A three-dimensional approach to parallel matrix multiplication

by R. C. Agarwal
S. M. Balle
F. G. Gustavson
M. Joshi
P. Palkar

A three-dimensional (3D) matrix multiplication algorithm for massively parallel processing systems is presented. The *P* processors are configured as a "virtual" processing cube with dimensions p_1, p_2, and p_3 proportional to the matrices' dimensions—*M*, *N*, and *K*. Each processor performs a single local matrix multiplication of size $M/p_1 \times N/p_2 \times K/p_3$. Before the local computation can be carried out, each subcube must receive a single submatrix of *A* and *B*. After the single matrix multiplication has completed, K/p_3 submatrices of this product must be sent to their respective destination processors and then summed together with the resulting matrix *C*. The 3D parallel matrix multiplication approach has a factor of $P^{1/6}$ less communication than the 2D parallel algorithms. This algorithm has been implemented on IBM POWERparallel™ SP2™ systems (up to 216 nodes) and has yielded close to the peak performance of the machine. The algorithm has been combined with Winograd's variant of Strassen's algorithm to achieve performance which exceeds the theoretical peak of the system. (We assume the MFLOPS rate of matrix multiplication to be 2*MNK*.)

1. Introduction

A parallel high-performance matrix multiplication P_GEMM[1] algorithm based on a three-dimensional approach is presented. For the parallel case, the algorithm is a natural generalization of the serial _GEMM routine. _GEMM computes $\mathbf{C} = \beta\mathbf{C} + \alpha\text{op}(\mathbf{A})\text{op}(\mathbf{B})$ where α, β are scalars, **A**, **B**, and **C** are matrices, and op(**X**) stands for **X**, $\mathbf{X}^T$, or $\mathbf{X}^C$. (Superior *T* indicates transpose, and superior *C* conjugate transpose.) The algorithm described has been implemented in both the double-precision and complex double-precision IEEE format, as well as for all combinations of matrix products involving matrices in their normal form, their transposed form, and their conjugates. For all of these data combinations, performance was the same.

Most parallel matrix multiplication algorithms used as building blocks in scientific applications are 2D algorithms. The primary issue is that the 3D algorithm moves a factor of $P^{1/6}$ less data than the known 2D algorithms. From this standpoint, the 3D algorithms appear to be a better choice than 2D algorithms. We show, in Section 3, that the 3D algorithm yields better performance than the 2D ScaLAPACK PDGEMM algorithm [2].

The literature describing matrix multiplication algorithms is very extensive. Some descriptions are given by Demmel,

[1] The _ symbol stands for S, D, C, and Z [1, 2]; i.e., single, double, complex single, and complex double (Z) precision.

0018-8646/95/$3.00 © 1995 IBM

Heath, and van der Vorst [3], by Choi, Dongarra, and Walker [4], by Huss-Lederman, Jacobson, and Tsao [5], by Agarwal, Gustavson, and Zubair [6], and by van de Geijn and Watts [7]. Aggarwal, Chandra, and Snir [8] show that a 3D-type algorithm is optimal for an LPRAM. Johnsson and Ho [9] and Ho, Johnsson, and Edelman [10] discuss 3D and other types of algorithms for Boolean cubes and hypercubes. Gupta and Kumar [11] discuss the scalability of many parallel matrix multiplication algorithms, including 2D as well as 3D versions. Like other authors, they demonstrate that the communication ratio of 3D over 2D is $P^{1/6}$. For distributed memory message-passing computers, our algorithm has the least amount of communication of all the 3D algorithms cited. It reduces the amount of communication required by the other 3D algorithms by a factor of 5/3 [11]. Lemmerling, Vanhamme, and Ho[2] describe several 1D, 2D, and some new 3D parallel algorithms. To the best of our knowledge, prior work has not addressed the problem of minimizing communication for matrices of arbitrary shape. In this paper, we provide a solution which minimizes communication for such matrices.

Our 3D algorithm can be combined in a straightforward manner with the $O(n^{2.81})$ matrix multiplication scheme developed by Strassen, thereby allowing it to take full advantage of the latter's high efficiency [1]. It is also possible to use Strassen's algorithm on the global matrices down to a level where the matrices fit into the local memory of the node, as described by Agarwal et al.[3] Bailey [12], Grayson, Shah, and van de Geijn [13], Balle [14, Section 2] and Douglas et al. [15] describe 2D implementations of Strassen's method.

In Section 2, we outline the 3D algorithm and its Strassen variation. Section 3 also demonstrates that the 3D approach yields very high performance on the IBM POWERparallel™ SP2™ system. Section 4 presents concluding remarks.

2. A 3D parallel P_GEMM algorithm

A matrix multiplication of size (M, N, K) requires MNK multiply-adds. This can be represented by a rectangular parallelepiped of size (M, N, K) in the computing space. To achieve computational load balance using $P = p_1p_2p_3$ processors, each processor must compute $1/P$th of this computational rectangular parallelepiped. Thus, the volume of the computational space assigned to each processor is fixed at MNK/P. This guarantees computational load balance if each such processor performs an identical computation of size MNK/P. In addition, to minimize communication, each processor must do this much computation with a minimum amount of data movement (communication). Assuming that each processor does a subcube (of size $m = n = k$) of the computation, the three faces of the subcube (corresponding to equal square submatrices of **A**, **B**, and **C**) represent a data movement of size $3Pm^2$, since these submatrices must be brought/sent to these P subcubes in order to perform the P DGEMM computations. We note that data movement of m^2 numbers is proportional to the area of a square of size m. Hence, our problem of minimal data communication can be viewed as the classical problem of minimizing surface area for a given volume. The optimal solution of this problem is that each of the P rectangular parallelepipeds must be a subcube of identical sides. This fact establishes a lower bound on the amount of communication necessary to perform this parallel multiplication; namely, $3Pm^2$. Assuming a three-dimensional processing grid of size (p_1, p_2, p_3), the subparallelepiped computed at each processor is of size $(M/p_1, N/p_2, K/p_3)$. To minimize communication, the following relationship must be true: $M/p_1 = m = N/p_2 = n = K/p_3 = k$. Finally, we note that when this relationship holds, the algorithm presented in this paper achieves this lower bound; i.e., the total amount of data moved for **A**, **B**, and **C** is Pm^2.

For simplicity, we consider a $2 \times 2 \times 2$ processing cube. (This example is consistent with a description of the general case; i.e., no information that would be given by such a description is altered or omitted.) The underlying idea can be described in terms of block matrices for a single 2×2 block partitioning of the matrices **A**, **B**, and **C**. Let

$$\mathbf{A} = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix}, \tag{1}$$

where **A** is an M by K matrix, **B** is a K by N matrix, and **C** is an M by N matrix. If we let $\beta = 0.0$ and $\alpha = 1.0$, we get $\mathbf{C} = AB$; i.e.,

$$\mathbf{C} = \begin{pmatrix} A_{00}B_{00} + A_{01}B_{10} & A_{00}B_{01} + A_{01}B_{11} \\ A_{10}B_{00} + A_{11}B_{10} & A_{10}B_{01} + A_{11}B_{11} \end{pmatrix}. \tag{2}$$

Now the block matrices A_{il} and B_{lj} both have the same order. Thus, all $P = 2^3$ products $A_{il}B_{lj}$ consist of an identical computation:

$$\text{Processor } (i, j, l) \text{ computes } A_{il}B_{lj}, \qquad 0 \le i, j, l < 2. \tag{3}$$

For large K, almost all of the computation cost in Equation (2) is consumed by the P products in Equation (3). This is the so-called volume-to-surface effect of matrix multiplication; for $M = N = K$ we have that matrix multiplication performs $2N^3$ FLOPS and matrix addition performs N^2 FLOPS. The computations in Equation (3)

[2] P. Lemmerling, L. Vanhamme, and C.-T. Ho, "On Matrix Multiplication Algorithms Using Message Passing Interface (MPI)," IBM Almaden Research Laboratory, San Jose, CA, 1995, unpublished note.

[3] R. C. Agarwal, S. M. Balle, F. G. Gustavson, and M. Joshi, "A Three-Dimensional Approach to Parallel Matrix Multiplication," Technical Report, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1995; in preparation.

are perfectly load-balanced. It follows that most of the computation in Equation (2) is done at 100% efficiency. The essence of the underlying idea is implicit in Equations (2) and (3): Form the matrices A_{il} and B_{lj} from the input data and place them on processor (i, j, l). Compute $A_{il}B_{lj}$ in parallel, thereby getting 100% efficiency most of the time the algorithm is computing. Finally, perform the matrix additions of Equation (2).

The communication part of the algorithm is done by simultaneously making calls to the MPI collective communication primitives *all-gather* and *all-to-all* [16, 17]. For the performance studies presented in Section 3, we used the equivalent MPL (Message Passing Library) primitives mp_concat and mp_index, respectively [18]. In the following, we define P to be the total number of processors, and p_1, p_2, and p_3 to be the number of processors in the d_1, d_2, and d_3 directions, respectively—thereby having $P = p_1p_2p_3$. The indices i, j, and l are used to identify the processors in the d_1, d_2, and d_3 directions, respectively: $0 \le i < p_1$, $0 \le j < p_2$, and $0 \le l < p_3$.

To describe the 3D matrix multiplication algorithm, we define the following variables: $(m, n, k) = (M/p_1, N/p_2, K/p_3)$, $k_2 = k/p_2$, $n_1 = n/p_1$, and $n_3 = n/p_3$. We use the colon notation [19] to describe submatrices of the global matrices **A**, **B**, and **C**. Thus, $\mathbf{A} = A(:, :) = A(0 : M - 1, 0 : K - 1)$. The indices (i, j, l) are also used as subscripts to identify submatrices of **A**, **B**, and **C**. We define

$$A_{il} = A(im : im + m - 1, lk : lk + k - 1), \quad (4)$$

$$B_{lj} = B(lk : lk + k - 1, jn : jn + n - 1), \quad (5)$$

$$C_{ij} = C(im : im + m - 1, jn : jn + n - 1), \quad (6)$$

with $0 \le i < p_1$, $0 \le j < p_2$, and $0 \le l < p_3$.

We choose to have the matrix **A** associated with the d_1–d_3 plane, with d_2 being the orthogonal dimension, as illustrated in **Figure 1**. The matrix **B** is similarly laid out in the d_2–d_3 plane, having d_1 as its orthogonal dimension. The d_1–d_2 plane holds the output matrix **C**, thereby making d_3 its orthogonal dimension. We must define certain submatrices of the submatrices A_{il}, B_{lj}, and C_{ij}. We consider the submatrix A_{il} and partition its k columns into p_2 sets, each of size k_2, of contiguous columns. We use the notation $A_{il}(j)$, $0 \le j < p_2$, to denote the submatrix of A_{il} that consists of the jth set of contiguous columns of A_{il}. Similarly, we need $B_{lj}(i)$, $0 \le i < p_1$, and $C_{ij}(l)$, $0 \le l < p_3$. This is a 3D block distribution, where the (rows, columns) of **A** are distributed on a (p_1, p_2p_3) grid (Figure 1) and similarly for the other matrices **B** and **C**. In particular, all matrices are equidistributed. These submatrices of submatrices are easily defined in terms of the colon notation:

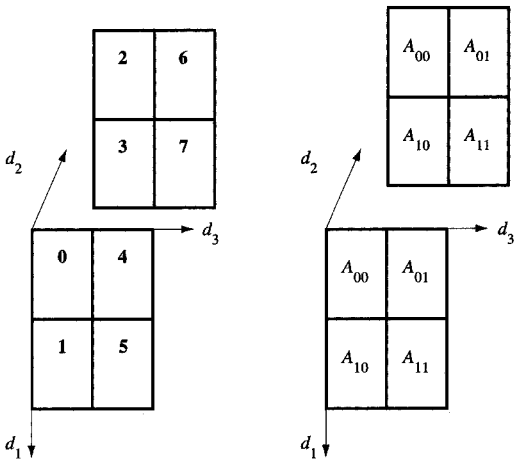

Figure 1

Layout of the global matrix **A** on a three-dimensional processor grid of dimension ($p_1 = 2$, $p_2 = 2$, $p_3 = 2$) after Step 2 of the algorithm. The boldface numbers **0–7** indicate the processor labels.

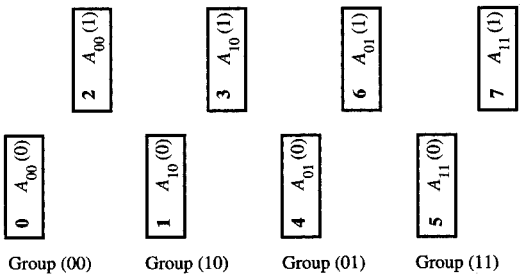

Figure 2

Input matrices $A_{il}(j)$ distributed across p_1p_3 groups G_{il}, each of size p_2. The boldface numbers **0–7** indicate the processor labels.

$$A_{il}(j) = A_{il}(:, jk_2 : jk_2 + k_2 - 1), \quad (7)$$

$$B_{lj}(i) = B_{lj}(:, in_1 : in_1 + n_1 - 1), \quad (8)$$

and

$$C_{ij}(l) = C_{ij}(:, ln_3 : ln_3 + n_3 - 1). \quad (9)$$

Let G_{il} be the group of processors j on which the matrices $A_{il}(j)$ reside, $0 \le j < p_2$ (**Figure 2**). G_{jl} and G_{ij} are similarly defined for the group of processors associated

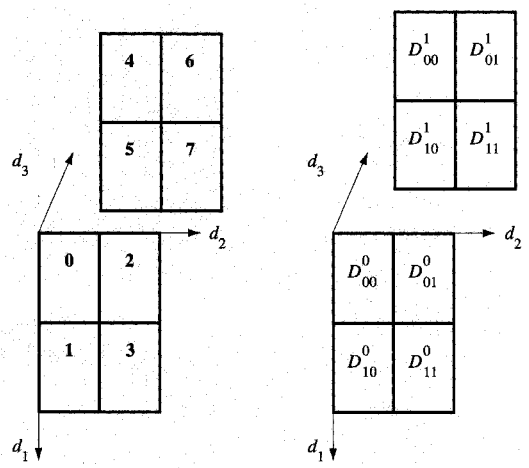

Figure 3

Layout of **D** on a three-dimensional processor grid of dimension ($p_1 = 2, p_2 = 2, p_3 = 2$). The boldface numbers **0–7** indicate the processor labels.

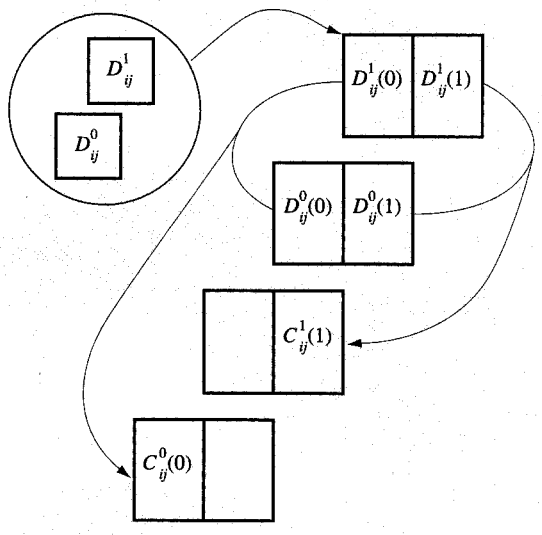

Figure 4

Generic picture of the matrices $D^l_{ij}(r)$ associated with the processor group G_{ij} consisting of the four matrices $D^0_{ij}(0{:}1)$ and $D^1_{ij}(0{:}1)$. The *all-to-all* gather of Step 5 places $D^0_{ij}(0)$ and $D^1_{ij}(0)$ on processor $(i, j, 0)$ and $D^0_{ij}(1)$ and $D^1_{ij}(1)$ on processor $(i, j, 1)$.

with B_{lj} and C_{ij}. The 3D algorithm features a single matrix multiplication, $A_{il}B_{lj}$, on every processor (i, j, l). We define an auxiliary m by n matrix D^l_{ij} to denote this product (**Figure 3**):

$$D^l_{ij} = A_{il}B_{lj}. \tag{10}$$

Like the p_3 submatrices $C_{ij}(l)$ of C_{ij}, we need to define p_3 submatrices of D^l_{ij} (**Figure 4**), each consisting of a contiguous block of n_3 columns of D^l_{ij}:

$$D^l_{ij}(r) = (A_{il}B_{lj})(:, rn_3 : rn_3 + n_3 - 1), \tag{11}$$

where $0 \le r < p_3$.

We are now in position to define the algorithm. The input matrices that reside on processor (i, j, l) are the matrices $A_{il}(j)$, $B_{lj}(i)$, and $C_{ij}(l)$ given by Equations (7), (8), and (9), respectively.

Algorithm 1: 3D parallel P_GEMM algorithm

1. i. Define p_1p_3 groups of processes G_{il} ($0 \le i < p_1$ and $0 \le l < p_3$) [16], each of size p_2, to handle the communication involving the global matrix **A** (Figure 2).
   ii. Define p_2p_3 groups of processes G_{lj} ($0 \le j < p_2$ and $0 \le l < p_3$), each of size p_1, to handle the communication involving the global matrix **B**.
   iii. Define p_1p_2 groups of processes G_{ij} ($0 \le i < p_1$ and $0 \le j < p_2$), each of size p_3, to handle the communication involving the global matrix **C**.
2. Simultaneously, for every group G_{il} defined in Step 1.i, using the input matrices $A_{il}(j)$, $(0 \le j < p_2)$, perform an *all-gather* [16, Section 4.5]. Each process (i, j, l) of G_{il} receives the same submatrix A_{il} [Equation (4)].
3. Similarly, simultaneously, for every group G_{lj} defined in Step 1.ii, using the input matrices $B_{lj}(i)$, $(0 \le i < p_1)$, perform an *all-gather* [16, Section 4.5]. Each process (i, j, l) of G_{lj} receives the same submatrix of B_{lj} [Equation (5)].
4. Perform a single local matrix–matrix product $D^l_{ij} = A_{il}B_{lj}$ on all P processes, as described by Equation (10).
5. Simultaneously, for every group G_{ij} defined in Step 1.iii, using the input matrices $D^l_{ij}(r)$ [Equation (11)], perform an *all-to-all* [16, Section 4.8]. Each process (i, j, l) of G_{ij} $(0 \le l < p_3)$ receives p_3 submatrices:
$$D^r_{ij}(l) = (A_{ir}B_{rj})(:, ln_3 : ln_3 + n_3 - 1). \tag{12}$$
6. On every process, compute
$$C_{ij}(l) = \beta C_{ij}(l) + \alpha \sum_{r=0}^{p_3-1} D^r_{ij}(l).$$

Combining Strassen's algorithm with the 3D P_GEMM algorithm

A straightforward variation of the 3D algorithm allows the use of an $O(n^{2.81})$ matrix multiplication algorithm devised by Strassen [20]. Our approach is to use the Winograd variant of Strassen's algorithm to perform the local computation instead of using _GEMM. In Step 4, we replace the single call to _GEMM with a call to _GEMMS [1].

3. Performance results

Performance results for the parallel 3D matrix multiplication are presented. These experiments were carried out on IBM POWERparallel SP2 systems [21, 22]. MPL message-passing subroutines are used as communication primitives [18].

Figures 5 and **6** show performance for the 3D parallel matrix multiplication of the matrix $\mathbf{C} = C + AB$ for PDGEMM and PZGEMM on SP2 Thin2 nodes. All timings were recorded using the wall clock and hence include the cost of communication and computation. For each experiment we report either the wall clock time or the "nominal MFLOP rate" per processor, or both. Figures 5 and 6 illustrate that even for relatively small matrices and/or a large number of processors, this approach yields very high performance.

Table 1 shows representative MFLOPS rates per processor for the cases $\mathbf{C} = C + AB$, $\mathbf{C} = C + A^TB$, $\mathbf{C} = C + AB^T$, and $\mathbf{C} = C + A^TB^T$ for real matrices. Similar results were obtained for other matrix sizes and different numbers of processors.

The MFLOPS rates presented in **Table 2** for the Winograd variant of the Strassen algorithm are "nominal rates computed by dividing $2n^3$ (the number of operations that would be executed by the conventional algorithm) by the actual compute time. This permits us to illustrate the improvements achieved by using Strassen's algorithm. In the complex case, there is an additional advantage, since it is possible to multiply two complex matrices together using three real matrix multiplications and five real matrix additions instead of four real matrix multiplications and two real matrix additions [1].

In **Table 3**, we compare the 2D ScaLAPACK PDGEMM algorithm, as implemented in PESSL [2], with the 3D algorithm for $P = 32$ processors. The PESSL numbers are preliminary numbers. Unfortunately we were not able to obtain a full set of performance numbers for all configurations for a large number of processors. The 3D algorithm shows relatively better performance for small matrices and more uniform performance for different values of the TRANS (type) parameter.

4. Conclusion

We have shown that our 3D approach to parallel matrix multiplication yields very high performance on

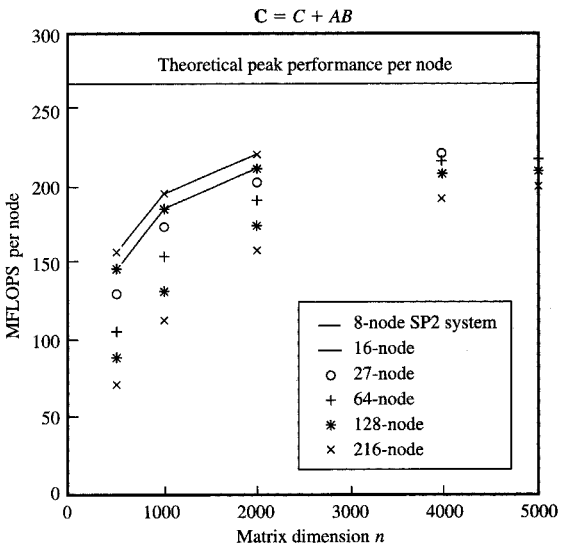

Figure 5

Performance results for the 3D parallel double-precision PDGEMM when using DGEMM [1] for the local call. The global input matrices are square.

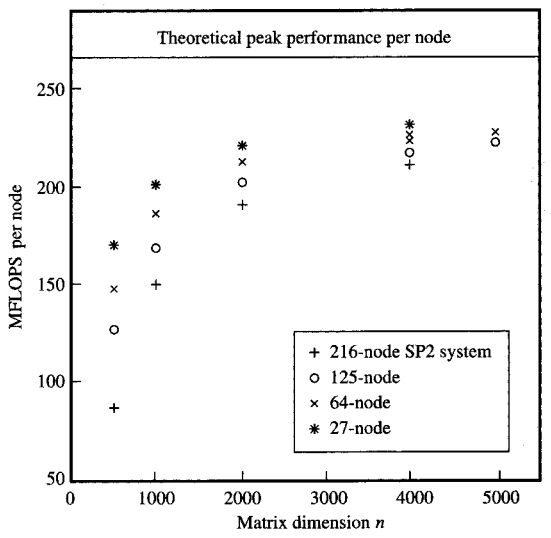

Figure 6

Performance results for the 3D parallel double-precision complex PZGEMM when using ZGEMM [1] for the local call. The global input matrices are square.

Table 1 MFLOPS rate per processor for the four cases for the double-precision IEEE format.

n	Number of nodes	$C = C + AB$	$C = C + A^TB$	$C = C + AB^T$	$C = C + A^TB^T$
1000	8	196	197	193	194
1000	16	186	186	184	185
5000	128	244	244	242	242

Table 2 Performance results for the 3D matrix multiplication algorithm when using DGEMMS [1] for the local call. The matrices to be multiplied are square of dimension 5000.

Number of SP2 nodes	Double precision*		Double-precision complex*	
	MFLOPS per node	Total MFLOPS	MFLOPS per node	Total MFLOPS
27	272	7337	304	8198
64	271	17344	369	23616
125	243	30320	329	41094
216	223	48907	361	77976

*IEEE format.

Table 3 Performance results for the 3D matrix multiplication algorithm and the PESSL PDGEMM [2] on a 32-Thin2-node SP2 system.

Size	Configuration		Type	Time		MFLOPS per node		PESSL/3D ratio
	PESSL	3D		PESSL	3D	PESSL	3D	
500	8,4	4,4,2	(n,n)	0.1140	0.0606	2192 (68)	4122 (129)	1.89
500	8,4	4,2,4	(t,n)	0.1414	0.0611	1768 (55)	4088 (128)	2.33
500	4,8	4,4,2	(n,t)	0.1361	0.0626	1837 (57)	3994 (125)	2.19
500	2,16	4,4,2	(t,t)	0.2953	0.0625	846 (26)	3997 (125)	4.81
1000	8,4	4,4,2	(n,n)	0.6005	0.3663	3330 (104)	5460 (171)	1.64
1000	8,4	4,4,2	(t,n)	0.6816	0.3637	2934 (92)	5499 (172)	1.87
1000	4,8	4,4,2	(n,t)	0.6848	0.3692	2920 (91)	5417 (169)	1.86
1000	2,16	4,4,2	(t,t)	1.0747	0.3691	1860 (58)	5417 (169)	2.91
2000	8,4	4,4,2	(n,n)	3.3710	2.4427	4746 (148)	6550 (205)	1.39
2000	8,4	4,4,2	(t,n)	3.8243	2.4574	4183 (131)	6511 (203)	1.55
2000	4,8	4,4,2	(n,t)	3.6743	2.4728	4354 (136)	6470 (202)	1.49
2000	2,16	4,4,2	(t,t)	5.3623	2.4716	2983 (93)	6473 (202)	2.17

massively parallel processing systems such as the IBM POWERparallel SP2 system. Our algorithm is perfectly load-balanced for both communication and computation. We have introduced a new scheme for partitioning matrices across processors on distributed memory computers that allows multiple use of the MPI collective communication primitives *all-gather* and *all-to-all*. Additionally, this choice of data distribution reduces the amount of communication from that required by the other 3D algorithms by a factor of 5/3. Our 3D algorithm not only results in less communication but also produces better node performance, as the submatrices multiplied at each node are larger and have a better aspect ratio. This is evidenced by the fact that most 2D algorithms perform $P^{1/2}$ local matrix multiplications of size $N/P^{1/2}$, while our 3D algorithm performs only one local matrix multiplication of size $N/P^{1/3}$. Our performance results for small matrices also emphasize this result. Another important result is that the Winograd variant of Strassen's algorithm can be incorporated in this algorithm in a straightforward manner to yield extremely high performance.

The amount of communication required to reshuffle the data from 2D to 3D is proportional to the sum of the sizes of the matrices **A**, **B**, and **C**. The 3D algorithm moves a factor $P^{1/6}$ less data than the 2D algorithms, which move a total amount of data equal to $P^{1/2}$ times the sum of the sizes of the **A** and **B** matrices. This means that even when the extra communication cost of reshuffling back and forth

between 2D and 3D is added to the total communication cost of the 3D algorithm, it still has less total communication cost than the 2D algorithms. Further investigations are still needed with respect to the reshuffling of data between the two data distributions. We are interested in 2D block and block cyclic layouts as well as in only rearranging submatrices of the global matrices **A**, **B**, and **C**.

The new scheme for partitioning matrices across processors presented in conjunction with the 3D matrix multiplication algorithm is applicable to most of the level-3 BLAS. Gustavson has shown that 26 of the 30 level-3 BLAS can be expressed in terms of this 3D distribution. This work is still ongoing research.

Instead of applying Strassen's algorithm at the local level, it can be used at the global level. This approach is of interest when the matrices to be multiplied are too big to fit into local memory. The variant of the 3D algorithm using the Strassen algorithm at a global level is on our list of future work.

Acknowledgments

We thank V. Kumar from the University of Minnesota and M. Zubair for their initial ideas regarding the analysis of communication in the 3D algorithm. We thank C.-T. Ho and M. Snir for discussions about their 3D algorithms and for information about the MPI and MPL communication routines. We thank A. Ho for discussions regarding the implementation of the MPI primitives on the IBM POWERparallel SP2 systems.

References

1. *IBM Engineering and Scientific Subroutine Library, Guide and Reference*, 1994, Order No. SC23-0526-01; available through IBM branch offices.
2. *IBM Parallel Engineering and Scientific Subroutine Library, Guide and Reference*, April 1995, Order No. GC23-3837; available through IBM branch offices.
3. J. W. Demmel, M. T. Heath, and H. A. van der Vorst, "Parallel Numerical Linear Algebra," *Acta Numerica 1993*, Cambridge University Press, 1993, pp. 111–197.
4. J. Choi, J. J. Dongarra, and D. W. Walker, "PUMMA: Parallel Universal Matrix Multiplication Algorithms on Distributed Memory Concurrent Computers," *Concurrency: Pract. & Exper.* **6,** 543–570 (October 1994).
5. S. Huss-Lederman, E. M. Jacobson, and A. Tsao, "Comparison of Scalable Parallel Matrix Multiplication Libraries," *Proceedings of the Scalable Parallel Libraries Conference*, IEEE Computer Society Press, 1994, pp. 142–149.
6. R. C. Agarwal, F. G. Gustavson, and M. Zubair, "A High-Performance Matrix Multiplication Algorithm on a Distributed-Memory Parallel Computer, Using Overlapped Communication," *IBM J. Res. Develop.* **38,** 673–681 (1994).
7. R. van de Geijn and J. Watts, "SUMMA: Scalable Universal Matrix Multiplication Algorithm," *Technical Report TR 95-13*, Department of Computer Science, University of Texas at Austin, 1995; submitted to *Concurrency: Pract. & Exper.*
8. A. Aggarwal, A. K. Chandra, and M. Snir, "Communication Complexity of PRAMs," *Theor. Comput. Sci.* **7,** 3–71 (1990).
9. S. L. Johnsson and C.-T. Ho, "Algorithms for Multiplying Matrices of Arbitrary Shapes Using Shared Memory Primitives on Boolean Cubes," *Technical Report TR-569*, Yale University, New Haven, CT, 1987.
10. C.-T. Ho, S. L. Johnsson, and A. Edelman, "Matrix Multiplication on Hypercubes Using Full Band Bandwidth and Constant Storage," *Proceedings of the Sixth Distributed Memory Computing Conference*, IEEE Computer Society Press, 1991, pp. 447–451.
11. A. Gupta and V. Kumar, "Scalability of Parallel Algorithms for Matrix Multiplication," Technical Report, Department of Computer Science, University of Minnesota, 1991; revised April 1994.
12. D. H. Bailey, "Extra High Speed Matrix Multiplication on the Cray-2," *SIAM J. Sci. Stat. Comput.* **9,** 603–607 (1988).
13. B. Grayson, A. P. Shah, and R. van de Geijn, "A High Performance Parallel Strassen Implementation," *Technical Report TR 95-24*, Department of Computer Science, University of Texas at Austin, 1995; submitted to *Parallel Proc. Lett.*
14. S. M. Balle, "Distributed-Memory Matrix Computations," *Technical Report UNIC-95-02* (Ph.D. thesis), Danish Computing Center for Research and Education, Technical University of Denmark, Copenhagen, February 1995.
15. C. C. Douglas, M. Heroux, G. Slishman, and R. M. Smith, "GEMMW: A Portable Level 3 BLAS Winograd Variant of Strassen's Matrix–Matrix Multiply Algorithm," *J. Comput. Phys.* **110,** 1–10 (1994).
16. Message-Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, University of Tennessee at Knoxville, May 2, 1995.
17. W. Gropp, E. Lusk, and A. Skjellum, *Using MPI: Portable Parallel Programming with the Message Passing Interface*, MIT Press, Cambridge, MA, 1994.
18. *IBM AIX Parallel Environment: Parallel Programming Subroutine Reference*, 1994, Order No. SH26-7228-02; available through IBM branch offices.
19. G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed., Johns Hopkins University Press, Baltimore, MD, 1989.
20. V. Strassen, "Gaussian Elimination Is Not Optimal," *Numer. Math.* **13,** 354–356 (1969).
21. "Scalable Parallel Computing," *IBM Syst. J.* **34,** No. 2 (1995).
22. H. Franke, C. E. Wu, M. Riviere, P. Pattnaik, and M. Snir, "MPI Programming Environment for IBM SP1/SP2," *Technical Report RC-19991*, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, March 1995.

Received September 13, 1995; accepted for publication September 27, 1995

Ramesh C. Agarwal *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (AGARWAL at YKTVMV, agarwal@watson.ibm.com).* Dr. Agarwal received a B.Tech. (Hons.) degree from the Indian Institute of Technology (IIT), Bombay. He was the recipient of The President of India Gold Medal while there. He received M.S. and Ph.D. degrees from Rice University and was awarded the Sigma Xi Award for best Ph.D. thesis in electrical engineering. He has been a member of the Mathematical Sciences Department at the IBM Thomas J. Watson Research Center since 1983. Dr. Agarwal has done research in many areas of engineering, science, and mathematics and has published over 60 papers in various journals. Currently, his primary research interest is in the area of algorithms and architecture for high-performance computing on workstations and scalable parallel machines. In 1974, Dr. Agarwal received the Senior Award for best papers from the IEEE Acoustics, Speech, and Signal Processing (ASSP) group. He has received several Outstanding Achievement Awards and a Corporate Award from IBM. Dr. Agarwal is a Fellow of the IEEE and a member of the IBM Academy of Technology.

Susanne M. Balle *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (susanne@watson.ibm.com).* Dr. Balle received her Ph.D. degree in 1995 in computational mathematics from the Danish Computing Center for Research and Education and the Technical University of Denmark. She received an M.S. in mechanical engineering and computational fluid dynamics from the Technical University of Denmark and a B.S. in mechanical engineering from Odense Teknikum, Denmark. From 1992 to 1995 she consulted for the Connection Machine Scientific Software Library (CMSSL) group at Thinking Machines Corporation. During fall 1993 and spring 1994, she was a visiting scholar at the Department of Mathematics at the University of California, Berkeley. Dr. Balle is currently on a one-year postdoctoral assignment in the Mathematical Sciences Department. Her primary research interests are numerical linear algebra, numerical analysis, and parallel distributed-memory computation.

Fred G. Gustavson *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (GUSTAV at YKTVMV, gustav@watson.ibm.com).* Dr. Gustavson is manager of Algorithms and Architectures in the Mathematical Sciences Department at the IBM Thomas J. Watson Research Center. He received his B.S. in physics, and his M.S. and Ph.D. degrees in applied mathematics, all from Rensselaer Polytechnic Institute. He joined IBM Research in 1963. One of his primary interests has been in developing theory and programming techniques for exploiting the sparseness inherent in large systems of linear equations. Dr. Gustavson has worked in the areas of nonlinear differential equations, linear algebra, symbolic computation, computer-aided design of networks, design and analysis of algorithms, and programming applications. He and his group are currently engaged in activities that are aimed at exploiting the novel features of the IBM family of RISC processors. These include hardware design for divide and square root, new algorithms for POWER2 for the Engineering and Scientific Subroutine Library (ESSL) and for other math kernels, and parallel algorithms for distributed memory processors. Dr. Gustavson has received an IBM Outstanding Contribution Award, an IBM Outstanding Innovation Award, an IBM Outstanding Invention Award, two IBM Outstanding Technical Achievement Awards, two IBM Corporate Technical Recognition Awards, and a Research Division Technical Group Award.

Mahesh Joshi *IBM Research Division, Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, New York 10598 (c1majo@watson.ibm.com).* Mr. Joshi received his M. Tech. (Integ.) degree in electrical engineering from the Indian Institute of Technology, Bombay, in 1993. His research areas for master's work were intelligent control, fuzzy logic, and numerical algorithms. He visited the IBM Thomas J. Watson Research Center from December 1993 to December 1995, on assignment from Tata Information Systems Ltd. During this visit he worked in the areas of performance optimization on POWERx architectures, parallel algorithm design and implementation on IBM's SPx machines, and technical support for software development. Mr. Joshi is one of the developers of the IBM Parallel Engineering and Scientific Subroutines Library (PESSL). He will commence working toward a Ph.D. degree in computer science at the University of Minnesota in December 1995. His research areas during the course of Ph.D. work will be broadly related to high-performance parallel computing.

Prasad Palkar *Visa International, San Mateo, California 94402.* Mr. Palkar graduated from the College of Engineering, Pune, India, in 1987 with a bachelor's degree in electronics and communications. He received his master of technology degree in computer science and engineering from the Indian Institute of Technology, Bombay, in 1990. Mr. Palkar visited the IBM Thomas J. Watson Research Center in 1993 and in 1994–1995 while he was an employee of Tata Information Systems Limited. He worked on the ESSL/6000 development team to develop the initial parallel matrix multiply code. His interests include performance optimization techniques, parallelizing compilers, and operating systems. Mr. Palkar currently works at Visa International, San Mateo, California.