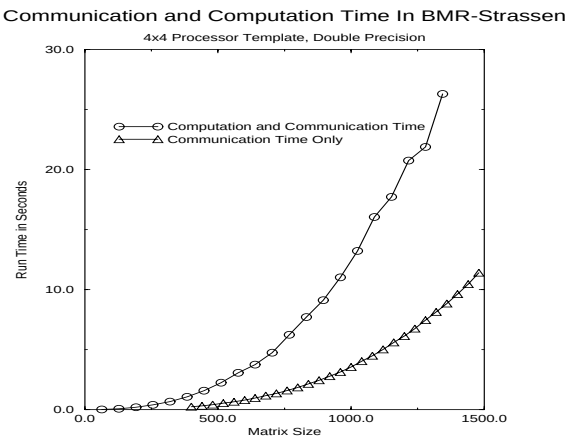

Figure 12: Communication time compared with total running time in the BMR-Strassen method on a 4×4 processor template.

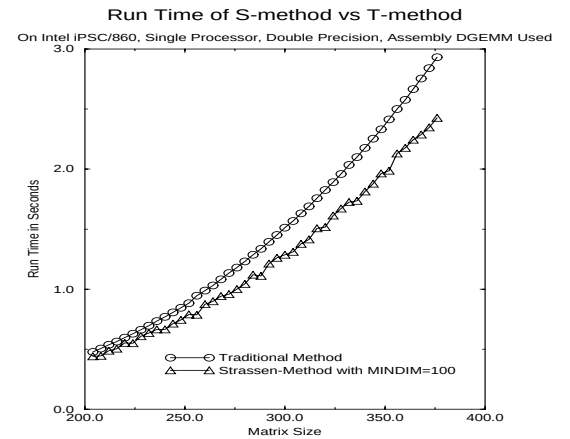

Figure 13: Running time of the assembly DGEMM routine vs that of the C routine of the S-method coupled with DGEMM on single processor. MINDIM=100 for the S-method.

Strassen's algorithm has been presented and compared with other parallel matrix multiplication algorithms. On the Intel iPSC/860, the BMR-Strassen method coupled with assembly BLAS routines offers the fastest approach to matrix multiplication. When the communication is not very costly compared to computation, the Strassen-BMR method may offer a faster approach. Strassen-BMR method also requires less additional space than BMR-Strassen method.

Acknowledgment. This paper results from an internship by Qingshan Luo and John B. Drake at the Mathematical Sciences Section at the Oak Ridge National Laboratory from June 1994 to July 1994. The internship is supported by the Tonya Internship for Private Economy at the University of the South. This paper is first written on July 29, 1994 and is revised on November 9, 1994. Correspondence should be sent to: Qingshan Luo, SPO, University of the South, Sewanee, TN 37383, E-mail: Luo@acm.org; or John B. Drake, Mathematical Sciences Section, Oak Ridge National Laboratory, PO Box 2008, Bldg. 6012, Oak Ridge, TN 37831-6367, E-mail: bbd@msr.epm.ornl.gov.

References

[1] CHOI, J., DONGARRA, J.J., WALKER, D.W., *PUMMA: Parallel Universal Matrix Multiplication Algorithm on Distributed Memory Concurrent Computers. Technical Report TM-12252, Oak Ridge National Laboratory*, August 1993.

[2] CHOU, C., DENG, Y., WANG, Y., *A Massively Parallel Method for Matrix Multiplication Based on Strassen's Method.* (Submitted to *SIAM Journal on Scientific Computing.*

[3] COPPERSMITH, D., WINOGRAD, S. *Matrix multiplication via arithmetic progressions.* In *Proceeding of the Nineteenth Annual ACM Symposium on Theory of Computing.* 1987, 1-6.

[4] DOUGLAS, C., HEROUX, M., SLISHMAN, G., *GEMMW: A Portable Level 3 BLAS Winograd Variant of Strassen's Matrix-Matrix Multiply Algorithm. Journal of Computational Physics 110* (1994), 1-10.

[5] DUNIGAN, T., *Early Experiences and Performance of the Intel Paragon. Technical Report TM-12194, Oak Ridge National Laboratory*, April 1994.

[6] FOX, G. C., HEY, A. I., OTTO, S., *Matrix Algorithms on the Hypercube I: Matrix Multiplication. Parallel Computing 4* (1987), 17.

[7] LADERMAN, J., PAN, V., SHA, X., *On Practical Algorithms for Accelerated Matrix Multiplication. Linear Algebra and Its Applications.* 1992, 557-588.

[8] STRASSEN, V. *Gaussian elimination is not optimal. Numer. Math. 13*(1969), 354-356.

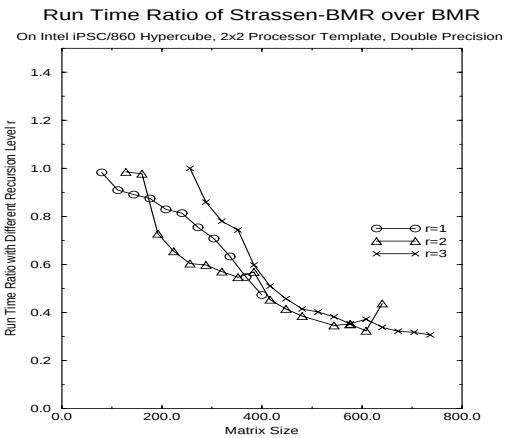

Figure 6: Running time ratio of the Strassen-BMR method to the BMR method on 2×2 processor template with r=1,2,3.

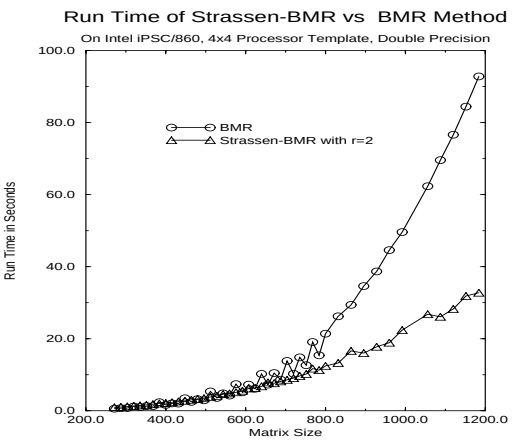

Figure 7: Running time of the Strassen-BMR method and the BMR method on 4×4 processor template with r=2.

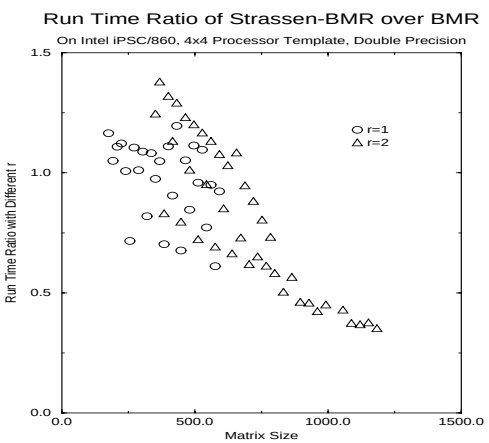

Figure 8: Running time ratio of the Strassen-BMR method to the BMR method on 4×4 processor template with r=1,2.

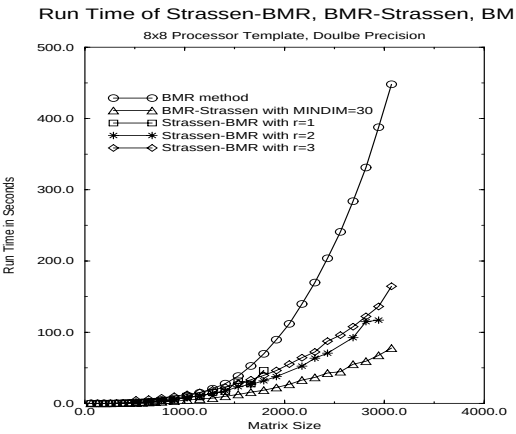

Figure 9: Running time of the Strassen-BMR method, the BMR-Strassen method, and the BMR method on a 8×8 processor template.

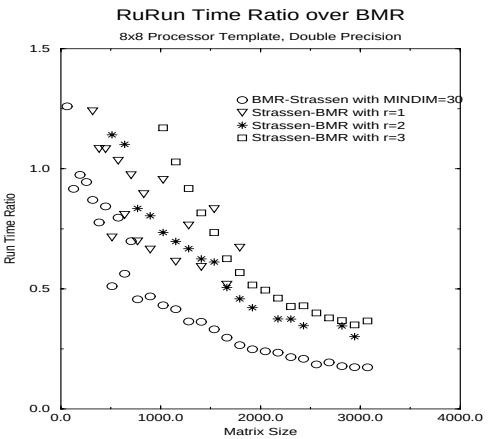

Figure 10: Running time ratio of the Strassen-BMR method and the BMR-Strassen method to the BMR method on a 8×8 processor template.

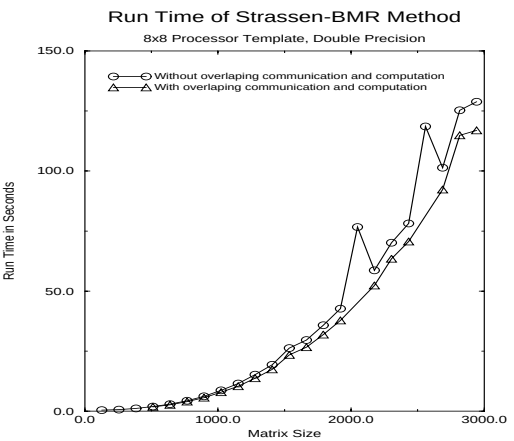

Figure 11: Running time of Strassen-BMR method with and without overlapping communication and computation on a 8×8 processor template with r=2.

the Strassen-BMR method should be faster than the BMR method when M is large enough. Compared with the running time of the BMR-Strassen method in Equation 7, it is noted that the Strassen-BMR has more communication time(third term) but less computation time(second term), if the recursion level r is the same. When the communication is slow, the quadratic term for communication will become comparable to the cubic term for computation. In such case the Strassen-BMR method will become slower than the Strassen-BMR method.

In the actual implementation of the Strassen-BMR method, 6 out of the 7 submatrix multiplications using BMR method at the bottom level are grouped into 3 pairs to overlap the computation and communication. This is possible because the communication is done mostly in background by message co-processors on the Intel iPSC/860. Each processor can still perform computing when waiting for a message, as long as computation and communication deal with different memory locations. The following pseudocode describes how to overlap communication and computation in computing the two submatrix multiplications A_1B_1 and A_2B_2.

$C_{1ij} = A_{1ij}B_{1ij}$ for all processors
Start rolling A_1 leftwards and B_1 upwards
$C_{2ij} = A_{2ij}B_{2ij}$ for all processors
Start rolling A_2 leftwards and B_2 upwards
DO $I = 1, P-1$
  Wait until the rollings of A_1 and B_1 are done
  $C_{1ij} = C_{1ij} + A_{1ij}B_{1ij}$ for all processors
  Wait until the rollings of A_1 and B_1 are done
  $C_{2ij} = C_{2ij} + A_{2ij}B_{2ij}$ for all processors
ENDDO
Roll A_1 leftwards and B_1 upwards
Roll A_2 leftwards and B_2 upwards

However, the requirement for additional space in the above algorithm becomes $\frac{4}{3}M^2$, which is twice as much as straightforward Strassen-BMR method. Note that it is still less than that of the BMR-Strassen method($\frac{5}{3}M^2$).

Performance Analysis. Performance tests for all the algorithms discussed here are carried out on an Intel iPSC/860 with 128 processors. Each processor is an Intel i860 chip with 8MB memory. A communication co-processor handles message passing between processors so that the computational processor is interrupted only minimally. All the algorithms discussed here are easily scalable to any number of processors and matrices of any dimensions. For simplicity only the results from the square processor templates and square matrices are presented. The entries of all the matrices tested here are random double precision numbers uniformly distributed between -1 and 1.

The Strassen-BMR method is found faster than BMR method, especially when the matrix size is large and the recursion level is high. Figure 5 through Figure 10 are results of the Strassen-BMR method compared with the BMR method, with 2×2, 4×4, and 8×8 processor templates. However, due to the limited matrix size, the Strassen-BMR method reaches its maximum performance when r is just around 3. The BMR-Strassen method is also found faster than the BMR method. Among the three methods, the BMR-Strassen method is the fastest and the Strassen-BMR method is the second fastest, while the BMR method is the slowest(see Figure 9). The overlapping of communication and computation in the Strassen-BMR method gives about 10% improvement (see Figure 11). Note that such overlapping is not beneficial in the other two methods.

Comparing the theoretical performance model for the BMR-Strassen method given by Equation 7 with that of the Strassen-BMR method given by Equation 12, the Strassen-BMR method has less computation time but more communication time than the BMR-Strassen. On the Intel iPSC/860, the ratio of computation speed to communication speed is fairly high. For double precision, the time to perform one floating point operation is about 300 faster than the time to send a number between processors[5]. Therefore the communication time is significant here. Even for the BMR-Strassen method, the communication time is about 40% of the total running time(see Figure 12). When the communication speed is not so slow(on Intel Paragon the communication speed is about only 10 times slower than the computation speed[5]), the Strassen-BMR method may be faster than the BMR-Strassen method.

As a cache based machine with limited memory access bandwidth, the computational performance of the i860 chip on a given operation can be greatly enhanced by judicious use of registers and cache. Normally this level of access is only available through assembly language coding. Intel provides an assembly version of the level-3 BLAS routines which includes the routine DGEMM for matrix-matrix multiplication for double precision numbers. These routines are much faster than standard Fortran BLAS routines. Figure 13 shows running times of the optimized, assembly DGEMM routine and our C routine using the S-method coupled with the assembly DGEMM. Our C routine is about 15% faster than the assembly DGEMM routine. Since the Strassen method is coded in the high level C language, further performance improvement might be obtained by assembly coding the S-method.

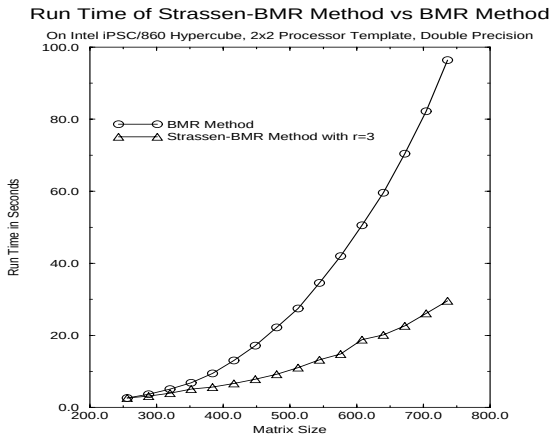

Figure 5: Running time of the Strassen-BMR method and the BMR method on 2×2 processor template with r=3.

Conclusions. A new parallel distributed memory

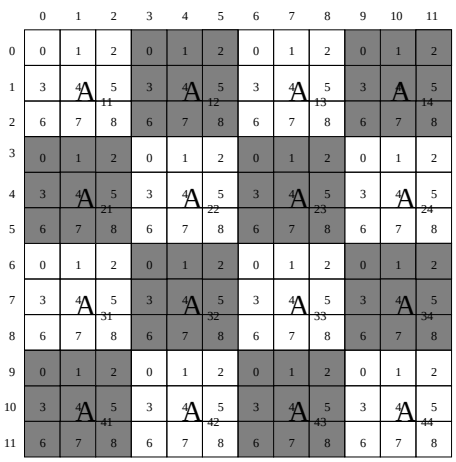

Figure 3: Matrix A with 12×12 blocks is distributed over a 3×3 processor template from a matrix point-of-view. The 9 processors are numbered from 0 to 8. This pattern is used in the Strassen-BMR method when the recursion level is 2.

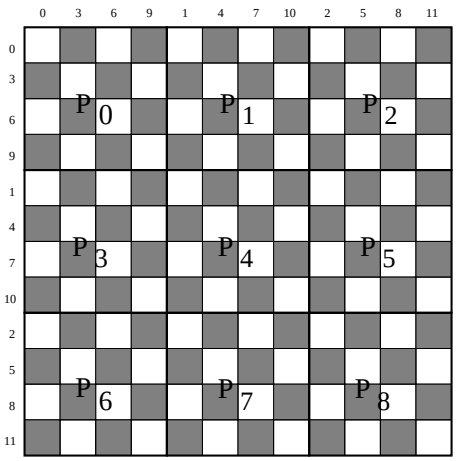

Figure 4: Same as Figure 3, but from a processor point-of-view.

(or subtraction) of submatrices performed in the S-method at all recursion levels can thus be performed in parallel without any interprocessor communication. From the processor point-of-view, each processor does its local submatrix additions and subtractions. Submatrix multiplications are calculated recursively using the S-method. At the last level of recursion the submatrix multiplications are calculated using the BMR method. Therefore, the Strassen-BMR method uses the S-method at the top level, and the T-method at the bottom level.

We use the same notations as in the BMR-Strassen method. Since there are 15 submatrix additions and subtractions and 7 submatrix multiplications in each recursion, the total running time for the Strassen-BMR method is

$$T(M) = 15T_{add}(\frac{M}{2}) + 7T(\frac{M}{2}), \tag{8}$$

where $T_{add}(M/2)$ is the running time to add or subtract submatrices of order $M/2$. Note that there are P^2 processors running in parallel. Therefore

$$T_{add}(\frac{M}{2}) = \frac{(\frac{M}{2})^2 t_{comp}}{P^2}. \tag{9}$$

Substitute the above formula into Equation 8 we have

$$T(M) = (\frac{15t_{comp}}{4P^2})M^2 + 7T(\frac{M}{2}) = sM^2 + 7T(\frac{M}{2}), \tag{10}$$

where $s = \frac{15t_{comp}}{4P^2}$. Use the above formula recursively to obtain

$$\begin{aligned} T(M) &= sM^2 + 7T(\tfrac{M}{2}) \\ &= sM^2 + 7(s(\tfrac{M}{2})^2 + 7T(\tfrac{M}{4})) \\ &= sM^2(1 + \tfrac{7}{4}) + 7^2 T(\tfrac{M}{2^2}) \\ &= sM^2(1 + \tfrac{7}{4}) + 7^2(s(\tfrac{M}{4})^2 + 7T(\tfrac{M}{8})) \\ &= sM^2(1 + \tfrac{7}{4} + (\tfrac{7}{4})^2) + 7^3 T(\tfrac{M}{2^3}) \\ &\vdots \\ &= sM^2(1 + \tfrac{7}{4} + \cdots + (\tfrac{7}{4})^{r-1}) + 7^r T(\tfrac{M}{2^r}) \\ &= sM^2 \tfrac{1-(\frac{7}{4})^r}{1-\frac{7}{4}} + 7^r T(M_0) \\ &\approx \tfrac{4}{3} s(\tfrac{7}{4})^r + 7^r T(M_0). \end{aligned} \tag{11}$$

At the bottom level, the Strassen-BMR method uses the BMR method for submatrix multiplications. Therefore we can use Equation 4 to find $T(M_0)$. Substituting the value of $T(M_0)$ and s we have

$$\begin{aligned} T(M) &\approx \tfrac{5(\frac{7}{4})^r t_{comp}}{P^2} M^2 \\ &\quad + 7^r(\tfrac{2t_{comp}}{P^2} {M_0}^3 + \tfrac{2B\beta}{P} {M_0}^2 + 2P\alpha) \\ &= (\tfrac{7}{8})^r \tfrac{2t_{comp}}{P^2} M^3 + \tfrac{5(\frac{7}{4})^r t_{comp}}{P^2} M^2 \\ &\quad + (\tfrac{7}{4})^r \tfrac{2B\beta}{P} M^2 + 7^r(2P\alpha) \end{aligned} \tag{12}$$

There are four terms in the above equation. The first term is a cubic term with respect to M. It is the computational dominant part and it decreases as the recursion level r increases. The second term is quadratic and it results from the additional submatrix additions and subtractions in the S-method. It increases as r increases. The last two terms represent the communication time. They increase as r increases. Since the first term is the dominant cubic term,

needed to perform an addition or subtraction of matrices of order $N/2$. Therefore $T_{add}(N/2) = t_{comp}(N/2)^2$, where t_{comp} is the execution time for one arithmetic operation. Therefore

$$T(N) = (\frac{15t_{comp}}{4})N^2 + 7T(\frac{N}{2}) = sN^2 + 7T(\frac{N}{2}),$$

where $s = \frac{15}{4}t_{comp}$. Use the above formula recursively we have,

$$\begin{aligned} T(N) &= sN^2 + 7T(\tfrac{N}{2}) \\ &= sN^2 + 7(s(\tfrac{N}{2})^2 + 7T(\tfrac{N}{4})) \\ &= sN^2(1+\tfrac{7}{4}) + 7^2T(\tfrac{N}{2^2}) \\ &= sN^2(1+\tfrac{7}{4}) + 7^2(s(\tfrac{N}{4})^2 + 7T(\tfrac{N}{8})) \\ &= sN^2(1+\tfrac{7}{4}+(\tfrac{7}{4})^2) + 7^3T(\tfrac{N}{2^3}) \\ &\vdots \\ &= sN^2(1+\tfrac{7}{4}+\cdots+(\tfrac{7}{4})^{r-1}) + 7^rT(\tfrac{N}{2^r}) \\ &= sN^2\tfrac{1-(\frac{7}{4})^r}{1-\frac{7}{4}} + 7^rT(N_0) \\ &\approx \tfrac{4}{3}s(\tfrac{7}{4})^rN^2 + 7^rT(N_0). \end{aligned} \quad (6)$$

Note that at the bottom level the T-method is used for submatrix multiplication. Therefore,

$$T(N_0) = 2t_{comp}{N_0}^3 = 2t_{comp}(\frac{N}{2^r})^3.$$

Substituting the above formula into Equation 6 and use the value of s we have

$$\begin{aligned} T(N) &\approx 5(\tfrac{7}{4})^rt_{comp}N^2 + 7^r2t_{comp}(\tfrac{N}{2^r})^3 \\ &= (\tfrac{7}{8})^r2t_{comp}N^3 + 5(\tfrac{7}{4})^rt_{comp}N^2. \end{aligned}$$

From Equation 5 we have

$$\begin{aligned} T_{comp} &\approx P\left((\tfrac{7}{8})^r2t_{comp}N^3 + 5(\tfrac{7}{4})^rt_{comp}N^2\right) \\ &= (\tfrac{7}{8})^r\tfrac{2t_{comp}}{P^2}M^3 + \tfrac{5(\frac{7}{4})^rt_{comp}}{P}M^2. \end{aligned}$$

The total running time is then expressed by

$$\begin{aligned} T_{total} &= T_{comp} + T_{roll} \\ &= (\tfrac{7}{8})^r\tfrac{2t_{comp}}{P^2}M^3 \\ &\quad + \tfrac{5(\frac{7}{4})^rt_{comp}}{P}M^2 + \tfrac{2B\beta}{P}M^2 + 2P\alpha. \end{aligned} \quad (7)$$

To use the BMR method between processors, additional space of size M^2 is needed. To use the S-method within individual processors, additional space of size $\frac{2}{3}M^2$ is needed. Therefore the total additional space requirement for the BMR-Strassen method is $\frac{5}{3}M^2$, which is larger than M^2 required by the BMR method.

Strassen-BMR Method. The motivation for the Strassen-BMR method comes from the observation that the S-method is most efficient for large matrices and therefore should be used at the top level (between processors) instead of the bottom level (within one processor). The 7 submatrix multiplications of the S-method at each recursion seems at first to lead to a task parallelism. The difficulty in implementing a task parallelism of the S-method on a distributed memory computer results from the fact that the matrices must be distributed among the processors. Submatrices in the S-method must be stored in different processors and if tasks are spawned these submatrices must be copied or moved to the appropriate processors[2].

For a distributed memory parallel algorithm the storage map of submatrices to processors is a primary concern. If the submatrices used in the S-method are stored among processors in the same pattern at each level of recursion, then they can be added or multiplied together just as if they are stored within one processor. Here we introduce a new pattern to store the matrices. Figure 1 and 2 show the pattern of storing matrix A with 6×6 blocks when the recursion level is 1. Figure 1 is from a matrix point-of-view. Note that the four submatrices with 3×3 blocks are stored among the 9 processors in the same pattern. Figure 2 is from a processor point-of-view. Each processor stores one block of the four submatrices. Figure 3 and 4 show the pattern when the recursion level is 2. The four submatrices with 6×6 blocks are stored in the same pattern, as well as the 16 submatrices with 3×3 blocks. This pattern can be easily replicated for higher levels of recursion.

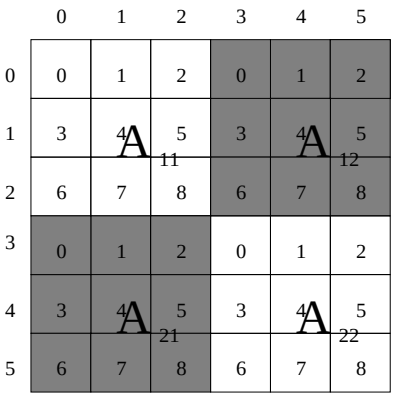

Figure 1: Matrix A with 6×6 blocks is distributed over a 3×3 processor template from a matrix point-of-view. The 9 processors are labeled from 0 to 8. This pattern is used in the Strassen-BMR method when the recursion level is 1.

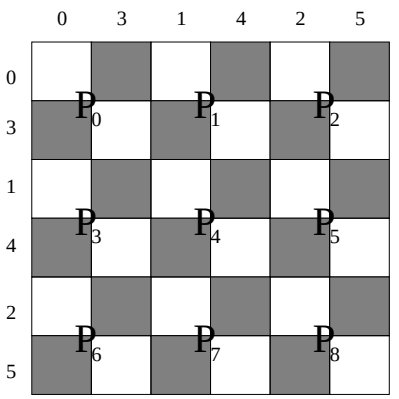

Figure 2: Same as Figure 1, but from a processor point-of-view.

These patterns of storing matrices make it possible for all the processors act like one processor. Each processor has a portion of each submatrix at each recursion level. The addition

$N = 2n + 1$, then the matrices A and B can be partitioned as

$$A = \begin{pmatrix} A_{2m\times 2k} & A_{2m\times 1} \\ A_{1\times 2k} & A_{1\times 1} \end{pmatrix}, B = \begin{pmatrix} B_{2k\times 2n} & B_{2k\times 1} \\ B_{1\times 2n} & B_{1\times 1} \end{pmatrix},$$

where the subscripts stand for dimensions of the submatrices. If $M = 2m+1$, $K = 2k+1$, and $N = 2n$, then matrices A and B can be partitioned as

$$A = \begin{pmatrix} A_{2m\times 2k} & A_{2m\times 1} \\ A_{1\times 2k} & A_{1\times 1} \end{pmatrix}, B = \begin{pmatrix} B_{2k\times 2n} \\ B_{1\times 2n} \end{pmatrix}.$$

If $M = 2m + 1$, $K = 2k$, and $N = 2n$, then matrices A and B can be partitioned as

$$A = \begin{pmatrix} A_{2m\times 2k} \\ A_{1\times 2k} \end{pmatrix}, B = \begin{pmatrix} B_{2k\times 2n} \end{pmatrix}.$$

If the evenness and oddness of M, K, and N are of some other patterns, similar partitions can be used. In all these cases, the product matrix C can be obtained by multiplying the submatrices of A and B using the T-method. The majority of the computation is in the submatrix multiplication $A_{2m\times 2k}B_{2k\times 2n}$, which can be done using the S-method since all the dimensions are even.

Fox's BMR Method. Fox's BMR method[6] is a commonly used parallel matrix multiply algorithm based on the T-method. It can be used on any rectangular processor templates and on matrices of any dimensions[1]. For simplicity of discussion, we only consider square processor templates and square matrices. Suppose we have P^2 processors logically organized in a $P \times P$ mesh. The processor in ith row and jth column has coordinates (i, j), where $0 \le i, j \le P-1$. Let matrices A, B, and C be of size $M \times M$. For simplicity of discussion we assume M is divisible by P. Let $N = M/P$. All matrices are partitioned into $P \times P$ blocks of $N \times N$ submatrices. The block with coordinates (i, j) is stored in the corresponding processor with the same coordinates. With the addition of a link between processors on opposite sides of the mesh(a torus interconnection), the mesh can be thought of as composed of rings of processors both in the horizontal and vertical directions. The BMR method requires communication between the processors of each ring in the mesh. The blocks of the matrix A are passed in parallel to the left along the horizontal rings. The blocks of the matrix B are passed to the top along the vertical rings. This communication pattern results in the rolling leftwards of matrix A and upwards of matrix B. Let A_{ij}, B_{ij}, C_{ij} stand for the blocks of A, B, C respectively stored in the processor with coordinates (i, j). The following pseudocode describes the BMR algorithm.

```
C_ij = A_ij B_ij for all processors (i, j)
DO I = 1, P - 1
   Roll A leftwards and B upwards
      (i.e., A_ij ← A_i,j+1 ; B_ij ← B_i+1,j )
   C_ij = C_ij + A_ij B_ij for all processors
ENDDO
Roll A leftwards, B upwards
```

The running time of the BMR method consists of two parts: the communication time T_{roll} and the computation time T_{comp}. On the Intel iPSC/860 Hypercube, the communication time for a single message is [5]

$$T = \alpha + \beta n,$$

where α is the latency, β is the byte-transfer rate, and n is the number of bytes in the message. In the BMR method, both matrices A and B are rolled P times. There are a total of $2P$ rolls. The total latency is $2P\alpha$. In each roll a submatrix of order $(M/P)\times(M/P)$ is passed from one processor to another, where M is the dimension of the matrices. Therefore the total byte transfer time is $2P\beta B(M/P)^2$, where B is the number of bytes used to store one entry of the matrices. The total communication time is

$$T_{roll} = 2P\alpha + \frac{2B\beta}{P}M^2. \tag{2}$$

The computation time is

$$T_{comp} = \frac{2t_{comp}}{P^2}M^3, \tag{3}$$

where t_{comp} is the execution time for one arithmetic operation. Here we assume that floating point addition and multiplication has the same speed. The total running time is

$$T(M) = \frac{2t_{comp}}{P^2}M^3 + \frac{2B\beta}{P}M^2 + 2P\alpha. \tag{4}$$

In order to make the BMR method work, an additional working space of size M^2 is needed to temporarily store the products of the submatrices of A and B.

BMR-Strassen Method. A natural approach to parallelize the S-method is to use the BMR method between processors and the serial S-method within one processor. Here this approach is called the BMR-Strassen method. Again for simplicity, we assume that we have a $P \times P$ processor template and matrices A, B, C are of order M. Suppose the recursion level in the S-method is r. Let $N = M/P$, $M_0 = M/2^r$, and $N_0 = M_0/P$. Assume N, M_0, and N_0 are all integers. The running time of the BMR-Strassen method consists of two parts: the communication time T_{roll} and the computation time T_{comp}. T_{roll} is the same as in the BMR method since the same communication method is used. Let $T(N)$ be the running time for one processor to do a submatrix multiplication of order N using the S-method. Since there are P submatrix multiplications of order N performed in one processor, we have

$$T_{comp} \approx PT(N). \tag{5}$$

The running time for the $P - 1$ submatrix additions can be ignored because matrix addition is much faster than matrix multiplication. Note that there are 15 submatrix additions and subtractions and 7 submatrix multiplications in the S-method. Therefore

$$T(N) = 15T_{add}(\frac{N}{2}) + 7T(\frac{N}{2}),$$

where $T_{add}(N/2)$ is the running time to add or subtract submatrices of order $N/2$. Note that $(N/2)^2$ operations are

A SCALABLE PARALLEL STRASSEN'S MATRIX MULTIPLY ALGORITHM FOR DISTRIBUTED MEMORY COMPUTERS

Qingshan Luo

The University of the South

John B. Drake

Oak Ridge National Laboratory

Abstract. We present a scalable parallel Strassen's matrix multiply algorithm for distributed memory, message passing computers. Strassen's algorithm to multiply two $N \times N$ matrices reduces the asymptotic operation count from $O(N^3)$ of the traditional algorithm to $O(N^{2.81})$. In a sequential implementation the Strassen's algorithm offers better performance even for relatively low order matrices. However, due to its complexity, the parallel Strassen's algorithm is less than straight forward. Here a scalable parallel Strassen's algorithm is presented and compared with several other parallel algorithms. Performances of these algorithms are tested on a 128-processor Intel iPSC/860.

Key words. Strassen's algorithm, matrix multiplication, parallel algorithms.

Introduction. Matrix multiplication(MM) is a basic linear algebra operation and is the dominant computational part of many scientific applications such as solving linear equations, matrix inversion, evaluation of eigenvalues, and evaluation of determinants. Its importance has increased in recent years with the availability of cache based processors which rival the performance of the traditional vector processor. Blocked matrix operations perform well on a cache based processor and many of the linear algebra algorithms have been recast in a block form to improve performance. The BLAS (Basic Linear Algebra Subroutines) library includes these block matrix operations as part of level 3 BLAS routines[1]. As a result, methods to speed up matrix multiplication have been studied intensively. The computational complexity of MM for $N \times N$ matrices has dropped from $O(N^3)$ of traditional method(hereafter referred as T-method) to $O(N^{2.81})$ of Strassen's method(hereafter referred as S-method) [8], and to $O(N^{2.376})$ of Coppersmith-Winograd method[3]. However, only the S-method offers better performance than the T-method for matrices of practical sizes, say, less than 10^{20} [7].

There have been mainly two approaches to parallelize the S-method. The first approach is to use the T-method at the top level(between processors) and the S-method at the bottom level(within one processor)[4]. The most commonly used T-method between processors is Fox's Broadcast-Multiply-Roll(BMR) method[6]. This approach is easily scalable, but it has low parallel efficiency [2]. Since the S-method is most efficient for large matrices, it is well suited to use at the top level, not the bottom level. The second approach is to use the S-method at both the top and the bottom level. This approach has been tried on Intel Paragon and the S-method gives much better performance than the T-method[2]. However, the S-method in [2] requires that the number of processors used in the computation to be a power of seven. This is a severe restriction since many MIMD computers use a hypercube or a mesh architecture and powers of seven numbers of processors are not a natural grouping. Therefore, the S-method in [2] is not scalable. Moreover, the S-method in [2] requires a large working space, with each matrix to be multiplied duplicated 3 or 4 times. For these reasons we explored the possibility of other parallel algorithms with more practical potential. In this paper, we introduce an algorithm which uses the S-method at the top level and the T-method at the bottom level.

Strassen's Method. The S-method does not require square matrices. Suppose matrix A is $M \times K$ and matrix B is $K \times N$. Then the product matrix $C = AB$ is $M \times N$. If M, K, and N are all even, then then three matrices are partitioned as

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}, \quad B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix},$$
$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}.$$

where all the submatrices evenly partition matrices A, B, and C. The computation of the product matrix C proceeds through the following 7 submatrix multiplications and 15 submatrix additions and subtractions.

$$\begin{array}{lll} S_1 = A_{21} + A_{22} & M_1 = S_2 S_6 & T_1 = M_1 + M_3 \\ S_2 = S_1 - A_{11} & M_2 = A_{11} B_{11} & T_2 = T_1 + M_4 \\ S_3 = A_{21} - A_{12} & M_3 = A_{12} B_{21} & T_3 = M_5 + M_6 \\ S_4 = A_{12} - S_2 & M_4 = S_3 S_7 & \\ S_5 = B_{12} - B_{11} & M_5 = S_1 S_5 & C_{11} = M_2 + M_3 \\ S_6 = B_{22} - S_5 & M_6 = S_4 B_{22} & C_{12} = T_1 + T_3 \\ S_7 = B_{22} - B_{12} & M_7 = A_{22} S_8 & C_{21} = T_2 - M_7 \\ S_8 = S_6 - B_{12}, & & C_{22} = T_2 + M_5 \end{array} \quad (1)$$

The S-method does the above computation recursively until one of the dimensions of the matrices is less than some number $MINDIM$. Then the T-method is used for submatrix multiplications. If carefully coded, this algorithm needs approximately $(\frac{2}{3} M \max\{K, N\})$ additional working space [4].

The above algorithm requires that M, K, and N are all even. If they are all odd, i.e., $M = 2m + 1$, $K = 2k + 1$, and