Yale University
Department of Computer Science

Algorithms for Multiplying Matrices of Arbitrary Shapes Using Shared Memory Primitives on Boolean Cubes

S. Lennart Johnsson and Ching-Tien Ho

YALEU/DCS/TR-569
October 1987

This work has been supported in part by the Office of Naval Research under Contracts N00014-84-K-0043 and N00014-86-K-0564. Approved for public release: distribution is unlimited.

Contents

Algorithms for Multiplying Matrices of Arbitrary Shapes Using Shared Memory Primitives on Boolean Cubes

S. Lennart Johnsson and Ching-Tien Ho
Department of Computer Science
Yale University
New Haven, CT 06520

Abstract. We investigate the multiplication of two arbitrarily shaped matrices on Boolean cube configured multiprocessors. We present algorithms in terms of generic communication primitives, which effectively allows the programmer to express the algorithms as shared memory algorithms. Some of the communication primitives we present yield communication within a factor of two of the lower bound, and have the best known routing times. For the multiplication of a $P \times Q$ matrix by a $Q \times R$ matrix three loops are required in a language like Fortran 77. One-, two- or all three loops may be parallelized. We show that with the communication primitives we use parallelizing all three loops yield a complexity that at most is equal to that of parallelizing two loops, which in turn is at most equal to that of parallelizing only one loop, for all matrix shapes. In parallelizing only one loop the processors shall be aligned with the axis (P, Q, or R) with the maximum number of elements to yield the lowest arithmetic and communication complexity. In parallelizing two loops the processors shall be assigned to the plane with the maximum number of elements. In parallelizing two or all three loops the aspect ratio of the processor array shall be the same as that of the matrix element domain. We also derive expressions for the optimal number of processors and show that for large start-up times for communication the optimum number of processors may be significantly smaller than the number of matrix elements in the dimensions parallelized. Experimental results for the Intel iPSC are presented.

1 Introduction

One of the most frequent operations in scientific and engineering computations is multiplication of matrices. We analyze this problem for arbitrarily shaped matrices and Boolean n-cube configured multiprocessors, and express the algorithms in generic communication primitives. The concurrent algorithms for matrix multiplication that we present here are general. Cannon's algorithm [1] for multiplication of square matrices on a mesh is a special case of the algorithms presented here, and so is Dekel's [2] algorithm for multiplication of square matrices on a Boolean cube. Another special case included in our formulation is the multiplication of $N \times N$ matrices on a Boolean cube of at least N^3 processors in $O(\log_2 N)$ time.

For uniprocessors there exists a well defined set of operators in which computations are conveniently expressed. For multiprocessors such a set has yet to be established. In particular global operations need to be considered. Broadcasting, or copying, of data from one processor to a set of other processors, possibly all other processors, or the concurrent copying from all processors to all other processors are important global operations. We refer to these operations as *one-to-all broadcasting* and *all-to-all broadcasting* [10]. For the reverse operations various reduction operators apply. The most common are '+', '−', '*max*', and '*min*'. Other pure communication operators are the splitting of a data set among all other nodes, i.e., scatter, or the reverse operation, i.e., gatter [13]. We refer to this communication as *one-to-all personalized communication*, or *all-to-all personalized communication* [10] for the case that every processor splits and distributes its data set to all other processors. These communication with operation primitives are very powerful for expressing parallel algorithms. With operations of this type being part of the machine instruction set efficient portable code can be written. The code

is also compact.

For multiprocessors proper partitioning significantly reduces the communications requirements. If each processor holds a square block matrix of M elements of each of the operands, then $2M\sqrt{M}$ arithmetic operations are required per communication of two blocks, i.e., $2M$ elements. The number of arithmetic operations per element communication is $\sqrt{M}$. However, if the aspect ratio of the blocks increases then the ratio between elementary arithmetic operations and element communications approaches 1. As the number of processors increases relative to the matrix size, the importance of efficient communication increases.

In languages like Fortran 77 a matrix multiplication requires three nested loops. Any one, two, or all three loops can be parallelized. With N processors one important consideration is how to factor $N = N_1 \times N_2 \times N_3$ in the row (P), column (R), and "outer product direction" (Q) in multiplying a $P \times Q$ by a $Q \times R$ matrix. We first investigate parallelizing one of the loops by a one-dimensional partitioning of the matrices, then parallelizing two loops, and finally all three loops. We show that the three-dimensional partitioning always yields a complexity that is at most as high as that of the two-dimensional partitioning, which in turn is at most as high as the one-dimensional partitioning. We also show that the aspect ratio of the processor array in the two- and three-dimensional partitionings shall be the same as that of the matrix guiding the choice of algorithm, i.e., A, C or D. The processors shall be aligned with the axis with the largest number of elements in the one-dimensional partitioning, and with the plain with the largest number of elements in the two-dimensional partitioning. Data permutations may be required to accomplish this processor allocation.

In the Boolean n-cube architectures we consider, storage is uniformly distributed among nodes of identical architecture. The main feature of Boolean cube configured architectures, and other architectures designed to be scalable to a large number of processors, is that a high storage and communication bandwidth can be achieved at a relatively low cost. Similarly, the processing capability is obtained through replication, which, in VLSI technology, is cheap. The ensemble architecture can be operated with a single instruction stream, SIMD (Single Instruction Multiple Data) [11], or each node, or a subset thereof, may have their own instruction stream, resulting in a MIMD (Multiple Instruction Multiple Data) architecture. We present algorithms suitable for both kinds of architectures.

The outline of this paper is as follows. In the next section the notation and definitions used throughout the paper are introduced. Section 3 introduces some communication routines and their complexity. Section 4 presents matrix multiplication algorithms and a complexity analysis using the generic communication primitives for one-, two-, and three-dimensional partitioning. Section 5 gives a summary and conclusions. Some of the estimated communication complexities have been verified through measurements on the Intel iPSC [6].

2 Notation and Definitions

Throughout the paper N denotes the number of processors. With respect to algorithms and data structures we factor N as $N_1 \times N_2 \times N_3$. We consider the matrix operation $A \leftarrow C \times D + E$ where all matrices are dense, C a $P \times Q$ matrix, D a $Q \times R$ matrix, and A and E $P \times R$ matrices. A Boolean n-cube has $N = 2^n$ nodes and diameter n. It can be constructed recursively by joining corresponding nodes of two $(n-1)$-cubes. Nodes can be given addresses such that adjacent nodes differ in precisely one bit, Figure 1. The distance between a pair of nodes $i = (i_{n-1}i_{n-2}\ldots i_0)$ and $j = (j_{n-1}j_{n-2}\ldots j_0)$ is equal to the *Hamming* distance between i and j, where $Hamming(i,j) = \sum_{k=0}^{n-1}(i_k \oplus j_k)$, and '$\oplus$' is the exclusive-or operator. Between any pair of nodes (i,j) there are $Hamming(i,j)$ paths of length $Hamming(i,j)$, and $n - Hamming(i,j)$ paths of length $Hamming(i,j)+2$ [14]. The fanout of every node is n, and the total number of communication links is $\frac{1}{2}nN$. The number of nodes at distance i from a node is $\binom{n}{i}$ and the average distance between nodes is $\frac{1}{2}n$.

For the communication system we consider *one-port* communication, for which communication can only take place on one port at a time for each processor, and *n-port* communication for which all ports on each processor can be used concurrently. Note that we assume one send and one receive operation (possibly through different ports) can be done in one communication step for *one-port* communication. In *one-to-all broadcasting* a single node communicates the same information to every other node. In the *all-to-all* case every node performs *one-to-*

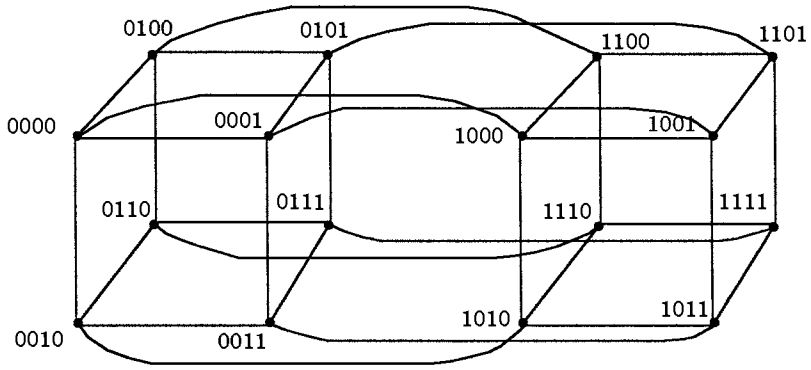


Figure 1: A recursive construction of Boolean cubes.

all broadcasting. In *one-to-all personalized communication* a node sends a unique piece of information to every other node. In *all-to-all personalized communication* every node performs *one-to-all personalized communication*. The time for a complete communication is denoted T(*number of ports used concurrently, source address, number of destinations; array identifier, data volume*). The symbol $\cdot$ is used as a dummy argument, and the $*$ denotes that the operation is applied to all values the argument can assume. The routing schemes are indexed similarly. T_{lb} denotes lower bound estimates.

The Intel iPSC can be configured with up to 128 processors. It has a message passing programming model. The operating system subdivides messages into $1k$ byte packets. With the current operating systems, NX, the start-up time is $\approx$ 1.5 *msec*. The storage bandwidth allows concurrent communication on $2-3$ ports. However, we have been unable to realize this potential effectively with any of the available operating systems. The concurrency in communication on different ports of the same processor amounts to an overlap of about 20%. The computational rate realized from FORTRAN is approximately 30 kflops per node. With a relatively large start-up time it is desirable to send long messages. This may require internal data movement. However, the copy time is also significant on the iPSC.

For the analysis we denote the maximum packet size by B_m, the communication start-up time with τ, the time for transmission of an element by t_c, and the time for an arithmetic operation by t_a. On the Intel iPSC there exists a block size $B_{copy} < B_m$ above which it is better to minimize copy time than start-up time. We have not included the time for data movement internal to a node in the complexity estimates below for reasons of clarity. For details see [4]. We assume that arithmetic and communication operations do not overlap in time, and that no two global communication operations overlap. We do not here include a start-up time for the arithmetic, but discuss features of the local computations of importance for processors with pipelined arithmetic units, or units with fast inner-product instructions, or cache based processors. The model is verified on the Intel iPSC/d5.

3 Communication Routines

The communication routines we use for matrix multiplication on the Boolean cube are the following. The routines are described in detail in [10].

One-to-all communication:

- A single Hamiltonian path for *one-port* communication, $GC(1, \cdot, N; \cdot, \cdot)$, and n rotated paths for *n-port* communication, $GC(n, \cdot, N; \cdot, \cdot)$.
- A Spanning Binomial Tree, SBT$(\cdot, \cdot, N; \cdot, \cdot)$.
- n Rotated Spanning Binomial Trees, nRSBT$(\cdot, \cdot, N; \cdot, \cdot)$.

Algorithm	Element transfers	start-ups	B_{opt}	*min* start-ups
GC$(1,\cdot,N;\cdot,M)$	$M+(N-2)B$	$\lceil\frac{M}{B}\rceil+N-2$	$\sqrt{\frac{M\tau}{(N-2)t_c}}$	$N-2+2\sqrt{(N-2)M\frac{t_c}{\tau}}$
SBT-b$(1,\cdot,N;\cdot,M)$	Mn	$\lceil\frac{M}{B_m}\rceil n$	M	n
nESBT-b$(1,\cdot,N;\cdot,M)$	$M+nB_m$	$\lceil\frac{M}{B_m}\rceil+n$	$\sqrt{\frac{M\tau}{nt_c}}$	$n+2\sqrt{nM\frac{t_c}{\tau}}$
nRSBT-b$(1,\cdot,N;\cdot,M)$	Mn	$2\sum_{i=0}^{n-1}\lceil\frac{Mi}{nB_m}\rceil+\lceil\frac{M}{B_m}\rceil$	M	$2n-1$
GC$(n,\cdot,N;\cdot,M)$	$\frac{M}{n}(N-1)$	$\lceil\frac{M}{nB_m}\rceil(N-1)$	$\frac{M}{n}$	$N-1$
SBT-b$(n,\cdot,N;\cdot,M)$	$M+(n-1)B$	$\lceil\frac{M}{B}\rceil+n-1$	$\sqrt{\frac{M\tau}{(n-1)t_c}}$	$n-1+2\sqrt{(n-1)M\frac{t_c}{\tau}}$
nESBT-b$(n,\cdot,N;\cdot,M)$	$\frac{M}{n}+nB$	$\lceil\frac{M}{nB}\rceil+n$	$\frac{1}{n}\sqrt{\frac{M\tau}{t_c}}$	$n+2\sqrt{M\frac{t_c}{\tau}}$
nRSBT-b$(n,\cdot,N;\cdot,M)$	M	$\lceil\frac{M}{nB_m}\rceil n$	$\frac{M}{n}$	n

Table 1: Estimated *one-to-all broadcasting* times for various spanning graphs.

- A Spanning Balanced n-Tree, SBnT$(\cdot,\cdot,N;\cdot,\cdot)$.

All-to-all communication:

- N translated Hamiltonian paths, called a Gray code exchange algorithm, GCEA$(\cdot,*,N;\cdot,\cdot)$.
- A single path for *one-port* communication and n rotated paths for *n-port* communication, called cyclic rotation algorithm CRA$(\cdot,*,N;\cdot,\cdot)$.
- N translated Spanning Binomial Trees, SBT$(\cdot,*,N;\cdot,\cdot)$.
- N translated nRSBT graphs, nRSBT$(\cdot,*,N;\cdot,\cdot)$.
- N translated Spanning Balanced n-Trees, SBnT$(\cdot,*,N;\cdot,\cdot)$.

Translation is accomplished by a bit-wise exclusive-or operation on the root of the graph, and rotation by a cyclic shift on the dimensions. When there is a need to distinguish between routing for broadcasting and routing for personalized communication, we do that by affixing -b, or -p to the name of the routing scheme. As an example, SBT-b$(1,*,N;A,M)$ denotes *all-to-all broadcasting* of an array A having M elements using *one-port* communication for a spanning binomial tree routing on N processors.

The Boolean cube is Hamiltonian. Such a path can be constructed by a *binary-reflected* Gray code [7]. For *n-port* communication n paths can be constructed through rotation of the dimensions. However, these paths are not edge-disjoint [10], in general, and messages can be pipelined only to a limited extent. For instance, it can be shown, that there does not exist 3 (directed) edge-disjoint Hamiltonian circuits in a 3-cube. However, there exist 4 (directed) edge-disjoint Hamiltonian circuits in a 4-cube. In the *Gray Code Exchange Algorithm* (GCEA), the sequence of exchange dimensions is exactly the sequence of dimensions encountered in traversing the cube in the binary-reflected Gray code order [12]. In the *cyclic rotation algorithm* (CRA), a node always sends to and receives data from the same neighbors for all routing steps, whereas in the former all the nodes have the same exchange sequence.

A Spanning Binomial Tree can be generated by complementing leading zeroes of the processor addresses. The path length to every node is minimal. A spanning tree does not fully use the bandwidth of the Boolean cube. A higher utilization can be achieved by spanning graphs, for instance by forming the union of n distinctly rotated SBTs, which is the nRSBT graph [10], or by rotation and translation, which can yield n Edge-disjoint SBTs (nESBT). Yet another tree that can be used for lower bound routing algorithms in the case of *all-to-all broadcasting* is a *Spanning Balanced n-Tree* (SBnT) [10,5]. In such a tree the node set of the cube is divided into n approximately equal sets, with each such set forming a subtree of the source node.

Model	Algorithm	Element transfers	start-ups	B_{opt}	*min* start-ups
one-port	CRA$(1,*,N;\cdot,M)$	$(N-1)M$	$\lceil\frac{M}{B_m}\rceil(N-1)$	M	$N-1$
	GCEA$(1,*,N;\cdot,M)$	$(N-1)M$	$\lceil\frac{M}{B_m}\rceil(N-1)$	M	$N-1$
	SBT-b$(1,*,N;\cdot,M)$	$(N-1)M$	$\sum_{i=0}^{n-1}\lceil\frac{2^iM}{B_m}\rceil$	$\frac{NM}{2}$	n
	SBnT-b$(1,*,N;\cdot,M)$	$(N-1)M$	$\max(2n-1,\frac{(N-1)M}{B_m})$	$\frac{(N-1)M}{n}$	$2n-1$
	nRSBT-b$(1,*,N;\cdot,M)$	$(N-1)M$	$\max(2n-1,\frac{(N-1)M}{B_m})$	$\frac{(N-1)M}{n}$	$2n-1$
n-port	GCEA$(n,*,N;\cdot,M)$	$\frac{1}{n}(N-1)M$	$\lceil\frac{M}{nB_m}\rceil(N-1)$	$\frac{M}{n}$	$N-1$
	SBT-b$(n,*,N;\cdot,M)$	$\frac{1}{2}NM$	$\sum_{i=0}^{n-1}\lceil\binom{n-1}{i}\frac{M}{B_m}\rceil$	$\frac{NM}{\sqrt{2\pi(n-1)}}$	n
	SBnT-b$(n,*,N;\cdot,M)$	$\frac{1}{n}(N-1)M$	$\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{M}{nB_m}\rceil$	$\sqrt{\frac{2}{\pi}}\frac{NM}{n^{3/2}}$	n
	nRSBT-b$(n,*,N;\cdot,M)$	$\frac{1}{n}(N-1)M$	$\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{M}{nB_m}\rceil$	$\sqrt{\frac{2}{\pi}}\frac{NM}{n^{3/2}}$	n

Table 2: The communication complexity of *all-to-all broadcasting*.

Model	Algorithm	Element transfers	start-ups	B_{opt}	*min* start-ups
one-port	SBT-p$(1,\cdot,N;\cdot,M)$	$(N-1)M$	$\sum_{i=0}^{n-1}\lceil\frac{2^iM}{B_m}\rceil$	$\frac{NM}{2}$	n
n-port	SBT-p$(n,\cdot,N;\cdot,M)$	$\frac{1}{2}NM$	$\sum_{i=0}^{n-1}\lceil\binom{n-1}{i}\frac{M}{B_m}\rceil$	$\frac{NM}{\sqrt{2\pi(n-1)}}$	n
	SBnT-p$(n,\cdot,N;\cdot,M)$	$\frac{1}{n}(N-1)M$	$\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{M}{nB_m}\rceil$	$\sqrt{\frac{2}{\pi}}\frac{NM}{n^{3/2}}$	n
	nRSBT-p$(n,\cdot,N;\cdot,M)$	$\frac{1}{n}(N-1)M$	$\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{M}{nB_m}\rceil$	$\sqrt{\frac{2}{\pi}}\frac{NM}{n^{3/2}}$	n

Table 3: The communication complexity of *one-to-all personalized communication*.

The communication complexities of *one-to-all broadcasting* are summarized in Table 1, and the complexities for *all-to-all broadcasting* are summarized in Table 2. For *one-port* communication the SBT-b$(1,*,N;\cdot,M)$ algorithm is optimal within a factor of 2 for sufficiently large maximum packet size, $B_m \geq \frac{NM}{2}$. For $B_m \leq \frac{(N-1)M}{n}$ the routing complexity of SBT-b$(1,*,N;\cdot,M)$, SBnT-b$(1,*,N;\cdot,M)$, and nRSBT-b$(1,*,N;\cdot,M)$ are approximately equal. For $B_m \leq M$ the routing complexity of all algorithms presented here are approximately equal, i.e., no better than a Hamiltonian path based routing. For *n-port* communication the nRSBT-b$(n,*,N;\cdot,M)$ and SBnT-b$(n,*,N;\cdot,M)$ routings are optimal for $B_m \geq \sqrt{\frac{2}{\pi}}\frac{NM}{n^{3/2}}$. For $B_m \leq \frac{M}{n}$ these two routings are no better than the GCEA$(n,*,N;\cdot,M)$ routing. The SBT-b$(n,*,N;\cdot,M)$ routing is inferior.

In *one-to-all personalized communication* there are $N-1$ distinct sets M to be sent from the source node. The root is the bottleneck. In personalized communication each internal node of the spanning graphs sends out data for all the nodes in the subgraph for which it is the root. The lower bound for *one-port one-to-all personalized communication* is $T_{l\ b}(1,\cdot,N;\cdot,M) = \max((N-1)Mt_c, n\tau)$. The SBT-p$(1,\cdot,N;\cdot,M)$ is optimal within a factor of 2 for sufficiently large maximum packet size, $B_m \geq \frac{1}{2}NM$. The SBT-p$(n,\cdot,N;\cdot,M)$ routing is not optimal. The lower bound for *n-port* communication is $\max(\frac{(N-1)M}{n}t_c, n\tau)$. The nRSBT-p$(n,\cdot,N;\cdot,M)$ and SBnT-p$(n,\cdot,N;\cdot,M)$ routings yield minimal (within a factor of 2) routing times if $B_m \geq \sqrt{\frac{2}{\pi}}\frac{NM}{n^{3/2}}$. *All-to-all personalized communication* performed by N SBTs amounts to a sequence of exchange operations in the different dimensions with *one-port* communication. Unlike the case in all-to-all broadcasting, the data volume being exchanged remains constant through all steps, and equal to the maximum in the broadcasting case, i.e., $\frac{1}{2}NM$. The SBT-p$(1,*,N;\cdot,M)$ is optimum within a factor of 2. With *n-port* communication the lower bound for the transmission time is reduced by a factor of n. Hence, $T_{l\ b}(n,*,N;\cdot,M) = \max(\frac{NM}{2}t_c, n\tau)$. The SBT-p$(n,*,N;\cdot,M)$ routing has a data transfer time a factor of n higher than the minimum [10]. But, the nRSBT-p$(n,*,N;\cdot,M)$ and the SBnT-p$(n,*,N;\cdot,M)$ routings can route in a time proportional to the lower bound if $B_m \geq \frac{NM}{2n}$ (or $\geq \frac{(N-1)M}{n}$ for the latter) [10]. Tables 3 and 4 summarize the communication complexities for *personalized communications*.

Model	Algorithm	Element transfers	start-ups	B_{opt}	*min* start-ups
one-port	SBT-p$(1,*,N;\cdot,M)$	$\frac{1}{2}nNM$	$\lceil\frac{NM}{2B_m}\rceil n$	$\frac{NM}{2}$	n
n-port	SBnT-p$(n,*,N;\cdot,M)$	$\frac{1}{2}NM$	$\sum_{i=1}^{n}\lceil\sum_{j=i}^{n}\binom{n}{j}\frac{M}{nB_m}\rceil$	$\frac{(N-1)M}{n}$	n
	nRSBT-p$(n,*,N;\cdot,M)$	$\frac{1}{2}NM$	$\lceil\frac{NM}{2nB_m}\rceil n$	$\frac{NM}{2n}$	n

Table 4: The communication complexity of *all-to-all personalized communication.*

4 Matrix Multiplication

We refer to the three loops in a Fortran like language for the operation $A \leftarrow C \times D + E$ as the P, Q, and R directions. One, two, or all three loops can be parallelized. We refer to these three forms of concurrent matrix multiplication as *one-*, *two-* and *three*-dimensional partitioning. The maximum number of processors that can be used depends on which loop(s) are parallelized, and the algorithms also depend thereupon. The maximum arithmetic speed-up for the three partitionings are $\max(P,Q,R)$, $\max(PQ,QR,PR)$, and $\frac{2PQR}{2+\log_2 Q}$ respectively. We will show that the communication complexity is lower for the three-dimensional partitioning than for the two-dimensional partitioning, which in turn is lower than that of the one-dimensional partitioning.

With multiple matrix elements per processor elements can be assigned *cyclicly* or *consecutively* to processors [8]. In two-dimensional *cyclic* partitioning, matrix element (i,j) is stored in processors $PID(i)\|PID(j)$, where $PID(i) = i \bmod N_1$, $PID(j) = j \bmod N_2$, $N_1 \times N_2 = N$. In the *consecutive* storage, matrix element (i,j) is stored in processor $PID(i)\|PID(j)$, where $PID(i) = \lfloor\frac{i}{\lceil\frac{P}{N_1}\rceil}\rfloor$, and $PID(j) = \lfloor\frac{j}{\lceil\frac{Q}{N_2}\rceil}\rfloor$ for a $P \times Q$ matrix. Note that in the *consecutive* partitioning matrix elements with the same high order bits are allocated to the same processor. In the *cyclic* partitioning, elements in the same processor have the same low order bits. The communication and computational complexity for matrix multiplication with matrices stored according to either of these two storage schemes is the same. We assume *consecutive* storage. In a three-dimensional partitioning $N_1 \cdot N_2 \cdot N_3 = N$, and the loop ranging over the set of Q outer-products is instantiated in N_3 spatial components. The two- and one-dimensional partitionings are degenerate cases of the three-dimensional partitioning. Algorithms for which the inner-products for each element of A are accumulated in the same location as the elements of A are called *in-place* algorithms. Algorithms accumulating inner-products through the communication of partial sums are called *in-space* algorithms [7].

4.1 One-Dimensional Partitioning

We consider four basic algorithm for the computation $A \leftarrow C \times D + E$ with all matrices stored by column-wise partitioning. The algorithms differ in which loop is parallelized for the multiplication, the required data permutations, and other communication operations.

- **Algorithm $A(\cdot,1,1)$.** Compute A *in-place* by *broadcasting* of C from every processor that has elements of C to every processor that has elements of D. Processor $k = PID(j)$ computes $CD(*, \lfloor\frac{j}{\lceil\frac{R}{N}\rceil}\rfloor)$ for all j mapped to k.
- **Algorithm $A(\cdot,1,2)$.** Compute A by a transpose of C and *broadcasting* of C^T from every processor that has elements of C^T to every processor that has elements of D. Processor $k = PID(j)$ computes $CD(*, \lfloor\frac{j}{\lceil\frac{R}{N}\rceil}\rfloor)$ for all j mapped to k.
- **Algorithm $A(\cdot,1,3)$.** Compute A by a transpose of C, *broadcasting* of D from every processor that has elements of D to every processor that has elements of C^T, and transpose A^T. Processor $k = PID(j)$ computes $C(\lfloor\frac{j}{\lceil\frac{P}{N}\rceil}\rfloor, *)D$.
- **Algorithm $A(\cdot,1,4)$.** Compute A *in-space* by a transpose of D, and *reduction* of partial inner products of A.

Column Partitioning:

$$\mathcal{A}(\cdot,1,1): \quad C_{*k}, D_{*k} \xrightarrow{\text{brd. C, } \leftrightarrow} C_{**}, D_{*k} \xrightarrow{\text{mpy., [R]}} A_{*k}$$

$$\mathcal{A}(\cdot,1,3): \quad C_{*k}, D_{*k} \xrightarrow{\text{txp. C, } \swarrow} C_{k*}, D_{*k} \xrightarrow{\text{brd. D, } \leftrightarrow} C_{k*}, D_{**} \xrightarrow{\text{mpy., [P]}} A_{k*} \xrightarrow{\text{txp. A, } \swarrow} A_{*k}$$

$$\mathcal{A}(\cdot,1,4): \quad C_{*k}, D_{*k} \xrightarrow{\text{txp. D, } \swarrow} C_{*k}, D_{k*} \xrightarrow{\text{mpy., [Q]}} A^k_{**} \xrightarrow{\text{red. A, } \leftrightarrow} A_{*k}$$

Row Partitioning:

$$\mathcal{A}(\cdot,1,1): \quad C_{k*}, D_{k*} \xrightarrow{\text{brd. D, } \updownarrow} C_{k*}, D_{**} \xrightarrow{\text{mpy., [P]}} A_{k*}$$

$$\mathcal{A}(\cdot,1,3): \quad C_{k*}, D_{k*} \xrightarrow{\text{txp. D, } \swarrow} C_{k*}, D_{*k} \xrightarrow{\text{brd. C, } \updownarrow} C_{**}, D_{*k} \xrightarrow{\text{mpy., [R]}} A_{*k} \xrightarrow{\text{txp. A, } \swarrow} A_{k*}$$

$$\mathcal{A}(\cdot,1,4): \quad C_{k*}, D_{k*} \xrightarrow{\text{txp. C, } \swarrow} C_{*k}, D_{k*} \xrightarrow{\text{mpy., [Q]}} A^k_{**} \xrightarrow{\text{red. A, } \updownarrow} A_{*k}$$

Figure 2: Notation summary of algorithms for one-dimensional partitioning.

The algorithms are identified by $\mathcal{A}$(*number of ports used concurrently, number of loops parallelized, algorithm identifier*). Algorithm $\mathcal{A}(\cdot,1,2)$ is clearly inferior to algorithm $\mathcal{A}(\cdot,1,1)$ and is not further considered for the one-dimensional partitioning, but will be considered for the two-dimensional partitioning. For row partitioning the roles of C and D are interchanged. Figure 2 characterizes the basic algorithms. The two subscripts denote the ordinal numbers of block rows and block columns. The superscript denotes the ordinal number of the partial inner product result. The number in the square brackets (eg. [R] in $\mathcal{A}(\cdot,1,1)$) is the number of processors that minimizes the arithmetic time for each algorithm.

4.1.1 Arithmetic Complexity

The total number of arithmetic operations in sequence, and the characteristics of the local computations organized as AXPY, or inner-product computations are summarized in Table 5. The vector length is shortest for algorithm $\mathcal{A}(\cdot,1,3)$, and the potential for register operations the smallest for algorithm $\mathcal{A}(\cdot,1,4)$. If $P \bmod N = Q \bmod N = R \bmod N = 0$, then all the algorithms use the processors evenly, and the arithmetic complexity is the same. But, if $P = c_1N$, $Q = c_2N$ and $R = 1$, then the arithmetic complexities are $2c_1c_2N^2$, $2c_1c_2N^2$, $2c_1c_2N$, and $2c_1c_2N + c_1nN$, respectively. Distributing D in space (Algorithms $\mathcal{A}(\cdot,1,3)$ and $\mathcal{A}(\cdot,1,4)$) is clearly more effective than distributing C for computations of type matrix-vector multiplication. By distributing D in space the processors are aligned with the axis with the largest number of elements.

Lemma 1 *For one-dimensional partitioning, the arithmetic complexity is minimized if the processors are aligned with the direction P, if $\frac{1}{P}\lceil\frac{P}{N}\rceil \leq \frac{1}{Q}\lceil\frac{Q}{N}\rceil, \frac{1}{R}\lceil\frac{R}{N}\rceil$, with Q if $\frac{1}{Q}\lceil\frac{Q}{N}\rceil \leq \frac{1}{P}\lceil\frac{P}{N}\rceil, \frac{1}{R}\lceil\frac{R}{N}\rceil$, and with R if $\frac{1}{R}\lceil\frac{R}{N}\rceil \leq \frac{1}{Q}\lceil\frac{Q}{N}\rceil, \frac{1}{P}\lceil\frac{P}{N}\rceil$.*

Corollary 1 *If P, Q, and R all are multiples of N, then the parallel arithmetic complexity is the same regardless of the axis with which the processors are aligned. For $P, Q, R \leq N$, the arithmetic complexity is minimized if the processors are aligned with the dimension with the largest number of components.*

4.1.2 Communication Routines and Their Complexities

The basic algorithms consist of different combinations of *broadcasting/reduction* and *personalized communication* for matrix transposition [9], in addition to the arithmetic operations. The choice of communication routine is affected by available temporary storage, start-up times and transmission rates of the channels. Tables 6, 7, and 8 give the communication complexity for the different multiplication algorithms (assuming column partitioning).

Algorithm	Number of Arithmetic operations	max. no. of processors	GEMV			inner-products	
			no. of	no. of AXPY	v-length	no. of	order
$\mathcal{A}(\cdot,1,1)$	$2PQ\lceil\frac{R}{N}\rceil$	R	$\frac{R}{N}$	Q	P	$P\frac{R}{N}$	Q
$\mathcal{A}(\cdot,1,2)$	$2PQ\lceil\frac{R}{N}\rceil$	R	$\frac{R}{N}$	Q	P	$P\frac{R}{N}$	Q
$\mathcal{A}(\cdot,1,3)$	$2QR\lceil\frac{P}{N}\rceil$	P	R	Q	$\frac{P}{N}$	$R\frac{P}{N}$	Q
$\mathcal{A}(\cdot,1,4)$	$PR(2\lceil\frac{Q}{N}\rceil-1)+P(\lceil\frac{R}{N}\rceil+\sum_{i=1}^{n}\lceil\frac{R}{2^i}\rceil)$	Q	R	$\frac{Q}{N}$	P	PR	$\frac{Q}{N}$

Table 5: The local arithmetic operations for one-dimensional partitioning.

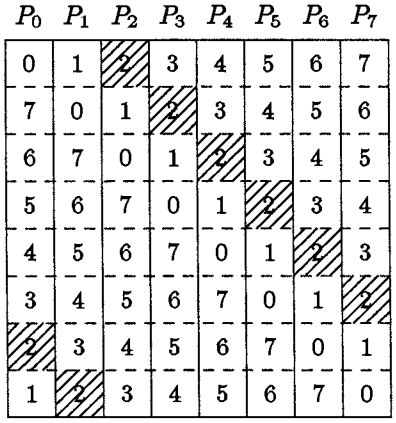


Figure 3: Computing $A \leftarrow C \times D + E$ by rotation of C.

For *one-port* communication the CRA$(1,*,N;\cdot,\cdot)$, GCEA$(1,*,N;\cdot,\cdot)$, and SBT$(1,*,N;\cdot,\cdot)$ routines all have the same complexity, if $B_m \leq$ the message size (per processor). The CRA$(1,*,N;\cdot,\cdot)$ yields a one-dimensional version of Cannon's [1] algorithm, and the GCEA$(1,*,N;\cdot,\cdot)$ routine yields a one-dimensional version of Dekel's [2] algorithm. With the CRA$(1,*,N;\cdot,\cdot)$ routine products are formed on the shaded parts of D during step i, $1 \leq i \leq N$, Figure 3. C is rotated $N-1$ times to the right (or left). The SBT-b$(1,*,N;\cdot,\cdot)$ and SBT-p$(1,*,N;\cdot,\cdot)$ algorithms are optimal for *one-port* communication and unlimited buffer size, i.e., $B_m \geq \frac{1}{2}PQ$ (or $\frac{PQ}{2N}$), for *all-to-all broadcasting* (or *all-to-all personalized communication*) of a $P \times Q$ matrix partitioned evenly by rows or columns. Figure 4 displays the steps during which a $\frac{Q}{N} \times \frac{R}{N}$ block of D is multiplied by a $P \times \frac{2^{i-1}Q}{N}$ block column of C for partitioning by columns and algorithm $\mathcal{A}(\cdot,1,1)$.

Figure 5 (left) shows the measured times of the GCEA and the SBT routings on the iPSC with fixed matrices,

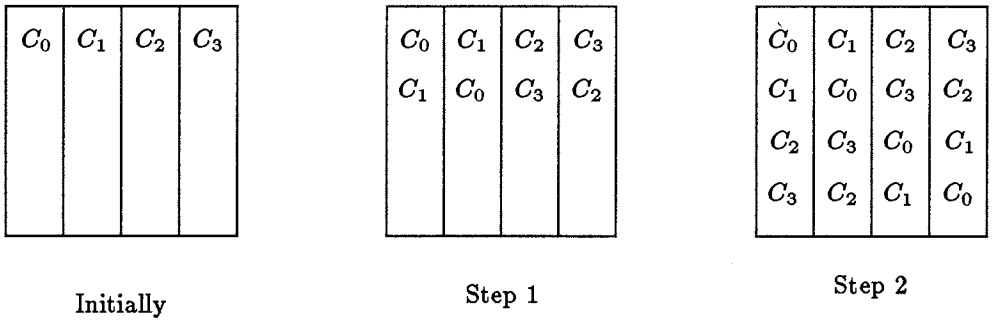


Figure 4: Computing $A \leftarrow C \times D + E$ by algorithm $\mathcal{A}(\cdot,1,1,)$ and SBT-b$(1,*,N;C,P\frac{Q}{N})$ routing applied to C.

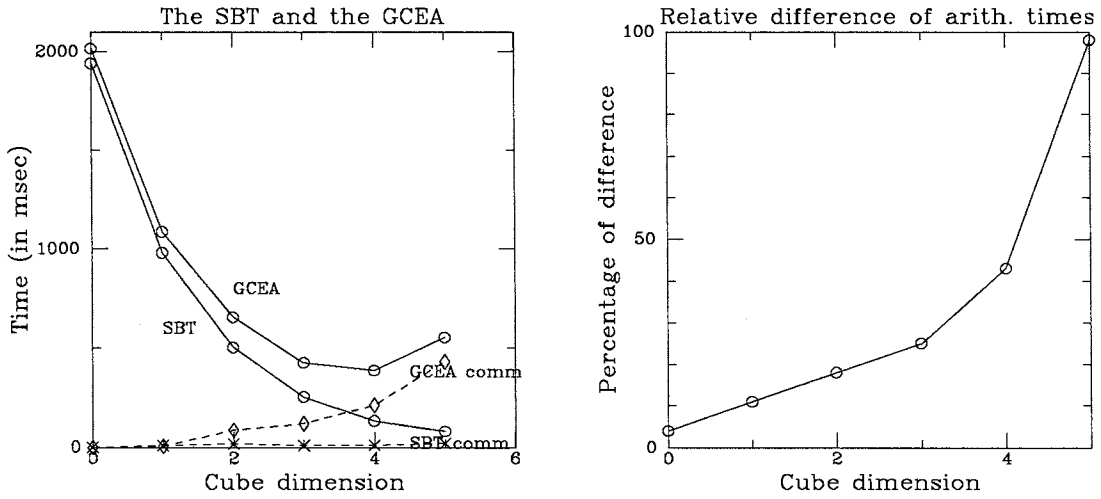


Figure 5: Measured times for matrix multiplication using the GCEA and the SBT routings on the iPSC. $P = Q = R = 32$. On the left, the solid lines are total times and the dashed lines are communication times. On the right, the relative difference of the arithmetic time is shown as a function of cube dimensions.

$P = Q = R = 32$, and varying cube dimensions. The communication times of the SBT routing are less than 10% for cubes with 4 or fewer dimensions. For the GCEA, the communication times are greater than the arithmetic times for cubes with 4 or more dimensions. For the GCEA communication routine the total time for a matrix multiplication has a minimum for a 4-cube. The arithmetic times of an algorithm based on the GCEA communication are higher than that of the SBT communication due to the larger loop overhead. With the GCEA communication N multiplication steps are required, whereas in the SBT communication one or two steps are sufficient. The measured multiplication time for the GCEA based multiplication routine is 10% - 100% higher than that of the SBT based routine as shown on the right of Figure 5.

The number of start-ups of the SBT routing grows linearly with the number of cube dimensions, if the buffer size is sufficiently large. In the last step of the SBT-b$(1, *, N; C, P\frac{Q}{N})$ routing half the matrix ($\frac{1}{2}PQ$ elements) is communicated. If the maximum buffer size is less than this size, additional start-ups are required. The required temporary storage and buffer size grows exponentially with the routing step. If the buffer size is less than the matrix partition residing in a processor, then the time complexities of the SBT and GCEA routings are the same. But, the SBT routing needs additional temporary storage, if multiplication is performed only after the completion of each step of the routing. For the case of limited temporary storage a hybrid of the SBT and the GCEA (or the CRA) routings can be used. The hybrid method will perform k steps of the SBT routing, $0 \leq k \leq n$, during which data is accumulated, followed by $2^{n-k} - 1$ steps of the GCEA or the CRA. Both the maximum temporary storage and the optimum packet size of the hybrid method are $\frac{2^k PQ}{N}$. Note that a single step of the SBT routing is indeed equivalent to one step of the GCEA routing, and it follows that the hybrid routing scheme with $k = n - 1$ and $k = n$ are the same. Figure 6 (left) shows the total time on a 5-cube of the iPSC with $P = Q = R = 32$. The optimum buffer sizes required for the 4- and the 5-cubes exceed the $1k$ bytes internal packet size of the iPSC, so the total time decreases only up to the 3-dimensional cube. The total times of the 4- and 5-cubes are greater than that of the 3-cube due to the different overheads. The temporary storage required with respect to the number of steps in the SBT routing are shown on the right.

With *n-port* communication the GCEA$(n, *, N; \cdot, \cdot)$ algorithm can be employed instead of the GCEA$(1, *, N; \cdot, \cdot)$ algorithm. The data transfer time is thereby reduced by a factor of n, and the total start-up time too, if $B_m \leq \lceil \frac{M}{n} \rceil$. It is also possible to use the SBT$(n, *, N; \cdot, \cdot)$ algorithm. The latter algorithm does not fully utilize the bandwidth of the cube. While the former algorithm does fully utilize the bandwidth of the cube, it requires many more start-ups, in general. The SBnT$(n, *, N; \cdot, \cdot)$ and nRSBT$(n, *, N; \cdot, \cdot)$ algorithms yield a

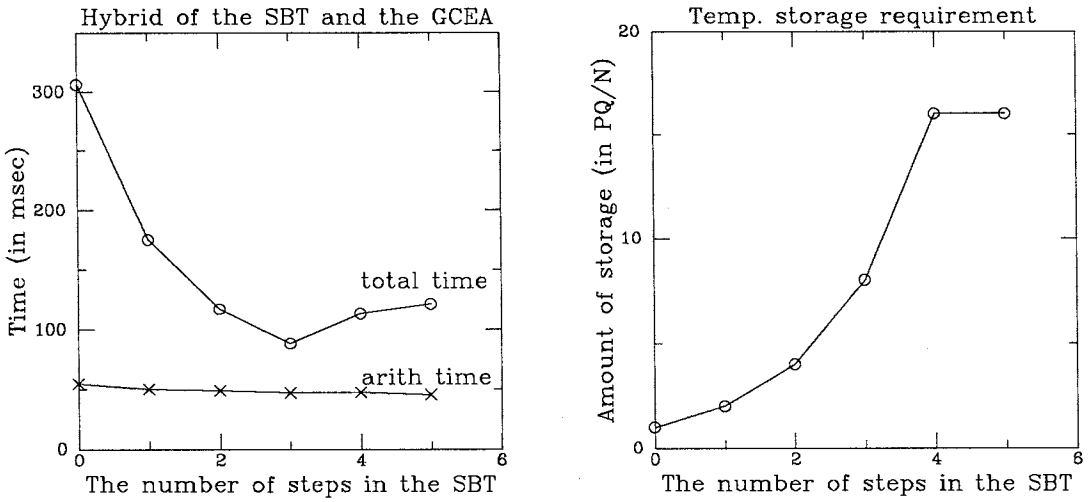


Figure 6: Measured times for matrix multiplication using the hybrid method of the GCEA and the SBT routings on a 5-dimensional cube of the iPSC. $P = Q = R = 32$.

lower time for data transfer regardless of the maximum packet size, and offers the potential for lower bound *all-to-all broadcasting* and *all-to-all personalized communication* within a factor of 2.

4.1.3 Summary and Comparison

We summarize the communication complexities of the multiplication algorithms in Tables 6, 7, and 8 for $P, Q, R \geq N$. Table 6 gives the communication operators, Table 7 the number of element transfers and the number of start-ups, and Table 8 the optimum buffer size and the corresponding total time with the optimum buffer. The superscript l denotes a linear array algorithm, and superscript c a Boolean cube algorithm. Note that in $\mathcal{A}(n,1,3)$, the SBnT-p$(n, *, N; C, \frac{P}{N_1}\frac{Q}{N_2})$ and the SBnT-b$(n, *, N; D, \frac{Q}{N_1}\frac{R}{N_2})$ can in fact be combined into one routing so that n start-ups results, if $B_m \geq \frac{P}{2n}\lceil\frac{Q}{N}\rceil + \sqrt{\frac{2}{\pi}}\frac{QR}{n^{3/2}}$. For some values of P, Q, R less than N, the communication complexity can be smaller than what is given in the tables, because some of the broadcastings and personalized communications can complete earlier. The communication complexity for the general case is complicated, and we only give that of $\mathcal{A}^c(1,1,4)$ as an example. The complexities of other algorithms can be derived similarly. The number of elements transferred is

$$(\max(k_q - k_r, 0) + 2^{k_r} - 1)P\lceil\frac{R}{N}\rceil + (1 + \frac{k_{qr} - 2}{2^{|k_q - k_r|+1}})\max(Q\lceil\frac{R}{N}\rceil, R\lceil\frac{Q}{N}\rceil),$$

where $k_q = \min(\log Q, n)$, $k_r = \min(\log R, n)$ and $k_{qr} = \min(\log Q, \log R, n)$. The number of start-ups is

$$\sum_{i=0}^{k_r-1}\left\lceil\frac{2^i P}{B_m}\lceil\frac{R}{N}\rceil\right\rceil + \max(k_q - k_r, 0)\left\lceil\frac{P}{B_m}\lceil\frac{R}{N}\rceil\right\rceil$$

$$+ \sum_{i=1}^{|k_q-k_r|}\left\lceil\frac{1}{2^i B_m}\max(Q\lceil\frac{R}{N}\rceil, R\lceil\frac{Q}{N}\rceil)\right\rceil + k_{qr}\left\lceil\frac{1}{2^{|k_q-k_r|+1}B_m}\max(Q\lceil\frac{R}{N}\rceil, R\lceil\frac{Q}{N}\rceil)\right\rceil.$$

The minimum number of start-ups is $2\max(k_q, k_r)$.

Lemma 2 *The communication complexity is minimized if for the multiplication the processors are aligned with the axis with the largest number of components.*

Comm. model	Algorithm	Communication operations
one-port	$\mathcal{A}^l$(1,1,1)	CRA$(1,*,N;C,P\frac{Q}{N})$ or GCEA$(1,*,N;C,P\frac{Q}{N})$
	$\mathcal{A}^c$(1,1,1)	SBT-b$(1,*,N;C,P\frac{Q}{N})$
	$\mathcal{A}^c$(1,1,3)	SBT-p$(1,*,N;C,P\frac{Q}{N})$+SBT-b$(1,*,N;D,Q\frac{R}{N})$+SBT-p$(1,*,N;A,R\frac{P}{N})$
	$\mathcal{A}^c$(1,1,4)	SBT-p$(1,*,N;D,Q\frac{R}{N})$+SBT-b$(1,*,N;A,P\frac{R}{N})$
n-port	$\mathcal{A}^l$(n,1,1)	GCEA$(n,*,N;C,P\frac{Q}{N})$
	$\mathcal{A}^c$(n,1,1)	SBnT-b$(n,*,N;C,P\frac{Q}{N})$ or nRSBT-b$(n,*,N;C,P\frac{Q}{N})$
	$\mathcal{A}^c$(n,1,3)	SBnT-p$(n,*,N;C,P\frac{Q}{N})$+SBnT-b$(n,*,N;D,Q\frac{R}{N})$+SBnT-p$(n,*,N;A,R\frac{P}{N})$
	$\mathcal{A}^c$(n,1,4)	SBnT-p$(n,*,N;D,Q\frac{R}{N})$+SBnT-b$(n,*,N;A,P\frac{R}{N})$

Table 6: Communication operators.

Algorithm	Element transfers	start-ups	*min* start-ups
$\mathcal{A}^l$(1,1,1)	$(N-1)P\lceil\frac{Q}{N}\rceil$	$(N-1)\lceil\frac{P}{B_m}\lceil\frac{Q}{N}\rceil\rceil$	$N-1$
$\mathcal{A}^c$(1,1,1)	$(N-1)P\lceil\frac{Q}{N}\rceil$	$\sum_{i=0}^{n-1}\lceil\frac{2^iP}{B_m}\lceil\frac{Q}{N}\rceil\rceil$	n
$\mathcal{A}^c$(1,1,3)	$(N-1)Q\lceil\frac{R}{N}\rceil+\frac{nP}{2}\lceil\frac{R}{N}\rceil+\frac{nP}{2}\lceil\frac{Q}{N}\rceil$	$\sum_{i=0}^{n-1}\lceil\frac{2^iQ}{B_m}\lceil\frac{R}{N}\rceil\rceil+n\lceil\frac{1}{B_m}\frac{P}{2}\lceil\frac{R}{N}\rceil\rceil+n\lceil\frac{1}{B_m}\frac{P}{2}\lceil\frac{Q}{N}\rceil\rceil$	$3n$
$\mathcal{A}^c$(1,1,4)	$(N-1)P\lceil\frac{R}{N}\rceil+\frac{nQ}{2}\lceil\frac{R}{N}\rceil$	$\sum_{i=0}^{n-1}\lceil\frac{2^iP}{B_m}\lceil\frac{R}{N}\rceil\rceil+n\lceil\frac{1}{B_m}\frac{Q}{2}\lceil\frac{R}{N}\rceil\rceil$	$2n$
$\mathcal{A}^l$(n,1,1)	$\frac{1}{n}(N-1)P\lceil\frac{Q}{N}\rceil$	$(N-1)\lceil\frac{P}{nB_m}\lceil\frac{Q}{N}\rceil\rceil$	$N-1$
$\mathcal{A}^c$(n,1,1)	$\frac{1}{n}(N-1)P\lceil\frac{Q}{N}\rceil$	$\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{P}{nB_m}\lceil\frac{Q}{N}\rceil\rceil$	n
$\mathcal{A}^c$(n,1,3)	$\frac{1}{n}(N-1)Q\lceil\frac{R}{N}\rceil+\frac{P}{2}\lceil\frac{R}{N}\rceil+\frac{P}{2}\lceil\frac{Q}{N}\rceil$	$\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{Q}{nB_m}\lceil\frac{R}{N}\rceil\rceil+n\lceil\frac{1}{nB_m}\frac{P}{2}\lceil\frac{R}{N}\rceil\rceil+n\lceil\frac{1}{nB_m}\frac{P}{2}\lceil\frac{Q}{N}\rceil\rceil$	$3n$
$\mathcal{A}^c$(n,1,4)	$\frac{1}{n}(N-1)P\lceil\frac{R}{N}\rceil+\frac{Q}{2}\lceil\frac{R}{N}\rceil$	$\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{P}{nB_m}\lceil\frac{R}{N}\rceil\rceil+n\lceil\frac{1}{nB_m}\frac{Q}{2}\lceil\frac{R}{N}\rceil\rceil$	$2n$

Table 7: The communication complexity using one-dimensional column partitioning, assuming $P,Q,R \geq N$.

Algorithm	B_{opt}	T_{min}
$\mathcal{A}^l$(1,1,1)	$P\lceil\frac{Q}{N}\rceil$	$2PQ\lceil\frac{R}{N}\rceil t_a+(N-1)(P\lceil\frac{Q}{N}\rceil t_c+\tau)$
$\mathcal{A}^c$(1,1,1)	$\frac{PQ}{2}$	$2PQ\lceil\frac{R}{N}\rceil t_a+(N-1)P\lceil\frac{Q}{N}\rceil t_c+n\tau$
$\mathcal{A}^c$(1,1,3)	$\frac{PQ}{2N},\frac{1}{2}QR,\frac{PR}{2N}$	$2QR\lceil\frac{P}{N}\rceil t_a+\{((N-1)Q+\frac{nP}{2})\lceil\frac{R}{N}\rceil+\frac{nP}{2}\lceil\frac{Q}{N}\rceil\}t_c+3n\tau$
$\mathcal{A}^c$(1,1,4)	$\frac{1}{2}Q\lceil\frac{R}{N}\rceil,\frac{1}{2}PR$	$\{PR(2\lceil\frac{Q}{N}\rceil-1)+P(\lceil\frac{R}{N}\rceil+\sum_{i=1}^{n}\lceil\frac{R}{2^i}\rceil)\}t_a+((N-1)P+\frac{nQ}{2})\lceil\frac{R}{N}\rceil t_c+2n\tau$
$\mathcal{A}^l$(n,1,1)	$\frac{1}{n}P\lceil\frac{Q}{N}\rceil$	$2PQ\lceil\frac{R}{N}\rceil t_a+(N-1)(\frac{1}{n}P\lceil\frac{Q}{N}\rceil t_c+\tau)$
$\mathcal{A}^c$(n,1,1)	$\sqrt{\frac{2}{\pi}}\frac{PQ}{n^{3/2}}$	$2PQ\lceil\frac{R}{N}\rceil t_a+\frac{1}{n}(N-1)P\lceil\frac{Q}{N}\rceil t_c+n\tau$
$\mathcal{A}^c$(n,1,3)	$\frac{P}{2n}\lceil\frac{Q}{N}\rceil,\sqrt{\frac{2}{\pi}}\frac{QR}{n^{3/2}},\frac{P}{2n}\lceil\frac{R}{N}\rceil$	$2QR\lceil\frac{P}{N}\rceil t_a+\{(\frac{1}{n}(N-1)Q+\frac{P}{2})\lceil\frac{R}{N}\rceil+\frac{P}{2}\lceil\frac{Q}{N}\rceil\}t_c+3n\tau$
$\mathcal{A}^c$(n,1,4)	$\frac{Q}{2n}\lceil\frac{R}{N}\rceil,\sqrt{\frac{2}{\pi}}\frac{PR}{n^{3/2}}$	$\{PR(2\lceil\frac{Q}{N}\rceil-1)+P(\lceil\frac{R}{N}\rceil+\sum_{i=1}^{n}\lceil\frac{R}{2^i}\rceil)\}t_a+(\frac{1}{n}(N-1)P+\frac{Q}{2})\lceil\frac{R}{N}\rceil t_c+2n\tau$

Table 8: The *optimum* buffer size and time for one-dimensional column partitioning, $P,Q,R \geq N$.

Algorithm	Temporary Storage		
	C	D	A
$\mathcal{A}^1(\cdot,1,1)$	$\frac{PQ}{N}$	0	0
$\mathcal{A}^1(\cdot,1,3)$	0	$\frac{QR}{N}$	0
$\mathcal{A}^1(\cdot,1,4)$	0	0	$\frac{PR}{N}$
$\mathcal{A}^c(\cdot,1,1)$	$\frac{1}{2}PQ$	0	0
$\mathcal{A}^c(\cdot,1,3)$	0	$\frac{1}{2}QR$	0
$\mathcal{A}^c(\cdot,1,4)$	0	0	PR

Table 9: The temporary storage requirements for one-dimensional column partitioning.

Theorem 1 *For P, Q, R multiples of N or $P, Q, R \leq N$, and one-dimensional partitioning, the total complexity for matrix multiplication is minimized if an algorithm is chosen such that for the multiplication the processors are aligned with the axis with the largest number of components.*

Theorem 1 follows from corollary 1 and lemma 2.

Corollary 2 *For the scaling of a column vector, column-vector $\times$ scalar, the processors shall be aligned with the P-axis (algorithm $\mathcal{A}(\cdot, 1,3)$), for scaling of a row vector the processors shall be aligned with the R-axis (algorithm $\mathcal{A}(\cdot,1,1)$), and for an outer product and $P = R$ the processors shall be aligned with the R-axis. For an inner-product the processors shall be aligned with the Q-axis (algorithm $\mathcal{A}(\cdot,1,4)$). For a (square matrix)-vector product algorithm $\mathcal{A}(\cdot,1,4)$ is preferable and for a vector-(square matrix) product algorithm $\mathcal{A}(\cdot,1,1)$ should be chosen.*

The matrix shapes for which the different algorithms yield the lowest total estimated running times for a few combinations of machine parameters are given in Figures 7 to 11. For the first several Figures it is assumed that P, Q, and R are multiples of N. In such a case the communication complexity is the only distinguishing feature between the different algorithms. As the aspect ratio of $\frac{\tau}{t_c}$ increases, Figure 8, the regions for algorithms $\mathcal{A}(\cdot,1,3)$ and $\mathcal{A}(\cdot,1,4)$ are gradually taken over by algorithm $\mathcal{A}(\cdot,1,1)$. As the number of processors increases, part of the $\mathcal{A}(\cdot,1,1)$ region is taken over by the $\mathcal{A}(\cdot,1,3)$ and $\mathcal{A}(\cdot,1,4)$ algorithms. For a sufficiently large number of processors (such as $N = 1024$) the cube is partitioned into three symmetric regions of approximately equal size, Figure 9. Note that for $\frac{P}{N} = \frac{Q}{N} = \frac{R}{N}$, the communication complexity of the $\mathcal{A}(\cdot,1,1)$ is less than that of the $\mathcal{A}(\cdot,1,4)$, which in turn is less than that of the $\mathcal{A}(\cdot,1,3)$ as shown in Table 8, and Figures 7 to 9.

Theorem 2 *The temporary storage needs are minimized with the algorithm that minimizes the communication complexity.*

Lemma 3 *The communication time decreases monotonically with increasing buffer size.*

The lemma is apparent from Table 7. The number of start-ups decreases as a function of B_m. The temporary storage requirements are summarized in Table 9. From Tables 7 and 8 it is also possible to derive an optimum for the number of processors N. For instance, for $B_m = B_{opt}$, $N_{opt} = \min(R, \sqrt{\frac{PQ(Rt_a - t_c)}{\tau}})$ if $Rt_a - t_c \geq 0$, otherwise $N_{opt} = 1$ for algorithm $\mathcal{A}^1(1,1,1)$. Table 10 shows the N_{opt} for various algorithms with $B_m \geq B_{opt}$ (left column) and $B_m \leq B(\cdot,1,\cdot)$ (right column). The values of $B(\cdot,1,\cdot)$ are shown in Table 11.

The special case where $1 \leq P$, Q, $R \leq N$ is shown in Figure 10. The qualitative behavior is the same for the case where P, Q, and R are multiples of N. The P, Q, R space is divided in a similar way with respect to arithmetic as it is with respect to communication.

With *n-port* communication the $\mathrm{SBnT}(n,*,N;\cdot,\cdot)$ or $\mathrm{nRSBT}(n,*,N;\cdot,\cdot)$ based algorithms offer a reduction in the data transfer time by a factor of n. The reduction in the number of start-ups is a factor of n for $B_m \leq \frac{PQ}{nN}$

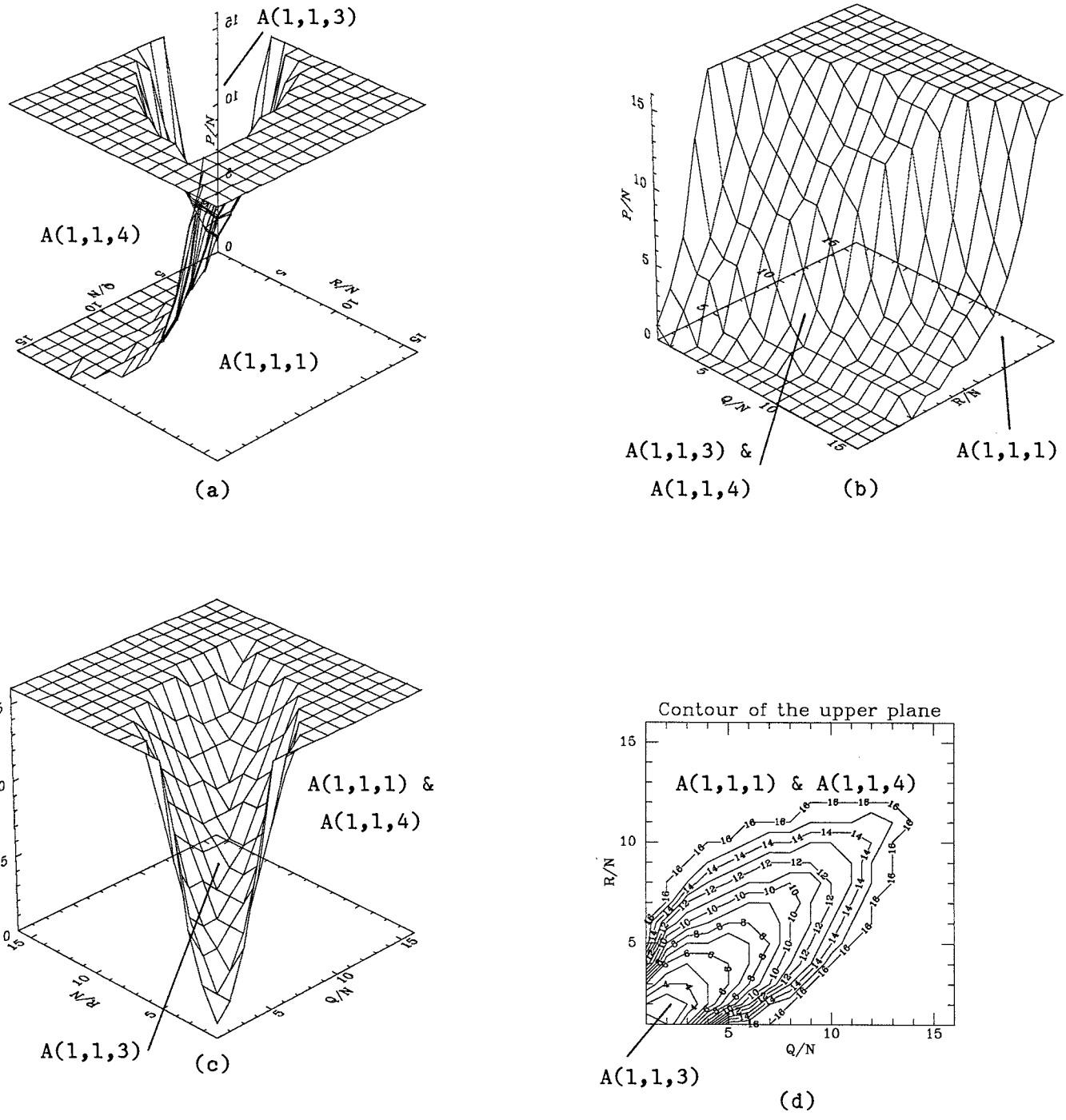


Figure 7: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = t_c$, $B_m = \infty$, column partitioning, *one-port* communication. (a), (b), and (c) present different views of the regions in P, Q, R space defined by the algorithm of lowest communication complexity. (d) shows the contour of the upper plane.

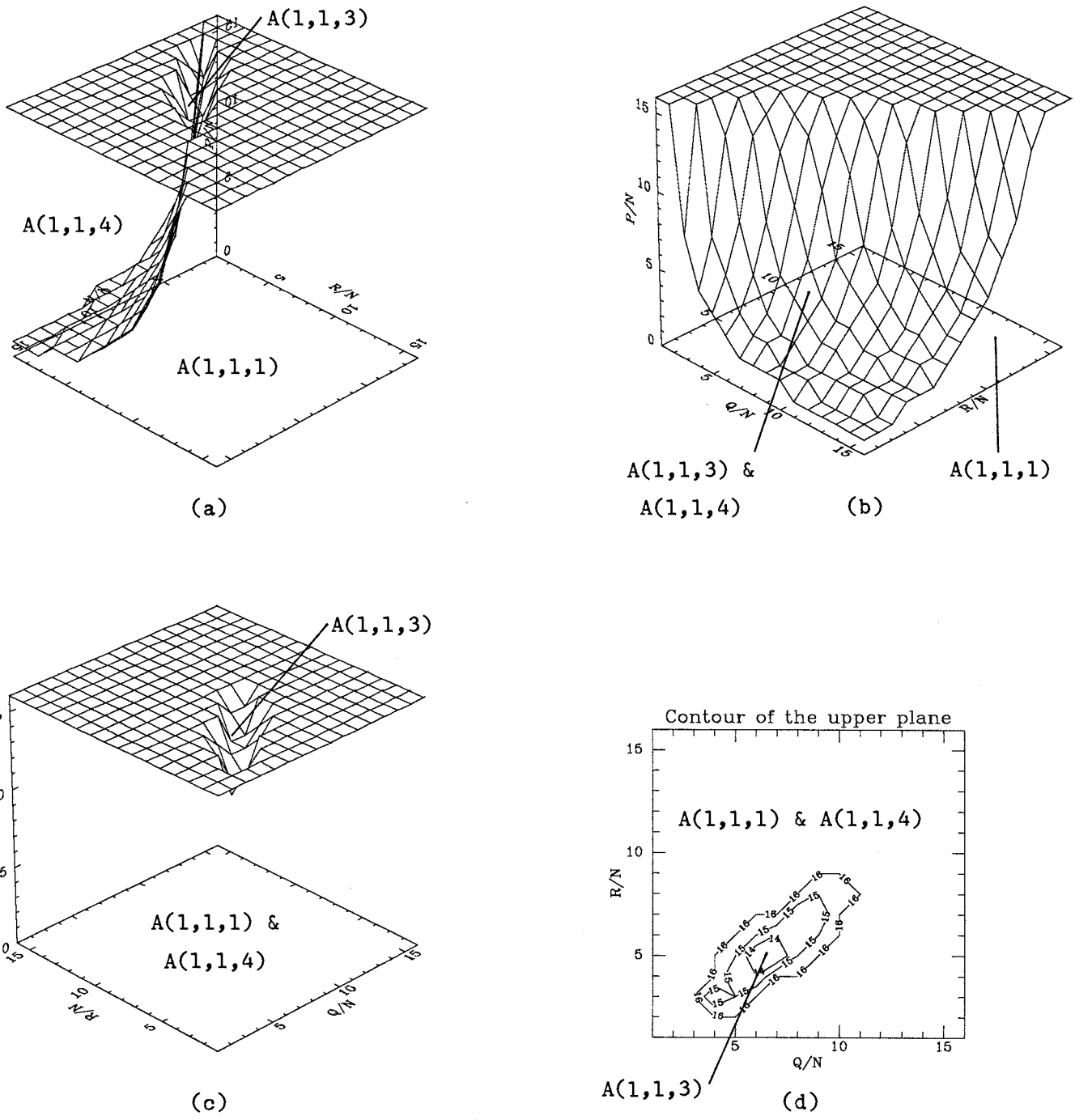


Figure 8: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = 1000t_c$, $B_m = \infty$, column partitioning, *one-port* communication. (a), (b), and (c) present different views of the regions in P, Q, R space defined by the algorithm of lowest communication complexity. (d) shows the contour of the upper plane.

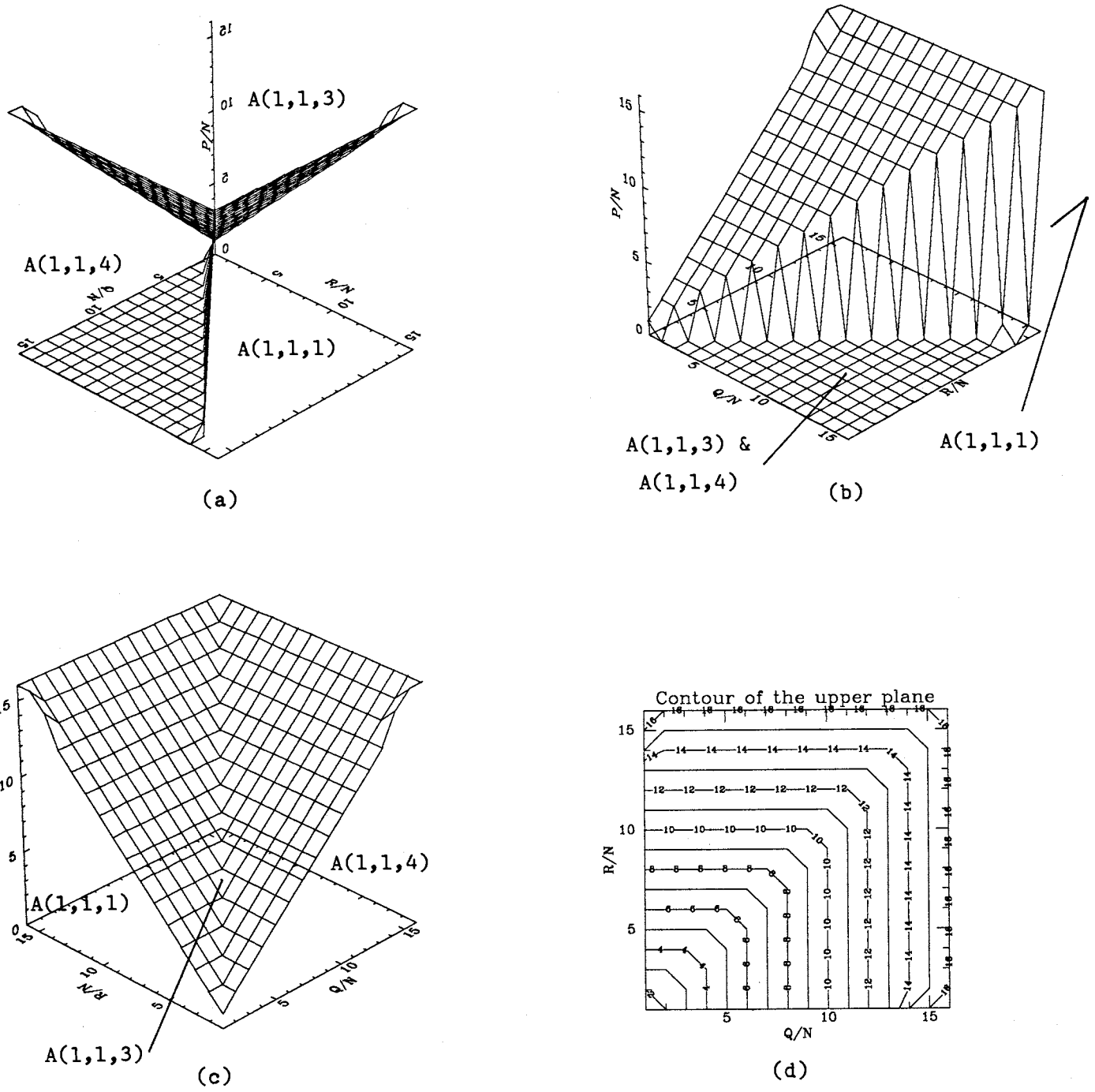


Figure 9: Lowest complexity algorithm as a function of matrix shape; $N = 1024$, $\frac{\tau}{t_c} = 1$ to 1000, $B_m = \infty$, column partitioning, *one-port* communication. (a), (b), and (c) present different views of the regions in P, Q, R space defined by the algorithm of lowest communication complexity. (d) shows the contour of the upper plane.

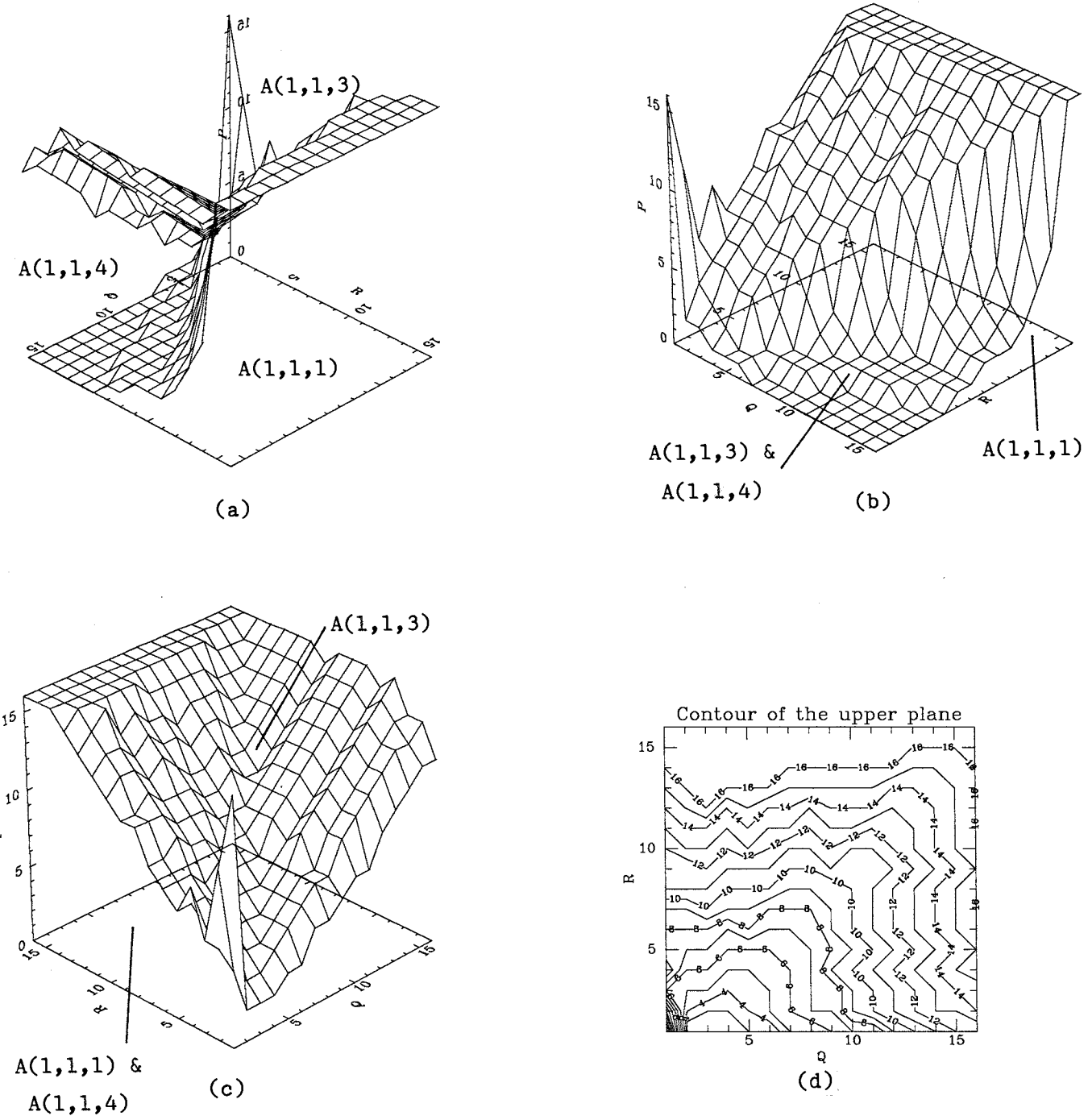
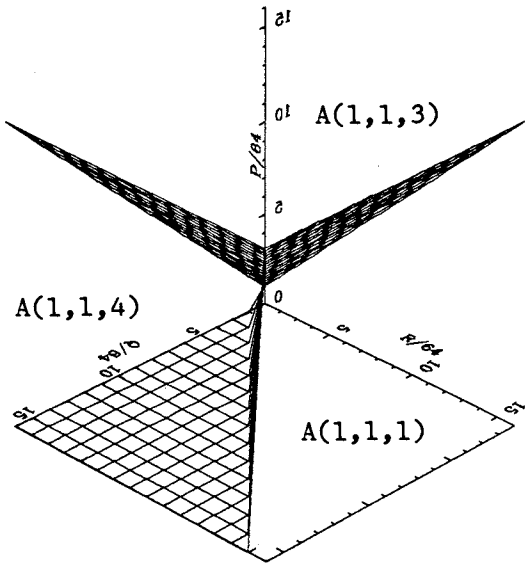
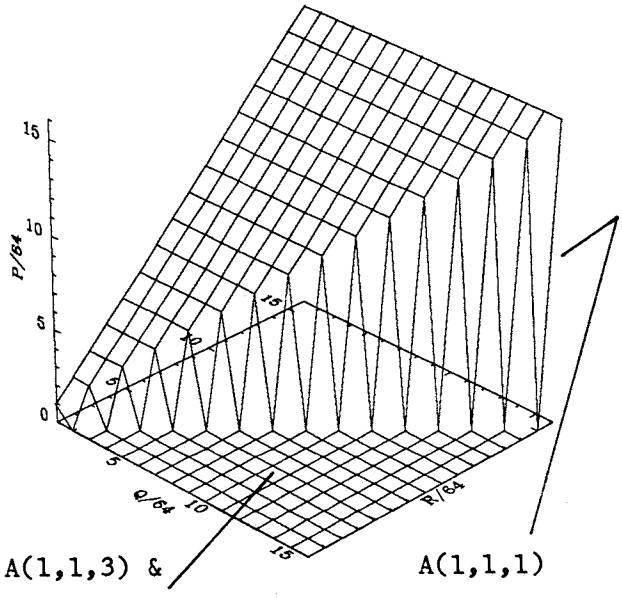


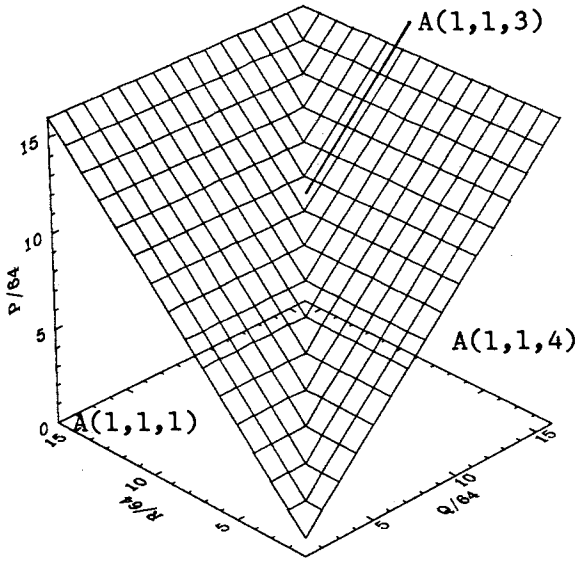
Figure 10: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = t_c = t_a$, $B_m = \infty$, column partitioning, *one-port* communication, $1 \leq P, Q, R \leq N$. (d) shows the contour of the upper plane.



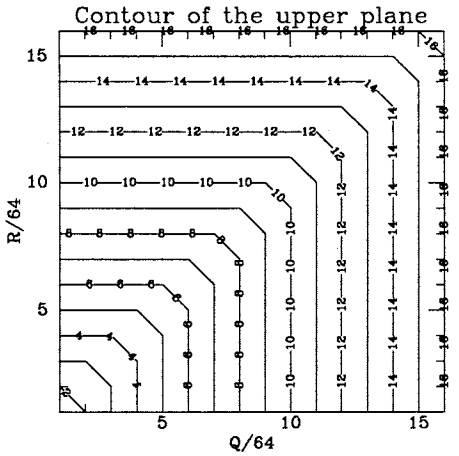
(a)



(b)



(c)



(d)

Figure 11: Lowest complexity algorithm as a function of matrix shape; $N = 1024$, $\tau = t_c = t_a$, $B_m = \infty$, column partitioning, *one-port* communication, $1 \leq P, Q, R \leq N$. (d) shows the contour of the upper plane.

Algorithm	N_{opt}, $B_m \geq B_{opt}$	N_{opt}, $B_m \leq B(\cdot,1,\cdot)$ (see Table 11)
$\mathcal{A}^l(1,1,1)$	$\min\ (R, \sqrt{\frac{PQ(Rt_a-t_c)}{\tau}})$	$R, \quad Rt_a > t_c + \frac{\tau}{B_m}; \quad 1, \quad Rt_a < t_c + \frac{\tau}{B_m}$
$\mathcal{A}^c(1,1,1)$	$\min\ (R, \frac{PQ(2Rt_a-t_c)}{\tau})$	$R, \quad 2Rt_a > t_c + \frac{\tau}{B_m}; \quad 1, \quad 2Rt_a < t_c + \frac{\tau}{B_m}$
$\mathcal{A}^c(1,1,3)$	$\min\ (P, \frac{2QR(2Pt_a-t_c)+(\log N-1)P(R+Q)t_c}{6\tau})$	$\log N_{opt} = \min(P, 1 + \frac{2QR}{P(Q+R)} - \frac{4QRt_a}{(Q+R)(t_c+\tau/B_m)})$
$\mathcal{A}^c(1,1,4)$	$\min\ (Q, \frac{2PQ(2Qt_a-t_c)+(\log N-1)QRt_c}{4\tau})$	$\log N_{opt} = \min(Q, 1 + \frac{2P}{Q} - \frac{4Pt_a}{t_c+\tau/B_m})$
$\mathcal{A}^l(n,1,1)$	$\approx \min\ (R, \frac{PQt_c}{\tau \log^2 N} + \frac{2Rt_a \log^2 N}{t_c})$	$\approx \min(R, \frac{2Rt_a \log^2 N}{t_c+\tau/B_m})$
$\mathcal{A}^c(n,1,1)$	$\approx \min\ (R, \frac{4PQRt_a}{\tau - PQt_c/\log^2 N})$	$\approx \min(R, \frac{4Rt_a \log^2 N}{t_c+\tau/B_m})$
$\mathcal{A}^c(n,1,3)$	$\approx \min\ (P, \frac{4PQRt_a+P(Q+R)t_c/2}{3\tau - QRt_c/\log^2 N})$	$\approx \min(P, P\log^2 N(\frac{4t_a}{t_c+\tau/B_m} + \frac{Q+R}{2QR}))$
$\mathcal{A}^c(n,1,4)$	$\approx \min\ (Q, \frac{4PQRt_a+QRt_c/2}{2\tau - PRt_c/\log^2 N})$	$\approx \min(Q, Q\log^2 N(\frac{4t_a}{t_c+\tau/B_m} + \frac{1}{2P}))$

Table 10: The optimum number of processors.

Algorithm	Buffer
$\mathcal{A}(1,1,1)$	$B(1,1,1) = \frac{PQ}{N}$
$\mathcal{A}(1,1,3)$	$B(1,1,3) = \min(\frac{QR}{N}, \frac{PR}{2N}, \frac{PQ}{2N})$
$\mathcal{A}(1,1,4)$	$B(1,1,4) = \min(\frac{PR}{N}, \frac{QR}{2N})$
$\mathcal{A}(n,1,1)$	$B(n,1,1) = \frac{PQ}{nN}$
$\mathcal{A}(n,1,3)$	$B(n,1,3) = \min(\frac{QR}{nN}, \frac{PR}{2nN}, \frac{PQ}{2nN})$
$\mathcal{A}(n,1,4)$	$B(n,1,4) = \min(\frac{PR}{nN}, \frac{QR}{2nN})$

Table 11: The packet size for which t_c and $\frac{\tau}{B_m}$ have the same coefficient.

for algorithm $\mathcal{A}^c(\cdot,1,1)$. The SBnT$(n,*,N;\cdot,\cdot)$-based (or the nRSBT$(n,*,N;\cdot,\cdot)$-based) algorithm has the same communication complexity as the GCEA$(n,*,N;\cdot,\cdot)$ algorithm, if $B_m \leq \frac{PQ}{nN}$. The reduction in the number of start-ups for the SBnT$(n,*,N;\cdot,\cdot)$ and nRSBT$(n,*,N;\cdot,\cdot)$ algorithms is $\frac{N-1}{n}$ for $B_m \geq \binom{n}{\frac{n}{2}}\frac{PQ}{nN}$ (and larger temporary storage).

The regions for the communication algorithm of lowest complexity are ordered in the same way for the *n-port* communication case as for the *one-port* communication case. The plot which shows the region of the lowest complexity algorithm with $\frac{\tau}{t_c} = \gamma$ and *one-port* communication is the same as that of $\frac{\tau}{t_c} = \frac{\gamma}{n}$ and *n-port* communication, if $B_m \geq B_{opt}$.

Theorem 3 *The communication and arithmetic complexity is the same for matrices embedded in a Boolean cube by binary code encoding and* binary-reflected *Gray code encoding.*

The complexity of broadcasting is independent of of the encoding [10], and so is the matrix transposition time [9]. Note, that the summation order for inner products differ for different algorithms, and that for non-associative operators, such as floating-point addition, some algorithms may loose more precision than others.

4.2 Two-Dimensional Partitioning

The algorithms described for the one-dimensional case have analogues in the two dimensional case. Algorithm $\mathcal{A}(\cdot,1,1)$ that computes A *in-place* by broadcasting C in its two-dimensional form requires broadcasting of elements of C along rows and broadcasting of elements of D along columns. The two broadcasting operations need to be synchronized in order to conserve storage. Cannon [1] has described such an algorithm for mesh configured

multiprocessors (that can be emulated on Boolean cubes) and Dekel et al. [2] described such an algorithm making use of the Boolean cube topology. These algorithms are special cases for particular broadcasting algorithms.

The algorithms corresponding to the four one-dimensional algorithms ($\mathcal{A}(\cdot,1,4)$ has two variations) are

- **Algorithm $\mathcal{A}(\cdot,2,1)$.** Compute A *in-place* by broadcasting of C in the row direction and D in the column direction such that each processor receives all elements of the rows of C mapped into that processor row and all elements of D mapped into the corresponding column of processors. Processor k,l then computes $C(\lfloor \frac{i}{\lceil \frac{P}{N_1} \rceil} \rfloor, *)D(*, \lfloor \frac{j}{\lceil \frac{R}{N_2} \rceil} \rfloor)$ for all i mapped to k and all j mapped to l. The communication operations are *broadcasting* from multiple sources within rows and columns.
- **Algorithm $\mathcal{A}(\cdot,2,2)$.** Transpose C, perform a multiple source *broadcast* along processor rows for the elements of C^T in that processor row, and accumulate inner products for A through multiple sink *reduction* in the column direction (of the processors). The accumulation can be made such that $\frac{P}{N_1}$ elements for each column of D are accumulated in each processor by *all-to-all reduction*. A processor k,l receives $C(*, \lfloor \frac{i}{\lceil \frac{Q}{N_1} \rceil} \rfloor)$ during the broadcasting operation, then computes the product $C(*, \lfloor \frac{i}{\lceil \frac{Q}{N_1} \rceil} \rfloor)D(\lfloor \frac{i}{\lceil \frac{Q}{N_1} \rceil} \rfloor, \lfloor \frac{j}{\lceil \frac{R}{N_2} \rceil} \rfloor)$. The summation over index i is the reduction operation along columns.
- **Algorithm $\mathcal{A}(\cdot,2,3)$.** Transpose C, perform multiple source *broadcasting* of the elements of D within processor rows, accumulate inner products in the column direction. The multiple sink *reduction* is performed such that each processor receives all $\frac{P}{N_2}$ elements of $\frac{R}{N_1}$ distinct columns of D, such that A^T is computed. (Alternatively, the accumulation can be made such that $\frac{P}{\max(N_1,N_2)}$ elements for each column are accumulated in a processor selected such that the proper allocation of A is obtained through a *some-to-all personalized communication* within rows.) Processor k,l computes $C(\lfloor \frac{i}{\lceil \frac{P}{N_2} \rceil} \rfloor, \lfloor \frac{j}{\lceil \frac{Q}{N_1} \rceil} \rfloor)D(\lfloor \frac{j}{\lceil \frac{Q}{N_1} \rceil} \rfloor, *)$ for all i, j such that $\lfloor \frac{i}{\lceil \frac{P}{N_2} \rceil} \rfloor = l$ and $\lfloor \frac{j}{\lceil \frac{Q}{N_1} \rceil} \rfloor = k$.
- **Algorithm $\mathcal{A}(\cdot,2,4)$.** Transpose D, perform a multiple source *broadcasting* of the elements of D^T within processor columns, accumulate the partial inner products for elements of A by multiple sink *reduction* along processor rows such that the elements of at most $\lceil \frac{R}{N_2} \rceil$ columns are accumulated within a processor column. After the transposition and broadcasting processor k,l has the elements $C(\lfloor \frac{i}{\lceil \frac{P}{N_1} \rceil} \rfloor, \lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor)D(\lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor, *)$ for all i such that $\lfloor \frac{i}{\lceil \frac{P}{N_1} \rceil} \rfloor = k$ and j such that $\lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor = l$.
- **Algorithm $\mathcal{A}(\cdot,2,5)$.** Transpose D, perform a multiple source *broadcasting* of the elements of C within processor columns, accumulate inner products for elements of A by multiple sink *reduction* along processor rows, such that each processor receives $\frac{P}{N_2}$ elements of A^T for each of $\frac{R}{N_1}$ columns of D. Processor k,l computes $C(*, \lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor)D(\lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor, \lfloor \frac{i}{\lceil \frac{R}{N_1} \rceil} \rfloor)$ for all i such that $\lfloor \frac{i}{\lceil \frac{R}{N_1} \rceil} \rfloor = k$ and j such that $\lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor = l$.

Figure 12 characterizes the 5 algorithms. The two subscripts in sequence are used to denote the ordinal numbers of block rows and block columns among the $N_1 \times N_2$ partitioned blocks. The "$*$" sign means union of all the block rows (or columns). The superscript denotes the ordinal number of the partial inner product result. The number in the square brackets (eg. [PR] in $\mathcal{A}(\cdot,2,1)$) is the minimum maximum number of processors to minimize the arithmetic time for each algorithm. Algorithm $\mathcal{A}(\cdot,2,2)$ has a matrix transpose in addition to the communication of C as in algorithm $\mathcal{A}(\cdot,2,1)$. But, unlike in the one-dimensional case algorithm $\mathcal{A}(\cdot,2,2)$ may have a higher processor utilization than algorithm $\mathcal{A}(\cdot,2,1)$.

In algorithm $\mathcal{A}(\cdot,2,2)$ the operation $C(*,*)D(*, \lfloor \frac{j}{\lceil \frac{R}{N_2} \rceil} \rfloor)$ requires a summation, which is made by *all-to-all reduction* such that processor k,l receives the product $C(\lfloor \frac{i}{\lceil \frac{P}{N_1} \rceil} \rfloor, *)D(*, \lfloor \frac{j}{\lceil \frac{R}{N_2} \rceil} \rfloor)$. The reduction in the first version of algorithm $\mathcal{A}(\cdot,2,3)$ is made such that processor k,l receives $C(\lfloor \frac{i}{\lceil \frac{P}{N_2} \rceil} \rfloor, *)D(*, \lfloor \frac{j}{\lceil \frac{R}{N_1} \rceil} \rfloor)$ for all i such that $\lfloor \frac{i}{\lceil \frac{P}{N_2} \rceil} \rfloor = l$ and j such that $\lfloor \frac{j}{\lceil \frac{R}{N_1} \rceil} \rfloor = k$. The local computations in algorithm $\mathcal{A}(\cdot,2,4)$ are the same

$$
\begin{aligned}
\mathcal{A}(\cdot,2,1): &\quad C_{kl}, D_{kl} \xrightarrow{\text{brd. C, } \leftrightarrow} C_{k*}, D_{kl} \xrightarrow{\text{brd. D, } \updownarrow} C_{k*}, D_{*l} \xrightarrow{\text{mpy., [PR]}} A_{kl} \\
\mathcal{A}(\cdot,2,2): &\quad C_{kl}, D_{kl} \xrightarrow{\text{txp. C, } \swarrow} C_{lk}, D_{kl} \xrightarrow{\text{brd. C, } \leftrightarrow} C_{*k}, D_{kl} \xrightarrow{\text{mpy., [QR]}} A^k_{*l} \xrightarrow{\text{red. A, } \updownarrow} A_{kl} \\
\mathcal{A}(\cdot,2,3): &\quad C_{kl}, D_{kl} \xrightarrow{\text{txp. C, } \swarrow} C_{lk}, D_{kl} \xrightarrow{\text{brd. D, } \leftrightarrow} C_{lk}, D_{k*} \xrightarrow{\text{mpy., [PQ]}} A^k_{l*} \xrightarrow{\text{red. A, } \updownarrow} A_{lk} \xrightarrow{\text{txp. A, } \swarrow} A_{kl} \\
\mathcal{A}(\cdot,2,4): &\quad C_{kl}, D_{kl} \xrightarrow{\text{txp. D, } \swarrow} C_{kl}, D_{lk} \xrightarrow{\text{brd. D, } \updownarrow} C_{kl}, D_{l*} \xrightarrow{\text{mpy., [PQ]}} A^l_{k*} \xrightarrow{\text{red. A, } \leftrightarrow} A_{kl} \\
\mathcal{A}(\cdot,2,5): &\quad C_{kl}, D_{kl} \xrightarrow{\text{txp. D, } \swarrow} C_{kl}, D_{lk} \xrightarrow{\text{brd. C, } \updownarrow} C_{*l}, D_{lk} \xrightarrow{\text{mpy., [QR]}} A^l_{*k} \xrightarrow{\text{red. A, } \leftrightarrow} A_{lk} \xrightarrow{\text{txp. A, } \swarrow} A_{kl}
\end{aligned}
$$

Figure 12: Notation summary of the algorithms for two-dimensional partitioning.

as in algorithm $\mathcal{A}(\cdot,2,3)$. The difference is that reductions corresponding to summation on j are performed along processor rows, instead of columns as in algorithm $\mathcal{A}(\cdot,2,3)$. One transposition of D suffice instead of a transposition of both C and A in algorithm $\mathcal{A}(\cdot,2,3)$.

4.2.1 Arithmetic Complexity

The arithmetic complexities and temporary storage requirements of algorithms $\mathcal{A}(\cdot,2,1)$ – $\mathcal{A}(\cdot,2,5)$ are given in Table 12.

Lemma 4 *The arithmetic complexity of all five algorithms is the same if P, Q, and R all are multiples of N_1 and N_2.*

Theorem 4 *For P, Q, $R \leq N_1$, N_2, the arithmetic complexity is minimized if the normal of the processing plane is aligned with the axis having the fewest elements.*

The lemma and the theorem can be proved by direct evaluation. For instance, if $Q < P, R$, then algorithm $\mathcal{A}(\cdot,2,1)$ for which the processing is in the P, R plane is of lowest arithmetic complexity, and if $P < Q, R$ then algorithms $\mathcal{A}(\cdot,2,2)$ and $\mathcal{A}(\cdot,2,5)$ have the lowest arithmetic complexity, and if $R < P, Q$ then algorithms $\mathcal{A}(\cdot,2,3)$ and $\mathcal{A}(\cdot,2,4)$ are the algorithms of choice. The first case is similar to an outer product, the second to producing a row vector, and the third to producing a column vector.

4.2.2 Communication Routines and Their Complexities

Algorithm $\mathcal{A}(\cdot,2,1)$:

Algorithm $\mathcal{A}(\cdot,2,1)$ carried out as a sequence of *one-to-all broadcasts* for a row and a column with a multiplication phase between each such operation constitutes an *outer-product* algorithm, but becomes an *inner-product* algorithm, if it is implemented by *multiple source broadcasting* followed by the multiplication operations. A linear array type algorithm used for *one-to-all broadcasting* conserve storage, but requires a total of $O(N_1^2, N_2^2)$ start-ups, and data transfer times. In order that *multiple source broadcasting* also conserve storage the computations for successive outer products must be pipelined, which requires a proper alignment of the operands. Alignment of the two matrices C and D may be explicit or implicit. We describe one algorithm of the latter type suitable for MIMD architectures, and two algorithms of the first type. For a few variations see [7].

With the mapping of the matrices defined previously, only the diagonal blocks have the proper index sets for the multiplication operation assuming $N_1 = N_2 = \sqrt{N}$. The outer products can all be initiated at the same time. The computations proceed from the main diagonal towards the bottom and right for each outer product. It can be shown that for $\sqrt{N}$ being odd, $2\sqrt{N} - 1$ steps suffice to complete the algorithm with constant size buffer. Each step consists of communications of two block matrices, one block matrix multiplication and one block matrix addition. For $\sqrt{N}$ being even, we can put one more buffer at the boundary processors and delay

Algorithm	Number of Arithmetic operations	Max. no. of processors
$\mathcal{A}(\cdot,2,1)$	$2Q\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	PR
$\mathcal{A}(\cdot,2,2)$	$P(2\lceil\frac{Q}{N_1}\rceil-1)\lceil\frac{R}{N_2}\rceil+\lceil\frac{R}{N_2}\rceil\sum_{i=1}^{n_1}\lceil\frac{P}{2^i}\rceil+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	QR
$\mathcal{A}(\cdot,2,3)$	$R(2\lceil\frac{Q}{N_1}\rceil-1)\lceil\frac{P}{N_2}\rceil+\lceil\frac{P}{N_2}\rceil\sum_{i=1}^{n_1}\lceil\frac{R}{2^i}\rceil+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	PQ
$\mathcal{A}(\cdot,2,4)$	$R(2\lceil\frac{Q}{N_2}\rceil-1)\lceil\frac{P}{N_1}\rceil+\lceil\frac{P}{N_1}\rceil\sum_{i=1}^{n_2}\lceil\frac{R}{2^i}\rceil+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	PQ
$\mathcal{A}(\cdot,2,5)$	$P(2\lceil\frac{Q}{N_2}\rceil-1)\lceil\frac{R}{N_1}\rceil+\lceil\frac{R}{N_1}\rceil\sum_{i=1}^{n_2}\lceil\frac{P}{2^i}\rceil+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	QR

Algorithm	GEMV			inner-products	
	no. of	no. of AXPY	v-length	no. of	order
$\mathcal{A}(\cdot,2,1)$	$\frac{R}{N_2}$	Q	$\frac{P}{N_1}$	$\frac{P}{N_1}\frac{R}{N_2}$	Q
$\mathcal{A}(\cdot,2,2)$	$\frac{R}{N_2}$	$\frac{Q}{N_1}$	P	$P\frac{R}{N_2}$	$\frac{Q}{N_1}$
$\mathcal{A}(\cdot,2,3)$	R	$\frac{Q}{N_1}$	$\frac{P}{N_2}$	$R\frac{P}{N_2}$	$\frac{Q}{N_1}$
$\mathcal{A}(\cdot,2,4)$	R	$\frac{Q}{N_2}$	$\frac{P}{N_1}$	$R\frac{P}{N_1}$	$\frac{Q}{N_2}$
$\mathcal{A}(\cdot,2,5)$	$\frac{R}{N_1}$	$\frac{Q}{N_2}$	P	$P\frac{R}{N_1}$	$\frac{Q}{N_2}$

Table 12: The local arithmetic operations for the two-dimensional partitioning.

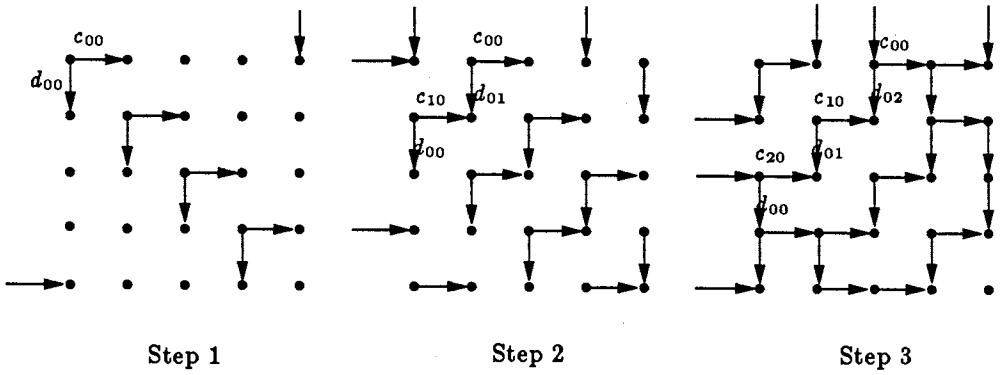


Figure 13: Computing $A \leftarrow C \times D + E$ by a pipelined outer product algorithm.

sending the message to the opposite side of the boundary processors. $2\sqrt{N}+1$ steps are sufficient to complete the algorithm. For $N_1 \neq N_2$ and for instance $N_1 > N_2$ the situation $N_1 = N_2$ can be simulated by a further partitioning in the second dimension such that $\frac{N_1}{N_2}$ ***virtual partitions*** are assigned to each real partition.

Figure 13 illustrates some of the steps. A total of $2\sqrt{N}+1$ steps are needed. Each step requires only nearest neighbor communication. For a MIMD mode of operation the coding of the algorithm is straightforward. In a SIMD mode masking is required to define the set of active processors in each step. Notice that if more than one matrix multiplication is performed, the subsequent matrix multiplications can be initiated every $\sqrt{N}$ cycles from the main diagonal without any communication and computation conflict. Hence, a total of K matrix multiplications only require $(K+1)\sqrt{N}+1$ steps compared to approximately $\frac{3K}{2}\sqrt{N}$ steps by Cannon's algorithm as described next.

Cannon [1,2] describes a matrix multiplication algorithm suitable for SIMD architectures configured as two-dimensional arrays. The algorithm consists of two phases; an alignment phase and a multiplication phase including $2Q$ (or $\max(N_1, N_2)$ for both N_1 and N_2 being powers of 2) communication steps for *one-port communication*. During the alignment phase row i, $i = \{0, 1, 2, \ldots, P-1\}$ of C is rotated left i positions, i.e. $C(i,j) \leftarrow C(i, j+i \bmod Q)$, and column j, $j = \{0, 1, 2, \ldots, Q-1\}$ rotated up j positions, $D(i,j) \leftarrow D(i+j \bmod Q, j)$. The alignment of the matrices allows all processors to participate in each step of the multiplication phase. After the

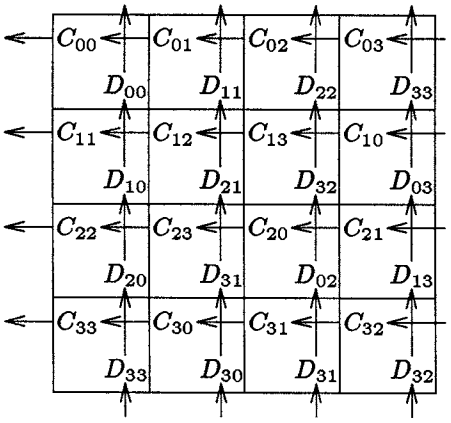


Figure 14: Matrix multiplication on a mesh according to [1,2].

alignment phase the matrix C is stored such that diagonals are aligned with columns of the array, and diagonals of D aligned with rows of the array. During the multiplication phase the matrix C is rotated left one step at a time, and the matrix D rotated up one step at a time. Figure 14 shows the data structures after the alignment phase. If the alignment is carried out as a sequence of shifts, and the CRA algorithm is used, then for *one-port* communication the row shifts use CRA$(1, *, N_2; C, \frac{P}{N_1}\frac{Q}{N_2})$ and require $\lfloor \frac{N_2}{2} \rfloor$ steps, and the column shifts require $\lfloor \frac{N_1}{2} \rfloor$ steps by CRA$(1, *, N_1; D, \frac{Q}{N_1}\frac{R}{N_2})$. The total data volume communicated across an edge for the alignment of C is $\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2} \rceil \lfloor \frac{N_2}{2} \rfloor$. For the alignment of D the number of element transfers is $\lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \lfloor \frac{N_1}{2} \rfloor$. There are $\max(N_1, N_2) - 1$ steps of the algorithm, and each such step requires communication of $\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2} \rceil$ elements for the rotation of C and $\lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2} \rceil$ elements for the rotation of D. If $N_2 > N_1$, then $\frac{N_2}{N_1}$ shifts of C are required for each shift of D. In a Boolean cube the number of communication steps for the alignment of C and D can be reduced to at most $2 \log N_1$ and $2 \log N_2$ steps, respectively [7].

It is also possible to base a matrix multiplication algorithm on the GCEA algorithm. As was the case with the CRA algorithm an alignment is required between C and D, and the movement synchronized. The use of the GCEA algorithm is equivalent to the following recursive procedure. Let C be partitioned into 4 blocks: C_{00}, C_{01}, C_{10}, and C_{11}. Similarly, let D be partitioned into D_{00}, D_{01}, D_{10} and D_{11} of appropriate sizes. Then, an exchange of blocks C_{10} and C_{11} and of blocks D_{01} and D_{11}, respectively, brings block matrices into positions such that four independent matrix multiplications can be performed on matrices of half the number of rows and columns of the original problem. To complete the matrix multiplication an exchange of blocks in the same row is made for C and in the same column for D, followed by a new multiplication of half sized matrices. If there is a different number of partitions in the two dimensions, say $N_1 > N_2$, then $\frac{N_1}{N_2}$ virtual partitions are added to each partition in the second dimension. The virtual partitions are local to a processor. For each communication in the second dimension there are $\frac{N_1}{N_2}$ communications in the first dimension. For the multiplication phase there are $N_2 - 1$ communication steps for C in the second dimension, each step communicating $\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2} \rceil$ elements. For D there are $N_1 - 1$ steps of $\lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2} \rceil$ each. The communications are between adjacent processors, if the matrices are embedded in the Boolean cube by a binary encoding of partitions.

Dekel [2] describes the above algorithm in detail for $P = Q = R = \sqrt{N}$. For $Q = 2^{\alpha_1} N_1 = 2^{\alpha_2} N_2$ the algorithm is directly portable. A certain number of the low order bits are mapped to the same processor for consecutive partitioning, and communications corresponding to those bits are internal to a processor. The set-up phase requires $\log N_2$ and $\log N_1$ communications respectively.

It is also possible to design a matrix multiplication algorithm based on the SBT-b algorithm. With this algorithm the number of communication steps for the multiplication phase is reduced to $\log N_2$ and $\log N_1$ respectively. The need for temporary storage is equal to $\frac{1}{2}Q(\lceil \frac{P}{N_1} \rceil + \lceil \frac{R}{N_2} \rceil)$. The alignment described for Dekel's algorithm is applicable. With a temporary storage of $Q(\lceil \frac{P}{N_1} \rceil + \lceil \frac{R}{N_2} \rceil)$, the alignment step can be eliminated. In this case, the computation is performed only after all necessary communications are done.

For *n-port* communication we only consider the SBnT-b and nRSBT-b routings. The maximum edge load is reduced compared to the *one-port* case. The communication time is

$$\min \left(\frac{(N_2-1)}{n_2} \left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil t_c + \sum_{i=1}^{n_2} \left\lceil \binom{n_2}{i} \left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil \frac{1}{n_2 B_m} \right\rceil \tau, \right.$$
$$\left. \frac{(N_1-1)}{n_1} \left\lceil \frac{Q}{N_1} \right\rceil \left\lceil \frac{R}{N_2} \right\rceil t_c + \sum_{i=1}^{n_1} \left\lceil \binom{n_1}{i} \left\lceil \frac{Q}{N_1} \right\rceil \left\lceil \frac{R}{N_2} \right\rceil \frac{1}{n_1 B_m} \right\rceil \tau \right).$$

The reduction in the element transfer time is by a factor of $\frac{n}{2}$. The number of start-ups is also reduced by the same factor if $B_m < \frac{2}{n}\lceil \frac{PQ}{N} \rceil$.

Algorithm $\mathcal{A}(\cdot,2,2)$:

For the transposition of C we use a transpose algorithm as described in [4,3]. The multiple source *broadcasting* of C^T can be made with algorithm CRA$(1,*,N_2;C^T,\frac{Q}{N_1}\frac{P}{N_2})$, GCEA$(1,*,N_2;C^T,\frac{Q}{N_1}\frac{P}{N_2})$, or SBT-b$(1,*,N_2;C^T,\frac{Q}{N_1}\frac{P}{N_2})$ for *one-port* communication. The reduction can be carried out by the same algorithms, executed in reverse order, for instance by SBT-b$(1,*,N_1;A,\frac{P}{N_1}\frac{R}{N_2})$.

Let N_{max} and N_{min} be $\max(N_1,N_2)$ and $\min(N_1,N_2)$, respectively. The transposition of the $P \times Q$ matrix partitioned by a $N_1 \times N_2$ processor hypercube can be viewed as $2\log_2 N_{min}$ steps of matrix transposition exchanging $\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2} \rceil$ elements between processors, $\log_2 \frac{N_{max}}{N_{min}}$ steps in which $\frac{1}{2}\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2} \rceil$ elements are exchanged between processors, and the transposition of $\frac{N_{max}}{N_{min}}$ local matrices of size $\frac{P}{N_{max}} \frac{Q}{N_{max}}$.

$$\begin{aligned}
T_{txp}(1,N_1,N_2;P,Q) &= 2\log N_{min} \left(\left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil t_c + \left\lceil \left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil \frac{1}{B_m} \right\rceil \tau \right) \\
&\quad + \log \frac{N_{max}}{N_{min}} \left(\frac{1}{2} \left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil t_c + \left\lceil \left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil \frac{1}{2B_m} \right\rceil \tau \right) \\
&= \left(\frac{n}{2} + \min(n_1,n_2) \right) \left(\left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil t_c + \tau \right), \qquad B_m \geq \left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil \\
&\leq n \left(\left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil t_c + \tau \right), \qquad B_m \geq \left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil .
\end{aligned}$$

With *n-port* communication, pipelining can be used. The communication time becomes

$$\begin{aligned}
T_{txp}(n,N_1,N_2;P,Q) &= \left(\left\lceil \left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil \frac{1}{B} \right\rceil + n - 1 \right) (Bt_c + \tau) \\
&= \left(\sqrt{\left\lceil \frac{P}{N_1} \right\rceil \left\lceil \frac{Q}{N_2} \right\rceil t_c} + \sqrt{(n-1)\tau} \right)^2, \qquad B_{opt} = \sqrt{\frac{\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2} \rceil \tau}{(n-1)t_c}}.
\end{aligned}$$

In the above expressions for the time of a transpose we have ignored the time for copying, which indeed is significant [4]. However, the expression is quite complex and does not qualitatively affect the results. The total communication complexity for algorithm $\mathcal{A}(\cdot,2,2)$ is given in Tables 13, 14, 15, and 16. Notice that in Tables 14, 15 and 16, it is assumed that $P,Q,R \geq N_1,N_2$. For arbitrary P,Q,R, the communication complexity is very complicated. We give the communication complexity of $\mathcal{A}^c(1,2,2)$ for arbitrary P,Q,R as an example. The complexities of other algorithms can be derived similarly. The number of elements transferred is

$$\max(\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2} \rceil, \lceil \frac{Q}{N_1} \rceil \lceil \frac{P}{N_2} \rceil) n + (\max(k_{r2}-k_{p2},0) + 2^{k_{p2}} - 1) \lceil \frac{Q}{N_1} \rceil \lceil \frac{P}{N_2} \rceil + (\max(k_{q1}-k_{p1},0) + 2^{k_{p1}} - 1) \lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil,$$

where $k_{r2} = \min(\log R, n_2)$, $k_{p2} = \min(\log P, n_2)$, $k_{q1} = \min(\log Q, n_1)$ and $k_{p1} = \min(\log P, n_1)$. The number of start-ups is

$$\left\lceil \frac{1}{B_m} \max(\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2} \rceil, \lceil \frac{Q}{N_1} \rceil \lceil \frac{P}{N_2} \rceil) \right\rceil n + \max(k_{r2}-k_{p2},0) \left\lceil \frac{1}{B_m} \lceil \frac{Q}{N_1} \rceil \lceil \frac{P}{N_2} \rceil \right\rceil + \sum_{i=0}^{k_{p2}-1} \left\lceil \frac{2^i}{B_m} \lceil \frac{Q}{N_1} \rceil \lceil \frac{P}{N_2} \rceil \right\rceil$$

Model	Algorithm	Communication operations
one-port	$\mathcal{A}^{t\times t}(1,2,1)$	CRA-b$(1,*,N_2;C,\frac{P}{N_1}\frac{Q}{N_2})$ + CRA-b$(1,*,N_1;D,\frac{Q}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{t\times t}(1,2,1)$	GCEA-b$(1,*,N_2;C,\frac{P}{N_1}\frac{Q}{N_2})$ + GCEA-b$(1,*,N_1;D,\frac{Q}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{t\times t}(1,2,2)$	TXP$(1,N_1,N_2;P,Q)$ + CRA-b$(1,*,N_2;C^T,\frac{Q}{N_1}\frac{P}{N_2})$ + CRA-b$(1,*,N_1;A,\frac{P}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{t\times t}(1,2,3)$	TXP$(1,N_1,N_2;P,Q)$ + CRA-b$(1,*,N_2;D,\frac{Q}{N_1}\frac{R}{N_2})$ + CRA-b$(1,*,N_1;A^T,\frac{R}{N_1}\frac{P}{N_2})$ + TXP$(1,N_1,N_2;R,P)$
	$\mathcal{A}^{t\times t}(1,2,4)$	TXP$(1,N_1,N_2;Q,R)$ + CRA-b $(1,*,N_1;D^T,\frac{R}{N_1}\frac{Q}{N_2})$ + CRA-b $(1,*,N_2;A,\frac{P}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{t\times t}(1,2,5)$	TXP$(1,N_1,N_2;Q,R)$ + CRA-b$(1,*,N_1;C,\frac{P}{N_1}\frac{Q}{N_2})$ + CRA-b$(1,*,N_2;A^T,\frac{R}{N_1}\frac{P}{N_2})$ + TXP$(1,N_1,N_2;R,P)$
	$\mathcal{A}^{c}(1,2,1)$	SBT-b$(1,*,N_2;C,\frac{P}{N_1}\frac{Q}{N_2})$ + SBT-b$(1,*,N_1;D,\frac{Q}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{c}(1,2,2)$	TXP$(1,N_1,N_2;P,Q)$ + SBT-b$(1,*,N_2;C^T,\frac{Q}{N_1}\frac{P}{N_2})$ + SBT-b$(1,*,N_1;A,\frac{P}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{c}(1,2,3)$	TXP$(1,N_1,N_2;P,Q)$ + SBT-b$(1,*,N_2;D,\frac{Q}{N_1}\frac{R}{N_2})$ + SBT-b$(1,*,N_1;A^T,\frac{R}{N_1}\frac{P}{N_2})$ + TXP$(1,N_1,N_2;R,P)$
	$\mathcal{A}^{c}(1,2,4)$	TXP$(1,N_1,N_2;Q,R)$ + SBT-b $(1,*,N_1;D^T,\frac{R}{N_1}\frac{Q}{N_2})$ + SBT-b $(1,*,N_2;A,\frac{P}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{c}(1,2,5)$	TXP$(1,N_1,N_2;Q,R)$ + SBT-b$(1,*,N_1;C,\frac{P}{N_1}\frac{Q}{N_2})$ + SBT-b$(1,*,N_2;A^T,\frac{R}{N_1}\frac{P}{N_2})$ + TXP$(1,N_1,N_2;R,P)$
n-port	$\mathcal{A}^{c}(n,2,1)$	SBnT-b$(n_2,*,N_2;C,\frac{P}{N_1}\frac{Q}{N_2})$ + SBnT-b$(n_1,*,N_1;D,\frac{Q}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{c}(n,2,2)$	TXP$(n,N_1,N_2;P,Q)$ + SBnT-b$(n_2,*,N_2;C^T,\frac{Q}{N_1}\frac{P}{N_2})$ + SBnT-b$(n_1,*,N_1;A,\frac{P}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{c}(n,2,3)$	TXP$(n,N_1,N_2;P,Q)$ + SBnT-b$(n_2,*,N_2;D,\frac{Q}{N_1}\frac{R}{N_2})$ + SBnT-b$(n_1,*,N_1;A^T,\frac{R}{N_1}\frac{P}{N_2})$ + TXP$(n,N_1,N_2;R,P)$
	$\mathcal{A}^{c}(n,2,4)$	TXP$(n,N_1,N_2;Q,R)$ +SBnT-b$(n_1,*,N_1;D^T,\frac{R}{N_1}\frac{Q}{N_2})$ + SBnT-b$(n_2,*,N_2;A,\frac{P}{N_1}\frac{R}{N_2})$
	$\mathcal{A}^{c}(n,2,5)$	TXP$(n,N_1,N_2;Q,R)$ + SBnT-b$(n_1,*,N_1;C,\frac{P}{N_1}\frac{Q}{N_2})$ + SBnT-b$(n_2,*,N_2;A^T,\frac{R}{N_1}\frac{P}{N_2})$ + TXP$(n,N_1,N_2;R,P)$

Table 13: Communication operators.

$$+\max(k_{q1}-k_{p1},0)\left\lceil\frac{1}{B_m}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\right\rceil+\sum_{i=0}^{k_{p1}-1}\left\lceil\frac{2^i}{B_m}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\right\rceil.$$

The minimum number of start-ups is $n+\max(k_{r2},k_{p2})+\max(k_{q1},k_{p1})$.

Algorithm $\mathcal{A}(\cdot,2,3)$:

For version 1 of algorithm $\mathcal{A}(\cdot,2,3)$ two matrix transposition operations are needed, one on the matrix C, and one on the matrix A. For the multiple source *broadcasting* of D within rows algorithm SBT-b$(1,*,N_2;D,\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil)$ can be used. For the multiple sink *reduction* SBT-b$(1,*,N_1;A^T,\lceil\frac{P}{N_2}\rceil\lceil\frac{R}{N_1}\rceil)$ is a possible choice. The complexity of the SBT-b$(1,*,\cdot;\cdot,\cdot)$ algorithm is the same as that of the CRA$(1,*,\cdot;\cdot,\cdot)$ or GCEA$(1,*,\cdot;\cdot,\cdot)$ if the buffer size $B_m\leq\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$, and $B_m\leq\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$, respectively. With *n-port* communication the SBnT-b$(n,*,\cdot;\cdot,\cdot)$ or the nRSBT-b$(n,*,\cdot;\cdot,\cdot)$ routings are used instead.

For version 2 of algorithm $\mathcal{A}(\cdot,2,3)$ the *reduction* is carried out such that the inner products for the set of $\frac{P}{N_2}$ rows of A allocated to a column of processors are accumulated with contiguous sets of $\frac{P}{N_1}$ inner products per processor in the same column, and in the processor row in which the inner product shall finally reside. The divide and conquer strategy is applied to $\frac{P}{N_2}$ (if $N_1>N_2$), and repeated R times, since every processor column contains every column of D. After this reduction operation the elements of A are in the proper processor row, but all elements of a row are confined to one or a few processors in that row. The desired distribution is obtained by *one-to-all personalized communication*, or *some-to-all personalized communication* if $N_1<N_2$. For *one-port* communication we choose the SBT-p$(1,*,\cdot;\cdot,\cdot)$ algorithm, and for *n-port* communication either the SBnT-p$(n,*,\cdot;\cdot,\cdot)$ or nRSBT-p$(n,*,\cdot;\cdot,\cdot)$ algorithm.

Algorithm	Element transfers	start-ups	*min* start-ups
$\mathcal{A}^c(1,2,1)$	$(N_2-1)\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil$ $+(N_1-1)\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$\sum_{i=0}^{n_2-1}\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{2^i}{B_m}\rceil$	n
$\mathcal{A}^c(1,2,2)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil n+\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_2-1)$ $+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_1-1)$	$\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{1}{B_m}\rceil n+\sum_{i=0}^{n_2-1}\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{2^i}{B_m}\rceil$	$2n$
$\mathcal{A}^c(1,2,3)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil n+\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_2-1)$ $+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_1-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil n$	$\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{1}{B_m}\rceil n+\sum_{i=0}^{n_2-1}\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{2^i}{B_m}\rceil+\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{1}{B_m}\rceil n$	$3n$
$\mathcal{A}^c(1,2,4)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil n+\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil(N_1-1)$ $+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_2-1)$	$\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{1}{B_m}\rceil n+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_2-1}\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{2^i}{B_m}\rceil$	$2n$
$\mathcal{A}^c(1,2,5)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil n+\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil(N_1-1)$ $+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_2-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil n$	$\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{1}{B_m}\rceil n+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_2-1}\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{2^i}{B_m}\rceil+\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{1}{B_m}\rceil n$	$3n$

Table 14: The communication complexity using two-dimensional partitioning.

Algorithm	B_{opt}	T_{min}
$\mathcal{A}^c(1,2,1)$	$\frac{N_2}{2}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil,\frac{N_1}{2}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$2Q\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_a+\{(N_2-1)\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil+(N_1-1)\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil)\}t_c+n\tau$
$\mathcal{A}^c(1,2,2)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil,\frac{N_2}{2}\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil,$ $\frac{N_1}{2}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$\{(2\lceil\frac{Q}{N_1}\rceil-1)\lceil\frac{R}{N_2}\rceil P+\sum_{i=1}^{n_1}\lceil\frac{R}{N_2}\rceil\lceil\frac{P}{2^i}\rceil+\lceil\frac{R}{N_2}\rceil\lceil\frac{P}{N_1}\rceil\}t_a+2n\tau$ $+\{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil n+\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_2-1)+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_1-1)\}t_c$
$\mathcal{A}^c(1,2,3)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil,\frac{N_2}{2}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil,$ $\frac{N_1}{2}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil,\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil$	$\{(2\lceil\frac{Q}{N_1}\rceil-1)\lceil\frac{P}{N_2}\rceil R+\sum_{i=1}^{n_1}\lceil\frac{P}{N_2}\rceil\lceil\frac{R}{2^i}\rceil+\lceil\frac{P}{N_2}\rceil\lceil\frac{R}{N_1}\rceil\}t_a+3n\tau$ $+\{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil n+\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_2-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_1-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil n\}t_c$
$\mathcal{A}^c(1,2,4)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil,\frac{N_1}{2}\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil,$ $\frac{N_2}{2}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$\{(2\lceil\frac{Q}{N_2}\rceil-1)\lceil\frac{P}{N_1}\rceil R+\sum_{i=1}^{n_2}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{2^i}\rceil+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\}t_a+2n\tau$ $+\{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil n+\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil(N_1-1)+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_2-1)\}t_c$
$\mathcal{A}^c(1,2,5)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil,\frac{N_1}{2}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil,$ $\frac{N_2}{2}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil,\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil$	$\{(2\lceil\frac{Q}{N_2}\rceil-1)\lceil\frac{R}{N_1}\rceil P+\sum_{i=1}^{n_2}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{2^i}\rceil+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\}t_a+3n\tau$ $+\{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil n+\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil(N_1-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_2-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil n\}t_c$

Table 15: The communication complexity for *optimum* buffer sizes, two-dimensional partitioning, and *one-port* communication.

Algorithm	B_{opt}	T_{min} – arith. time
$\mathcal{A}^c$(n,2,1)	$\sqrt{\frac{2}{\pi}}\frac{N_2}{n_2^{3/2}}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil$, $\sqrt{\frac{2}{\pi}}\frac{N_1}{n_1^{3/2}}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$\max(\frac{N_2-1}{n_2}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil t_c + n_2\tau$, $\frac{N_1-1}{n_1}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_c + n_1\tau)$
$\mathcal{A}^c$(n,2,2)	$\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\frac{2}{\pi}}\frac{N_2}{n_2^{3/2}}\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil$, $\sqrt{\frac{2}{\pi}}\frac{N_1}{n_1^{3/2}}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$n\tau + (\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{N_2-1}{n_2} + \lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{N_1-1}{n_1})t_c$
$\mathcal{A}^c$(n,2,3)	$\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\frac{2}{\pi}}\frac{N_2}{n_2^{3/2}}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$, $\sqrt{\frac{2}{\pi}}\frac{N_1}{n_1^{3/2}}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil$	$n\tau + (\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{N_2-1}{n_2} + \lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{N_1-1}{n_1})t_c$
$\mathcal{A}^c$(n,2,4)	$\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\frac{2}{\pi}}\frac{N_1}{n_1^{3/2}}\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil$, $\sqrt{\frac{2}{\pi}}\frac{N_2}{n_2^{3/2}}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$n\tau + (\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{N_1-1}{n_1} + \lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{N_2-1}{n_2})t_c$
$\mathcal{A}^c$(n,2,5)	$\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\frac{2}{\pi}}\frac{N_1}{n_1^{3/2}}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil$, $\sqrt{\frac{2}{\pi}}\frac{N_2}{n_2^{3/2}}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil$	$n\tau + (\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{N_1-1}{n_1} + \lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{N_2-1}{n_2})t_c$

Table 16: The communication complexity for *optimum* buffer sizes, two-dimensional partitioning, and *n-port* communication. The arithmetic time is the same as in the *one-port* case and is omitted from the last column.

4.2.3 Summary and Comparison

Tables 13 to 19 summarize the communication and arithmetic complexities. Note that if P, Q and R are multiples of N_1 and N_2, then the arithmetic complexities of the algorithms are the same, and indeed the same as for a one-dimensional partitioning.

Theorem 5 *For P, Q, R multiples of N_1, N_2 or P, Q, $R \leq N_1$, N_2, and two-dimensional partitioning, the arithmetic and communication complexities are minimized if the processing plane is the same as the plane with the maximum number of matrix elements, and the aspect ratio of the processing domain is the same as the aspect ratio of the corresponding matrix domain. The optimum aspect ratio of the processing domain is independent of architectural parameters such as τ, t_c, or t_a.*

Figure 15 shows the optimum values of $\frac{N_1}{N_2}$ for algorithm $\mathcal{A}$(1,2,1) as a function of $\frac{P}{R}$. The same Figure also applies to the other algorithms by the appropriate relabeling of the axis. Figure 16 shows the measured and predicted communication times of algorithm $\mathcal{A}^c(1,2,1)$ on the iPSC as a function of $\frac{N_1}{N_2}$ for different values of $\frac{P}{R} = \{2, 8, 32\}$. The measured minima are the same as the predicted.

With a significant communication time it might be preferable to use fewer than the maximum possible number of processors. Table 18 shows the approximate optimum values of N for algorithms $\mathcal{A}$(1,2,1) to $\mathcal{A}$(1,2,5) where $\alpha = \frac{\tau}{t_c}$ and $\beta = \frac{t_a}{t_c}$. N_{opt} is derived from Table 17 using some simplified approximations. For instance, it is assumed that $\frac{Q}{R} \ll \frac{N}{n^2}$ for $\mathcal{A}$(1,2,2). For algorithm $\mathcal{A}$(1,2,1), N_{opt} is approximated by PR, if $Q \gg 4\frac{\tau t_a}{t_c^2}$, $t_a \geq t_c$, or if $Q \geq \frac{\tau}{2t_a}$, $t_a \gg t_c$; and $PR\frac{2Qt_a}{\tau}$ if $Q < \frac{\tau}{2t_a}$, $t_a \gg t_c$. Similar results can be derived for algorithms $\mathcal{A}$(1,2,2) to $\mathcal{A}$(1,2,5).

Lemma 5 *For sufficiently high data transfer rate compared to the arithmetic rate, the arithmetic load per processor shall be greater than the ratio of the start-up time and the time for an arithmetic operation in order to achieve maximum performance. In the extreme case where $N_{opt} = N_{max}$ processors are used, the time for the inner product should be greater than $\frac{\tau}{t_a}$.*

Algorithm	N_1	N_2	T_{min}, $B_m \geq B_{opt}$
$\mathcal{A}^c(1,2,1)$	$\sqrt{\frac{PN}{R}}$	$\sqrt{\frac{RN}{P}}$	$\frac{2PQR}{N}t_a + \frac{Q}{\sqrt{N}}(2\sqrt{PR} - \frac{P+R}{\sqrt{N}})t_c + n\tau$
$\mathcal{A}^c(1,2,2)$	$\sqrt{\frac{QN}{R}}$	$\sqrt{\frac{RN}{Q}}$	$\frac{2PQR}{N}t_a + \frac{P}{\sqrt{N}}(2\sqrt{QR} + \frac{nQ-(Q+R)}{\sqrt{N}})t_c + 2n\tau$
$\mathcal{A}^c(1,2,3)$	$\sqrt{\frac{QN}{P}}$	$\sqrt{\frac{PN}{Q}}$	$\frac{2PQR}{N}t_a + \frac{R}{\sqrt{N}}(2\sqrt{PQ} + \frac{nP(1+\frac{Q}{R})-(P+Q)}{\sqrt{N}})t_c + 3n\tau$
$\mathcal{A}^c(1,2,4)$	$\sqrt{\frac{PN}{Q}}$	$\sqrt{\frac{QN}{P}}$	$\frac{2PQR}{N}t_a + \frac{R}{\sqrt{N}}(2\sqrt{PQ} + \frac{nQ-(P+Q)}{\sqrt{N}})t_c + 2n\tau$
$\mathcal{A}^c(1,2,5)$	$\sqrt{\frac{RN}{Q}}$	$\sqrt{\frac{QN}{R}}$	$\frac{2PQR}{N}t_a + \frac{P}{\sqrt{N}}(2\sqrt{QR} + \frac{nR(1+\frac{Q}{P})-(Q+R)}{\sqrt{N}})t_c + 3n\tau$

Table 17: The *optimum* values of N_1 and N_2 for P, Q and R being multiples of N and *one-port* communication.

Algorithm	N_{opt}	N_{opt}, $P/Q/R \gg \alpha\beta$	N_{opt}, $P/Q/R \ll \alpha\beta$
$\mathcal{A}(1, 2, 1)$	$\frac{Q^2PR(1+\sqrt{1+8\alpha\beta/Q})^2}{4Q^2}$	$\min(PR, \frac{Q^2PR}{4\alpha^2})$, if $Q \gg 8\alpha\beta$	$\min(PR, \frac{2PQR\beta}{\alpha})$, if $Q \ll 8\alpha\beta$
$\mathcal{A}(1, 2, 2)$	$\frac{P^2QR(1+\sqrt{1+16\alpha\beta/P})^2}{16\alpha^2}$	$\min(QR, \frac{P^2QR}{16\alpha^2})$, if $P \gg 16\alpha\beta$	$\min(QR, \frac{PQR\beta}{\alpha})$, if $P \ll 16\alpha\beta$
$\mathcal{A}(1, 2, 3)$	$\frac{R^2PQ(1+\sqrt{1+24\alpha\beta/R})^2}{36\alpha^2}$	$\min(PQ, \frac{R^2PQ}{36\alpha^2})$, if $R \gg 24\alpha\beta$	$\min(PQ, \frac{2PQR\beta}{3\alpha})$, if $R \ll 24\alpha\beta$
$\mathcal{A}(1, 2, 4)$	$\frac{R^2PQ(1+\sqrt{1+16\alpha\beta/R})^2}{16\alpha^2}$	$\min(PQ, \frac{R^2PQ}{16\alpha^2})$, if $R \gg 16\alpha\beta$	$\min(PQ, \frac{PQR\beta}{\alpha})$, if $R \ll 16\alpha\beta$
$\mathcal{A}(1, 2, 5)$	$\frac{P^2QR(1+\sqrt{1+24\alpha\beta/P})^2}{36\alpha^2}$	$\min(QR, \frac{P^2PQ}{36\alpha^2})$, if $P \gg 24\alpha\beta$	$\min(QR, \frac{2PQR\beta}{3\alpha})$, if $P \ll 24\alpha\beta$

Table 18: The optimum number of processors for different algorithms. $\frac{\tau}{t_c} = \alpha$ and $\frac{t_a}{t_c} = \beta$.

Lemma 6 *For P, Q, and R multiples of N_1 and N_2 the complexities of algorithms $\mathcal{A}(\cdot,2,3)$ and $\mathcal{A}(\cdot,2,5)$ are always higher than that of min ($\mathcal{A}(\cdot,2,2)$, $\mathcal{A}(\cdot,2,4)$), if the optimum values of N_1 and N_2 are chosen for each algorithm.*

Proof: From Table 15, if $T_{min}^{\mathcal{A}(\cdot,2,3)} \leq T_{min}^{\mathcal{A}(\cdot,2,2)}$ then $R < P$. Also, by exchanging N_1 and N_2 in $\mathcal{A}(\cdot,2,4)$, if $T_{min}^{\mathcal{A}(\cdot,2,3)} \leq T_{min}^{\mathcal{A}(\cdot,2,4)}$ then $P < R$. Similar arguments can be applied to $\mathcal{A}(\cdot,2,5)$. For *n-port* communication, Table 16 is used instead. ∎

Lemma 7 *The complexity is monotonically decreasing as a function of B_m.*

The smallest value of B_m that minimizes the number of start-ups, B_{opt}, is given in Table 15.

Figures 17 to 19 show how the P, Q, R space is divided into regions according to algorithm of minimum complexity as a function of N, $\frac{\tau}{t_c}$, $\frac{P}{N}$, $\frac{Q}{N}$ and $\frac{R}{N}$. Each plot is generated from the complexity estimates and with the optimum values of N_1 and N_2. It is assumed that P, Q and R are multiples of N_1 and N_2, and $B_m = \infty$. Decreasing B_m has the same effect as decreasing $\frac{\tau}{t_c}$. For sufficiently small B_m, the coefficient of $\frac{\tau}{B_m}$ becomes the same as the coefficient of t_c; the data transfer becomes the dominating factor. All Figures contains four plots. The upper left plot (a) shows the boundary between $\mathcal{A}$(1,2,2) (above the boundary) and $\mathcal{A}$(1,2,1) + $\mathcal{A}$(1,2,4). Plot (b), the upper right plot, shows the boundary between $\mathcal{A}$(1,2,4) (above the boundary) and $\mathcal{A}$(1,2,1) + $\mathcal{A}$(1,2,2). Plot (c), the lower left plot, shows the boundary between $\mathcal{A}$(1,2,1) (below the boundary) and $\mathcal{A}$(1,2,2) + $\mathcal{A}$(1,2,4). Plot (d), the lower right plot, shows a contour of plot (c). For $N = 16$ and $\tau = t_c$, the volume for which $\mathcal{A}$(1,2,1) is optimum are larger than the volume for which $\mathcal{A}$(1,2,2) (and $\mathcal{A}$(1,2,4)) is optimum, Figure 17. With an increasing ratio of $\frac{\tau}{t_c}$ and small N, the regions for the $\mathcal{A}$(1,2,2) and $\mathcal{A}$(1,2,4) algorithms both decrease as shown from Figure 17 to Figure 18.

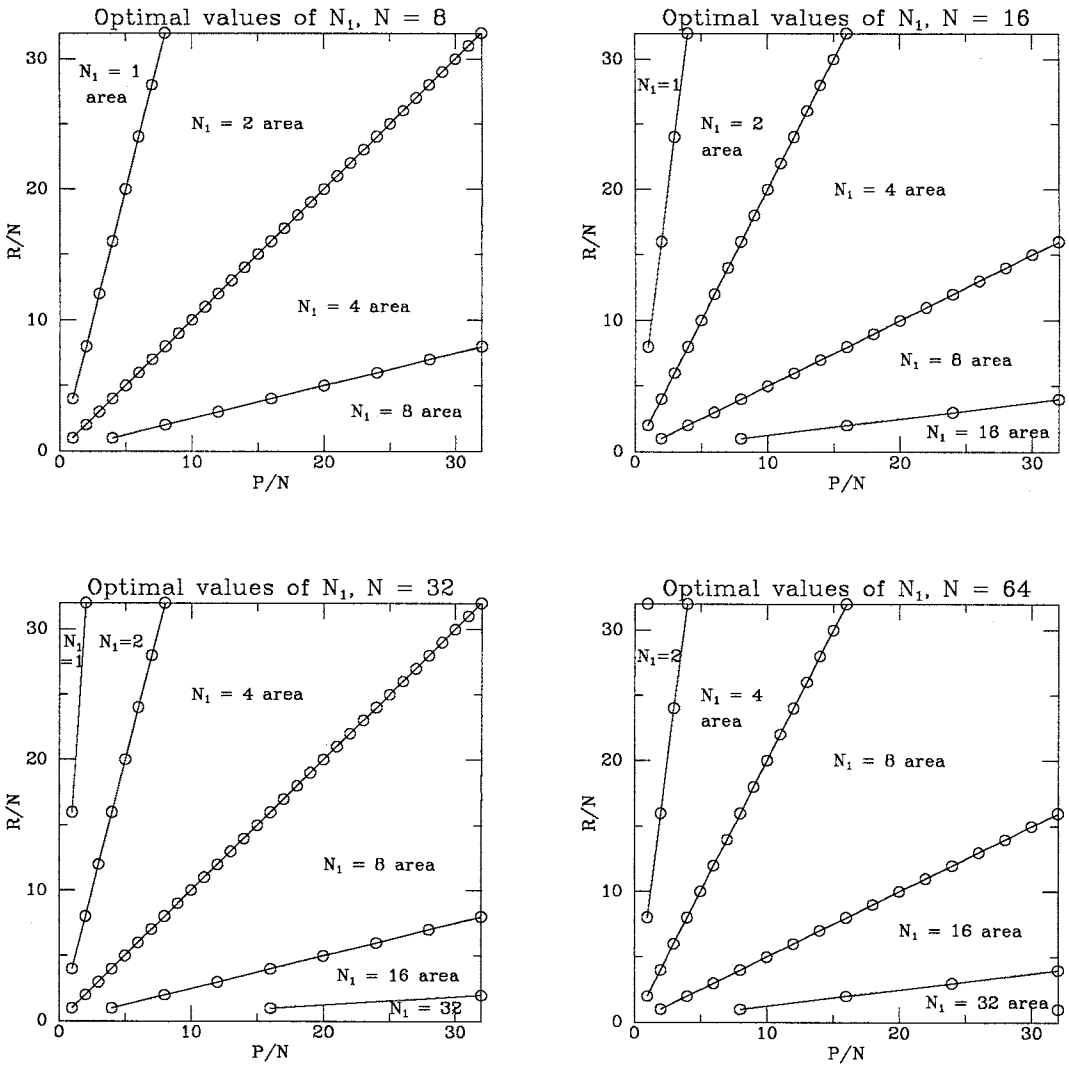


Figure 15: Optimal values of $\frac{N_1}{N_2}$ for algorithm $\mathcal{A}(1,2,1)$.

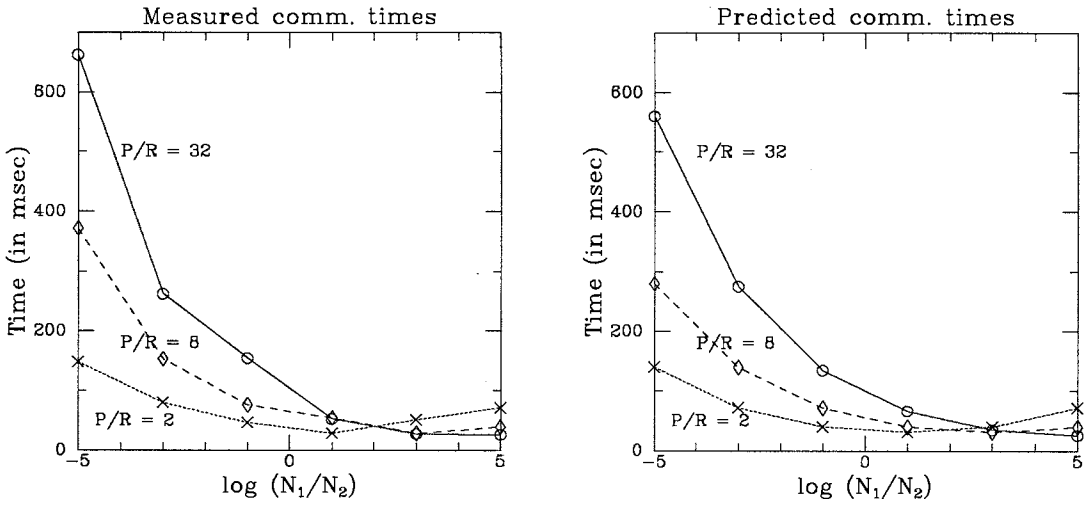


Figure 16: Measured and predicted communication times for algorithm $\mathcal{A}(1,2,1)$ on an Intel iPSC as a function of $\frac{N_1}{N_2}$ with $N = Q = 32$ and $PR = 1024N$.

As the number of processors increases, such as from Figure 17 to 19, the region in which algorithms $\mathcal{A}(1,2,2)$ or $\mathcal{A}(1,2,4)$ is optimum increases. Increasing the ratio of $\frac{\tau}{t_c}$ for large N has less effect than that for small N. In fact, when $N = 1024$, the Figure for $\frac{\tau}{t_c} = 1$ is the same as that for $\frac{\tau}{t_c} = 1000$, Figure 19. For $\frac{P}{N} = \frac{Q}{N} = \frac{R}{N}$, the communication complexity of algorithm $\mathcal{A}(1,2,1)$ is less than that of algorithms $\mathcal{A}(1,2,2)$ and $\mathcal{A}(1,2,4)$; the complexities of $\mathcal{A}(1,2,2)$ and $\mathcal{A}(1,2,4)$ are the same. The volume for which $\mathcal{A}(1,2,1)$ is optimum is always at least $\frac{1}{3}$ of the whole volume. The volumes for $\mathcal{A}(1,2,2)$ and $\mathcal{A}(1,2,4)$ are the same and symmetrical to $P = R$ plane. They are at most $\frac{1}{3}$ of the whole volume. As N increases, the volume for each of the 3 algorithms approaches $\frac{1}{3}$ of the whole volume. For $\mathcal{A}(1,2,1)$, the shape of its optimum volume is a pyramid with a square base at $\frac{Q}{N} = 0$ plane.

Theorem 6 *The temporary storage is minimized for the algorithm that minimizes the communication complexities.*

Theorem 6 implies that for P, Q, R multiples of N, the temporary storage is minimized for the algorithm that minimized the total complexity. The temporary storage requirements for the different algorithms are summarized in Table 19. For instance, if algorithm $\mathcal{A}(1,2,1)$ is optimum, then $\max(PQ, QR, PR) = PR$, and the required temporary storage is $\frac{Q(P+R)}{N}$.

Theorem 7 *The complexity of matrix multiplication is the same for both matrices encoded in binary code and binary-reflected Gray code.*

For broadcasting and reduction the same algorithms can be used for both embeddings. Moreover, the same transposition algorithm also apply to both embeddings [7]. The difference between the two encodings is the order in which operations are carried out, and the alignment of operands.

With *n-port* communication and unlimited buffer size the communication time is minimized, if the data transfer time is minimized. The optimum value of N_1 for algorithm $\mathcal{A}(n,2,3)$ minimizes $R\frac{N_1-1}{n_1} + Q\frac{N_2-1}{n_2}$ if P, Q, R are multiples of N. As in the *one-port* case the optimum is independent of τ, t_c, and t_a. Moreover, the optimum is only a function of the ratio $\frac{R}{Q}$, as in the *one-port* case. Figure 20 shows the optimum values of N_1

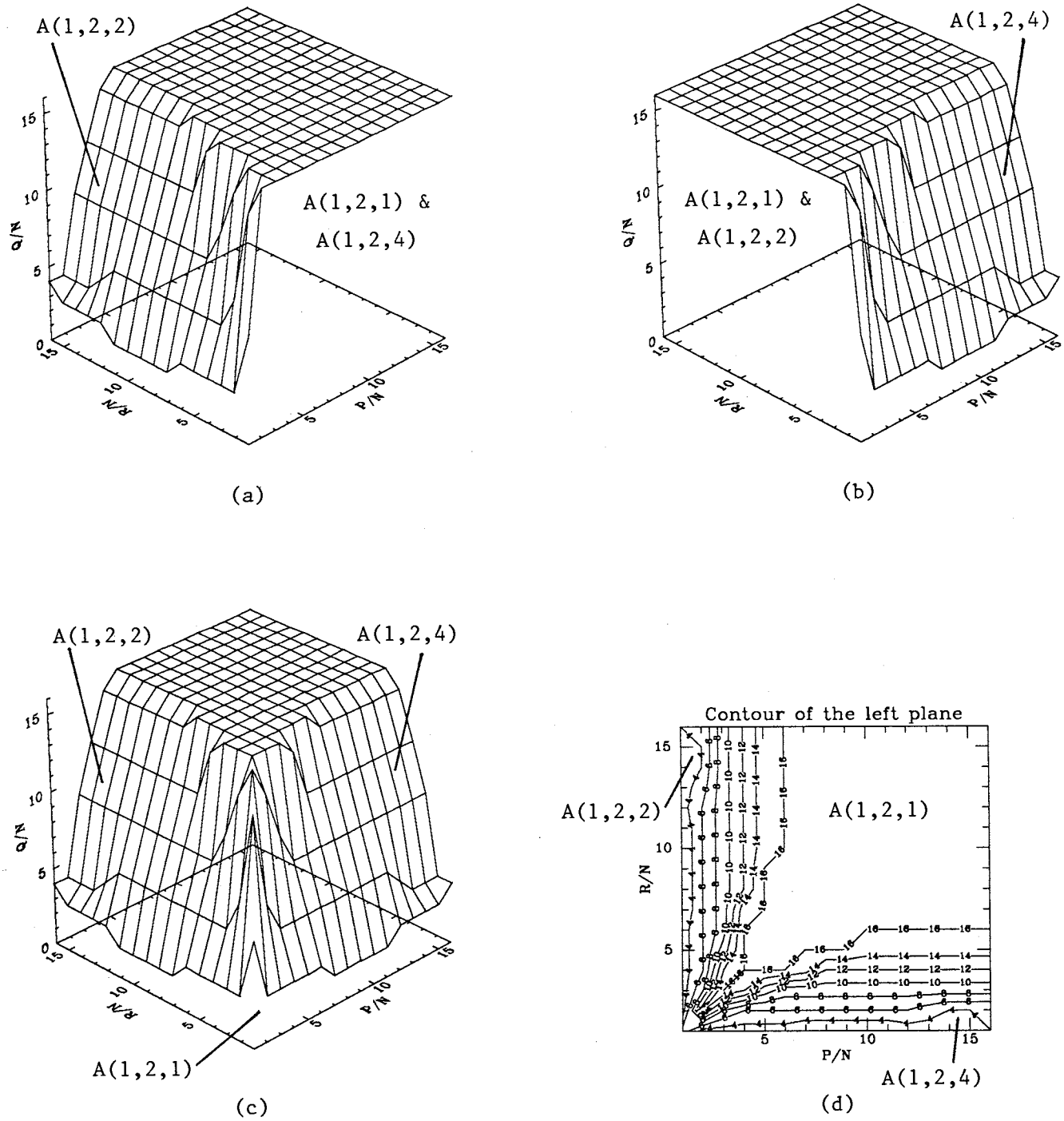


Figure 17: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = t_c$, $B_m = \infty$, two dimensional partitioning, *one-port* communication. (a) The boundary between $\mathcal{A}(1,2,2)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,4)$. (b) The boundary between $\mathcal{A}(1,2,4)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,2)$. (c) The boundary between $\mathcal{A}(1,2,1)$ and $\mathcal{A}(1,2,2) + \mathcal{A}(1,2,4)$. (d) Contour plot of (c).

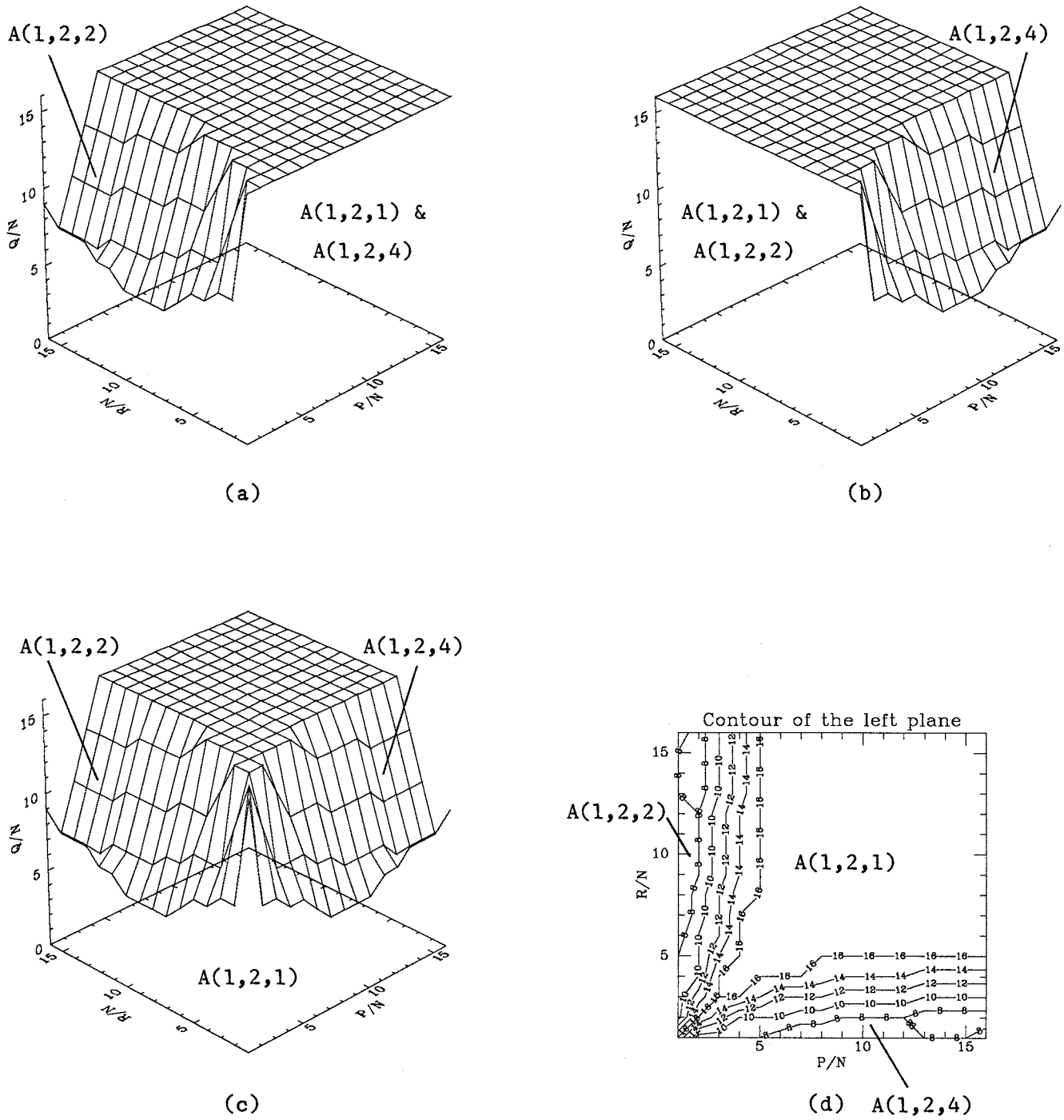


Figure 18: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = 100t_c$, $B_m = \infty$, two dimensional partitioning, *one-port* communication. (a) The boundary between $\mathcal{A}(1,2,2)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,4)$. (b) The boundary between $\mathcal{A}(1,2,4)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,2)$. (c) The boundary between $\mathcal{A}(1,2,1)$ and $\mathcal{A}(1,2,2) + \mathcal{A}(1,2,4)$. (d) Contour plot of (c).

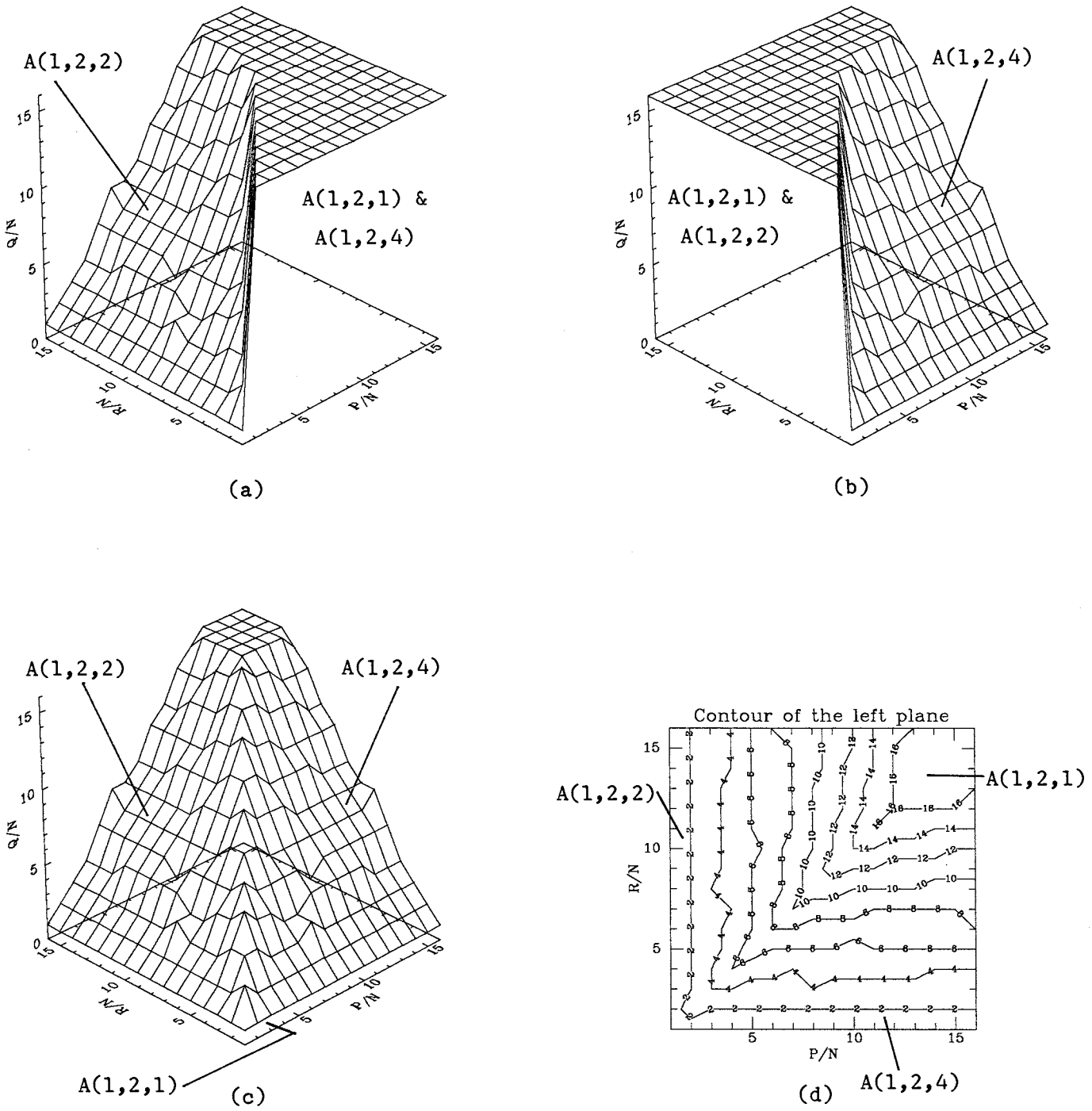


Figure 19: Lowest complexity algorithm as a function of matrix shape; $N = 1024$, $\frac{\tau}{t_c} = 1$ to 1000, $B_m = \infty$, two dimensional partitioning, *one-port* communication. (a) The boundary between $\mathcal{A}(1,2,2)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,4)$. (b) The boundary between $\mathcal{A}(1,2,4)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,2)$. (c) The boundary between $\mathcal{A}(1,2,1)$ and $\mathcal{A}(1,2,2) + \mathcal{A}(1,2,4)$. (d) Contour plot of (c).

Algorithm	Temporary storage		
	C	D	A
$\mathcal{A}^{l\times l}(\cdot,2,1)$	$\frac{PQ}{N}$	$\frac{QR}{N}$	0
$\mathcal{A}^{l\times l}(\cdot,2,2)$	$\frac{PQ}{N}$	0	$\frac{PR}{N}$
$\mathcal{A}^{l\times l}(\cdot,2,3)$	0	$\frac{QR}{N}$	$\frac{PR}{N}$
$\mathcal{A}^{l\times l}(\cdot,2,4)$	0	$\frac{QR}{N}$	$\frac{PR}{N}$
$\mathcal{A}^{l\times l}(\cdot,2,5)$	$\frac{PQ}{N}$	0	$\frac{PR}{N}$
$\mathcal{A}^{c}(\cdot,2,1)$	$\frac{1}{2}\frac{PQ}{N_1}$	$\frac{1}{2}\frac{QR}{N_2}$	0
$\mathcal{A}^{c}(\cdot,2,2)$	$\frac{1}{2}\frac{PQ}{N_1}$	0	$\frac{PR}{N_2}$
$\mathcal{A}^{c}(\cdot,2,3)$	0	$\frac{1}{2}\frac{QR}{N_1}$	$\frac{PR}{N_2}$
$\mathcal{A}^{c}(\cdot,2,4)$	0	$\frac{1}{2}\frac{QR}{N_2}$	$\frac{PR}{N_1}$
$\mathcal{A}^{c}(\cdot,2,5)$	$\frac{1}{2}\frac{PQ}{N_2}$	0	$\frac{PR}{N_1}$

Table 19: Temporary storage requirements for matrix multiplication using two-dimensional partitioning.

for algorithm $\mathcal{A}(n,2,2)$. For algorithms $\mathcal{A}(n,2,3)$, $\mathcal{A}(n,2,4)$ and $\mathcal{A}(n,2,5)$, $(Q,\ R)$ are replaced by $(Q,\ P)$, $(P,\ Q)$ and $(R,\ Q)$ respectively. For $\mathcal{A}(n,2,1)$, Q is replaced by P, approximately. For a sufficiently small buffer size the number of start-ups is proportional to the number of element transfers, and the optimum values of N_1 and N_2 remain the same as for an unlimited buffer size.

Figures 21 and 22 show the partitioning of the P, Q, R space according to the algorithm of minimum complexity with *n-port* communication. The four plots of each Figure are organized the same way as in the *one-port* case. For P, Q, and R multiples of N algorithm $\mathcal{A}(n,2,1)$ is preferable with respect to execution time for most values of P, Q, and R. Algorithm $\mathcal{A}(n,2,1)$ shall be chosen if $P,\ R \geq Q$. Algorithm $\mathcal{A}(n,2,2)$ shall be chosen only if $Q,\ R > P$, and $\mathcal{A}(n,2,4)$ only if $Q,\ P > R$. The volume for $\mathcal{A}(n,2,2)$ $(\mathcal{A}(n,2,4))$ is significantly smaller than that for $\mathcal{A}(1,2,2)$ $(\mathcal{A}(1,2,4))$ of the corresponding *one-port* case. For the case $1 \leq (P,Q,R) \leq N_1,N_2$ the arithmetic complexity is also of concern, and the processor utilization is different. Algorithm $\mathcal{A}(n,2,1)$ is always preferable for $P,\ R > Q$; algorithm $\mathcal{A}(n,2,2)$ is mostly preferable for $Q,\ R > P$, and algorithm $\mathcal{A}(n,2,4)$ mostly for $Q,\ P > R$. The choice of algorithm with respect to arithmetic complexity is similar to the choice with respect to communication complexity. The boundaries in the case $1 \leq (P,Q,R) \leq N$ are illustrated in Figures 23 and 24. Note that for $\tau = t_c$, $N = 1024$, various ratios of t_a to t_c, such as 0, 1 or 100, all result in the optimum regions as Figure 23. Interestingly, it has a same shape as Figure 19. For $\tau = t_c$, $N = 1024$, $\frac{t_a}{t_c} = 1$ has the same Figure as $\frac{t_a}{t_c} = 1000$, which is almost the same as $\frac{t_a}{t_c} = 0$ of Figure 24.

4.3 Three-Dimensional Partitioning

Before considering the three dimensional case, we study the communication complexity of broadcasting from each node in some k-dimensional subcube to the whole n-cube. Assume that the amount of data to be broadcast is M per source node. The broadcast can be realized by all-to-all broadcasting in the k-cube containing source nodes followed by one-to-all broadcasting of data 2^kM in the $(n-k)$-cube. Alternatively, we can carry out the one-to-all broadcasting in the $(n-k)$-cube with data M concurrently for all the k independent subcubes. Then perform all-to-all broadcasting within the k-cube for all the $n-k$ independent subcubes. The latter is obviously preferable to the former. We denote the some-to-all broadcasting by SBT-b$(1, 2^k, 2^n; \cdot, M)$, which can be realized by SBT-b$(1, \cdot, 2^{n-k}; \cdot, M)$ + SBT-b$(1, *, 2^k; \cdot, M)$. The second argument represents the number of source nodes in this case. The complexity is

$$T = \left\{(n-k) + (2^k - 1)\right\} Mt_c + \left\{(n-k)\left\lceil \frac{M}{B_m} \right\rceil + \sum_{i=0}^{k-1} \left\lceil \frac{2^iM}{B_m} \right\rceil \right\} \tau.$$

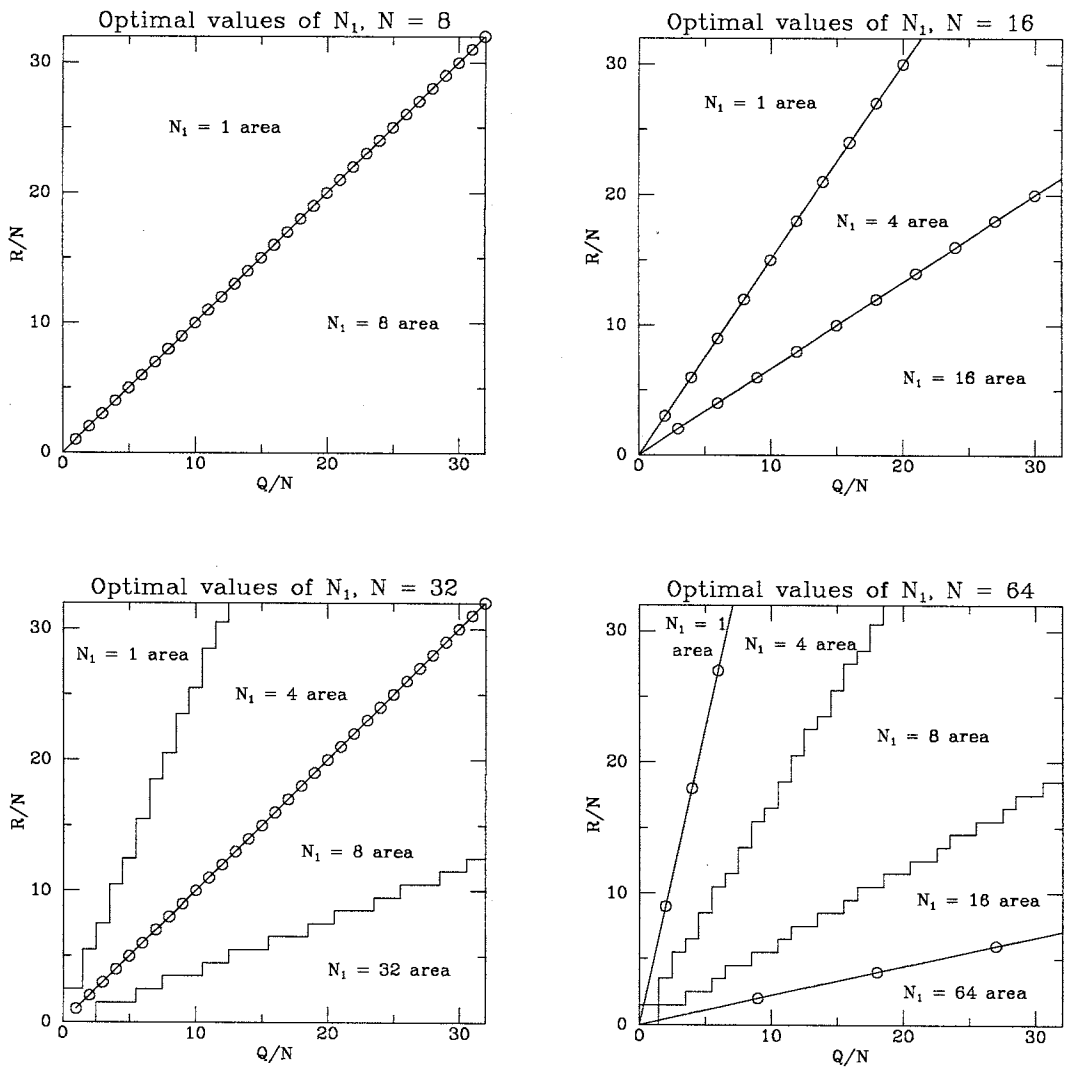


Figure 20: Optimal values of N_1 for algorithm $\mathcal{A}(n,2,2)$.

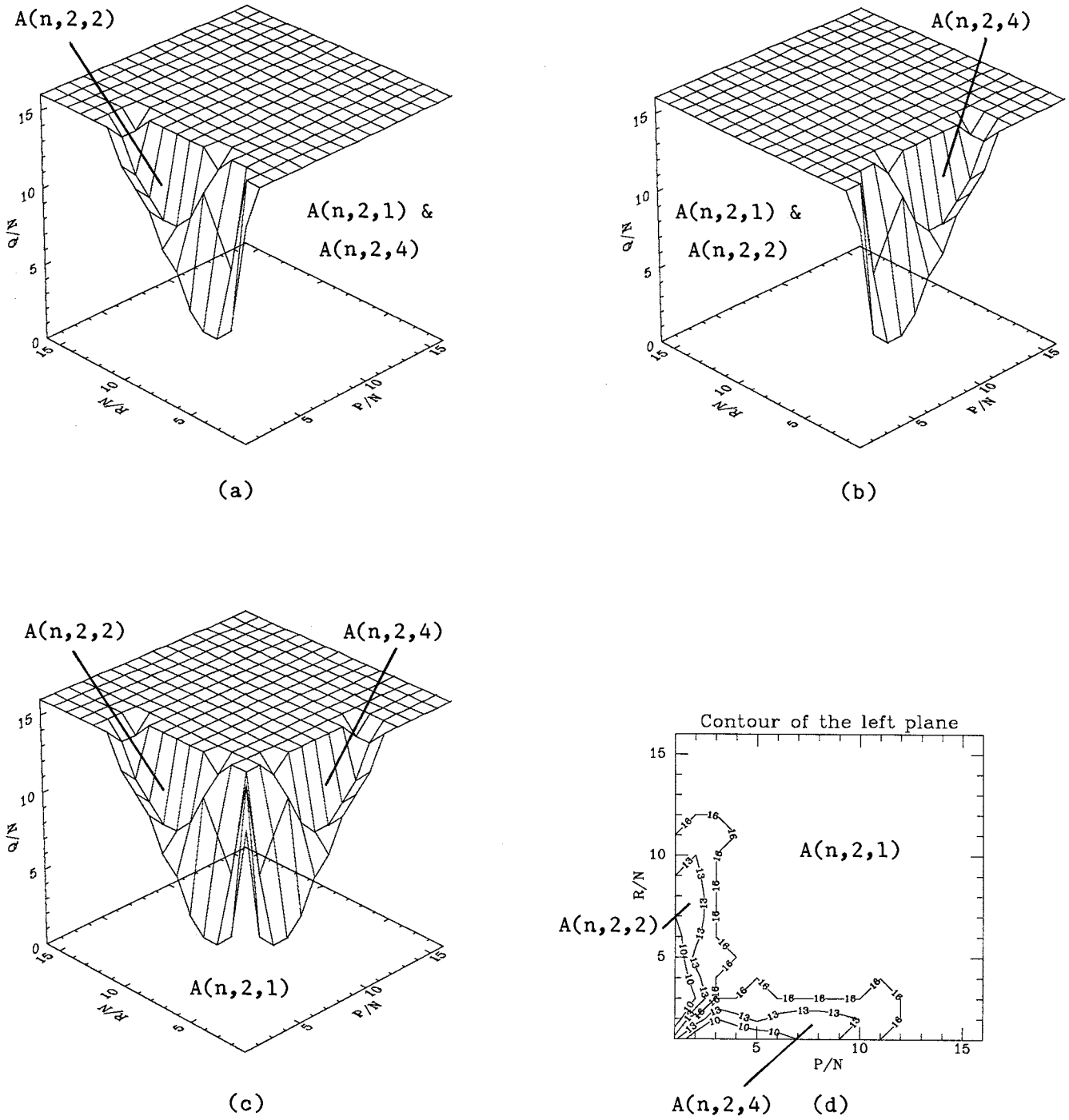


Figure 21: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = t_c$, $B_m = \infty$, two dimensional partitioning, *n-port* communication. (a) The boundary between $\mathcal{A}(1,2,2)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,4)$. (b) The boundary between $\mathcal{A}(1,2,4)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,2)$. (c) The boundary between $\mathcal{A}(1,2,1)$ and $\mathcal{A}(1,2,2) + \mathcal{A}(1,2,4)$. (d) Contour plot of (c).

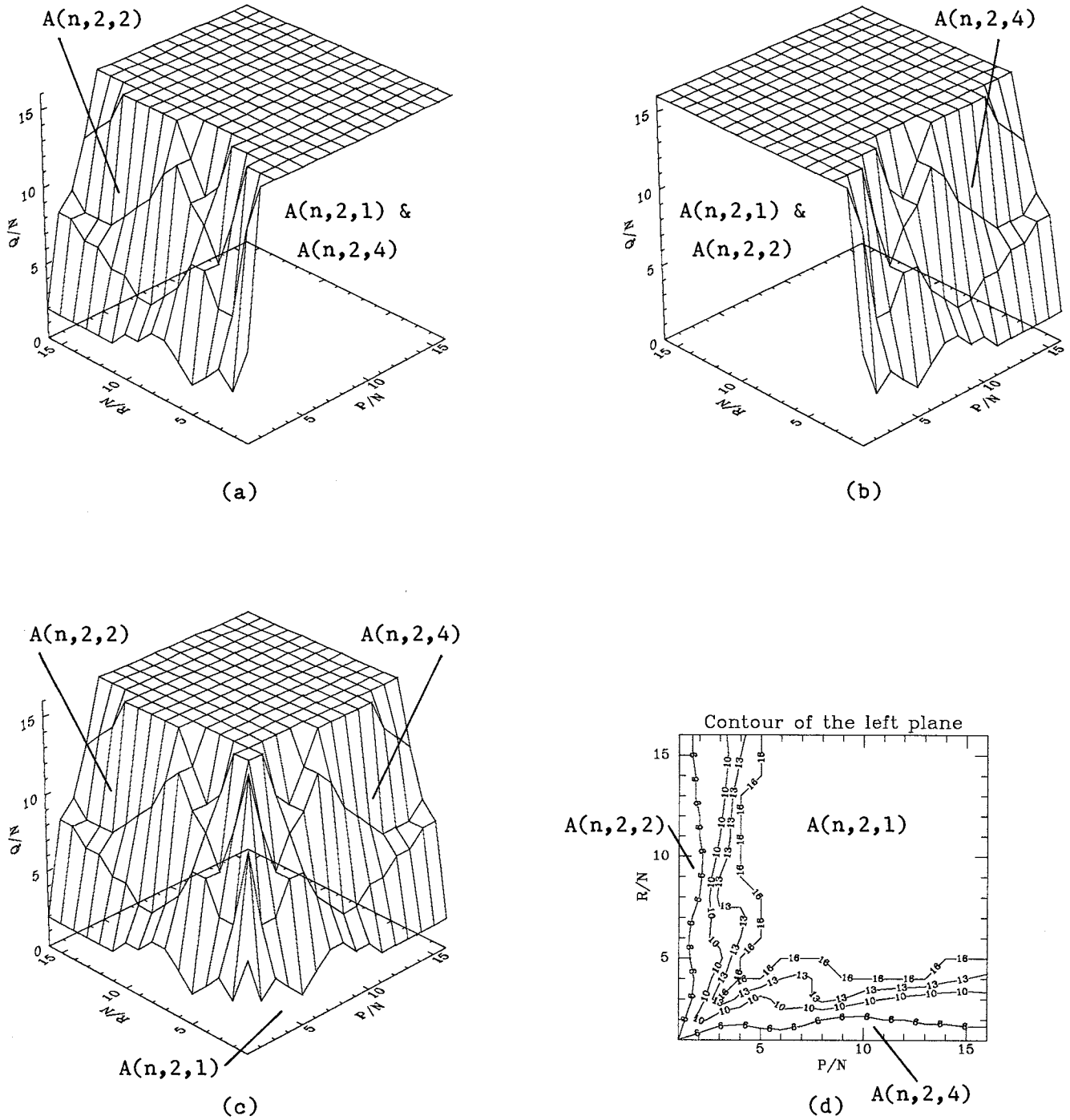


Figure 22: Lowest complexity algorithm as a function of matrix shape; $N = 1024$, $\tau = t_c$, $B_m = \infty$, two dimensional partitioning, *n-port* communication. (a) The boundary between $\mathcal{A}(1,2,2)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,4)$. (b) The boundary between $\mathcal{A}(1,2,4)$ and $\mathcal{A}(1,2,1) + \mathcal{A}(1,2,2)$. (c) The boundary between $\mathcal{A}(1,2,1)$ and $\mathcal{A}(1,2,2) + \mathcal{A}(1,2,4)$. (d) Contour plot of (c).

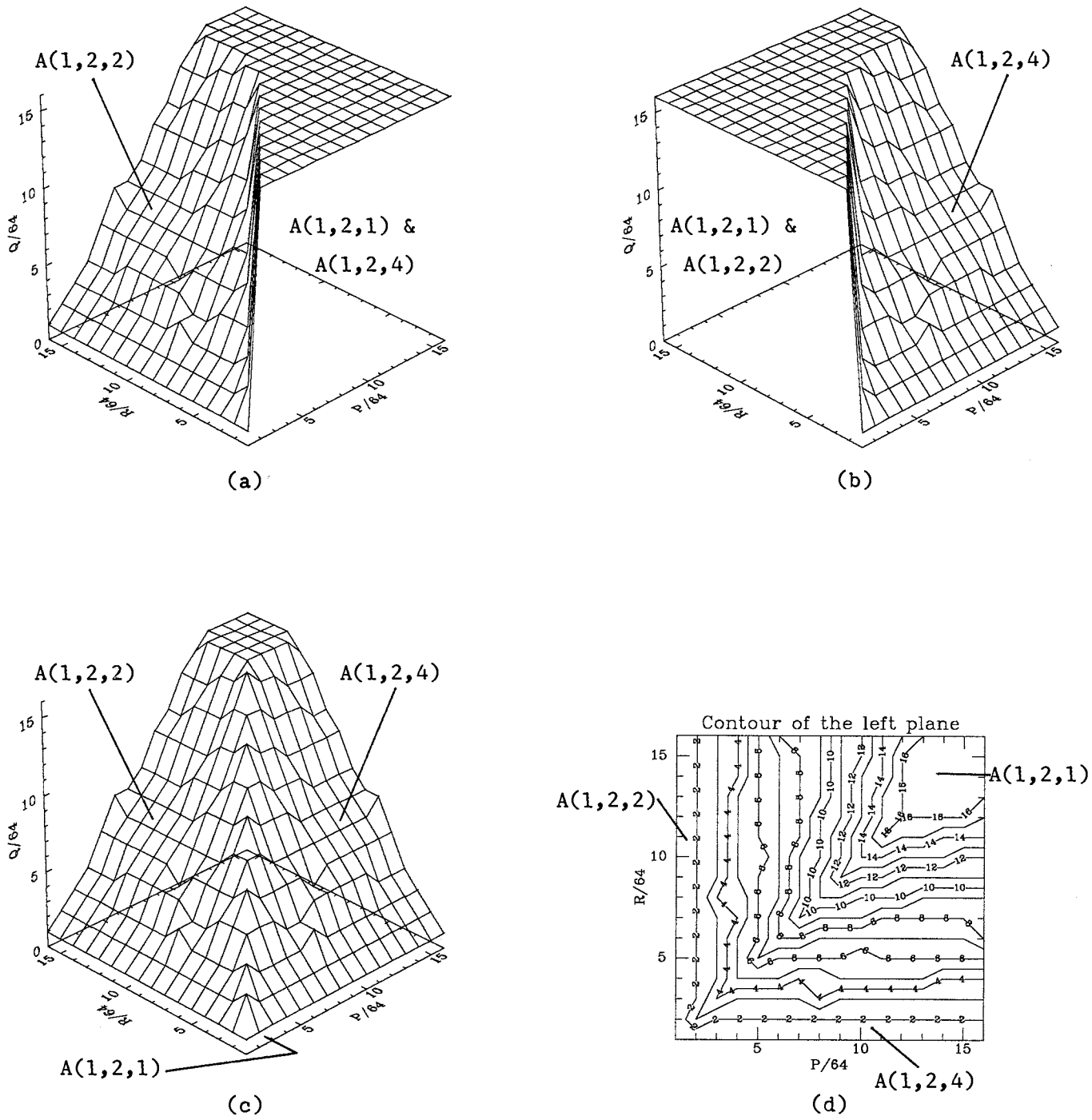


Figure 23: The total complexity for P, Q, and R smaller than N, $N = 1024$, $t_a = 0$ (1 or 1000) t_c, $\tau = t_c$, $B_m = \infty$, two-dimensional partitioning, *one-port* communication.

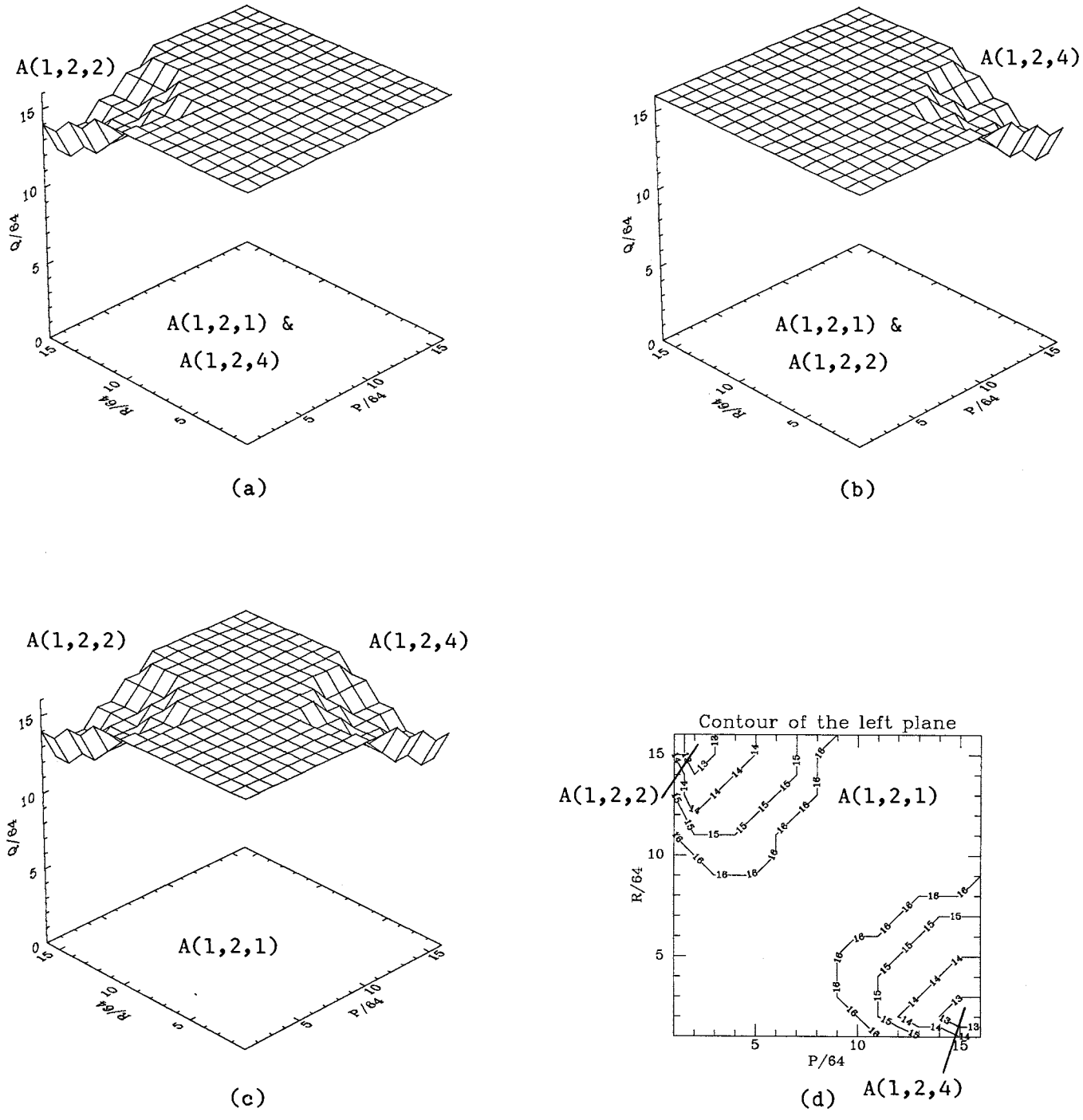


Figure 24: The total complexity for P, Q, and R smaller than N, $N = 1024$, $t_a = 0$, $\tau = 1000t_c$, $B_m = \infty$, two-dimensional partitioning, *one-port* communication.

In applying the some-to-all broadcasting in the following, $M = 1$ if $k < n$, so we choose the SBT algorithm instead of the nESBT algorithm for the one-to-all broadcasting part.

In the case of a three dimensional partitioning of the Boolean cube each $N_1 \times N_2$ subset of processors compute the product of a $P \times \frac{Q}{N_3}$ matrix and a $\frac{Q}{N_3} \times R$ matrix. If the matrices are initially allocated such that there are distinct submatrices $P \times \frac{Q}{N_3}$ and $\frac{Q}{N_3} \times R$ assigned to each set of $\frac{N}{N_3}$ processors then the multiplication in each subset is the same as in the two dimensional partitioning, except that Q is replaced by $\frac{Q}{N_3}$. In addition, there is an accumulation phase at the end. The accumulation can be made by the SBT-b$(1, *, N_3; A, \frac{P}{N_1}\frac{R}{N_2})$ or SBT-b$(n_3, *, N_3; A, \frac{P}{N_1}\frac{R}{N_2})$ algorithms depending on the communication capability of the system. The arithmetic time for this part of the computation is $\lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \log N_3 t_a$ without any pipelining, and all partial products being accumulated in the same way. The matrix A is allocated among $\frac{N}{N_3}$ processors. If there are several elements of the matrix A that are stored in the same processor, then the accumulation can be made faster by using *all-to-all reduction*. The complexity becomes $\sum_{i=1}^{n_3} \lceil \frac{\lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil}{2^i} \rceil t_a$. The communication complexity for the reduction is $\sum_{i=1}^{n_3} \lceil \frac{\lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil}{2^i} \rceil t_c + \sum_{i=1}^{n_3} \lceil \frac{\lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil}{2^i B_m} \rceil \tau$. When $P \leq N_1$ and $R \leq N_2$, it is an *all-to-one* reduction, and the communication complexity of the reduction is $(\lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil t_c + \lceil \frac{\lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil}{B_m} \rceil \tau) \log N_3$. When $\lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \geq N_3$, it is an *all-to-all reduction*, and the communication complexity of the reduction is approximately $(1 - \frac{1}{N_3}) \lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil t_c + \sum_{i=1}^{n_3} \lceil \frac{\lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil}{2^i B_m} \rceil \tau$.

To change the processor allocation from a two-dimensional partitioning, $N_1' \cdot N_2' = N$, to a three-dimensional partitioning, $N_1 \cdot N_2 \cdot N_3 = N$, one-dimensional matrix transpositions within each block of C and D are needed. Let $N_1' = \alpha_1 N_1$ and $N_2' = \alpha_2 N_2$, i.e., $N_3 = \alpha_1 \alpha_2$. The matrix C needs to be changed from an $\alpha_1 N_1 \times \alpha_2 N_2$ partitioning to an $N_1 \times \alpha_1 \alpha_2 N_2$ partitioning. Similarly, the matrix D needs to be changed from an $\alpha_1 N_1 \times \alpha_2 N_2$ partitioning to an $\alpha_1 \alpha_2 N_1 \times N_2$ partitioning. Conceptually, $\alpha_1 \alpha_2$ can also be viewed as the third dimension. Each $\frac{P}{N_1} \times \frac{Q}{\alpha_2 N_2}$ block matrix of C needs to be transposed from α_1 block rows to α_1 block columns. Similarly for D that each $\frac{Q}{\alpha_1 N_1} \times \frac{R}{N_2}$ block matrix of D needs to be transposed from α_2 block columns to α_2 block rows. Since the optimum ratio of $\frac{N_1'}{N_2'} = \frac{N_1}{N_2}$ (as will become clear later), $\alpha_1 = \alpha_2 = \sqrt{N_3}$ (assuming n_3 is even), if N_1', N_2', N_1, N_2 and N_3 are optimally chosen.

In order to guarantee that a subset of columns of C resides in the same subcube as the corresponding subset of rows of D, an extra permutation is required. This permutation is equal to transposing a matrix partitioned by $\alpha_1 \times \alpha_2$ blocks. Figure 25 shows an example of the conversion from two-dimensional to three-dimensional partitioning.

Let $\beta_1 = \log_2 \alpha_1$ and $\beta_2 = \log_2 \alpha_2$. Before conversion, the address field of elements of matrix C used to encode the processor id is:

$$(\underbrace{r_{p-1} \cdots r_{p-n_1}}_{n_1} \underbrace{r_{p-n_1-1} \cdots r_{p-n_1-\beta_1}}_{\beta_1} r_{p-n_1-\beta_1-1} \cdots r_0, \underbrace{c_{q-1} \cdots c_{q-n_2}}_{n_2} \underbrace{c_{q-n_2-1} \cdots c_{q-n_2-\beta_2}}_{\beta_2} c_{q-n_2-\beta_2-1} \cdots c_0),$$

and after conversion:

$$(\underbrace{r_{p-1} \cdots r_{p-n_1}}_{n_1} r_{p-n_1-1} \cdots r_0,$$

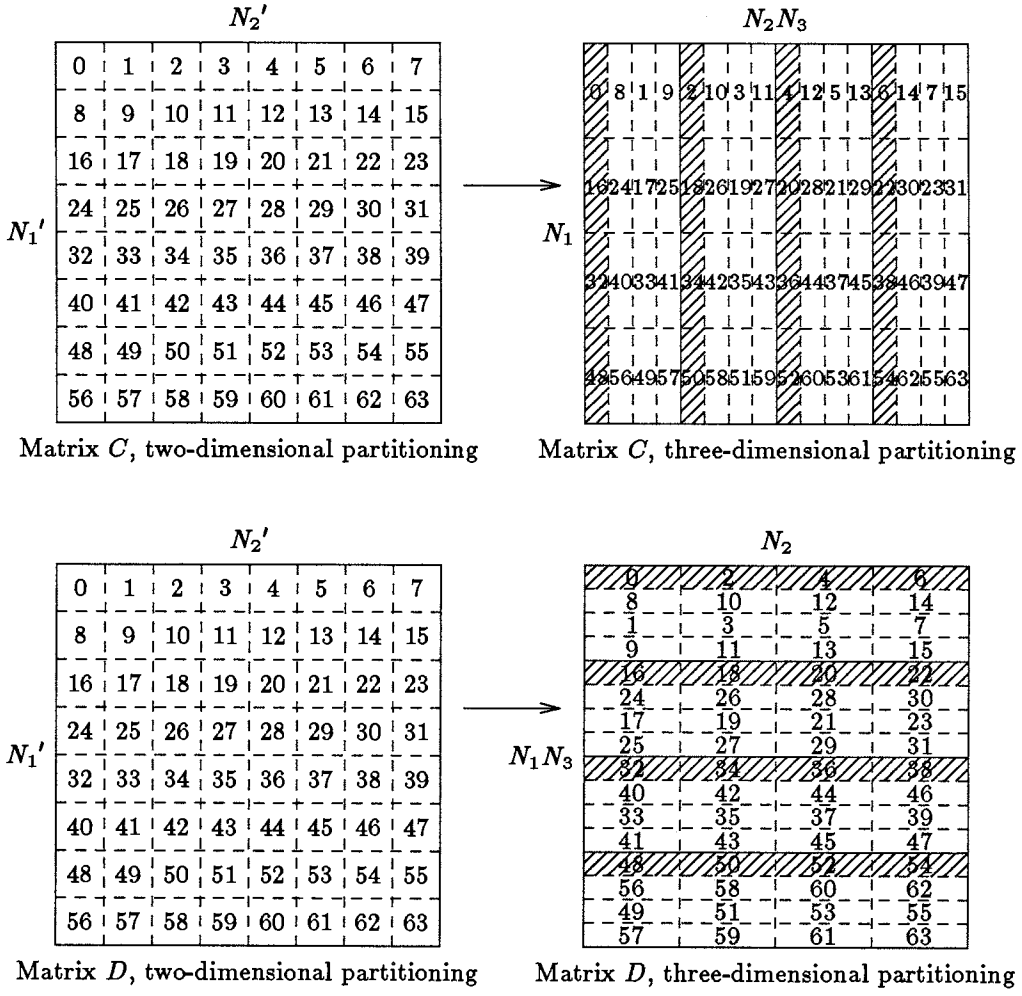
$$\underbrace{c_{q-1} \cdots c_{q-n_2}}_{n_2} \underbrace{c_{q-n_2-1} \cdots c_{q-n_2-\beta_2}}_{\beta_2} \underbrace{c_{q-n_2-\beta_2-1} \cdots c_{q-n_2-\beta_2-\beta_1}}_{\beta_1} c_{q-n_2-\beta_2-\beta_1-1} \cdots c_0).$$

The processor id before the conversion is :

$$pid(r, c) = \underbrace{r_{p-1} \cdots r_{p-n_1}}_{n_1} \| \underbrace{r_{p-n_1-1} \cdots r_{p-n_1-\beta_1}}_{\beta_1} \| \underbrace{c_{q-1} \cdots c_{q-n_2}}_{n_2} \| \underbrace{c_{q-n_2-1} \cdots c_{q-n_2-\beta_2}}_{\beta_2},$$

and the processor id after the conversion is:

$$pid(r, c) = \underbrace{r_{p-1} \cdots r_{p-n_1}}_{n_1} \| \underbrace{c_{q-n_2-\beta_2-1} \cdots c_{q-n_2-\beta_2-\beta_1}}_{\beta_1} \| \underbrace{c_{q-1} \cdots c_{q-n_2}}_{n_2} \| \underbrace{c_{q-n_2-1} \cdots c_{q-n_2-\beta_2}}_{\beta_2}.$$



Matrix C, two-dimensional partitioning

Matrix C, three-dimensional partitioning

Matrix D, two-dimensional partitioning

Matrix D, three-dimensional partitioning

Figure 25: Conversion from two-dimensional to three-dimensional partitioning. $N_1' = N_2' = 8$, $N_1 = N_2 = N_3 = 4$ and $\alpha_1 = \alpha_2 = 2$. The shaded areas on the right denote the submatrices partitioned by $N_1 \times N_2$ blocks.

Before conversion, the address field of elements of matrix D used to encode the processor id is:

$$(\underbrace{r_{q-1}\cdots r_{q-n_1}}_{n_1}\underbrace{r_{q-n_1-1}\cdots r_{q-n_1-\beta_1}}_{\beta_1}r_{q-n_1-\beta_1-1}\cdots r_0,\ \underbrace{c_{r-1}\cdots c_{r-n_2}}_{n_2}\underbrace{c_{r-n_2-1}\cdots c_{r-n_2-\beta_2}}_{\beta_2}c_{r-n_2-\beta_2-1}\cdots c_0),$$

and after conversion:

$$(\underbrace{r_{q-1}\cdots r_{q-n_1}}_{n_1}\underbrace{r_{q-n_1-1}\cdots r_{q-n_1-\beta_2}}_{\beta_2}\underbrace{r_{q-n_1-\beta_2-1}\cdots r_{q-n_1-\beta_2-\beta_1}}_{\beta_1}r_{q-n_1-\beta_2-\beta_1-1}\cdots r_0,$$

$$\underbrace{c_{r-1}\cdots c_{r-n_2}}_{n_2}c_{r-n_2-1}\cdots c_0).$$

The processor id before the conversion is:

$$pid(r,c)=\underbrace{r_{q-1}\cdots r_{q-n_1}}_{n_1}\|\underbrace{r_{q-n_1-1}\cdots r_{q-n_1-\beta_1}}_{\beta_1}\|\underbrace{c_{r-1}\cdots c_{r-n_2}}_{n_2}\|\underbrace{c_{r-n_2-1}\cdots c_{r-n_2-\beta_2}}_{\beta_2},$$

and the processor id after the conversion is:

$$pid(r,c)=\underbrace{r_{q-1}\cdots r_{q-n_1}}_{n_1}\|\underbrace{r_{q-n_1-\beta_2-1}\cdots r_{q-n_1-\beta_2-\beta_1}}_{\beta_1}\|\underbrace{c_{r-1}\cdots c_{r-n_2}}_{n_2}\|\underbrace{r_{q-n_1-1}\cdots r_{q-n_1-\beta_2}}_{\beta_2}.$$

The algorithm can be implemented as

$$\text{SBT-p}(1,*,\sqrt{2^{k_q}};C,\frac{PQ}{N})+\text{SBT-p}(1,*,\sqrt{2^{k_q}};D,\frac{QR}{N})+\text{TXP}(1,\alpha_1,\alpha_2;\lceil\frac{Q}{N_1}\rceil,\lceil\frac{R}{N_2}\rceil)$$

$$+\text{SBT-b}(1,2^{k'_q},2^{k_p};D,\lceil\frac{R}{N_2}\rceil\lceil\frac{Q}{N_1N_3}\rceil)+\text{SBT-b}(1,2^{k''_q},2^{k_r};C,\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2N_3}\rceil)+\text{SBT-b}(1,2^{k_{pr}},2^{k_q};A,\left\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{1}{N_3}\right\rceil)$$

where $k_p=\min(\log P,n_1)$, $k_r=\min(\log R,n_2)$, $k_q=\min(\log Q,n_3)$, $k'_q=\min(\log\lceil\frac{Q}{N_3}\rceil,n_1)$, $k''_q=\min(\log\lceil\frac{Q}{N_3}\rceil,n_2)$ and $k_{pr}=\min(\log\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil,n_3)$. Also let $k'_p=\min(\log\lceil\frac{P}{N_1}\rceil,n_3)$, $k'_r=\min(\log\lceil\frac{R}{N_2}\rceil,n_3)$, $k'_{q3}=\min(\log\lceil\frac{Q}{N_1}\rceil,n_3)$, $k''_{q3}=\min(\log\lceil\frac{Q}{N_2}\rceil,n_3)$, $k_{pq}=\min(k'_p,k''_{q3})$ and $k_{qr}=\min(k'_r,k'_{q3})$. The total complexity is

$$\begin{aligned}
T &= \left\{(2\lceil\frac{Q}{N_3}\rceil-1)\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil+\sum_{i=1}^{n_3}\left\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{1}{2^i}\right\rceil+\left\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{1}{N_3}\right\rceil\right\}t_a\\
&+\left\{(1+\frac{k_{pq}-2}{2^{|k'_p-k''_{q3}|+1}})\max(\lceil\frac{P}{N_1N_3}\rceil\lceil\frac{Q}{N_2}\rceil,\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2N_3}\rceil)\right.\\
&+(1+\frac{k_{qr}-2}{2^{|k'_r-k'_{q3}|+1}})\max(\lceil\frac{Q}{N_1N_3}\rceil\lceil\frac{R}{N_2}\rceil,\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2N_3}\rceil)\\
&+k_q\left\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{1}{N_3}\right\rceil\\
&+(\max(k_p-k'_q,0)+2^{k'_q}-1)\lceil\frac{Q}{N_1N_3}\rceil\lceil\frac{R}{N_2}\rceil\\
&+(\max(k_r-k''_q,0)+2^{k''_q}-1)\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2N_3}\rceil\\
&\left.+(\max(k_q-k_{pr},0)+2^{k_{pr}}-1)\left\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{1}{N_3}\right\rceil\right\}t_c\\
&+\left\{\sum_{i=1}^{|k'_p-k''_{q3}|}\left\lceil\frac{1}{2^iB_m}\max(\lceil\frac{P}{N_1N_3}\rceil\lceil\frac{Q}{N_2}\rceil,\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2N_3}\rceil)\right\rceil\right.\\
&\left.+k_{pq}\left\lceil\frac{1}{2^{|k'_p-k''_{q3}|+1}B_m}\max(\lceil\frac{P}{N_1N_3}\rceil\lceil\frac{Q}{N_2}\rceil,\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2N_3}\rceil)\right\rceil\right.
\end{aligned}$$

$$
\begin{aligned}
&+ \sum_{i=1}^{|k'_r - k'_{q3}|} \left\lceil \frac{1}{2^i B_m} \max(\lceil \frac{Q}{N_1 N_3} \rceil \lceil \frac{R}{N_2} \rceil, \lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2 N_3} \rceil) \right\rceil \\
&+ k_{qr} \left\lceil \frac{1}{2^{|k'_r - k'_{q3}|+1} B_m} \max(\lceil \frac{Q}{N_1 N_3} \rceil \lceil \frac{R}{N_2} \rceil, \lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2 N_3} \rceil) \right\rceil \\
&+ k_q \left\lceil \lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \frac{1}{N_3 B_m} \right\rceil \\
&+ (\max(k_p - k'_q, 0) \left\lceil \lceil \frac{Q}{N_1 N_3} \rceil \lceil \frac{R}{N_2} \rceil \frac{1}{B_m} \right\rceil + \sum_{i=0}^{k_p - 1} \left\lceil \lceil \frac{Q}{N_1 N_3} \rceil \lceil \frac{R}{N_2} \rceil \frac{2^i}{B_m} \right\rceil \\
&+ (\max(k_r - k''_q, 0) \left\lceil \lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2 N_3} \rceil \frac{1}{B_m} \right\rceil + \sum_{i=0}^{k_r - 1} \left\lceil \lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2 N_3} \rceil \frac{2^i}{B_m} \right\rceil \\
&+ (\max(k_q - k_{pr}, 0) \left\lceil \lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \frac{1}{N_3 B_m} \right\rceil + \sum_{i=1}^{k_{pr}} \left\lceil \lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \frac{2^i}{N_3 B_m} \right\rceil \right\} \tau.
\end{aligned}
$$

The total communication complexity for $\min(PQ, QR, PR) \geq N$ is

$$
\begin{aligned}
T_{comm} = &\left\{ \frac{n_3}{2} \left\lceil \lceil \frac{P}{N_1 N_3} \rceil \lceil \frac{Q}{N_2} \rceil \frac{1}{2} \right\rceil + \frac{n_3}{2} \left\lceil \lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2 N_3} \rceil \frac{1}{2} \right\rceil + n_3 \left\lceil \lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \frac{1}{N_3} \right\rceil \right. \\
&\left. + (N_1 - 1) \lceil \frac{Q}{N_1 N_3} \rceil \lceil \frac{R}{N_2} \rceil + (N_2 - 1) \lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2 N_3} \rceil + (N_3 - 1) \left\lceil \lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \frac{1}{N_3} \right\rceil \right\} t_c \\
&+ \left\{ \frac{n_3}{2} \left\lceil \lceil \frac{P}{N_1 N_3} \rceil \lceil \frac{Q}{N_2} \rceil \frac{1}{2B_m} \right\rceil + \frac{n_3}{2} \left\lceil \lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2 N_3} \rceil \frac{1}{2B_m} \right\rceil + n_3 \left\lceil \lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \frac{1}{N_3 B_m} \right\rceil \right. \\
&\left. + \sum_{i=0}^{n_1 - 1} \left\lceil \lceil \frac{Q}{N_1 N_3} \rceil \lceil \frac{R}{N_2} \rceil \frac{2^i}{B_m} \right\rceil + \sum_{i=0}^{n_2 - 1} \left\lceil \lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2 N_3} \rceil \frac{2^i}{B_m} \right\rceil + \sum_{i=0}^{n_3 - 1} \left\lceil \lceil \frac{P}{N_1} \rceil \lceil \frac{R}{N_2} \rceil \frac{2^i}{N_3 B_m} \right\rceil \right\} \tau,
\end{aligned}
$$

and for $PQR \leq N$ is

$$
T_{comm} = (\log PQR + 2 \log Q)\tau + (\log PQR + 2 \log Q) t_c.
$$

In analyzing the arithmetic complexity, we consider three cases:

1. $PR \geq N$: The arithmetic complexity is $\frac{2PQR}{N} t_a$ if $N_1 \leq P$, $N_2 \leq R$ and $N_3 \leq Q$.
2. $PQR \geq N > PR$: The arithmetic complexity is $(\frac{2PQR}{N} + \log \frac{N}{PR}) t_a$, if $N_1 \leq P$, $N_2 \leq R$ and $N_3 \leq Q$. Note that it is sufficient to minimize the arithmetic time for any N_3 satisfying $\frac{N}{PR} \leq N_3 \leq Q$. This can be derived from the above arithmetic complexity by assuming $\frac{PR}{N_1 N_2}$ = some constant.
3. $N > PQR$: The arithmetic complexity is $(2 + \log Q) t_a$. The efficiency is $\frac{2}{2 + \log Q}$ for $N = PQR$.

In summary, the minimum arithmetic complexity is

$$
(\frac{2PQR}{N'} + \log \lceil \frac{N'}{PR} \rceil) t_a \quad \text{where } N' = \min(N, PQR).
$$

When P, Q and R are multiples of N_1, N_2 and N_3 respectively, the optimum packet size

$$
B_{opt} = \max(\frac{QR}{2N_2N_3}, \frac{PQ}{2N_1N_3}, \frac{PR}{2N_1N_2}).
$$

The minimum number of start-ups is $n + 2n_3$. The data transfer time is at most

$$
\frac{3n_3Q(P+R)}{4N} + (N_1 - 1)\frac{QR}{N} + (N_2 - 1)\frac{PQ}{N} + (N_3 - 1)\frac{PR}{N}.
$$

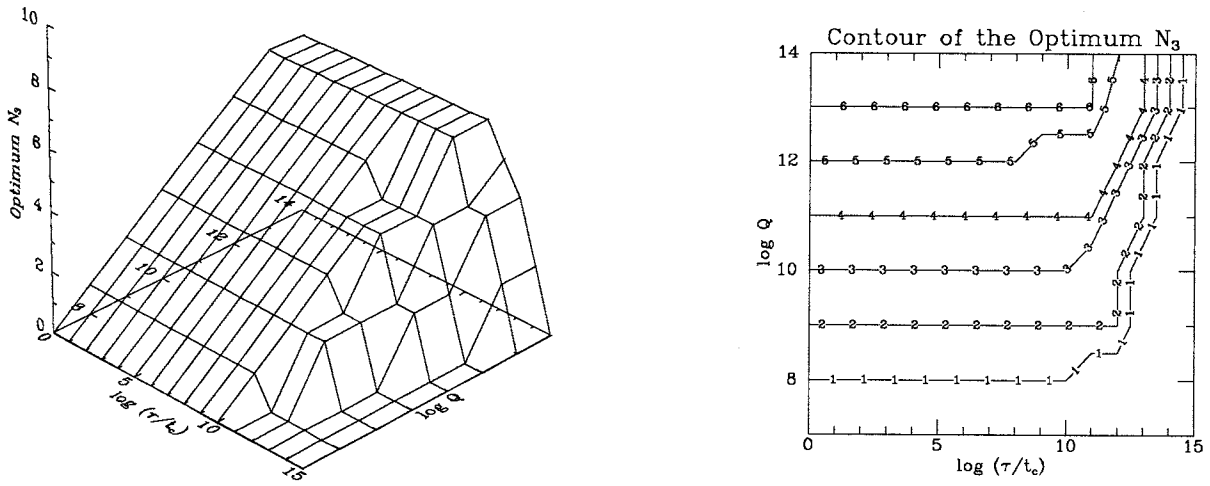


Figure 26: Optimum values of N_3 as a function of $\log \frac{\tau}{t_c}$ and aspect ratios of the matrices; $PQR = 1024^3$, $P = R$ or $P = 2R$, $B_m = \infty$. The right plot is the contour of the left plot.

The optimum values of N_1, N_2 and N_3 for algorithm $\mathcal{A}(1,3,1)$ with respect to the data transfer time are $\sqrt[3]{\frac{P^2N}{QR}}$, $\sqrt[3]{\frac{R^2N}{PQ}}$ and $\sqrt[3]{\frac{Q^2N}{PR}}$, respectively, by considering the higher order terms only. With the optimum values of N_1, N_2, N_3 and $B_m \geq B_{opt}$, the minimum total complexity becomes

$$\begin{aligned} T_{min} \approx & \left\{ 2\lceil \frac{PQR}{N} \rceil + \log(\min\,(Q, \lceil \frac{N}{PR} \rceil)) \right\} t_a \\ & + \left\{ 3(\frac{PQR}{N})^{\frac{2}{3}} - \frac{PQ+QR+PR}{N} + \frac{3n_3Q(P+R)}{4N} \right\} t_c \\ & + (n+2n_3)\tau. \end{aligned}$$

Figures 26 and 27 show the optimum values of N_3 with various parameters. In Figure 26 the number of processors is fixed, and the ratio of τ to t_c and Q to P (and R) varied. The optimum N_3 increases as $\frac{Q}{P}$ increases. For a sufficiently large $\frac{\tau}{t_c}$, the advantage of the third dimension is offset by the extra $2n_3$ start-up times. Therefore, the optimum N_3 decreases. The optimum N_3 is independent of t_a in the $PR > N$ domain. Figure 27 shows the optimum N_3 as a function of $\frac{\tau}{t_c}$, matrix size (for upper plots) and $\frac{t_a}{t_c}$ (for lower plots). The optimum N_3 in the upper plots is independent of t_a because $PR \geq N$. The lower plots consider the case where $PR < N$. Increasing N_3 up to $\min(\frac{N}{PR}, Q)$ will increase the processor utilization. Increasing N_3 up to $\sqrt[3]{N}$ will decrease the data transfer time for $P = Q = R$ (by ignoring the lower order term). These advantages will be offset by the extra start-up time, $2n_3\tau$ and, to a less extent, the ignored lower order term of the data transfer time, $\frac{3n_3Q(P+R)}{4N}$.

The optimum number of processors, N_{opt}, can be derived as

$$N_{opt} \approx PQR \cdot \min\,(1, \frac{2t_a}{(1+\frac{2n_3}{n})\tau}), \qquad \text{for } t_c \ll t_a.$$

Increasing N (up to PQR) decreases arithmetic time and data transfer time, but will increase the start-up time. The ratio of element transfers to arithmetic operations is $\frac{3}{2\sqrt[3]{\frac{PQR}{N}}}$.

For algorithms $\mathcal{A}(1,3,2)$ to $\mathcal{A}(1,3,5)$, the optimum value of N_3 is one, and the optimum values of N_1 and N_2 are the same as in the two-dimensional case. The complexity of algorithm $\mathcal{A}(1,3,1)$ is at most equal to that of $\mathcal{A}(1,2,2)$ to $\mathcal{A}(1,2,5)$.

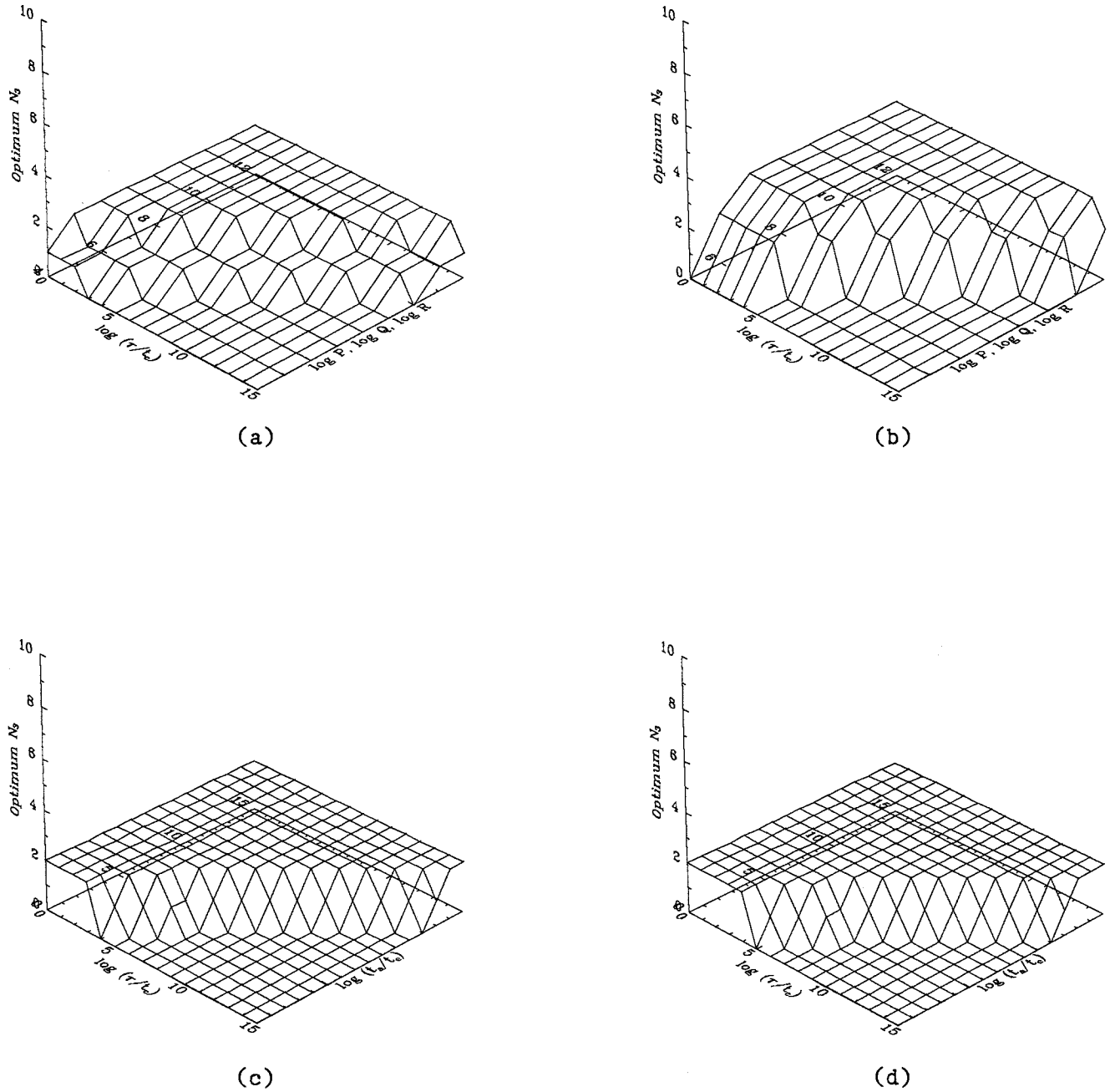


Figure 27: Optimum values of N_3 as a function of $\log \frac{\tau}{t_c}$, $\log \frac{t_a}{t_c}$ and matrix size with $B_m = \infty$. (a) $N = 128$, $P = Q = R$. (b) $N = 1024$, $P = Q = R$. (c) $N = 1024$, $P = Q = R = 16$. (d) $N = 4096$, $P = Q = R = 32$.

5 Conclusions

- For the one-dimensional partitioning algorithms $\mathcal{A}(\cdot,1,1)$, $\mathcal{A}(\cdot,1,2)$, and $\mathcal{A}(\cdot,1,3)$ yield inner-products of the highest order, and algorithms $\mathcal{A}(\cdot,1,1)$ and $\mathcal{A}(\cdot,1,4)$ AXPY of maximum vector length.
- For P, Q and R multiples of N, one-, two- and three-dimensional partitionings yield the same linear speedup with respect to arithmetic.
- For PQ, QR and PR multiples of N and $\max(PQ, QR, PR) \geq N > \max(P, Q, R)$, two- and three-dimensional partitionings yield the same linear speedup with respect to arithmetic. The one-dimensional partitioning is inferior.
- For $PQR \geq N > \max(PQ, QR, PR)$, a three-dimensional partitioning yields a lower arithmetic complexity than the two-dimensional partitioning, which in turn has a lower complexity than the one-dimensional partitioning.
- With the optimum aspect ratio for the partitioning, the number of elements transferred are approximately $\min(PQ,\ QR,\ PR)$, $2\sqrt{\frac{\min(P^2QR, Q^2PR, R^2PQ)}{N}}$ and $3\sqrt[3]{(\frac{PQR}{N})^2}$ for one-, two- and three-dimensional partitioning, respectively. For square matrices, the number of elements transferred compares as 1, $\frac{2}{\sqrt{N}}$ and $\frac{3}{\sqrt[3]{N^2}}$ approximately. With a bounded buffer size, the communication complexity is propotional to the number of element transfers.
- The minimum number of start-ups for algorithms $\mathcal{A}(\cdot,1,1)$ and $\mathcal{A}(\cdot,2,1)$ are the same, and so is that of $\mathcal{A}(\cdot,1,3)$ and $\mathcal{A}(\cdot,2,3)$. Algorithm $\mathcal{A}(\cdot,2,4)$ may have n fewer start-ups than algorithm $\mathcal{A}(n,1,4)$. The number of start-ups in the three-dimensional case may be lower than in the one- and two-dimensional cases.
- For the one-dimensional partitioning the algorithm that minimizes the complexity is mostly the one that have the processors aligned with the largest dimension, i.e., algorithm $\mathcal{A}(\cdot,1,1)$ for $R > P, Q$, algorithm $\mathcal{A}(\cdot,1,3)$ for $P > Q, R$, and algorithm $\mathcal{A}(\cdot,1,4)$ for $Q > P, R$.
- For the two-dimensional partitioning the algorithm that minimizes the complexity mostly has the normal for the processing plane in which the multiplication takes place aligned with the direction of minimum data points. Hence, algorithm $\mathcal{A}(\cdot,2,1)$ is chosen if $Q < P, R$, algorithms $\mathcal{A}(\cdot,2,2)$ or $\mathcal{A}(\cdot,2,5)$ if $P < Q, R$ and algorithms $\mathcal{A}(\cdot,2,3)$ or $\mathcal{A}(\cdot,2,4)$ if $R < P, Q$.
- The optimum aspect ratio of the two-dimensional partitioning is mostly dependent upon the ratio of the number of elements in the two dimensions spanning the plane in which the multiplications take place, i.e., $\frac{P}{R}$ for algorithm $\mathcal{A}(\cdot,2,1)$, $\frac{Q}{R}$ for algorithm $\mathcal{A}(\cdot,2,2)$, $\frac{Q}{P}$ for algorithm $\mathcal{A}(\cdot,2,3)$, $\frac{P}{Q}$ for algorithm $\mathcal{A}(\cdot,2,4)$ and $\frac{R}{Q}$ for algorithm $\mathcal{A}(\cdot,2,5)$. The optimum aspect ratio of the three-dimensional partitioning is mostly dependent upon $P : Q : R$.
- With the optimum algorithm with respect to matrix shape the temporary storage requirements and the optimum packet size are minimized.
- For algorithm $\mathcal{A}(1, *, 1)$,

$$N^{3d}_{opt} : N^{2d}_{opt} : N^{1d}_{opt} \approx PQR \cdot \min\ (1, \frac{2t_a}{(1+\frac{2n_3}{n})\tau}) : PR \cdot \min\ (1, \frac{2Qt_a}{\tau}) : R \cdot \min\ (1, \frac{2PQt_a}{\tau}).$$

  If matrix C is initially partitioned into N_1 by N_2N_3 blocks and matrix D is initially partitioned into N_1N_3 by N_2 blocks where $\frac{P}{N_1} = \frac{R}{N_2} = \frac{Q}{N_3}$, then $N^{3d}_{opt} \approx PQR \cdot \min\ (1, \frac{2t_a}{\tau})$.
- The same complexity results hold for both a binary-reflected Gray code encoding and a Binary encoding of the matrix elements. It also holds for both consecutive storage and cyclic storage [7].

With a temporary storage or the maximum packet size equal to the size of the partitioned matrices, and *one-port* communication, $N-1$ communication steps are required for the one dimensional partitioning while only

$O(N_1, N_2)$ and $O(N_1, N_2, N_3)$ communication steps are required for the two- and three-dimensional partitioning respectively. For square matrices and *one-port* communication, the communication complexity compares as N, $2\sqrt{N}$ and $3\sqrt[3]{N}$ for one-, two- and three-dimensional partitioning respectively.

For the one-dimensional partitioning, the CRA and the GCEA algorithms require constant storage, and for the two-dimensional partitioning, Cannon's and Dekel's algorithms have the same property. The Spanning Binomial Tree algorithm only requires $\log N$ communication steps at the expense of larger (exponentially growing) temporary storage. With limited temporary storage (and maximum buffer size), a hybrid method can be used by performing k steps of the SBT algorithm and $2^{n-k} - 1$ steps of a linear time algorithm. The number of communication steps can be halved, approximately, by doubling the temporary storage and the optimum buffer size. With *n-port* communication, the edge load is reduced by a factor of n. The start-up time becomes more significant.

Acknowledgement

This work has been supported in part by the Office of Naval Research under Contracts N00014-84-K-0043 and N00014-86-K-0564.

References

[1] L.E. Cannon. *A Cellular Computer to Implement the Kalman Filter Algorithm.* PhD thesis, Montana State University, 1969.

[2] Eliezer Dekel, David Nassimi, and Sartaj Sahni. Parallel matrix and graph algorithms. *SIAM J. Computing*, 10:657–673, 1981.

[3] Ching-Tien Ho and S. Lennart Johnsson. Algorithms for matrix transposition on Boolean n-cube configured ensemble architectures. In *Int. Conf. on Parallel Processing*, pages 621–629, IEEE Computer Society, 1987.

[4] Ching-Tien Ho and S. Lennart Johnsson. *Matrix Transposition on Boolean n-cube Configured Ensemble Architectures.* Technical Report YALEU/DCS/RR-494, Yale University, Dept. of Computer Science, September 1986.

[5] Ching-Tien Ho and S. Lennart Johnsson. *Spanning Balanced Trees in Boolean cubes.* Technical Report YALEU/DCS/RR-508, Yale University, Dept. of Computer Science, January 1987.

[6] *Intel iPSC System Overview.* Intel Corp., January 1986.

[7] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *Journal of Parallel and Distributed Computing*, 4(2):133–172, April 1987. (Report YALEU/DCS/RR-361, January 1985).

[8] S. Lennart Johnsson. *Data Permutations and Basic Linear Algebra Computations on Ensemble Architectures.* Technical Report YALEU/DCS/RR-367, Yale University, Dept. of Computer Science, February 1985.

[9] S. Lennart Johnsson and Ching-Tien Ho. Matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. on Algebraic and Discrete Methods.* To appear. Revised edition of YALEU/DCS/RR-494.

[10] S. Lennart Johnsson and Ching-Tien Ho. *Spanning Graphs for Optimum Broadcasting and Personalized Communication in Hypercubes.* Technical Report Report YALEU/DCS/RR-500, Yale University, Dept. of Computer Science, November 1986. To appear in IEEE Trans. Computers.

[11] Flynn M.J. Very high-speed computing systems. *Proc. of the IEEE*, 12:1901–1909, 1966.

[12] E M. Reingold, J Nievergelt, and N Deo. *Combinatorial Algorithms.* Prentice Hall, 1977.

[13] Yousef Saad and Martin H. Schultz. *Data Communication in Hypercubes.* Technical Report YALEU/DCS/RR-428, Dept. of Computer Science, Yale University, October 1985.

[14] Yousef Saad and Martin H. Schultz. *Topological properties of Hypercubes.* Technical Report YALEU/DCS/RR-389, Dept. of Computer Science, Yale University, June 1985.