Stability of Block Algorithms with Fast Level-3 BLAS

JAMES W. DEMMEL
University of California
and
NICHOLAS J. HIGHAM
University of Manchester

Block algorithms are becoming increasingly popular in matrix computations. Since their basic unit of data is a submatrix rather than a scalar, they have a higher level of granularity than point algorithms, and this makes them well suited to high-performance computers. The numerical stability of the block algorithms in the new linear algebra program library LAPACK is investigated here. It is shown that these algorithms have backward error analyses in which the backward error bounds are commensurate with the error bounds for the underlying level-3 BLAS (BLAS3). One implication is that the block algorithms are as stable as the corresponding point algorithms when conventional BLAS3 are used. A second implication is that the use of BLAS3 based on fast matrix multiplication techniques affects the stability only insofar as it increases the constant terms in the normwise backward error bounds. For linear equation solvers employing LU factorization, it is shown that fixed precision iterative refinement helps to mitigate the effect of the larger error constants. Despite the positive results presented here, not all plausible block algorithms are stable; we illustrate this with the example of LU factorization with block triangular factors and describe how to check a block algorithm for stability without doing a full error analysis.

Categories and Subject Descriptors: G.1 3 [**Numerical Analysis**]: Numerical Linear Algebra

General Terms: Algorithms

Additional Key Words and Phrases: Backward error analysis, block algorithm, iterative refinement, LAPACK, level-3 BLAS, LU factorization, QR factorization

1. INTRODUCTION

A *block algorithm* in matrix computations is defined in terms of operations on submatrices rather than matrix elements. Such algorithms are well suited

The work of J. W. Demmel was supported by NSF grants DCR-8552474 and ASC-8715728. He is also a Presidential Young Investigator.

Authors' addresses: J. W. Demmel, Computer Science Division and Mathematics Department, University of California, Berkeley, CA 94720, e-mail: na.demmel@na-net.ornl.gov; N. J. Higham, Department of Mathematics, University of Manchester, Manchester, M13 9PL, UK, e-mail: na.nhigham@na-net.ornl. gov

to many high-performance computers because their data locality properties lead to efficient usage of memory hierarchies [16, 17, 18, Ch. 1].

When a block algorithm is coded in FORTRAN, advantage can be taken of the level-3 Basic Linear Algebra Subprograms (BLAS3). The BLAS3 are a set of FORTRAN primitives for various types of matrix multiplication, together with solution of a triangular system with multiple right-hand sides [11, 12]. For a suitably coded block algorithm, the bulk of the computation is carried out by calls to the BLAS3.

The BLAS3 specifications [11] stipulate the input, output, and call sequence for each routine, but allow freedom of implementation, subject to the requirement that the routines be numerically stable. This freedom includes not only the various ways to order matrix multiplication, but the use of algorithms algebraically different from the conventional ones. Of chief interest here are algorithms that achieve a more favorable operation count (for suitable dimensions) through the use of a fast matrix multiplication technique. We refer to such BLAS3 implementations as "fast BLAS3."

One set of fast BLAS3 is proposed in [21]. There it is shown how asymptotic speedups can be produced in all the BLAS3 routines by the use of Strassen's method for matrix multiplication [32], which forms the product of two $n \times n$ matrices in $O(n^{\log_2 7})$ operations ($\log_2 7 \approx 2.807$). A set of fast BLAS3 can also be built from Winograd's matrix multiplication method [36] (which has an operation count of $O(n^3)$ with different constants than the conventional technique) or one of the methods with a lower exponent than Strassen's (although the practical utility of the latter methods has yet to be demonstrated [23]). In the case of complex matrices, all these possibilities can be combined with the technique analyzed in [25], which enables the product of two complex matrices to be formed using only three real matrix multiplications. Several researchers are experimenting with the use of fast BLAS3 in linear equation solvers. In particular, we mention the work of Bailey et al. [3], who use Strassen's method for the matrix multiplications arising in the LAPACK *LU* factorization routine SGETRF.

Our purpose is to investigate the numerical stability of block algorithms that employ fast BLAS3. We restrict our attention mainly to the block algorithms used in LAPACK [4, 9]. For block size 1, the algorithms in LAPACK are classical point algorithms that are well known to be numerically stable; that is, each computed answer is the exact answer to a perturbed problem, where the norm of the perturbation is bounded by the product of the unit roundoff, a modest constant depending on the dimensions and the norm of the data. (To be precise, this statement is true modulo the possibility of a large growth factor in Gaussian elimination with partial pivoting, and a weaker definition of stability for matrix inversion [15].) For block sizes $r > 1$, with conventional BLAS3, it is generally accepted that the same stability results are valid, although we are not aware of any detailed proofs (in the case of block *LU* factorization one can argue that the same arithmetic operations are carried out as for $r = 1$, albeit in a different order). The question of particular interest here is the effect on the stability of using *fast* BLAS3 when $r > 1$. We show that, for all BLAS3 implementations of interest, backward error bounds hold for the block algorithms that are commensurate with the error bounds for the BLAS3 themselves. This is clearly the best we could expect to prove.

As our model for floating-point arithmetic, we take

$$fl(x \pm y) = x(1+\alpha) \pm y(1+\beta), \quad |\alpha|, |\beta| \le u,$$
$$fl(x \operatorname{op} y) = (x \operatorname{op} y)(1+\delta), \quad |\delta| \le u, \operatorname{op} = *, /,$$

where u is the unit roundoff. We need to make some assumptions about the stability of the BLAS3. The BLAS3 primitives involve three types of matrix multiplication: a general product AB, a cross product A^TA, and the product of a triangular matrix with a full matrix. It is sufficient to assume that all these products satisfy the following general condition: if $A \in \mathbb{R}^{m\times n}$, $B \in \mathbb{R}^{n\times p}$ and $\hat{C}$ is the computed approximation to $C = AB$, then

$$\hat{C} = AB + \Delta C, \quad \|\Delta C\| \le c_1(m,n,p)u\|A\|\,\|B\| + O(u^2), \tag{1.1}$$

where $c_1(m,n,p)$ denotes a constant depending on m, n, and p. Here the matrix norm is defined by

$$\|X\| = \max_{i,j}|x_{ij}|.$$

Note that for this norm, with A and B dimensioned as above, $\|AB\| \le n\|A\|\,\|B\|$ is the best such inequality.

We also assume that the computed solution $\hat{X}$ to the triangular systems $TX = B$, where $T \in \mathbb{R}^{m\times m}$ and $B \in \mathbb{R}^{m\times p}$, satisfies

$$T\hat{X} = B + \Delta B, \quad \|\Delta B\| \le c_2(m,p)u\|T\|\,\|\hat{X}\| + O(u^2). \tag{1.2}$$

For conventional BLAS3 implementations, conditions (1.1) and (1.2) hold with $c_1(m,n,p) = n^2$ and $c_2(m,p) = m(m+1)$ [21].

For the fast BLAS3 proposed in [21], based on Strassen's method, (1.1) and (1.2) hold with c_1 and c_2 rather complicated functions of the dimensions m, n, p, and the threshold n_0 that determines the level of recursion. In the special case, $m = n = p = 2^k$, $n_0 = 2^l$, the constants reduce to [21]:

$$c_1(n,n,n) = \left(\frac{n}{n_0}\right)^{\log_2 12}(n_0^2 + 5n_0) - 5n,$$

$$c_2(n,n) = \left(\frac{n}{n_0}\right)^{\log_2 12}\left(\frac{n_0^2}{11} + \frac{23}{55}n_0\right) + \frac{10}{11}n_0^2 + \frac{35}{11}n_0 - \frac{143}{55}n.$$

Condition (1.1) also holds when the multiplication is done by Winograd's method with scaling, or, in the case of complex matrices, by the method of [25] combined with any method for real matrix multiplication that satisfies (1.1) (see the error analysis in [6] and [25]).

Note that for conventional multiplication we have the following componentwise version of (1.1):

$$\hat{C} = AB + \Delta C, \quad |C| \le nu|A||B| + O(u^2). \tag{1.3}$$

Here $|A|$ denotes the matrix $(|a_{ij}|)$ and inequalities hold componentwise. Similarly, for the substitution algorithms for solving triangular systems, we have

$$T\hat{X} = B + \Delta B, \quad |\Delta B| \le (m+1)u|T||\hat{X}| + O(u^2). \tag{1.4}$$

We stress that these bounds are much stronger than (1.1) and (1.2). For example, if $D = \operatorname{diag}(d_i)$ with $d_i > 0$ for all i, then scaling $AB \to AD \cdot D^{-1}B$ leaves C and the upper bound in (1.3) unchanged, and scaling $AB \to DA \cdot BD$ causes the bound of (1.3) to scale in the same way as C; (1.1) does not share these favorable scaling properties. For further remarks on the differences between (1.1) and (1.3), see [21]. A consequence of (1.3) and (1.4) is that for some block algorithms it is possible to obtain stronger backward error results than the usual normwise ones (perhaps for certain classes of matrix only); for examples see [8] and [22]. These stronger results are usually not valid for any of the fast BLAS3 discussed above.

The block algorithms in LAPACK break into two main classes: those based on LU factorization and those involving orthogonal transformations. In the next section we give an error analysis of block LU factorization. We show that iterative refinement in fixed precision is beneficial for all BLAS3 implementations and point out the instability of LU factorization with block triangular factors. We also explain how to investigate the stability of a block algorithm without doing a full error analysis. In Section 3 we consider the use of aggregated Householder transformations, which form the basis of a variety of block algorithms involving orthogonal transformations. Some concluding remarks are given in Section 4.

Block algorithms for matrix inversion are analyzed by DuCroz and Higham [15], so we do not consider them here. It is shown in [15] that one particular block algorithm for inverting a triangular matrix is unstable, even though the point algorithm from which it is derived is stable; this serves to emphasize that stability of block algorithms cannot be taken for granted.

2. LU FACTORIZATION

2.1 Error Analysis

In this section we examine in detail the stability of block LU factorization. Initially we assume that no pivoting is used and that the factorization succeeds; below we discuss the addition of pivoting.

Consider a block implementation of the outer product form of LU factorization [17, p. 91], [18, p. 100]. The algorithm may be described through the partitioning

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} L_{11} & 0 \\ L_{21} & I_{n-r} \end{bmatrix} \begin{bmatrix} I_r & 0 \\ 0 & B \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ 0 & I_{n-r} \end{bmatrix} \in \mathbb{R}^{n \times n},$$

where A_{11} is $r \times r$. One step of the block algorithm consists of factoring $A_{11} = L_{11}U_{11}$, solving the multiple right-hand side triangular systems $L_{11}U_{12} = A_{12}$ and $L_{21}U_{11} = A_{21}$ for U_{12} and L_{21}, respectively, and forming

$B = A_{22} - L_{21}U_{12}$. This procedure is then repeated on B. The block operations defining U_{12}, L_{21}, and B are level 3 BLAS operations.

We assume that the block level LU factorization is done in such a way that the computed LU factors of $A_{11} \in \mathbb{R}^{r \times r}$ satisfy

$$\hat{L}_{11}\hat{U}_{11} = A_{11} + \Delta A_{11}, \quad \|\Delta A_{11}\| \le c_3(r)u\|\hat{L}_{11}\|\|\hat{U}_{11}\| + O(u^2). \tag{2.1}$$

We claim that under these assumptions, together with (1.1) and (1.2), the LU factors of $A \in \mathbb{R}^{n \times n}$ computed using a block size r satisfy

$$\hat{L}\hat{U} = A + \Delta A, \quad \|\Delta A\| \le u\big(\delta(n,r)\|A\| + \theta(n,r)\|\hat{L}\|\,\|\hat{U}\|\big) + O(u^2), \tag{2.2}$$

where $\delta(n,r)$ and $\theta(n,r)$ are constants depending on n and r. The proof is essentially inductive. For $n = r$, (2.2) holds with

$$\delta(r,r) = 0, \quad \theta(r,r) = c_3(r), \tag{2.3}$$

in view of (2.1). Consider the first block stage of the factorization. The assumptions imply that

$$\hat{L}_{11}\hat{U}_{12} = A_{12} + \Delta A_{12}, \quad \|\Delta A_{12}\| \le c_2(r, n-r)u\|\hat{L}_{11}\|\,\|\hat{U}_{12}\| + O(u^2), \tag{2.4}$$

$$\hat{L}_{21}\hat{U}_{11} = A_{21} + \Delta A_{21}, \quad \|\Delta A_{21}\| \le c_2(r, n-r)u\|\hat{L}_{21}\|\,\|\hat{U}_{11}\| + O(u^2). \tag{2.5}$$

To obtain $B = A_{22} - L_{21}U_{12}$ we first compute $C = \hat{L}_{21}\hat{U}_{12}$, obtaining

$$\hat{C} = \hat{L}_{21}\hat{U}_{12} + \Delta C, \quad \|\Delta C\| \le c_1(n-r, r, n-r)u\|\hat{L}_{21}\|\,\|\hat{U}_{12}\| + O(u^2),$$

and then subtract from A_{22}, obtaining

$$\hat{B} = A_{22} - \hat{C} + F, \quad \|F\| \le u\big(\|A_{22}\| + \|\hat{C}\|\big) + O(u^2). \tag{2.6}$$

It follows that

$$\begin{aligned} \hat{B} &= A_{22} - \hat{L}_{21}\hat{U}_{12} + \Delta B, \\ \|\Delta B\| &\le u\big(\|A_{22}\| + \|\hat{L}_{21}\|\,\|\hat{U}_{12}\| + c_1(n-r, r, n-r)\|\hat{L}_{21}\|\,\|\hat{U}_{12}\|\big) \\ &\quad + O(u^2). \end{aligned} \tag{2.7}$$

The remainder of the algorithm consists of the computation of the LU factorization of $\hat{B}$, and by our inductive assumption (2.2), the computed LU factors satisfy

$$\begin{aligned} \hat{L}_{22}\hat{U}_{22} &= \hat{B} + \Delta\hat{B}, \\ \|\Delta\hat{B}\| &\le \delta(n-r, r)u\|\hat{B}\| + \theta(n-r, r)u\|\hat{L}_{22}\|\,\|\hat{U}_{22}\| + O(u^2). \end{aligned} \tag{2.8}$$

Combining (2.7) and (2.8), and bounding $\|\hat{B}\|$ using (2.6), we obtain

$$\begin{aligned}\hat{L}_{21}\hat{U}_{12} + \hat{L}_{22}\hat{U}_{22} &= A_{22} + \Delta A_{22},\\ \|\Delta A_{22}\| \le u\Big([1 + \delta(n-r,r)]\|A_{22}\| &\\ &+ [1 + c_1(r, n-r, n-r) + \delta(n-r,r)]\|\hat{L}_{21}\|\,\|\hat{U}_{12}\|\\ &+ \theta(n-r,r)\|\hat{L}_{22}\|\,\|\hat{U}_{22}\|\Big) + O(u^2). \qquad (2.9)\end{aligned}$$

Collecting together (2.1), (2.4), (2.5), and (2.9), we have

$$\hat{L}\hat{U} = A + \Delta A, \qquad (2.10)$$

where bounds on $\|\Delta A_{ij}\|$ are given in the equations just mentioned. These bounds for the blocks of ΔA can be weakened slightly and expressed together in the more succinct form

$$\|\Delta A\| \le u\big(\delta(n,r)\|A\| + \theta(n,r)\|\hat{L}\|\,\|\hat{U}\|\big) + O(u^2) \qquad (2.11)$$

where

$$\begin{aligned}\delta(n,r) &= 1 + \delta(n-r,r),\\ \theta(n,r) &= \max\{c_3(r), c_2(r,n-r),\\ &\qquad 1 + c_1(r,n-r,n-r) + \delta(n-r,r) + \theta(n-r,r)\}.\end{aligned}$$

Using (2.3) it follows that $\delta(n,r) \le n/r$.

These recurrences show that the basic error constants in assumptions (1.1), (1.2), and (2.1) combine additively at worst. Thus the backward error analysis for the LU factorization is commensurate with the error analysis for the particular implementation of the BLAS3 employed in the block factorization. In the case of the conventional BLAS3, we obtain a generalization of the classical Wilkinson result for $r = 1$, with $\theta(n,r) = O(n^3)$.

Although the above analysis is phrased in terms of the block outer product form of LU factorization, the same result holds for other "*ijk*" block forms (with slightly different constants), for example the gaxpy or sdot forms.

If we incorporate partial pivoting in the above factorization, then two of the block steps are coalesced: L_{11} and L_{21} are obtained by using Gaussian elimination with partial pivoting (GEPP) to compute the factorization

$$P_1 = \begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} = \begin{bmatrix} L_{11} \\ L_{21} \end{bmatrix} U_{11}. \qquad (2.12)$$

Thus each main step of the algorithm involves two, rather than three, BLAS3 operations. If the obvious analog of (2.1) holds for (2.12), then (2.10) (with A replaced by PA) and (2.11) remain valid, with minor changes in the recurrences for $\delta(n,r)$ and $\theta(n,r)$.

There is no difficulty in extending the analysis to cover solution of $Ax = b$ using the computed LU factorization. Invoking the usual error analysis for substitution (see [18, Sec. 3.1], for example), we find that $(A + \Delta A)\hat{x} = b$, with

$$\|\Delta A\| \le u\big(\delta(n,r)\|A\| + (\theta(n,r) + 2n^2)\|\hat{L}\|\,\|\hat{U}\|\big) + O(u^2). \quad (2.13)$$

For a linear system with multiple right-hand sides, $AX = B$ with $X, B \in \mathbb{R}^{n\times p}$, both substitution stages are BLAS3 operations. Using (1.2), it is straightforward to show that the computed $\hat{X}$ satisfies

$$\begin{aligned}\|A\hat{X} - B\| \le &\Big\{\big[c_2(n,p)(n^2+n) + \theta(n,r)n\big]\|\hat{L}\|\,\|\hat{U}\| + \delta(n,r)n\|A\|\Big\}\\ &\times\|\hat{X}\|u + O(u^2). \qquad (2.14)\end{aligned}$$

We mention that the error analysis given in this section adapts in a straightforward way to block LU factorization for banded matrices, block factorization of symmetric indefinite matrices [18, p. 168], and block Cholesky factorization of (banded) symmetric positive definite matrices [26].

2.2 Iterative Refinement

The LAPACK routines for solving linear equations support fixed precision iterative refinement [10], that is, iterative refinement in which no extra precision is used in calculating the residuals. The benefits of this process can be explained in terms of the componentwise relative backward error $\omega(y)$ of an approximate solution y to $Ax = b$. This quantity is defined by

$$\begin{aligned}\omega(y) &\equiv \min\{\epsilon: (A + \Delta A)y = b + \Delta b, |\Delta A| \le \epsilon|A|, |\Delta b| \le \epsilon|b|\},\\ &= \max_i \frac{|b - Ay|_i}{(|A||y| + |b|)_i}, \qquad (2.15)\end{aligned}$$

where the latter equality is proved by Oettli and Prager [27]. A small value for $\omega(y)$ implies that y is the solution of a system in which each element of A and b has undergone a small relative perturbation—in particular, zero elements are not perturbed. However, in general, all that can be guaranteed for the $\hat{x}$ from GEPP is that the normwise relative backward error $\eta(\hat{x}) = O(u)$, where

$$\begin{aligned}\eta(y) &\equiv \min\{\epsilon: (A + \Delta A)y = b + \Delta b, \|\Delta A\|_\infty \le \epsilon\|A\|_\infty, \|\Delta b\|_\infty \le \epsilon\|b\|_\infty\}\\ &= \frac{\|r\|_\infty}{\|A\|_\infty\|y\|_\infty + \|b\|_\infty}. \qquad (2.16)\end{aligned}$$

Skeel [31] showed that as long as A is not too ill-conditioned and the components of the vector $|A||x|$ do not vary too much in magnitude, then $\omega(y) = O(u)$ for the vector y obtained from GEPP with one step of fixed precision iterative refinement. Skeel's result, together with further analysis by Arioli et al. [1], provides the theoretical foundation for the inclusion of fixed precision iterative refinement in LAPACK.

In LAPACK, iterative refinement is terminated if

(1) $\omega \le u$,

(2) ω has not decreased by a factor of at least 2 from the previous iteration, or

(3) five iterations have been performed.

These criteria have been chosen to be robust in the fact of different BLAS implementations and machine arithmetics.

Skeel's result is applicable only when conventional BLAS3 are used. To investigate the effect of using fast BLAS3, we can make use of work by Higham [24] that covers fixed precision iterative refinement with an arbitrary linear equation solver. For our purposes, the results in [24] require a bound of the form $|b - A\hat{x}| \le uG|\hat{x}|$ for the given solver, where G is a nonnegative matrix. For block LU factorization with partial pivoting, we have

$$|b - A\hat{x}| \le |\Delta A||\hat{x}|,$$

where $\|\Delta A\|$ is bounded in (2.13). We assume that the residual for the refinement step is computed in the conventional way, via inner products or saxpy operations as in LAPACK. Since we have only a normwise bound on ΔA, we cannot apply a direct generalization in [24] of Skeel's result (to do so we would need to have $|\Delta A| \le uG|A|$ with $\|G\|$ bounded independently of A). However, we can invoke the weaker Theorem 2.1 in [24] to obtain

$$|b - A\hat{y}| \le (n + 2)u(|A||\hat{y}| + |b|) + O(u^2), \qquad (2.17)$$

where $\hat{y}$ is the computed vector obtained after one step of fixed precision iterative refinement. This result has two main features. First, it is asymptotic, and the second order term prevents us concluding from (2.17) and (2.15) that $\omega(\hat{y}) \le (n + 2)u$. However, if the components of $|A||\hat{y}| + |b|$ do not vary too much in magnitude, it is likely that this inequality will be satisfied (if not, extra refinement steps may help to achieve a small ω). The second point is that ΔA does not appear in the first order term of (2.17)—it is hidden in the second order term where it multiplies a vector with elements of $O(u)$. This means that the refinement step tends to suppress any instability manifested in ΔA.

LAPACK also supports iterative refinement for linear systems with multiple right-hand sides, $AX = B$ where $X, B \in \mathbb{R}^{n \times p}$. In this case it is appropriate to consider whether a small componentwise relative backward error is achieved for each individual system $Ax_i = b_i$, $i = 1{:}p$. If conventional BLAS3 are used for the computation of $\hat{X}$ and for the refinement process, then Skeel's result is applicable to each system $Ax_i = b_i$.

Suppose that fast BLAS3 are used in computing $\hat{X}$. First, we obtain a bound of the form $|b_i - A\hat{x}_i| \le uG_i|\hat{x}_i|$ for each i. It is necessary to do this in an indirect way, as follows. We note first that (2.14) implies

$$\|A\hat{x}_i - b_i\|_\infty \le M\mu_i u\|A\|_\infty\|\hat{x}_i\|_\infty + O(u^2), \quad i = 1{:}p,$$

where

$$M = \left[c_2(n,p)(n^2 + n) + \theta(n,r)n\right]\frac{\|\hat{L}\|\|\hat{U}\|}{\|A\|_\infty} + \delta(n,r)n,$$

$$\mu_i = \frac{\|\hat{X}\|}{\|\hat{x}_i\|_\infty} \ge 1.$$

From (2.16), it follows that

$$(A + \Delta A_i)\hat{x}_i = b_i, \quad \|\Delta A_i\|_\infty \le M\mu_i u\|A\|_\infty + O(u^2). \tag{2.18}$$

Hence we have $|b_i - A\hat{x}_i| \le |\Delta A_i||\hat{x}_i|$, with $\|\Delta A_i\|_\infty$ bounded as in (2.18). In invoking Theorem 2.1 of [24], we need to specify how the residuals $R = B - AX$ are computed and how the corrections for the refinement are computed. We assume that the residuals are computed using conventional BLAS3. If fast BLAS3 are used, we can do no better than to obtain a *normwise* bound for each $b_i - A\hat{y}_i$ that is proportional to the fast BLAS3 error constant c_1 (this is not surprising, since it is a general principle for iterative techniques that the stability or accuracy is limited by the quality of the computed residuals).

If we use fast BLAS3 for the substitutions on the correction step, then for each computed correction we have a result analogous to (2.18). Theorem 2.1 of [24] than shows that (2.17) holds for b_i and $\hat{y}_i$, but the potentially very large μ_i term, and its analog for the refinement step, are present in the second order term of (2.17), making the result of limited practical value. If the substitutions are done using conventional BLAS3, then (2.17) holds for b_i and $\hat{y}_i$, with μ_i alone present in the second order term; hence we would expect a small $\omega(\hat{y}_i)$ as long as μ_i is not too large. Finally, we note that if all the substitutions in the computation of $\hat{X}$ and in the refinement process are done with conventional BLAS3, then we can set $\mu_i = 1$ in the above analysis and obtain the same computed results as if the refinement were carried out on each system $Ax_i = b_i$ independently.

Our overall conclusion is that fixed precision iterative refinement can be beneficial for GEPP with fast BLAS3 in two ways, assuming that residuals are computed using conventional BLAS3. First, it may lead to a componentwise relative backward error of order u, although the theoretical backing is weaker than when conventional BLAS3 are used. Second, the refinement will, in any case, tend to counteract any "mild instability" induced by the potentially faster growth of errors in the fast BLAS3.

We present some numerical results for illustration. Our experiments were performed in MATLAB, for which $u \approx 2.2 \times 10^{-16}$. We solved $Ax = b$ by block outer product LU factorization with partial pivoting, using both conventional BLAS3 and a fast BLAS3; the latter uses conventional triangular solves and does matrix multiplication by Strassen's method with $n_0 = 1$ (recursion down to the scalar level). Iterative refinement was applied with the convergence test $\omega \le u$.

We give detailed results for three matrices taken from the test collection [19]: pascal(n) is a symmetric positive definite matrix constructed from the elements of Pascal's triangle; triw(n, α) is upper triangular with 1s on the diagonal and every entry in the upper triangle equal to α; and ipjfact(n, 1) is the symmetric positive definite matrix with (i, j) entry $1/(i + j)!$. In each case b was chosen randomly, with elements from the uniform distribution on [0, 1]. Tables I–III show the componentwise relative backward errors ω for the iterates, with the normwise relative backward errors η in parentheses;

Table I. A = pascal(8)

$\kappa_\infty(A) = 3.96 \times 10^7$, $\text{cond}(A) = 4.60 \times 10^6$

Conv., $r = 1$.	Conv., $r = 2$.	Fast, $r = 2$.
3.03e-16 (2.62e-18)	1.15e-16 (3.54e-18)	1.74e-14 (1.21e-16)
4.91e-17 (8.60e-18)		4.92e-17 (8.60e-18)

Table II. A = triw(16, $-5)^T$

$\kappa_\infty(A) = 3.57 \times 10^{13}$, $\text{cond}(A) = 9.40 \times 10^{11}$

Conv., $r = 1$.	Conv., $r = 2$.	Fast, $r = 2$.
8.42e-17 (4.28e-19)	8.42e-17 (4.28e-19)	3.12e-9 (4.24e-19)
		1.68e-16 (1.27e-18)

Table III. A = ipjfact (7, 1)

$\kappa_\infty(A) = 1.69 \times 10^{14}$, $\text{cond}(A) = 6.85 \times 10^{10}$

Conv., $r = 1$.	Conv., $r = 2$.	Fast, $r = 2$.
8.45e-16 (1.51e-20)	3.68e-16 (1.23e-20)	2.09e-12 (5.49e-20)
1.98e-17 (1.07e-20)	5.89e-18 (3.19e-21)	7.07e-18 (2.15e-21)

the column heading "Conv." denotes conventional BLAS3. We also report the condition numbers $\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty$ and $\text{cond}(A) = \| \, |A^{-1}| \, |A| \, \|_\infty$.

In these three specially chosen examples, all the η values for the original computed solution are less than u, but the ω values for the fast BLAS3 are substantially larger than for the conventional BLAS3. Note how iterative refinement reduces the ω values below u in one step in the first three examples. More typical, less extreme behavior is illustrated in Table IV, where rand(n) is a random matrix with elements from the uniform $[0, 1]$ distribution.

The matrices in the first three examples each have nonnegative elements of widely varying magnitude. These matrices were tried because it is known that Strassen's method can provide poor relative accuracy when forming the product of such matrices in floating-point arithmetic [21]. It is interesting to recall the result that the computed solution to $Ax = b$ obtained by GEPP in floating-point arithmetic is invariant under row or column scalings by powers of the machine base, as long as the same pivot sequence is chosen [7, p. 181].

Table IV. A = rand(32)

$\kappa_\infty(A) = 6.00 \times 10^2$, $\text{cond}(A) = 3.33 \times 10^2$

Conv., $r = 1$.	Conv., $r = 8$.	Fast, $r = 8$.
1.48e-16 (2.67e-17)	2.34e-16 (4.56e-17)	6.81e-16 (1.07e-16)
	2.31e-17 (7.16e-18)	2.99e-17 (8.38e-18)

This invariance property does not hold when Strassen's method is used in the BLAS3—this observation provides some further insight into the first three examples.

2.3 Block-Triangular *LU* Factorization

Next, we discuss the computation of a true block LU factorization $A = LU \in \mathbb{R}^{n \times n}$, where L and U are block lower triangular and block upper triangular, respectively [17, 29]. This factorization is not used in LAPACK; we consider it here because it provides a salutary example of how a plausible block algorithm can be unstable.

Assuming that $A_{11} \in \mathbb{R}^{r \times r}$ is nonsingular, we can write

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} I & 0 \\ L_{21} & I \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ 0 & B \end{bmatrix} = LU, \qquad (2.19)$$

which leads to the following algorithm for computing the block triangular factors L and U:

(1) Solve $L_{21}A_{11} = A_{21}$ for L_{21}.
(2) $B = A_{22} - L_{21}A_{12}$.
(3) Compute the block LU factorization of B recursively.

There is some freedom in how step 1 is accomplished. Assume for the moment that L_{21} is computed via Gaussian elimination with partial pivoting and that conventional BLAS3 are used throughout. Then it is straightforward to show that the computed LU factors $\hat{L}$ and $\hat{U}$ satisfy $\hat{L}\hat{U} = A + \Delta A$, where ΔA satisfies a bound of the form

$$|\Delta A| \le u\big(\delta(n,r)|A| + \theta(n,r)|\hat{L}||\hat{U}|\big) + O(u^2). \qquad (2.20)$$

This is a componentwise analog of (2.11). The same form of bound (2.20) holds for standard LU factorization (without pivoting); one might therefore surmise that block LU factorization has similar stability to standard LU factorization. However, the stability is not at all satisfactory, as the following numerical experiment makes clear.

We implemented two versions of the block LU algorithm in MATLAB, using conventional BLAS3. Version Gepp uses Gaussian elimination with partial pivoting in step 1. Version Inv explicitly computes A_{11}^{-1} using Gaussian elimination with partial pivoting and then forms $L_{21} = A_{21} \times A_{11}^{-1}$; such

Table V. Relative residuals for block LU factorization

$$\kappa_\infty(A_{16}(-2)) = O(10^{16})$$
$$\kappa_\infty(D_{16}(10^{-4})) = O(10^{15})$$

	Moler matrix $A_{16}(-2)$		Dorr matrix $D_{16}(10^{-4})$	
r	Version Gepp	Version Inv	Version Gepp	Version Inv
1	0.00	0.00	5.89e-17	2.94e-17
2	0.00	0.00	2.92e-14	2.49e-14
3	6.64e-17	3.36e-16	1.06e-12	1.05e-12
4	2.56e-16	2.40e-15	7.11e-12	1.75e-11
5	3.67e-16	1.47e-14	7.59e-09	1.58e-09
6	1.18e-15	1.13e-13	1.51e-10	1.47e-10
7	4.09e-15	5.95e-13	1.42e-05	1.15e-05
8	1.66e-15	1.83e-11	1.28e-19	2.33e-19
9	1.02e-14	1.18e-10	9.98e-17	8.52e-17
10	1.14e-13	4.82e-10	3.67e-14	3.03e-14
11	1.56e-13	1.93e-8	1.54e-12	9.49e-13
12	7.63e-13	1.11e-7	1.52e-10	1.48e-10
13	3.89e-13	1.54e-6	3.10e-8	2.84e-8
14	1.71e-12	8.49e-6	8.98e-6	6.30e-6
15	2.95e-11	1.13e-4	5.44e-4	4.53e-4

use of explicit inverses can lead to greater efficiency on parallel machines [17, 29].

We report results for two matrices from [19]: the symmetric positive definite Moler matrix $A_n(\alpha) = \mathrm{triw}(n,\alpha)^T \mathrm{triw}(n, \alpha) \in \mathbb{R}^{n \times n}$, where $\mathrm{triw}(n, \alpha)$ is defined in Section 2.2, and the Dorr matrix $D_n(\alpha)$, which is an unsymmetric row diagonally dominant tridiagonal matrix. ($D_n(\alpha)$ has diagonal dominance factors $\gamma_i := |d_{ii}| - |d_{i,i-1}| - |d_{i,i+1}| = (n+1)^2\alpha$ for $i = 1, n$ and $\gamma_i = 0$ otherwise; we perturbed the diagonal elements $d_{22}, \dots, d_{n-1,n-1}$ to ensure that $\gamma_i \geq 10^{-15}$ for the computed matrix). The relative residuals $\|A - \hat{L}\hat{U}\|_\infty / \|A\|_\infty$ for the block LU factorization are displayed in Table V for block sizes $r = 1, 2, \dots, n-1$. The results reveal instability of both implementations; the instability increases steadily with the block size for the Moler matrix, but is less regular for the Dorr matrix. Interestingly, for the transpose of the Dorr matrix in Table V, which is diagonally dominant by columns, all the residuals are of order the unit roundoff (in contrast, standard LU factorization performs equally well on row and column diagonally dominant tridiagonal matrices [20]).

The essential reason for this instability is that the norm of the term $|\hat{L}||\hat{U}|$ in (2.20) depends on the condition of constituents of Schur complements of A, and, consequently, $\| |\hat{L}||\hat{U}| \|_\infty$ can greatly exceed $\|A\|_\infty$. For example, if $n - r \le r$, then there is just one block stage of the factorization, and from (2.19) we have

$$|L||U| = \begin{bmatrix} I & 0 \\ |A_{21}A_{11}^{-1}| & I \end{bmatrix} \begin{bmatrix} |A_{11}| & |A_{12}| \\ 0 & |A_{22} - A_{21}A_{11}^{-1}A_{12}| \end{bmatrix},$$

whose (2,1) submatrix is $|A_{21}A_{11}^{-1}||A_{11}|$. For standard LU factorization, the corresponding submatrix $(|\tilde{L}||\tilde{U}|)_{21} = |A_{21}\tilde{U}_{11}^{-1}||\tilde{U}_{11}|$, where $A_{11} = \tilde{L}_{11}\tilde{U}_{11}$. If A is symmetric positive definite, for example, then $|A_{21}\tilde{U}_{11}^{-1}||\tilde{U}_{11}|$ is guaranteed to have ∞-norm of order $\|A\|_\infty$ in view of standard results (see, for example, [18, Sect. 4.2]), but $|A_{21}A_{11}^{-1}||A_{11}|$ can have a much larger norm. For the Moler matrix $A_{16}(-2)$, with $r = 12$, $\| |A_{21}\tilde{U}_{11}^{-1}||\tilde{U}| \|_\infty = 288$, while $\| |A_{21}A_{11}^{-1}||A_{11}| \|_\infty = 1.6 \times 10^7$ ($\|A_{16}(-2)\|_\infty = 455$).

The instability of block LU factorization has not, to our knowledge, been pointed out before in the literature. However, we know of applications with diagonally dominant matrices in which the factorization has been found to perform stably. An interesting topic for further research is to identify particular classes of matrix for which the factorization is guaranteed to be stable. For example, it is clear that block LU factorization performs stably when A is symmetric positive definite and sufficiently well conditioned.

2.4 Recognizing a Stable Block Decomposition

How can we distinguish a stable block algorithm from an unstable one? Here we present an informal approach that can allow easy recognition without the need for a full error analysis. Our approach has some similarities with that of Wilkinson [35], who describes a general approach to analyzing unblocked algorithms.

We consider LU factorization with partial pivoting, although the same approach applies to the other factorizations mentioned at the end of Section 2.1 and to QR factorization. We view the algorithm as a sequence of computed decompositions

$$P_i L_i U_i + R_i = A + \Delta A_i, \quad i = 0:m, \tag{2.21}$$

where one or more such decompositions describe a single step of the block algorithm. P_i is a permutation matrix and ΔA_i represents rounding errors introduced by the first i steps of the algorithm. Initially, $P_0 = L_0 = I$, $U_0 = A$, and $R_0 = \Delta A_0 = 0$, while L_m is unit lower triangular, U_m is upper triangular, and $R_m = 0$. The term R_i is introduced for notational convenience.

The matrices P_i, L_i, U_i, and R_i are transformed to P_{i+1}, L_{i+1}, U_{i+1} and R_{i+1} by a single matrix operation, either unblocked (BLAS1 or BLAS2) or blocked (BLAS3). The rounding error introduced by this matrix operation is $\Delta A_{i+1} - \Delta A_i$. If each ΔA_i is small ($\|\Delta A_i\| = O(u\|A\|)$), then the algorithm is stable.

To illustrate, the first stage of the block LU factorization in Section 2.1 (ignoring pivoting) may be written, using (2.21) with $i = 0{:}4$, as

$$\begin{aligned}
A &= I\begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix} \\
&= \begin{bmatrix} L_{11} & 0 \\ 0 & I \end{bmatrix}\begin{bmatrix} U_{11} & 0 \\ A_{21} & A_{22} \end{bmatrix} + \begin{bmatrix} 0 & A_{12} \\ 0 & 0 \end{bmatrix} + \begin{bmatrix} \Delta A_{11} & 0 \\ 0 & 0 \end{bmatrix} \\
&= \begin{bmatrix} L_{11} & 0 \\ 0 & I \end{bmatrix}\begin{bmatrix} U_{11} & U_{12} \\ A_{21} & A_{22} \end{bmatrix} + \begin{bmatrix} \Delta A_{11} & \Delta A_{12} \\ 0 & 0 \end{bmatrix} \qquad (2.22) \\
&= \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix}\begin{bmatrix} U_{11} & U_{12} \\ 0 & -L_{21}U_{12} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & A_{22} \end{bmatrix} + \begin{bmatrix} \Delta A_{11} & \Delta A_{12} \\ \Delta A_{21} & 0 \end{bmatrix} \\
&= \begin{bmatrix} L_{11} & 0 \\ L_{21} & I \end{bmatrix}\begin{bmatrix} U_{11} & U_{12} \\ 0 & B \end{bmatrix} + \begin{bmatrix} \Delta A_{11} & \Delta A_{12} \\ \Delta A_{21} & \Delta A_{22} \end{bmatrix}.
\end{aligned}$$

It is clear from the assumptions (1.1) and (1.2) that each rounding error term ΔA_{ij} has a bound of order $u\|A\|$ as long as there is no undue element growth, and (2.22) shows that the contribution of the ΔA_{ij} is additive. It follows that the whole process is stable. (Note that an analog of (2.22) holds for the method of section 2.3, with $|\Delta A|$ bounded as in (2.20), but here undue element growth can make $|L||U| \gg |A|$.)

One possible obstacle to stability, which does not occur in LAPACK, would be computing part of a decomposition (say $A_{11} = L_{11}U_{11}$, above), using it to compute other parts of the decomposition (L_{21}, U_{12}, and B), and then recomputing it by some method yielding different rounding errors. There is no motivation for this in Gaussian elimination, but it is conceivable that such redundant operations would be needed to use the BLAS3. (There are redundant operations in the use of block orthogonal transformations in the next section, but they do not lead to this difficulty). How could this recomputation damage stability? Suppose we refactorize $A_{11} \approx \hat{L}_{11}\hat{U}_{11}$ stably after the last step above, and replace L_{11} by $\hat{L}_{11}$ and U_{11} by $\hat{U}_{11}$. This will change ΔA_{12} by $(\hat{L}_{11} - L_{11})U_{12}$ and ΔA_{21} by $L_{21}(\hat{U}_{11} - U_{11})$, and neither of these quantities is guaranteed to be small.

3. ORTHOGONAL TRANSFORMATIONS

In this section we consider block algorithms based on orthogonal transformations. The algorithms of interest include QR factorization, orthogonal reduction to Hessenberg, tridiagonal or bidiagonal form, the unsymmetric QR algorithm, and algorithms for generalized eigenvalue or singular value computations. The techniques used in LAPACK for constructing block versions of these algorithms are based on the aggregation of Householder transformations. Our aim is therefore to analyze the stability of these aggregation techniques.

One form of aggregation is the "WY" representation of Bischof and Van Loan [5]. This involves representing the product $Q_r = P_r P_{r-1} \dots P_1$ of r Householder transformations $P_i = I - u_i u_i^T \in \mathbb{R}^{n \times n}$ $(u_i^T u_i = 2)$ in the form

$$Q_r = I + W_r Y_r^T, \quad W_r, Y_r \in \mathbb{R}^{n \times r}.$$

This can be done using the recurrence

$$W_1 = -u_1, Y_1 = u_1, \quad W_i = [W_{i-1}, -u_i], Y_i = [Y_{i-1}, Q_{i-1}^T u_i].$$

Using the WY representation, a block QR factorization can be developed as follows. Partition $A \in \mathbb{R}^{m \times n} (m \geq n)$ as

$$A = [A_1, B], \quad A_1 \in \mathbb{R}^{m \times r},$$

and compute the Householder QR factorization of A_1,

$$P_r P_{r-1} \dots P_1 A_1 = \begin{bmatrix} R_1 \\ 0 \end{bmatrix}.$$

The product $P_r P_{r-1} \dots P_1 = I + W_r Y_r^T$ is accumulated as the P_i are generated and then B is updated according to

$$B \leftarrow (I + W_r Y_r^T) B = B + W_r (Y_r^T B),$$

which involves only BLAS3 operations. The process is now repeated on the last $m - r$ rows of B.

An alternative form of accumulation is proposed in [14] for $r = 2$, extended to general r in [13], and used in [2]. In the context of orthogonal similarity reduction to Hessenberg form, the technique involves expressing

$$P_r P_{r-1} \dots P_1 A P_1 \dots P_{r-1} P_r = A - U_r V_r^T - W_r U_r^T, \tag{3.1}$$

where $U_r, V_r, W_r \in \mathbb{R}^{n \times r}$. We refer the reader to [13] for details of how to obtain this representation. The key point is that, once again, only BLAS3 operations are involved in computing the update, once U_r, V_r, and W_r have been formed.

Now we consider numerical stability. We concentrate on the WY technique, and comment that similar analysis applies to the alternative method of aggregation (3.1), as well as to the more storage efficient compact WY representation of Schreiber and Van Loan [30]. First, we note that the construction of the W and Y matrices is done in a stable manner (indeed, it does not involve the BLAS3): Bischof and Van Loan [5] show that the computed $\hat{Q} = I + \hat{W}\hat{Y}^T$ is such that

$$\|\hat{Q}^T \hat{Q} - I\| = O(u), \tag{3.2}$$

$$\|\hat{W}\| = O(1), \quad \|\hat{Y}\| = O(1). \tag{3.3}$$

Condition (3.2) implies that

$$\hat{Q} = U + \Delta U, \quad U^T U = I, \quad \|\Delta U\| = O(u), \tag{3.4}$$

that is, $\hat{Q}$ is close to an exactly orthogonal matrix.

Next we consider the application of $\hat{Q}$. Suppose we form $C = \hat{Q}B = (I + \hat{W}\hat{Y}^T)B$, obtaining

$$\hat{C} = fl\big(B + fl(\hat{W}(\hat{Y}^T B))\big).$$

Analyzing this BLAS3-based computation using (1.1) and (3.4), it is straightforward to show that

$$\begin{gathered} \hat{C} = UB + \Delta C = U(B + U^T \Delta C), \\ \|\Delta C\| \le c_4 n(c_1(r, n, n) + c_1(n, r, n))u\|B\| + O(u^2), \end{gathered} \tag{3.5}$$

where c_4 is a constant of order 1. This result shows that the computed update is an exact orthogonal update of a perturbation of B, where the norm of the perturbation is bounded in terms of the error constants for the BLAS3.

In the case $r = 1$, (3.5) reduces to Wilkinson's result on the application of a single Householder transformation [34, p. 160]. Wilkinson uses this result to obtain a backward error result for the application of a sequence of Householder transformations [34, pp. 160–161]. With the use of (3.5), it is straightforward to show that Wilkinson's method of analysis can be adapted to accommodate WY updates. Alternatively, to obtain a backward error result for a sequence of orthogonal similarity transformations, we can simply insert the bound (3.5) into the general analysis of Parlett [28, Sec. 6-5] or Wilkinson [33]. It follows that the standard backward error analysis results for Householder transformation algorithms remain valid when the WY technique is used, as long as the constants in the error bounds are replaced by appropriate linear combinations of c_1 terms.

Our overall conclusions are as follows. First, algorithms that employ aggregated Householder transformations with conventional BLAS3 are as stable as the corresponding point algorithms. Second, the use of fast BLAS3 for applying the updates affects stability only through the constants in the backward error bounds.

4. CONCLUDING REMARKS

Our main conclusion is that the use of fast BLAS3 satisfying (1.1) and (1.2) with LAPACK block algorithms is "safe" from a numerical standpoint. The algorithms retain their normwise backward stability, but the actual backward errors may increase to reflect any decreased accuracy in the BLAS3. Also, any special properties relating to componentwise backward error may be lost in switching to fast BLAS3.

How large the increase in backward errors will be, on average, is difficult to say, since the theoretical error bounds tend to be quite pessimistic (this applies particularly to the bound for Strassen's method—see [21]). A quantitative assessment of the speed versus stability tradeoff must await the accrual of experience in using fast BLAS3 for the values of n for which useful speedups are obtained.

In the important application of solving $Ax = b$ by LU factorization, the use of fast BLAS3 need carry no stability penalty: we have shown that fixed precision iterative refinement with conventionally computed residuals can improve normwise stability and can even produce a small componentwise relative backward error, although we have not been able to state useful conditions under which such improvements are guaranteed.

Finally, we reiterate a point discussed in [21]. When replacing conventional BLAS3 by fast BLAS3 in an *iterative* algorithm, it is important to consider the implications for the convergence tests—a change in the BLAS3 may necessitate a retuning of the algorithm parameters. Ideally, a convergence test will be effective across different BLAS3 implementations and computer arithmetics and so will not need tuning. The stopping criterion used in LAPACK for iterative refinement of linear equations (see Section 2.2) has been designed with this goal in mind. Experience will tell whether the goal has been achieved!

ACKNOWLEDGMENTS

We thank a referee for comments that lead to a substantial improvement of Section 2.3.

REFERENCES

1. ARIOLI, M., DEMMEL, J. W., AND DUFF, I. S. Solving sparse linear systems with sparse backward error. *SIAM J. Matrix Anal. Appl. 10* (1989), 165–190.
2. BAI, Z., AND DEMMEL, J. W. On a block implementation of Hessenberg multishift QR iteration. *Int. J. High Speed Comput. 1* (1989), 97–121.
3. BAILEY, D. H., LEE, K., AND SIMON, H. D. Using Strassen's algorithm to accelerate the solution of linear systems. *J. Supercomput. 4* (1991), 357–371.
4. BISCHOF, C. H., AND DONGARRA, J. J. A project for developing a linear algebra library for high-performance computers. Preprint MCS-P105-0989, Mathematics and Computer Science Div., Argonne National Lab., 1989.
5. BISCHOF, C. H., AND VAN LOAN, C. F. The WY representation for products of Householder matrices. *SIAM J. Sci. Stat. Comput. 8* (1987), s2–s13.
6. BRENT, R. P. Error analysis of algorithms for matrix multiplication and triangular decomposition using Winograd's identity. *Numer. Math. 16* (1970), 145–156.
7. DAHLQUIST, G., AND BJÖRCK, Å. *Numerical Methods.* Prentice-Hall, Englewood Cliffs, N.J., 1974.
8. DEMMEL, J. W. On floating point errors in Cholesky. LAPACK Working Note 14, Dept. of Computer Science, Univ. of Tennessee, Knoxville, 1989.
9. DEMMEL, J. W., DONGARRA, J J., DU CROZ, J. J., GREENBAUM, A., HAMMARLING, S. J., AND SORENSEN, D. C. Prospectus for the development of a linear algebra library for high-performance computers. Tech. Mem. 97, Mathematics and Computer Science Div., Argonne National Lab., 1987.
10. DEMMEL, J W., DU CROZ, J. J., HAMMARLING, S. J., AND SORENSON, D. C. Guidelines for the design of symmetric eigenroutines, SVD, and iterative refinement and condition estimation for linear systems. LAPACK Working Note 4, Tech. Memor. 111, Mathematics and Computer Science Div., Argonne National Lab., 1988.
11. DONGARRA, J. J., DU CROZ, J. J., DUFF, I. S., AND HAMMARLING, S. J. A set of Level 3 basic linear algebra subprograms. *ACM Trans. Math. Softw. 16* (1990), 1–17.

12. DONGARRA, J. J., DU CROZ, J. J., DUFF, I. S. AND HAMMARLING, S. J. Algorithm 679. A set of Level 3 basic linear algebra subprograms: Model implementation and test programs. *ACM Trans. Math. Sotfw. 16* (1990), 18–28.
13. DONGARRA, J. J., HAMMARLING, S. J., AND SORENSEN, D. C. Block reduction of matrices to condensed forms for eigenvalue computations. *J. Comput. Appl. Math. 27* (1989), 215–227.
14. DONGARRA, J. J., KAUFMAN, L., AND HAMMARLING, S. J. Squeezing the most out of eigenvalue solvers on high-performance computers. *Linear Algebra Appl. 77* (1986), 113–136.
15. DU CROZ, J. J., AND HIGHAM, N. J. Stability of methods for matrix inversion. *IMA J. Numer. Anal. 12*, (1992), 1–19.
16. GALLIVAN, K., JALBY, W., MEIER, U., AND SAMEH, A. H. Impact of hierarchical memory systems on linear algebra algorithm design. *Int. J. Supercomput. Appl. 2* (1988), 12–48.
17. GALLIVAN, K. A., PLEMMONS, R. J., AND SAMEH, A. H. Parallel algorithms for dense linear algebra computations. *SIAM Rev. 32* (1990), 54–135.
18. GOLUB, G. H., AND VAN LOAN C. F. *Matrix Computations.* 2nd ed., Johns Hopkins University Press, Baltimore, 1989.
19. HIGHAM, N. J. Algorithm 694: A collection of test matrices in MATLAB. *ACM Trans. Math. Softw. 17*, (1991), 289–305.
20. HIGHAM, N. J. Bounding the error in Gaussian elimination for tridiagonal systems. *SIAM J. Matrix Anal. Appl. 11* (1990), 521–530.
21. HIGHAM, N. J. Exploiting fast matrix multiplication within the level 3 BLAS. *ACM Trans. Math. Softw. 16* (1990), 352–368.
22. HIGHAM, N. J. How accurate is Gaussian elimination? In Numerical Analysis 1989, In *Proceedings of the 13th Dundee Conference*, D. F. Griffiths and G. A. Watson, Eds., Pitman Research Notes in Mathematics 228, Longman, 1990, pp. 137–154.
23. HIGHAM, N. J. Is fast matrix multiplication of practical use? *SIAM News. 23* (Nov. 1990), 12 + .
24. HIGHAM, N. J. Iterative refinement enhances the stability of *QR* factorization methods for solving linear equations. *BIT 31*, (1991), 447–468.
25. HIGHAM, N. J. Stability of a method for multiplying complex matrices with three real matrix multiplications. Numerical Analysis Rep. No. 181, Univ. of Manchester, England, 1990; to appear in *SIAM J. Matrix Anal. Appl.*
26. MAYES, P., AND RADICATI, G. Banded Cholesky factorization using level 3 BLAS, LAPACK Working Note 12, Mathematics and Computer Science Div., Argonne National Lab. 1989.
27. OETTLI, W., AND PRAGER, W. Compatibility of approximate solution of linear equations with given error bounds for coefficients and right-hand sides. *Numer. Math. 6* (1964), 405–409.
28. PARLETT, B. N. *The Symmetric Eigenvalue Problem.* Prentice-Hall, Englewood Cliffs, N.J., 1980.
29. SCHREIBER, R. S. Block algorithms for parallel machines. In *Numerical Algorithms for Modern Parallel Computer Architectures*, M. H. Schultz, Ed., IMA Volumes In Mathematics and Its Applications 13, Springer-Verlag, Berlin, 1988, pp. 197–207.
30. SCHREIBER, R. S., AND VAN LOAN, C. F. A storage efficient WY representation for products of Householder transformations. *SIAM J. Sci. Stat. Comput. 10* (1989), 53–57.
31. SKEEL, R. D. Iterative refinement implies numerical stability for Gaussian elimination. *Math. Comput. 35* (1980), 817–832.
32. STRASSEN, V. Gaussian elimination is not optimal. *Numer. Math. 13* (1969), 354–356.
33. WILKINSON, J. H. Error analysis of eigenvalue techniques based on orthogonal transformations. *J. Soc. Indust. Appl. Math. 10* (1962), 162–195.
34. WILKINSON, J. H. *The Algebraic Eigenvalue Problem.* Oxford University Press, New York, 1965.
35. WILKINSON, J. H. Error analysis revisited. *IMA Bull. 22* (1986), 192–200.
36. WINOGRAD, S. A new algorithm for inner product. *IEEE Trans. Comput. C-18* (1968), 693–694.

Received August 1990; accepted August 1991