CALU: A Communication Optimal LU Factorization Algorithm

James Demmel
Laura Grigori
Hua Xiang

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-29
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-29.html

March 15, 2010

Acknowledgement

This work has been supported in part by the French National Research Agency (ANR) through the COSINUS program (project PETAL no. ANR-08-COSI-009), by Microsoft (Award #024263), by Intel (Award #024894) and U.C. Discovery (Award #DIG07-10227).

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 ca-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. Hence we expect that CALU will behave as GEPP and it will be also 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 of GEPP. However, we present examples showing that neither GEPP nor CALU is uniformly more stable than the other.

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

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

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 [5]. These bounds were obtained by showing through reduction that lower bounds on dense matrix multiplication [15, 16] represent lower bounds for dense LU factorization as well. 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 $\Omega(n^3/W^{1/2})$ number of words and $\Omega(n^3/W^{3/2})$ number of messages, where W denotes the fast memory size and we assume a message consists of at most W words in consecutive memory locations. On a parallel machine with P processors, if we consider that the local memory size used on each processor is on the order of n^2/P, so 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

*INRIA Saclay - Ile de France, Laboratoire de Recherche en Informatique Universite Paris-Sud 11, France (laura.grigori@inria.fr). This work has been supported in part by French National Research Agency (ANR) through COSINUS program (project PETAL no ANR-08-COSI-009).

†Computer Science Division and Mathematics Department, UC Berkeley, CA 94720-1776, USA (demmel@cs.berkeley.edu). Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227).

‡School of Mathematics and Statistics, Wuhan University, Wuhan 430072, P. R. China (hxiang@whu.edu.cn).

and parallel machines. In the sequential case, the DGETRF routine in LAPACK 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 [20, 11]. We will see later in the paper that DGETRF minimizes neither the bandwidth nor the latency in some cases. Recursive LU attains the bandwidth lower bound but not the latency lower bound in general. In the parallel case, the PDGETRF routine in ScaLAPACK [3] 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 latency lower bound 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 ca-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. 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 with GEPP being the operator used to select pivot rows at each node of the reduction tree. The reduction tree is selected depending on the underlying architecture. In this paper we study in particular binary tree based and flat tree based CALU. It has been shown in [10], where the algorithm has been presented for the first time, that binary tree based CALU leads to important speedups in practice over ScaLAPACK on distributed memory computers. In [7] 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. Hence we expect that CALU will behave as GEPP and it will be also 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} denotes the absolute value of the element of A at row i and column j, and k denotes the matrix obtained at the k-th 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 show in this paper that there are very sparse matrices, formed by Kronecker products involving the Wilkinson matrix, that nearly attain the bound. Moreover, there are Wilkinson-like matrices for which GEPP is stable and CALU has 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 binary tree based and flat tree based CALU. We discuss both the stability of the LU factorization and of the linear solver, in terms of pivot growth and backward errors. The results show that in practice CALU is stable. Later in the paper Figure 3.3 displays the ratio of the relative error $\|PA - LU\|/\|A\|$, the normwise backward error, and the componentwise backward error of CALU versus GEPP for all the matrices in our test set. It shows that CALU leads to backward errors within a factor of 10 of the GEPP backward errors (except for one matrix for which the ratio of the normwise backward error of CALU to GEPP is within a factor of 26).

We also discuss the stability of block versions of pairwise pivoting [19] and parallel pivoting [21], 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 used in the context of multi-core architectures [2]. It is simple to see that block parallel pivoting is unstable. With an increasing number of blocks per panel (determined by the number of processors), the growth factor is getting larger. In the extreme case when the block size is equal to 1, the growth factor is exponential 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.

The paper is organized as follows. Section 2 presents the algebra of CALU and the new ca-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, that we refer to as ca-pivoting strategy. We use several notations. We refer to the submatrix of A formed by elements of row indices from i to j and column indices from d to 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 $(m-b) \times (n-b)$. As in other classic right looking algorithms, CALU first computes the LU factorization of the first block-column (panel), then determines the block U_{12}, and updates the trailing matrix A_{22}. The algorithm continues on the block A_{22}.

The main difference with respect to other existing algorithms lies in the panel factorization. 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, and the LU factorization with no pivoting of the panel is performed.

We illustrate ca-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 $P = 4$. For the sake of simplicity, 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. In the following we use a binary reduction tree. 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. Note that this step aims at identifying in each block-row a set of b linearly independent rows, which correspond to the first b rows of $\bar{\Pi}_{i0}^T A_i$, with $i = 0 \ldots 3$.

From the P sets of local pivot rows, we perform a binary tree (of depth $\log_2 P = 2$ in our example) of GEPP factorizations of matrices of size $2b \times b$ to identify b global pivot rows. The 2 GEPP factorizations at the first level of our depth-2 binary tree are shown here, combined in one matrix. This decomposition leads to a $Pb \times Pb$ permutation matrix $\bar{\Pi}_1$, a $Pb \times 2b$ factor $\bar{L}_1$ and a $2b \times b$ factor $\bar{U}_1$.

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

The global pivot rows are obtained after applying one more GEPP factorization (at the root of our depth-2 binary tree) on the pivot rows identified previously. The permutation matrices $\bar{\Pi}_0, \bar{\Pi}_1$ do not have the same dimensions. By abuse of notation, we consider that $\bar{\Pi}_1$ is extended by the appropriate identity matrices to the dimension of $\bar{\Pi}_0$.

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

The permutations identified in the preprocessing step are applied to the original matrix A. Then the LU factorization with no pivoting of the first block-column is performed, the block-row of U is computed and the trailing matrix is updated. Note that $U_{11} = \bar{U}_2$. The factorization continues on the trailing matrix $\bar{A}$. We consider that $\bar{\Pi}_2$ is extended by the appropriate identity matrices to the dimension of $\bar{\Pi}_0$.

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

The ca-pivoting strategy has several important characteristics. First, when $b = 1$ or $P = 1$, ca-pivoting is equivalent to partial pivoting. Second, the elimination of each column of A leads to a rank-1 update of the trailing matrix. The rank-1 update property is shown experimentally to be very important for the stability of LU factorization [21]. A large rank update might lead to an unstable LU factorization, as for example in another strategy suitable for parallel computing called parallel pivoting [21].

Different reduction trees can be used during the preprocessing step of TSLU. We illustrate them using an arrow notation having 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:

$$\begin{tikzcd}
A_{00} \arrow[r] & f(A_{00}) \arrow[dr] & & \\
 & & f(A_{01}) \arrow[dr] & \\
A_{10} \arrow[r] & f(A_{10}) \arrow[ur] & & f(A_{02}) \\
A_{20} \arrow[r] & f(A_{20}) \arrow[dr] & & \\
 & & f(A_{11}) \arrow[uur] & \\
A_{30} \arrow[r] & f(A_{30}) \arrow[ur] & &
\end{tikzcd}$$

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

$$\begin{tikzcd}
A_{00} \arrow[r] & f(A_{00}) \arrow[dr] & \\
A_{10} \arrow[r] & f(A_{10}) \arrow[r] & f(A_{01}) \\
A_{20} \arrow[r] & f(A_{20}) \arrow[ur] & \\
A_{30} \arrow[r] & f(A_{30}) \arrow[uur] &
\end{tikzcd}$$

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

$$\begin{tikzcd}
A_{00} \arrow[r] & f(A_{00}) \arrow[r] & f(A_{01}) \arrow[r] & f(A_{02}) \arrow[r] & f(A_{03}) \\
A_{10} \arrow[urr] & & & & \\
A_{20} \arrow[uurrr] & & & & \\
A_{30} \arrow[uuurrrr] & & & &
\end{tikzcd}$$

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 pivot growth 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 pivot growth on a small set of input matrices (see, for example, [13]) 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 equation (3.1)), and vice-versa ($W_{EG-GEPP}$ in equation (3.1)). These two examples (presented here slightly more generally) are from V. Volkov [22]. This shows that GEPP is not uniformly more stable than CALU.

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

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

where W is a Wilkinson matrix of order $b \times b$ with $W(i,j) = -1$ for $i > j$, $W(i,i) = 1$, and $W(:,b) = 1$, I_b is the identity matrix of dimension $b \times b$, 0 is a matrix of size $b \times b$ of zeros, and e is a vector of dimension $b \times 1$ with all elements of value 1. We consider that CALU divides the input matrix into two blocks, each of dimension $3b \times 2b$. For the $W_{EG-CALU}$ matrix, 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 matrix, while GEPP does not. For the $W_{EG-GEPP}$ matrix, GEPP uses pivots from W matrix 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 quantify the stability of CALU using several metrics that include pivot growth and attained 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 similar results 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 U factor obtained from the CALU factorization of a nonsingular matrix A is nonsingular.*

Proof. Consider that CALU uses a binary tree to factor each panel. The panel is of full rank. 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, and hence there is no loss of information. Therefore the final block of pivot rows is of full rank. This reasoning applies to every panel factorization. ☐

Before proving a general result that applies to CALU using any reduction tree, we discuss first 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 consider that TSLU applied on the first block column $[A_{11}; A_{21}; A_{31}]$ performs first GEPP of $[A_{21}; A_{31}]$. Without loss of generality we consider 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):

$$\begin{tikzcd}
A_{11} \arrow[rr] & & \bar{A}_{11} \\
A_{21} \arrow[dr] & & \\
 & A_{21} \arrow[uur] & \\
A_{31} \arrow[ur] & &
\end{tikzcd}$$

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

$$\begin{pmatrix} \Pi_{11} & \Pi_{12} & \\ \Pi_{21} & \Pi_{22} & \\ & & I_{m-2b} \end{pmatrix} \cdot \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \\ A_{31} & A_{32} \end{pmatrix} = \begin{pmatrix} \bar{A}_{11} & \bar{A}_{12} \\ \bar{A}_{21} & \bar{A}_{22} \\ A_{31} & A_{32} \end{pmatrix} =$$

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

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 on G pivot on the diagonal. This is because equation (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 it can be seen from equation (3.3).

The following equalities prove the lemma:

$$\begin{aligned}L_{31}L_{21}^{-1}A_{21}\bar{U}_{11}^{-1}\bar{U}_{12} + A_{32}^s = L_{31}U_{21}\bar{U}_{11}^{-1}\bar{U}_{12} + A_{32}^s = A_{31}\bar{U}_{11}^{-1}\bar{U}_{12} + A_{32}^s = \\ = \bar{L}_{31}\bar{U}_{12} + A_{32}^s = A_{32}\end{aligned}$$

□

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, 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 $A^s(j, 1:b)$ 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 (sometimes slightly perturbed), and hence can be bounded. Before we prove the theorem, we introduce some notations and also perturb slightly the blocks of A that are used in the reduction tree and that are singular. We perturb them to make them nonsingular, which simplifies the analysis, but the algorithm does not depend on their nonsingularity.

DEFINITION 3.3. *Consider a reduction tree T of height H and a given node s_k situated at level k, where level H corresponds to the root of T. Let $s_{k+1}, \ldots, s_H$ be its ancestor nodes , where s_H is the root of the tree. For each node s_h situated at level h, the GEPP factorization $\Pi_{s_h,h}A_{s_h,h} = L_{s_h,h}U_{s_h,h}$ is performed. Thus $A_{s_h,h}$ is the $cb \times b$ submatrix gotten from stacking the b rows selected by each of s_h's c children atop one another.*

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

$$\bar{A}_h = (\Pi_{s_h,h}A_{s_h,h})(1:b, 1:b) + diag(\sum_{i=1}^{b} \epsilon_i e_i),$$

where ϵ_i are randomly generated small numbers, $e_i = (\ldots 1 \ldots)^T$ is a vector of dimension $b \times 1$ with the only nonzero element 1 in position $(i,1)$ if $U_{s_h,h}(i,i) = 0$, the vector of 0s otherwise. The matrix $\bar{A}_h$ consists of the b rows selected by node s_h to pass to its parent, modified by arbitrarily tiny perturbations on its diagonal if they are needed to keep the b rows linearly independent. We then have with high probability a nonsingular matrix $\bar{A}_h$ and 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, 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

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

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

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

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

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

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

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

where

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

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

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

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

The following corollary shows similarities between CALU and GEPP of a larger matrix. Since GEPP is stable in practice, we expect CALU to be also stable 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 G whose entries are the same as the entries of A, sometimes slightly perturbed, or zeros.*

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 factor L is bounded in absolute value by 2^{bH}.*

Proof. Consider the first block column factorization, 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:

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

By using the relations $|A(j,1:b) \cdot \bar{U}_0^{-1}| \leq 1$ and $|\bar{A}_{i-1} \cdot \bar{U}_i^{-1}| \leq 1$ for $i = 1 \dots H$, we have the following:

$$
\begin{aligned}
|L(j,1:b)| &= |A(j,1:b)\cdot \bar{U}_H^{-1}| = \\
&= |A(j,1:b)\cdot \bar{A}_0^{-1}\cdot \bar{A}_0\cdot \bar{A}_1^{-1}\cdot \bar{A}_1\dots \bar{A}_{H-1}^{-1}\cdot \bar{A}_{H-1}\cdot \bar{U}_H^{-1}| = \\
&= |A(j,1:b)\cdot \bar{U}_0^{-1}\cdot \bar{L}_0^{-1}\cdot \bar{A}_0\cdot \bar{U}_1^{-1}\cdot \bar{L}_1^{-1}\cdot \bar{A}_1\dots \bar{U}_{H-1}^{-1}\cdot \bar{L}_{H-1}^{-1}\cdot \bar{A}_{H-1}\cdot \bar{U}_H^{-1}| \leq \\
&\leq |A(j,1:b)\cdot \bar{U}_0^{-1}|\cdot |\bar{L}_0^{-1}|\cdot |\bar{A}_0\cdot \bar{U}_1^{-1}|\cdot |\bar{L}_1^{-1}|\dots |\bar{L}_{H-1}^{-1}|\cdot |\bar{A}_{H-1}\cdot \bar{U}_H^{-1}| \leq \\
&\leq 2^{bH}
\end{aligned}
$$

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

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

We give an example of a matrix formed by Wilkinson-type sub-matrices 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. With the same notation as in Theorems 3.4 and 3.6, the submatrices $\bar{A}_i$ are formed as following. 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 $|L|$. 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 $\||L||U|\|_\infty \leq (1 + 2(n^2-n)g_W)\|A\|_\infty$ (Lemma 9.6 in section 9.3 of [12]). The growth factor g_W is defined in (3.7), where $a_{ij}^{(k)}$ denotes the entry in position (i,j) obtained after k steps of elimination.

$$
g_W = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\max_{ij} |a_{ij}|} \tag{3.7}
$$

The growth factor of CALU is equal to the growth factor of matrix G in equation (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 [13] 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 equation (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 equation (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 $m \times (b+1)$ and 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

Table 3.1
Bounds for $|L|$ and for the growth factor g 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.

	matrix of size $m \times (b+1)$		
	CALU(b,H)		GEPP
	upper bound	attained	upper bound
$\lvert L\rvert$	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
$\lvert L\rvert$	2^{bH}	$2^{(b-2)H-(b-1)}$	1
g_W	$2^{n(H+1)-1}$	2^{n-1}	2^{n-1}

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,

$$W_s = \begin{pmatrix} I_H \otimes W + S \otimes N & e_1^T \\ e_{n-1} & \end{pmatrix}, \tag{3.8}$$

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 order $1 \times n-1$, and e_{n-1} is the vector $(0, \ldots, 0, 1)$ of dimension $1 \times n-1$. For example, when $H = 3$ this matrix becomes

$$\begin{pmatrix} W & & & e_1 \\ N & W & & \\ & N & W & \\ & & e_b^T & \end{pmatrix}. \tag{3.9}$$

The matrix W_s gets pivot growth 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 pivot growth of about $.1 \cdot 2^{n-1}$, which is within a constant factor of the maximum pivot growth 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 $n = 2^k$, and the sample size is $\max\{10 * 2^{10-k}, 3\}$. We discuss several metrics, that concern the LU decomposition and the linear solver using it, such as the growth factor, normwise and componentwise backward errors. Additional results, that consider as well several special matrices [14] including sparse matrices are described in Appendix B.

In this section we present results for the growth factor g_T defined in (3.10), which was introduced by Trefethen and Schreiber in [21]. The authors have introduced a 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 B we also present results for g_W , defined in (3.7), as well as the growth factor g_D defined in (3.11), which was introduced in [4]. As for g_W, $a_{ij}^{(k)}$ denotes the entry in position (i, j) obtained after k steps of elimination.

$$g_T = \frac{\max_{i,j,k} |a_{ij}^{(k)}|}{\sigma_A} \tag{3.10}$$

$$g_D = \max_j \left\{ \frac{\max_i |u_{ij}|}{\max_i |a_{ij}|} \right\} \tag{3.11}$$

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 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 [21].

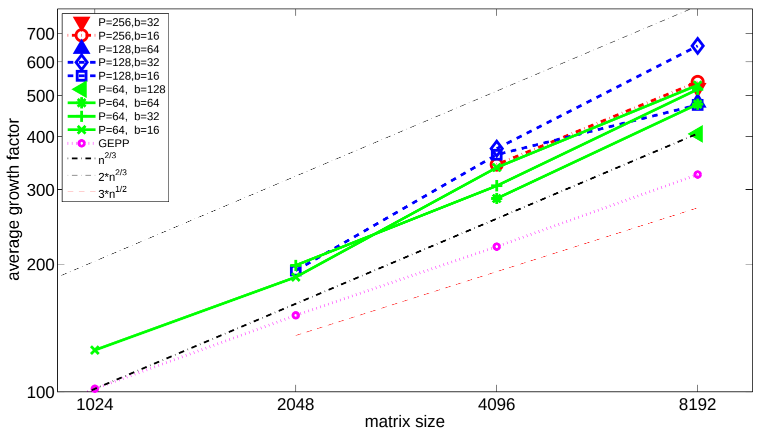

FIG. 3.1. *Growth factor g_T of binary tree based CALU for random matrices.*

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 factor of both binary tree based and flat tree based CALU have similar behavior 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 [8], and denoted as HPL1, HPL2 and HPL3

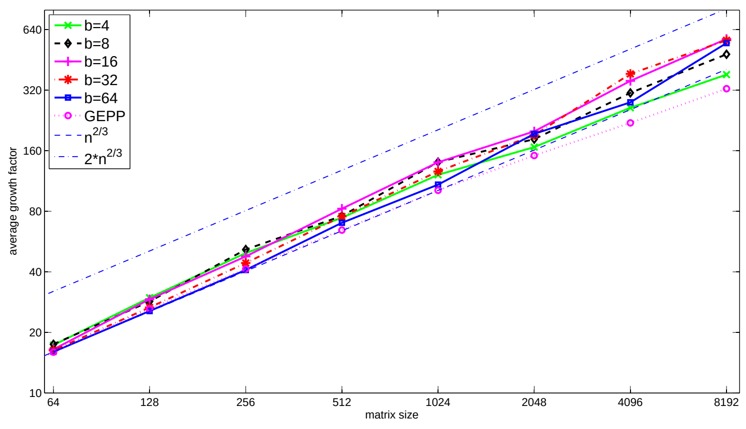

FIG. 3.2. *Growth factor g_T of flat tree based CALU for random matrices.*

(equations (3.12) to (3.14)).

$$\mathrm{HPL1} = ||Ax-b||_\infty/(\epsilon||A||_1 * N), \tag{3.12}$$
$$\mathrm{HPL2} = ||Ax-b||_\infty/(\epsilon||A||_1||x||_1), \tag{3.13}$$
$$\mathrm{HPL3} = ||Ax-b||_\infty/(\epsilon||A||_\infty||x||_\infty * N). \tag{3.14}$$

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,

$$\eta := \frac{||r||}{||A||\ ||x|| + ||b||}. \tag{3.15}$$

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

$$w := \max_i \frac{|r_i|}{(|A|\ |x| + |b|)_i}, \tag{3.16}$$

is used, where the computed residual is $r = b - Ax$. The residual is computed in working precision [18] 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 B we present more detailed results on random matrices. We also consider different special matrices, including sparse matrices, described in Table 7.1. 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 7.4, 7.5 and 7.6.

The new ca-pivoting strategy 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 ca-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: 20 random matrices from Table 3.2 and 37 special matrices from Table 7.1.

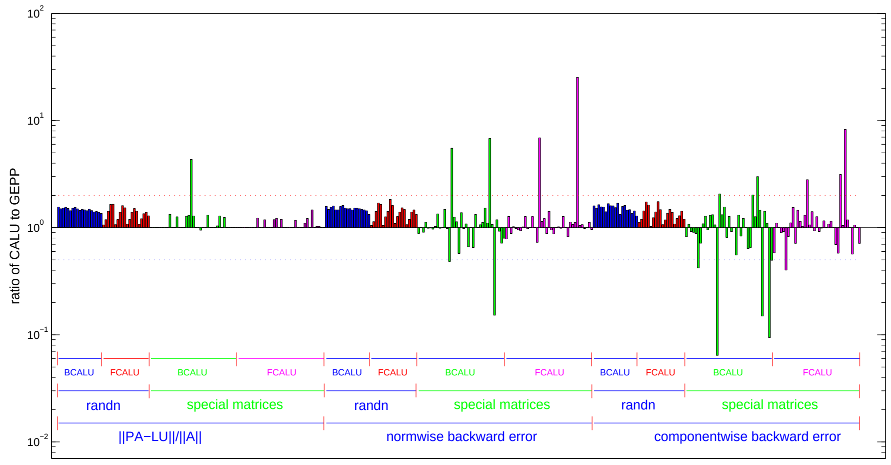

Fig. 3.3. *A summary of all our experimental data, showing the ratio of CALU's backward error to GEPP's backward error for all test matrices. Each vertical bar represents such a ratio for one test matrix, so bars above* $10^0 = 1$ *mean CALU's backward error is larger, and bars below* 1 *mean GEPP's 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 equation* (3.15) *is the metric. For the last third of the bars, labeled "componentwise backward error",* w *in equation* (3.16) *is the metric. The test matrices are further labeled either as "randn", which are randomly generated, or "special", listed in Table 7.1. 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 7.2 -7.6 contain all the raw data.*

The results presented in this section and in Appendix B 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 ca-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.

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.

$$\begin{tikzcd}
A_0 \arrow[r] & U_{00} \arrow[dr] & & \\
& & U_{01} \arrow[ddr] & \\
A_1 \arrow[r] & U_{10} \arrow[ur] & & \\
& & & U_{02} \\
A_2 \arrow[r] & U_{20} \arrow[dr] & & \\
& & U_{11} \arrow[uur] & \\
A_3 \arrow[r] & U_{30} \arrow[ur] & &
\end{tikzcd}$$

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

$$\begin{tikzcd}
A_0 \arrow[r] & U_{00} \arrow[r] & U_{01} \arrow[r] & U_{02} \arrow[r] & U_{03} \\
A_1 \arrow[urr] & & & & \\
A_2 \arrow[uurrr] & & & & \\
A_3 \arrow[uuurrrr] & & & &
\end{tikzcd}$$

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 [21]). 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 multicore architectures [2, 17], and are referred to as tiled algorithms in [2].

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 [21]. It

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								
8096	-	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

is thought [21] 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 [21]. Note that it involves a rank update equal to the number of rows at each step of elimination. The experiments performed in [21] 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 factorizations that involve U factors previously computed. This can propagate ill-conditioning through the factorization.

We discuss here the stability in terms of pivot growth for block parallel pivoting and pairwise pivoting. We perform our tests in Matlab, using matrices from a normal distribution. The pivot growth 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 large number of processors. We conclude that block parallel pivoting is unstable.

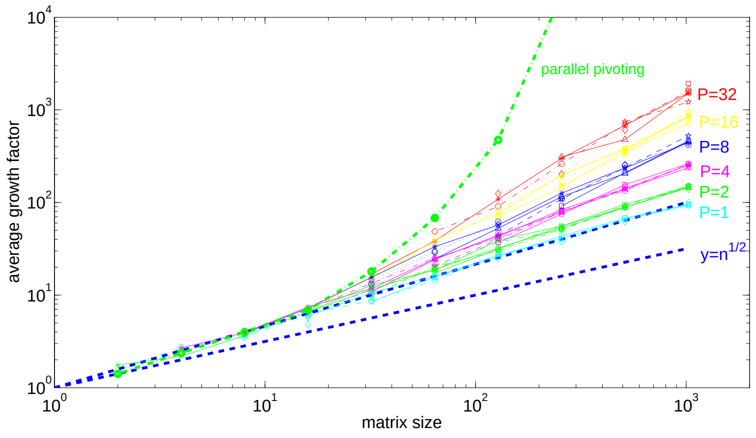

FIG. 4.1. *Growth factor of block parallel pivoting for varying block size b and number of processors P.*

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 ca-pivoting bears some similarities to the batched pivoting strategy [9]. To factor a block column partitioned as $[A_0; A_1; \ldots; A_{P-1}]$, batched pivoting uses also 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,

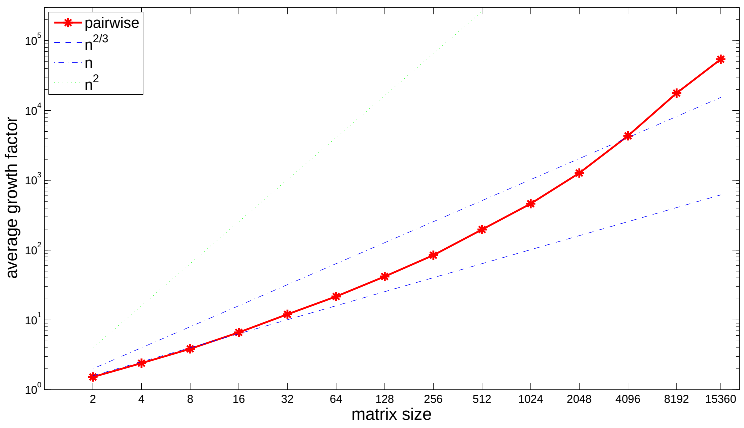

FIG. 4.2. *Growth factor of pairwise pivoting for varying matrix size.*

which corresponds to the first computation in our preprocessing step. From the P sets of b rows, the set considered by some criterion as the best will be used to factor the entire block column. Hence, the different P sets are not combined as in CALU. Also batched pivoting fails when each block row A_i is singular, while the block-column is nonsingular. This can happen for example in the case of 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.

CALU factors the input matrix by iterating over panels. At each iteration it factors the current panel and then it updates the trailing matrix. The only difference with GEPP lies in the panel factorization. The trailing matrix update can be performed by any existing algorithm, depending on the underlying architecture. Hence we will not detail this step in this section. We focus on the description of the TSLU algorithm used for panel factorization.

As described in section 2, 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 is performed as a reduction operation, with GEPP performed at each node of the reduction tree. The execution of TSLU is driven by the reduction tree.

We will see later that for one level of parallelism, the binary tree leads to an optimal algorithm. However today's parallel machines have a more complex architecture, with multiple levels of parallelism and memory hierarchies. The choice of the reduction tree will depend on the machine architecture. In the following we describe first parallel TSLU and CALU, and then sequential TSLU and CALU.

We present in Algorithm 1 a parallel implementation of TSLU that uses as input an arbitrary all-reduction tree (that is the result is available on all processors). In the preprocessing step, the algorithm traverses the reduction tree bottom-up. The input matrix is distributed over P processors using a 1-D block row layout. 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 [5].

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 $L$.
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 $L$ 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 $\bar{\Pi} = \bar{\Pi}_L \dots \bar{\Pi}_1 \bar{\Pi}_0$, where $\bar{\Pi}_i$ represents the per-
    mutation 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: $U_{i,L}$ is the $U$ factor obtained at step (15), for all processors $i \in S$.
```

For binary tree based parallel TSLU, the computation on the critical path is composed of the GEPP factorization of a block of size $mb/P \times b$ and $\log_2 P$ GEPP factorizations of blocks of size $2b \times b$ each. Its runtime estimation is displayed in Table 5.1.

Table 5.2 displays the runtime estimation of parallel CALU for an $m \times n$ matrix distributed using a two-dimensional block cyclic layout. The algorithm is described in detail in [10]. The panel factorization is performed using binary tree based TSLU. The other steps of the algorithm are similar to the PDGETRF routine in ScaLAPACK that implements Gaussian elimination with partial pivoting.

Sequential TSLU based on a flat tree consists of reading in memory blocks that fit in the memory of size W and are as large as possible. 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 $b^2 + b_1 b \leq W$. Hence $b_1 \approx (W - b^2)/b = W_1/b$,

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	$2mb^2 - \frac{b^3}{3}$	$\frac{3mb}{W-b^2} + b$	$3mb + 2b^2$
Seq. TSLU var. 2	$2mb^2 - \frac{b^3}{3}$	$\frac{5mb}{W-b^2}$	$5mb$

with $W_1 = W - b^2$, and $W_1 \geq W/2$. In the preprocessing step of TSLU, GEPP of the first block is computed. The b rows are kept in memory, and then the second block is read. In the preprocessing step the matrix is read once. Assuming that the blocks are stored in contiguous memory, $mb/b_1 b = mb/W_1$ messages are sent. At the end of the preprocessing step, the b pivot rows are in fast memory. 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. The rows can be read in fast memory using one message, since $W > b^2$. 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*. This leads to a total of $b+1$ messages at most ($2b^2$ words) to permute the rows that are in the first b positions of A. 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 $mb/(W - b^2)$. 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 $b > mb/(W - b^2)$. To reduce the number of messages, an implementation of the algorithm should choose the first strategy if $b = \min(b, mb/(W - b^2))$, the second otherwise. During the LU factorization with no pivoting, the matrix is read and written once. This leads to $2mb/(W - b^2)$ 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). The size of each block $m/P_r \times n/P_c$ is chosen such that 3 blocks fit into memory, that is $3mn/P \leq W$. This is necessary for performing the updates on the trailing matrix, when 3 blocks are necessary. This approach is similar to sequential CAQR discussed in [5].

We analyze a sequential CALU based on a right-looking algorithm. At step k of the algorithm, the LU factorization of panel k is computed using flat tree based TSLU. Then the trailing matrix is permuted and updated. We choose to analyze a right-looking variant for the ease of understanding. We note that the memory accesses are the same for right-looking and left-looking for the panel factorization and the updates. However the permutations are simpler to explain for the right-looking variant. The detailed counts are presented in Appendix A.

The number of messages and the number of words of CALU increase with increasing P and P_c. P is given by the size of the memory W and so fixed. Since we impose that 3 blocks fit in fast memory, $P = \frac{3mn}{W}$. Also we impose that the number of rows of a block is larger or equal to the number of columns. Hence P_c is minimized when it is equal to $\sqrt{nP/m}$, that is the blocks are square. This is similar to 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 $P = \frac{3mn}{W}$ 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	$\frac{15\sqrt{3}mn^2}{2W^{3/2}} + \frac{15mn}{2W}$
# words	$\frac{5\sqrt{3}mn^2}{2\sqrt{W}} - \frac{5\sqrt{3}n^3}{6\sqrt{W}} + 5\left(mn - \frac{n^2}{2}\right)$
# flops	$mn^2 - \frac{n^3}{3} + \frac{2}{\sqrt{3}}mn\sqrt{W} - \frac{1}{\sqrt{3}}n^2\sqrt{W}$

analysis performed for sequential CAQR [5]. We have then $P_c = \frac{\sqrt{3}n}{\sqrt{W}}$, $P_r = \frac{\sqrt{3}m}{\sqrt{W}}$, and $W_1 = \frac{2W}{3}$. With this choice, the runtime of sequential CALU is presented in Table 5.2.

6. Lower bounds on communication.

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 [15] in the sequential case. A simpler proof and its extension to the parallel case is presented by Irony et al. in [16]. 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 [5]. 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 [15, 16] represent lower bounds for LU decomposition, using the following reduction of matrix multiplication to LU:

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

For sequential LU decomposition of a matrix of size $m \times n$, a lower bound on the number of words and number of messages communicated between fast and slow memory is

$$\# \text{ words} \geq \Omega\left(\frac{mn^2}{\sqrt{W}}\right), \tag{6.1}$$

$$\# \text{ messages} \geq \Omega\left(\frac{mn^2}{W^{3/2}}\right), \tag{6.2}$$

where W is the size of fast memory.

For parallel LU decomposition of a matrix of size $m \times n$, 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 $(mn^2 - n^3)/P$. A lower bound on the number of words and number of messages at least one of the processors must communicate is:

$$\# \text{ words} \geq \Omega\left(\sqrt{\frac{mn^3}{P}}\right). \tag{6.3}$$

$$\# \text{ messages} \geq \Omega\left(\sqrt{\frac{nP}{m}}\right). \tag{6.4}$$

In the following we show that CALU attains the lower bounds on communication. We discuss first 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 $O(mb/W)$, 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 [20, 11] and LAPACK's DGETRF [1].

Recursive LU partitions the matrix along the columns into two submatrices. The LU factorization is obtained by factoring the left half of the matrix, updating the right half of the matrix, and then factoring the right half of the matrix. The factorizations are performed recursively. The recursion is stopped when a column is reached. The number of words transferred by this algorithm is $O(mn^2/\sqrt{W}) + mn$. This can be obtained by an analysis similar to the one performed for recursive QR [5]. Hence recursive LU minimizes the number of words communicated. But it does not always attain the lower bound for the number of messages. The base case for the recursion is one column factorization. When the matrix is stored using square blocks of size $\sqrt{W} \times \sqrt{W}$, the number of messages is at least $O(mn/\sqrt{W})$. When $n < W$, this is larger than the lower bound on number of messages.

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 3 blocks fit in memory, that is $3mn/P \leq W$. The communication cost of the trailing matrix update is the same as for sequential CALU, described in Appendix A. The overall communication cost due to the panel factorization is:

$$\#words = \begin{cases} O\left(mn\right) & \text{if } \frac{mn}{P_c} \leq W \\ O\left(\frac{mn^2}{P_c}\right) & \text{if } \frac{mn}{P_c} > W \end{cases}$$
$$\#messages = O(nP_r)$$

The total number of words communicated in DGETRF is:

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

In the first case, $m > W$, one column does not fit in memory. We choose $P_c = \sqrt{3}n/\sqrt{W}$, and $P_r = \sqrt{3}m/\sqrt{W}$. The number of words communicated is $O(mn^2/\sqrt{W})$. In this case DGETRF attains the bandwidth lower bound. 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 $\sqrt{W}/\sqrt{n}$, when $W > n$. In the third case, P_c is chosen such that the panel fits in memory, that is $P_c = mn/W$. Then the number of words communicated is $O(m^2n^2/W)$, which exceeds the lower bound by a factor of $m/\sqrt{W}$. 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 $m > W$, 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 $O(nP_r) = O(\frac{mn}{\sqrt{W}})$ messages exchanged between slow and fast memory. If $W = O(n)$, this term attains the lower bound on number of messages. But not if $n < W$.

We discuss now parallel CALU, whose performance model is presented in table 5.2. To attain the communication bounds presented in equation (6.4), 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 [5]:

$$P_r = \sqrt{\frac{mP}{n}} \text{ , } P_c = \sqrt{\frac{nP}{m}} \text{ and } b = \log^{-2}\left(\sqrt{\frac{mP}{n}}\right) \cdot \sqrt{\frac{mn}{P}}, \tag{6.5}$$

The idea behind this layout is to choose b close to its maximum value, such that the latency lower bound 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 6.1. It attains the lower bounds on both latency and bandwidth, modulo polylog factors.

We now compare CALU to parallel GEPP as for example implemented in the routine PDGETRF of ScaLAPACK. For bandwidth, both algorithms have the same communication cost. For latency, CALU is smaller by a factor of the order of b (depending P_r and P_c). The reduction in the number of messages within processor columns comes from the reduction in the factorization of a panel. Due to partial pivoting, PDGETRF has an $O(n \log P)$ term in the number of messages. Hence it does not attain the latency lower bound.

7. Conclusions.

This paper discusses 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.

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

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 equation (6.5). *Some lower-order terms are omitted.*

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.

Runtime estimation of sequential CALU. We analyze the performance of sequential CALU. The number of memory accesses detailed for each iteration and each step of the algorithm are as follows. At iteration k, the LU factorization of the current panel is performed using TSLU. We present the total number of memory accesses using the second approach for TSLU. We note $W_1 = W - n^2/P_c^2$.

$$\#words\ var.\ 2 = \frac{5n}{P_c}\sum_{k=1}^{P_c}\left(m-(k-1)\frac{n}{P_c}\right) \approx 5\left(mn-\frac{n^2}{2}\right)$$
$$\#messages\ var.\ 2 = \frac{5n}{W_1 P_c}\sum_{k=1}^{P_c}\left(m-(k-1)\frac{n}{P_c}\right) \approx \frac{5}{W_1}\left(mn-\frac{n^2}{2}\right)$$

The permutations used for the factorization of the panel at step k are applied on the trailing matrix. We use the following strategy to perform the permutations for each panel of the trailing matrix, similar to the strategy by blocks used for sequential TSLU described in section 5. The rows that correspond to the first b positions of the current panel are first read in memory with the first block. Then for each row that needs to be pivoted, its corresponding block is read, and the rows are interchanged.

At most P_r blocks are modified, hence:

$$\#words \leq \frac{2n}{P_c}\sum_{k=1}^{P_c}(P_c-k)\left(m-(k-1)\frac{n}{P_c}\right) \approx (mn-\frac{n^2}{3})P_c$$
$$\#messages \leq \sum_{k=1}^{P_c} 2(P_c-k)P_r = P_cP$$

The trailing matrix update iterates at each step k over the panels of the trailing matrix. For each such panel, the update involves reading it in fast memory and writing it back to slow memory, and reading in fast memory the current panel. This leads to at most $3P_r$ messages exchanged.

$$\#words \leq \frac{3n}{P_c}\sum_{k=1}^{P_c}(P_c-k)\left(m-(k-1)\frac{n}{P_c}\right) \approx \frac{3mn-n^2}{2}P_c$$
$$\#messages \leq 3P_r\sum_{k=1}^{P_c}(P_c-k) \approx \frac{3PP_c}{2}$$

We note that the number of messages and the number of words of CALU increase with increasing P and P_c. P is given by the size of the memory W and so fixed. Since we impose that 3 blocks fit in fast memory, $P = \frac{3mn}{W}$. Also we impose that the number of rows of a block is larger or equal to the number of columns. Hence P_c is minimized when it is equal to $\sqrt{nP/m}$, that is the blocks are square. This is similar to the analysis performed for sequential CAQR [5]. We have then $P_c = \frac{\sqrt{3}n}{\sqrt{W}}$, $P_r = \frac{\sqrt{3}m}{\sqrt{W}}$, and $W_1 = \frac{2W}{3}$. With this choice, the runtime of sequential CALU becomes:

$$\begin{aligned}T_{Seq.CALU\ var.\ 2} \leq & \left(\frac{15\sqrt{3}mn^2}{2W^{3/2}}+\frac{15mn}{2W}\right)\alpha \\ & + \left(\frac{5\sqrt{3}mn^2}{2\sqrt{W}}-\frac{5\sqrt{3}n^3}{6\sqrt{W}}+5\left(mn-\frac{n^2}{2}\right)\right)\beta \\ & + \left(mn^2-\frac{n^3}{3}+\frac{2}{\sqrt{3}}mn\sqrt{W}-\frac{1}{\sqrt{3}}n^2\sqrt{W}\right)\gamma\end{aligned}$$

Runtime estimation of parallel CALU. In the following we estimate the performance of parallel CALU. The input matrix of size $m \times n$ is distributed over a grid of processors of size $P_r \times P_c$, $P_r \cdot P_c = P$, using a block cyclic layout. The size of the block is b. We detail the estimation at each iteration j of the algorithm. We note the size of the active matrix $m_j \times n_j$, with $m_j = m-(j-1)b$ and $n_j = n-(j-1)b$.

1. Compute TSLU factorization of current panel j.

$$\begin{aligned}&\textstyle\sum_{j=1}^{n/b}\left[\left[\frac{2m_jb^2}{P_r}+\frac{b^3}{3}(5\log_2 P_r-1)\right]\gamma+\log_2 P_r\alpha+b^2\log_2 P_r\beta\right] \\ &\approx \left[\frac{1}{P_r}\left(2mn-n^2\right)b+\frac{nb^2}{3}(5\log_2 P_r-1)\right]\gamma+\frac{n}{b}\log_2 P_r\alpha+nb\log_2 P_r\beta\end{aligned}$$

2. Broadcast the pivot information along the rows of the process grid.

$$\sum_{j=1}^{n/b} \log_2 P_c(\alpha + b\beta) = \log_2 P_c \left(\frac{n}{b}\alpha + n\beta \right)$$

3. *PDLAPIV*: Apply the pivot information to the remainder of the rows. We consider that this is performed using an all to all reduce operation.

$$\sum_{j=1}^{n/b} \log_2 P_r \left(\alpha + \frac{nb}{P_c}\beta \right) = \frac{n}{b}\log_2 P_r \alpha + \frac{n^2}{P_c}\log_2 P_r \beta$$

4. *PDTRSM*: Broadcast the $b \times b$ upper part of panel j of L along row of processes owning block row j of U. Compute block row j of U.

$$\sum_{j=1}^{n/b} \left(\log_2 P_c \alpha + \log_2 P_c \frac{b^2}{2}\beta + \frac{n_j - b}{P_c} b^2 \gamma \right) = \frac{n}{b}\log_2 P_c \alpha + \frac{nb}{2}\log_2 P_c \beta + \frac{n^2 b}{2P_c}\gamma$$

5. *PDGEMM*: Broadcast the block column j of L along rows of processors of the grid.

$$\sum_{j=1}^{n/b} \log_2 P_c \left(\alpha_r + \frac{(m_j - b)b}{P_r}\beta_r \right) = \log_2 P_c \left(\frac{n}{b}\alpha_r + \frac{1}{P_r}\left(mn - \frac{n^2}{2} \right)\beta_r \right)$$

Broadcast the block row j of U along columns of processors of the grid.

$$\frac{n}{b}\log_2 P_r \alpha_c + \frac{n^2}{2P_c}\log_2 P_r \beta_c$$

Perform a rank-b update on the trailing matrix.

$$\sum_{j=1}^{n/b} 2b \frac{m_j - b}{P_r}\frac{n_j - b}{P_c} \approx \frac{1}{P}\left(mn^2 - \frac{n^3}{3} \right)\gamma$$

Appendix B. 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 7.2 and 7.3 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 7.4, 7.5, and 7.6 display results obtained for the special matrices presented in Table 7.1. 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 7.1: 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	Counter-example 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 semi-definite 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 semi-definite 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	$A = D*(\text{eye}(n) + 10^{-7}*\text{rand}(n))$, where $D = \text{diag}(10^{14*(0:n-1)/n})$ [4].
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.

Table 7.2
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 7.3

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 7.4

Stability results for GEPP on special matrices.

	matrix	cond(A,2)	g_W	$\Vert L\Vert_1$	$\Vert L^{-1}\Vert_1$	$\max_{ij} \lvert U_{ij}\rvert$	$\min_{kk} \lvert U_{kk}\rvert$	cond(U,1)	$\frac{\Vert PA-LU\Vert_F}{\Vert A\Vert_F}$	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 7.5
Stability of CALU based on a binary tree for special matrices.

matrix	g_W	τ_{ave}	$\tau_{\min}$	$\|\|L\|\|_1$	$\|\|L^{-1}\|\|_1$	$\max\limits_{ij}\|U_{ij}\|$	$\min\limits_{kk}\|U_{kk}\|$	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 7.6
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

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, PA, USA, 1999.

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

[3] 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. ISSN 1058-9244.

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

[5] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-optimal parallel and sequential QR and LU factorizations*, Tech. Report UCB/EECS-2008-89, UC Berkeley, 2008. LAPACK Working Note 204.

[6] J. W. Demmel, L. Grigori, M. Hoemmen, and J. Langou, *Communication-avoiding parallel and sequential QR and LU factorizations: theory and practice*, Tech. Report UCB/EECS-2008-89, University of California Berkeley, EECS Department, 2008. LAWN #204.

[7] S. Donfack, L. Grigori, and A. Kumar Gupta, *Adapting communication-avoiding lu and qr factorizations to multicore architectures*, Proceedings of IPDPS, (2010).

[8] J. Dongarra, P. Luszczek, and A. Petitet, *The LINPACK Benchmark: Past, Present and Future*, Concurrency: Practice and Experience, 15 (2003), pp. 803–820.

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

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

[11] F. Gustavson, *Recursion Leads to Automatic Variable Blocking for Dense Linear-Algebra Algorithms*, IBM Journal of Research and Development, 41 (1997), pp. 737–755.

[12] N. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, second ed., 2002.

[13] N. Higham and D. J. Higham, *Large growth factors in gaussian elimination with pivoting*, SIMAX, 10 (1989), pp. 155–164.

[14] Nicholas J. Higham, *The Matrix Function Toolbox.* `http://www.ma.man.ac.uk/~higham/mftoolbox`.

[15] 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, NY, USA, 1981, ACM, pp. 326–333.

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

[17] 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*, Tech. Report TR-08-04, University of Texas at Austin, 2008. FLAME Working Note 29.

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

[19] D. C. Sorensen, *Analysis of pairwise pivoting in Gaussian elimination*, IEEE Transactions on Computers, 3 (1985), p. 274278.

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

[21] L. N. Trefethen and R. S. Schreiber, *Average-case stability of Gaussian elimination*, SIAM J. Matrix Anal. Appl., 11 (1990), pp. 335–360.

[22] V. Volkov, *Private communication.*