bounds on dense matrix multiplication [17, 18] represent lower bounds for dense LU factorization as well (a more general proof can be found in [2]). These bounds show that a sequential algorithm that computes the LU factorization of a dense $n \times n$ matrix transfers between slow and fast memory at least $\Omega(n^3/W^{1/2})$ words and $\Omega(n^3/W^{3/2})$ messages, where W denotes the fast memory size and we assume that a message consists of at most W words in consecutive memory locations. On a parallel machine

*Received by the editors March 16, 2010; accepted for publication (in revised form) by S. C. Eisenstat September 27, 2011; published electronically November 10, 2011.
http://www.siam.org/journals/simax/32-4/78892.html

†INRIA Saclay - Ile de France, Laboratoire de Recherche en Informatique, Université Paris-Sud 11, 91405 Orsay, France (laura.grigori@inria.fr). This author's work was supported in part by French National Research Agency (ANR) through COSINUS program (projects PETAL ANR-08-COSI-009 and PETALH ANR-10-COSI-013).

‡Computer Science Division and Mathematics Department, UC Berkeley, Berkeley, CA 94720-1776 (demmel@cs.berkeley.edu). This author's research was supported by Microsoft (award 024263) and Intel (award 024894) funding and by matching funding by U.C. Discovery (award DIG07-10227), as well as U.S. Department of Energy grants DE-SC0003959, DE-SC0004938, and DE-FC02-06-ER25786, as well as Lawrence Berkeley National Laboratory contract DE-AC02-05CH11231.

§School of Mathematics and Statistics, Wuhan University, Wuhan 430072, People's Republic of China (hxiang@whu.edu.cn). This author's research was partly supported by the National Natural Science Foundation of China under grant 10901125.

with P processors, if we consider that the local memory size used on each processor is on the order of n^2/P, it results from the previous bounds that a lower bound on the number of words is $\Omega(n^2/\sqrt{P})$ and a lower bound on the number of messages is $\Omega(\sqrt{P})$. Here we consider square matrices, but later we consider the more general case of an $m \times n$ matrix.

Gaussian elimination with partial pivoting (GEPP) is one of the most stable algorithms for solving a linear system through LU factorization. At each step of the algorithm, the maximum element in each column of L is permuted in diagonal position and used as a pivot. Efficient implementations of this algorithm exist for sequential and parallel machines. In the sequential case, the DGETRF routine in LAPACK [1] implements a block GEPP factorization. The algorithm iterates over block columns (panels). At each step, the LU factorization with partial pivoting of the current panel is computed, a block row of U is determined, and the trailing matrix is updated. Another efficient implementation is recursive GEPP [24, 13]. We will see later in this paper that DGETRF minimizes neither the number of words nor the number of messages in some cases. Recursive GEPP attains the lower bound on the number of words but not the lower bound on the number of messages in general. In the parallel case, the PDGETRF routine in ScaLAPACK [5] distributes the input matrix over processors using a block cyclic layout. With this partition, every column is distributed over several processors. Finding the maximum element in a column of L necessary for partial pivoting incurs one reduction operation among processors. This gives an overall number of messages at least equal to the number of columns of the matrix. Hence this algorithm cannot attain the lower bound on the number of messages of $\Omega(\sqrt{P})$ and is larger by a factor of at least $n/\sqrt{P}$.

CALU uses a new strategy that we refer to as tournament pivoting. This strategy has the property that the communication for computing the panel factorization does not depend on the number of columns. It depends only on the number of blocks in the sequential case and on the number of processors in the parallel case. The panel factorization is performed as follows. A preprocessing step aims at finding at low communication cost b rows that can be used as pivots to factor the entire panel, where b is the panel width. Then the b rows are permuted into the first positions and the LU factorization with no pivoting of the entire panel is performed. The preprocessing step is performed as a reduction operation where the reduction operator is the selection of b pivot rows using GEPP at each node of the reduction tree. The reduction tree is selected depending on the underlying architecture. In this paper we study in particular the binary-tree-based and the flat-tree-based CALU. It has been shown in [11], where the algorithm was presented for the first time, that the binary-tree-based CALU leads to important speedups in practice over ScaLAPACK on distributed memory computers. In [8] the algorithm is adapted to multicore architectures and is shown to lead to speedups for matrices with many more rows than columns.

The main part of this paper focuses on the stability of CALU. First, we show that the Schur complement obtained after each step of performing CALU on a matrix A is the same as the Schur complement obtained after performing GEPP on a larger matrix whose entries are the same as the entries of A (plus some randomly generated ϵ entries) and zeros. More generally, the entire CALU process is equivalent to GEPP on a large, but very sparse, matrix formed by entries of A (sometimes slightly perturbed) and zeros. Hence we expect that CALU will behave as GEPP and it will also be very stable in practice. However, for CALU the upper bound on the growth factor is worse than for GEPP. The growth factor plays an important role in the backward error analysis of Gaussian elimination. It is computed using the values of the elements of A

during the elimination process, $g_W = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{i,j} |a_{ij}|}$, where $[a_{ij}^{(k)}]$ is the matrix obtained at the kth step of elimination. For GEPP the upper bound of the growth factor is 2^{n-1}, while for CALU is on the order of 2^{nH}, where n is the number of columns of the input matrix and H is the depth of the reduction tree.

For GEPP the upper bound is attained on a small set of input matrices that are variations of one particular matrix, the Wilkinson matrix. We also show that there are very sparse matrices, formed by Kronecker products involving the Wilkinson matrix, that nearly attain the bound for GEPP. We were not able to find matrices for which CALU exceeds GEPP's upper bound and we conjecture that the growth factor of CALU is also bounded by 2^{n-1}. In addition, there are Wilkinson-like matrices for which CALU is stable and GEPP has an exponential growth factor and vice-versa.

Second, we present experimental results for random matrices and for a set of special matrices, including sparse matrices, for the binary-tree-based and the flat-tree-based CALU. We discuss both the stability of the LU factorization and the linear solver, in terms of growth factor and backward errors. The results show that in practice CALU is stable. Later in this paper, Tables A.1 through A.5 present the backward errors measured three ways: by $\|PA - LU\|/\|A\|$, by the normwise backward error $\|Ax - b\|/(\|A\|\|x\| + \|b\|)$, and by the componentwise backward error (after iterative refinement in working precision). Figure 3.3 shows the ratios of these errors, dividing backward errors of CALU by GEPP. For random matrices, all CALU's backward errors are at most 1.9x larger than GEPP's backward errors. We also test "special" matrices, including known difficult examples: (1) The ratios of $\|PA - LU\|/\|A\|$ are at most 1 in over 69% of cases (i.e., CALU is at least as stable as GEPP), and always 1.5 or smaller, except for one ratio of 4.3, in which case both backward errors are much smaller than 2^{-53} = machine epsilon. (2) The ratios of normwise backward errors are at most 1 in over 53% of cases, and always 1.5 or smaller, except for 5 ratios ranging up to 26, in which cases all backward errors are less than 4x machine epsilon. (3) The ratios of componentwise backward errors are at most 1 in over 52% of cases, and always 3.2 or smaller, except for one ratio of 8.3.

We also discuss the stability of block versions of pairwise pivoting [3, 23] and parallel pivoting [26], two different pivoting schemes. These methods are of interest since, with an optimal layout, block pairwise pivoting is communication optimal in a sequential environment and block parallel pivoting is communication optimal in a parallel environment. Block pairwise pivoting has been introduced and used in the context of out-of-core algorithms [3, 28, 19], updated factorizations [20], and multicore architectures [4, 21]. It is simple to see that block parallel pivoting is unstable. As the number of blocks per panel increases (determined by the number of processors), so does the growth factor. In the extreme case when the block size is equal to 1, the growth factor increases exponentially with dimension on random examples. For pairwise pivoting we study the growth factor for the case when the block size is equal to 1. This method is more stable, but it shows a growth more than linear of the factor with respect to the matrix size. Hence a more thorough analysis for larger matrices is necessary to understand the stability of pairwise pivoting. We note another approach used for GPUs [25], which first randomizes the input matrix such that the probability of encountering a small pivot is small, and then performs its LU factorization with no pivoting.

QR factorization is another stable method for computing the solution of linear systems; however, it performs twice as many floating point operations as LU factorization in the case of dense matrices (this factor can be larger in the case of sparse

matrices). For dense matrices, communication avoiding QR (CAQR) [6] is a family of algorithms that have some similarities with CALU. CAQR computes the panel factorization as a reduction operation as in CALU. But the reduction operator is the QR factorization, which is performed at each node of the tree on matrices formed by R factors previously computed. This is different from CALU which operates on rows of the original matrix. In other words, panel factorization in CAQR does not include a preprocessing step that needs to be completed before starting to update the trailing matrix. With an optimal layout of the input matrix, CAQR attains the lower bounds on communication for sequential and parallel machines. It has the same communication cost as CALU, modulo polylogarithmic or constant factors. In a situation where communication costs completely dominate the difference in flop costs between CALU and CAQR, it is conceivable that CAQR could be the faster way to solve $Ax = b$ (as well as guaranteeing backward stability).

This paper is organized as follows. Section 2 presents the algebra of CALU and the new tournament pivoting scheme. Section 3 discusses the stability of CALU. It describes similarities between GEPP and CALU and upper bounds of the growth factor of CALU. It also presents experimental results for random matrices and several special matrices showing that CALU is stable in practice. Section 4 discusses two alternative approaches for solving linear systems via LU-like factorization. Section 5 presents parallel and sequential CALU algorithms and their performance models. Section 6 recalls lower bounds on communication and shows that CALU attains them. Section 7 concludes the paper.

2. CALU matrix algebra. In this section we describe the main steps of the CALU algorithm for computing the LU factorization of a matrix A of size $m \times n$. CALU uses a new pivoting strategy, which we refer to as tournament pivoting. (In [11] we referred to this strategy as ca-pivoting.) Here is our notation. We refer to the submatrix of A formed by rows i through j and columns d through e as $A(i\colon j, d\colon e)$. If A is the result of the multiplication of two matrices B and C, we refer to the submatrix of A as $(BC)(i\colon j, d\colon e)$. The matrix $[B;C]$ is the matrix obtained by stacking the matrices B and C atop one another.

CALU is a block algorithm that factorizes the input matrix by traversing iteratively blocks of columns. At the first iteration, the matrix A is partitioned as follows:

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix},$$

where A_{11} is of size $b \times b$, A_{21} is of size $(m-b) \times b$, A_{12} is of size $b \times (n-b)$, and A_{22} is of size $(m-b) \times (n-b)$. We present a right looking version of the algorithm, in which first the LU factorization of the first block-column (panel) is computed, then the block U_{12} is determined, and the trailing matrix A_{22} is updated. The algorithm continues on the block A_{22}.

The LU factorization of each panel is computed using tournament pivoting, and this is the main difference between CALU and other block algorithms. The panel can be seen as a tall and skinny matrix, and so we refer to its factorization as TSLU. It is performed in two steps. The first step is a preprocessing step, which identifies at low communication cost a set of good pivot rows. These rows are used as pivots in the second step for the LU factorization of the entire panel. That is, in the second step the b pivot rows are permuted into the first b positions of the panel (maintaining the order determined by the first step), and the LU factorization with no pivoting of the panel is performed.

We illustrate tournament pivoting on the factorization of the first panel. CALU considers that the panel is partitioned in P block-rows. We present here the simple case $P = 4$, and we suppose that m is a multiple of 4. The preprocessing step is performed as a reduction operation, where GEPP is the operator used to select new pivot rows at each node of the reduction tree. We now use a binary reduction tree to exemplify tournament pivoting. We number its levels starting with 0 at the leaves.

The preprocessing starts by performing GEPP of each block-row A_i. This corresponds to the reductions performed at the leaves of the binary tree (the right subscript 0 refers to the level in the reduction tree):

$$\begin{aligned} A(:, 1:b) &= \begin{bmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \end{bmatrix} = \begin{bmatrix} \bar{\Pi}_{00}\bar{L}_{00}\bar{U}_{00} \\ \bar{\Pi}_{10}\bar{L}_{10}\bar{U}_{10} \\ \bar{\Pi}_{20}\bar{L}_{20}\bar{U}_{20} \\ \bar{\Pi}_{30}\bar{L}_{30}\bar{U}_{30} \end{bmatrix} \\ &= \begin{bmatrix} \bar{\Pi}_{00} & & & \\ & \bar{\Pi}_{10} & & \\ & & \bar{\Pi}_{20} & \\ & & & \bar{\Pi}_{30} \end{bmatrix} \cdot \begin{bmatrix} \bar{L}_{00} & & & \\ & \bar{L}_{10} & & \\ & & \bar{L}_{20} & \\ & & & \bar{L}_{30} \end{bmatrix} \cdot \begin{bmatrix} \bar{U}_{00} \\ \bar{U}_{10} \\ \bar{U}_{20} \\ \bar{U}_{30} \end{bmatrix} \\ &\equiv \bar{\Pi}_0\bar{L}_0\bar{U}_0. \end{aligned}$$

In this decomposition, the first factor $\bar{\Pi}_0$ is an $m \times m$ block diagonal matrix, where each diagonal block $\bar{\Pi}_{i0}$ is a permutation matrix. The second factor, $\bar{L}_0$, is an $m \times Pb$ block diagonal matrix, where each diagonal block $\bar{L}_{i0}$ is an $m/P \times b$ lower unit trapezoidal matrix. The third factor, $\bar{U}_0$, is a $Pb \times b$ matrix, where each block $\bar{U}_{i0}$ is a $b \times b$ upper triangular factor. This step has identified P sets of local pivot rows. These rows are linearly independent and they correspond to the first b rows (or less if the block was singular) of $\bar{\Pi}_{i0}^T A_i$, with $i = 0, \ldots, 3$. The global pivot rows are obtained from the P sets of local pivot rows by performing a binary tree (of depth $\log_2 P = 2$ in our example) of GEPP factorizations of matrices of size $2b \times b$. At the first level of our depth-2 binary tree, 2 sets of pivot rows are obtained by performing 2 GEPP factorizations. The decompositions, combined here in one matrix, lead to a $Pb \times Pb$ permutation matrix $\bar{\Pi}_1$, a $Pb \times 2b$ factor $\bar{L}_1$, and a $2b \times b$ factor $\bar{U}_1$:

$$\begin{aligned} \begin{bmatrix} \left(\bar{\Pi}_0^T A\right)(1:b, 1:b) \\ \left(\bar{\Pi}_0^T A\right)(m/P+1:m/P+b, 1:b) \\ \hline \left(\bar{\Pi}_0^T A\right)(2m/P+1:2m/P+b, 1:b) \\ \left(\bar{\Pi}_0^T A\right)(3m/P+1:3m/P+b, 1:b) \end{bmatrix} &= \begin{bmatrix} \bar{\Pi}_{01}\bar{L}_{01}\bar{U}_{01} \\ \bar{\Pi}_{11}\bar{L}_{11}\bar{U}_{11} \end{bmatrix} \\ &= \begin{bmatrix} \bar{\Pi}_{01} & \\ & \bar{\Pi}_{11} \end{bmatrix} \cdot \begin{bmatrix} \bar{L}_{01} & \\ & \bar{L}_{11} \end{bmatrix} \cdot \begin{bmatrix} \bar{U}_{01} \\ \bar{U}_{11} \end{bmatrix} \\ &\equiv \bar{\Pi}_1\bar{L}_1\bar{U}_1. \end{aligned}$$

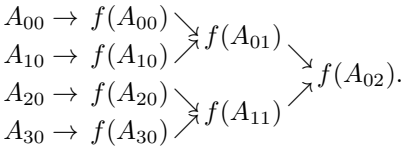
At the root of our depth-2 binary tree, the global pivot rows are obtained by applying GEPP on the two sets of pivot rows identified at level 1. This is shown in the following equation, where we consider that $\bar{\Pi}_1$ is extended by the appropriate identity matrices to the dimension of $\bar{\Pi}_0$:

$$\begin{bmatrix} \left(\bar{\Pi}_1^T\bar{\Pi}_0^T A\right)(1:b, 1:b) \\ \left(\bar{\Pi}_1^T\bar{\Pi}_0^T A\right)(2m/P+1:2m/P+b, 1:b) \end{bmatrix} = \bar{\Pi}_{02}\bar{L}_{02}\bar{U}_{02} \equiv \bar{\Pi}_2\bar{L}_2\bar{U}_2.$$

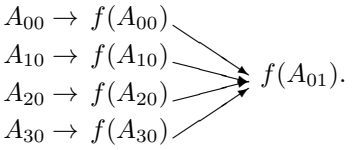
We consider again by abuse of notation that $\bar{\Pi}_2$ is extended by the appropriate identity matrices to the dimension of $\bar{\Pi}_0$. The global pivot rows are permuted to the diagonal positions by applying the permutations identified in the preprocessing step to the original matrix A. The LU factorization with no pivoting of the first panel is performed. Note that $U_{11} = \bar{U}_2$. Then the block-row of U is computed and the trailing matrix is updated. The factorization continues on the trailing matrix $\bar{A}$. This is shown in the following equation:

$$\bar{\Pi}_2^T \bar{\Pi}_1^T \bar{\Pi}_0^T A = \begin{bmatrix} L_{11} & \\ L_{21} & I_{n-b} \end{bmatrix} \cdot \begin{bmatrix} I_b & \\ & \bar{A} \end{bmatrix} \cdot \begin{bmatrix} U_{11} & U_{12} \\ & U_{22} \end{bmatrix}.$$

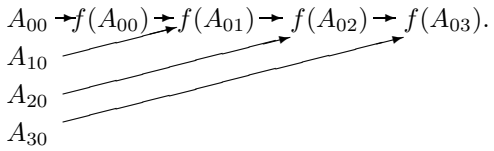
Different reduction trees can be used during the preprocessing step of TSLU. We illustrate them using an arrow notation with the following meaning. The function $f(B)$ computes GEPP of matrix B, and returns the b rows used as pivots. The input matrix B is formed by stacking atop one another the matrices situated at the left side of the arrows pointing to $f(B)$. A binary tree of height two is represented in the following picture:



A reduction tree of height one leads to the following factorization:



The flat-tree-based TSLU is illustrated using the arrow abbreviation as



Finally, we note that the name "tournament pivoting" applies to all these variations of TSLU, interpreting them all as running a tournament where at each round a subset of b competing rows is selected to advance in the tournament until the best b overall are chosen, in rank order from first to bth.

The tournament pivoting strategy has several properties. It is equivalent to partial pivoting for $b = 1$ or $P = 1$. The elimination of each column of A leads to a rank-1 update of the trailing matrix, as in GEPP. As shown experimentally in [26], the rank-1 update property can be important for the stability of LU factorization [26]. A large rank update, as for example in block parallel pivoting [26] might lead to an unstable LU factorization.

3. Numerical stability of CALU. In this section we present results showing that CALU has stability properties similar to Gaussian elimination with partial pivoting. First, we show that the Schur complement obtained after each step of CALU is the same as the Schur complement obtained after performing GEPP on a larger

matrix whose entries are the same as the entries of the input matrix (sometimes slightly perturbed) and zeros. Second, we show that the upper bound on the growth factor for CALU is much larger than for GEPP. However, the first result suggests that CALU should be stable in practice. Another way to see this is that GEPP only gives big growth factor on a small set of input matrices (see, for example, [15]) which are all variations of one particular matrix, the Wilkinson matrix. Furthermore there are Wilkinson-like matrices for which GEPP gives modest growth factor but CALU gives exponential growth ($W_{EG-CALU}$ in (3.1)), and vice-versa ($W_{EG-GEPP}$ in (3.1)). These two examples (presented here slightly more generally) are from Volkov [27]. They show that GEPP is not uniformly more stable than CALU. However, they are rather very special cases, since such matrices could be probably also found for Gaussian elimination with no pivoting, which is known to be unstable in practice.

The matrices $W_{EG-CALU}$ and $W_{EG-GEPP}$ of size $6b \times 2b$ are as following:

$$W_{EG-CALU} = \left(\begin{array}{cc} I_b & ee^T \\ 0 & W \\ 0 & 0 \\ \hline I_b & 0 \\ 0 & W \\ -I_b & 2I_b - ee^T \end{array}\right), \qquad W_{EG-GEPP} = \left(\begin{array}{cc} I_b & ee^T \\ 0 & I_b \\ 0 & 0 \\ \hline I_b & 0 \\ I_b & 2I_b \\ 0 & 2W \end{array}\right). \tag{3.1}$$

Here I_b is the $b \times b$ identity matrix, 0 is the $b \times b$ zero matrix, e is a $b \times 1$ vector with all $e_i = 1$, and W is a $b \times b$ Wilkinson matrix, i.e., $W(i,j) = -1$ for $i > j$, $W(i,i) = 1$, and $W(:,b) = 1$. We suppose that CALU divides the input matrix into two blocks, each of dimension $3b \times 2b$. For $W_{EG-CALU}$, the growth factor of GEPP is 2 while the growth factor of CALU is 2^{b-1}. This is because CALU uses pivots from W, while GEPP does not. For $W_{EG-GEPP}$, GEPP uses pivots from W and hence has an exponential growth of 2^{b-1}. For this matrix, CALU does not use pivots from W and its growth factor is 1.

Third, we measure the stability of CALU using several metrics that include growth factor and normwise backward stability. We perform our tests in MATLAB, using matrices from a normal distribution with varying size from 1024 to 8192, and a set of special matrices. We have also performed experiments on different matrices such as matrices drawn from different random distributions and dense Toeplitz matrices, and we have obtained results similar to those presented here.

3.1. Similarities with Gaussian elimination with partial pivoting. In this section we discuss similarities that exist between computing the LU factorization of a matrix A using CALU and computing the LU factorization using GEPP of a larger matrix G. The matrix G is formed by elements of A, sometimes slightly perturbed, and zeros. We first prove a related result.

LEMMA 3.1. *The CALU tournament pivoting strategy chooses for each panel factorization a set of rows that spans the row space of the panel.*

Proof. At each step of the preprocessing part of the panel factorization, two (or more) blocks A_1 and A_2 are used to determine a third block B. Since Gaussian elimination is used to choose pivot rows and determine B, $row\_span([A_1; A_2]) = row\_span(B)$. This is true at every node of the reduction tree. Therefore the final block of pivot rows spans the row space of the panel. This reasoning applies to every panel factorization. ∎

Before proving a general result that applies to CALU using any reduction tree, we first discuss a simple case of a reduction tree of height one. In the following, I_b denotes the identity matrix of size $b \times b$. Let A be an $m \times n$ matrix partitioned as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix},$$

where A_{11}, A_{21} are of size $b \times b$, A_{31} is of size $(m-2b) \times b$, A_{12}, A_{22} are of size $b \times (n-b)$, and A_{32} is of size $(m-2b) \times (n-b)$. In this example we suppose that TSLU is applied on the first block column $[A_{11}; A_{21}; A_{31}]$, and first performs GEPP of $[A_{21}; A_{31}]$. Without loss of generality we further suppose that the permutation returned at this stage is the identity; that is, the pivots are chosen on the diagonal. Second, TSLU performs GEPP on $[A_{11}; A_{21}]$, and the pivot rows are referred to as $\bar{A}_{11}$. With the arrow notation defined in section 2, the panel factorization uses the following tree (we do not display the function f, instead each node of the tree displays the result of GEPP):

$$\begin{array}{lll} A_{11} & \longrightarrow & \bar{A}_{11}. \\ A_{21} & \searrow & \nearrow \\ A_{31} & & A_{21} \end{array}$$

We refer to the block obtained after performing TSLU on the first block column and updating A_{32} as A_{32}^s. The goal of the following lemma is to show that A_{32}^s can be obtained from performing GEPP on a larger matrix. The result can be easily generalized to any reduction tree of height one.

LEMMA 3.2. *Let A be a nonsingular $m \times n$ matrix partitioned as*

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix},$$

where A_{11}, A_{21} are of size $b \times b$, A_{31} is of size $(m-2b) \times b$, A_{12}, A_{22} are of size $b \times (n-b)$, and A_{32} is of size $(m-2b) \times (n-b)$. Consider the GEPP factorizations

$$\begin{aligned} \begin{pmatrix} \Pi_{11} & \Pi_{12} & \\ \Pi_{21} & \Pi_{22} & \\ & & I_{m-2b} \end{pmatrix} \cdot \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix} &= \begin{pmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \\ A_{31} & A_{32} \end{pmatrix} \\ &= \begin{pmatrix} \bar{L}_{11} & & \\ \bar{L}_{21} & I_b & \\ \bar{L}_{31} & & I_{m-2b} \end{pmatrix} \cdot \begin{pmatrix} \bar{U}_{11} & \bar{U}_{12} \\ & \bar{A}_{22}^s \\ & A_{32}^s \end{pmatrix} \end{aligned} \tag{3.2}$$

and

$$\Pi \begin{pmatrix} A_{21} \\ A_{31} \end{pmatrix} = \begin{pmatrix} L_{21} \\ L_{31} \end{pmatrix} \cdot \begin{pmatrix} U_{21} \\ \end{pmatrix}, \tag{3.3}$$

where we suppose that $\Pi = I_{m-b}$.

The matrix A_{32}^s can be obtained after $2b$ steps of GEPP factorization of a larger matrix G; that is,

$$G = \begin{pmatrix} \bar{A}_{11} & & \bar{A}_{12} \\ A_{21} & A_{21} & \\ & -A_{31} & A_{32} \end{pmatrix}$$
$$= \begin{pmatrix} \bar{L}_{11} & & \\ A_{21}\bar{U}_{11}^{-1} & L_{21} & \\ & -L_{31} & I_{m-2b} \end{pmatrix} \cdot \begin{pmatrix} \bar{U}_{11} & & \bar{U}_{12} \\ & U_{21} & -L_{21}^{-1}A_{21}\bar{U}_{11}^{-1}\bar{U}_{12} \\ & & A_{32}^s \end{pmatrix}.$$

Proof. The first b steps of GEPP applied to G pivot on the diagonal. This is because (3.2) shows that the rows of A_{21} which could be chosen as pivots are already part of $\bar{A}_{11}$. The second b steps pivot on the diagonal as well, as can be seen from (3.3).

The following equalities prove the lemma:

$$L_{31}L_{21}^{-1}A_{21}\bar{U}_{11}^{-1}\bar{U}_{12} + A_{32}^s = L_{31}U_{21}\bar{U}_{11}^{-1}\bar{U}_{12} + A_{32}^s = A_{31}\bar{U}_{11}^{-1}\bar{U}_{12} + A_{32}^s$$
$$= \bar{L}_{31}\bar{U}_{12} + A_{32}^s = A_{32}. \qquad \square$$

In the following we prove a result that applies to any reduction tree. We consider the CALU factorization of a nonsingular matrix A of size $m \times n$. After factoring the first block column using TSLU, the rows of the lower triangular factor which were not involved in the last GEPP factorization at the root of the reduction tree are not bounded by 1 in absolute value as in GEPP. We consider such a row j and we refer to the updated $A(j, b+1:n)$ after the first block column elimination as $A^s(j, b+1:n)$. The following theorem shows that $A^s(j, b+1:n)$ can be obtained by performing GEPP on a larger matrix G whose entries are of the same magnitude as entries of the original matrix A, and hence can be bounded.

Some of the intermediate matrices on which TSLU performs GEPP can be exactly singular, even if the original matrix A is not; this may happen frequently if A is sparse. We need to account for this, either (1) by permitting fewer than b rows to be advanced in a stage of the tournament, or (2) by introducing tiny perturbations in the matrix when required to preserve nonsingularity. If A is nonsingular, then either way the final outcome of each b-column panel factorization via TSLU will be b independent rows (in exact arithmetic). Even though approach (1) is likely to be more efficient in practice (especially for sparse matrices), our proof of numerical stability will use approach (2) to simplify notation, by allowing us to assume that all submatrices are nonsingular.

We proceed by introducing a variant of GEPP, called ϵ-GEPP, that replaces any zero pivot $U(i,i) = 0$ encountered by $U(i,i) = \epsilon$, where ϵ is any (arbitrarily tiny) nonzero number. This is equivalent to adding ϵ to the ith diagonal entry $(PA)(i,i)$ of the permuted A, and then doing standard GEPP; we denote the correspondingly perturbed A by $A^{(\epsilon)}$. This assures that ϵ-GEPP always identifies b independent rows when applied to any $b' \times b$ matrix, with $b' \geq b$. And since ϵ is arbitrarily tiny, it will be as stable as GEPP as measured by $\|PA - LU\|/\|A\|$.

DEFINITION 3.3. *Let T be a reduction tree and let H be its height. Let $s_k, \ldots, s_H$ be a path of tree vertices in which the height of vertex s_h is h and s_{h+1} is the tree parent of s_h for all $h = k, \ldots, H-1$. For node s_h at level h, let $A_{s_h,h}$ be the $c \cdot b \times b$ submatrix obtained by stacking the b rows selected by each of s_h's c tree children atop one another, and let $\Pi_{s_h,h} A_{s_h,h}^{(\epsilon)} = L_{s_h,h} U_{s_h,h}$ be its ϵ-GEPP factorization.*

The matrices associated with the ancestor nodes of s_k in T are defined for all $s_h = s_k, s_{k+1}, \ldots, s_H$ and $h = k, \ldots, H$ as

$$\bar{A}_h = (\Pi_{s_h,h} A^{(\epsilon)}_{s_h,h})(1:b, 1:b),$$

with its GEPP factorization

$$\bar{A}_h = \bar{L}_h \bar{U}_h.$$

THEOREM 3.4. *Let A be a nonsingular $m \times n$ matrix that is to be factored using CALU. Consider the first block column factorization using TSLU, and let Π be the permutation returned after this step. Let j be the index of a row of A that is involved for the last time in a GEPP factorization of the CALU reduction tree at node s_k of level k.*

Consider the matrices associated with the ancestor nodes of s_k in T as described in Definition 3.3*, and let*

$$\bar{A}_H = (\Pi A)(1:b, 1:b),$$
$$\hat{A}_H = (\Pi A)(1:b, b+1:n).$$

The updated row $A^s(j, b+1:n)$ obtained after the first block column factorization of A by TSLU, that is,

$$\begin{pmatrix} \bar{A}_H & \hat{A}_H \\ A(j,1:b) & A(j,b+1:n) \end{pmatrix} = \begin{pmatrix} \bar{L}_H & \\ L(j,1:b) & 1 \end{pmatrix} \cdot \begin{pmatrix} \bar{U}_H & \hat{U}_H \\ & A^s(j,b+1:n) \end{pmatrix}, \tag{3.4}$$

is equal to the updated row obtained after performing GEPP on the leading $(H-k+1)b$ columns of a larger matrix G of dimension $((H-k+1)b+1) \times ((H-k+1)b+1)$, that is,

$$G = \begin{pmatrix} \bar{A}_H & & & & & & \hat{A}_H \\ \bar{A}_{H-1} & \bar{A}_{H-1} & & & & & \\ & \bar{A}_{H-2} & \bar{A}_{H-2} & & & & \\ & & \ddots & \ddots & & & \\ & & & \bar{A}_k & \bar{A}_k & & \\ & & & & (-1)^{H-k}A(j,1:b) & A(j,b+1:n) \end{pmatrix}$$

$$= \begin{pmatrix} \bar{L}_H & & & & & \\ \bar{A}_{H-1}\bar{U}_H^{-1} & \bar{L}_{H-1} & & & & \\ & \bar{A}_{H-2}\bar{U}_{H-1}^{-1} & \bar{L}_{H-2} & & & \\ & & \ddots & \ddots & & \\ & & & \bar{A}_k\bar{U}_{k+1}^{-1} & \bar{L}_k & \\ & & & & (-1)^{H-k}A(j,1:b)\bar{U}_k^{-1} & 1 \end{pmatrix}$$

$$\cdot \begin{pmatrix} \bar{U}_H & & & & \hat{U}_H \\ & \bar{U}_{H-1} & & & \hat{U}_{H-1} \\ & & \bar{U}_{H-2} & & \hat{U}_{H-2} \\ & & & \ddots & \vdots \\ & & & \bar{U}_k & \hat{U}_k \\ & & & & A^s(j,b+1:n) \end{pmatrix}, \tag{3.5}$$

where

$$\hat{U}_{H-i} = \begin{cases} \bar{L}_H^{-1}\hat{A}_H & \text{if } i = 0; \\ -\bar{L}_{H-i}^{-1}\bar{A}_{H-i}\bar{U}_{H-i+1}\hat{U}_{H-i+1} & \text{if } 0 < i \leq H-k. \end{cases} \tag{3.6}$$

Proof. Since the matrix is nonsingular, the final step of ϵ-GEPP does not need to modify $\bar{A}_H$ to make its rows independent, whereas the other entries $\bar{A}_k, \ldots, \bar{A}_{H-1}$ of G may have had ϵ perturbations.

From (3.5), $A^s(j, b+1:n)$ can be computed as follows:

$$\begin{aligned} &A^s(j, b+1:n) \\ &\quad = A(j, b+1:n) - \begin{pmatrix} 0 & \ldots & 0 & (-1)^{H-k}A(j, 1:b) \end{pmatrix} \\ &\qquad \cdot \left(\begin{pmatrix} \bar{A}_H & & & \\ & \bar{A}_{H-1} & & \\ & & \ddots & \\ & & & \bar{A}_k \end{pmatrix} \cdot \begin{pmatrix} I_b & & & \\ I_b & I_b & & \\ & \ddots & \ddots & \\ & & I_b & I_b \end{pmatrix} \right)^{-1} \cdot \begin{pmatrix} \hat{A}_H \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\ &\quad = A(j, b+1:n) - \begin{pmatrix} 0 & \ldots & 0 & (-1)^{H-k}A(j, 1:b) \end{pmatrix} \\ &\qquad \cdot \begin{pmatrix} I_b & & & \\ -I_b & I_b & & \\ & \ddots & \ddots & \\ (-1)^{H-k}I_b & \ldots & -I_b & I_b \end{pmatrix} \cdot \begin{pmatrix} \bar{A}_H^{-1} & & & \\ & \bar{A}_{H-1}^{-1} & & \\ & & \ddots & \\ & & & \bar{A}_k^{-1} \end{pmatrix} \cdot \begin{pmatrix} \hat{A}_H \\ 0 \\ \vdots \\ 0 \end{pmatrix} \\ &\quad = A(j, b+1:n) - A(j, 1:b)\bar{A}_H^{-1}\hat{A}_H. \end{aligned}$$

The last equality represents the computation of $A^s(j, b+1:n)$ obtained from (3.4), and this ends the proof. ∎

The following corollary shows similarities between CALU and GEPP of a larger matrix G_{CALU}. Since GEPP is stable in practice, we expect CALU to be also stable in practice. However, we note a weakness of this argument. It is not impossible that the larger matrix G_{CALU} is closer to a Wilkinson-like matrix than A, and GEPP could generate a large growth factor. But we have never observed this in practice.

COROLLARY 3.5. *The Schur complement obtained after each step of performing CALU on a matrix A is equivalent to the Schur complement obtained after performing GEPP on a larger matrix whose entries are the same as the entries of A, sometimes slightly perturbed, or zeros.*

In other words, Corollary 3.5 says that the entire CALU process is equivalent to GEPP on a different, possibly much larger, matrix G_{CALU}. We describe briefly an approach to build this matrix. Consider a row k of the trailing matrix obtained after performing the first two steps in CALU, that is, after two TSLUs and updates were performed. We form a matrix G_k which contains only blocks from the original matrix A as following. Similar to the construction of matrix G in (3.5), the updated row k can be obtained by factoring a matrix F formed by $b(H+1)+1$ rows of the second panel whose values correspond to those obtained after one step of CALU. To form G_k, we use the fact that each such row, that we note E^s, can be obtained by factoring a matrix whose elements are the same as elements of A or zeros, as in (3.5). Our construction requires multiple copies of E^s (or more generally blocks) in various places. For illustration, let E^s be the updated block obtained when taking the Schur

TABLE 3.1

Bounds for the elements of $|L|$ and for the growth factor g_W obtained from factoring a matrix of size $m \times (b+1)$ and $m \times n$ using CALU and GEPP. CALU uses a reduction tree of height H and a block of size b. For the matrix of size $m \times (b+1)$, the result for CALU corresponds to the first step of panel factorization based on TSLU.

	Matrix of size $m \times (b+1)$		
	TSLU(b,H)		GEPP
	Upper bound	Attained	Upper bound
$\|L\|$	2^{bH}	$2^{(b-2)H-(b-1)}$	1
g_W	$2^{b(H+1)}$	2^b	2^b
	Matrix of size $m \times n$		
	CALU(b,H)		GEPP
	Upper bound	Attained	Upper bound
$\|L\|$	2^{bH}	$2^{(b-2)H-(b-1)}$	1
g_W	$2^{n(H+1)-1}$	2^{n-1}	2^{n-1}

complement of B in

$$\begin{pmatrix} B & C \\ D & E \end{pmatrix}.$$

Suppose that we need to make the following copies of E^s appear:

$$\begin{pmatrix} E^s & E^s & V & \\ & V & E^s & V \end{pmatrix},$$

where V are other blocks. Then we just build the larger matrix

$$\begin{pmatrix} B & & C & C & & \\ & B & & & C & \\ D & & E & E & V & \\ & D & & V & E & V \end{pmatrix}, \tag{3.7}$$

and the elimination of each block B on the diagonal leads to the update of all blocks E from the same row. Using this approach, the matrix G_k can be obtained by extending matrix F and adding in front of the diagonal a submatrix of dimension $b(H+1)$ for each row of F. We note that G_k is very sparse, its GEPP factorization involves $b(H+1)+1$ independent factorizations of submatrices of dimension $b(H+1)$, followed by the factorization of submatrix F of dimension $b(H+1)+1$. Each independent factorization updates only one row of F. Hence the growth factor depends only on one independent factorization and on the factorization of F.

The reasoning can continue for the following steps of CALU. This leads to a large G_{CALU} matrix, however very sparse and with many independent factorizations that will update only subparts of the matrix. It can be seen that the growth factor does not depend on the dimension of G_{CALU}, and it is bounded by $2^{n(H+1)-1}$, as displayed in Table 3.1.

In the following theorem, we use the same approach as in Theorem 3.4 to bound the L factor obtained from the CALU factorization of a matrix A.

THEOREM 3.6. *Let A be a nonsingular $m \times n$ matrix that is to be factored by CALU based on a reduction tree of height H and using a block of size b. The elements of the factor L are bounded in absolute value by 2^{bH}.*

Proof. Consider the first block column factorization using TSLU, and let Π be the permutation returned after this step. Let j be the index of a row of A that is

involved only in a GEPP factorization at the leaf (node s_0, level 0) of the CALU reduction tree. Without loss of generality, we suppose that $\Pi(j,j) = 1$, that is row j is not permuted from its original position. Consider the matrices associated with the ancestor nodes of s_0 in the reduction tree T as described in Definition 3.3. The jth row of the L factor satisfies the relation:

$$\begin{pmatrix} \bar{A}_H \\ A(j,1:b) \end{pmatrix} = \begin{pmatrix} \bar{L}_H \\ L(j,1:b) \end{pmatrix} \bar{U}_H.$$

We have the following:

$$\begin{aligned} |L(j,1:b)| &= |A(j,1:b) \cdot \bar{U}_H^{-1}| \\ &= |A(j,1:b) \cdot \bar{A}_0^{-1} \cdot \bar{A}_0 \cdot \bar{A}_1^{-1} \cdot \bar{A}_1 \ldots \bar{A}_{H-1}^{-1} \cdot \bar{A}_{H-1} \cdot \bar{U}_H^{-1}| \\ &= |A(j,1:b) \cdot \bar{U}_0^{-1} \cdot \bar{L}_0^{-1} \cdot \bar{A}_0 \cdot \bar{U}_1^{-1} \cdot \bar{L}_1^{-1} \cdot \bar{A}_1 \ldots \bar{U}_{H-1}^{-1} \\ &\quad \cdot \bar{L}_{H-1}^{-1} \cdot \bar{A}_{H-1} \cdot \bar{U}_H^{-1}| \\ &\leq |A(j,1:b) \cdot \bar{U}_0^{-1}| \cdot |\bar{L}_0^{-1}| \cdot |\bar{A}_0 \cdot \bar{U}_1^{-1}| \cdot |\bar{L}_1^{-1}| \ldots |\bar{L}_{H-1}^{-1}| \cdot |\bar{A}_{H-1} \cdot \bar{U}_H^{-1}|. \end{aligned}$$

The elements of $|A(j,1:b) \cdot \bar{U}_0^{-1}|$ and $|\bar{A}_{i-1} \cdot \bar{U}_i^{-1}| \leq 1$, for $i = 1, \ldots, H$, are bounded by 1. In addition $\bar{L}_{i-1}$, for $i = 1, \ldots, H$, is a $b \times b$ unit lower triangular matrix whose elements are bounded by 1 in absolute value. The elements of each row j of $|\bar{L}_i^{-1}| \cdot |\bar{A}_i \cdot \bar{U}_{i+1}^{-1}|$ are bounded by 2^{j-1}. Hence, the elements of $|L(j,1:b)|$ are bounded by 2^{bH}.

The same reasoning applies to the following steps of factorization, and this ends the proof. □

Theorem 3.6 shows that the elements of $|L|$ are bounded by 2^{bH}. For a flat reduction tree with $H = n/b$, this bound becomes of order 2^n. This suggests that the more levels in the reduction tree we have, the less stable the factorization may become.

We give an example of a matrix formed by Wilkinson-type submatrices whose factor L obtained from CALU factorization has an element of the order of $2^{(b-2)H-(b-1)}$, which is close to the bound in Theorem 3.6. This matrix is formed by the following submatrices $\bar{A}_i$ (we use the same notation as in Theorems 3.4 and 3.6). Let W be a unit lower triangular matrix of order $b \times b$ with $W(i,j) = -1$ for $i > j$ (the same definition of a Wilkinson-type matrix as before). Let v be a vector of dimension $H+1$ defined as following: $v(1) = 1$, and $v(i) = v(i-1)(2^{b-2}+1)+1$ for all $i = 2 : H+1$. Then $\bar{A}_i = W + v(H-i+1) \cdot e_b \cdot e_1^T$, and $A(j,1:b) = (e_1 + v(H+1) \cdot e_b)^T$.

The upper bound for $|L|$ is much larger for CALU than for GEPP. However, we note that for the backward stability of the LU factorization, the growth factor plays an important role, not $|L|$. This is shown in the following lemma from [16], which uses the growth factor g_W defined in (3.8), where $a_{ij}^{(k)}$ denotes the entry in position (i,j) obtained after k steps of elimination:

$$g_W = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{ij} |a_{ij}|}. \tag{3.8}$$

LEMMA 3.7 (see Lemma 9.6, section 9.3 of [16]). *Let $A = LU$ be the Gaussian elimination without pivoting of A. Then $\||L||U|\|_\infty$ is bounded using the growth factor g_W by the relation $\||L||U|\|_\infty \leq (1 + 2(n^2 - n)g_W)\|A\|_\infty$.*

The growth factor obtained after performing one panel factorization in CALU (using TSLU) is equal to the growth factor of matrix G in (3.5) of Theorem 3.4. This

theorem implies that the growth factor can be as large as $2^{b(H+1)}$. It is shown in [15] that the L factor of matrices that attain the maximum growth factor is a dense unit lower triangular matrix. Hence the growth factor of matrix G in (3.5) cannot attain the maximum value of $2^{b(H+1)}$, since its L factor is lower block bidiagonal. In addition, matrix G has a special form as described in (3.5). We were not able to find matrices that attain the worst case growth factor, the largest growth factor we could observe is of order 2^b. For matrices for which a large $|L|$ is attained, the growth factor is still of the order of 2^b, since the largest element in $|L|$ is equal to the largest element in $|A|$. We conjecture that the growth factor of G is bounded by 2^b.

Table 3.1 summarizes bounds derived in this section for CALU and also recalls bounds for GEPP. It considers a matrix of size $m \times (b+1)$ for which one TSLU factorization is performed, and also the general case of a matrix of size $m \times n$. It displays bounds for $|L|$ and for the growth factor g_W.

As an additional observation, we note that matrices whose L factor is lower block bidiagonal can attain a growth factor within a constant factor of the maximum. One example is the following very sparse W_s matrix of dimension $n \times n$ with $n = bH + 1$, formed by Kronecker products involving the Wilkinson-type matrix W:

$$W_s = \begin{pmatrix} I_H \otimes W + S \otimes N & e_1 \\ e_{n-1}^T & \end{pmatrix}, \tag{3.9}$$

where W is unit lower triangular of order $b \times b$ with $W(i,j) = -1$ for $i > j$, N is of order $b \times b$ with $N(i,j) = -1$ for all i, j, I_H is the identity matrix of order $H \times H$, S is a lower triangular matrix of order $H \times H$ with $S(i,j) = 1$ for $i = j+1$, 0 otherwise, e_1 is the vector $(1, 0, \ldots, 0)$ of dimension $(n-1) \times 1$, and e_{n-1} is the vector $(0, \ldots, 0, 1)$ of dimension $(n-1) \times 1$. For example, when $H = 3$ this matrix becomes

$$\begin{pmatrix} W & & & e_1 \\ N & W & & \\ & N & W & \\ & & e_b^T & \end{pmatrix}. \tag{3.10}$$

The matrix W_s gets growth factor of $.25 \cdot 2^{n-1} \cdot (1 - 2^{-b})^{H-2}$. Hence by choosing b and H so that $H \approx 2^b$, it gets growth factor of about $.1 \cdot 2^{n-1}$, which is within a constant factor of the maximum growth factor 2^{n-1} of a dense $n \times n$ matrix.

3.2. Experimental results. We present experimental results showing that CALU is stable in practice and compare them with those obtained from GEPP. The results focus on CALU using a binary tree and CALU using a flat tree, as defined in section 2.

In this section we focus on matrices whose elements follow a normal distribution. In MATLAB notation, the test matrix is $A = \mathsf{randn}(n, n)$, and the right-hand side is $b = \mathsf{randn}(n, 1)$. The size of the matrix is chosen such that n is a power of 2; that is, $n = 2^k$. The sample size is in general 3, but we use only 1 or 2 matrices when the size of the matrix is large (more precisely, the sample size is $\max\{10 * 2^{10-k}, 3\}$). We discuss several metrics, which concern the LU decomposition and the linear solver using it, such as the growth factor and normwise and componentwise backward errors. Additional results that consider as well several special matrices [14], including sparse matrices, are described in Appendix A.

In this section we present results for the growth factor g_T defined in (3.11), which was introduced by Trefethen and Schreiber in [26]. The authors have introduced a

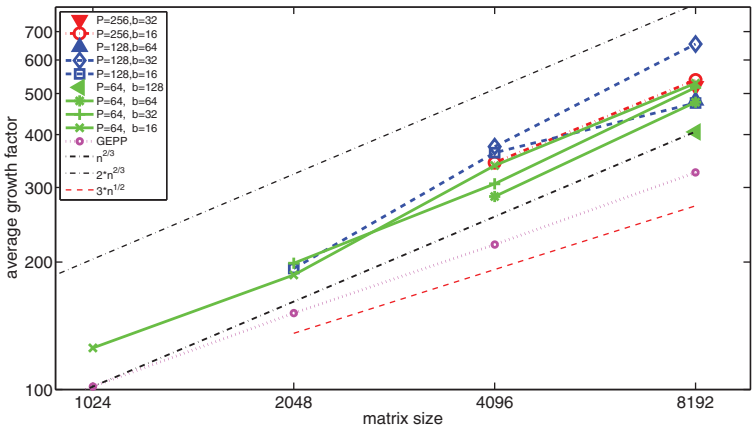


FIG. 3.1. *Growth factor g_T of binary-tree-based CALU for random matrices.*

statistical model for the average growth factor, where σ_A is the standard deviation of the initial element distribution of A. In the data presented here $\sigma_A = 1$. They observed that the average growth factor g_T is close to $n^{2/3}$ for partial pivoting and $n^{1/2}$ for complete pivoting (at least for $n \leqslant 1024$). In Appendix A we also present results for g_W , defined in (3.8), as well as the growth factor g_D defined in (3.12), which was introduced in [7]. As for g_W, $a_{ij}^{(k)}$ denotes the entry in position (i, j) obtained after k steps of elimination.

$$g_T = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\sigma_A}, \tag{3.11}$$

$$g_D = \max_j \left\{ \frac{\max_i |u_{ij}|}{\max_i |a_{ij}|} \right\}. \tag{3.12}$$

Figure 3.1 displays the values of the growth factor g_T of the binary-tree-based CALU, for different block sizes b and different number of processors P. As explained in section 2, the block size determines the size of the panel, while the number of processors determines the number of block rows in which the panel is partitioned. This corresponds to the number of leaves of the binary tree. We observe that for matrices of size up to 8192, the growth factor of binary-tree-based CALU grows as $C \cdot n^{2/3}$, where C is a small constant around 1.5. We can also note that the growth factor of GEPP is of order $O(n^{2/3})$, which matches the result in [26]. However, these results might not apply as n grows much larger. Indeed, it is conjectured in [26] that the growth factor of GEPP is of order $O(n^{1/2})$ as $n \to \infty$.

Figure 3.2 shows the values of the growth factor g_T for flat-tree-based CALU with varying block size b from 4 to 64. The curves of the growth factor lie between $n^{2/3}$ and $2n^{2/3}$ in our tests on random matrices. The growth factors of both binary-tree-based and flat-tree-based CALU have similar behaviors to the growth factor of GEPP.

Table 3.2 presents results for the linear solver using binary-tree-based and flat-tree-based CALU, together with GEPP for the comparison. The normwise backward stability is evaluated by computing three accuracy tests as performed in the HPL (high-performance linpack) benchmark [9], and denoted as HPL1, HPL2, and HPL3

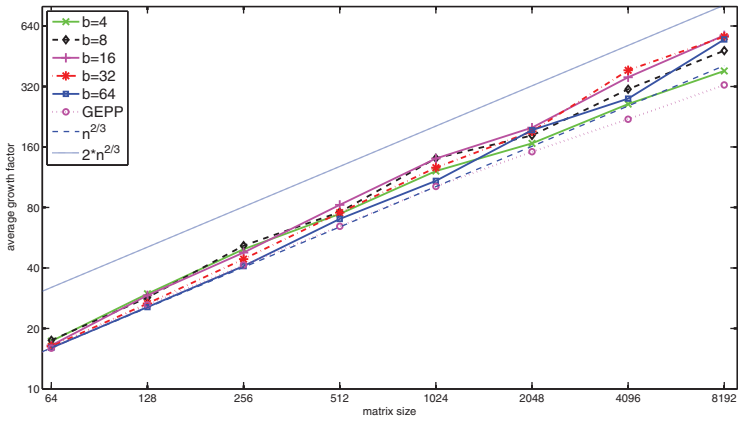


FIG. 3.2. *Growth factor g_T of flat-tree-based CALU for random matrices.*

(equations (3.13) to (3.15)),

$$\text{HPL1} = ||Ax - b||_\infty / (\epsilon ||A||_1 * N), \tag{3.13}$$

$$\text{HPL2} = ||Ax - b||_\infty / (\epsilon ||A||_1 ||x||_1), \tag{3.14}$$

$$\text{HPL3} = ||Ax - b||_\infty / (\epsilon ||A||_\infty ||x||_\infty * N). \tag{3.15}$$

In HPL, the method is considered to be accurate if the values of the three quantities are smaller than 16. More generally, the values should be of order $O(1)$. We also display the normwise backward error, using the 1-norm,

$$\eta := \frac{||r||}{||A||\ ||x|| + ||b||}. \tag{3.16}$$

We also include results obtained by iterative refinement, which can be used to improve the accuracy of the solution. For this, the componentwise backward error,

$$w := \max_i \frac{|r_i|}{(|A|\ |x| + |b|)_i}, \tag{3.17}$$

is used, where the computed residual is $r = b - Ax$. The residual is computed in working precision [22] as implemented in LAPACK [1]. The iterative refinement is performed as long as the following three conditions are satisfied: (1) the componentwise backward error is larger than eps; (2) the componentwise backward error is reduced by half; (3) the number of steps is smaller than 10. In Table 3.2, w_b denotes the componentwise backward error before iterative refinement and N_{IR} denotes the number of steps of iterative refinement. N_{IR} is not always an integer since it represents an average. We note that for all the sizes tested in Table 3.2, CALU leads to results within a factor of 10 of the GEPP results.

In Appendix A we present more detailed results on random matrices. We also consider different special matrices, including sparse matrices, described in Table A.6. There we include different metrics, such as the norm of the factors, their conditioning, the value of their maximum element, and the backward error of the LU factorization. For the special matrices, we compare the binary-tree-based and the flat-tree-based CALU with GEPP in Tables A.3, A.4, and A.5.

TABLE 3.2
Stability of the linear solver using binary-tree-based and flat-tree-based CALU and GEPP.

n	P	b	η	w_b	N_{IR}	HPL1	HPL2	HPL3
Binary-tree-based CALU								
8192	256	32	6.2E-15	4.1E-14	2	3.6E-2	2.2E-2	4.5E-3
		16	5.8E-15	3.9E-14	2	4.5E-2	2.1E-2	4.1E-3
	128	64	6.1E-15	4.2E-14	2	5.0E-2	2.2E-2	4.6E-3
		32	6.3E-15	4.0E-14	2	2.5E-2	2.1E-2	4.4E-3
		16	5.8E-15	4.0E-14	2	3.8E-2	2.1E-2	4.3E-3
	64	128	5.8E-15	3.6E-14	2	8.3E-2	1.9E-2	3.9E-3
		64	6.2E-15	4.3E-14	2	3.2E-2	2.3E-2	4.4E-3
		32	6.3E-15	4.1E-14	2	4.4E-2	2.2E-2	4.5E-3
		16	6.0E-15	4.1E-14	2	3.4E-2	2.2E-2	4.2E-3
4096	256	16	3.1E-15	2.1E-14	1.7	3.0E-2	2.2E-2	4.4E-3
	128	32	3.2E-15	2.3E-14	2	3.7E-2	2.4E-2	5.1E-3
		16	3.1E-15	1.8E-14	2	5.8E-2	1.9E-2	4.0E-3
	64	64	3.2E-15	2.1E-14	1.7	3.1E-2	2.2E-2	4.6E-3
		32	3.2E-15	2.2E-14	1.3	3.6E-2	2.3E-2	4.7E-3
		16	3.1E-15	2.0E-14	2	9.4E-2	2.1E-2	4.3E-3
2048	128	16	1.7E-15	1.1E-14	1.8	6.9E-2	2.3E-2	5.1E-3
	64	32	1.7E-15	1.0E-14	1.6	6.5E-2	2.1E-2	4.6E-3
		16	1.6E-15	1.1E-14	1.8	4.7E-2	2.2E-2	4.9E-3
1024	64	16	8.7E-16	5.2E-15	1.6	1.2E-1	2.1E-2	4.7E-3
Flat-tree-based CALU								
8192	-	4	4.1E-15	2.9E-14	2	1.4E-2	1.5E-2	3.1E-3
	-	8	4.5E-15	3.1E-14	1.7	4.4E-2	1.6E-2	3.4E-3
	-	16	5.6E-15	3.7E-14	2	1.9E-2	2.0E-2	3.3E-3
	-	32	6.7E-15	4.4E-14	2	4.6E-2	2.4E-2	4.7E-3
	-	64	6.5E-15	4.2E-14	2	5.5E-2	2.2E-2	4.6E-3
4096	-	4	2.2E-15	1.4E-14	2	9.3E-3	1.5E-2	3.1E-3
	-	8	2.6E-15	1.7E-14	1.3	1.3E-2	1.8E-2	4.0E-3
	-	16	3.0E-15	1.9E-14	1.7	2.6E-2	2.0E-2	3.9E-3
	-	32	3.8E-15	2.4E-14	2	1.9E-2	2.5E-2	5.1E-3
	-	64	3.4E-15	2.0E-14	2	6.0E-2	2.1E-2	4.1E-3
2048	-	4	1.3E-15	7.9E-15	1.8	1.3E-1	1.6E-2	3.7E-3
	-	8	1.5E-15	8.7E-15	1.6	2.7E-2	1.8E-2	4.2E-3
	-	16	1.6E-15	1.0E-14	2	2.1E-1	2.1E-2	4.5E-3
	-	32	1.8E-15	1.1E-14	1.8	2.3E-1	2.3E-2	5.1E-3
	-	64	1.7E-15	1.0E-14	1.2	4.1E-2	2.1E-2	4.5E-3
1024	-	4	7.0E-16	4.4E-15	1.4	2.2E-2	1.8E-2	4.0E-3
	-	8	7.8E-16	4.9E-15	1.6	5.5E-2	2.0E-2	4.9E-3
	-	16	9.2E-16	5.2E-15	1.2	1.1E-1	2.1E-2	4.8E-3
	-	32	9.6E-16	5.8E-15	1.1	1.5E-1	2.3E-2	5.6E-3
	-	64	8.7E-16	4.9E-15	1.3	7.9E-2	2.0E-2	4.5E-3
GEPP								
8192	-		3.9E-15	2.6E-14	1.6	1.3E-2	1.4E-2	2.8E-3
4096	-		2.1E-15	1.4E-14	1.6	1.8E-2	1.4E-2	2.9E-3
2048	-		1.1E-15	7.4E-15	2	2.9E-2	1.5E-2	3.4E-3
1024	-		6.6E-16	4.0E-15	2	5.8E-2	1.6E-2	3.7E-3

Tournament pivoting does not ensure that the element of maximum magnitude is used as pivot at each step of factorization. Hence $|L|$ is not bounded by 1 as in Gaussian elimination with partial pivoting. To discuss this aspect, we compute at each elimination step k the threshold τ_k, defined as the quotient of the pivot used at step k divided by the maximum value in column k. In our tests we compute the minimum value of the threshold $\tau_{\min} = \min_k \tau_k$ and the average value of the threshold $\tau_{ave} = (\sum_{k=1}^{n-1} \tau_k)/(n-1)$, where n is the number of columns. The average maximum element of L is $1/\tau_{\min}$. We observe that in practice the pivots used by tournament

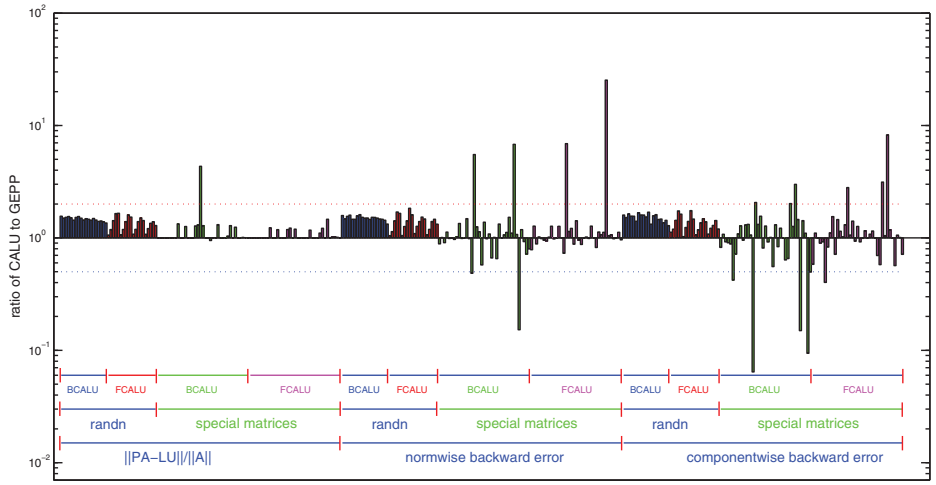


FIG. 3.3. *A summary of all our experimental data, showing the ratio of CALU's backward error to GEPP's backward error for all test matrices. Each vertical bar represents such a ratio for one test matrix, so bars above* $10^0 = 1$ *mean CALU's backward error is larger, and bars below* 1 *mean GEPP's backward error is larger. There are a few examples where the backward error of each is exactly* 0*, and the ratio* 0/0 *is shown as* 1*. As can be seen, nearly all ratios are between* 1 *and* 10*, with a few outliers up to* 26 *(GEPP more stable) and down to* .06 *(CALU more stable). For each matrix and algorithm, the backward error is measured* 3 *ways. For the first third of the bars, labeled* $\|PA - LU\|/\|A\|$*, this is backward error metric, using the Frobenius norm. For the middle third of the bars, labeled "normwise backward error,"* η *in* (3.16) *is the metric. For the last third of the bars, labeled "componentwise backward error,"* w *in* (3.17) *is the metric. The test matrices are further labeled either as "randn," which are randomly generated, or "special," listed in Table* A.6*. Finally, each test matrix is done using both CALU with a binary reduction tree (labeled BCALU) and with a flat reduction tree (labeled FCALU). Tables* A.1–A.5 *contain all the raw data.*

pivoting are close to the elements of maximum magnitude in the respective columns. For binary-tree-based and flat-tree-based CALU, the minimum threshold $\tau_{\min}$ is larger than 0.24 on all our test matrices. This means that in our tests $|L|$ is bounded by 4.2.

For all the matrices in our test set, the componentwise backward error is reduced to 10^{-16} after 2 or 3 steps of iterative refinement for all methods.

Figure 3.3 summarizes all our stability results for CALU. This figure displays the ratio of the relative error $\|PA - LU\|/\|A\|$, the normwise backward error η, and the componentwise backward error w of CALU versus GEPP. Results for all the matrices in our test set are presented as follows: 20 random matrices from Table 3.2 and 37 special matrices from Table A.6.

The results presented in this section and in Appendix A show that binary-tree-based and flat-tree-based CALU are stable, and have the same behavior as GEPP, including the ill-conditioned matrices in our test set.

4. Alternative algorithms. We consider in this section several other approaches to pivoting that avoid communication, and appear that they might be as stable as tournament pivoting, but can in fact be unstable. These approaches are based as well on a block algorithm, that factors the input matrix by traversing blocks of columns (panels) of size b. But in contrast to CALU, they compute only once the panel factorization as follows. Each panel factorization is performed by computing a sequence of LU factorizations until all the elements below the diagonal are eliminated and an upper triangular matrix is obtained. The idea of performing the LU factorization as

a reduction operation is present as well. But the LU factorization performed at nodes of the reduction tree uses U factors previously computed, and not rows of the original matrix as in CALU. Because of this, we conjecture that it is not possible to reduce these factorizations to performing GEPP on a larger matrix formed by elements of the input matrix and zeros, as we are able to do for CALU.

We present first a factorization algorithm that uses a binary tree and is suitable for parallel computing. Every block column is partitioned in P block-rows $[A_0; A_1; \ldots; A_{P-1}]$. Consider, for example, $P = 4$ and suppose that the number of rows m divides 4. We illustrate this factorization using an "arrow" abbreviation. In this context, the notation has the following meaning: each U factor is obtained by performing the LU factorization with partial pivoting of all the matrices at the other ends of the arrows stacked atop one another.

The procedure starts by performing independently the LU factorization with partial pivoting of each block row A_i. After this stage there are four U factors. The algorithm continues by performing the LU factorization with partial pivoting of pairs of U factors stacked atop one another, until the final U_{02} factor is obtained,

$$\begin{array}{l} A_0 \to U_{00} \searrow \\ A_1 \to U_{10} \nearrow \end{array} U_{01} \searrow \quad U_{02}. \qquad \begin{array}{l} A_2 \to U_{20} \searrow \\ A_3 \to U_{30} \nearrow \end{array} U_{11} \nearrow$$

A flat tree can be used as well, and the execution of the factorization on this structure is illustrated using the "arrow" abbreviation as

$$\begin{array}{l} A_0 \to U_{00} \to U_{01} \to U_{02} \to U_{03}. \\ A_1 \nearrow \\ A_2 \nearrow \\ A_3 \nearrow \end{array}$$

When the block size b is equal to 1 and when the number of processors P is equal to the number of rows m, the binary-tree-based and the flat-tree-based factorizations correspond to two known algorithms in the literature: parallel pivoting and pairwise pivoting (discussed, for example, in [26]). Hence, we refer to these extensions as block parallel pivoting and block pairwise pivoting. Factorization algorithms based on block pairwise pivoting are used in the context of out-of-core algorithms [28, 19], updated factorizations [20], and multicore architectures [4, 21], and are referred to as incremental pivoting based algorithms [19] or tiled algorithms [4].

There are two important differences between these algorithms and the classic LU factorization algorithm. First, in LU factorization, the elimination of each column of A leads to a rank-1 update of the trailing matrix. The rank-1 update property and the fact that the elements of L are bounded are two properties that are shown experimentally to be very important for the stability of LU factorization [26]. It is thought [26] that the rank-1 update inhibits potential element growth during the factorization. A large rank update might lead to an unstable LU factorization. Parallel pivoting is known to be unstable; see, for example, [26]. Note that the elimination of each column leads to a rank update of the trailing matrix equal to the number of rows involved at each step of elimination. The experiments performed in [26] on random matrices show that pairwise pivoting uses in practice a low rank update. Second, block parallel pivoting and block pairwise pivoting use in their computation

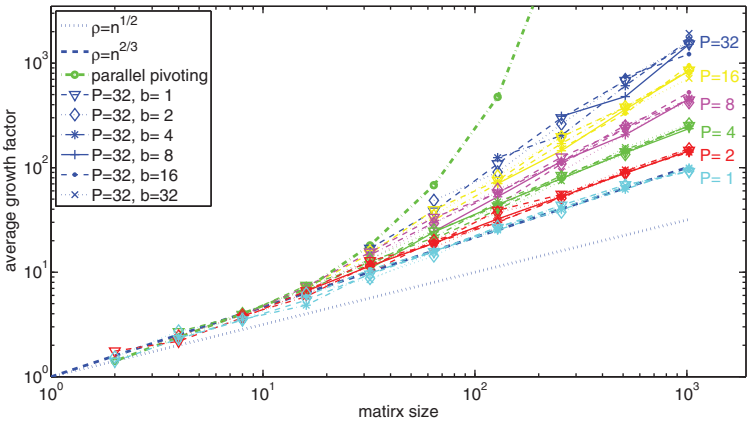


FIG. 4.1. *Growth factor of block parallel pivoting for varying block size b and number of processors P. The legend at the left of the plot indicates the symbols used for P = 32 and b varied. The same symbols, but different colors, are used for different values of P.*

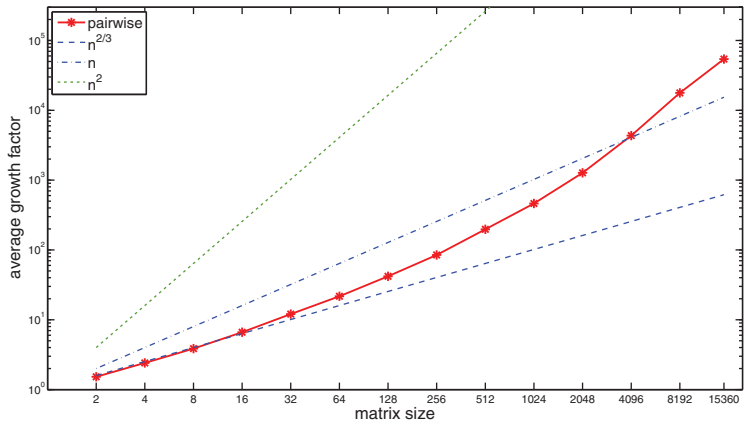


FIG. 4.2. *Growth factor of pairwise pivoting for varying matrix size.*

factorizations that involve U factors previously computed. This can propagate ill-conditioning through the factorization.

We discuss here the stability in terms of growth factor for block parallel pivoting and pairwise pivoting. We perform our tests in MATLAB, using matrices from a normal distribution. The growth factor of block parallel pivoting is displayed in Figure 4.1. We vary the number of processors P on which each block column is distributed, and the block size b used in the algorithm. The matrix size varies from 2 to 1024. We can see that the number of processors P has an important impact on the growth factor, while b has little impact. The growth factor increases with increasing P, with an exponential growth in the extreme case of parallel pivoting. Hence, for large number of processors, block parallel pivoting is unstable. We note further that using iterative refinement does not help improve the stability of the algorithm for a large number of processors. We conclude that block parallel pivoting is unstable.

The growth factor of pairwise pivoting is displayed in Figure 4.2. The matrix size varies from 2 to 15360 (the maximum size we were able to test with our code). We

note that for small matrix size, pairwise pivoting has a growth factor on the order of $n^{2/3}$. With increasing matrix size, the growth of the factor is faster than linear. For $n > 2^{12}$, the growth factor becomes larger than n. This suggests that further experiments are necessary to understand the stability of pairwise pivoting and its block version.

We note that tournament pivoting bears some similarities to the batched pivoting strategy [10]. To factor a block column partitioned as $[A_0; A_1; \dots ; A_{P-1}]$, batched pivoting also uses two steps. It identifies first b rows, that are then used as pivots for the entire block column. The identification of the b rows is different from CALU. In batched pivoting, each block A_i is factored using Gaussian elimination with partial pivoting. One of the P sets of b pivot rows is selected, based on some criterion, and used to factor the entire block column. Hence, the different P sets are not combined as in CALU. In particular when all the blocks A_i are singular, batched pivoting will fail, even if the block-column is nonsingular. This can happen for sparse matrices.

5. CALU algorithm. In this section we describe the CALU factorization algorithm in more detail than before, in order to model its performance, and show that it is optimal. We use the classical (γ, α, β) model that describes a parallel machine in terms of the time per floating point operation (add and multiply) γ, the network latency α, and the inverse of the bandwidth β. In this model the time to send a message of n words is estimated to be $\alpha + n\beta$. A broadcast or a reduce of n words between P processors is estimated to correspond to $\log_2 P$ messages of size n. We omit low order terms in our estimations.

As described in section 2, CALU factors the input matrix by iterating over panels. At each iteration it factors the current panel using TSLU and then it updates the trailing matrix. The trailing matrix update can be performed by any existing algorithm, and so we will not detail it in this paper. We focus mainly on the description of the TSLU algorithm used for panel factorization. TSLU performs the panel factorization in two steps: a preprocessing step to find good pivots, followed by the LU factorization of the panel that uses these pivots. The preprocessing step uses tournament pivoting, and it is performed as a reduction operation, with GEPP being performed at each node of the reduction tree. TSLU can take as input an arbitrary reduction tree, and this algorithmic flexibility is illustrated in Algorithm 1, that presents a parallel TSLU implementation. However, the performance of TSLU and CALU is modeled for specific trees. This is because, as we will see in the following section, a binary-tree-based TSLU/CALU and a flat-tree-based TSLU/CALU lead to optimal algorithms for parallel and sequential machines, respectively.

In the parallel TSLU implementation presented in Algorithm 1, the input matrix is distributed over P processors using a one-dimensional (1-D) block row layout. For ease of presentation, the algorithm uses an all-reduction tree; that is, the result is available on all the processors. In the preprocessing step, the algorithm traverses the reduction tree bottom-up. At the leaves, each processor computes independently the GEPP factorization of its block. Then at each node of the reduction tree, the processors exchange the pivot rows they have computed at the previous step. A matrix is formed by the pivot rows stacked atop one another and it is factored using GEPP. The pivots used in the final GEPP factorization at the root of the reduction tree are the pivots that will be used to factor the entire panel. The description of TSLU follows the same approach as the presentation of parallel TSQR in [6].

The runtime estimation of this algorithm when using a binary tree is displayed in Table 5.1. We recall also in Table 5.2 the runtime estimation of binary-tree-based

Algorithm 1 Parallel TSLU.

Require: S is the set of P processors, $i \in S$ is my processor's index.
Require: All-reduction tree with height H.
Require: The $m \times b$ input matrix $A(:, 1:b)$ is distributed using a 1-D block row layout; $A_{i,0}$ is the block of rows belonging to my processor i.
1: Compute $\Pi_{i,0}A_{i,0} = L_{i,0}U_{i,0}$ using GEPP.
2: **for** k from 1 to H **do**
3: **if** I have any neighbors in the all-reduction tree at this level **then**
4: Let q be the number of neighbors.
5: Send $(\Pi_{i,k-1}A_{i,k-1})(1:b, 1:b)$ to each neighbor j
6: Receive $(\Pi_{j,k-1}A_{j,k-1})(1:b, 1:b)$ from each neighbor j
7: Form the matrix $A_{i,k}$ of size $qb \times b$ by stacking the matrices $(\Pi_{j,k-1}A_{j,k-1})(1:b, 1:b)$ from all neighbors.
8: Compute $\Pi_{i,k}A_{i,k} = L_{i,k}U_{i,k}$ using GEPP.
9: **else**
10: $A_{i,k} := \Pi_{i,k-1}A_{i,k-1}$
11: $\Pi_{i,k} := I_{b\times b}$
12: **end if**
13: **end for**
14: Compute the final permutation $\bar{\Pi} = \bar{\Pi}_H; \ldots; \bar{\Pi}_1\bar{\Pi}_0$, where $\bar{\Pi}_i$ represents the permutation matrix corresponding to each level in the reduction tree, formed by the permutation matrices of the nodes at this level extended by appropriate identity matrices to the dimension $m \times m$.
15: Compute the Gaussian elimination with no pivoting of $(\bar{\Pi}A)(:, 1:b) = LU$
Ensure: $U_{i,H}$ is the U factor obtained at step (15), for all processors $i \in S$.

TABLE 5.1
Performance models of parallel and sequential TSLU for "tall-skinny" matrices of size $m \times b$, with $m \gg b$. Parallel TSLU uses a binary tree and sequential TSLU uses a flat tree. Some lower order terms are omitted.

Algorithm	# flops	# messages	# words
Par. TSLU	$\frac{2mb^2}{P} + \frac{b^3}{3}(5\log_2 P - 1)$	$\log_2 P$	$b^2 \log_2 P$
Seq. TSLU var. 1	$2mb^2 - \frac{b^3}{3}$	$\frac{3mb}{M-b^2} + b$	$3mb + 2b^2$
Seq. TSLU var. 2	$2mb^2 - \frac{b^3}{3}$	$\frac{5mb}{M-b^2}$	$5mb$

parallel CALU for an $m \times n$ matrix distributed using a two-dimensional (2-D) block cyclic layout. More details and the algorithm are described in [11]. The panel factorization is performed using binary-tree-based TSLU. The other steps of parallel CALU are similar to the PDGETRF routine in ScaLAPACK that implements Gaussian elimination with partial pivoting.

We now analyze briefly flat-tree-based sequential TSLU and sequential CALU, for which the runtime estimations are presented in Tables 5.1 and 5.2, respectively. A more detailed study is presented in section 5 and Appendix A of the technical report on which this paper is based [12]. Sequential TSLU based on a flat tree consists of reading in the memory of size M blocks of the input matrix that fit in memory and that are as large as possible. For this, the matrix is considered to be partitioned in blocks of size $b_1 \times b$, where $b_1 \geq b$ is chosen such that a $b \times b$ matrix and a block fits in memory; that is, $b^2 + b_1 b \leq M$. We assume that the blocks are stored in contiguous memory. We have $b_1 \approx (M - b^2)/b = M_1/b$, with $M_1 = M - b^2$, and $M_1 \geq M/2$. The

TABLE 5.2

Performance models of parallel (binary-tree-based) and sequential (flat-tree-based) CALU and PDGETRF routine when factoring an $m \times n$ matrix, $m \geq n$. For parallel CALU and PDGETRF, the input matrix is distributed in a 2-D block cyclic layout on a $P_r \times P_c$ grid of processors with square $b \times b$ blocks. For sequential CALU, the matrix is partitioned into $P = \frac{3mn}{M}$ blocks. Some lower order terms are omitted.

	Parallel CALU
# messages	$\frac{3n}{b}\log_2 P_r + \frac{3n}{b}\log_2 P_c$
# words	$\left(nb + \frac{3n^2}{2P_c}\right)\log_2 P_r + \frac{1}{P_r}\left(mn - \frac{n^2}{2}\right)\log_2 P_c$
# flops	$\frac{1}{P}\left(mn^2 - \frac{n^3}{3}\right) + \frac{1}{P_r}\left(2mn - n^2\right)b + \frac{n^2 b}{2P_c} + \frac{nb^2}{3}(5\log_2 P_r - 1)$
	PDGETRF
# messages	$2n\left(1 + \frac{2}{b}\right)\log_2 P_r + \frac{3n}{b}\log_2 P_c$
# words	$\left(\frac{nb}{2} + \frac{3n^2}{2P_c}\right)\log_2 P_r + \log_2 P_c \frac{1}{P_r}\left(mn - \frac{n^2}{2}\right)$
# flops	$\frac{1}{P}\left(mn^2 - \frac{n^3}{3}\right) + \frac{1}{P_r}\left(mn - \frac{n^2}{2}\right)b + \frac{n^2 b}{2P_c}$
	Sequential CALU
# messages	$\frac{15\sqrt{3}mn^2}{2M^{3/2}} + \frac{15mn}{2M}$
# words	$\frac{5\sqrt{3}mn^2}{2\sqrt{M}} - \frac{5\sqrt{3}n^3}{6\sqrt{M}} + 5\left(mn - \frac{n^2}{2}\right)$
# flops	$mn^2 - \frac{n^3}{3} + \frac{2}{\sqrt{3}}mn\sqrt{M} - \frac{1}{\sqrt{3}}n^2\sqrt{M}$

preprocessing step of TSLU starts by performing GEPP of the first block to select b rows that are kept in memory. Then the following blocks of the matrix are read. For each block, a new set of b rows is selected by computing GEPP on the previously selected rows and this new block. Thus, in the preprocessing step the matrix is read once, using $mb/b_1 b = mb/M_1$ messages. At the end of the preprocessing step, the b pivot rows are in fast memory, and the pivoting needs to be applied on the matrix. The rows that are in the first b positions and are not used as pivots need to be stored at different locations in memory. These rows can be read in fast memory using one message, since $M > b^2$. Two approaches can be used for writing the rows back to slow memory at their new positions. The first approach consists of using one message for writing each row. We refer to this approach in Table 5.1 as *Seq. TSLU var.* 1. The second approach consists of permuting rows by reading in fast memory and writing back in slow memory blocks of the matrix that are as large as possible; that is, of size $mb/(M - b^2)$. At most the whole matrix is read and written once. We refer to this approach in Table 5.1 as *Seq. TSLU var.* 2. This approach can lead to fewer number of messages exchanged, at the cost of more words transferred, in particular when $b > mb/(M - b^2)$. During the LU factorization with no pivoting, the matrix is read and written once. This leads to $2mb/(M - b^2)$ messages.

For sequential CALU, we consider that the matrix is partitioned into $P = P_r \times P_c$ blocks (here P does not refer to the number of processors, the algorithm is executed on one processor) and we analyze a right-looking algorithm. Following the approach of sequential CAQR discussed in [6], we impose that three square blocks fit in fast memory; that is, $P = \frac{3mn}{M}$. This is necessary for performing the updates on the trailing matrix, when three blocks are necessary. We then have $P_c = \frac{\sqrt{3}n}{\sqrt{M}}$, $P_r = \frac{\sqrt{3}m}{\sqrt{M}}$, and $M_1 = \frac{2M}{3}$. With this choice, the runtime of sequential CALU is presented in Table 5.2.

6. Lower bounds on communication. In this section we discuss the optimality of CALU in terms of communication. We first recall communication complexity

bounds for dense matrix multiplication and dense LU factorization. A lower bound on the volume of communication for the multiplication of two square dense matrices of size $n \times n$ using a $O(n^3)$ sequential algorithm (not Strassen like) was introduced first by Hong and Kung [17] in the sequential case. A simpler proof and its extension to the parallel case is presented by Irony, Toledo, and Tiskin in [18]. By using the simple fact that the size of each message is limited by the size of the memory, a lower bound on the number of messages can be deduced [6]. Memory here refers to fast memory in the sequential case and to local memory of a processor in the parallel case.

It is shown in [6] that the lower bounds for matrix multiplication presented in [17, 18] represent lower bounds for LU decomposition, using the following reduction of matrix multiplication to LU:

$$\begin{pmatrix} I & & -B \\ A & I & \\ & & I \end{pmatrix} = \begin{pmatrix} I & & \\ A & I & \\ & & I \end{pmatrix} \begin{pmatrix} I & & -B \\ & I & A \cdot B \\ & & I \end{pmatrix}.$$

Consider a matrix of size $m \times n$ and its LU decomposition. On a sequential machine with fast memory of size M, a lower bound on the number of words and on the number of messages communicated between fast and slow memory during its LU decomposition is

$$\#\text{ words} \geq \Omega\left(\frac{mn^2}{\sqrt{M}}\right), \quad \#\text{ messages} \geq \Omega\left(\frac{mn^2}{M^{3/2}}\right). \tag{6.1}$$

On a parallel machine, it is considered that the size of the local memory of each processor is on the order of $O(mn/P)$ words and the number of flops the algorithm performs is at least $(mn^2 - n^3)/P$ [6]. This leads to a lower bound on the number of words and number of messages that at least one of the processors must communicate during the LU decomposition of

$$\#\text{ words} \geq \Omega\left(\sqrt{\frac{mn^3}{P}}\right), \quad \#\text{ messages} \geq \Omega\left(\sqrt{\frac{nP}{m}}\right). \tag{6.2}$$

In the following we show that CALU attains the lower bounds on communication. We first discuss sequential TSLU and CALU, whose performance models are shown in Tables 5.1 and 5.2. Sequential TSLU is optimal in terms of communication, modulo constant factors. The number of messages exchanged is $O(mb/M)$; that is, on the order of the number of messages necessary to read the matrix. The volume of communication is $O(mb)$; that is, on the order of the size of the matrix. Sequential CALU attains the lower bounds on communication, modulo constant factors, in terms of both number of messages and volume of communication.

In contrast to CALU, previous algorithms do not always minimize communication. We discuss here two algorithms, recursive LU [24, 13] and LAPACK's DGETRF [1]. An analysis similar to the one performed for recursive QR [6] shows that recursive LU minimizes the number of words communicated, but it does not always attain the lower bound for the number of messages. More details can be found in [12].

DGETRF uses a block algorithm to implement Gaussian elimination with partial pivoting. As for sequential CALU, we consider that the matrix is partitioned into $P_r \times P_c$ blocks, with $P = P_r \cdot P_c$, and we analyze a right-looking variant of the

algorithm. With this partitioning, each block is of size $m/P_r \times n/P_c$, and P, P_r, P_c do not refer to the number of processors, since the algorithm is sequential. The LAPACK implementation of DGETRF refers to n/P_c as the "block size." The size of the blocks is chosen such that three blocks fit in memory; that is, $3mn/P \leq M$. The total number of words communicated in DGETRF is

$$\#words_{DGETRF} = \begin{cases} O\left(\frac{mn^2}{P_c}\right) + O\left(mnP_c\right) & \text{if } m > M; \\ O\left(\frac{mn^2}{P_c}\right) + O\left(mnP_c\right) & \text{if } m \leq M \text{ and } \frac{mn}{P_c} > M; \\ O\left(mn\right) + O\left(mnP_c\right) & \text{if } m \leq M \text{ and } \frac{mn}{P_c} \leq M; \\ O\left(mn\right) & \text{if } mn \leq M. \end{cases}$$

In the first case, $m > M$, one column does not fit in memory. We choose $P_c = \sqrt{3}n/\sqrt{M}$, and $P_r = \sqrt{3}m/\sqrt{M}$. The number of words communicated is $O(mn^2/\sqrt{M})$. In this case DGETRF attains the lower bound on the number of words. In the second case, at least one column fits in memory, but P_c is such that the panel does not fit in memory. The number of words communicated is minimized by choosing $P_c = \sqrt{n}$, and so the amount of words communicated is $O(mn^{1.5})$. It exceeds the lower bound by a factor of $\sqrt{M}/\sqrt{n}$, when $M > n$. In the third case, P_c is chosen such that the panel fits in memory; that is, $P_c = mn/M$. Then the number of words communicated is $O(m^2n^2/M)$, which exceeds the lower bound by a factor of $m/\sqrt{M}$. In the last case the whole matrix fits in memory, and this case is trivial.

DGETRF does not always minimize the number of messages. Consider the case $m > M$, when the matrix is partitioned in square blocks of size mn/P, such that the number of words communicated is reduced. The panel factorization involves a total of $O(nP_r) = O(\frac{mn}{\sqrt{M}})$ messages exchanged between slow and fast memory. If $M = O(n)$, this term attains the lower bound on number of messages. But not if $n < M$.

We discuss now parallel CALU, whose performance model is presented in Table 5.2. To attain the communication bounds presented in (6.2), we need to choose an optimal layout; that is, optimal values for P_r, P_c, and b. We choose the same layout as optimal CAQR in [6]:

$$P_r = \sqrt{\frac{mP}{n}}, \quad P_c = \sqrt{\frac{nP}{m}}, \quad \text{and } b = \log^{-2}\left(\sqrt{\frac{mP}{n}}\right) \cdot \sqrt{\frac{mn}{P}}. \tag{6.3}$$

The idea behind this layout is to choose b close to its maximum value, such that the lower bound on the number of messages is attained, modulo polylog factors. In the same time, the number of extra floating point operations performed due to this choice of b represent a lower order term. With this layout, the performance of parallel CALU is given in Table 6.1. It attains the lower bounds on both the number of words and the number of messages, modulo polylog factors.

We now compare CALU to parallel GEPP as for example implemented in the routine PDGETRF of ScaLAPACK. Both algorithms communicate the same number of words. But the number of messages communicated by CALU is smaller by a factor of the order of b (depending P_r and P_c) than PDGETRF. This is because PDGETRF has an $O(n \log P)$ term in the number of messages due to partial pivoting. Hence PDGETRF does not attain the lower bound on the number of messages.

TABLE 6.1

Performance models of parallel (binary-tree-based) CALU with optimal layout. The matrix factored is of size $m \times n$, $m \geq n$, and $n \times n$. The values of P_r, P_c, and b used in the optimal layout are presented in (6.3)*. Some lower order terms are omitted.*

	Parallel CALU with optimal layout	Lower bound
	Input matrix of size $m \times n$	
# messages	$3\sqrt{\frac{nP}{m}} log^2\left(\sqrt{\frac{mP}{n}}\right) \log P$	$\Omega\left(\sqrt{\frac{nP}{m}}\right)$
# words	$\sqrt{\frac{mn^3}{P}}\left(log^{-1}\left(\sqrt{\frac{mP}{n}}\right) + \log\frac{P^2m}{n}\right)$	$\Omega\left(\sqrt{\frac{mn^3}{P}}\right)$
# flops	$\frac{1}{P}\left(mn^2 - \frac{n^3}{3}\right) + \frac{5mn^2}{2Plog^2\left(\sqrt{\frac{mP}{n}}\right)} + \frac{5mn^2}{3Plog^3\left(\sqrt{\frac{mP}{n}}\right)}$	$\frac{1}{P}\left(mn^2 - \frac{n^3}{3}\right)$
	Input matrix of size $n \times n$	
# messages	$3\sqrt{P}\log^3 P$	$\Omega\left(\sqrt{P}\right)$
# words	$\frac{n^2}{\sqrt{P}}\left(2\log^{-1} P + 1.25\log P\right)$	$\Omega\left(\frac{n^2}{\sqrt{P}}\right)$
# flops	$\frac{1}{P}\frac{2n^3}{3} + \frac{5n^3}{2P\log^2 P} + \frac{5n^3}{3P\log^3 P}$	$\frac{1}{P}\left(mn^2 - \frac{n^3}{3}\right)$

7. Conclusions. This paper studies CALU, a communication optimal LU factorization algorithm. The main part focuses on showing that CALU is stable in practice. First, we show that the growth factor of CALU is equivalent to performing GEPP on a larger matrix, whose entries are the same as the entries of the input matrix (slightly perturbed) and zeros. Since GEPP is stable in practice, we expect CALU to be also stable in practice. Second, extensive experiments show that CALU leads to results of (almost) the same order of magnitude as GEPP.

The paper also discusses briefly parallel and sequential algorithms for TSLU and CALU and their performance models. We show in particular that binary-tree-based TSLU and CALU minimize communication between processors of a parallel machine, and flat-tree-based TSLU and CALU minimize communication between slow and fast memory of a sequential machine.

Two main directions are followed in our future work. The first direction focuses on using a more stable factorization at each node of the reduction tree of CALU. The goal is to decrease the upper bound of the growth factor of CALU. The second direction focuses on the design and implementation of CALU on real machines that are formed by multiple levels of memory hierarchy and heterogeneous parallel units. We are interested in developing algorithms that are optimal over multiple levels of the memory hierarchy and over different levels of parallelism and implement them.

Appendix A. We present experimental results for binary-tree-based CALU and flat-tree-based CALU, and we compare them with GEPP. We show results obtained for the LU decomposition and the linear solver.

Tables A.1 and A.2 display the results obtained for random matrices. They show the growth factors, the threshold, the norm of the factors L and U and their inverses, and the relative error of the decomposition.

Tables A.3, A.4, and A.5 display results obtained for the special matrices presented in Table A.6. We include in our set sparse matrices. The size of the tested matrices is $n = 4096$. For binary-tree-based CALU we use $P = 64$, $b = 8$, and for flat-tree-based CALU we use $b = 8$. With $n = 4096$ and $b = 8$, this means the flat tree has $4096/8 = 512$ leaf nodes and its height is 511. When iterative refinement fails to reduce the componentwise backward error to the order of 10^{-16}, we indicate the number of iterations done before failing to converge and stopping by putting it in parentheses.

TABLE A.1
Stability of the LU decomposition for binary-tree-based CALU and GEPP on random matrices.

Binary-tree-based CALU												
n	P	b	g_W	g_D	g_T	τ_{ave}	τ_{min}	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
8192	256	32	8.5E+1	1.1E+2	4.9E+2	0.84	0.40	3.6E+3	3.3E+3	2.0E+5	2.4E+2	1.1E-13
		16	8.9E+1	1.1E+2	5.2E+2	0.86	0.37	3.7E+3	3.3E+3	2.0E+5	9.0E+2	1.1E-13
	128	64	8.6E+1	9.8E+1	4.7E+2	0.84	0.42	3.1E+3	3.2E+3	2.0E+5	4.2E+2	1.1E-13
		32	9.0E+1	1.2E+2	5.1E+2	0.84	0.38	3.5E+3	3.3E+3	2.0E+5	2.2E+2	1.1E-13
		16	8.5E+1	1.1E+2	4.9E+2	0.86	0.37	4.1E+3	3.2E+3	2.0E+5	5.1E+2	1.1E-13
	64	128	7.2E+1	8.8E+1	3.9E+2	0.85	0.47	2.9E+3	3.1E+3	1.9E+5	4.6E+2	1.0E-13
		64	8.2E+1	9.4E+1	4.7E+2	0.84	0.44	3.2E+3	3.2E+3	1.9E+5	2.2E+2	1.1E-13
		32	7.4E+1	9.9E+1	4.3E+2	0.84	0.40	3.3E+3	3.3E+3	2.0E+5	3.0E+2	1.1E-13
		16	8.3E+1	1.1E+2	5.0E+2	0.86	0.35	3.9E+3	3.2E+3	2.0E+5	6.8E+2	1.1E-13
4096	256	16	6.2E+1	7.5E+1	3.5E+2	0.87	0.41	1.7E+3	1.7E+3	7.4E+4	4.4E+2	5.6E-14
	128	32	5.3E+1	7.3E+1	3.0E+2	0.86	0.40	1.7E+3	1.7E+3	7.5E+4	3.2E+2	5.7E-14
		16	7.3E+1	9.0E+1	3.9E+2	0.87	0.38	1.9E+3	1.7E+3	7.4E+4	3.5E+2	5.7E-14
	64	64	5.5E+1	7.0E+1	2.9E+2	0.86	0.46	1.5E+3	1.7E+3	7.1E+4	4.3E+2	5.6E-14
		32	5.2E+1	6.8E+1	3.0E+2	0.86	0.41	1.7E+3	1.7E+3	7.5E+4	1.7E+2	5.8E-14
		16	5.4E+1	6.8E+1	3.1E+2	0.88	0.39	1.7E+3	1.7E+3	7.4E+4	1.5E+3	5.6E-14
2048	128	16	4.1E+1	4.8E+1	2.1E+2	0.89	0.41	8.9E+2	8.7E+2	2.7E+4	3.5E+2	2.9E-14
	64	32	3.6E+1	4.7E+1	1.9E+2	0.88	0.46	7.8E+2	9.1E+2	2.7E+4	1.6E+2	2.9E-14
		16	3.7E+1	4.7E+1	1.9E+2	0.89	0.40	9.0E+2	8.8E+2	2.7E+4	1.4E+2	2.9E-14
1024	64	16	2.4E+1	3.4E+1	1.2E+2	0.90	0.43	4.7E+2	4.7E+2	1.0E+4	3.1E+2	1.4E-14
GEPP												
8192	-		5.5E+1	7.6E+1	3.0E+2	1	1	1.9E+3	2.6E+3	8.7E+3	6.0E+2	7.2E-14
4096	-		3.6E+1	5.1E+1	2.0E+2	1	1	1.0E+3	1.4E+3	2.3E+4	1.9E+2	3.9E-14
2048	-		2.6E+1	3.6E+1	1.4E+2	1	1	5.5E+2	7.4E+2	6.1E+4	1.8E+2	2.0E-14
1024	-		1.8E+1	2.5E+1	9.3E+1	1	1	2.8E+2	4.1E+2	1.6E+5	4.3E+2	1.1E-14

TABLE A.2
Stability of the LU decomposition for flat-tree-based CALU on random matrices.

n	b	g_W	g_D	g_T	τ_{ave}	τ_{min}	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\|\|U\|\|_1$	$\|\|U^{-1}\|\|_1$	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$
8192	4	7.0E+1	9.5E+1	3.8E+2	0.97	0.44	2.7E+3	2.7E+3	1.7E+5	3.5E+2	7.7E-14
	8	8.4E+1	1.1E+2	4.8E+2	0.93	0.42	3.2E+3	2.9E+3	1.8E+5	7.3E+2	8.6E-14
	16	1.0E+2	1.1E+2	5.8E+2	0.87	0.36	3.6E+3	3.1E+3	1.9E+5	2.1E+2	1.0E-13
	32	1.0E+2	1.1E+2	5.7E+2	0.81	0.35	4.0E+3	3.4E+3	2.0E+5	3.9E+2	1.2E-13
	64	9.6E+1	1.2E+2	5.5E+2	0.80	0.38	3.6E+3	3.3E+3	2.0E+5	1.6E+3	1.2E-13
4096	4	4.7E+1	6.4E+1	2.6E+2	0.97	0.48	1.2E+3	1.4E+3	6.3E+4	1.5E+2	4.1E-14
	8	5.8E+1	7.1E+1	3.1E+2	0.93	0.40	1.5E+3	1.5E+3	6.8E+4	1.1E+2	4.6E-14
	16	6.2E+1	7.0E+1	3.6E+2	0.88	0.35	2.2E+3	1.7E+3	7.1E+4	3.4E+2	5.4E-14
	32	7.2E+1	7.9E+1	3.9E+2	0.83	0.37	1.9E+3	1.7E+3	7.6E+4	3.1E+2	6.2E-14
	64	5.0E+1	6.1E+1	2.8E+2	0.83	0.42	1.7E+3	1.7E+3	7.1E+4	4.9E+2	5.9E-14
2048	4	3.2E+1	4.1E+1	1.7E+2	0.97	0.51	7.2E+2	7.7E+2	2.5E+4	6.2E+2	2.2E-14
	8	3.5E+1	4.9E+1	1.8E+2	0.93	0.43	8.4E+2	8.2E+2	2.6E+4	1.7E+2	2.4E-14
	16	3.8E+1	5.0E+1	2.0E+2	0.88	0.35	9.8E+2	8.9E+2	2.7E+4	1.0E+3	2.9E-14
	32	3.7E+1	4.5E+1	1.9E+2	0.85	0.40	8.7E+2	8.9E+2	2.7E+4	8.8E+2	3.1E-14
	64	3.7E+1	4.8E+1	1.9E+2	0.87	0.45	8.6E+2	8.7E+2	2.7E+4	2.2E+2	2.9E-14
1024	4	2.4E+1	2.8E+1	1.2E+2	0.97	0.48	3.5E+2	4.2E+2	9.2E+3	1.9E+2	1.1E-14
	8	2.8E+1	3.4E+1	1.4E+2	0.94	0.42	4.5E+2	4.4E+2	9.9E+3	1.7E+2	1.3E-14
	16	2.8E+1	3.4E+1	1.4E+2	0.90	0.39	4.7E+2	4.7E+2	9.9E+3	3.6E+2	1.4E-14
	32	2.5E+1	3.3E+1	1.3E+2	0.88	0.44	4.4E+2	4.6E+2	9.9E+3	3.2E+2	1.5E-14
	64	2.2E+1	2.8E+1	1.1E+2	0.91	0.50	3.9E+2	4.5E+2	9.7E+3	3.2E+2	1.4E-14

TABLE A.3
Stability results for GEPP on special matrices.

	Matrix	cond(A,2)	g_W	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij}\|U_{ij}\|$	$\min_{kk}\|U_{kk}\|$	cond(U,1)	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
well-conditioned	hadamard	1.0E+0	4.1E+3	4.1E+3	4.1E+3	4.1E+3	1.0E+0	5.3E+5	0.0E+0	3.3E-16	4.6E-15	2
	house	1.0E+0	5.1E+0	8.9E+2	2.6E+2	5.1E+0	5.7E-2	1.4E+4	2.0E-15	5.6E-17	6.3E-15	3
	parter	4.8E+0	1.6E+0	4.8E+1	2.0E+0	3.1E+0	2.0E+0	2.3E+2	2.3E-15	8.3E-16	4.4E-15	3
	ris	4.8E+0	1.6E+0	4.8E+1	2.0E+0	1.6E+0	1.0E+0	2.3E+2	2.3E-15	7.1E-16	4.7E-15	2
	kms	9.1E+0	1.0E+0	2.0E+0	1.5E+0	1.0E+0	7.5E-1	3.0E+0	2.0E-16	1.1E-16	6.7E-16	1
	toeppen	1.0E+1	1.1E+0	2.1E+0	9.0E+0	1.1E+1	1.0E+1	3.3E+1	1.1E-17	7.2E-17	3.0E-15	1
	condex	1.0E+2	1.0E+0	2.0E+0	5.6E+0	1.0E+2	1.0E+0	7.8E+2	1.8E-15	9.7E-16	6.8E-15	3
	moler	1.9E+2	1.0E+0	2.2E+1	2.0E+0	1.0E+0	1.0E+0	4.4E+1	3.8E-14	2.6E-16	1.7E-15	2
	circul	3.7E+2	1.8E+2	1.0E+3	1.4E+3	6.4E+2	3.4E+0	1.2E+6	4.3E-14	2.1E-15	1.2E-14	1
	randcorr	1.4E+3	1.0E+0	3.1E+1	5.7E+1	1.0E+0	2.3E-1	5.0E+4	1.6E-15	7.8E-17	8.0E-16	1
	poisson	1.7E+3	1.0E+0	2.0E+0	3.4E+1	4.0E+0	3.2E+0	7.8E+1	2.8E-16	1.4E-16	7.5E-16	1
	hankel	2.9E+3	6.2E+1	9.8E+2	1.5E+3	2.4E+2	4.5E+0	2.0E+6	4.2E-14	2.5E-15	1.6E-14	2
	jordbloc	5.2E+3	1.0E+0	1.0E+0	1.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	2.0E-17	8.3E-17	0
	compan	7.5E+3	1.0E+0	2.0E+0	4.0E+0	7.9E+0	2.6E-1	7.8E+1	0.0E+0	2.0E-17	6.2E-13	1
	pei	1.0E+4	1.0E+0	4.1E+3	9.8E+0	1.0E+0	3.9E-1	2.5E+1	7.0E-16	6.6E-18	2.3E-17	0
	randcolu	1.5E+4	4.6E+1	9.9E+2	1.4E+3	3.2E+0	5.6E-2	1.1E+7	4.0E-14	2.3E-15	1.4E-14	1
	sprandn	1.6E+4	7.4E+0	7.4E+2	1.5E+3	2.9E+1	1.7E+0	1.3E+7	3.4E-14	8.5E-15	9.3E-14	2
	riemann	1.9E+4	1.0E+0	4.1E+3	3.5E+0	4.1E+3	1.0E+0	2.6E+6	5.7E-19	2.0E-16	1.7E-15	2
	compar	1.8E+6	2.3E+1	9.8E+2	1.4E+3	1.1E+2	3.1E+0	2.7E+7	2.3E-14	1.2E-15	8.8E-15	1
	tridiag	6.8E+6	1.0E+0	2.0E+0	1.5E+3	2.0E+0	1.0E+0	5.1E+3	1.4E-18	2.6E-17	1.2E-16	0
	chebspec	1.3E+7	1.0E+0	5.4E+1	9.2E+0	7.1E+6	1.5E+3	4.2E+7	1.8E-15	2.9E-18	1.6E-15	1
	lehmer	1.8E+7	1.0E+0	1.5E+3	2.0E+0	1.0E+0	4.9E-4	8.2E+3	1.5E-15	2.8E-17	1.7E-16	0
	toeppd	2.1E+7	1.0E+0	4.2E+1	9.8E+2	2.0E+3	2.9E+2	1.3E+6	1.5E-15	5.0E-17	3.3E-16	1
	minij	2.7E+7	1.0E+0	4.1E+3	2.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	7.8E-19	4.2E-18	0
	randsvd	6.7E+7	4.7E+0	9.9E+2	1.4E+3	6.4E-2	3.6E-7	1.4E+10	5.6E-15	3.4E-16	2.0E-15	2
	forsythe	6.7E+7	1.0E+0	1.0E+0	1.0E+0	1.0E+0	1.5E-8	6.7E+7	0.0E+0	0.0E+0	0.0E+0	0
ill-conditioned	fiedler	2.5E+10	1.0E+0	1.7E+3	1.5E+1	7.9E+0	4.1E-7	2.9E+8	1.6E-16	3.3E-17	1.0E-15	1
	dorr	7.4E+10	1.0E+0	2.0E+0	3.1E+2	3.4E+5	1.3E+0	1.7E+11	6.0E-18	2.3E-17	2.2E-15	1
	demmel	1.0E+14	2.5E+0	1.2E+2	1.4E+2	1.6E+14	7.8E+3	1.7E+17	3.7E-15	7.1E-21	4.8E-9	2
	chebvand	3.8E+19	2.0E+2	2.2E+3	3.1E+3	1.8E+2	9.0E-10	4.8E+22	5.1E-14	3.3E-17	2.6E-16	1
	invhess	4.1E+19	2.0E+0	4.1E+3	2.0E+0	5.4E+0	4.9E-4	3.0E+48	1.2E-14	1.7E-17	2.4E-14	(1)
	prolate	1.4E+20	1.2E+1	1.4E+3	4.6E+3	5.3E+0	5.9E-13	4.7E+23	1.6E-14	4.7E-16	6.3E-15	(1)
	frank	1.7E+20	1.0E+0	2.0E+0	2.0E+0	4.1E+3	5.9E-24	1.9E+30	2.2E-18	4.9E-27	1.2E-23	0
	cauchy	5.5E+21	1.0E+0	3.1E+2	1.9E+2	1.0E+7	2.3E-15	2.1E+24	1.4E-15	6.1E-19	5.2E-15	(1)
	hilb	8.0E+21	1.0E+0	3.1E+3	1.3E+3	1.0E+0	4.2E-20	2.2E+22	2.2E-16	6.0E-19	2.0E-17	0
	lotkin	5.4E+22	1.0E+0	2.6E+3	1.3E+3	1.0E+0	3.6E-19	2.3E+22	8.0E-17	3.0E-18	2.3E-15	(1)
	kahan	1.1E+28	1.0E+0	1.0E+0	1.0E+0	1.0E+0	2.2E-13	4.1E+53	0.0E+0	9.7E-18	4.3E-16	1

TABLE A.4
Stability of CALU based on a binary tree for special matrices.

Matrix	g_W	τ_{ave}	$\tau_{\min}$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij}\|U_{ij}\|$	$\min_{kk}\|U_{kk}\|$	cond(U,1)	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
hadamard	4.1E+3	1.00	1.00	4.1E+3	3.8E+3	4.1E+3	1.0E+0	1.2E+6	0.0E+0	2.9E-16	3.7E-15	2
house	5.1E+0	1.00	1.00	8.9E+2	2.6E+2	5.1E+0	5.7E-2	1.4E+4	2.0E-15	5.6E-17	6.8E-15	3
parter	1.6E+0	1.00	1.00	4.8E+1	2.0E+0	3.1E+0	2.0E+0	2.3E+2	2.3E-15	7.5E-16	4.1E-15	3
ris	1.6E+0	1.00	1.00	4.8E+1	2.0E+0	1.6E+0	1.0E+0	2.3E+2	2.3E-15	8.0E-16	4.2E-15	3
kms	1.0E+0	1.00	1.00	2.0E+0	1.5E+0	1.0E+0	7.5E-1	3.0E+0	2.0E-16	1.1E-16	5.9E-16	1
toeppen	1.1E+0	1.00	1.00	2.1E+0	9.0E+0	1.1E+1	1.0E+1	3.3E+1	1.1E-17	7.1E-17	1.3E-15	1
condex	1.0E+0	1.00	1.00	2.0E+0	5.6E+0	1.0E+2	1.0E+0	7.8E+2	1.8E-15	9.4E-16	4.8E-15	3
moler	1.0E+0	1.00	1.00	2.2E+1	2.0E+0	1.0E+0	1.0E+0	4.4E+1	3.8E-14	2.7E-16	1.8E-15	3
circul	2.3E+2	0.91	0.41	1.8E+3	1.7E+3	7.6E+2	3.1E+0	2.0E+6	5.7E-14	2.8E-15	1.6E-14	1
randcorr	1.0E+0	1.00	1.00	3.1E+1	5.7E+1	1.0E+0	2.3E-1	5.0E+4	1.6E-15	7.7E-17	7.7E-16	1
poisson	1.0E+0	1.00	1.00	2.0E+0	3.4E+1	4.0E+0	3.2E+0	7.8E+1	2.8E-16	1.4E-16	9.8E-16	1
hankel	9.3E+1	0.92	0.42	1.8E+3	1.7E+3	4.3E+2	2.5E+0	2.3E+6	5.3E-14	3.7E-15	2.2E-14	2
jordbloc	1.0E+0	1.00	1.00	1.0E+0	1.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	2.0E-17	8.8E-17	0
compan	1.0E+0	1.00	1.00	2.0E+0	4.0E+0	7.9E+0	2.6E-1	7.8E+1	0.0E+0	9.9E-18	4.0E-14	1
pei	1.0E+0	1.00	1.00	4.1E+3	9.8E+0	1.0E+0	3.9E-1	2.5E+1	7.0E-16	3.6E-17	4.7E-17	0
randcolu	4.7E+1	0.91	0.40	2.1E+3	1.6E+3	3.8E+0	4.8E-2	1.4E+7	5.2E-14	2.9E-15	1.8E-14	2
sprandn	8.0E+0	0.93	0.41	1.2E+3	1.8E+3	3.6E+1	1.4E+0	2.4E+7	4.5E-14	9.6E-15	1.4E-13	2
riemann	1.0E+0	1.00	1.00	4.1E+3	5.1E+2	4.1E+3	1.0E+0	1.7E+8	2.5E-18	1.1E-16	1.4E-15	2
compar	3.5E+1	0.91	0.42	1.7E+3	1.6E+3	1.8E+2	2.8E+0	3.3E+7	3.0E-14	1.7E-15	1.1E-14	1
tridiag	1.0E+0	1.00	1.00	2.0E+0	1.5E+3	2.0E+0	1.0E+0	5.1E+3	1.4E-18	2.5E-17	1.1E-16	0
chebspec	1.0E+0	1.00	1.00	5.4E+1	9.2E+0	7.1E+6	1.5E+3	4.2E+7	1.8E-15	3.2E-18	1.6E-15	1
lehmer	1.0E+0	1.00	0.78	1.9E+3	5.0E+2	1.0E+0	4.9E-4	1.7E+6	1.5E-15	1.8E-17	9.3E-17	0
toeppd	1.0E+0	1.00	1.00	4.2E+1	9.8E+2	2.0E+3	2.9E+2	1.3E+6	1.5E-15	5.1E-17	4.3E-16	1
minij	1.0E+0	1.00	1.00	4.1E+3	2.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	5.1E-19	3.5E-18	0
randsvd	8.3E+0	0.91	0.33	1.8E+3	1.6E+3	8.5E-2	3.0E-7	2.4E+10	7.4E-15	4.5E-16	2.5E-15	2
forsythe	1.0E+0	1.00	1.00	1.0E+0	1.0E+0	1.0E+0	1.5E-8	6.7E+7	0.0E+0	0.0E+0	0.0E+0	0
fiedler	1.0E+0	1.00	0.90	1.7E+3	1.5E+1	7.9E+0	4.1E-7	2.9E+8	1.6E-16	3.5E-17	6.4E-16	1
dorr	1.0E+0	1.00	1.00	2.0E+0	3.1E+2	3.4E+5	1.3E+0	1.7E+11	6.0E-18	2.6E-17	1.4E-15	1
demmel	2.8E+0	0.98	0.38	1.3E+2	1.3E+2	2.2E+14	6.2E+3	2.0E+17	3.8E-15	1.1E-20	9.8E-9	3
chebvand	3.1E+2	0.91	0.42	2.2E+3	3.4E+3	2.3E+2	8.4E-10	3.4E+23	6.6E-14	3.7E-17	3.2E-16	1
invhess	2.0E+0	1.00	1.00	4.1E+3	2.0E+0	5.4E+0	4.9E-4	3.0E+48	1.2E-14	1.2E-16	7.1E-14	(2)
prolate	1.7E+1	0.95	0.39	1.6E+3	5.8E+3	7.5E+0	6.6E-12	1.4E+23	2.0E-14	5.1E-16	9.1E-15	(1)
frank	1.0E+0	1.00	1.00	2.0E+0	2.0E+0	4.1E+3	5.9E-24	1.9E+30	2.2E-18	7.4E-28	1.8E-24	0
cauchy	1.0E+0	1.00	0.34	3.1E+2	2.0E+2	1.0E+7	2.3E-15	6.0E+24	1.4E-15	7.2E-19	7.4E-15	(1)
hilb	1.0E+0	0.92	0.37	3.2E+3	1.6E+3	1.0E+0	1.3E-19	1.8E+22	2.2E-16	5.5E-19	2.2E-17	0
lotkin	1.0E+0	0.93	0.48	2.7E+3	1.4E+3	1.0E+0	4.6E-19	7.5E+22	8.0E-17	2.2E-18	2.1E-16	0
kahan	1.0E+0	1.00	1.00	1.0E+0	1.0E+0	1.0E+0	2.2E-13	4.1E+53	0.0E+0	7.7E-18	2.1E-16	0

TABLE A.5
Stability of CALU based on a flat tree for special matrices.

Matrix	g_W	τ_{ave}	$\tau_{\min}$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max_{ij}\|U_{ij}\|$	$\min_{kk}\|U_{kk}\|$	cond(U,1)	$\frac{\|\|PA-LU\|\|_F}{\|\|A\|\|_F}$	η	w_b	N_{IR}
hadamard	4.1E+3	1.00	1.00	4.1E+3	4.1E+3	4.1E+3	1.0E+0	5.3E+5	0.0E+0	2.6E-16	2.6E-15	2
house	5.1E+0	1.00	1.00	8.9E+2	2.6E+2	5.1E+0	5.7E-2	1.4E+4	2.0E-15	7.1E-17	6.9E-15	3
parter	1.6E+0	1.00	1.00	4.8E+1	2.0E+0	3.1E+0	2.0E+0	2.3E+2	2.3E-15	7.3E-16	4.4E-15	3
ris	1.6E+0	1.00	1.00	4.8E+1	2.0E+0	1.6E+0	1.0E+0	2.3E+2	2.3E-15	7.2E-16	4.2E-15	2
kms	1.0E+0	1.00	1.00	2.0E+0	1.5E+0	1.0E+0	7.5E-1	3.0E+0	2.0E-16	1.0E-16	6.2E-16	1
toeppen	1.1E+0	1.00	1.00	2.1E+0	9.0E+0	1.1E+1	1.0E+1	3.3E+1	1.1E-17	6.9E-17	1.2E-15	2
condex	1.0E+0	1.00	1.00	2.0E+0	5.6E+0	1.0E+2	1.0E+0	7.8E+2	1.8E-15	9.1E-16	5.6E-15	3
moler	1.0E+0	1.00	1.00	2.2E+1	2.0E+0	1.0E+0	1.0E+0	4.4E+1	3.8E-14	2.7E-16	1.9E-15	1
circul	2.0E+2	0.93	0.39	1.6E+3	1.6E+3	6.3E+2	3.3E+0	2.6E+6	5.2E-14	2.7E-15	1.9E-14	2
randcorr	1.0E+0	1.00	1.00	3.1E+1	5.7E+1	1.0E+0	2.3E-1	5.0E+4	1.6E-15	7.6E-17	5.7E-16	1
poisson	1.0E+0	1.00	1.00	2.0E+0	3.4E+1	4.0E+0	3.2E+0	7.8E+1	2.8E-16	1.4E-16	1.1E-15	1
hankel	8.3E+1	0.93	0.40	1.7E+3	1.8E+3	3.4E+2	4.5E+0	2.8E+6	4.9E-14	3.2E-15	1.9E-14	2
jordbloc	1.0E+0	1.00	1.00	1.0E+0	1.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	2.0E-17	8.5E-17	0
compan	1.0E+0	1.00	1.00	2.0E+0	4.0E+0	7.9E+0	2.6E-1	7.8E+1	0.0E+0	1.5E-17	8.1E-13	1
pei	1.0E+0	1.00	1.00	4.1E+3	9.8E+0	1.0E+0	3.9E-1	2.5E+1	7.0E-16	4.6E-17	6.3E-17	0
randcolu	5.6E+1	0.93	0.30	2.1E+3	1.6E+3	4.8E+0	5.6E-2	1.4E+7	4.8E-14	2.6E-15	1.5E-14	2
sprandn	8.3E+0	0.94	0.41	1.2E+3	1.8E+3	4.0E+1	1.4E+0	2.4E+7	4.2E-14	1.0E-14	1.3E-13	2
riemann	1.0E+0	1.00	1.00	4.1E+3	3.5E+0	4.1E+3	1.0E+0	2.6E+6	5.7E-19	1.7E-16	1.6E-15	1
compar	2.9E+1	0.93	0.41	1.6E+3	1.5E+3	1.3E+2	2.8E+0	2.2E+7	2.8E-14	1.7E-15	1.1E-14	1
tridiag	1.0E+0	1.00	1.00	2.0E+0	1.5E+3	2.0E+0	1.0E+0	5.1E+3	1.4E-18	2.5E-17	1.1E-16	0
chebspec	1.0E+0	1.00	1.00	5.4E+1	9.2E+0	7.1E+6	1.5E+3	4.2E+7	1.8E-15	2.6E-18	1.6E-15	1
lehmer	1.0E+0	1.00	1.00	1.5E+3	2.0E+0	1.0E+0	4.9E-4	8.2E+3	1.5E-15	2.8E-17	1.9E-16	0
toeppd	1.0E+0	1.00	1.00	4.2E+1	9.8E+2	2.0E+3	2.9E+2	1.3E+6	1.5E-15	5.1E-17	3.2E-16	1
minij	1.0E+0	1.00	1.00	4.1E+3	2.0E+0	1.0E+0	1.0E+0	8.2E+3	0.0E+0	7.7E-19	4.6E-18	0
randsvd	6.1E+0	0.93	0.44	1.4E+3	1.5E+3	7.7E-2	3.2E-7	2.1E+10	6.6E-15	4.3E-16	2.3E-15	2
forsythe	1.0E+0	1.00	1.00	1.0E+0	1.0E+0	1.0E+0	1.5E-8	6.7E+7	0.0E+0	0.0E+0	0.0E+0	0
fiedler	1.0E+0	1.00	1.00	1.7E+3	1.5E+1	7.9E+0	4.1E-7	2.9E+8	1.6E-16	2.7E-17	7.0E-16	1
dorr	1.0E+0	1.00	1.00	2.0E+0	3.1E+2	3.4E+5	1.3E+0	1.7E+11	6.0E-18	2.6E-17	1.3E-15	1
demmel	2.0E+0	0.98	0.37	1.2E+2	1.3E+2	1.6E+14	7.4E+3	2.1E+17	4.0E-15	7.6E-21	1.5E-8	2
chebvand	3.2E+2	0.93	0.32	3.7E+3	3.2E+3	3.2E+2	9.1E-10	4.7E+23	6.2E-14	3.7E-17	2.7E-16	1
invhess	2.0E+0	1.00	1.00	4.1E+3	2.0E+0	5.4E+0	4.9E-4	3.0E+48	1.2E-14	4.4E-16	2.0E-13	(2)
prolate	1.9E+1	0.95	0.30	1.3E+3	4.7E+3	8.2E+0	1.2E-11	4.9E+22	2.3E-14	4.9E-16	7.4E-15	(1)
frank	1.0E+0	1.00	1.00	2.0E+0	2.0E+0	4.1E+3	5.9E-24	1.9E+30	2.2E-18	5.2E-27	1.2E-23	0
cauchy	1.0E+0	1.00	0.47	3.1E+2	2.1E+2	1.0E+7	2.3E-15	1.7E+24	1.5E-15	6.0E-19	2.9E-15	(1)
hilb	1.0E+0	0.93	0.24	3.0E+3	1.6E+3	1.0E+0	3.1E-19	2.7E+21	2.2E-16	5.9E-19	2.1E-17	0
lotkin	1.0E+0	0.93	0.44	2.6E+3	1.9E+3	1.0E+0	2.4E-19	4.3E+22	8.1E-17	3.4E-18	2.3E-15	(2)
kahan	1.0E+0	1.00	1.00	1.0E+0	1.0E+0	1.0E+0	2.2E-13	4.1E+53	0.0E+0	9.3E-18	3.1E-16	1

TABLE A.6 *Special matrices in our test set.*

No.	Matrix	Remarks
1	hadamard	Hadamard matrix. hadamard(n), where n, $n/12$, or $n/20$ is power of 2.
2	house	Householder matrix, A = eye(n) $-$ $\beta * v * v'$, where $[v, \beta, s]$ = gallery("house", randn(n, 1)).
3	parter	Parter matrix, a Toeplitz matrix with most of singular values near π. gallery("parter", n), or $A(i,j) = 1/(i-j+0.5)$.
4	ris	Ris matrix, matrix with elements $A(i,j) = 0.5/(n-i-j+1.5)$. The eigenvalues cluster around $-\pi/2$ and $\pi/2$. gallery("ris", n).
5	kms	Kac–Murdock–Szego Toeplitz matrix. Its inverse is tridiagonal. gallery("kms", n) or gallery("kms", n, rand).
6	toeppen	Pentadiagonal Toeplitz matrix (sparse).
7	condex	Counterexample matrix to condition estimators. gallery("condex", n).
8	moler	Moler matrix, a symmetric positive definite (spd) matrix. gallery("moler", n).
9	circul	Circulant matrix, gallery("circul", randn(n, 1)).
10	randcorr	Random $n \times n$ correlation matrix with random eigenvalues from a uniform distribution, a symmetric positive semidefinite matrix. gallery("randcorr", n).
11	poisson	Block tridiagonal matrix from Poisson's equation (sparse), A = gallery("poisson",sqrt(n)).
12	hankel	Hankel matrix, A = hankel(c, r), where c=randn(n, 1), r=randn(n, 1), and $c(n) = r(1)$.
13	jordbloc	Jordan block matrix (sparse).
14	compan	Companion matrix (sparse), A = compan(randn(n+1,1)).
15	pei	Pei matrix, a symmetric matrix. gallery("pei", n) or gallery("pei", n, randn).
16	randcolu	Random matrix with normalized cols and specified singular values. gallery("randcolu", n).
17	sprandn	Sparse normally distributed random matrix, A = sprandn(n, n,0.02).
18	riemann	Matrix associated with the Riemann hypothesis. gallery("riemann", n).
19	compar	Comparison matrix, gallery("compar", randn(n), unidrnd(2)$-$1).
20	tridiag	Tridiagonal matrix (sparse).
21	chebspec	Chebyshev spectral differentiation matrix, gallery("chebspec", n, 1).
22	lehmer	Lehmer matrix, a symmetric positive definite matrix such that $A(i,j) = i/j$ for $j \geq i$. Its inverse is tridiagonal. gallery("lehmer", n).
23	toeppd	Symmetric positive semidefinite Toeplitz matrix. gallery("toeppd", n).
24	minij	Symmetric positive definite matrix with $A(i,j)$ = min(i, j). gallery("minij", n).
25	randsvd	Random matrix with preassigned singular values and specified bandwidth. gallery("randsvd", n).
26	forsythe	Forsythe matrix, a perturbed Jordan block matrix (sparse).
27	fiedler	Fiedler matrix, gallery("fiedler", n), or gallery("fiedler", randn(n, 1)).
28	dorr	Dorr matrix, a diagonally dominant, ill-conditioned, tridiagonal matrix (sparse).
29	demmel	$A = D*$(eye(n) + $10^{-7}*$rand(n)), where D = diag($10^{14*(0:n-1)/n}$) [7].
30	chebvand	Chebyshev Vandermonde matrix based on n equally spaced points on the interval $[0, 1]$. gallery("chebvand", n).
31	invhess	A=gallery("invhess", n, rand($n-1$, 1)). Its inverse is an upper Hessenberg matrix.
32	prolate	Prolate matrix, a spd ill-conditioned Toeplitz matrix. gallery("prolate", n).
33	frank	Frank matrix, an upper Hessenberg matrix with ill-conditioned eigenvalues.
34	cauchy	Cauchy matrix, gallery("cauchy", randn(n, 1), randn(n, 1)).
35	hilb	Hilbert matrix with elements $1/(i+j-1)$. A =hilb(n).
36	lotkin	Lotkin matrix, the Hilbert matrix with its first row altered to all ones. gallery("lotkin", n).
37	kahan	Kahan matrix, an upper trapezoidal matrix.

Acknowledgment. The authors thank the anonymous reviewers for their useful comments that helped us improve this paper.

REFERENCES

[1] 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*, SIAM, Philadelphia, 1999.

[2] 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.

[3] D. W. Barron and H. P. F. Swinnerton-Dyer, *Solution of simultaneous linear equations using a magnetic-tape store*, Comput. J., 3 (1960), pp. 28–33.

[4] A. Buttari, J. Langou, J. Kurzak, and J. Dongarra, *A class of parallel tiled linear algebra algorithms for multicore architectures*, Parallel Comput., 35 (2009), pp. 38–53.

[5] J. Choi, J. Dongarra, L. S. Ostrouchov, A. P. Petitet, D. W. Walker, and R. C. Whaley, *The design and implementation of the ScaLAPACK LU, QR and Cholesky factorization routines*, Scientific Programming, 5 (1996), pp. 173–184.

[6] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-Optimal Parallel and Sequential QR and LU Factorizations*, Technical report UCB/EECS-2008-89, UC Berkeley, Berkeley, CA, 2008.

[7] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[8] S. Donfack, L. Grigori, and A. Kumar Gupta, *Adapting communication-avoiding LU and QR factorizations to multicore architectures*, Proceedings of IPDPS, 2010.

[9] J. Dongarra, P. Luszczek, and A. Petitet, *The LINPACK benchmark: Past, present and future*, Concurrency: Practice and Experience, 15 (2003), pp. 803–820.

[10] T. Endo and K. Taura, *Highly Latency Tolerant Gaussian Elimination*, in Proceedings of 6th IEEE/ACM International Workshop on Grid Computing, 2005, pp. 91–98.

[11] L. Grigori, J. W. Demmel, and H. Xiang, *Communication avoiding Gaussian elimination*, Proceedings of the ACM/IEEE SC08 Conference, 2008.

[12] L. Grigori, J. W. Demmel, and H. Xiang, *CALU: A Communication Optimal LU Factorization Algorithm*, Technical report UCB/EECS-2010-29, EECS Department, University of California, Berkeley, Berkeley, CA, 2010.

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

[14] N. J. Higham, *The Matrix Function Toolbox*; available online at http://www.ma.man.ac.uk/~higham/mftoolbox.

[15] N. J. Higham and D. J. Higham, *Large growth factors in Gaussian elimination with pivoting*, SIAM J. Matrix Anal. Appl., 10 (1989), pp. 155–164.

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

[17] J.-W. Hong and H. T. Kung, *I/O complexity: The red-blue pebble game*, in STOC '81: Proceedings of the Thirteenth Annual ACM Symposium on Theory of Computing, New York, 1981, ACM, pp. 326–333.

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

[19] T. Joffrain, E. S. Quintana-Ortí, and R. A. van de Geijn, *Rapid development of high-performance out-of-core solvers*, in Proceedings of PARA 2004, Lecture Notes in Comput. Sci. 3732, Springer-Verlag, Berlin, Heidelberg, 2005, pp. 413–422.

[20] E. Quintana-Ortí and R. A. van de Geijn, *Updating an LU factorization with pivoting*, ACM Trans. Math. Software, 35 (2008), pp. 11:1–11:16.

[21] G. Quintana-Orti, E. S. Quintana-Orti, E. Chan, F. G. Van Zee, and R. van de Geijn, *Programming Algorithms-By-Blocks for Matrix Computations on Multithreaded Architectures*, Technical report TR-08-04, University of Texas at Austin, Austin, TX, 2008.

[22] R. D. Skeel, *Iterative refinement implies numerical stability for Gaussian elimination*, Math. Comp., 35 (1980), pp. 817–832.

[23] D. C. Sorensen, *Analysis of pairwise pivoting in Gaussian elimination*, IEEE Trans. Comput., 34 (1985), pp. 274–278.

[24] S. Toledo, *Locality of reference in LU Decomposition with partial pivoting*, SIAM J. Matrix Anal. Appl., 18 (1997), pp. 1065–1081.

[25] S. Tomov, J. Dongarra, and M. Baboulin, *Towards dense linear algebra for hybrid gpu accelerated manycore systems*, Parallel Computing, 36 (2010), pp. 232–240.
[26] L. N. Trefethen and R. S. Schreiber, *Average-case stability of Gaussian elimination*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 335–360.
[27] V. Volkov, *Private communication*.
[28] E. L. Yip, *Subroutines for Out of Core Solutions of Large Complex Linear Systems*, Technical report NASA-CR-159142, NASA, Washington, DC, 1979.