The Journal of Supercomputing, 4, 357–371 (1990)

Using Strassen's Algorithm to Accelerate the Solution of Linear Systems*

DAVID H. BAILEY
NASA Ames Research Center, Mail Stop T045-1, Moffett Field, CA 94035

KING LEE
Computer Science Department, California State University, Bakersfield, CA 93309

HORST D. SIMON
Computer Sciences Corporation, NASA Ames Research Center, Mail Stop T045-1, Moffett Field, CA 94035

(Received March 1990; final version accepted September 1990.)

Abstract. Strassen's algorithm for fast matrix-matrix multiplication has been implemented for matrices of arbitrary shapes on the CRAY-2 and CRAY Y-MP supercomputers. Several techniques have been used to reduce the scratch space requirement for this algorithm while simultaneously preserving a high level of performance. When the resulting Strassen-based matrix multiply routine is combined with some routines from the new LAPACK library, LU decomposition can be performed with rates significantly higher than those achieved by conventional means. We succeeded in factoring a 2048 × 2048 matrix on the CRAY Y-MP at a rate equivalent to 325 MFLOPS.

Key words. Strassen's algorithm, fast matrix multiplication, linear systems, LAPACK, vector computers. AMS Subject Classification 65F05, 65F30, 68A20. CR Subject Classification F.2.1, G.1.3, G.4

1. Introduction

The fact that matrix multiplication can be performed with fewer than $2n^3$ arithmetic operations has been known since 1969, when V. Strassen [1969] published an algorithm that asymptotically requires only about $4.7n^{2.807}$ operations. Since then, other such algorithms have been discovered, and currently the best known result is due to Coppersmith and Winograd [1987], which reduces the exponent of n to only 2.376. Unfortunately, these newer algorithms are significantly more complicated than Strassen's. To our knowledge a thorough investigation of the usefulness of these techniques for an actual implementation has not yet been carried out. It appears that these asymptotically faster algorithms only offer an improvement over Strassen's scheme when the matrix size n is much larger than currently feasible. Thus the remainder of this paper will focus on an implementation and analysis of Strassen's algorithm.

Although Strassen's scheme has been known for over 20 years, only recently has it been seriously considered for practical usage. Partly this is due to an unfortunate myth that has persisted within the computer science community regarding the *crossover* point for Strassen's algorithm—the size of matrices for which an implementation of Strassen's algorithm becomes

*This work is supported through NASA Contract NAS 2-12961.

more efficient than the conventional scheme. For many years it was thought that this level was well over 1000×1000 [Gentleman 1988]. Even recently published reference works have propagated the unfounded assertion (e.g., [Press et al. 1986, p. 76]) that Strassen's algorithm is not suitable for matrices of reasonable size. In fact, for some new workstations, such as the Sun-4 and the Silicon Graphics IRIS 4D, Strassen is faster for matrices as small as 16×16. For Cray systems the crossover point is roughly 128, as will be seen later, so that square matrices of size 2048 on a side can be multiplied nearly twice as fast using a Strassen-based routine (see [Bailey 1988] and below).

Another reason that Strassen's algorithm has not received much attention from practitioners is that it has been widely thought to be numerically unstable. Again, this assertion is not really true, but instead is a misreading of the paper in which the numerical stability of Strassen's algorithm was first studied [Miller 1975]. In this paper, Miller showed that if one adopts a very strict definition of numerical stability, then indeed only the conventional scheme is numerically stable. However, if one adopts a slightly weaker definition of stability, one similar to that used for linear equation solutions, for example, then Strassen's algorithm satisfies this condition. The most extensive study of the stability of Strassen's algorithm is to be found in a recent paper by Higham [1989]. Using both theoretical and empirical techniques, he finds that although Strassen's algorithm is not quite as stable as the conventional scheme, it appears to be sufficiently stable to be used in a wide variety of applications. In any event, Strassen's algorithm certainly appears to be worth further study, including implementation in real-world calculations.

This paper will describe in detail the implementation of a Strassen-based routine for multiplying matrices of arbitrary size and shape (i.e., not just square power-of-two matrices) on Cray supercomputers. A number of advanced techniques have been employed to reduce the scratch space requirement of this implementation, while preserving a high level of performance. When the resulting routine is substituted for the Level 3 BLAS subroutine SGEMM [Dongarra et al. 1988a, 1988b] in the newly developed LAPACK package [Bischof et al. 1988], it is found that LU decomposition can be performed at rates significantly higher than with a conventional matrix multiply kernel. Thus it appears that Strassen's algorithm can indeed be used to accelerate practical-sized linear algebra calculations.

This study is based on the authors' implementation of Strassen's algorithm for the CRAY Y-MP, and all results are based on this implementation. Since the completion of this study, however, the authors learned that Cray Research, Inc., has developed a library implementation of Winograd's variation of Strassen's algorithm. Readers interested in using Strassen's algorithm on Cray systems are directed to this routine, which is known as *SGEMMS*, available under UNICOS 4.0. and later [Cray Research, Inc. 1989]. Furthermore in [Higham 1990] it is pointed out that the IBM ESSL library contains routines for real and complex matrix multiplication by Strassen's method tuned for the IBM 3090 machines.

2. Performance of the Strassen Algorithm

The Strassen algorithm multiplies matrices A and B by partitioning the matrices and recursively forming the products of the submatrices. Let us assume, for the moment, that A and B are $n \times n$ matrices and that n is a power of 2. If we partition A and B into four submatrices of equal size,

$$\begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \tag{1}$$

and compute

$$\begin{aligned} P_1 &= (A_{11} + A_{22})(B_{11} + B_{12}) \\ P_2 &= (A_{21} + A_{22})(B_{11}) \\ P_3 &= (A_{11})(B_{12} - B_{22}) \\ P_4 &= (A_{22})(B_{21} - B_{11}) \\ P_5 &= (A_{11} + A_{12})B_{22} \\ P_6 &= (A_{21} - A_{11})(B_{11} + B_{12}) \\ P_7 &= (A_{12} - A_{22})(B_{21} + B_{22}), \end{aligned} \tag{2}$$

then it can be seen that

$$\begin{aligned} C_{11} &= P_1 + P_4 - P_5 + P_7 \\ C_{12} &= P_3 + P_5 \\ C_{21} &= P_2 + P_4 \\ C_{22} &= P_1 + P_3 - P_2 + P_6 . \end{aligned} \tag{3}$$

If the conventional matrix multiplication algorithm is used in (2), then there will be approximately $7 \cdot 2(n/2)^3$ arithmetic operations in forming the matrix products in (2) and $18 \cdot (n/2)^2$ arithmetic operations involved in adding and subtracting the submatrices on the right side of (2) and (3).

Ignoring for the moment the n^2 terms, we see that the number of arithmetic operations has been reduced from $2n^3$ to $(7/8) \cdot 2n^3$ arithmetic operations in going from the conventional algorithm to the Strassen algorithm. We may continue to apply the Strassen algorithm until the matrices are so small the conventional algorithm is faster for them. Denote this point as $2Q$. The number of times we can apply the reduction is

$$k = \lfloor log_2(n/Q) \rfloor \quad n > Q.$$

The total number of arithmetic operations performed by the conventional n^3 algorithm on submatrices is

$$(7/8)^k \cdot 2n^3 \approx 2Q^{3-log_2 7}\, n^{log_2 7} = 2Q^{0.2}\, n^{2.8}. \tag{4}$$

In the following the performance of the computation of the matrix product AB will be given in MFLOPS, where the MFLOPS for implementations of Strassen's algorithm are also based on $2n^3$ floating point operations. Since the number of floating point operations for Strassen's algorithm is actually less than $2n^3$, we will obtain MFLOPS performance for the new implementation that occasionally exceeds the peak advertised speed of the machine. We have chosen this form for expressing performance because the performance improvements of the new implementation over the traditional matrix multiplication algorithm are expressed more clearly. All numerical experiments were carried out on the CRAY Y-MP of the NAS Systems Division of NASA Ames Research Center. This is an early (serial number 1002) machine with a 6.3-nsec cycle time, and hence a peak performance of 318 MFLOPS per processor. All our results are single processor results. No attempt was made to use all eight processors and multitasking techniques.

The performance of the conventional matrix multiplication algorithm on vector machines is not a smooth function of n, but peaks at points when n is a multiple of the vector register length, drops immediately afterwards, and then increases again to the next multiple of the vector register length. For the CRAY Y-MP there is a 14% drop at $n = 64$, an 8% drop at $n = 128$, and a 4% drop at $n = 256$ (Table 1). All measurements in Table 1 were made on a CRAY Y-MP in multiuser mode. The performance in Table 1 was obtained by using an assembly coded matrix multiplication subroutine provided by Cray Research in SCILIB [Cray Research, Inc. 1989]. Here we list the average of four runs. Performance may vary depending on the load.

For the Strassen algorithm, with $Q = 64$, we expect to see an increase in MFLOPS at each level of recursion due to the reduced number of operations (ignoring n^2 terms). At $n = 130$, however, the Strassen algorithm would require seven matrix multiplications with $n = 65$, and these multiplications would be performed at the low rate of about 244 MFLOPS compared with 269 MFLOPS using the conventional algorithm for $n = 130$ (Table 1). The lower performance would cancel out the gain in reduction in the number of operations (Figure 1). If, on the other hand, we set $Q = 80$, we can be sure that the minimum size of matrices that we multiply is 80 and in this way avoid the *vector length mod 64 = 1* effect at vector length 65. However, we may still see this effect when $n = 258$ where we multiply seven matrices of size 129. But the speed of matrix multiplication of matrices of size 129 is 269 MFLOPS, whereas the speed is 244 MFLOPS when the size is 65 (Figure 2).

The optimal value of Q depends to a large extent on the relative speed of the computation of n^2 and n^3 terms. To see this let S be the speed of the conventional algorithm, S_1 be the

Table 1. Matrix multiplication performance using the conventional algorithm.

n	MFLOPS
64	284
65	244
66	247
67	250
128	289
129	267
130	269
131	271
256	291
257	280
258	281
259	282
∞	296

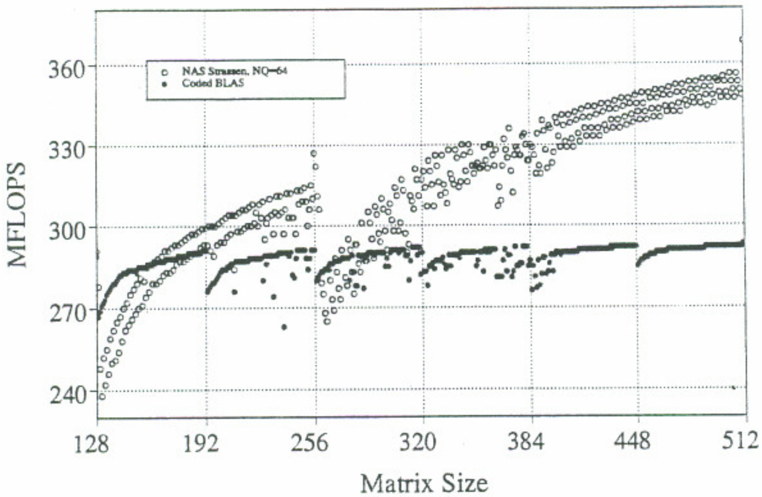

Figure 1. NAS Strassen versus coded BLAS, NQ = 64.

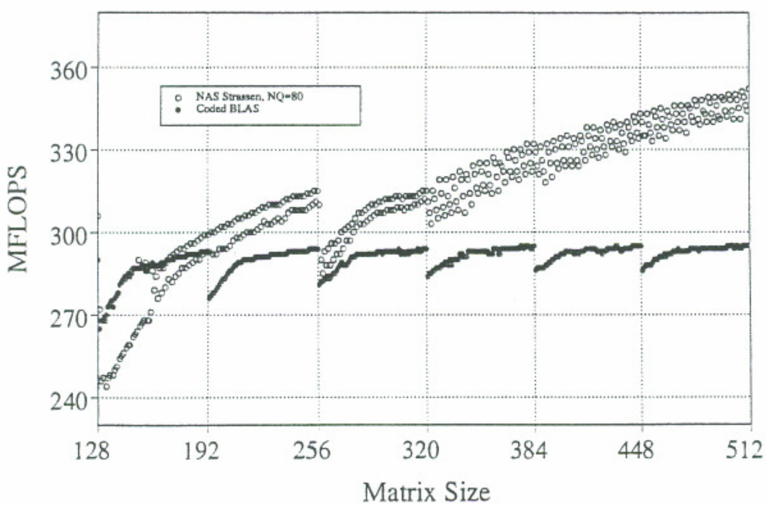

Figure 2. NAS Strassen versus coded BLAS, NQ = 80.

speed when computing the n^3 terms, S_2 be the speed when computing the n^2 terms, and $P = S_1/S_2$. We assume that $S \approx S_1$ and $S > S_2$. For the moment let us consider the n^2 terms as including all operations not involved in the n^3 terms. Thus the n^2 terms include moving submatrices and procedure calls as well as arithmetic operations. To get any gain from the bottom level of the recursion the fraction of the time spent in the slower computation of the n^2 terms must be sufficiently small so as not to offset a 10% reduction in the operation count. In other words, the larger the value of P the less time we must spend in the n^2 terms. We can decrease the time spent in the n^2 terms by increasing Q, the minimal size of the matrices at the lowest level of recursion. Increasing Q has the effect of increasing the number of operations in the n^3 terms and decreasing the number of operations in the n^2 terms. Thus Strassen will work best (small Q and many levels of recursion) when P is relatively small.

In a detailed analysis of arithmetic operations of the Strassen algorithm, Higham [1989] has shown that, assuming the speed of scalar multiplication is the same as scalar addition, $Q = 8$ minimizes the operation count for square matrices of any size greater than eight. Since the computations of the n^2 terms are slower than those of the n^3 terms, the value of eight will be a lower bound on the optimal value of Q. For machines with conventional architecture like Sun workstations a reasonable value for optimal Q might be 16. For a balanced vector machine like the CRAY Y-MP, chaining and more intensive use of registers for the n^3 terms would increase the ratio P and a reasonable value of an optimal Q may be around 80. While the CRAY-2 does not have chaining, it can still produce one addition and one multiplication result in one clock cycle. It also has fast local memory and slow main memory, and that would further increase the ratio P. A few measurements indicate that the optimal value of Q on the CRAY-2 is about 200. In view of the complex dependence of Q on the architecture it seems that the best way to determine Q is heuristically.

The operation count for the n^2 terms is $18 \cdot (n/2)^2$. On vector machines this count is not a good indication of the time taken to perform the computations because the speed of computing the n^2 terms will be much slower than the speed of computing the n^3 terms for the reasons given above. Let us assume that the time to perform the computation of the n^2 terms is $C(n/2)^2$, where we can adjust the value of C to take into account the relative speed of performing the n^2 terms and the n^3 terms. At the k-th level of recursion, the time spent on the n^2 terms is $7^{k-1} \cdot C(n/2^k)^2$. The total time spent on the n^2 terms is, assuming $k = \lfloor log_2(n/Q) \rfloor$,

$$T = C \left(\left(\frac{n}{2}\right)^2 + 7\left(\frac{n}{2^2}\right)^2 + \ldots + 7^{k-1}\left(\frac{n}{2^k}\right)^2 \right) \tag{5}$$

$$\approx \frac{C}{3 \cdot Q^{0.8}} n^{2.8}. \tag{6}$$

We should first note that each term in the series in equation (5) refers to a level of recursion and that the magnitude of the terms in the series is rapidly increasing. This means that most of the contribution of the n^2 terms occur deep in the recursion. Secondly, the coefficient of $n^{2.8}$ in (6) will, in general, be comparable to the coefficient of $n^{2.8}$ in (4). Thirdly, note that k is not a continuous function of n, but jumps when $n = 2^iQ$, $i = 2, \ldots$ The fraction of time spent in the n^2 terms will be at a local maximum at $n = 2^iQ$ and will decrease as n increases until just before $n = 2^{i+1}Q$, at which point another term is added to the series in (5) and the fraction will take a jump. This accounts for part of the performance drop at $n = 128$ and at $n = 256$ in Figure 1.

Let R be the ratio of the time spent in the n^2 terms to the n^3 terms and T be the total time to perform the matrix multiplication. Then from (4), (5), and (6) we can see that

$$R = \frac{C\left(\frac{1}{4} + \ldots + \frac{7^{k-1}}{4^k}\right)}{\left(\frac{7}{8}\right)^k \cdot 2n} \tag{7}$$

$$\approx \frac{C}{6Q} \tag{8}$$

$$T \approx \left(\frac{C}{3Q^{.8}} + 2Q^{.2}\right) n^{2.8}. \tag{9}$$

Flow traces of the Strassen algorithm, with $Q = 64$, were taken for $n = 128$ and $n = 256$, and R was found to be 0.10 and 0.12, respectively. The corresponding values of C turn out to be 90 and 71. Since for $n = 128$ the ratio of the number of arithmetic operations performed in the n^2 terms and in the n^3 terms is about 0.02, we can infer that, ignoring overhead, the speed of the n^2 terms is one-fifth the speed of the n^3 terms. If one takes a value of 80 for C, then for large n, R will tend toward 0.20, and the n^2 terms will account for about 16% of the coefficient in (9).

If we fix C and n in (9), then T will have a minimum when $Q = 2/3C$. This value of Q, Q_0, is the value of Q that theoretically minimizes the time to perform matrix multiplication for square matrices for a particular value of C and n. If we substitute this value into (8) and (9), we find

$$R = \frac{1}{4}$$

$$T = \frac{5}{2} Q_0^{.2} n^{2.8}$$

$$= \frac{5}{2}\left(\frac{2}{3}C\right)^{.2} n^{2.8}.$$

Therefore, when $Q = Q_0$ we should expect that for square matrices about one-fifth of the time is spent computing the n^2 terms. The total time for the matrix multiplication depends on $Q_0^{.2}$ or $C^{.2}$ so that the smaller the ratio P, the smaller the coefficient of $n^{2.8}$.

It is difficult to determine the optimal value Q_0 over all values of n, especially for vector machines. First, if n is not a power of 2, that is to say if n has an odd factor, special corrections will have to be made (see below) that will increase the number of operations in the n^2 terms. This means that C is a function of n. Secondly, the effect of different vector lengths in the computations may affect C. One problem is that one value of Q may give relatively good performance on matrices of size N and poor performance for matrices of size M, and at another value of Q we might have fair performances for matrices of these sizes. We saw that with $Q = 64$ we had good performance when $n = 128$, but poor performance when $n = 130$. With $Q = 80$ we had fair performance with $n = 128$ and $n = 130$ (no Strassen in both cases). In other words, for every square matrix there may be a different optimal Q, and the situation for rectangular matrices will be even more complicated. On the CRAY Y-MP we found a value of 80 for C would yield a Q_0 of about 57. A value of 80 may be preferable to avoid the *vector length mod 64 = 1* effect.

Figure 3 plots our Strassen against the Cray Strassen. The code of the n^2 terms was written in Fortran for the NAS Strassen and written in assembly for the Cray Strassen. The code for the n^3 terms for both programs were written in assembly. The value of C should be smaller for the Cray program, and that leads to a smaller value for the coefficient in (9). The assembly coding of the important order n^2 computations probably accounts for the performance differences observed in Figure 3.

We have assumed that at each stage of the recursion we could partition the matrices into submatrices of size $(n/2)$. In the event that n is odd we may multiply matrices of size $n - 1$ and make a correction after the multiplication is performed. The complexity of the correction will be $O(n^2)$, and the work involved in the correction is in addition to the n^2 terms. This correction for odd dimensions will be expensive if it occurs in the stages of recursion corresponding to the last terms of the series in (5). In those cases, not only will there be large numbers of corrections to be made, but also the corrections will be made with relatively

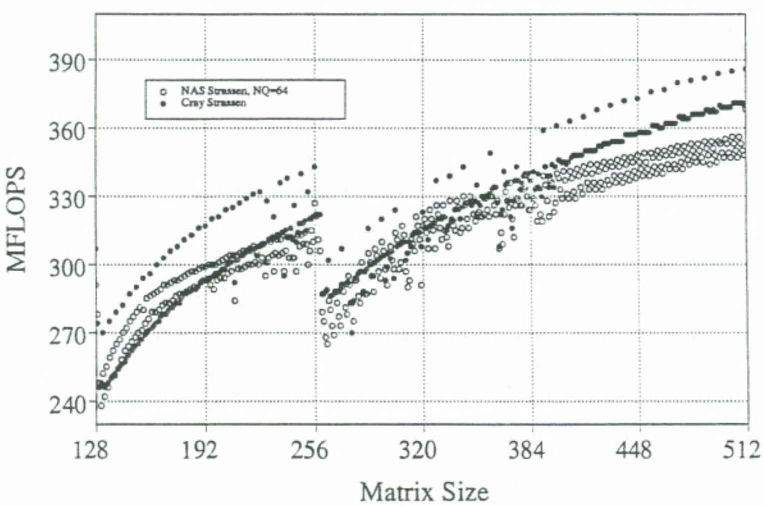

Figure 3. Cray Strassen versus NAS Strassen. NQ = 64.

Table 2. Multiplication performance using NAS Strassen.

n	MFLOPS
260	279
261	275
262	268
263	265
264	284
265	280
266	273
267	269
268	288

short vectors. These corrections introduce a pattern of variation in performance. For example, if $Q = 64$ and $n = 260$, then at each stage of the recursion n is even, and no corrections need be made. When $n = 261$, one correction on a matrix of size 261 has to be made. If $n = 262$, we have to make seven corrections on matrices of size 131. If $n = 263$, we have to make one correction for a matrix of size 263, and seven for matrices of size 131. If $n = 264$ we do not have to make any corrections. This pattern will repeat for the next four dimensions. Table 2 contains the MFLOPS performance for the Strassen algorithm when n takes on values from 260 to 268.

The Strassen algorithm can be applied to nonsquare matrices as well as square matrices. Let A be $l \times m$ and B be $m \times n$; the conventional algorithm requires approximately $2lmn$ arithmetic operations. We stop recursion when the minimum of l, m, n is less than $2Q$. If one or two of the dimensions is much greater than the smallest dimension, then the n^2 terms become a smaller fraction of the total operation count, and the reduction in operation count becomes more pronounced. Table 3 gives the performance for several rectangular matrices. We would like to point out that in 1970 Brent used both the odd dimension correction and Strassen for rectangular matrices in his unpublished report [Brent 1970].

In summary, the performance of the Strassen algorithm is influenced by the following factors: the reduction in the number of operations, the dimensions of the matrices on which the conventional algorithm operates (e.g., 65 versus 127 on the Y-MP), the proportion of the computation due to the n^2 terms, and the number of times we have to correct for odd dimensions.

3. Memory Requirements

The straightforward way of implementing the Strassen algorithm would be to compute the submatrices $P_1, P_2, \ldots, P_7$, appearing in equations (2), and then compute C_{11}, C_{12}, C_{21}, C_{22}. For this method we need two scratch arrays to hold the operands on the right side of equations (2) and seven scratch arrays to hold the matrices $P_1, \ldots, P_7$. Then the amount of scratch space required is $9(n/2)^2$. At the k-th level of recursion the space requirement will be $9(n/2^k)^2$. The total space required will be

$$9n^2\left(\frac{1}{4} + \frac{1}{16} + \ldots + \frac{1}{4^k}\right) = 3n^2\left[1 - \left(\frac{1}{4}\right)^k\right].$$

Table 3. Matrix multiplication for rectangular matrices using NAS Strassen.

l	m	n	MFLOPS
128	128	128	291
256	128	128	300
512	128	128	304
128	128	128	291
128	256	128	305
128	512	128	312
128	128	128	291
128	128	256	296
128	128	512	298
128	128	128	290
256	256	128	311
512	512	128	321
128	128	128	291
256	128	256	303
512	128	512	309
128	128	128	289
128	256	256	309
128	512	512	319

Note that each term of the series corresponds to a level of recursion and that the first two terms of the series account for more than 90% of the sum.

An alternative method is to compute P_1, store that result in C_{11} and C_{22}, compute P_2, store that result in C_{21}, subtract it from C_{22}, and so on. We need two matrices to hold the operands on the right sides of equations (2) and only one to hold the matrices $P_1, \ldots, P_7$. The total space required would be

$$3n^2\left(\frac{1}{4} + \frac{1}{16} + \ldots + \frac{1}{4^k}\right) = n^2\left[1 - \left(\frac{1}{4}\right)^{k+1}\right].$$

Even though the number of arithmetic operations is the same for both methods, the second method would run more slowly on vector machines because there is more data movement. The fast (first) method holds more intermediate results in registers in the computation of $C_{11}, \ldots, C_{22}$ in equations (3). The penalty in speed for the second method is machine-dependent. The difference between the fast and slow versions was less than 3% for n between 128 and 512.

Fortunately, it is possible to combine the best features of both methods. We can get most of the benefits of the smaller memory requirements of the slow method if we use that method at the first one or two levels of recursion corresponding to the early terms of (3). We may expect to get most of the benefits of the faster method by using that method deep in the recursion, corresponding to the later terms of (5).

If one level of the slow method were used, and the remaining levels used the faster method, the scratch space requirement would be $1.5 \cdot n^2$; if two levels of the slow method were used, the scratch space requirement would be $1.25 \cdot n^2$; if three levels of the slow method were used, the space requirement would be $1.125 \cdot n^2$. The most desirable version for a computer may be dictated by the architecture and configuration. Using zero or one slow level might be appropriate for the CRAY-2 because it has large common memory and it is relatively slow on the n^2 terms. Using one or two levels of the slow method would be appropriate for the CRAY Y-MP because it has relatively small memory per processor and is relatively efficient computing the n^2 terms.

For the case of rectangular matrices, a bound for the required scratch space for the slower and faster versions turns out to be

$$\frac{lm}{3} + \frac{mn}{3} + \frac{7 \cdot ln}{3} \quad \text{(fast version)}$$

$$\frac{lm}{3} + \frac{mn}{3} + \frac{ln}{3} \quad \text{(slow version).}$$

We might mention that the Cray Research implementation of Winograd's variation of the Strassen algorithm required memory bounded by $2.34 \cdot \max(l, m) \cdot \max(m, n)$ (see [Cray Research, Inc. 1989].

In the special cases when one or two of the dimensions of A or B is much less than the other dimension(s), further savings of scratch memory are possible. Take, for example, the case when $l = K$, $m = 4K$, $n = 4K$. The product AB can be computed by multiplying A with four $4K \times K$ submatrices of B. The amount of scratch space required for the slow version would be $3K^2$ instead of $8K^2$. Multiplying A by the submatrices of B in this manner does not increase the count of operations, but some bookkeeping and short vector effects are introduced.

The NAS implementation is called SSGEMM and uses the same calling sequence as the LINPACK subroutine SGEMM. The scratch memory is in common, and there is a default size. However, the size could be increased by compiling and loading a function such as the following:

```
INTEGER FUNCTION GETSCR
PARAMETER (ISIZE = 100000)
COMMON/SCRATCH/X(ISIZE)
GETSCR = ISIZE
END
```

When SSGEMM is called, it is first determined whether there is enough scratch memory to use the full Strassen algorithm. If there is not enough memory to use the full Strassen algorithm, it is determined whether there is enough memory to use a partial Strassen. This means that, for example, only two levels of recursion in the Strassen algorithm are used, when with more memory three or more levels could have been used. If partial Strassen cannot be used, then the subroutine SSGEM will call SGEMM, the conventional matrix multiply routine that does not need any scratch memory. Our implementation uses multiple copies of the code, and there are at most six levels of recursion allowed. This means that our version could in principle multiply 4000×4000 matrices. We note that the Cray subroutine SGEMMS uses the same calling sequence as SGEMM, except that the last parameter is a scratch array.

In summary we have implemented a flexible, general purpose, matrix-matrix multiplication subroutine in the style of the Level 3 BLAS [Dongarra et al. 1988a, 1988b]. This subroutine can be used in all contexts, where the Level 3 BLAS routine SGEMM is used, subject to the availability of the additional workspace. We will now demonstrate this point with the linear equation solving routine from LAPACK, which makes extensive use of SGEMM and Level 3 BLAS.

4. Applications to LAPACK

LAPACK [Bischof et al. 1988] is a new library of linear algebra routines being developed with the objective of achieving very high performance across a wide range of advanced systems. The main feature of this package is its reliance on block algorithms that preserve data locality, together with a facility that permits near-optimal tuning on many different systems.

SGETRF is a LAPACK subroutine that uses dense matrix multiplication. This subroutine performs an LU decomposition on an $n \times n$ matrix A by repeatedly calling the subroutine SGETF2 to perform an LU decomposition on a diagonal submatrix of size NB, calling STRSM to compute the superdiagonal block of U, and calling SGEMM to perform matrix multiplication to update the diagonal and subdiagonal blocks. The matrix multiplications are performed on matrices of sizes $J \times NB$ and $NB \times (n - J)$ for $J = NB, 2 \cdot NB, \ldots, n$. The parameter NB, also referred to as block size, can be chosen for performance tuning.

A Fortran version of this subroutine from Argonne National Laboratory was linked to call SGETF2 and STRSM from the Cray libraries and either the Cray SCILIB version of SGEMM or our Strassen version of SGEMM.

Table 4 gives the approximate time spent in each of the three subroutines and Table 5 gives the performance results for different sizes of n and NB. The timings are given for the case when the leading dimension of A is 2049. If the leading dimension were 2048, the performance would be less due to memory stride conflicts in STRSM. Table 5 shows that no single value of NB gives uniformly the best performance for varying problem sizes. Even when A has dimension 2048, we are performing matrix multiplication with matrices

Table 4. Fraction of time spent in subroutines of SGETRF NAS Strassen, $NB = 512$.

N	SGETF2	SGEMM	STRSM
512	0.99	0.00	0.00
1024	0.51	0.29	0.19
1536	0.33	0.36	0.30
2048	0.25	0.39	0.36

Table 5. Performance of SGETRF in MFLOPS as a function of n and NB.

	n			
	512	1024	1536	2048
NB	Coded BLAS			
128	262	284	290	291
256	250	281	288	291
512	229	274	285	289
768	229	250	281	288
1024	229	260	280	286
	Strassen			
128	250	290	299	304
256	243	297	311	317
512	217	290	314	325
768	216	272	305	318
1024	216	258	294	319

whose minimum dimension is 512 at the rate of 368 MFLOPS, and this rate is achieved only on part of the computation. In the case when $n = 2048$ and $NB = 512$, three matrix multiplications are performed. The dimensions of the matrix factors are 1536 by 512 and 512 by 512; 1024 by 1024 and 1024 by 512; and 512 by 1536 and 1536 by 512. With $Q = 64$ and using one level slow method we see that the scratch memory requirement is bounded by 11/12 megawords. As mentioned earlier, a bound for the memory requirement for implementation of the algorithm by Cray Research is $2.34 \cdot \max(l, m) \cdot \max(m, n)$. The matrix multiplication that requires the most scratch space is the one that multiplies 512 by 1536 and 1536 by 512 matrices, and the scratch memory requirement to form this product is 5.75 megawords. It is possible to decrease the memory requirements by a factor of about two for the Cray Strassen by partitioning the matrices into two submatrices, performing the multiplications on submatrices and combining the products of the submatrices.

Higham [1989] discusses several other Level 3 BLAS subroutines that may use the Strassen algorithm. One is a subroutine to multiply an upper triangular matrix U with a general matrix B. Higham writes

$$A = \begin{pmatrix} U_{11} & U_{12} \\ 0 & U_{22} \end{pmatrix} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} U_{11}B_{11} + U_{12}B_{21} & U_{11}B_{12} + U_{12}B_{22} \\ U_{22}B_{21} & U_{22}B_{B21} \end{pmatrix}.$$

The two dense matrix multiplications involving U_{12} may be computed using the Strassen algorithm, and the remaining products are products of triangular matrices with general dense matrices and can be computed recursively. It can be shown, assuming square matrices, that the number of arithmetic operations is $O(n^{2.8})$. However, the asymptotic speed is approached more slowly than in the case of matrix multiplication. For example, for 1024×1024 matrices half the operations would be computed at the rate of 368 MFLOPS, (the rate of the Strassen algorithm for $n = 512$), a quarter of the operations would be carried out at the rate of

326 MFLOPS ($n = 256$), and one-eighth at 289 MFLOPS. The remaining one-eighth would be carried out at the conventional triangular matrix multiply speed. The improvement in speed for the triangular matrix multiply should be significantly better than that of the SGETRF decomposition because a larger fraction of the operations can be computed by the Strassen algorithm. We should expect similar performance improvements for the other Level 3 BLAS subroutines discussed by Higham that use matrix multiplication.

5. Conclusions

The speed in terms of effective MFLOPS for the Strassen algorithm increases without bound with increasing size of the matrix. On the CRAY Y-MP, the Strassen algorithm increased the performance by 10% every time the dimension doubled. For $n = 1024$ the conventional Cray matrix multiply routine had a performance of 296 MFLOPS, while our Strassen could run at over 400 MFLOPS and the Cray version even faster. The increase in performance with matrix size is not a smooth function of the size of the matrices, but shows minor oscillations and jumps. The causes of the jumps and oscillations are the drop in operation count in the n^3 terms, short vector effects, the effect of n^2 terms, and corrections for even and odd matrices.

We succeeded in implementing a general purpose matrix-matrix multiplication routine for the CRAY Y-MP, which can handle rectangular matrices of arbitrary dimension. Even for moderately sized matrices this routine outperforms the functionally equivalent Level 3 BLAS subroutine based on the traditional multiplication algorithm. Because of its flexibility, this subroutine can be used as a computational kernel for higher level applications. This has been demonstrated by integrating this routine with the linear equation solver in LAPACK.

Many LAPACK subroutines are using SGEMM, not just the dense unsymmetric LU factorization. The use of Strassen's method potentially could speed up a number of linear equations and eigenvalue computations. In addition, complex routines in LAPACK could be improved using the *trick* discussed in [Higham 1990]. Generally, we believe that the performance of LAPACK and Level 3 BLAS subroutines that use dense matrix multiplication will improve if the Strassen algorithm is employed; the exact improvement will depend on the size of the problem and to a large part on the fraction of the computation that can take advantage of the Strassen algorithm.

Acknowledgment

We wish to thank Nick Higham for reviewing an earlier version of this manuscript and suggesting numerous improvements to the presentation of our results.

References

Bailey, D.H. 1988. Extra high speed matrix multiplication on the CRAY-2. *SIAM J. Sci. Stat. Comp.*, 9, 3: 603–607.

Bischof, C., Demmel, J., Dongarra, J., DuCroz, J., Greenbaum, A., Hammarling, S., and Sorensen, D. 1988. LAPACK working note #5—Provisional contents. Tech. rept. ANL-88-38, Argonne Nat. Laboratory (Sept.).

Brent, R.P. 1970. Algorithms for matrix multiplication. Tech. rept. CS 157, Comp. Sci. Dept., Stanford Univ.

Coppersmith, D., and Winograd, S. 1987. Matrix multiplication via arithmetic progression. In *Proc., 19th Annual ACM Symp. on the Theory of Computing*, pp. 1–6.

Cray Research, Inc. 1989. *UNICOS Math and Scientific Library Reference Manual*. No. SR-2081. Version 5.0 (Mar.).

Dongarra, J.J., DuCroz, J., Duff, I., and Hammarling, S. 1988a. A set of Level 3 basic linear algebra subprograms. Tech. rept. MCS-P1-0888, MCSD, Argonne Nat. Laboratory (Aug.).

Dongarra, J.J., DuCroz, J., Duff, I., and Hammarling, S. 1988b. A set of Level 3 basic linear algebra subprograms: Model implementation and test programs. Tech. rept. MCS-P2-0888, MCSD, Argonne Nat. Laboratory (Aug.).

Gentleman, M.J. 1988. Private commun.

Higham, N.J. 1989. Exploiting fast matrix multiplication within the Level 3 BLAS. Tech. rept. TR 89-984, Dept. of Comp. Sci., Cornell Univ., Ithaca, N.Y. (Apr.). (To appear in ACM TOMS.)

Higham, N.J. 1990. Stability of a method for multiplying complex matrices with three real matrix multiplications. Numerical Analysis Rept. 181, Dept. of Math., Univ. of Manchester (Jan.).

Miller, W. 1975. Computational complexity and numerical stability. *SIAM J. Computing*, 4: 97–107.

Press, W.H., Flannery, B.P., Teukolsky, S.A., and Vetterling, T. 1986. *Numerical Recipes*. Cambridge Univ. Press, N.Y.

Strassen, V. 1969. Gaussian elimination is not optimal. *Numer. Math.*, 13: 354–356.

