Exploiting Fast Matrix Multiplication Within the Level 3 BLAS

Nicholas J. Higham*

TR 89-984
April 1989

Department of Computer Science
Cornell University
Ithaca, NY 14853-7501

Exploiting Fast Matrix Multiplication Within the Level 3 BLAS

Nicholas J. Higham *

Department of Computer Science
Upson Hall
Cornell University
Ithaca, New York 14853

Abstract

The Level 3 BLAS (BLAS3) are a set of specifications of Fortran 77 subprograms for carrying out matrix multiplications and the solution of triangular systems with multiple right-hand sides. They are intended to provide efficient and portable building blocks for linear algebra algorithms on high performance computers. We describe algorithms for the BLAS3 operations that are asymptotically faster than the conventional ones. These algorithms are based on Strassen's method for fast matrix multiplication, which is now recognized to be a practically useful technique once matrix dimensions exceed about 100. We pay particular attention to the numerical stability of these "fast BLAS3". Error bounds are given and their significance is explained and illustrated with the aid of numerical experiments. Our conclusion is that the fast BLAS3, although not as strongly stable as conventional implementations, are stable enough to be suitable for many applications.

1980 Mathematics Subject Classification: Primary 65F05, 65G05.
Computing Reviews: G.1.3.
Key words: Level 3 BLAS, matrix multiplication, Strassen's algorithm, triangular systems, numerical stability, error analysis.

* On leave from the University of Manchester.

1. Introduction

In 1969 Strassen [21] showed how to multiply two $n \times n$ matrices with less than $4.7n^{\log_2 7}$ arithmetic operations. Since $\log_2 7 \approx 2.807 < 3$, his method improves asymptotically on the standard algorithm for matrix multiplication, which requires $O(n^3)$ operations.

Some have regarded Strassen's algorithm as being of theoretical interest only (see, for example, [18, p.76], [20, p.533]). However, in 1970 Brent [5] implemented Strassen's algorithm in Algol-W on an IBM 360/67 and concluded that in this environment Strassen's method (with just one level of recursion) runs faster than the conventional method for $n \geq 110$. Furthermore, recently, Bailey [2] compared his Fortran implementation of Strassen's algorithm for the Cray-2 with the Cray library routine for matrix multiplication and observed speedup factors ranging from 1.45 for $n = 128$ to 2.01 for $n = 2048$ (although 35% of these speedups are due to Cray specific techniques). These empirical results of Brent and Bailey show that Strassen's algorithm is indeed of practical interest when n is in the hundreds.

The exponent for matrix multiplication has been reduced several times to the current record value of 2.376 [7], but as far as we know none of these asymptotically faster algorithms is quicker than Strassen's method for values of n for which dense matrix multiplication is currently performed in practice ($n \leq 10,000$, say).

In this work we show how Strassen's algorithm can be exploited in all the Level 3 Basic Linear Algebra Subprograms (BLAS3). The BLAS3 [9, 10] are a set of specifications of Fortran 77 subprograms for carrying out matrix-matrix operations. They are intended to provide efficient and portable building blocks for linear algebra algorithms on high performance computers. One reason for the importance of the BLAS3 is that organizing algorithms in a block structure (treating matrices as arrays of smaller matrices) and using calls to the high level BLAS3 primitives is an effective way to achieve high performance on machines with a hierarchy of memory (such as cache memory, global memory, or vector registers); see, for example, [8, 11, 12, 19].

In section 2 we present Strassen's algorithm, in its most general form for evaluating products of rectangular matrices, and we discuss practical issues concerning its implementation. In section 3 we summarize the BLAS3 primitives and describe fast algorithms for the BLAS3 operations involving symmetry and triangularity; these algorithms are recursive and make use of Strassen's method. In addition to having asymptotically smaller operation counts than conventional BLAS3 implementations the ones we propose have much scope for parallelization, by virtue of their divide and conquer nature.

As explained in [9, p.12] "Although it is intended that the Level 3 BLAS be implemented as efficiently as possible, it is essential that efficiency should not be achieved at the cost of sacrificing numerical stability." We therefore pay particular attention to the stability properties of the algorithms discussed here. Rounding error bounds are given and analysed in section 4, and section 5 contains experiments designed to give further insight into the stability properties of these "fast BLAS3". Our conclusion is that while fast BLAS3 are not as strongly numerically stable as conventional implementations of the BLAS3, they are stable enough to be of use in many applications.

2. Fast Multiplication of Rectangular Matrices

Strassen's method is usually presented as a way to multiply square matrices. This is true of the original paper [21] as well as most subsequent descriptions. An exception is the unpublished report of Brent [5] which treats the rectangular case, and which we follow here.

To develop the general version of Strassen's method consider the product $C = AB$, where A and B are matrices of dimensions $m \times n$ and $n \times p$ respectively. Assume, for the moment, that

$$m = 2^i, \qquad n = 2^j, \qquad p = 2^k.$$

Partitioning each of C, A and B into four equally sized blocks, the product

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} \tag{2.1}$$

can be accomplished using the following formulae:

$$\begin{aligned}
P_1 &= (A_{11} + A_{22})(B_{11} + B_{22}), \\
P_2 &= (A_{21} + A_{22})B_{11}, \\
P_3 &= A_{11}(B_{12} - B_{22}), \\
P_4 &= A_{22}(B_{21} - B_{11}), \\
P_5 &= (A_{11} + A_{12})B_{22}, \\
P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}), \\
P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}), \\
\\
C_{11} &= P_1 + P_4 - P_5 + P_7, \\
C_{12} &= P_3 + P_5, \\
C_{21} &= P_2 + P_4, \\
C_{22} &= P_1 + P_3 - P_2 + P_6.
\end{aligned} \tag{2.2}$$

These equations are easily confirmed by substitution. Counting the additions (A) and multiplications (M) we find that while conventional multiplication requires

$$mnp\mathrm{M} + m(n-1)p\mathrm{A}, \tag{2.3}$$

Strassen's algorithm, using conventional multiplication at the block level, requires

$$\tfrac{7}{8}mnp\mathrm{M} + \left(\tfrac{7}{8}m(n-2)p + \tfrac{5}{4}mn + \tfrac{5}{4}np + \tfrac{8}{4}mp\right)\mathrm{A}. \tag{2.4}$$

Thus if m, n and p are large Strassen's algorithm reduces the arithmetic by a factor of about $\frac{7}{8}$. Since all the blocks in (2.1) have even dimensions we can use the idea recursively on the multiplications associated with the P_i. A total of $q = \min(i,j,k)$ recursions are possible, after which we have matrix multiplications of size $(2^{i-q} \times 2^{j-q}) \times (2^{j-q} \times 2^{k-q})$, in which one of the dimensions is 1 and conventional multiplication must be used. Overall, the number of scalar multiplications is

$$7^q 2^{i+j+k-3q} = \left(\tfrac{7}{8}\right)^q 2^{i+j+k} = \min(m,n,p)^{\log_2 \frac{7}{8}} mnp. \tag{2.5}$$

In the case $m = n = p$ this reduces to $n^{\log_2 7} = n^{2.807\ldots}$. There does not seem to be any simple formula for the number of additions in the general case, but it is of the same order as the number of multiplications.

There are several ways of modifying the algorithm to handle odd dimensions. One technique is to pad A and B with zeros to achieve even dimensions, compute the extended product recursively, and then extract the desired product by "unpadding". Two ways to pad are as follows. One can extend any odd dimension to the next even one by padding with a single row and/or column of zeros; in this case padding may have to be done on each step of the recursion (e.g. if $m = n = p = 2^r + 1$ for some r). Or, as suggested in [21], one can pad once and for all to make each dimension a power of 2 times t, for some small t, and then recur until some dimension is t, at which point conventional multiplication is used. Finally there is the "chopping" approach suggested in [5]: here one temporarily drops last rows and/or columns to achieve even dimensions, computes the product, and then reinstates the lost information via a rank 1, 2 or 3 correction. For example, if m, n and p are all odd, we can write

$$AB = \begin{bmatrix} A_1 & a_c \\ a_r^T & \alpha \end{bmatrix} \begin{bmatrix} B_1 & b_c \\ b_r^T & \beta \end{bmatrix} = \begin{bmatrix} A_1 B_1 & 0 \\ 0 & 0 \end{bmatrix} + \Delta,$$

where

$$\Delta = \begin{bmatrix} a_c \\ \alpha \end{bmatrix} [\, b_r^T \quad \beta \,] + \begin{bmatrix} 0 & A_1 b_c \\ a_r^T B_1 & a_r^T b_c \end{bmatrix}.$$

This last approach is the most attractive for practical computation, because it has the lowest storage requirement and is the easiest to implement in a language such as Fortran that does not allow dynamic expansion of arrays. (We mention that in the Matlab language [17], which supports arbitrary re-dimensioning of arrays, the padding approach is trivial to implement.)

In practice it is expedient not to recur to the level of scalars or vectors, but to use conventional multiplication once the dimensions are so small that any further reduction in the number of arithmetic operations is offset by an increase in bookkeeping costs. For insight into how the cutoff level should be determined it is helpful to compare the number of operations in Strassen's algorithm with one level of recursion with that for conventional multiplication. Assuming the dimensions are even, we have, on subtracting (2.4) from (2.3),

$$\tfrac{1}{8}mnp\mathrm{M} + \left(\tfrac{1}{8}mnp - \tfrac{5}{4}mn - \tfrac{5}{4}np - \tfrac{5}{4}mp\right)\mathrm{A}.$$

If we assume "M $\approx$ A" this reduces to

$$\tfrac{1}{4}\left(mnp - 5(mn + np + mp)\right)\mathrm{M},$$

from which we can conclude that Strassen's method with one level of recursion requires less arithmetic than conventional multiplication if $mnp \geq 5(mn + np + mp)$. This suggests that a criterion of the form "if $mnp \leq n_0(mn + np + mp)/3$" is appropriate to terminate the recursions, where n_0 is a machine and compiler dependent value that must be chosen empirically (we divide by 3 to make the test reduce to "if $n \leq n_0$" when $m = n = p$). In his Fortran implementation for square matrices on the Cray-2 Bailey [2] found that $n_0 = 127$ minimized the execution time.

It is interesting to ask what value of n_0 minimizes the number of arithmetic operations. We can answer this question in the case $m = n = p = 2^k$. Let $n_0 = 2^r$. The number of multiplications and additions can be shown to be

$$M(k) = 7^{k-r}8^r, \qquad A(k) = 4^r(2^r + 5)7^{k-r} - 6 \cdot 4^k.$$

The sum $M(k) + A(k)$ is minimized over all integers r by $r = 3$; interestingly, this value is independent of k. Thus for all $n = 2^k \geq 8$, the choice $n_0 = 8$ minimizes a reasonable measure of the computational work. For $n_0 = 8$ there are about 1.6 times more additions than multiplications when n is large; for $n_0 = 1$ this ratio increases to around 6. Moreover, the total number of arithmetic operations for $n_0 = 1$ is about 1.8 times that for $n_0 = 8$ when n is large; this emphasizes that the choice of n_0 can have a significant impact on the efficiency of Strassen's method. Finally, we note the interesting statistics that the total number of arithmetic operations for Strassen's method with $n_0 = 8$ is smaller than that for conventional multiplication by a factor 0.76 for $n = 128$ and 0.52 for $n = 1024$.

3. Fast Algorithms for the BLAS3

The BLAS3 cover four basic matrix-matrix operations.

(a) Matrix-matrix products:

$$C \leftarrow \alpha AB + \beta C, \quad A \in \mathbf{R}^{m\times n},\, B \in \mathbf{R}^{n\times p},\, C \in \mathbf{R}^{m\times p}.$$

(b) Rank-r and rank-$2r$ updates of a symmetric matrix $C \in \mathbf{R}^{n\times n}$:

$$C \leftarrow \alpha A^T A + \beta C, \quad A \in \mathbf{R}^{r\times n},$$
$$C \leftarrow \alpha A^T B + \alpha B^T A + \beta C, \quad A, B \in \mathbf{R}^{r\times n}.$$

(c) Multiplication of a matrix by a triangular matrix:

$$B \leftarrow \alpha TB, \quad T \in \mathbf{R}^{m\times m} \text{ triangular}, \quad B \in \mathbf{R}^{m\times p}.$$

(d) Solving a triangular system of equations with multiple right-hand sides:

$$B \leftarrow \alpha T^{-1}B, \quad T \in \mathbf{R}^{m\times m} \text{ nonsingular and triangular}, \quad B \in \mathbf{R}^{m\times p}.$$

This is a simplified description. The BLAS3 include variations such as $C \leftarrow \alpha A^T B + \beta C$ in (a), and $B \leftarrow \alpha BT^{-T}$ in (c). For our purposes it is sufficient to give algorithms for (a)–(d), as the variations are handled with obvious modifications.

The general matrix product in (a) can be implemented using Strassen's method as described in section 2. (Note that it may take less operations to compute $(\alpha A)B$ or $A(\alpha B)$ rather than $\alpha(AB)$, depending on the dimensions of A and B.)

In the rest of this section we develop recursive algorithms for operations (b)–(d). For ease of presentation we assume that all matrix dimensions are a power of 2. For general dimensions one can use the chopping technique discussed in section 2, taking advantage of structure when forming the low rank correction matrix. When stating operation counts we assume full recursion (i.e. until some dimension is 1). In practice one would have a cutoff threshold: once the problem size is sufficiently small conventional multiplication, or in the case of (d), the substitution algorithm for triangular systems would be used.

In (b) the rank-r update requires the computation of $\mathrm{CP}(A) := A^T A$, where $A \in \mathbf{R}^{r\times n}$ ("CP" stands for cross product). Partitioning A into four blocks of dimensions $r/2 \times n/2$ we have

$$A^T A = \begin{bmatrix} A_{11}^T & A_{21}^T \\ A_{12}^T & A_{22}^T \end{bmatrix} \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} A_{11}^T A_{11} + A_{21}^T A_{21} & A_{11}^T A_{12} + A_{21}^T A_{22} \\ \text{symm.} & A_{12}^T A_{12} + A_{22}^T A_{22} \end{bmatrix}.$$

A recursive algorithm to compute $\mathrm{CP}(A)$ is as follows:

$$\mathrm{CP}(A) = \begin{bmatrix} \mathrm{CP}(A_{11}) + \mathrm{CP}(A_{21}) & \mathtt{Strass}(A_{11}^T, A_{12}) + \mathtt{Strass}(A_{21}^T, A_{22}) \\ \text{symm.} & \mathrm{CP}(A_{12}) + \mathrm{CP}(A_{22}) \end{bmatrix}, \tag{3.1}$$

where $\mathtt{Strass}(A, B)$ denotes the use of Strassen's method to compute AB. Thus, the idea is to apply the algorithm recursively to the "$A^T A$" sub-products and use Strassen's method on the others.

The number of multiplications required by this algorithm is

$$rn^2\left(\tfrac{2}{3}\mu^{\log_2 7/8} - \tfrac{1}{6}\mu^{-1}\right) + \tfrac{1}{2}rn, \qquad \mu = \min(r, n), \tag{3.2}$$

which reduces to $\frac{2}{3}n^{\log_2 7} + \frac{1}{3}n^2$ when $n = r$. In comparison with the count (2.5) for Strassen's method, the dominant term in (3.2) has the same exponent but a smaller constant: $\frac{2}{3}$ instead of 1. Note that this improvement is not as good as for conventional multiplication, where symmetry of the product halves the work. We mention that S. Vavasis (private communication) has devised a "$\frac{3}{5}n^{\log_2 7}$" method. His method employs three calls to Strassen's method and two recursive calls to the method itself. Thus, compared to (3.1) it involves 5 rather than 6 recursive calls on each level, but the formulae defining the method are more complicated and so it uses more additions and transpositions on each level.

The rank-$2r$ update in (b) can be handled by computing $D = A^T B$ using Strassen's algorithm and then forming $C \leftarrow \alpha D + \alpha D^T + \beta C$.

Next we turn to the multiplication by a triangular matrix in (c), $B \leftarrow TB$. We can use a technique analogous to the one just discussed for computing $\mathrm{CP}(A)$. Assuming T is upper triangular we can partition T and B into four equally sized blocks and write

$$A = \begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} T_{11}B_{11} + T_{12}B_{21} & T_{11}B_{12} + T_{12}B_{22} \\ T_{22}B_{21} & T_{22}B_{22} \end{bmatrix}.$$

The idea is to use Strassen's method on the full matrix products $T_{12}B_{21}$ and $T_{12}B_{22}$, and the same algorithm recursively on the other products. The operation count is essentially the same as for the cross product algorithm above; just replace n by m and r by p in (3.2).

Finally, we consider the BLAS3 operation (d). Our task is to compute $T^{-1}B$, where $T \in \mathbf{R}^{m\times m}$ is triangular and $B \in \mathbf{R}^{m\times p}$. We partition T and B into four equally sized blocks to obtain

$$\begin{aligned}\begin{bmatrix} T_{11} & T_{12} \\ 0 & T_{22}\end{bmatrix}^{-1}\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22}\end{bmatrix} &= \begin{bmatrix} T_{11}^{-1} & -T_{11}^{-1}T_{12}T_{22}^{-1} \\ 0 & T_{22}^{-1}\end{bmatrix}\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22}\end{bmatrix} \\ &= \begin{bmatrix} T_{11}^{-1}(B_{11} - T_{12}(T_{22}^{-1}B_{21})) & T_{11}^{-1}(B_{12} - T_{12}(T_{22}^{-1}B_{22})) \\ T_{22}^{-1}B_{21} & T_{22}^{-1}B_{22}\end{bmatrix} \\ &\equiv \begin{bmatrix} P_4 & P_3 \\ P_2 & P_1\end{bmatrix},\end{aligned}$$

where the P_i are computed by recursive applications of the same algorithm, and the products $T_{12}P_1$ and $T_{12}P_2$ (involved in the computation of P_3 and P_4 respectively) are computed using Strassen's algorithm. In the case $B = I$ this method reduces to the matrix inversion technique described in [21]. The operation count is exactly the same as for the method described for (b).

4. Numerical Stability

Although Strassen's method is well known, its numerical stability, that is, its behavior in floating point arithmetic, is much less widely appreciated. Partly this is because the early error analysis of the method in [5] was not published (Brent's paper [6] contains some material from [5], but not the error analysis of Strassen's method). Miller [16] states a stability result for Strassen's method in general terms. His result is presented in a more specific form by Bailey in [2], though unfortunately an error in the statement makes the result too strong. Bini and Lotti [3] give an error analysis of a class of fast matrix multiplication techniques which includes Strassen's; when specialized to Strassen's method their quite general error bound is similar to the result given below, but it has an extra factor $\log_2 n$.

It is not difficult to do an error analysis of Strassen's method, at least in the case $m = n = p = 2^k$. We did an analysis before seeing [5] and arrived at almost exactly the same result by the same route. The technique is to guess the form of the error bound (e.g. by looking at the 2×2 case) and then to prove the guess correct inductively, at the same time determining the unknown constant term.

The result may be stated as follows. Let $n_0 = 2^r$ be the threshold beyond which conventional multiplication is used. If u denotes the unit roundoff and $\widehat{C}$ denotes the computed product $C = AB$ from Strassen's method, then under the usual model of floating point arithmetic (see, e.g., [12])

$$\|\widehat{C} - C\| \le \left(\frac{n}{n_0}\right)^{\log_2 12} (n_0^2 + 5)u\|A\|\|B\| + O(u^2), \tag{4.1}$$

where $\|A\| = \max_{i,j} |a_{ij}|$ (note that this is not a consistent matrix norm since $\|AB\| \le \|A\|\|B\|$ is generally false). For comparison, if $C = AB$ is computed the usual way then

$$|\widehat{C} - C| \le nu|A||B| + O(u^2), \tag{4.2}$$

where $|\cdot|$ denotes the operation of replacing each matrix element by its absolute value, and (4.2) implies the (generally much weaker) bound

$$\|\widehat{C} - C\| \le n^2 u\|A\|\|B\| + O(u^2). \tag{4.3}$$

To interpret the above bounds note first that all three fall short of the ideal bound

$$|\widehat{C} - C| \le u|C| + O(u^2), \tag{4.4}$$

which says that each component of C is computed with high relative accuracy. Nevertheless (4.2) is a strong bound—the best we can expect when we accept the possibility of numerical cancellation. It treats each element of the error matrix $E = \widehat{C} - C$ individually, and is similar to (4.4) if $|A||B| \approx |C|$.

The norm bounds (4.1) and (4.3) are weaker than (4.2), since they provide the same bound for each element of E. Note also that the scaling $AB \to (AD)(D^{-1}B)$, where D is diagonal, leaves (4.2) unchanged but alters (4.1) and (4.3).

The bounds (4.3) and (4.1) differ only in the constant term. For Strassen's method the greater the depth of recursion the bigger the constant in (4.1): if we use just one level of recursion ($n_0 = n/2$) then the constant is $3n^2 + 60$, whereas with full recursion ($n_0 = 1$) the constant is $6n^{\log_2 12} = 6n^{3.585\ldots}$.

To summarise, Strassen's method has less favorable stability properties than conventional multiplication in two respects: it satisfies a weaker error bound (normwise rather than componentwise) and it has a larger constant in the bound (how much larger depending on n_0). The normwise bound is a consequence of the fact that Strassen's method adds together elements of A matrix-wide (and similarly for B); for example in (2.2) A_{11} is added to A_{22}, A_{12} and A_{21}. This intermingling of elements is particularly undesirable when A or B has elements of widely differing magnitudes because then large errors can contaminate small components of the product. This phenomenon is well illustrated by the example

$$C = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \epsilon \\ \epsilon & \epsilon^2 \end{bmatrix},$$

which is evaluated exactly in floating point arithmetic if we use conventional multiplication. However Strassen's method computes

$$c_{22} = 2(1 + \epsilon^2) + (\epsilon - \epsilon^2) - 1 - (1 + \epsilon).$$

Because c_{22} involves sub-terms of order unity the error $\widehat{c}_{22} - c_{22}$ will be of order u. Thus the relative error $|\widehat{c}_{22} - c_{22}|/|c_{22}| = O(u/\epsilon^2)$, which is much larger than u if ϵ is small. This is an example where Strassen's method does not satisfy the bound (4.2).

Another interesting property of Strassen's method is that it always involves some genuine subtractions (assuming that all additions are of nonzero terms). This is easily deduced from the formulae (2.2). As noted in [12], this makes Strassen's method unattractive in applications where all the elements of A and B are nonnegative (for example, in Markov processes [13]). Here, conventional multiplication yields low relative error componentwise because in (4.2) $|A||B| = |AB| = |C|$, yet comparable accuracy cannot be guaranteed for Strassen's method.

We mention that Winograd discovered a variant of Strassen's method that requires only 15 additions instead of 18 (see [1, p.247, 4, p.133]). However, for this variant the error bound corresponding to (4.1) contains a larger exponent: $\log_2 18 \approx 4.170$ in place of $\log_2 12 \approx 3.585$. This result is proved in [3] (in which both error bounds have an extra factor $\log_2 n$), where it is also shown that Strassen's method has the minimum exponent in its error bound over the set of all fast matrix multiplication methods that are based on computation of a 2×2 matrix product in 7 multiplications and that employ integer constants of the form $\pm 2^i$, where i is an integer (this set breaks into 26 equivalence classes).

Finally, we comment on the stability properties of the methods of section 3. For square matrices of dimension n the recursive methods for the symmetric matrix update and the triangular matrix times matrix product both satisfy the same bound (4.1) as Strassen's method; this is not surprising since in both methods nearly all the work is done by calls to Strassen's method. The method for solving triangular systems $TX = B$ satisfies the following bound (under the same assumptions as for (4.1)):

$$(4.5) \qquad T\widehat{X} = B + E, \qquad \|E\| \le \left[\left(\frac{n}{n_0} \right)^{\log_2 12} (n_0^2 + 1) + n_0^2 \right] u \|T\| \|\widehat{X}\| + O(u^2).$$

For comparison consider the computed solution $\overline{X}$ obtained using back substitutions. The ith column $\overline{x}_i$ of $\overline{X}$ satisfies (see, for example, [14])

$$(T + E_i)\overline{x}_i = b_i, \qquad |E_i| \le (n+1)u|T|.$$

It follows that

$$TX̄ = B + F, \qquad |F| \le (n+1)u|T||\overline{X}|, \tag{4.6}$$

and the latter bound implies $\|F\| \le n(n+1)u\|T\|\|\overline{X}\|$. Thus the same comments apply as for Strassen's method: the error bound for the fast $TX = B$ solver has a weaker form than that for the conventional technique and has a larger constant.

5. Numerical Experiments

We have carried out numerical experiments to gain insight into the bounds of section 4 and to explore the effect of using the fast matrix multiplication techniques in place of conventional multiplication within one particular algorithm.

All computations were performed in Matlab [17] on a Sun 3/50 workstation. The (double precision) unit roundoff $u_d = 2^{-52} \approx 2.2 \times 10^{-16}$. Our Matlab codes use recursion and are quite short (under 50 lines of code for each fast BLAS3 routine). Because of the overhead of interpretation and recursion in Matlab, these "fast" routines are in fact quite slow, and so we do not report timings.

In the first experiment we looked at the error in Strassen's algorithm. Let $*$ denote conventional multiplication. For several A and B we computed $C = A * B$ in double precision, and then $C_* = A*B$ and $C_S = \mathtt{Strass}(A, B, n_0)$ in simulated single precision ($u_s = 2^{-23} \approx 1.2 \times 10^{-7}$). We define the following quantities:

$$\rho_N(\widehat{C}) = \frac{\|\widehat{C} - C\|}{n^2 u_s \|A\| \|B\|} \quad \text{(normwise relative residual)},$$

$$\rho_C(\widehat{C}) = \max_{i,j} \left\{ \frac{|\widehat{c}_{ij} - c_{ij}|}{n u_s \big(|A||B|\big)_{ij}} \right\} \quad \text{(componentwise relative residual)},$$

$$e_N(\widehat{C}) = \frac{\|\widehat{C} - C\|}{u_s \|C\|} \quad \text{(normwise relative error)},$$

$$e_C(\widehat{C}) = \max_{i,j} \left\{ \frac{|\widehat{c}_{ij} - c_{ij}|}{u_s |c_{ij}|} \right\} \quad \text{(componentwise relative error)}.$$

The quantities $\rho_N(C_*) \le 1$ and $\rho_C(C_*) \le 1$ (to within $O(u_s)$) measure the sharpness of the bounds (4.3) and (4.2) respectively, while $\rho_N(C_S)$ and $\rho_C(C_S)$ indicate whether the computed Strassen product C_S satisfies these same bounds. The quantity

$$\rho_S(C_S) = \frac{\|C_S - C\|}{d_n u_s \|A\| \|B\|} \le 1, \qquad d_n = \left(\frac{n}{n_0}\right)^{\log_2 12} (n_0^2 + 5),$$

measures the sharpness of the residual bound (4.1) for Strassen's method.

We present results for square A and B of dimension $n = 64$ in Table 5.1. We tried both $n_0 = 32$ and $n_0 = 4$, to see how the depth of recursion affects the errors.

In practical use of Strassen's method n and n_0 would be somewhat larger than these values, but they are sufficient to give insight into the error behavior.

The matrices used are defined as follows: urand_i and nrand_i are random matrices with elements from the uniform $[0,1]$ and normal $(0,1)$ distributions, respectively. Z_i is a random matrix with 2-norm condition number 10^4. P is the Pascal matrix, made up from the numbers in Pascal's triangle; its (i,j) element is $(i+j-2)!/\left[(i-1)!(j-1)!\right]$. In each case the same A and B were used for each of the two n_0 values. The C_* statistics are listed in the columns $n_0 = 32$ although they are independent of n_0.

Table 5.1 Results for Strassen's method; $m = n = 64$.

(A,B):	$(\text{urand}_1,$	$\text{urand}_2)$	$(\text{nrand}_1,$	$\text{nrand}_2)$	$(Z_1,$	$Z_2)$	$(P,$	$\text{urand}_3)$
n_0:	32	4	32	4	32	4	32	4
$\rho_N(C_S)$	3.23e-2	8.17e-2	8.39e-3	2.94e-2	5.08e-3	1.83e-2	1.19e-2	5.14e-2
$\rho_C(C_S)$	2.00e-1	1.12e0	7.85e-2	2.81e-1	9.00e-2	7.08e-1	4.11e22	1.75e34
$e_N(C_S)$	2.51e0	6.33e0	4.06e0	1.42e1	3.87e0	1.39e1	1.00e0	4.35e0
$e_C(C_S)$	1.28e1	7.18e1	8.50e4	4.34e4	1.65e4	1.63e5	2.63e24	1.12e36
$\rho_S(C_S)$	1.64e-2	3.01e-3	8.97e-4	8.22e-5	9.50e-4	9.36e-5	2.85e-4	2.38e-4
$\rho_N(C_*)$	2.55e-2		7.01e-3		4.31e-3		6.89e-3	
$\rho_C(C_*)$	1.18e-1		5.35e-2		5.22e-2		9.37e-2	
$e_N(C_*)$	1.98e0		3.39e0		3.28e0		5.83e-1	
$e_C(C_*)$	7.57e0		3.27e4		7.98e3		6.00e0	

The results display several interesting features:

- The results confirm the error bounds (4.1–4.3), since $\rho_S(C_S) \le 1$, $\rho_N(C_*) \le 1$ and $\rho_C(C_*) \le 1$ in all cases. The bounds are one or more orders of magnitude from being equalities, and the bound (4.1) for Strassen's method is particularly weak.
- $\rho_N(C_S) \le 1$ in each case, that is, in these examples Strassen's method satisfies the norm bound (4.3) for conventional multiplication. The componentwise bound (4.2) is severely violated by Strassen's method in the Pascal matrix example! In this example A and B are nonnegative and A has elements of widely varying magnitude—both these

properties are unfavorable for Strassen's method, as noted in section 4.

• The $e_N(C_S)$ and $e_N(C_*)$ values show that the two multiplication techniques gave products with similar normwise relative errors. The componentwise relative errors were also similar for the first three products, being large for the second and third products in accord with the fact that neither method satisfies the bound (4.4).

• The error measures for Strassen's method are in most cases bigger for $n_0 = 4$ than for $n_0 = 32$, as the bound (4.1) "predicts".

Next, we consider the fast triangular system solver of section 3. We chose $n = p = 64$ and solved four systems in double precision, first with the fast algorithm, obtaining X_S, and then with back substitutions, obtaining X_B. We computed the relative residuals

$$\rho_N(\widehat{X}) = \frac{\|T\widehat{X} - B\|_\infty}{(n+1)u_d\|T\|_\infty\|\widehat{X}\|_\infty}, \qquad \rho_C(X) = \max_{i,j}\left\{\frac{(|T\widehat{X} - B|)_{ij}}{(n+1)u_d(|T||\widehat{X}|)_{ij}}\right\},$$

which are bounded by 1 for $\widehat{X} = X_B$, in view of (4.6). A quantity $\rho_S(X_S)$ measures the weakness in the bound (4.5) and is defined analogously to $\rho_S(C_S)$ above. In each case B was of the type nrand described above. T_1, T_2, T_3 and T_4 are the triangular factors from the QR factorizations of random matrices with 2-norm condition numbers 10, 10^5, 10^{10} and 10^{15} respectively. The results are given in Table 5.2.

Table 5.2 Results for the fast $TX = B$ solver; $m = p = 64$.

T:	T_1		T_2		T_3		T_4	
$\kappa_2(T)$:	10		10^5		10^{10}		10^{15}	
n_0:	32	4	32	4	32	4	32	4
$\rho_N(X_S)$	1.62e-3	4.39e-3	5.19e-4	2.42e-3	5.21e-4	2.28e-3	5.37e4	2.00e-3
$\rho_C(X_S)$	2.20e-2	6.38e-2	1.99e-2	1.72e0	1.52e-2	2.57e2	1.47e-2	5.80e4
$\rho_S(X_S)$	1.03e-4	1.04e-5	8.07e-5	1.51e-5	6.66e-5	1.94e-5	9.67e-5	1.42e-5
$\rho_N(X_B)$	2.12e-3		9.76e-4		8.73e-4		7.62e-4	
$\rho_C(X_B)$	2.20e-2		1.99e-2		1.52e-2		1.47e-2	

The key features of the results are:

• In these tests X_S satisfies the normwise residual bound satisfied by X_B, but does not always satisfy the componentwise bound (4.6).

• Generally, the various bounds are far from being equalities, particularly the bound (4.5) for X_S.

We summarise a third experiment in which we used the fast BLAS3 within the matrix multiplication rich polar decomposition algorithm of [15]. This algorithm employs the iteration

$$X_{k+1} = X_k + \tfrac{1}{2} X_k (I - X_k^T X_k), \qquad X_k \in \mathbf{R}^{n\times n},$$

which we implemented as

$$X_{k+1} = X_k + \tfrac{1}{2}\texttt{Strass}(X_k, I - \mathrm{CP}(X_k)).$$

We computed the polar decompositions of various A of dimensions 32 and 64, with $n_0 = 4, 8$ or 16. A natural measure of the quality of the computed polar factors is their backward error. In all cases where the iteration converged the backward error was of the same order of magnitude as when conventional multiplication was used, namely, as small as could be expected. In several cases, with $n_0 = 4$ or 8, the iteration failed to converge, although it had converged for the same matrices when using conventional multiplication. This behavior can be explained by the less accurate answers returned by the fast BLAS3. Relaxing the convergence tolerance slightly restored convergence and still gave an acceptable backward error in all cases.

Taking into account the error analysis of section 3 and the experiments of this section the numerical stability properties of the fast BLAS3 may be summarised as follows. Here, by "conventional BLAS3" we mean the BLAS3 implemented using conventional multiplication and the substitution algorithm. The conventional BLAS3 satisfy strong componentwise bounds for the residuals of the computed matrices. The fast BLAS3 do not satisfy any such componentwise bounds, as simple examples show, although in specific cases componentwise small residuals may be obtained (e.g. in most of our numerical tests). The two BLAS3 implementations satisfy similar *normwise* bounds, but the fast BLAS3 have larger constant terms, which increase as the cutoff threshold n_0

decreases. In our tests with $n \leq 64$, the normwise residuals for the fast BLAS3 were never more than twice as large as those for the conventional BLAS3.

We feel that for many applications the fast BLAS3 have adequate stability properties. An important consideration is whether, when the BLAS3 are employed as building blocks in an algorithm, it is crucial that componentwise small residuals be achieved. From examination of the error analysis this appears not to be an issue in most block algorithms, such as block Gaussian elimination. It is also important to appreciate the potentially faster growth of errors with n than for the conventional BLAS3. This faster growth may necessitate some minor algorithm re-tuning, as in our polar decomposition example, and it may reduce the achievable accuracy. However, in practice one is unlikely to have more than a few levels of recursion for reasons of efficiency, so the additional error growth may not be too serious.

To conclude, we point out that while we have implemented and tested all the algorithms described here in Matlab on a workstation, we have not implemented the algorithms in a compiled language on a high performance computer. This task we leave to further work, but Bailey's results [2] for Strassen's method assure us that the fast BLAS3 will yield useful speedups for n in the hundreds on appropriate machines.

Acknowledgements

I am grateful to Richard Manuck of the Stanford Mathematics and Computer Science Library for providing me with a copy of [5]. Part of this work was done during a visit to the Computer Science Department at Stanford University in January 1989. I thank Gene Golub, Walter Murray and Michael Saunders for financial support and for making my stay so pleasant. Des Higham corrected an error in (2.4) in a draft manuscript and made many helpful suggestions. Steve Vavasis also offered helpful comments.

Appendix. Matlab M-files.

```
function C = strass(A, B, nmin, pad)
%STRASS   Strassen's fast matrix multiplication algorithm for
%         rectangular matrices.
%         C = STRASS(A, B, NMIN, PAD), where A and B are arbitrary
%         matrices, computes the product C=A*B.
%         NMIN >= 1 is used in a test that determines at what level of
%         recursion to use conventional multiplication.
%         PAD = 1 to deal with odd dimensions by padding with zeros, else
%         PAD = 0 to chop to even dimensions, with a correction step.
%         Defaults: NMIN = 4, PAD = 0.

if nargin < 4, pad = 0; end
if nargin < 3, nmin = 4; end
[m,n] = size(A); [n,p] = size(B);
if m*n*p <= nmin*(m*n+n*p+m*p)/3 | min([m n p]) == 1, C = A*B; return, end
meven = ~rem(m,2); neven = ~rem(n,2); peven = ~rem(p,2);
mold = m; pold = p;

if pad
   % Pad with zeros as necessary to make dimensions even.
   if ~meven, A = [A; zeros(1,n)]; m = m+1; end
   if ~neven, A = [A zeros(m,1)]; B = [B; zeros(1,p)]; n = n+1; end
   if ~peven, B = [B zeros(n,1)]; p = p+1; end
else
   % Chop: reduce dimensions to make them even, and correct later.
   if ~meven, m = m-1; end
   if ~neven, n = n-1; end
   if ~peven, p = p-1; end
end

m2 = m/2; n2 = n/2; p2 = p/2;
im = 1:m2; in = 1:n2; ip = 1:p2;
jm = m2+1:m; jn = n2+1:n; jp = p2+1:p;
P1 = strass( A(im,in)+A(jm,jn), B(in,ip)+B(jn,jp), nmin, pad);
P2 = strass( A(jm,in)+A(jm,jn), B(in,ip), nmin, pad);
P3 = strass( A(im,in), B(in,jp)-B(jn,jp), nmin, pad);
P4 = strass( A(jm,jn), B(jn,ip)-B(in,ip), nmin, pad);
P5 = strass( A(im,in)+A(im,jn), B(jn,jp), nmin, pad);
P6 = strass( A(jm,in)-A(im,in), B(in,ip)+B(in,jp), nmin, pad);
P7 = strass( A(im,jn)-A(jm,jn), B(jn,ip)+B(jn,jp), nmin, pad);
C = [ P1+P4-P5+P7  P3+P5;  P2+P4  P1+P3-P2+P6 ];

if pad
   C = C(1:mold, 1:pold);  % Undo zero padding
else
   % Undo chopping: correct for odd dimensions
   if ~meven
       C(m+1,1:p) = A(m+1,1:n)*B(1:n,1:p);
   end
   if ~peven
       C(:,p+1) = A(:,1:n)*B(1:n,p+1);
   end
   if ~neven
      C = C + A(:,n+1)*B(n+1,:);
   end
end
```

```
function C = cprod(A, nmin)
%CPROD    Strassen-based algorithm for forming cross product matrix C=A'*A.
%         Usage: C = CPROD(A, NMIN), where A is an arbitrary matrix.
%         NMIN >= 1 is used in a test that determines at what level to
%         terminate the recursions.  Default: NMIN = 4.

if nargin < 2, nmin = 4; end
[m,n] = size(A);

if m <= nmin | n == 1, C = A'*A; return, end

C = zeros(n);
meven = ~rem(m,2); neven = ~rem(n,2);

% Chop: reduce dimensions to make them even, and correct later.
if ~meven, m = m-1; end
if ~neven, n = n-1; end

m2 = m/2; n2 = n/2;
im = 1:m2; in = 1:n2;
jm = m2+1:m; jn = n2+1:n;

C(in, in) = cprod(A(im,in), nmin) + cprod(A(jm,in), nmin);
C(in, jn) = strass(A(im,in)', A(im,jn), nmin) + ...
            strass(A(jm,in)', A(jm,jn), nmin);
C(jn, in) = C(in, jn)';
C(jn, jn) = cprod(A(im, jn), nmin) + cprod(A(jm, jn), nmin);

% Undo chopping: corrections for odd dimensions
if ~neven
    temp = A(1:m,:)'*A(1:m,n+1);
    C(:,n+1) = temp;
    C(n+1,1:n) = temp(1:n)';   % Using symmetry.
end

if ~meven
   C = C + A(m+1,:)'*A(m+1,:); % Should exploit symmetry here too.
end
```

```
function C = triprod(T, B, nmin)
%TRIPROD   Strassen-based algorithm for forming product C=T*B, where
%          T is square and upper triangular and B is arbitrary.
%          Usage: C = TRIPROD(T, B, NMIN).
%          NMIN >= 1 is used in a test that determines at what level to
%          terminate the recursions.  Default: NMIN = 4.

if nargin < 3, nmin = 4; end
[m,n1] = size(T);
[n2,p] = size(B);
if m ~= n1 | n1 ~= n2, error('Arguments have incorrect dimensions'), end

if m <= nmin | p == 1, C = T*B; return, end

C = zeros(m,p);
meven = ~rem(m,2); peven = ~rem(p,2);

% Reduce dimensions to make them even; correct later.
if ~meven, m = m-1; end
if ~peven, p = p-1; end

m2 = m/2; p2 = p/2;
im = 1:m2; ip = 1:p2;
jm = m2+1:m; jp = p2+1:p;

C(im, ip) = triprod(T(im,im), B(im,ip), nmin) + ...
            strass(T(im,jm), B(jm,ip), nmin);
C(im, jp) = triprod(T(im,im), B(im,jp), nmin) + ...
            strass(T(im,jm), B(jm,jp), nmin);
C(jm, ip) = triprod(T(jm,jm), B(jm,ip), nmin);
C(jm, jp) = triprod(T(jm,jm), B(jm,jp), nmin);

% Corrections for odd dimensions
if ~peven
    C(:, p+1) = T(:,1:m)*B(1:m, p+1);
end
if ~meven
   C = C + T(:,m+1)*B(m+1,:);
end
```

```
function X = trisol(T, B, nmin)
%TRISOL   Strassen-based algorithm for solving TX = B,
%         where T is square and upper triangular and B is arbitrary.
%         Usage: X = TRISOL(T, B, NMIN).
%         NMIN >= 1 is used in a test that determines at what level to
%         terminate the recursions.  Default: NMIN = 4.

%         Note: flop count is misleading because base level '\' uses
%         Gaussian elimination and condition estimation.

if nargin < 3, nmin = 4; end
[m,n1] = size(T);
[n2,p] = size(B);
if m ~= n1 | n1 ~= n2, error('Arguments have incorrect dimensions'), end

if m <= nmin | p == 1, X = T\B; return, end

X = zeros(m,p);
meven = ~rem(m,2); peven = ~rem(p,2);

% Reduce dimensions to make them even; correct later.
if ~meven, m = m-1; end
if ~peven, p = p-1; end

m2 = m/2; p2 = p/2;
im = 1:m2; ip = 1:p2;
jm = m2+1:m; jp = p2+1:p;

if ~meven
    % Rank-1 correction to be applied before we do the main solve
    X(m+1,1:p) = B(m+1,1:p)/T(m+1,m+1);
    B(1:m,1:p) = B(1:m,1:p) - T(1:m,m+1)*X(m+1,1:p);
end

X(jm,ip) = trisol(T(jm,jm), B(jm,ip), nmin);
TX = strass(T(im,jm), X(jm,ip), nmin);
X(im, ip) = trisol(T(im,im), B(im,ip) - TX, nmin);

X(jm,jp) = trisol(T(jm,jm), B(jm,jp), nmin);
TX = strass(T(im,jm), X(jm,jp), nmin);
X(im, jp) = trisol(T(im,im), B(im,jp) - TX, nmin);

% Final correction for odd dimensions
if ~peven
    X(:, p+1) = T\B(:, p+1);
end
```

REFERENCES

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, Massachusetts, 1974.

[2] D.H. Bailey, Extra high speed matrix multiplication on the Cray-2, SIAM J. Sci. Stat. Comput., 9 (1988), pp. 603–607.

[3] D. Bini and D. Lotti, Stability of fast algorithms for matrix multiplication, Numer. Math., 36 (1980), pp. 63–72.

[4] G. Brassard and P. Bratley, *Algorithmics: Theory and Practice*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[5] R.P. Brent, Algorithms for matrix multiplication, Technical Report CS 157, Computer Science Department, Stanford University, 1970.

[6] R.P. Brent, Error analysis of algorithms for matrix multiplication and triangular decomposition using Winograd's identity, Numer. Math., 16 (1970), pp. 145–156.

[7] D. Coppersmith and S. Winograd, Matrix multiplication via arithmetic progression, Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing, 1987, pp. 1–6.

[8] M.J. Daydé and I.S. Duff, Use of Level 3 BLAS in *LU* factorization on the Cray-2, the ETA-10P, and the IBM 3090-200/VF, Report CSS 229, Computer Science and Systems Division, Harwell Laboratory, 1988.

[9] J.J. Dongarra, J.J. Du Croz, I.S. Duff and S.J. Hammarling, A set of Level 3 basic linear algebra subprograms, Preprint No. 1, Mathematics and Computer Science Division, Argonne National Laboratory, 1988.

[10] J.J. Dongarra, J.J. Du Croz, I.S. Duff and S.J. Hammarling, A set of Level 3 basic linear algebra subprograms: model implementation and test programs, Preprint No. 2, Mathematics and Computer Science Division, Argonne National Laboratory, 1988.

[11] K. Gallivan, W. Jalby and U. Meier, The use of BLAS3 in linear algebra on a parallel processor with a hierarchical memory, SIAM J. Sci. Stat. Comput.,

8 (1987), pp. 1079–1084.

[12] G.H. Golub and C.F. Van Loan, *Matrix Computations*, Second Edition, Johns Hopkins University Press, Baltimore, Maryland, 1989; in press.

[13] D.P. Heyman, Further comparisons of direct methods for computing stationary distributions of Markov chains, SIAM J. Alg. Disc. Meth., 8 (1987), pp. 226–232.

[14] N.J. Higham, The accuracy of solutions to triangular systems, Numerical Analysis Report No. 158, University of Manchester, England, 1988; to appear in SIAM J. Numer. Anal.

[15] N.J. Higham and R.S. Schreiber, Fast polar decomposition of an arbitrary matrix, Technical Report 88-942, Department of Computer Science, Cornell University, 1988.

[16] W. Miller, Computational complexity and numerical stability, SIAM J. Computing, 4 (1975), pp. 97–107.

[17] C.B. Moler, J.N. Little and S. Bangert, *Pro-Matlab User's Guide*, The MathWorks, Inc., 21 Eliot St., South Natick, Massachusetts 01760, 1987.

[18] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, Cambridge, 1986.

[19] R.S. Schreiber, 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.

[20] R. Sedgewick, *Algorithms*, Second Edition, Addison-Wesley, Reading, Massachusetts, 1988.

[21] V. Strassen, Gaussian elimination is not optimal, Numer. Math., 13 (1969), pp. 354–356.