Updating an LU factorization with Pivoting

ENRIQUE S. QUINTANA-ORTÍ
Universidad Jaime I
and
ROBERT A. VAN DE GEIJN
The University of Texas at Austin

We show how to compute an LU factorization of a matrix when the factors of a leading principle submatrix are already known. The approach incorporates pivoting akin to partial pivoting, a strategy we call *incremental pivoting*. An implementation using the Formal Linear Algebra Methods Environment (FLAME) Application Programming Interface (API) is described. Experimental results demonstrate practical numerical stability and high performance on an Intel Itanium2 processor based server.

Categories and Subject Descriptors: G.1.3 [**Numerical Analysis**]: Numerical Linear Algebra; G.4 [**Mathematical Software**]: —*Efficiency*

General Terms: Algorithms; Performance

Additional Key Words and Phrases: LU factorization, linear systems, updating, pivoting

1. INTRODUCTION

In this paper we consider the LU factorization of a nonsymmetric matrix, A, partitioned as

$$A \rightarrow \left(\begin{array}{c|c} B & C \\ \hline D & E \end{array}\right) \tag{1}$$

when a factorization of B is to be reused as the other parts of the matrix change. This is known as the updating of an LU factorization.

Applications arising in Boundary Element Methods (BEM) often lead to very large dense linear systems [Cwik et al. 1994; Geng et al. 1996]. For many of these applications the goal is to optimize a feature of an object. For example, BEM may be used to model the radar signature of an airplane. In an effort to minimize this signature, it may be necessary to optimize the shape of a certain component of the airplane. If the degrees of freedom associated with this component are ordered last among all degrees of freedom, the matrix presents the structure given in (1). Now, as the shape of the component is modified, it is only the matrices C, D,

Authors' addresses: Enrique S. Quintana-Ortí, Departamento de Ingeniería y Ciencia de Computadores, Universidad Jaime I, 12.071 – Castellón, Spain, quintana@icc.uji.es. Robert van de Geijn, Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, rvdg@cs.utexas.edu.

and E that change together with the right-hand side vector of the corresponding linear system. Since the dimension of B is frequently much larger than those of the remaining three matrices, it is desirable to factorize B only once and to update the factorization as C, D, and E change. A standard LU factorization with partial pivoting does not provide a convenient solution to this problem, since the rows to be swapped during the application of the permutations may not lie only within B.

Little literature exists on this important topic. We have been made aware that an unblocked out-of-core (OOC) algorithm similar to our algorithm was reported in [Yip 1979], but we have not been able to locate a copy of that report. The proposed addition of this functionality to LAPACK is discussed in [Demmel and Dongarra 2005]. We already discussed preliminary results regarding the algorithm proposed in the current paper in a conference paper [Joffrain et al. 2005], in which its application to OOC LU factorization with pivoting is the main focus[1]. In [Gunter and van de Geijn 2005] the updating of a QR factorization via techniques that are closely related to those proposed for the LU factorization in the current paper is reported.

The paper is organized as follows: in Section 2 we review algorithms for computing the LU factorization with partial pivoting. In Section 3, we discuss how to update an LU factorization by considering the factorization of a 2×2 blocked matrix. The key insight of the paper is found in this section: High-performance blocked algorithms can be synthesized by combining the pivoting strategies of LINPACK and LAPACK. Numerical stability is discussed in Section 4 and performance is reported in Section 5. Concluding remarks are given in the final section.

We hereafter assume that the reader is familiar with Gauss transforms, their properties, and how they are used to factor a matrix. We start indexing elements of vectors and matrices at 0. Capital letters, lower case letter, and lower case Greek letters will be used to denote matrices, vectors, and scalars, respectively. The identity matrix of order n is denoted by I_n.

2. THE LU FACTORIZATION WITH PARTIAL PIVOTING

Given an $n \times n$ matrix A, its LU factorization with partial pivoting is given by $PA = LU$. Here P is a permutation matrix of order n, L is $n \times n$ unit lower triangular, and U is $n \times n$ upper triangular. We will denote the computation of P, L, and U by

$$[A, p] := [\{L \backslash U\}, p] = \mathrm{LU}(A), \tag{2}$$

where $\{L \backslash U\}$ is the matrix whose strictly lower triangular part equals L and whose upper triangular part equals U. Matrix L has ones on the diagonal, which need not be stored, and the factors L and U overwrite the original contents of A. The permutation matrix is generally stored in a vector p of n integers.

Solving the linear system $Ax = b$ now becomes a matter of solving $Ly = Pb$ followed by $Ux = y$. These two stages are referred to as *forward substitution* and *backward substitution*, respectively.

[1] More practical approaches to OOC LU factorization with partial pivoting exist [Toledo 1997; 1999; Toledo and Gustavson 1996; Klimkowski and van de Geijn 1995], which is why that application of the approach is not further mentioned so as not to distract from the central message of this paper.

```
Algorithm: $[A,\ p] := [\{L\backslash U\},\ p] = \mathrm{LU}_{\mathrm{UNB}}(A)$

Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$ and $p \rightarrow \left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right)$
    where $A_{TL}$ is $0 \times 0$ and $p_T$ has 0 elements

while $n(A_{TL}) < n(A)$ do
    Repartition
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ \hline p_2 \end{array}\right)$
        where $\alpha_{11}$ and $\pi_1$ are scalars

    LINPACK variant:
    $\left[\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right), \pi_1\right] := \textsc{Pivot}\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)$
    if $\alpha_{11} \neq 0$ then
        $\left(\begin{array}{c} a_{12}^T \\ \hline A_{22} \end{array}\right) := P(\pi_1)\left(\begin{array}{c} a_{12}^T \\ \hline A_{22} \end{array}\right)$
        $a_{21} := a_{21}/\alpha_{11}$
        $A_{22} := A_{22} - a_{21}a_{12}^T$
    endif

    LAPACK variant:
    $\left[\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right), \pi_1\right] := \textsc{Pivot}\left(\begin{array}{c} \alpha_{11} \\ \hline a_{21} \end{array}\right)$
    if $\alpha_{11} \neq 0$ then
        $\left(\begin{array}{c|c|c} a_{10}^T & & a_{12}^T \\ \hline A_{20} & & A_{22} \end{array}\right) := P(\pi_1)\left(\begin{array}{c|c|c} a_{10}^T & & a_{12}^T \\ \hline A_{20} & & A_{22} \end{array}\right)$
        $a_{21} := a_{21}/\alpha_{11}$
        $A_{22} := A_{22} - a_{21}a_{12}^T$
    endif

    Continue with
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & a_{01} & A_{02} \\ \hline a_{10}^T & \alpha_{11} & a_{12}^T \\ \hline A_{20} & a_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right) \leftarrow \left(\begin{array}{c} p_0 \\ \hline \pi_1 \\ \hline p_2 \end{array}\right)$
endwhile
```

Fig. 1. LINPACK and LAPACK unblocked algorithms for the LU factorization.

2.1 Unblocked right-looking LU factorization

Two unblocked algorithms for computing the LU factorization with partial pivoting are given in Figure 1. There, $n(\cdot)$ stands for the number of columns of a matrix; the thick lines in the matrices/vectors denote how far computation has progressed; $\textsc{Pivot}(x)$ determines the element in x with largest magnitude, swaps that element with the top element, and returns the index of the element that was swapped; and $P(\pi_1)$ is the permutation matrix constructed by interchanging row 0 and row π_1 of the identity matrix. The dimension of a permutation matrix will not be specified since it is obvious from the context in which it is used. We believe the rest of the notation to be intuitive [Bientinesi and van de Geijn 2006; Bientinesi et al. 2005]. Both algorithms correspond to what is usually known as the right-looking variant. Upon completion matrices L and U overwrite A. These algorithms also yield the LU factorization of a matrix with more rows than columns.

The LINPACK variant, $\mathrm{LU}_{\mathrm{UNB}}^{\mathrm{LIN}}$ hereafter, computes the LU factorization as a sequence of Gauss transforms interleaved with permutation matrices:

$$L_{n-1}\left(\begin{array}{c|c} I_{n-1} & 0 \\ \hline 0 & P(\pi_{n-1}) \end{array}\right) \cdots L_1 \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & P(\pi_1) \end{array}\right) L_0 P(\pi_0) A = U.$$

For the LAPACK variant, $\mathrm{LU}_{\mathrm{UNB}}^{\mathrm{LAP}}$, it is recognized that by swapping those rows of matrix L that were already computed and stored to the left of the column that is currently being eliminated, the order of the Gauss transforms and the

```
Algorithm: $[A,\ p] := [\{L\backslash U\},\ p] = \mathrm{LU}_{\mathrm{BLK}}(A)$
Partition $A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right)$ and $p \rightarrow \left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right)$
    where $A_{TL}$ is $0 \times 0$ and $p_T$ has 0 elements
while $n(A_{TL}) < n(A)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right) \rightarrow \left(\begin{array}{c} p_0 \\ \hline p_1 \\ \hline p_2 \end{array}\right)$
        where $A_{11}$ is $b \times b$ and $p_1$ has $b$ elements

    LINPACK variant:
    $\left[\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right),\ p_1\right] := \left[\left(\begin{array}{c} \{L\backslash U\}_{11} \\ \hline L_{21} \end{array}\right),\ p_1\right] = \mathrm{LU}_{\mathrm{UNB}}^{\mathrm{LAP}}\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right)$
    $\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right) := P(p_1)\left(\begin{array}{c} A_{12} \\ \hline A_{22} \end{array}\right)$
    $A_{12} := U_{12} = L_{11}^{-1}A_{12}$
    $A_{22} := A_{22} - L_{21}U_{12}$

    LAPACK variant:
    $\left[\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right),\ p_1\right] := \left[\left(\begin{array}{c} \{L\backslash U\}_{11} \\ \hline L_{21} \end{array}\right),\ p_1\right] = \mathrm{LU}_{\mathrm{UNB}}^{\mathrm{LAP}}\left(\begin{array}{c} A_{11} \\ \hline A_{21} \end{array}\right)$
    $\left(\begin{array}{c|c|c} A_{10} & & A_{12} \\ \hline A_{20} & & A_{22} \end{array}\right) := P(p_1)\left(\begin{array}{c|c|c} A_{10} & & A_{12} \\ \hline A_{20} & & A_{22} \end{array}\right)$
    $A_{12} := U_{12} = L_{11}^{-1}A_{12}$
    $A_{22} := A_{22} - L_{21}U_{12}$

    Continue with
        $\left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} A_{00} & A_{01} & A_{02} \\ \hline A_{10} & A_{11} & A_{12} \\ \hline A_{20} & A_{21} & A_{22} \end{array}\right)$ and $\left(\begin{array}{c} p_T \\ \hline p_B \end{array}\right) \leftarrow \left(\begin{array}{c} p_0 \\ \hline p_1 \\ \hline p_2 \end{array}\right)$
endwhile
```

Fig. 2. LINPACK and LAPACK blocked algorithms for the LU factorization built upon an LAPACK unblocked factorization.

permutation matrices can be rearranged so that $P(p)A = LU$. Here $P(p)$, with $p = (\ \pi_0 \,|\, \cdots \,|\, \pi_{n-1}\ )^T$, denotes the $n \times n$ permutation

$$\left(\begin{array}{c|c} I_{n-1} & 0 \\ \hline 0 & P(\pi_{n-1}) \end{array}\right) \cdots \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & P(\pi_1) \end{array}\right) P(\pi_0).$$

Both algorithms will execute to completion even if an exact zero is encountered on the diagonal of U. This is important since it is possible that matrix B in (1) is singular even if A is not.

The difference between the two algorithms becomes most obvious when forward substitution is performed. For the LINPACK variant forward substitution requires the application of permutations and Gauss transforms to be interleaved. For the LAPACK algorithm, the permutations are applied first on the right-hand side vector, after which a clean lower triangular solve yields the desired (intermediate) result: $Ly = P(p)b$. Depending on whether the LINPACK or the LAPACK variant was used for the LU factorization, we denote the forward substitution stage respectively by $y := \mathrm{FS}^{\mathrm{LIN}}(A,p,b)$ or $y := \mathrm{FS}^{\mathrm{LAP}}(A,p,b)$, where A and p are assumed to contain the outputs of the corresponding factorization.

2.2 Blocked right-looking LU factorization

It is well-known that high performance can be achieved in a portable fashion by casting algorithms in terms of matrix-matrix multiplication [Kågström et al. 1995; Gustavson et al. 1998; Kågström et al. 1998; Gunnels et al. 2001]. In Figure 2 we show LINPACK(-like) and LAPACK blocked algorithms, $\mathrm{LU}_{\mathrm{BLK}}^{\mathrm{LIN}}$ and $\mathrm{LU}_{\mathrm{BLK}}^{\mathrm{LAP}}$ respectively, both built upon an LAPACK unblocked algorithm. The former algorithm really combines the LAPACK style of pivoting, within the factorization of a panel of width b, with the LINPACK style of pivoting. The two algorithms attain high performance on modern architectures with (multiple levels of) cache memory by casting the bulk of the computation in terms of the matrix-matrix multiplication $A_{22} := A_{22} - L_{21}U_{12}$, also called a rank-k update, which is known to achieve high performance [Goto and van de Geijn 2006]. The algorithms also apply to matrices with more rows than columns.

As the LINPACK and LAPACK blocked algorithms are based on the LAPACK unblocked algorithm (which completes even if the current panel is singular), both blocked algorithms will complete even for a singular matrix. If matrix A in (1) is nonsingular, then the upper triangular factor will also be nonsingular, which is what is needed in order to use the factored matrix to solve a linear system.

3. UPDATING AN LU FACTORIZATION

In this section we discuss how to compute the LU factorization of the matrix in (1) in such a way that the LU factorization with partial pivoting of B can be reused if D, C, and E change. We consider A in (1) to be of dimension $n \times n$, with square B and E of orders n_B and n_E, respectively. For reference, factoring the matrix in (1) using the standard LU factorization with partial pivoting costs $\frac{2}{3}n^3$ flops (floating-point arithmetic operations). In this expression (and future computational cost estimates) we neglect insignificant terms of lower-order complexity, including the cost of pivoting the rows.

3.1 Basic procedure

We propose employing the following procedure, consisting of 5 steps, which computes an *LU factorization with incremental pivoting* of the matrix in (1):

Step 1: Factor B. Compute the LU factorization with partial pivoting

$$[B, p] := [\{L \backslash U\}, p] = \mathrm{LU}_{\mathrm{BLK}}^{\mathrm{LAP}}(B).$$

This step is skipped if B was already factored. If the factors are to be used for future updates to C, D, and E, then a copy of U is needed since it is overwritten by subsequent steps.

Step 2: Update C consistent with the factorization of B:

$$C := \mathrm{FS}^{\mathrm{LAP}}(B, p, C).$$

Step 3: Factor $\left(\begin{array}{c} U \\ \hline D \end{array}\right)$**.** Compute the LU factorization with partial pivoting

$$\left[\left(\begin{array}{c} U \\ \hline D \end{array}\right), \bar{L}, r\right] := \left[\left(\begin{array}{c} \{\bar{L}\backslash\bar{U}\} \\ \hline \check{L} \end{array}\right), r\right] = \mathrm{LU}^{\mathrm{LIN}}_{\mathrm{BLK}}\left(\begin{array}{c} U \\ \hline D \end{array}\right).$$

Here $\bar{U}$ overwrites the upper triangular part of B (where U was stored before this operation). The lower triangular matrix $\bar{L}$ that results needs to be stored separately, since both L, computed in Step 1 and used at Step 2, and $\bar{L}$ are needed during the forward substitution stage when solving a linear system.

Step 4: Update $\left(\begin{array}{c} C \\ \hline E \end{array}\right)$ consistent with the factorization of $\left(\begin{array}{c} U \\ \hline D \end{array}\right)$:

$$\left(\begin{array}{c} C \\ \hline E \end{array}\right) := \mathrm{FS}^{\mathrm{LIN}}\left(\left(\begin{array}{c} \bar{L} \\ \hline D \end{array}\right), r, \left(\begin{array}{c} C \\ \hline E \end{array}\right)\right).$$

Step 5: Factor E**.** Finally, compute the LU factorization with partial pivoting

$$[E, s] := \left[\{\tilde{L}\backslash\tilde{U}\}, s\right] = \mathrm{LU}^{\mathrm{LAP}}_{\mathrm{BLK}}(E).$$

Overall, the 5 steps of the procedure apply Gauss transforms and permutations to reduce A to an upper triangular matrix as follows:

$$\left(\begin{array}{c|c} I & 0 \\ \hline 0 & \tilde{L}^{-1}P(s) \end{array}\right) \left(\begin{array}{c|c} \bar{L} & 0 \\ \hline \check{L} & I \end{array}\right)^{-1} P(r) \underbrace{\left(\begin{array}{c|c} L^{-1}P(p) & 0 \\ \hline 0 & I \end{array}\right) \left(\begin{array}{c|c} B & C \\ \hline D & E \end{array}\right)}_{\text{Steps 1 and 2}} =$$

$$\left(\begin{array}{c|c} I & 0 \\ \hline 0 & \tilde{L}^{-1}P(s) \end{array}\right) \underbrace{\left(\begin{array}{c|c} \bar{L} & 0 \\ \hline \check{L} & I \end{array}\right)^{-1} P(r) \left(\begin{array}{c|c} U & \hat{C} \\ \hline D & E \end{array}\right)}_{\text{Steps 3 and 4}} =$$

$$\underbrace{\left(\begin{array}{c|c} I & 0 \\ \hline 0 & \tilde{L}^{-1}P(s) \end{array}\right) \left(\begin{array}{c|c} \bar{U} & \check{C} \\ \hline 0 & \check{E} \end{array}\right)}_{\text{Step 5}} = \left(\begin{array}{c|c} \bar{U} & \check{C} \\ \hline 0 & \tilde{U} \end{array}\right),$$

where $\{L\backslash U\}$, $\left\{\left(\begin{array}{c|c} \bar{L} & 0 \\ \hline \check{L} & I \end{array}\right) \backslash \left(\begin{array}{c} \bar{U} \\ \hline 0 \end{array}\right)\right\}$, and $\{\tilde{L}\backslash\tilde{U}\}$ are the triangular factors computed, respectively, in the LU factorizations in Steps 1, 3, and 5; p, r, and s are the corresponding permutation vectors; $\hat{C}$ is the matrix that results from overwriting C with $L^{-1}P(p)C$; and $\left(\begin{array}{c} \check{C} \\ \hline \check{E} \end{array}\right)$ are the blocks that result from $\left(\begin{array}{c|c} I & 0 \\ \hline 0 & \tilde{L}^{-1}P(s) \end{array}\right) \left(\begin{array}{c} \hat{C} \\ \hline E \end{array}\right)$.

Operation	Approximate cost (in flops)		
	Basic procedure	Structure-Aware LAPACK procedure	Structure-Aware LINPACK procedure
1: Factor B	$\frac{2}{3}n_B^3$	$\frac{2}{3}n_B^3$	$\frac{2}{3}n_B^3$
2: Update C	$n_B^2 n_E$	$n_B^2 n_E$	$n_B^2 n_E$
3: Factor $\left(\frac{U}{D}\right)$	$n_B^2 n_E$ $+\frac{2}{3}n_B^3$	$n_B^2 n_E$ $+\frac{1}{2}bn_B^2$	$n_B^2 n_E$ $+\frac{1}{2}bn_B^2$
4: Update $\left(\frac{C}{E}\right)$	$2n_B n_E^2$ $+n_B^2 n_E$	$2n_B n_E^2$ $+n_B^2 n_E$	$2n_B n_E^2$ $+bn_B n_E$
5: Factor E	$\frac{2}{3}n_E^3$	$\frac{2}{3}n_E^3$	$\frac{2}{3}n_E^3$
Total	$\frac{2}{3}n^3$ $+\frac{2}{3}n_B^3 + n_B^2 n_E$	$\frac{2}{3}n^3$ $+n_B^2\left(\frac{1}{2}b + n_E\right)$	$\frac{2}{3}n^3$ $+bn_B\left(\frac{n_B}{2} + n_E\right)$

Table I. Computational cost (in flops) of the different approaches to compute the LU factorization of the matrix in (1). The highlighted costs are those incurred in excess of the cost of a standard LU factorization.

3.2 Analysis of the basic procedure

For now, the factorization in Step 3 does not take advantage of any zeroes below the diagonal of U: After matrix B is factored and C is updated, the matrix $\left(\begin{array}{c|c} U & C \\ \hline D & E \end{array}\right)$ is factored as if it is a matrix without special structure. Its cost is stated in the column labeled "Basic procedure" in Table I. There we only report significant terms: we assume that $b \ll n_E, n_B$ and report only those costs that equal at least $O(bn_E n_B)$, $O(bn_E^2)$, or $O(bn_B^2)$. *If* n_E *is small* (that is, $n_B \approx n$), the procedure clearly does not benefit from the existence of an already factored B. Also, the procedure requires additional storage for the $n_B \times n_B$ lower triangular matrix $\bar{L}$ computed in Step 3.

We describe next how to reduce both the computational and storage requirements by exploiting the upper triangular structure of U during Steps 3 and 4.

3.3 Exploiting the structure in Step 3

A blocked algorithm that exploits the upper triangular structure of U is given in Figure 3 and illustrated in Figure 4. We name this algorithm $\mathrm{LU}_{\mathrm{BLK}}^{\mathrm{SA-LIN}}$ to reflect that it computes a "Structure-Aware" (SA) LU factorization. At each iteration of the algorithm, the panel of b columns consisting of $\left(\frac{U_{11}}{D_1}\right)$ is factored using the LAPACK unblocked algorithm $\mathrm{LU}_{\mathrm{UNB}}^{\mathrm{LAP}}$. (In our implementation this algorithm is modified to, in addition, take advantage of the zeroes below the diagonal of U_{11}.) As part of the factorization, U_{11} is overwritten by $\{\bar{L}_1 \backslash \bar{U}_{11}\}$. However, in order to preserve the strictly lower triangular part of U_{11} (where part of the matrix L, that was computed in Step 1, is stored), we employ the $b \times b$ submatrix $\bar{L}_1$ of the $n_B \times b$ array $\bar{L}$ (see Figure 3). As in the LINPACK blocked algorithm in Figure 2, the LAPACK and LINPACK styles of pivoting are combined: the current panel of columns are pivoted using the LAPACK approach but the permutations from this factorization are only applied to $\left(\frac{U_{12}}{D_2}\right)$.

The cost of this approach is given in Step 3 of the column labeled "SA LINPACK procedure" in Table I. The cost difference comes from the updates of U_{12} in Figure 3

```
Algorithm: $\left[\left(\begin{array}{c} U \\ \hline D \end{array}\right), \bar{L}, r\right] := \mathrm{LU}_{\mathrm{BLK}}^{\mathrm{SA-LIN}}\left(\begin{array}{c} U \\ \hline D \end{array}\right)$

Partition $U \rightarrow \left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right)$, $D \rightarrow \left(\begin{array}{c|c} D_L & D_R \end{array}\right)$, $\bar{L} \rightarrow \left(\begin{array}{c} \bar{L}_T \\ \hline \bar{L}_B \end{array}\right)$, $r \rightarrow \left(\begin{array}{c} r_T \\ \hline r_B \end{array}\right)$
    where $U_{TL}$ is $0 \times 0$, $D_L$ has 0 columns, $\bar{L}_T$ has 0 rows, and $r_T$ has 0 elements
while $n(U_{TL}) < n(U)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right)$, $\left(\begin{array}{c|c} D_L & D_R \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} D_0 & D_1 & D_2 \end{array}\right)$,
        $\left(\begin{array}{c} \bar{L}_T \\ \hline \bar{L}_B \end{array}\right) \rightarrow \left(\begin{array}{c} \bar{L}_0 \\ \hline \bar{L}_1 \\ \hline \bar{L}_2 \end{array}\right)$, $\left(\begin{array}{c} r_T \\ \hline r_B \end{array}\right) \rightarrow \left(\begin{array}{c} r_0 \\ \hline r_1 \\ \hline r_2 \end{array}\right)$
            where $U_{11}$ is $b \times b$, $D_1$ has $b$ columns, $\bar{L}_1$ has $b$ rows, and $r_1$ has $b$ elements
    ------------------------------------------------------------
    $\left[\left(\begin{array}{c} \{\bar{L}_1 \backslash U_{11}\} \\ \hline D_1 \end{array}\right), r_1\right] := \mathrm{LU}_{\mathrm{UNB}}^{\mathrm{LAP}}\left(\begin{array}{c} U_{11} \\ \hline D_1 \end{array}\right)$
    $\left(\begin{array}{c} U_{12} \\ \hline D_2 \end{array}\right) := P(r_1)\left(\begin{array}{c} U_{12} \\ \hline D_2 \end{array}\right)$
    $U_{12} := \bar{L}_1^{-1} U_{12}$
    $D_2 := D_2 - D_1 U_{12}$
    ------------------------------------------------------------
    Continue with
        $\left(\begin{array}{c|c} U_{TL} & U_{TR} \\ \hline 0 & U_{BR} \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} U_{00} & U_{01} & U_{02} \\ \hline 0 & U_{11} & U_{12} \\ \hline 0 & 0 & U_{22} \end{array}\right)$, $\left(\begin{array}{c|c} D_L & D_R \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} D_0 & D_1 & D_2 \end{array}\right)$,
        $\left(\begin{array}{c} \bar{L}_T \\ \hline \bar{L}_B \end{array}\right) \leftarrow \left(\begin{array}{c} \bar{L}_0 \\ \hline \bar{L}_1 \\ \hline \bar{L}_2 \end{array}\right)$, $\left(\begin{array}{c} r_T \\ \hline r_B \end{array}\right) \leftarrow \left(\begin{array}{c} r_0 \\ \hline r_1 \\ \hline r_2 \end{array}\right)$
endwhile
```

Fig. 3. SA-LINPACK blocked algorithm for the LU factorization of $\left(U^T,\ D^T\right)^T$ built upon an LAPACK blocked factorization.

which, provided $b \ll n_B$, is insignificant compared to $\frac{2}{3}n^3$.

An SA LAPACK blocked algorithm for Step 3 only differs from that in Figure 3 in that, at a certain iteration, after the LU factorization of the current panel is computed, these permutations have to be applied to $\left(\begin{array}{c} U_{10} \\ \hline D_0 \end{array}\right)$ as well. As indicated in Step 3 of the column labeled "SA LAPACK procedure", this does not incur extra cost for *this step*. However, it does require an $n_B \times n_B$ array for storing $\bar{L}$ (see Figure 4) and, as we will see next, makes Step 4 more expensive. On the other hand, the SA LINPACK algorithm only requires a $n_B \times b$ additional work space for

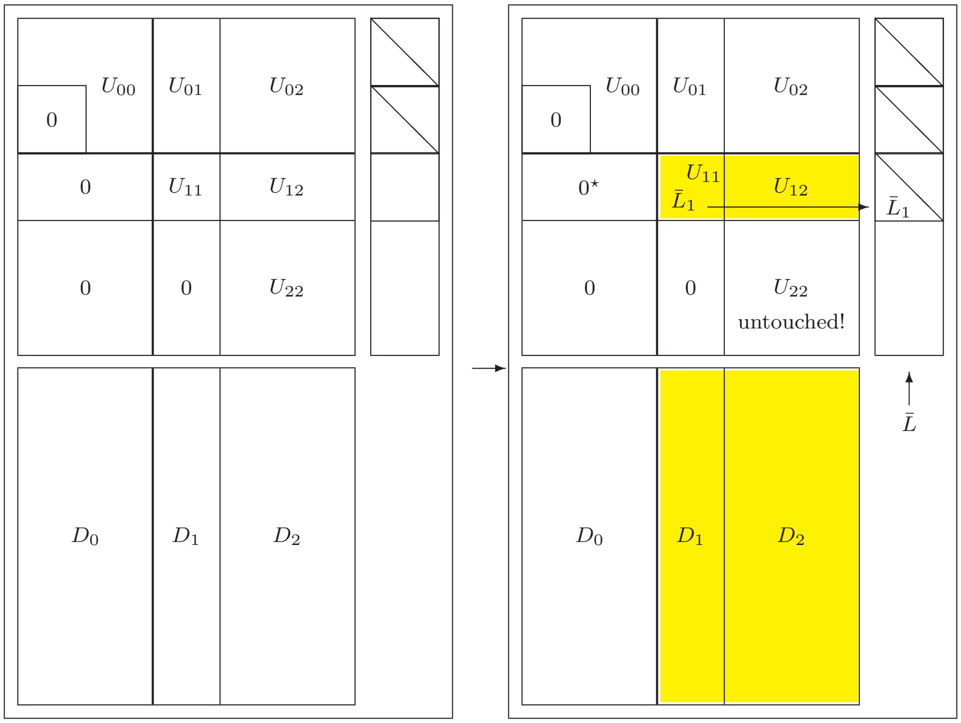

Fig. 4. Illustration of an iteration of the SA LINPACK blocked algorithm used in Step 3 and how it preserves most of the zeroes in U. The zeroes below the diagonal are preserved, except within the $b \times b$ diagonal blocks, where pivoting will fill below the diagonal. The shaded areas are the ones updated as part of the current iteration. The fact that U_{22} is not updated demonstrates how computation can be reduced. If the SA LAPACK blocked algorithm was used, then nonzeroes would appear during this iteration in the block marked as $0^\star$, due to pivoting; as a result, upon completion, zeros would be lost in the full strictly lower triangular part of U.

storing the factors, as indicated in Figure 4.

3.4 Revisiting the update in Step 4

The same optimizations made in Step 3 must now be carried over to the update of $\left(\frac{C}{E}\right)$. The algorithm for this is given in Figure 5. Computation corresponding to zeroes is avoided so that the cost of performing the update is $2n_B n_E^2 + bn_B n_E$ flops, as indicated in Step 4 of Table I.

Applying the SA LAPACK blocked algorithm in Step 3 destroys the structure of the lower triangular matrix, which cannot be recovered during the forward substitution stage in Step 4, and explains the additional cost reported for this variant in Table I.

3.5 Key contribution

The difference in cost of the three different approaches analyzed in Table I is illustrated in Figure 6. It reports the ratios between the costs of the different procedures described above and that of the LU factorization with partial pivoting for a matrix

```
Algorithm: $\left[\left(\begin{array}{c} C \\ \hline E \end{array}\right)\right] := \mathrm{FS}_{\mathrm{BLK}}^{\mathrm{SA-LIN}}\left(\left(\begin{array}{c} \bar{L} \\ \hline D \end{array}\right), r, \left(\begin{array}{c} C \\ \hline E \end{array}\right)\right)$

 Partition $\bar{L} \rightarrow \left(\begin{array}{c} \bar{L}_T \\ \hline \bar{L}_B \end{array}\right)$, $D \rightarrow \left(\begin{array}{c|c} D_L & D_R \end{array}\right)$, $r \rightarrow \left(\begin{array}{c} r_T \\ \hline r_B \end{array}\right)$, $C \rightarrow \left(\begin{array}{c} C_T \\ \hline C_B \end{array}\right)$,
     where $\bar{L}_T$ and $C_T$ have 0 rows, $D_L$ has 0 columns, and $r_T$ has 0 elements
while $n(D_L) < n(D)$ do
    Determine block size $b$
    Repartition
        $\left(\begin{array}{c} \bar{L}_T \\ \hline \bar{L}_B \end{array}\right) \rightarrow \left(\begin{array}{c} \bar{L}_0 \\ \hline \bar{L}_1 \\ \bar{L}_2 \end{array}\right)$, $\left(\begin{array}{c|c} D_L & D_R \end{array}\right) \rightarrow \left(\begin{array}{c|c|c} D_0 & D_1 & D_2 \end{array}\right)$,
        $\left(\begin{array}{c} r_T \\ \hline r_B \end{array}\right) \rightarrow \left(\begin{array}{c} r_0 \\ \hline r_1 \\ r_2 \end{array}\right)$, $\left(\begin{array}{c} C_T \\ \hline C_B \end{array}\right) \rightarrow \left(\begin{array}{c} C_0 \\ \hline C_1 \\ C_2 \end{array}\right)$,
            where $\bar{L}_1$ and $C_1$ have $b$ rows, $D_1$ has $b$ columns,
            and $r_1$ has $b$ elements

    $\left(\begin{array}{c} C_1 \\ \hline E \end{array}\right) := P(r_1)\left(\begin{array}{c} C_1 \\ \hline E \end{array}\right)$
    $C_1 := \bar{L}_1^{-1} C_1$
    $E := E - D_1 C_1$

    Continue with
        $\left(\begin{array}{c} \bar{L}_T \\ \hline \bar{L}_B \end{array}\right) \leftarrow \left(\begin{array}{c} \bar{L}_0 \\ \bar{L}_1 \\ \hline \bar{L}_2 \end{array}\right)$, $\left(\begin{array}{c|c} D_L & D_R \end{array}\right) \leftarrow \left(\begin{array}{c|c|c} D_0 & D_1 & D_2 \end{array}\right)$,
        $\left(\begin{array}{c} r_T \\ \hline r_B \end{array}\right) \leftarrow \left(\begin{array}{c} r_0 \\ r_1 \\ \hline r_2 \end{array}\right)$, $\left(\begin{array}{c} C_T \\ \hline C_B \end{array}\right) \leftarrow \left(\begin{array}{c} C_0 \\ C_1 \\ \hline C_2 \end{array}\right)$,
endwhile
```

Fig. 5. SA-LINPACK blocked algorithm for the update of $\left(C^T,\ E^T\right)^T$ consistent with the SA-LINPACK blocked LU factorization of $\left(U^T,\ D^T\right)^T$.

with $n_B = 1000$ and different values of n_E using $b = 32$. The analysis shows that the overhead of the SA LINPACK procedure is consistently low. On the other hand, as $n_E/n \rightarrow 1$ the cost of the basic procedure, which is initially twice as expensive as that of the LU factorization with partial pivoting, is decreased. The SA LAPACK procedure only presents a negligible overhead when $n_E \rightarrow 0$ that is, when the dimension of the update is very small.

The key insight of the proposed approach is the recognition that combining LINPACK- and LAPACK-style pivoting allows one to use a blocked algorithm while avoiding filling most of the zeroes in the lower triangular part of U. This, in turn, makes the extra cost of Step 4 acceptable. In other words, for the SA LINPACK

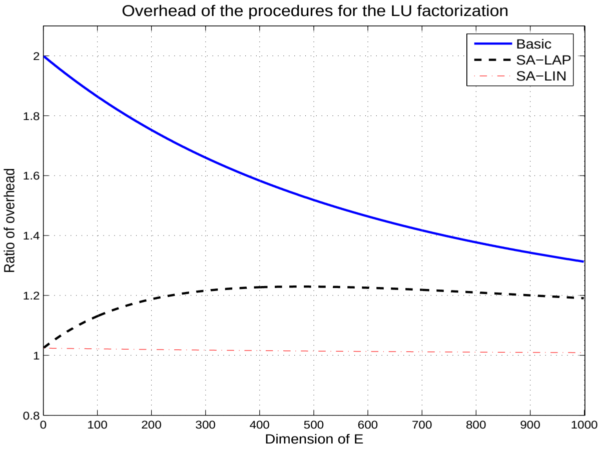

Fig. 6. Overhead cost of the different approaches to compute the LU factorization in (1) with respect to the cost of the LU factorization with partial pivoting.

procedure, the benefit of the higher performance of the blocked algorithm comes at the expense of a lower-order amount of extra computation. The extra memory for the SA LINPACK procedure consists of an $n_B \times n_B$ upper triangular matrix and an $n_B \times b$ array.

4. REMARKS ON NUMERICAL STABILITY

The algorithm for the LU factorization with incremental pivoting carries out a sequence of row permutations (corresponding to the application of permutations) which are different from those that would be performed in an LU factorization with partial pivoting. Therefore, the numerical stability of this algorithm is also different. In this section we provide some remarks on the stability of the new algorithm. We note that all three procedures described in the previous section (basic, SA LINPACK, and SA LAPACK) perform the same sequence of row permutations.

The numerical (backward) stability of an algorithm that computes the LU factorization of a matrix A depends on the growth factor [Stewart 1998]

$$\rho = \frac{\|L\|\|U\|}{\|A\|}, \tag{3}$$

which is basically determined by the problem size and the pivoting strategy. For example, the growth factors of complete, partial, and *pairwise* ([Wilkinson 1965, p. 236]) pivoting have been demonstrated to be bounded as $\rho_c \leq n^{1/2}(2{\cdot}3^{1/2}\cdots n^{1/n-1})$, $\rho_p \leq 2^{n-1}$, and $\rho_w \leq 4^{n-1}$, respectively [Sorensen 1985; Stewart 1998]. Statistical models and extensive experimentations in [Trefethen and Schreiber 1990] showed that, on average, $\rho_c \approx n^{1/2}$, $\rho_p \approx n^{2/3}$, and $\rho_w \approx n$, inferring that in practice partial/pairwise pivoting are both numerically stable, and pairwise pivoting can be expected to numerically behave only slightly worse than partial pivoting.

The new algorithm applies partial pivoting during the factorization of B and then

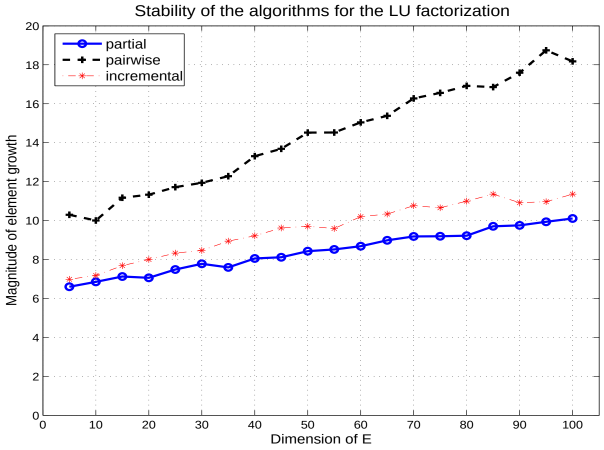

Fig. 7. Element growth in the LU factorization using different pivoting techniques.

again in the factorization of $\begin{pmatrix} U \\ D \end{pmatrix}$. This can be considered as a blocked variant of pairwise pivoting. Thus, we can expect an element growth for the algorithm that is between those of partial and pairwise pivoting. Next we elaborate an experiment that provides evidence in support of this observation.

In Figure 7 we report the element growths observed during the computation of the LU factorization of matrices as in (1), with $n_B = 100$ and dimensions for E ranging from $n_E = 5$ to 100 using partial, incremental, and pairwise pivoting. The entries of the matrices are generated randomly, chosen from a uniform distribution in the interval $(0.0, 1.0)$. The experiment was carried out on an Intel Xeon processor using MATLAB® 7.0.4 (IEEE double-precision arithmetic). The results report the average element growth for 100 different matrices for each matrix dimension. The figure shows that the growth factor of incremental pivoting is smaller than that of pairwise pivoting and approximates that of partial pivoting. A similar behavior was obtained for other matrix types: uniform distribution in $(-1.0, 1.0)$, normal distribution with mean 0.0 and deviation 1.0 (N$[0.0, 1.0]$), symmetric matrices with elements in N$[0.0, 1.0]$, and Toeplitz matrices with elements in N$[0.0, 1.0]$. Only for orthogonal matrices with Haar distribution [Trefethen and Schreiber 1990], we obtained significantly different results. In that case, incremental pivoting attained a smaller element growth than pairwise pivoting, and both outperformed the element growth of partial pivoting. Explaining the behavior of this case is beyond the scope of this paper.

For those who are not sufficiently satisfied with the element growth of incremental pivoting, we propose to perform a few refinement iterations of the solution to $Ax = b$, at a cost of $O(n^2)$ flops per step, as this guarantees stability at a low computational cost [Higham 2002].

5. PERFORMANCE

In this section we report results for a high-performance implementation of the SA LINPACK procedure.

5.1 Implementation

The FLAME library (Version 0.9) was used to implement a high-performance LU factorization with partial pivoting and the SA LINPACK procedure. The benefit of this API is that the code closely resembles the algorithms as they are presented in Figures 1–3 and 5. The performance of the FLAME LU factorization with partial pivoting is highly competitive with LAPACK and vendor implementations of this operation.

The implementations can be examined by visiting

`http://www.cs.utexas.edu/users/flame/Publications/` .

5.2 Platform

Performance experiments were performed in double-precision arithmetic on a Intel Itanium2 (1.5 GHz) processor based workstation capable of attaining 6 GFLOPS (10^9 flops per second). For reference, the algorithm for the FLAME LU factorization with partial pivoting delivered 4.8 GFLOPS for a 2000 × 2000 matrix. A block size $b = 128$ was employed in this procedure for all experiments reported next. The implementation was linked to the GotoBLAS R1.6 Basic Linear Algebra Subprograms (BLAS) library [Goto 2004]. The BLAS routine DGEMM which is used to compute $C := C - AB$ ($C \in \mathbb{R}^{m \times n}$, $A \in \mathbb{R}^{m \times k}$, and $B \in \mathbb{R}^{k \times n}$) attains the best performance when the common dimension of A and B, k, is equal to 128. Notice that most computation in the SA LINPACK procedure is cast in terms of this operation, with $k = b$.

The performance benefits reported on this platform are representative of the benefits that can be expected on other current architectures.

5.3 Results

In Figure 8(top) we show the speedup attained when an existing factorization of B is reused by reporting the time required to factor (1) with the high-performance LU factorization with partial pivoting divided by the time required to update an existing factorization of B via the SA LINPACK procedure (Steps 2-5). In that figure, $n_B = 1000$ and n_E is varied from 0 to 1000. The results are reported when different block sizes b are chosen. The DGEMM operation, in terms of which most computation is cast, attains the best performance when $b = 128$ is chosen. However, this generates enough additional flops that the speedup is better when b is chosen to be smaller. When n_E is very small, $b = 8$ (for Steps 2–5) yields the best performance. As n_E increases, performance improves by choosing $b = 32$ (for Steps 2–5).

The effect of the overhead of the extra computations is demonstrated in Figure 8(bottom). There, we report the ratio of the time required by Steps 1-5 of the SA LINPACK procedure divided by the time required by the LU factorization with partial pivoting of (1). The results in the figure may be somewhat disturbing: The algorithm that views the matrix as four quadrants attains as good, or even

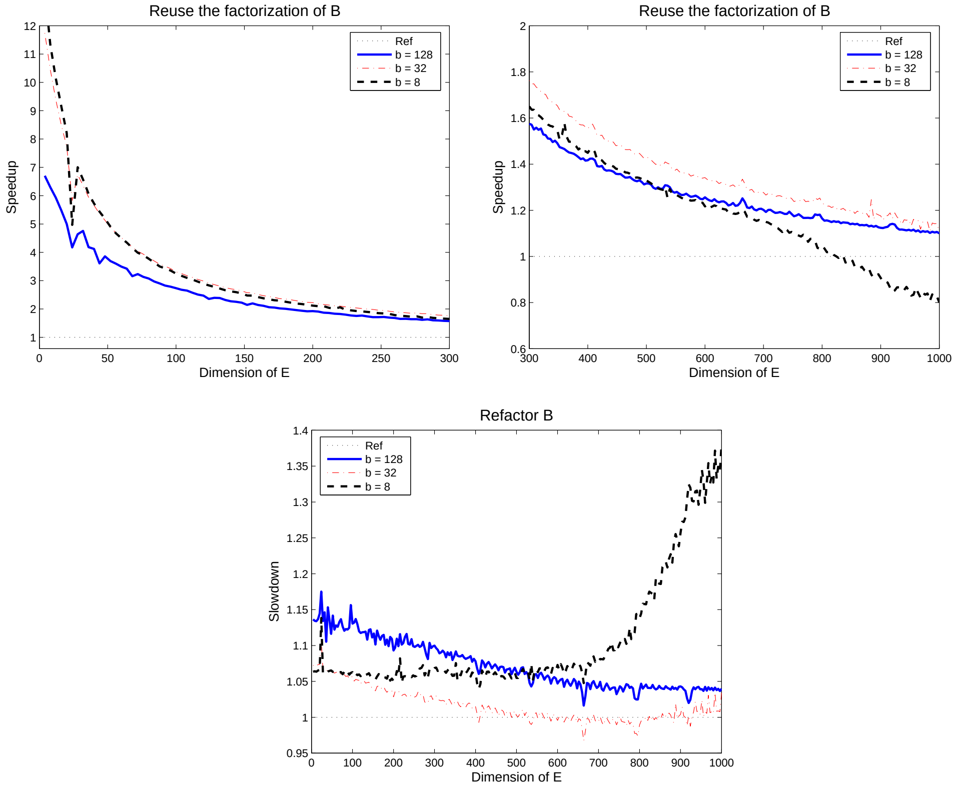

Fig. 8. Top: Speedup attained when B is not refactored, over LU factorization with partial pivoting of the entire matrix. Bottom: Slowdown for the first factorization (when B must also be factored).

better, performance than the algorithm that views the matrix as a single unit and performs less computation. The likely explanation is that the standard LU factorization would also benefit from a variable block size as the problem size changes, rather than fixing it at $b = 128$. We did not further investigate this issue since we did not want to make raw performance the primary focus of the paper.

6. CONCLUSIONS

We have proposed blocked algorithms for updating an LU factorization. They have been shown to attain high performance and to greatly reduce the cost of an update to a matrix for which a partial factorization already exists. The key insight is the synthesis of LINPACK- and LAPACK-style pivoting. While some additional computation is required, this is more than offset by the improvement in performance that comes from casting computation in terms of matrix-matrix multiplication.

We acknowledge that the question of the numerical stability of the new algorithm relative to that of LU factorization with partial pivoting remains open. Strictly speaking, the LU factorization with partial pivoting is itself not numerically stable and it is practical experience that has shown it to be effective in practice. Theoretical results that rigorously bound the additional element growth in the LU factorization with incremental pivoting are in order, but are beyond the scope of the present paper.

Acknowledgments

This research was partially sponsored by NSF grants ACI-0305163, CCF-0342369 and CCF-0540926, and an equipment donation from Hewlett-Packard. Primary support for this work came from the *J. Tinsley Oden Faculty Fellowship Research Program* of the Institute for Computational Engineering and Sciences (ICES) at UT-Austin.

For further information on FLAME visit `www.cs.utexas.edu/users/flame`.

REFERENCES

Bientinesi, P., Gunnels, J. A., Myers, M. E., Quintana-Ortí, E. S., and van de Geijn, R. A. 2005. The science of deriving dense linear algebra algorithms. *ACM Trans. Math. Soft. 31,* 1 (March), 1–26.

Bientinesi, P. and van de Geijn, R. 2006. Representing dense linear algebra algorithms: A farewell to indices. Tech. Rep. FLAME Working Note 17, CS-TR-2006-10, Department of Computer Sciences, The University of Texas at Austin.

Cwik, T., van de Geijn, R., and Patterson, J. 1994. The application of parallel computation to integral equation models of electromagnetic scattering. *Journal of the Optical Society of America A 11,* 4 (April), 1538–1545.

Demmel, J. and Dongarra, J. 2005. LAPACK 2005 prospectus: Reliable and scalable software for linear algebra computations on high end computers. LAPACK Working Note 164 UT-CS-05-546, University of Tennessee. February.

Geng, P., Oden, J. T., and van de Geijn, R. 1996. Massively parallel computation for acoustical scattering problems using boundary element methods. *Journal of Sound and Vibration 191,* 1, 145–165.

Goto, K. 2004. `http://www.tacc.utexas.edu/resources/software/`.

Goto, K. and van de Geijn, R. 2006. Anatomy of high-performance matrix multiplication. *ACM Trans. Math. Soft.*. Submitted.

Gunnels, J. A., Henry, G. M., and van de Geijn, R. A. 2001. A family of high-performance matrix multiplication algorithms. In *Computational Science - ICCS 2001, Part I*, V. N. Alexandrov, J. J. Dongarra, B. A. Juliano, R. S. Renner, and C. K. Tan, Eds. Lecture Notes in Computer Science 2073. Springer-Verlag, 51–60.

Gunter, B. and van de Geijn, R. 2005. Parallel out-of-core computation and updating of the QR factorization. *ACM Trans. Math. Soft. 31,* 1 (March), 60–78.

Gustavson, F., Henriksson, A., Jonsson, I., Kågström, B., and Ling, P. 1998. Superscalar GEMM-based level 3 BLAS – the on-going evolution of a portable and high-performance library. In *Applied Parallel Computing, Large Scale Scientific and Industrial Problems*, B. K. et al., Ed. Lecture Notes in Computer Science 1541. Springer-Verlag, 207–215.

Higham, N. J. 2002. *Accuracy and Stability of Numerical Algorithms*, Second ed. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

Joffrain, T., Quintana-Ortí, E. S., and van de Geijn, R. A. 2005. Rapid development of high-performance out-of-core solvers. In *PARA 2004*, J. Dongarra, K. Madson, and J. Waśniewski, Eds. LNCS 3732. Springer-Verlag, 413–422.

Kågström, B., Ling, P., and Loan, C. V. 1995. Gemm-based level 3 blas: High-performance model, implementations and performance evaluation benchmark. LAPACK Working Note #107 CS-95-315, Univ. of Tennessee. Nov.

Kågström, B., Ling, P., and Loan, C. V. 1998. GEMM-based level 3 BLAS: High performance model implementations and performance evaluation benchmark. *ACM Trans. Math. Soft. 24,* 3, 268–302.

Klimkowski, K. and van de Geijn, R. 1995. Anatomy of an out-of-core dense linear solver. In *Proceedings of the International Conference on Parallel Processing 1995*. Vol. III - Algorithms and Applications. 29–33.

Sorensen, D. C. 1985. Analysis of pairwise pivoting in Gaussian elimination. *IEEE Trans. on Computers c-34,* 3, 274–278.

Stewart, G. W. 1998. *Matrix Algorithms. Volume I: Basic Decompositions.* SIAM, Philadelphia.

Toledo, S. 1997. Locality of reference in lu decomposition with partial pivoting. *SIAM Journal on Matrix. Anal. Appl. 18,* 4.

Toledo, S. 1999. A survey of out-of-core algorithms in numerical linear algebra. In *External Memory Algorithms and Visualization*, J. Abello and J. S. Vitter, Eds. American Mathematical Society Press, Providence, RI, 161–180.

Toledo, S. and Gustavson, F. 1996. The design and implementation of SOLAR, a portable library for scalable out-of-core linear algebra computations. In *Proceedings of the fourth workshop on I/O in parallel and distributed systems*. 28–40.

Trefethen, L. N. and Schreiber, R. S. 1990. Average-case stability of Gaussian elimination. *SIAM J. Matrix Anal. Appl. 11,* 3, 335–360.

Wilkinson, J. H. 1965. *The Algebraic Eigenvalue Problem.* Oxford University Press, London.

Yip, E. L. 1979. Fortran subroutines for Out-of-Core solutions of large complex linear systems. Tech. Rep. CR-159142, NASA.