Yale University
Department of Computer Science

Matrix Multiplication on Boolean Cubes Using Generic Communication Primitives

S. Lennart Johnsson and Ching-Tien Ho

YALEU/DCS/TR-530
September 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.

† A revised edition of TR-530, March 1987. To appear in Proceedings of ARO workshop on Parallel Processing and Medium-Scale Multiprocessors, 1986.

Matrix Multiplication on Boolean Cubes Using Generic Communication Primitives

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

Abstract. Generic primitives for matrix operations as defined by the level one, two and three of the BLAS are of great value in that they make user programs much simpler, and hide most of the architectural detail of importance for performance in the primitives. We describe generic shared memory primitives such as *one-to-all* and *all-to-all broadcasting*, and *one-to-all* and *all-to-all personalized communication*, and implementations thereof that are within a factor of two of the best known lower bounds. We describe algorithms for the multiplication of arbitrarily shaped matrices using these primitives. Of the three loops required for a standard matrix multiplication algorithm expressed in Fortran all three can be parallelized. We show that if one loop is parallelized, then the processors shall be aligned with the loop with the maximum number of matrix elements for the minimum arithmetic *and* communication time. In parallelizing two loops the processing plane shall be aligned with the loops having the most elements. Depending on the initial matrix allocation data permutations may be required to accomplish the processor/loop alignment. This permutation is included in our analysis. We show that in parallelizing two loops the optimum aspect ratio of the processing plane is equal to the ratio of the number of matrix elements in the two loops being parallelized.

1 Introduction

One of the most frequent operations in scientific and engineering computations is multiplication of matrices. We analyze this problem for arbitrary matrix shapes and Boolean n-cube configured *ensemble architectures* [22] and express the algorithms in generic communication primitives.

It is well known that for the classical multiplication algorithms requiring $PR(2Q-1)$ arithmetic operations for the multiplication of a $P \times Q$ matrix by a $Q \times R$ matrix, the access scheme has a significant influence on the actual running time for most architectures. Performance variations of up to one order of magnitude have been observed. Some access schemes make effective use of pipelining, while others do not. Some have locality of access, making good use of cache based architectures, others not. Some also result in accesses to the same storage bank at a rate causing the performance to be determined by storage bank bandwidth instead of the total storage or processor bandwidths.

In the Boolean n-cube architectures we consider, storage is uniformly distributed among nodes of identical architecture. The global architecture can be considered homogeneous. We also assume that the matrices to be multiplied are uniformly distributed throughout the machine. For a multiprocessor with few nodes relative to the total number of matrix elements, the time for the arithmetic operations, and local storage operations, are likely to dominate over the interprocessor communication times. 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 $\sqrt{M}$. The ratio between elementary arithmetic operations and element communications approaches 1 as the aspect ratio of the blocks increases. As the number of processors increases relative to the matrix size, the importance of efficient communication increases.

The *granularity* is said to be *fine* if there are only a few elements per processor. Different data allocations and data movements may in such cases result in significantly different processor utilization and communication time. The emphasis of this paper is on the communication efficiency. The problem of efficient communication can be studied in the context of embedding one graph, the *guest graph*, in another, the *host graph*. The first graph captures the communication needs, or data dependencies, of the algorithm, the other models the processor ensemble. The communication needs of the classical matrix multiplication algorithms imply some form of broadcast operation. For instance, in computing $A \leftarrow C \times D + E$, every element of a column of C multiplies every element of the corresponding row of D. The guest graphs that implement some generic communication patterns that we consider are linear and multidimensional arrays, and a variety of spanning graphs that yield lower bound communication for different kinds of communication operations, or different capabilities of the hardware. The topological properties of the hardware is captured by the host graph.

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) [18], or each node, or a subset thereof, may have their own instruction stream, resulting in a MIMD (Multiple Instruction Multiple Data). We present algorithms suitable for both kinds of architectures. In the Boolean cube configured architectures there is a nonuniformity in the distance from a processor to other processors, or storage modules. This nonuniformity offers the potential for performance enhancements through the exploitation of locality.

The outline of this paper is as follows. In the next section the notation and definitions used throughout the paper are introduced. Linear arrays and two-dimensional meshes can be simulated on Boolean cubes without a slow down. The embedding of such graphs are discussed in Section 3. Section 4 presents some spanning graphs and the complexity of *one-to-all broadcasting* and *all-to-all broadcasting* under various conditions. This section also contains a discussion of *one-to-all personalized communication* and *all-to-all personalized communication*. Section 5 presents matrix multiplication algorithms based on a few generic communication primitives both for one- and two-dimensional partitioning of the matrices, and a complexity analysis. Section 6 gives a summary and conclusions. Some of the estimated communication complexities have been verified through measurements on the Intel iPSC [13].

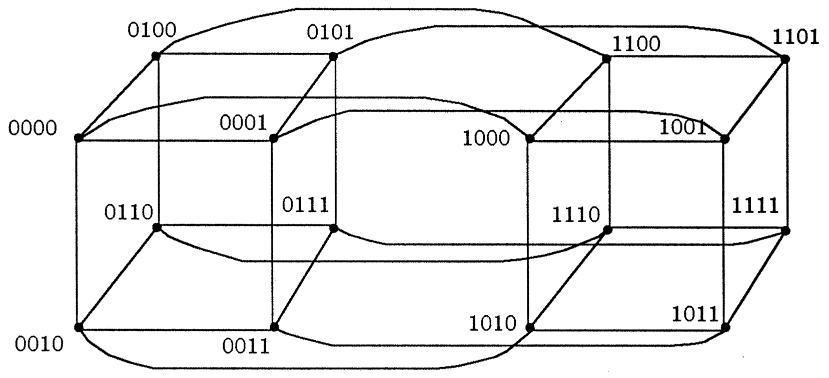


Figure 1: A recursive construction of Boolean cubes.

2 Notation and Definitions

Throughout the paper N denotes the number of processors. For a two-dimensional mesh $N = N_1 \times N_2$ and for a Boolean n-cube $N = 2^n$. We consider the multiplication of a dense matrix of size $P \times Q$ by a matrix of size $Q \times R$. The diameter of an $N_1 \times N_2$ mesh is $N_1 + N_2 - 2$, if there is no wrap-around, and is $\lfloor \frac{N_1}{2} \rfloor + \lfloor \frac{N_2}{2} \rfloor$ otherwise. There are 4 paths between any pair of "internal nodes" in a mesh. For i and j in different rows and columns, the length of two of these paths is $|i - j|_1$ for nodes i and j. $|\cdot|_1$ denotes the 1-norm. The other two paths are of length $|i - j|_1 + 4$.

A Boolean n-cube has diameter $n = \log N$, $\binom{n}{i}$ nodes at distance i from a given node, and n disjoint paths between any pair of nodes. The paths are either of the same length as the *Hamming* distance between the end points of the paths, or the *Hamming* distance plus two [21]. The fanout of every node is n, and the total number of communication links is $\frac{1}{2}N \log N$. The average distance between nodes is $\frac{1}{2}n$. A Boolean n-cube can be constructed recursively by joining corresponding nodes of two $(n-1)$-cubes. It follows that nodes can be given addresses such that adjacent nodes differ in precisely one bit, Figure 1. The distance between a pair of nodes i and j is equal to the *Hamming* distance between the nodes, where $Hamming(i,j) = |i \oplus j|$, and $\oplus$ is the bit-wise exclusive-or operator, and $|i|$ denotes the number of bits of i that is equal to one.

For the embedding of a *guest graph* in a *host graph*, an edge of the guest graph is, in general, mapped onto a path in the host graph. Let $p(i,j)$ be the path length between nodes i and j. Define the *edge dilation* of an edge (i,j) in the guest graph with respect to the mapping function ϕ to be $p(\phi(i), \phi(j))$. Hence, if the edge dilation is one for every edge in the guest graph, then the communication time for an algorithm on the host graph is the same as on the guest graph, assuming that $\phi(i) \neq \phi(j)$, if $i \neq j$. In many cases there is a trade-off between the maximum edge dilation, or average edge dilation, and the *expansion* = (*number of nodes in the host graph*) / (*the number of nodes in the guest graph*).

Different Boolean cube configured architectures are capable of supporting concurrent communication on a different number of ports. In *one-port* communication, communication can only take place on one port at a time for each processor. In *n-port* communication all ports on each processor can be used concurrently. The communication time is denoted $T(\cdot,\cdot)$, where the first argument refers to the number of ports used concurrently, and the second to the number of spanning graphs used concurrently, i.e., the number of source nodes for the broadcasting or personalized communication. The routing schemes are indexed similarly. T_{lb} denotes lower bound estimates.

In the following we also use the notions of *one-to-all broadcasting*, and *all-to-all broadcasting*. 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-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*. 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.

The Yale version of the Intel iPSC is a 64 node multiprocessor usually configured as two 5-dimensional Boolean cubes. It has a message passing programming model. Up to $16k$ bytes can be passed in each communication, but the operating system subdivides messages of a size greater than $1k$ byte into $1k$ byte packets. We refer to the user defined packets as *external packets* and the operating system defined packets as *internal packets*. The size of the internal packets is denoted B_m. We denote the time for an arithmetic operation by t_a, the time for communication of a floating-point number (4 bytes) by t_c, and the start-up time for a communication by τ. With the initial operating system we recorded a start-up time of $\tau \approx 8$ *msec* for each external packet, and a start-up time of $\tau \approx 6$ *msec* for each internal packet. In a second version of the operating system the start-up time for external packets was reduced to $\tau \approx 5$ *msec*, and with the current operating systems, NX, the start-up time for external packets is reduced to $\tau \approx 1.5$ *msec*. Although there are n ports per processor in an n-cube, the storage bandwidth is only sufficient to support 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.

3 Embedding of Arrays in Boolean Cubes

We assume that successive nodes of a path are labeled with successive integers with the least label being 0. Mapping the array nodes to processors according to the binary encoding of the integers does *not* preserve proximity. The binary encodings of $\frac{N}{2}$ and $\frac{N}{2}-1$ differ in all bits and are assigned to processors at distance n. However, it is well known that the Boolean cube is Hamiltonian. The Gray code has the property that the binary codes of successive integers differ in precisely one bit. The *binary-reflected Gray code* can be defined recursively as shown below [19]. Let G_i^j be the j-bit Gray code of i, and $G(n)$ be the entire (cyclic) sequence of n-bit Gray code numbers (of length 2^n). Then, $G(n)$ can be represented in matrix form as

$$G(n) = \begin{pmatrix} G_0^n \\ G_1^n \\ \vdots \\ G_{2^n-2}^n \\ G_{2^n-1}^n \end{pmatrix}.$$

$$\text{Then } G(n+1) = \begin{pmatrix} 0G_0^n \\ 0G_1^n \\ \vdots \\ 0G_{2^n-2}^n \\ 0G_{2^n-1}^n \\ 1G_{2^n-1}^n \\ 1G_{2^n-2}^n \\ \vdots \\ 1G_1^n \\ 1G_0^n \end{pmatrix}, \text{ or alternatively, } G(n+1) = \begin{pmatrix} G_0^n0 \\ G_0^n1 \\ G_1^n1 \\ G_1^n0 \\ G_2^n0 \\ G_2^n1 \\ \vdots \\ G_{2^n-1}^n1 \\ G_{2^n-1}^n0 \end{pmatrix}.$$

Moreover, it is *cyclic* in that $Hamming(G_0^k, G_{2^k-1}^k) = 1$. The cyclic property means that loops with 2^n nodes can be embedded in an n-cube with edge *dilation* one and *expansion* one. Moreover, any loop of even length can be embedded with dilation one. Any odd length loop must have at least one edge of dilation two, when embedded in the cube [14]. The expansion for a loop with N_L nodes embedded in an n-cube is $\frac{N}{N_L}$, which is at most $2 - \frac{4}{N+2}$.

A multidimensional array $N_1 \times N_2 \times \ldots N_r$ can be embedded in a Boolean cube preserving adjacency by simply partitioning the address space of the cube node addresses such that $\lceil \log_2 N_i \rceil$ bits are assigned to the embedding of the nodes in dimension i of the array. The nodes in each dimension are embedded using a Gray code. With this simple embedding the expansion is 1 if $N_i = 2^{n_i}, \quad \forall i \in \{1, 2, \ldots, r\}$, but can be as high as $\prod_{i=1}^{r}(2 - \frac{4}{N_i+2})$. Reduced expansion can be obtained at the expense of increased dilation [2,14,11].

4 Spanning Graphs, Broadcasting, and Personalized Communication

Matrix multiplication, as many other linear algebra algorithms, implies a broadcasting of elements from one location to all other locations, or a subset of other locations. A degenerate form of spanning tree is a Hamiltonian path. Such a tree has a maximum height. The path can be generated by a binary-reflected Gray code. The minimum time for broadcasting is clearly bounded from below by the length of the longest path. To reduce the time for broadcasting compared to that given by a Hamiltonian path, it is necessary to use a spanning tree with a shorter longest path; for instance, a tree of minimum height. A *Spanning Binomial Tree* [7,1,14] is such a tree. However, broadcasting based on such a spanning tree does not necessarily minimize the time [17] for broadcasting. We will now briefly describe the embedding of Hamiltonian paths, Spanning Binomial Trees, n Edge-Disjoint Spanning Binomial Trees, Spanning Balanced n-Trees, and n Rotated Spanning Binomial Trees. The communication complexities for routing according to these different spanning graphs are given. For details of the derivations see [17]. The different graphs are all optimal for at least one of the algorithms considered here. The data set to be communicated to a node from the source is M. Hence, in broadcasting the data set in the source is M, but in personalized communication it is $(N-1)M$.

4.1 One-to-All Broadcasting

The matrix multiplication algorithms we will describe all require some form of broadcasting, in most cases *all-to-all broadcasting*, but since the *all-to-all broadcasting* is a composition of one-to-all broadcasting we first describe the simpler case. The communication complexities are summarized in Table 1.

4.1.1 Hamiltonian Paths

A Hamiltonian path is the simplest form of broadcasting tree. With *one-port* communication, the time for broadcasting is proportional to the time for the first packet to propagate to the furthest node, and the number of subsequent packets arriving during every other cycle. With a Hamiltonian path generated by a binary-reflected Gray code the distribution of edges over the different dimensions is $\frac{1}{2}N$, $\frac{1}{4}N$, ..., 1. We denote the Hamiltonian path routing based on a binary-reflected Gray code by GC. The estimated routing time is given in Table 1.

If *n-port* communication is possible, n different paths can be generated by rotating the dimensions used in the Gray code such that for path i dimension k of the initial path is translated into dimension $(i + k) \bmod n$. For instance, let sequence 0 be the original sequence, then the three rotated sequences for a 3-cube are as follows: *sequence* $0 = (0,1,0,2,0,1,0)$, *sequence* $1 = (1,2,1,0,1,2,1)$, and *sequence* $2 = (2,0,2,1,2,0,2)$. For *n-port* communication the data set M is partitioned into n sets, and each set broadcasted along a separate path. However, the n paths are not edge-disjoint, and several messages along the same path cannot be pipelined with messages along all other paths. For instance, it can be shown that there does not exist 3 (directed) edge-disjoint Hamiltonian circuits in a 3-cube, but there exist 4 (directed) edge-disjoint Hamiltonian circuits in a 4-cube.

4.1.2 Spanning Binomial Trees

The most commonly used spanning tree for broadcasting in a Boolean cube, or the reverse operation, reduction, as in inner product computations, is a *Spanning Binomial Tree* (SBT). An n-level binomial tree can be constructed recursively by adding one edge between the roots of two $(n-1)$-level binomial trees, and making either root the new root. A 0-level binomial tree has one node. It follows that:

- An n-level binomial tree has $\binom{n}{i}$ nodes at level i, i.e., the same as the number of nodes at distance i form a node in a Boolean cube.
- The n-level binomial tree is composed of n subtrees each of which is a binomial tree of $0, 1, \ldots, n-1$ levels respectively. The k-level subtree has 2^k nodes.

Figure 2 shows a 5-level binomial tree, which is also a spanning tree of the 5-cube.

For *one-port* communication the source node first sends a packet to the largest subtree, then the broadcasting task is effectively reduced to broadcasting in two same-sized binomial trees, each having half the number of nodes of the original tree. The number of routing steps is minimal for a maximum packet size $B_m \geq M$. However, the bandwidth is not used effectively. A lower bound for the broadcasting time is $T_{l\,b}(1,1) = \max(Mt_c, n\tau)$. The minimum time for SBT(1,1) routing is higher than the lower bound by a factor of n for the data transmission time.

With *n-port* communication, pipelining can be employed. The routing time is reduced approximately by a factor of n, Table 1.

4.1.3 Edge-Disjoint Spanning Binomial Trees

A spanning binomial tree uses $N-1$ of a total of $\frac{N}{2}\log N$ or $N \log N$ communication channels in the cube. Increased use of the bandwidth of the cube can be accomplished by dividing the data set up into n pieces, sending each such piece to a distinct neighbor of the source, and then

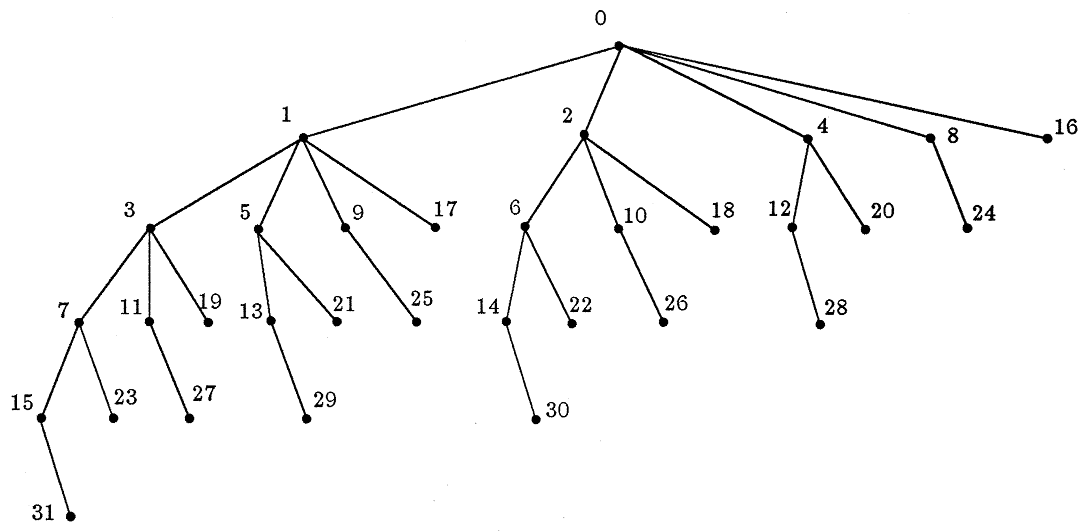


Figure 2: A 5-level binomial tree.

performing a concurrent broadcast from these n secondary sources. Each such broadcast can be done using SBT routing. If these n SBTs are distinct rotations of each other, then it can be shown that they are edge-disjoint [9]. We refer to this spanning graph as an nESBT graph for *n Edge-disjoint Spanning Binomial Trees.* It uses $(N-1)\log N$ (directed) edges, i.e., all cube edges except those directed towards the source with bidirectional internode communication. Furthermore, it is possible to devise scheduling disciplines for *one-port* communication such that no contention for communication channels occurs [9]. The speed-up of the nESBT(1,1) routing over the SBT(1,1) routing is n for $\frac{M}{B_m} \gg n$.

Note that the nESBT routing requires less temporary storage than SBT routing. The optimum packet size for SBT routing is larger by a factor of $\sqrt{nM\frac{t_c}{\tau}}$. Note that if $M \leq \frac{\tau}{nt_c}$ then the optimum packet size, B_{opt}, is M. So, if $M \leq \frac{\tau}{nt_c}$ and $B_m \geq M$, then the SBT is superior to the nESBT by one routing step. Note further that the routings SBT(n,1) and nESBT(1,1) are approximately equal in complexity, as are the corresponding optimal packet sizes.

The Intel iPSC effectively is restricted to communication on one port at a time. Figure 3 shows the measured times for SBT and nESBT routing as well as the relative speed-up as functions of cube dimensions.

A lower bound for *n-port one-to-all broadcasting* is $T_{lb}(n,1) = \max(\frac{M}{n}t_c, n\tau)$. The nESBT$(n,1)$ routing is higher than this lower bound by at most a factor of 2 (as in the *one-port* case). The nESBT$(n,1)$ routing is faster than the SBT$(n,1)$ routing by a factor of n, if $\frac{M}{B_m} \gg n$. The optimum packet size for the SBT$(n,1)$ routing is larger than that of the nESBT$(n,1)$ routing by a factor of approximately $\sqrt{n}$.

4.1.4 Rotated Spanning Binomial Trees

A higher edge utilization than in the SBT graph is obtained by superimposing n distinctly rotated spanning binomial trees. The data set is divided into n sets, and the load on the edges

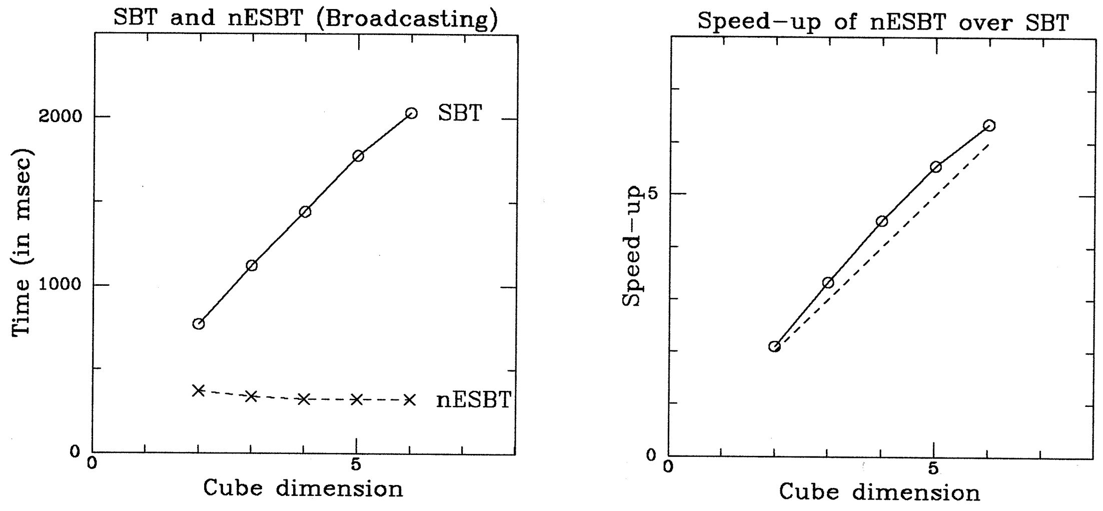


Figure 3: Measured broadcast times of the SBT and nESBT routings on the Intel iPSC for a $60k$ bytes message with packet size $1k$ bytes. On the right, the dashed line means speed-up $= \log N$.

from the source is perfectly balanced if $M \bmod n = 0$. We call this form of spanning graph nRSBT for *n Rotated Spanning Binomial Trees* [17]. The different instances of the SBTs are clearly not edge-disjoint. The maximum edge-load, i.e., the maximum number of elements that traverse an edge is the same as for the SBT graph, and a factor of n higher than for the nESBT graph. Concurrent communication on n ports reduces the data transfer time by a factor of n, and the number of start-ups by a factor of approximately 2 compared to *one-port* communication. Though the nRSBT$(*, 1)$ routing is not competitive with the nESBT$(*, 1)$ routing, it is an effective routing scheme for *all-to-all broadcasting* as described in the next section.

4.1.5 Spanning Balanced n-Trees

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) [9,17,12]. 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.

Let $\mathcal{M}(i,j)$ be the maximum set of consecutive indices containing all the 0-bit positions immediately to the right of bit j in the binary encoding of i, cyclically. Bit 0 is the least significant bit. We also make use of the *base* of i defined as the number of right rotation of i that minimizes the rotated value. Let $R(i) = (a_0 a_{n-1} a_{n-2} \dots a_1)$, where $i = (a_{n-1} a_{n-2} \dots a_0)$. Furthermore, let $J = \{j_0, j_1, \dots, j_m\}$, $j_0 < j_1 < \dots j_m$, where $R^j(i) < R^k(i)$, for all $j \in J$, $k \notin J$. Then $base(i) = j_0$. We use j to denote $base(i)$ in the following. For the definition of the *parent* and *children* functions we first find the position k of the first bit cyclically to the right of bit j ($j = base(i)$) that is equal to 1, i.e., $a_k = 1$, and $a_m = 0, \forall m \in \mathcal{M}(i,j)$. $k = j$ if

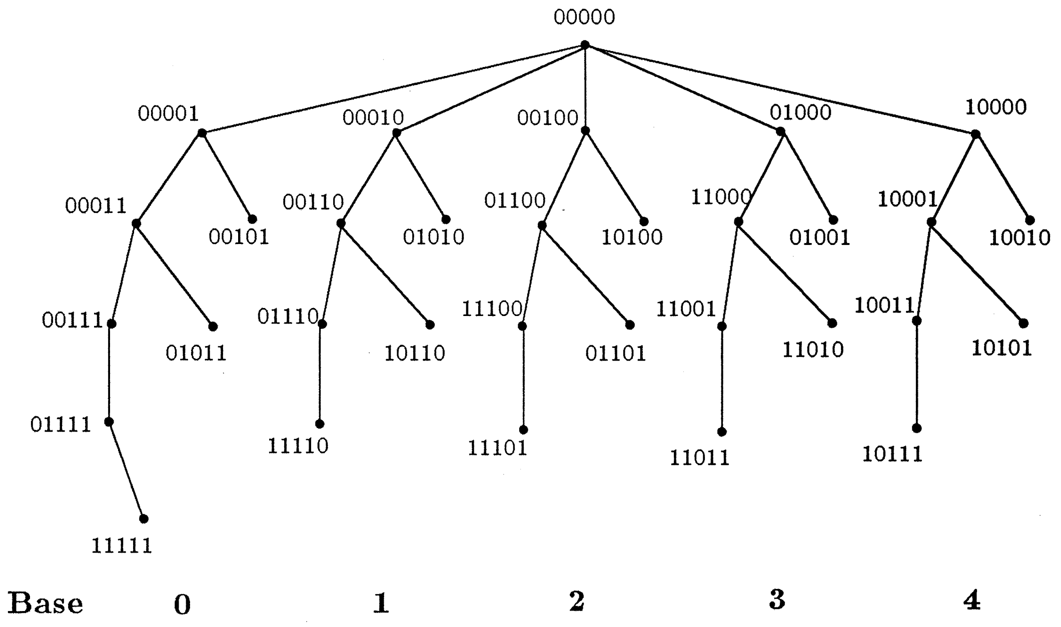


Figure 4: A *Spanning Balanced n-tree* in a 5-cube.

$i = (0...01_j0...0)$, and $k = -1$ if $i = 0$. Then

$$children_{SBnT}(i,0) = \begin{cases} \{(a_{n-1}a_{n-2}...\bar{a}_m...a_0)\}, \forall m \in \{0,1,\ldots,n-1\}, & \text{if } i = 0; \\ \{q_m = (a_{n-1}a_{n-2}...\bar{a}_m...a_0)\}, & \\ \qquad \forall m \in \mathcal{M}(i,j) \text{ and } base(q_m) = base(i), & \text{if } i \neq 0. \end{cases}$$

$$parent_{SBnT}(i,0) = \begin{cases} \phi, & \text{if } i = 0; \\ (a_{n-1}a_{n-2}...\bar{a}_k...a_0), & \text{otherwise.} \end{cases}$$

Figure 4 shows a SBnT for a 5-cube. For *one-to-all* broadcasting the SBnT routing is of higher complexity than the nESBT routing, but it is superior for *all-to-all* broadcasting.

4.1.6 Summary and Discussion

A complete binary tree cannot be embedded in a Boolean cube of the same size preserving adjacency [21,3,6,23], but it can be embedded such that only one tree edge is mapped into a path of length two, and the intermediate node is the "extra" node of the cube. Such a tree is referred to as a *Two-rooted Complete Binary Tree*. (A complete binary tree of $2^n - 1$ nodes can be embedded in an $(n+1)$-cube preserving adjacency [14,3]).

Note that routing according to a Hamiltonian path yields a lower complexity than the SBT routing if $\frac{M}{B_m} > \frac{N-3}{n-2}$ [17]. *One-to-all* broadcasting based on the two-rooted complete binary tree is inferior to the nESBT routing, but may be superior to the SBT routing and the Hamiltonian path routing. Routing based on the nRSBT graph is inferior to the SBT routing for *one-to-all* broadcasting for *one-port* communication, and is superior for *n-port* communication.

The nESBT(1,1) routing is of lowest complexity, except if $B_m \geq M$ and $M \leq \frac{\tau}{nt_c}$ in which case the SBT(1,1) requires one less cycle. Similarly, the nESBT$(n,1)$ routing is superior to

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

Table 1: Estimated *one-to-all broadcasting* times for various spanning graphs. The upper part is for *one-port* communication and the lower part is for *n-port* communication.

the other routings, except if $B_m \geq \frac{M}{n}$ and $M \leq \frac{\tau}{t_c}$ in which case the nRSBT(n, 1) routing is superior.

The *one-to-all broadcasting* complexity estimates are summarized in Table 1.

4.2 All-to-All Broadcasting

For matrix multiplication, $A \leftarrow C \times D + E$ every element of a row of D has to interact with every element of the corresponding column of C. *All* elements of a subset of elements are broadcasted to *all* elements of another subset. In the case of matrix multiplication the elements of the subsets can be aligned such that the elements in a subset have contiguous addresses. We assume this form of alignment in all of the algorithms we present. For each source, one of the spanning graphs described previously can be used. The instances can be pure *translations* of each other, or translations combined with some other operation such as *rotation*, *reflection*, or other *permutation*.

4.2.1 N Hamiltonian Paths

One possibility is to embed a single Hamiltonian path by a binary-reflected Gray code, and use this path for broadcasting all the elements. The path is used as a conveyor belt. Another alternative is to let the path for the elements of node i be a translation of the path for elements of say node 0. In the latter case the first edge of every path is in the same dimension, so is the second, etc. Figure 5 shows the paths for nodes 0 - 3 in a 3-cube.

We refer to the first routing as a *Cyclic Rotation Algorithm* (CRA), and the latter as a *Gray Code Exchange Algorithm* (GCEA), because the sequence of exchange dimensions is exactly the sequence of dimensions encountered in traversing the cube in the binary-reflected Gray code order. For a 3-cube, the sequence of dimensions is $(0, 1, 0, 2, 0, 1, 0)$. In the *cyclic rotation algorithm*, a node always sends to and receives from the same neighbors for all routing steps, whereas in the latter all the nodes have the same exchange sequence.

The *cyclic rotation algorithm* can be expressed as follows in pseudo code. In the following,

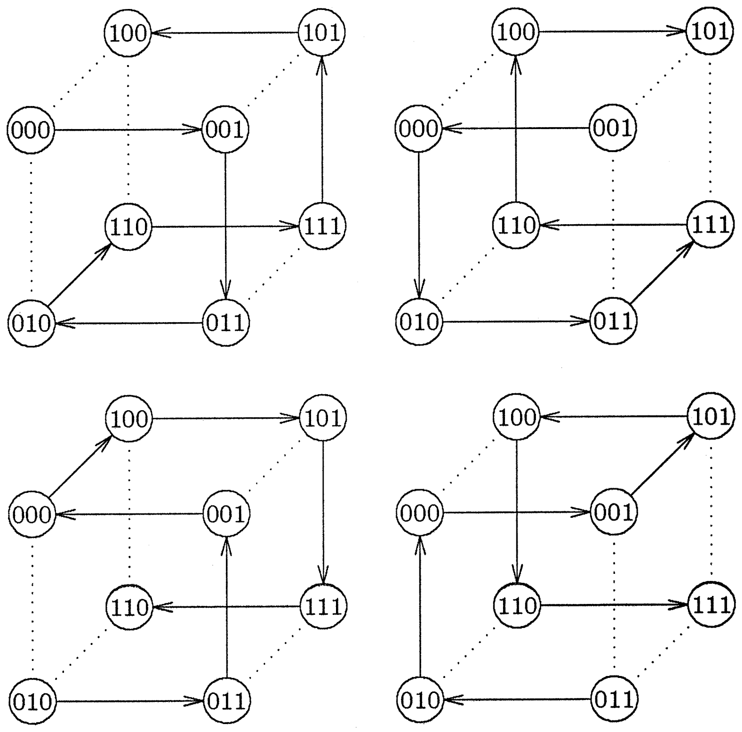


Figure 5: Four translated binary reflected Gray-code paths in a 3-cube.

G and IG represent the Gray code function and the inverse Gray code function respectively.

```
/* The Cyclic Rotation Algorithm, one-port communication: */
/* Let port [j] connect to the neighbor of dimension j. */
/* pid is the processor address in binary repr.*/

SUBROUTINE INIT_CRA
COMMON /CRA/ outport, inport
gid = IG (pid)
outport = port [j] where j is the rightmost 0-bit of gid.
inport = port [j] where j is the rightmost 0-bit of (gid+1) mod N
END

SUBROUTINE CRA (buf, length)
COMMON /CRA/ outport, inport
CALL SEND (outport, buf, length)
CALL RECV (inport, buf, length)
END
```

and the *Gray Code Exchange Algorithm* as

```
/* The Gray Code Exchange Algorithm, one-port communication: */

SUBROUTINE GCEA (i, buf, length)
j = the position of the rightmost 0-bit of i-1          /* lsb = position 0. */
CALL SEND (port [j], buf, length)
CALL RECV (port [j], buf, length)
END
```

With *one-port* communication both algorithms yield the same communication time. With *n-port* communication the generalization of the *cyclic rotation algorithm* leads to n (directed) Hamiltonian circuits. The *Gray code exchange algorithm* is generalized by using n distinctly rotated Gray code sequences. The paths of the CRA(n, N) algorithm are not edge-disjoint, and the load non-uniform. However, in the GCEA(n, N) algorithm, even though the paths are not edge-disjoint the load is even, and minimal.

The *Gray Code Exchange Algorithm* for *n-port* communication can be written as

```
/* The Gray Code Exchange Algorithm, n-port communication: */

SUBROUTINE GCEA_n (i, buf, length)
Split buf into n parts.
buf1 [k] = $k^{th}$ part of buf, k = 0, 1, ..., n-1.
j = the position of the rightmost 0-bit of i-1
FOR k = 0, 1, ..., n-1 DO concurrently
        CALL SEND (port [(j+k) mod n], buf1 [k], length/n)
        CALL RECV (port [(j+k) mod n], buf1 [k], length/n)
ENDFOR
END
```

4.2.2 Spanning Binomial Trees

In the *Gray code exchange algorithm* the data volume is the same in each exchange operation. An alternative exchange algorithm is obtained by noticing that after the first exchange each of the two subcubes can contain the entire data set, with each node containing its original data and the data of its neighbor in the exchange dimension. The process can be repeated recursively. Hence, n steps are needed instead of $N-1$ steps for the *Gray code exchange algorithm.* The penalty is a requirement for larger memory. In the final exchange half of the entire data set is involved, i.e., $\frac{1}{2}NM$. This alternate exchange algorithm [17,20] is equivalent to the embedding of N spanning binomial trees. The different trees are translations of each other. We refer to it as the SBT-b$(*, N)$ (-b for broadcasting). The maximum number of tree edges mapped to a cube edge is $\frac{1}{2}N$. In the i^{th} step of the *one-port* version $2^{i-1}M$ data elements are exchanged.

$T^{SBT}_{min}(1, N)$ is in fact proportional to the lower bound for *one-port* communication, since each processor needs to send/receive $(N-1)M$ elements. Note also that if $B_m \leq M$, then $T^{SBT}(1, N) = T^{GCEA}(1, N)$. In the GCEA$(1, N)$ algorithm, $\frac{M}{B_m}$ packets are exchanged along all interprocessor connections in a given dimension for each routing step. One dimension is used $\frac{1}{2}N$ times, another $\frac{1}{4}N$ times, etc. In the SBT-b$(1, N)$ algorithm each dimension is only routed once, but the number of packets are $\frac{M}{B_m}$, $2\frac{M}{B_m}$, ..., $\frac{N}{2}\frac{M}{B_m}$ for $B_m \leq M$.

For *n-port* communication the lower bound for the data transfer time is $\frac{(N-1)M}{n}t_c$. Pipelining the communications in the *SBT exchange algorithm* reduces the data transfer time only by a factor of 2. No further reduction is possible due to the fact that $\frac{1}{2}N$ tree edges are mapped onto some cube edges. However, all-to-all broadcasting in a time proportional to the lower bound is possible by using N nRSBT graphs, or N SBnT graphs for the routing [17].

4.2.3 Spanning Balanced n-Trees and n Rotated Spanning Binomial Trees

The nRSBT and SBnT routings can also be used. The time for data transmission in the case of *one-port* communication is the same as for the SBT-b$(1, N)$ routing, but $n-1$ additional start-ups are required for the optimum case. However, the required buffer space is lower. For buffer spaces lower than the optimum for nRSBT-b$(1, N)$ and SBnT-b$(1, N)$ the communication complexities are comparable.

The nRSBT and SBnT routings offer a potential for lower bound routing in the case of *n-port* communication. $T_{min}^{nRSBT-b}(n, N) = T_{min}^{SBnT-b}(n, N) = \frac{(N-1)M}{n} t_c + n\tau \leq 2T_{lb}(n, N)$. The main difference between nRSBT-b(n, N) routing and SBnT-b(n, N) routing is that in the SBnT-b(n, N) routing the node set is divided into n parts, whereas in the nRSBT-b(n, N) the data set is divided into n parts.

The SBnT-b(n, N) and nRSBT-b(n, N) routing algorithms for *n-port* communication in pseudo codes are as follows:

```
/* The SBnT Algorithm for n-port all-to-all broadcasting: */

SUBROUTINE SBnT (init_data, length, final_data)
final_data [pid] = init_data
msg = pid || init_data
M = (length of pid) + length
DO i = 0, n-1
      outbuf [i] = msg
ENDDO
DO step = 1, n
      CALL SEND (port [i], outbuf [i]), for i = 0, 1, ..., n-1 concurrently.
      CALL RECV (port [i], inbuf [i]), for i = 0, 1, ..., n-1 concurrently.
      FOR i = 0 to n-1 DO
            FOR each msg of length M in inbuf [i] DO
                  /* Let msg = src || curr_data. */
                  {c [1], c [2], ..., c [k]} = children (pid, src)
                  /* At the last step, children will be empty. */
                  /* Let p [j] be the output port connecting to c [j]. */
                  DO j = 1, k
                        append msg of length M to outbuf [p [j]].
                  ENDDO
                  final_data [src] = curr_data
            ENDFOR
      ENDFOR
ENDDO

/* The nRSBT Algorithm for n-port all-to-all broadcasting: */

SUBROUTINE nRSBT (init_data, length, final_data)
EQUIVALENCE (A, final_data)
Split init_data into n parts.
curr_length = length / n          /* initial length of SEND and RECV */
DO i = 1, length
      A [(pid * curr_length) + (i mod curr_length), i / curr_length] = init_data [i]
ENDDO
```

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

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

```
send_ptr = pid * curr_length + 1          /* initial index of row in A for send */
DO i = 0, n-1
    CALL SEND (port [(i + k) mod n], A [send_ptr, k], curr_length),
        for k = 0, 1, ..., n-1 concurrently.
    IF ($i^{th}$ bit of pid .EQ. 0) THEN
        recv_ptr = send_ptr + curr_length
    ELSE
        recv_ptr = send_ptr - curr_length
        send_ptr = recv_ptr
    ENDIF
    CALL RECV (port [(i + k) mod n], A [recv_ptr, k], curr_length),
        for k = 0, 1, ..., n-1 concurrently.
    curr_length = curr_length * 2
ENDDO
```

4.2.4 Summary and Comparison

The times for *all-to-all* broadcasting using *one-port* and *n-port* communications are summarized in Table 2.

For *one-port* communication the SBT-b$(1, N)$ 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$(1, N)$, SBnT-b$(1, N)$, and nRSBT-b$(1, N)$ 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) and SBnT-b(n, N) routings are optimal within a factor of 2 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) routing. The SBT-b(n, N) routing is inferior.

4.3 One-to-All Personalized Communication

In broadcasting the data set M is replicated $N-1$ times, such that every node gets a copy of the same data set. 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. In *one-to-all broadcasting* the data volume across an edge is the same for all edges of the routing graph, but in personalized communication the data volume decreases with increased distance from the source.

One-to-all personalized communication is the same as transposing a vector stored entirely in one node. The transpose of the vector is stored across all processors. *All-to-all personalized communication* is the operation performed in transposing a matrix stored by one-dimensional partitioning.

The lower bound for *one-port*, *one-to-all personalized communication* is $T_{lb} = \max((N-1)Mt_c, n\tau)$. The SBT-p(1,1) is optimal within a factor of 2 for sufficiently large maximum packet size, $B_m \geq \frac{1}{2}NM$.

The spanning binomial tree is very unbalanced. Half of the nodes are in one subtree, a quarter in another, etc. With *n-port* communication the transmission time can be reduced by at most a factor of 2, since $\frac{1}{2}NM$ data elements have to be passed over the same communication channel. The SBT-p$(n,1)$ 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$(n,1)$ and SBnT-p$(n,1)$ routings is optimal within a factor of 2 if $B_m \geq \sqrt{\frac{2}{\pi}\frac{NM}{n^{3/2}}}$.

4.4 All-to-All Personalized Communication

All-to-all personalized communication can be performed by N SBTs. This algorithm 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$(1,N)$ routing is optimal within a factor of 2.

With *n-port* communication the lower bound for the transmission time is reduced by a factor of n. Hence, $T_{lb} = \max(\frac{NM}{2}t_c, n\tau)$. The SBT-p$(n,N)$ routing cannot attain the lower bound [17]. But, the nRSBT-p(n,N) and the SBnT-p(n,N) 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) [17].

4.5 Summary and Comparison

Tables 3 and 4 summarize the communication complexities for *personalized communications*.

On the Intel iPSC the communication start-up time is significant, so it is desirable to reduce the number of start-ups by sending long messages. In the SBT-p$(*,N)$ algorithm blocks to be exchanged between processors are not necessarily contiguous, and vary throughout the algorithm. Hence, minimizing the communication time requires internal data movement. However, the copy time is also significant on the iPSC. We have not included the time for data movement internal to a node in the expressions above for reasons of clarity. There exists a block size

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

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

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

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

$B_{copy} < B_m$ above which it is better to minimize copy time than start-up time. For details see [10].

Ignoring copy time the SBT-p($1, N$) algorithm is of the same order as the lower bound performance for *one-port* communication: $T = (\frac{NM}{2} t_c + \tau)n$ if $B \geq \frac{1}{2}NM$. For *n-port* communication SBnT(n, N) and nRSBT(n, N) routings are optimum within a factor of 2, if $B_m \geq \frac{(N-1)M}{n}$ and $B_m \geq \frac{NM}{2n}$, respectively.

5 Dense Matrix Multiplication

For multiprocessors configured as one-, two- or multidimensional arrays both one- and two-dimensional partitionings of matrices are common. The partitioning can be done either by rows or columns, or both, or by diagonals. One-dimensional partitionings are natural for linear arrays and two-dimensional partitionings for meshes. In addition, the partitioning can be made either cyclically, or consecutively [15]. In the two-dimensional *cyclic* storage, 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. In the *consecutive* partitioning matrix elements with the same high order bits are allocated to the same processor, i.e., the high order bits are used to assign an element to a processor, if P (or Q) is a power of 2. In the *cyclic* partitioning, elements in the same processor have the same low order bits.

A Boolean cube can simulate both one- and two-dimensional arrays without any communications overhead, and has additional communication capabilities. For the multiplication of a $P \times Q$ matrix by a $Q \times R$ matrix on N processors, one- and two-dimensional partitionings yield a linear speed-up of the arithmetic operations if at least P, or R, is a multiple of N. For $P, R < N \leq PR$, the two-dimensional partitioning yields a linear speed-up of the arithmetic operations. However, the communication complexity for the different algorithms and partitionings is not the same.

5.1 One-Dimensional Partitioning

In a one-dimensional partitioning with columns or rows divided evenly among the processors, cyclically or consecutively, a linear array algorithm is an obvious choice. Assume that the computation is $A \leftarrow C \times D + E$ and that all matrices are stored in the same manner. With the matrices partitioned columnwise *all-to-all broadcasting* can be performed on C. A is computed *in-place*, i.e., the products forming an element of A are accumulated in a fixed location (the same as the location of the corresponding element of D). Alternatively, C can be transposed and either followed by an *all-to-all broadcasting* for an *in-place* algorithm, or an *all-to-all broadcasting* performed on D, which yields A^T. A transpose of A^T yields A. A fourth alternative is to transpose D, which implies a computation of inner products throughout space, and the elements of A computed by a reduction over the number of processors for each element. The reduction is an *all-to-all reduction* on A. Several variations for each of the basic algorithms are also considered. The basic algorithms are:

- **Algorithm 1.** Compute A *in-place* by *all-to-all broadcasting* of C. Processor $k = PID(j)$ computes $CD(*, \lfloor \frac{j}{\lceil \frac{R}{N} \rceil} \rfloor)$ for all j mapped to k.

- **Algorithm 2.** Compute A by a transpose of C and an *all-to-all broadcasting* of C^T. Processor $k = PID(j)$ computes $CD(*, \lfloor \frac{j}{\lceil \frac{R}{N} \rceil} \rfloor)$ for all j mapped to k.

- **Algorithm 3.** Compute A by a transpose of C, *all-to-all broadcasting* of D, and transpose of A^T. Processor $k = PID(j)$ computes $C(\lfloor \frac{j}{\lceil \frac{P}{N} \rceil} \rfloor, *)D$.

- **Algorithm 4.** Compute A *in-space* by a transpose of D, and *all-to-all reduction* of partial inner products of A.

The second alternative is clearly inferior to the first, so we only consider the other three in the following. However, for the two-dimensional partitioning, the corresponding one of algorithm 2 may perform better than any other corresponding algorithm for certain aspect ratios.

For row partitioning the roles of C and D are interchanged. Figure 6 characterizes the basic algorithms. The two subscripts in sequence are used to denote the ordinal numbers of block rows and block columns among the N partitioned blocks rows (or columns). The superscript denotes the ordinal number of the partial inner product result. The number in the square brackets (eg. [R] in algorithm 1) is the minimum maximum number of processors to minimize the arithmetic time for each algorithm.

The total number of arithmetic operations is the same in all cases. We only consider the classical algorithm with $PR(2Q-1)$ arithmetic operations. The parallel arithmetic complexity is $2PQ\lceil \frac{R}{N} \rceil$ for algorithm 1, $2QR\lceil \frac{P}{N} \rceil$ for algorithm 3, and $PR(2\lceil \frac{Q}{N} \rceil - 1) + P(\lceil \frac{R}{N} \rceil + \sum_{i=1}^{n} \lceil \frac{R}{2^i} \rceil)$ for algorithm 4. The second term of the last expression is bounded from below by $P(n+1)$ and from above by $P\lceil \frac{R}{N} \rceil N$. If all dimensions are multiples of N there is no difference in the arithmetic complexity, but if for instance $P = c_1 N$, $Q = c_2 N$ and $R = 1$, then the arithmetic complexities are $2c_1c_2N^2$, $2c_1c_2N$, and $2c_1c_2N + c_1nN$, respectively. Distributing D in space (algorithms 3 and 4) is clearly more effective than distributing C. Similarly, if $P = 1$ and Q and R are multiples of N, then distributing C is preferable with respect to the parallel arithmetic complexity (algorithms 1 and 4).

Column Partitioning:

$$\begin{aligned}
A1: &\quad C_{*k}, D_{*k} \xrightarrow{\text{brd. C, } \leftrightarrow} C_{**}, D_{*k} \xrightarrow{\text{mpy., [R]}} A_{*k} \\
A3: &\quad C_{*k}, D_{*k} \xrightarrow{\text{txp. C, } \nearrow} C_{k*}, D_{*k} \xrightarrow{\text{brd. D, } \leftrightarrow} C_{k*}, D_{**} \xrightarrow{\text{mpy., [P]}} A_{k*} \xrightarrow{\text{txp. A, } \nearrow} A_{*k} \\
A4: &\quad C_{*k}, D_{*k} \xrightarrow{\text{txp. D, } \nearrow} C_{*k}, D_{k*} \xrightarrow{\text{mpy., [Q]}} A^{k}_{**} \xrightarrow{\text{red. A, } \leftrightarrow} A_{*k}
\end{aligned}$$

Row Partitioning:

$$\begin{aligned}
A1: &\quad C_{k*}, D_{k*} \xrightarrow{\text{brd. D, } \updownarrow} C_{k*}, D_{**} \xrightarrow{\text{mpy., [P]}} A_{k*} \\
A3: &\quad C_{k*}, D_{k*} \xrightarrow{\text{txp. D, } \nearrow} C_{k*}, D_{*k} \xrightarrow{\text{brd. C, } \updownarrow} C_{**}, D_{*k} \xrightarrow{\text{mpy., [R]}} A_{*k} \xrightarrow{\text{txp. A, } \nearrow} A_{k*} \\
A4: &\quad C_{k*}, D_{k*} \xrightarrow{\text{txp. C, } \nearrow} C_{*k}, D_{k*} \xrightarrow{\text{mpy., [Q]}} A^{k}_{**} \xrightarrow{\text{red. A, } \updownarrow} A_{*k}
\end{aligned}$$

Figure 6: Notation summary of algorithms for one-dimensional partitioning.

P_0	P_1	P_2	P_3	P_4	P_5	P_6	P_7
0	1	▨2	3	4	5	6	7
7	0	1	▨2	3	4	5	6
6	7	0	1	▨2	3	4	5
5	6	7	0	1	▨2	3	4
4	5	6	7	0	1	▨2	3
3	4	5	6	7	0	1	▨2
▨2	3	4	5	6	7	0	1
1	▨2	3	4	5	6	7	0

Figure 7: Computing $A \leftarrow C \times D + E$ by rotation of C.

The matrix transpose operation implies *all-to-all personalized communication*. Hence, the three types of algorithms considered consist of different combinations of *all-to-all broadcasting/reduction* and *all-to-all personalized communication* on matrices of different shapes, in addition to the arithmetic operations. The communication complexity of the different operations are given in the previous sections. Tables 5, 6, and 7 give the communication complexity for the different multiplication algorithms (assuming column partitioning).

For *one-port* communication the CRA$(1, N)$, GCEA$(1, N)$, and SBT$(1, N)$ algorithms are of interest. The first two are included only because they yield the same complexity as the SBT$(1, N)$ algorithm if $B_m \leq M$.

With the CRA$(1, N)$ algorithm and either cyclic or consecutive storage, the number of communication steps is $N - 1$ and the number of arithmetic steps is N. In the *in-place* algorithm C is rotated $N - 1$ times to the right (or left), and in the *in-space* algorithm D and the partial accumulation of A are rotated left (or right). In the *in-place* algorithm products are formed on the shaded parts of D during step i, $i = \{0, 1, \ldots, N-1\}$, Figure 7. Sample codes for column partitioning, consecutive storage, and an *in-place* algorithm based on the CRA$(1, N)$ and GCEA$(1, N)$ *all-to-all broadcasting* algorithms are given below.

/* Matrix multiplication using the Cyclic Rotation Algorithm: */

```
/* Gray code encoding. */

gid = IG (pid)
/* Let C and D and A be the $gid^{th}$ (or $pid^{th}$ for binary code encoding) */
/* block columns of matrices C, D and A respectively. */
CALL INIT_CRA
A = C * $gid^{th}$ block rows of D          /* block matrix multiplication */
/* For binary code encoding: A = C * $pid^{th}$ block rows of D. */
DO i = 1, N-1
        CALL CRA (C, $P\lceil\frac{Q}{N}\rceil$)
        A = A + C * $((gid + i) \bmod N)^{th}$ block rows of D
        /* For binary code encoding:
                A = A + C * $G((pid + i) \bmod N)^{th}$ block rows of D. */
ENDDO

/* Matrix multiplication using the Gray Code Exchange Algorithm: */
/* Binary code encoding. */

gid = IG (pid)
/* Let C and D and A be the $pid^{th}$ block columns. */
/* of matrices C, D and A respectively. */
A = C * $pid^{th}$ block rows of D
/* For Gray code encoding: A = C * $(gid)^{th}$ block rows of D. */
DO i = 1, N-1
        CALL GCEA (i, C, $P\lceil\frac{Q}{N}\rceil$)
        A = A + C * $(pid \oplus G(i))^{th}$ block rows of D
        /* For Gray code encoding: A = A + C * $(gid \oplus i)^{th}$ block rows of D. */
ENDDO
```

The SBT$(1, N)$ algorithm takes advantage of the topology of the Boolean cube, in that the spanning tree is of minimum height. The SBT-b$(1, N)$ and SBT-p$(1, N)$ algorithms are optimal for *one-port* communication and unlimited buffer size, i.e., $B_m \geq \frac{1}{2}PQ$ for *all-to-all broadcasting* or *all-to-all personalized communication* of a $P \times Q$ matrix partitioned evenly by rows or columns. Figure 8 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 the *in-place* algorithm.

Figure 9 (left) shows the measured times of the GCEA and the SBT algorithms on the iPSC with fixed matrices, $P = Q = R = 32$, and varying cube dimensions. The communication times of the SBT algorithm 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 9.

While the number of start-ups of the SBT algorithm grows linearly with the number of cube dimensions, the maximum required temporary storage and the optimum buffer size grow exponentially, if the size of the partitioned matrices is fixed. If the maximum buffer size is

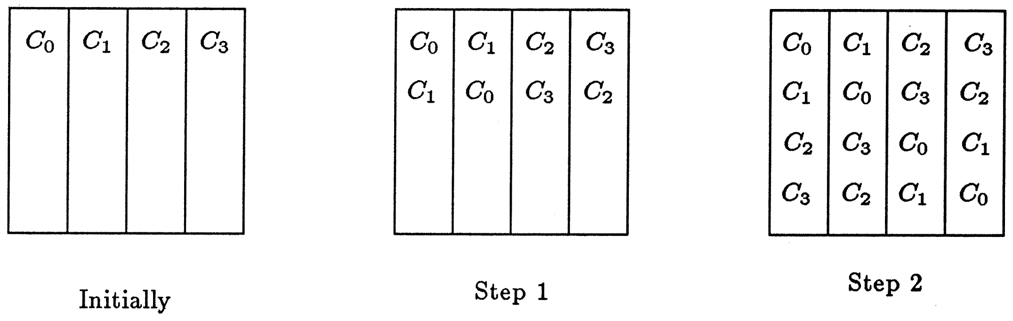


Figure 8: Computing $A \leftarrow C \times D + E$ by an *in-place* SBT-b$(1, N)$ algorithm applied to C.

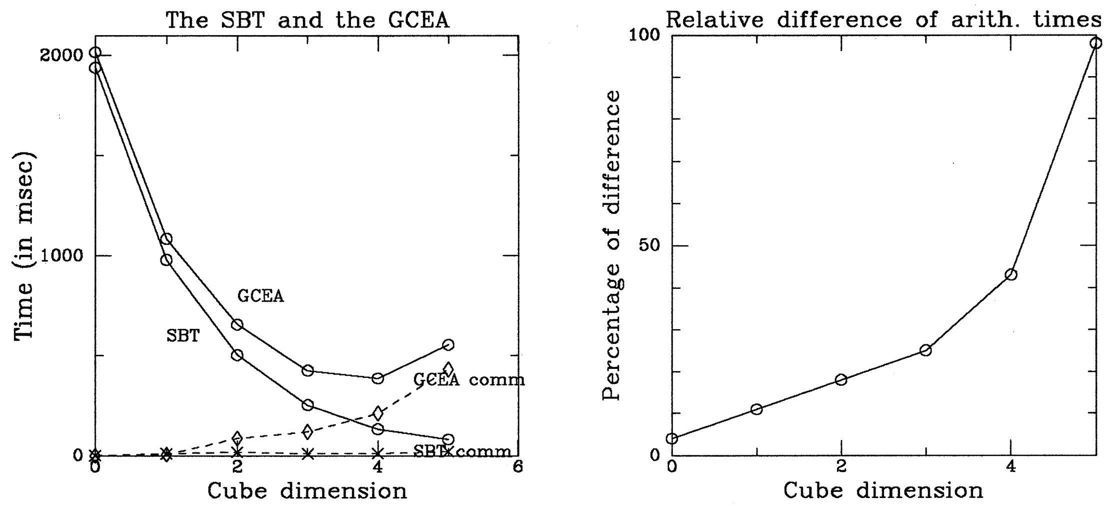
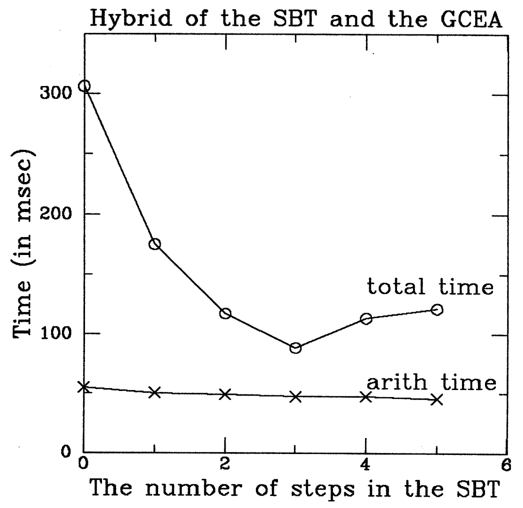


Figure 9: Measured times for matrix multiplication using the GCEA and the SBT algorithms on the iPSC. $P = Q = R = 32$. On the left, the solid lines are total times and the dashed lines are communication times. On the right, the relative difference of the arithmetic time is shown as a function of cube dimensions.



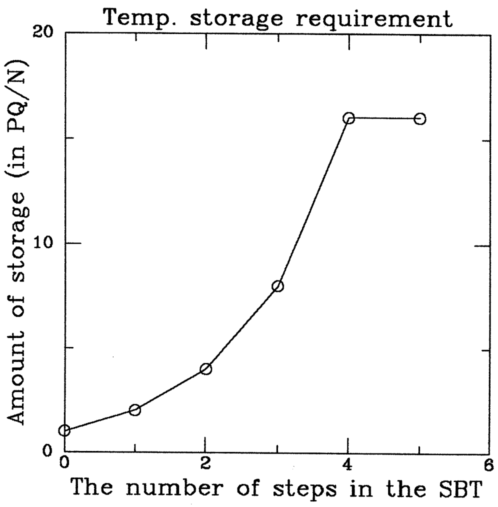


Figure 10: Measured times for matrix multiplication using the hybrid method of the GCEA and the SBT algorithms on a 5-dimensional cube of the iPSC. $P = Q = R = 32$.

less than half the matrix size, additional start-ups are needed. If the buffer size is less than the matrix partition residing in a processor, then the time complexities of the SBT and GCEA algorithms are the same. But, the SBT algorithm needs additional temporary storage. For the case of limited temporary storage a hybrid of the SBT and the GCEA (or the CRA) algorithms can be used. The hybrid method will perform k steps of the SBT algorithm during which data is cumulated, followed by $2^{n-k} - 1$ steps of the GCEA or the CRA. The maximum temporary storage and the optimum buffer size is $\frac{2^k PQ}{N}$. Figure 10 (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 different overheads involved. The temporary storages required with respect to the number of steps in the SBT are shown on the right. Note that the algorithm with $k = n$ is the same as that of $k = n - 1$.

The above algorithms are all *in-place* algorithms. The *in-space* algorithms use one instance of broadcasting and one instance of personalized communication. Algorithm 3 uses two instances of personalized communication. For the *in-space* algorithm and *one-port* communication we use the SBT-p$(1, N)$ algorithm for the transpose operation and the SBT-b$(1, N)$ algorithm for the gathering of inner products. The following is a sample code.

```
/* Matrix multiplication in-space by matrix transposition: */

CALL TRANSPOSE (D)
DO i = 1, P
      DO j = 1, R
            A (i, j) = C (i, 1) * D (j, 1)
      ENDDO
ENDDO
DO k = 2, $\lceil \frac{Q}{N} \rceil$
```

```
        DO i = 1, P
              DO j = 1, R
                    A (i, j) = A (i, j) + C (i, k) * D (j, k)
              ENDDO
        ENDDO
  ENDDO
  curr_A = A          /* A is a P × R matrix */
  DO i = 0, n-1
        IF (the $i^{th}$ bit of pid .EQ. 0) THEN
              outbuf = right half block columns of curr_A
              curr_A = left half block columns of curr_A
        ELSE
              outbuf = left half block columns of curr_A
              curr_A = right half block columns of curr_A
        ENDIF
        CALL SEND (port [i], outbuf, $P\lceil \frac{R}{2^{i+1}} \rceil$)
        CALL RECV (port [i], inbuf, $P\lceil \frac{R}{2^{i+1}} \rceil$)
        curr_A = curr_A + inbuf          /* Matrix addition */
  ENDDO
```

For the transposition of D each block column is divided into $\frac{Q}{N} \times \frac{R}{N}$ blocks, and each such block sent to a distinct processor. After this communication and the associated arithmetic operations there are N partial inner products, one per processor, for each element of A. The accumulation of the partial sums amounts to sending a $P \times \lceil \frac{R}{N} \rceil$ block from every processor to the processor storing the corresponding block column of A. The total temporary storage required per processor is $P \times R$.

With *n-port* communication the GCEA(n, N) algorithm can be employed instead of the GCEA$(1, N)$ 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) 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) and nRSBT(n, N) algorithms yield a 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.*

An *in-place* matrix multiplication algorithm for column partitioning and consecutive storage using *all-to-all broadcasting* based on SBnT-b(n, N) (or nRSBT-b(n, N)) routing can be expressed as follows:

```
/* Matrix multiplication using the SBnT (or nRSBT) Algorithm: */

CALL SBnT (C, $P\lceil \frac{Q}{N} \rceil$, C2)
/* or CALL nRSBT (C, $P\lceil \frac{Q}{N} \rceil$, C2) */
A = C2 [0] * $0^{th}$ block rows of D
DO i = 1, N-1
      A = A + C2 [i] * $i^{th}$ block rows of D
ENDDO
```

For an *in-space* algorithm and *n-port* communication we choose the SBnT-p(n, N) and SBnT-b(n, N) (or nRSBT-p(n, N) and nRSBT-b(n, N)) algorithms.

Communication model	Algorithm	Communication operations
one-port	$A1^l(1)$	CRA$(1,N)[C]$ or GCEA$(1,N)[C]$
	$A1(1)$	SBT-b$(1,N)[C]$
	$A3(1)$	SBT-p$(1,N)[C]$+SBT-b$(1,N)[D]$+SBT-p$(1,N)[A]$
	$A4(1)$	SBT-p$(1,N)[D]$+SBT-b$(1,N)[A]$
n-port	$A1^l(n)$	GCEA$(n,N)[C]$
	$A1(n)$	SBnT-b$(n,N)[C]$ or nRSBT-b$(n,N)[C]$
	$A3(n)$	SBnT-p$(n,N)[C]$+SBnT-b$(n,N)[D]$+SBnT-p$(n,N)[A]$
	$A4(n)$	SBnT-p$(n,N)[D]$+SBnT-b$(n,N)[A]$

Table 5: Algorithm classification.

Algorithm	Element transfers	**start-ups**
$A1^l(1)$	$(N-1)P\lceil\frac{Q}{N}\rceil$	$\lceil\frac{PQ}{NB_m}\rceil(N-1)$
$A1(1)$	$(N-1)P\lceil\frac{Q}{N}\rceil$	$\sum_{i=0}^{n-1}\lceil\frac{2^iPQ}{NB_m}\rceil$
$A3(1)$	$((N-1)Q+\frac{n}{2}P)\lceil\frac{R}{N}\rceil+\frac{n}{2}P\lceil\frac{Q}{N}\rceil$	$(\lceil\frac{PQ}{2NB_m}\rceil+\lceil\frac{PR}{2NB_m}\rceil)n+\sum_{i=0}^{n-1}\lceil\frac{2^iPR}{NB_m}\rceil$
$A4(1)$	$((N-1)P+\frac{n}{2}Q)\lceil\frac{R}{N}\rceil$	$\lceil\frac{RQ}{2NB_m}\rceil n+\sum_{i=0}^{n-1}\lceil\frac{2^iPR}{NB_m}\rceil$
$A1^l(n)$	$\frac{1}{n}(N-1)P\lceil\frac{Q}{N}\rceil$	$\lceil\frac{PQ}{nNB_m}\rceil(N-1)$
$A1(n)$	$\frac{1}{n}(N-1)P\lceil\frac{Q}{N}\rceil$	$\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{PQ}{nNB_m}\rceil$
$A3(n)$	$\frac{1}{n}\{((N-1)Q+\frac{n}{2}P)\lceil\frac{R}{N}\rceil+\frac{n}{2}P\lceil\frac{Q}{N}\rceil\}$	$\approx(\lceil\frac{PQ}{2nNB_m}\rceil+\lceil\frac{PR}{2nNB_m}\rceil)n+\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{QR}{nNB_m}\rceil$
$A4(n)$	$\frac{1}{n}((N-1)P+\frac{n}{2}Q)\lceil\frac{R}{N}\rceil$	$\approx\lceil\frac{QR}{2nNB_m}\rceil n+\sum_{i=1}^{n}\lceil\binom{n}{i}\frac{PR}{nNB_m}\rceil$

Table 6: The communication complexity of matrix multiplication using one-dimensional column partitioning.

5.1.1 Summary and Comparison

We summarize the communication complexities of the multiplication algorithms in Tables 5, 6, and 7. Table 5 defines the algorithms in terms of communication operations. All the algorithms use the Boolean cube topology unless superscripted by l, which denotes a linear array algorithm. The argument denotes the number of ports for concurrent communication. Note that in $A3(n)$, the SBnT-p$(n,N)[C]$ and the SBnT-b$(n,N)[D]$ can in fact be combined into one complicated 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 row partitioning a corresponding set of algorithms can be devised. The roles of C and D are interchanged. If it is possible to choose the partitioning form, column partitioning should be used for $P \leq R$ for a given type of algorithm.

For the column partitioning scheme and *one-port* communication, the $A4$ algorithm yields lower complexity than the $A1$ algorithm if $(N-1)P\lceil\frac{Q}{N}\rceil > ((N-1)P+\frac{n}{2}Q)\lceil\frac{R}{N}\rceil$, or with $\alpha=\lceil\frac{R}{N}\rceil$ and $\beta=\lceil\frac{Q}{N}\rceil$, if $P \geq \frac{n}{2}\frac{\alpha\beta}{\beta-\alpha}$ approximately. Hence, if D is a vector, then in many instances $Q \gg R$, and the above condition yields $P > \frac{n}{2}\alpha$.

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 11 to 14. All these Figures consist of four plots. Plot (a), the upper left plot, shows the two boundary planes which

Algorithm	B_{opt}	*min* start-ups	Arithmetic
$A1^l(1)$	$P\lceil \frac{Q}{N} \rceil$	$N-1$	$2PQ\lceil \frac{R}{N} \rceil$
$A1(1)$	$\frac{PQ}{2}$	n	$2PQ\lceil \frac{R}{N} \rceil$
$A3(1)$	$\frac{PQ}{2N}, \frac{1}{2}QR, \frac{PR}{2N}$	$3n$	$2QR\lceil \frac{P}{N} \rceil$
$A4(1)$	$\frac{1}{2}Q\lceil \frac{R}{N} \rceil, \frac{1}{2}PR$	$2n$	$PR(2\lceil \frac{Q}{N} \rceil - 1) + P(\lceil \frac{R}{N} \rceil + \sum_{i=1}^{n} \lceil \frac{R}{2^i} \rceil)$
$A1^l(n)$	$\frac{1}{n}P\lceil \frac{Q}{N} \rceil$	$N-1$	$2PQ\lceil \frac{R}{N} \rceil$
$A1(n)$	$\sqrt{\frac{2}{\pi}\frac{PQ}{n^{3/2}}}$	n	$2PQ\lceil \frac{R}{N} \rceil$
$A3(n)$	$\frac{P}{2n}\lceil \frac{Q}{N} \rceil, \sqrt{\frac{2}{\pi}\frac{QR}{n^{3/2}}}, \frac{P}{2n}\lceil \frac{R}{N} \rceil$	$3n$	$2QR\lceil \frac{P}{N} \rceil$
$A4(n)$	$\frac{Q}{2n}\lceil \frac{R}{N} \rceil, \sqrt{\frac{2}{\pi}\frac{PR}{n^{3/2}}}$	$2n$	$PR(2\lceil \frac{Q}{N} \rceil - 1) + P(\lceil \frac{R}{N} \rceil + \sum_{i=1}^{n} \lceil \frac{R}{2^i} \rceil)$

Table 7: The *optimum* communication complexity of matrix multiplication using one-dimensional column partitioning. For the amount of element transfers, see Table 6.

partition the $16 \times 16 \times 16$ parameter cube into 3 disjoint regions. The region for the $A1$ is below the lower plane (and above the $\frac{P}{N} = 0$ plane). The region for the $A4$ is between these two planes. The region for the $A3$ is above the upper plane (and below the $\frac{P}{N} = 16$ plane). Plot (b), the upper right plot, shows the lower plane. Plot (c), the lower left plot, shows the upper plane. Plot (d), the lower right plot, shows the contour of the upper plane. As the aspect ratio of $\frac{\tau}{t_c}$ increases as from Figure 11 to Figure 12, the regions of the $A3$ and the $A4$ are gradually taken over by the $A1$. As the number of processors increase, part of the $A1$ region is taken over by the $A3$ and the $A4$. For sufficiently large number of processors (such as $N = 1024$) the cube is partitioned into three symmetric regions of approximately equal size according to the three axes, Figure 13. Note that for $\frac{P}{N} = \frac{Q}{N} = \frac{R}{N}$, the communication complexity of the $A1$ is less than that of the $A4$, which in turn is less than the $A3$ as shown in Table 7, and Figures 11 to 13. The special case where $1 \leq P, Q, R \leq N$ is shown in Figure 14.

With *n-port* communication the SBnT(n, N) or nRSBT(n, N) 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 $B_m \leq \frac{PQ}{nN}$ for the algorithm $A1$. The SBnT(n, N)-based (or the nRSBT(n, N)-based) algorithm has the same communication complexity as the GCEA(n, N) algorithm if $B_m \leq \frac{PQ}{nN}$. The reduction in the number of start-ups for the SBnT(n, N) and nRSBT(n, N) algorithms is $\frac{N-1}{n}$ for $B_m \geq \binom{n}{\frac{n}{2}}\frac{PQ}{nN}$ (and larger temporary storage).

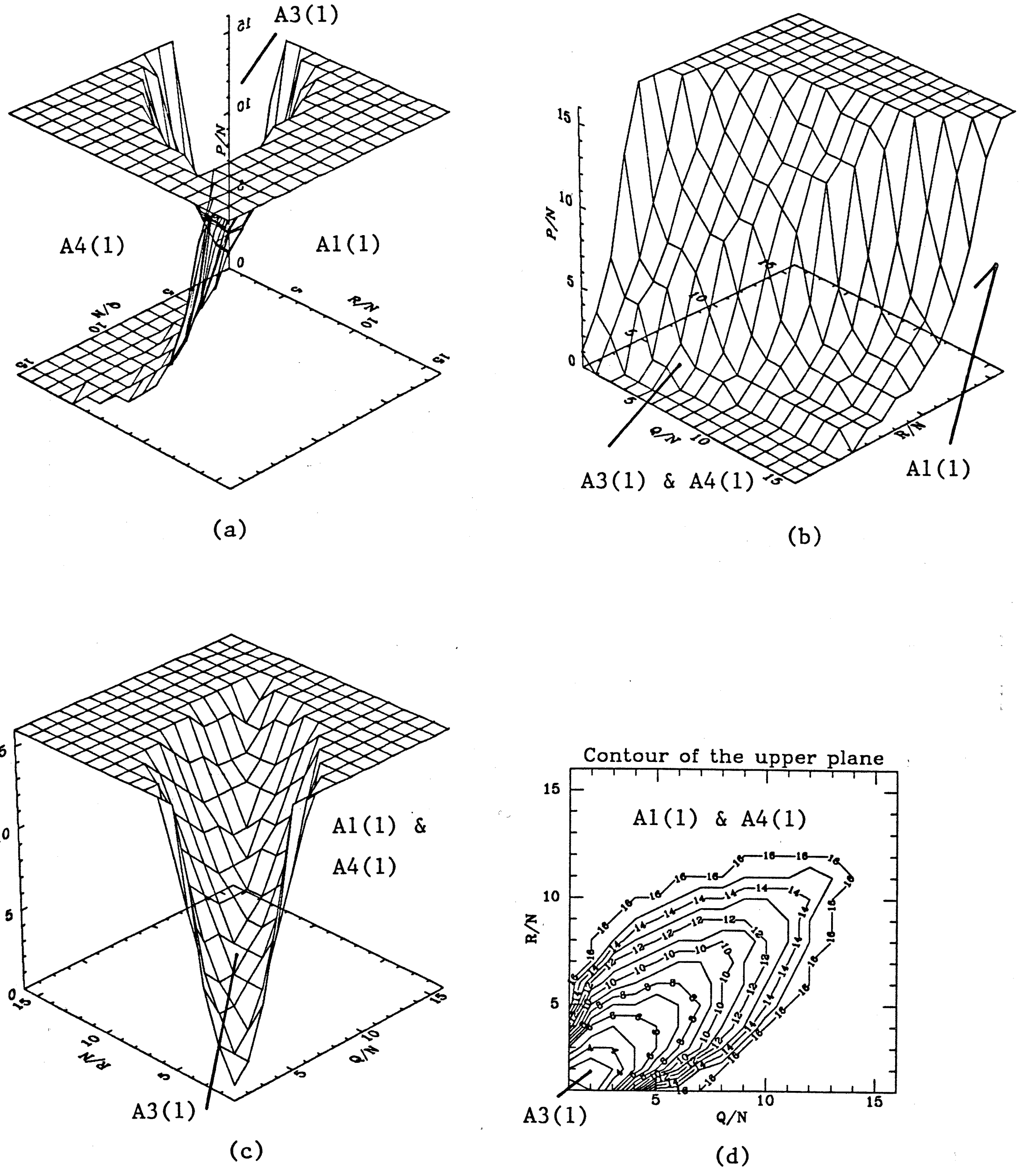
The criterion for choosing between the $A1$ or $A4$ algorithms is the same in the *n-port* and *one-port* cases. The plot which shows the region of the lowest complexity algorithm with $\frac{\tau}{t_c} = \gamma$ and *one-port* communication is the same as that of $\frac{\tau}{t_c} = \frac{\gamma}{n}$ and *n-port* communication.

If $Q < N$ and $R < N$ and the linear array is embedded in a Boolean cube, then the number of steps can be reduced to k, where $2^{k-1} < \max(Q, R) \leq 2^k$.

All the algorithms described work for both binary code and Gray code encodings due to the fact that summing the inner product can be done in an arbitrary order. While the local data are accessed differently, the communication patterns are the same.

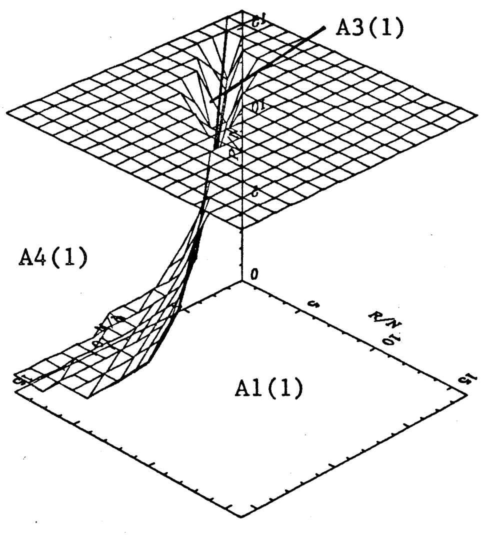
5.2 Two-Dimensional Partitioning

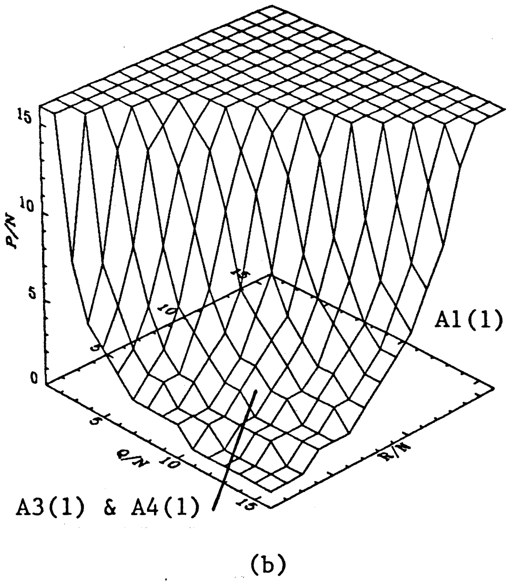
For the two-dimensional partitioning we assume a binary-reflected Gray code embedding of the $N_1 \times N_2$ grid in the Boolean cube. However, it can be shown that with a binary code embedding

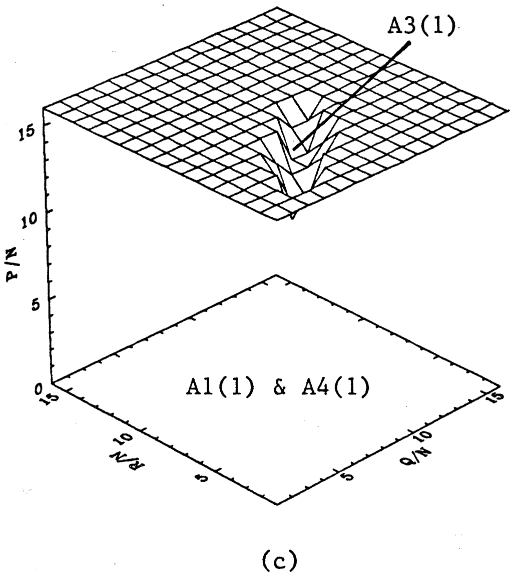


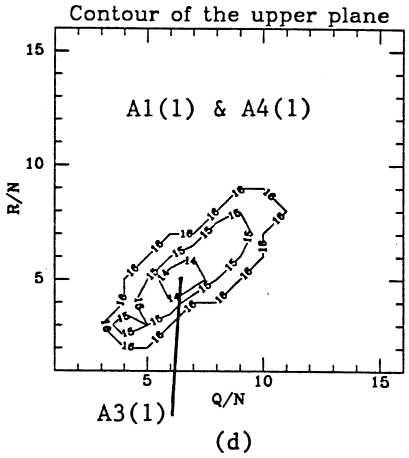
(a) The region for the $A1$ is below the lower plane (in the lower right part). The region for the $A4$ is between the two planes (in the lower left part). The region for the $A3$ is above the upper plane (in the upper central part). Note that these two boundary planes coalesce on the right side. (b) The lower plane. (c) The upper plane. Note that (a), (b) and (c) are shown from different views. (d) The contour of the upper plane.

Figure 11: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = t_c$, $B_m = \infty$, column partitioning, *one-port* communication.



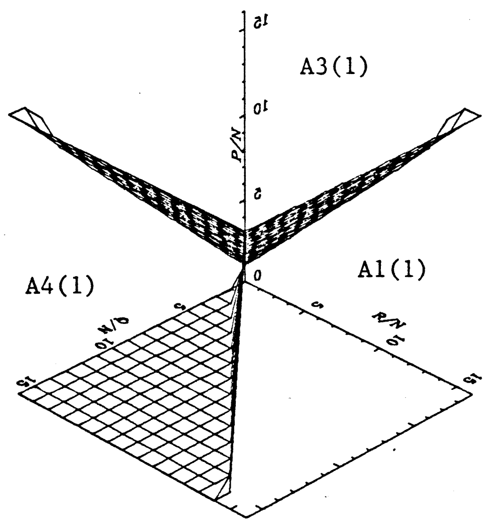




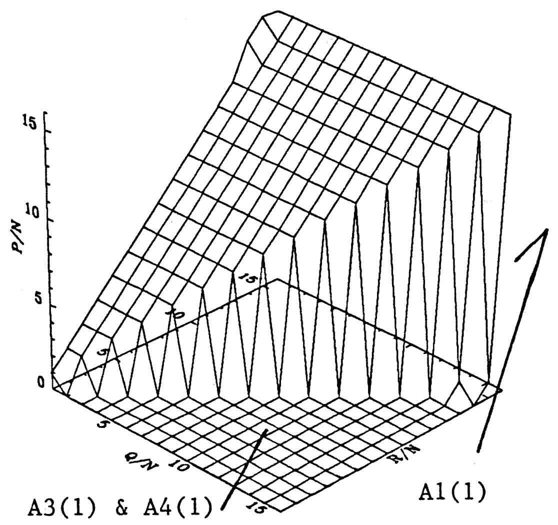


(a) The region for the $A1$ is below the lower plane (in the lower right part). The region for the $A4$ is between the two planes (in the lower left part). The region for the $A3$ is above the upper plane (in the upper central part). (b) The lower plane. (c) The upper plane. (d) The contour of the upper plane.

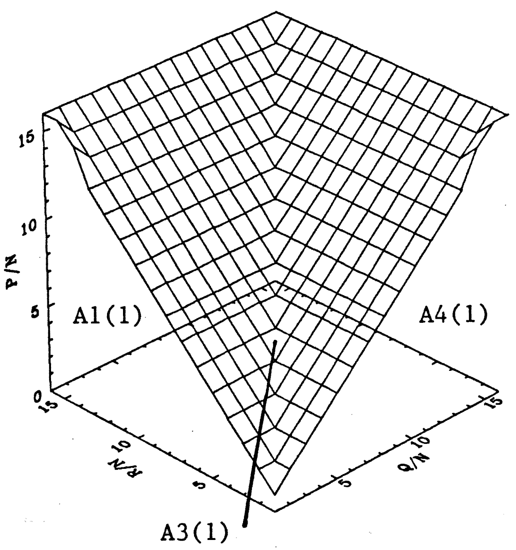
Figure 12: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = 1000t_c$, $B_m = \infty$, column partitioning, *one-port* communication.



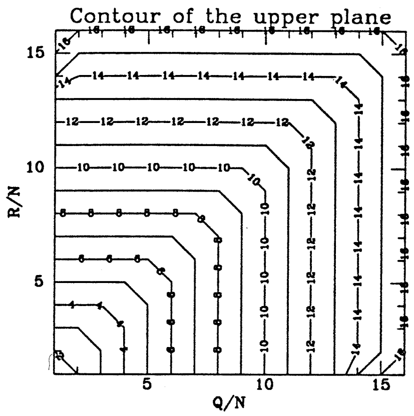
(a)



(b)



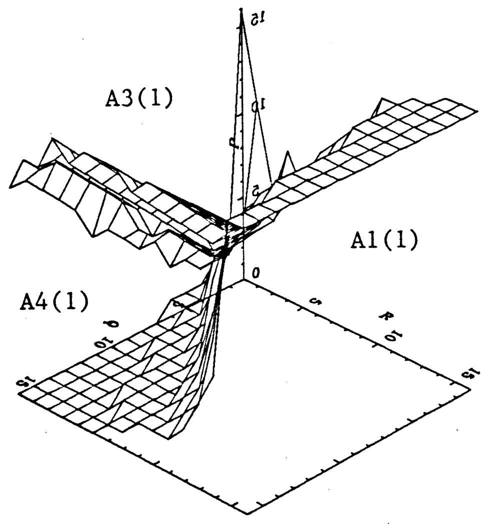
(c)



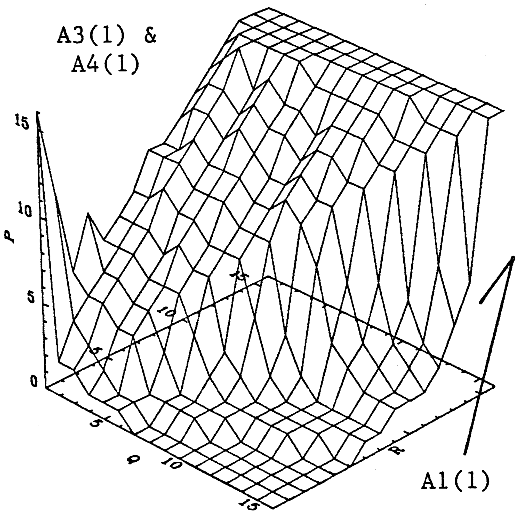
(d)

(a) The region for the *A*1 is below the lower plane (in the lower right part). The region for the *A*4 is between the two planes (in the lower left part). The region for the *A*3 is above the upper plane (in the upper central part). (b) The lower plane. (c) The upper plane. (d) The contour of the upper plane.

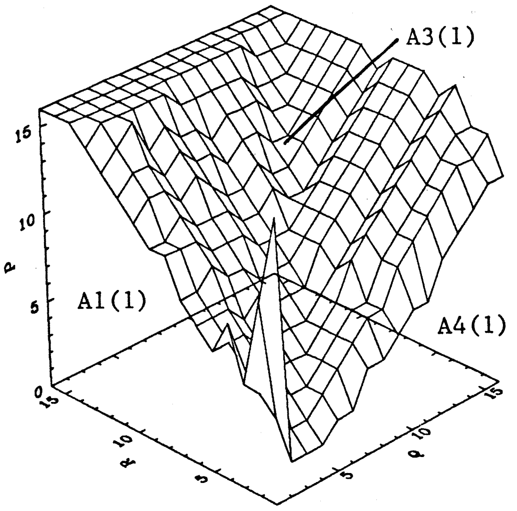
Figure 13: Lowest complexity algorithm as a function of matrix shape; $N = 1024$, $\tau = t_c$, $B_m = \infty$, column partitioning, *one-port* communication.



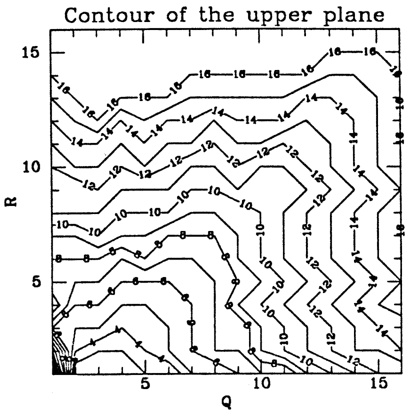
(a)



(b)



(c)



(d)

(a) The region for the $A1$ is below the lower plane (in the lower right part). The region for the $A4$ is between the two planes (in the lower left part). The region for the $A3$ is above the upper plane (in the upper central part). (b) The lower plane. (c) The upper plane. (d) The contour of the upper plane.

Figure 14: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = t_c$, $B_m = \infty$, column partitioning, *one-port* communication. Note that $1 \leq P,\ Q,\ R \leq N$.

of the $N_1 \times N_2$ grid in the Boolean cube, the communication pattern remains the same. The algorithms described for the one-dimensional case have analogues in the two-dimensional case. The *in-place* algorithm that in the one-dimensional case implies broadcasting of C in the row direction for column partitioning, in the two-dimensional case also implies broadcasting of D in the column direction. The two broadcasting operations need to be synchronized in order to conserve storage. We will describe the initialization and synchronization requirements below. The algorithms corresponding to the four one-dimensional algorithms ($A4$ has two variations) are

- **Algorithm 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 *all-to-all broadcasting* within rows and columns.

- **Algorithm 2.** Transpose C, perform an *all-to-all broadcast* along processor rows for the elements of C^T in that processor row, and accumulate inner products for A through *all-to-all 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 3.** Transpose C, perform *all-to-all broadcasting* of the elements of D within processor rows, accumulate inner products in the column direction. The *all-to-all 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 4.** Transpose D, perform an *all-to-all broadcasting* of the elements of D^T within processor columns, accumulate the partial inner products for elements of A by *all-to-all 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 5.** Transpose D, perform an *all-to-all broadcasting* of the elements of C within processor columns, accumulate inner products for elements of A by *all-to-all 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 15 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.

$$\begin{aligned}
A1:&\quad C_{kl}, D_{kl} \xRightarrow{\text{brd. C, } \leftrightarrow} C_{k*}, D_{kl} \xRightarrow{\text{brd. D, } \updownarrow} C_{k*}, D_{*l} \xRightarrow{\text{mpy., [PR]}} A_{kl}\\
A2:&\quad C_{kl}, D_{kl} \xRightarrow{\text{txp. C, } \nearrow} C_{lk}, D_{kl} \xRightarrow{\text{brd. C, } \leftrightarrow} C_{*k}, D_{kl} \xRightarrow{\text{mpy., [QR]}} A^{k}_{*l} \xRightarrow{\text{red. A, } \updownarrow} A_{kl}\\
A3:&\quad C_{kl}, D_{kl} \xRightarrow{\text{txp. C, } \nearrow} C_{lk}, D_{kl} \xRightarrow{\text{brd. D, } \leftrightarrow} C_{lk}, D_{k*} \xRightarrow{\text{mpy., [PQ]}} A^{k}_{l*} \xRightarrow{\text{red. A, } \updownarrow} A_{lk} \xRightarrow{\text{txp. A, } \nearrow} A_{kl}\\
A4:&\quad C_{kl}, D_{kl} \xRightarrow{\text{txp. D, } \nearrow} C_{kl}, D_{lk} \xRightarrow{\text{brd. D, } \updownarrow} C_{kl}, D_{l*} \xRightarrow{\text{mpy., [PQ]}} A^{l}_{k*} \xRightarrow{\text{red. A, } \leftrightarrow} A_{kl}\\
A5:&\quad C_{kl}, D_{kl} \xRightarrow{\text{txp. D, } \nearrow} C_{kl}, D_{lk} \xRightarrow{\text{brd. C, } \updownarrow} C_{*l}, D_{lk} \xRightarrow{\text{mpy., [QR]}} A^{l}_{*k} \xRightarrow{\text{red. A, } \leftrightarrow} A_{lk} \xRightarrow{\text{txp. A, } \nearrow} A_{kl}
\end{aligned}$$

Figure 15: Notation summary of the algorithms for two-dimensional partitioning.

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 $A1$) is the minimum maximum number of processors to minimize the arithmetic time for each algorithm. Algorithm $A2$ has a matrix transpose in addition to the communication of C as in algorithm $A1$. But, unlike in the one-dimensional case algorithm $A2$ may have a higher processor utilization than algorithm $A1$.

5.2.1 Algorithm 1

Algorithm 1 if carried out as a sequence of *one-to-all broadcasts* with a multiplication phase between each such operation constitutes an *outer-product* algorithm, but becomes an *inner-product* algorithm, if it is implemented by *all-to-all broadcasting* followed by the multiplication operations. If storage is to be conserved, then a linear array type algorithm, such as the CRA or GCEA algorithms, needs to be used. In order to avoid $O(N_1^2, N_2^2)$ start-ups, and correspondingly high data transfer times, it is of interest to pipeline the computations for successive outer products. This effectively leads to *all-to-all broadcasting* with constant storage. 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 [14].

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 proceeds as triangle shaped wave fronts emanating 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 sending the message to the opposite side of the boundary processors. $2\sqrt{N} + 1$ steps are sufficient to complete the algorithm.

Figure 16 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 are 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 [4,5] describes a matrix multiplication algorithm suitable for SIMD architectures

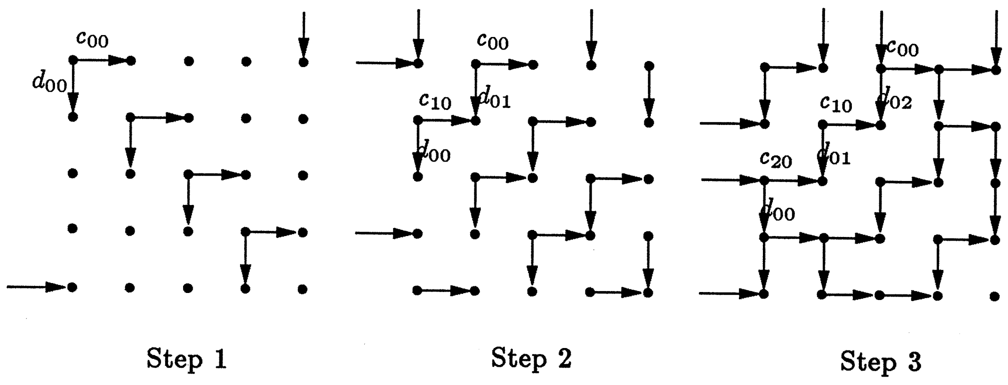
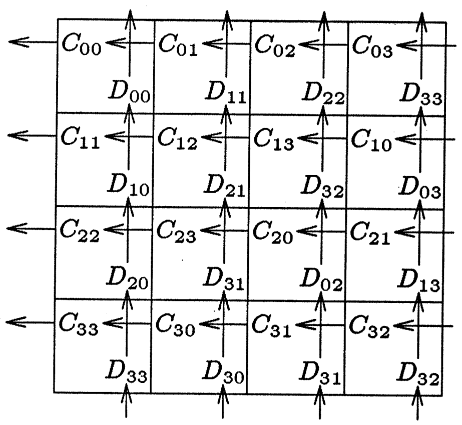


Figure 16: Computing $A \leftarrow C \times D + E$ by a pipelined outer product algorithm.



C_{00} C_{01} C_{02} C_{03}
D_{00} D_{11} D_{22} D_{33}
C_{11} C_{12} C_{13} C_{10}
D_{10} D_{21} D_{32} D_{03}
C_{22} C_{23} C_{20} C_{21}
D_{20} D_{31} D_{02} D_{13}
C_{33} C_{30} C_{31} C_{32}
D_{33} D_{30} D_{31} D_{32}

Figure 17: Matrix multiplication on a mesh according to [4,5].

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, N_2 being powers of 2) communication steps for *one-port communication*. During the alignment phase the matrix C is rotated left and the matrix D rotated up. The alignment of the matrices allows all processors to participate in each step of the multiplication phase. After the 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 17 shows the data structures after the alignment phase. The alignment defines a particular permutation and any suitable routing algorithm for the Boolean cube can be used. If the permutation is carried out as a sequence of shifts, and the CRA algorithm is used, then for *one-port* communication the row shifts use CRA$(1, N_2)$ and require $\lfloor \frac{N_2}{2} \rfloor$ steps, and the column shifts require $\lfloor \frac{N_1}{2} \rfloor$ steps by CRA$(1, N_1)$.

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, replace P, N_1 and N_2 by R, N_2 and N_1 respectively. 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.

/* Matrix multiplication according to Cannon's Algorithm: */

```
/* Gray code encoding. Assume $N_1 \geq N_2$ and $\frac{N_1}{N_2} = k$. */

rid = IG ($\lfloor \frac{pid}{N_2} \rfloor$)
cid = IG ($pid \bmod N_2$)
gid = rid || cid
/* Let C, D and A be the (rid, cid) block of the matrices C, D and A, respectively. */
/* Partition the local matrix C into k column blocks denoted C[i], i = {1, ..., k}. */
/* Similarly for the local matrices D and A. */
/* Alignment phase: */
DO i = 1, rid
    /* C is rotated left. */
    CALL CRA_row (C, $\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_2} \rceil$)
        /* CRA within the subcube of the least $n_2$ dimensions */
ENDDO
DO i = 1, cid
    /* D is rotated up. */
    CALL CRA_column (D, $\lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2} \rceil$)
ENDDO
/* Multiplication phase: */
DO i = 1, k
    A [i] = C [i] * D [i]
ENDDO
Do j = 1, $N_1 - 1$
    CALL CRA_row (C[1], $\lceil \frac{P}{N_1} \rceil \lceil \frac{Q}{N_1} \rceil$)
    Left rotate C[i] locally.
    CALL CRA_column (D, $\lceil \frac{Q}{N_1} \rceil \lceil \frac{R}{N_2} \rceil$)
    DO i = 1, k
        A [i] = A [i] + C [i] * D [i]
    ENDDO
ENDDO
```

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 [14].

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$(1, N)$ 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. For the multiplication phase there are $N_2 - 1$ communication steps for C in the x-direction, 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 [5] describes the above algorithm in detail for $P = Q = R = \sqrt{N}$. For $P = 2^{\alpha_1} N_1$, $Q = 2^{\alpha_2} N_1 = 2^{\alpha_3} N_2$ and $R = 2^{\alpha_4} N_2$ 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.

```
/* Matrix multiplication according to Dekel's Algorithm: */
/* Binary code encoding. One-port communication, $N_1 \geq N_2$ and $\frac{N_1}{N_2} = k$. */

rid = $\lfloor \frac{pid}{N_2} \rfloor$
cid = $pid \bmod N_2$
/* Alignment phase: */
SWAP C[$j$] with C[$j \oplus rid$] locally for $j \in \{1, 2, ..., k\}$.
DO i = 1, $n_2$
          IF (($i^{th}$ bit of rid) .EQ. 1) THEN
                    CALL SEND (port[i], C, $\frac{PQ}{N}$)
                    CALL RECV (port[i], C, $\frac{PQ}{N}$)
          ENDIF
ENDDO
DO i = 1, $n_1$
          IF (($i^{th}$ bit of cid) .EQ. 1) THEN
                    CALL SEND (port[$n_2 + i$], D, $\frac{QR}{N}$)
                    CALL RECV (port[$n_2 + i$], D, $\frac{QR}{N}$)
          ENDIF
ENDDO
/* Multiplication phase: */
DO i = 1, k
          A [i] = C [i] * D [i]
ENDDO
DO i = 1, $N_1 - 1$
          IF (i mod k .EQ. 0) THEN
                    CALL GCEA ($\frac{i}{k}$, C, $\frac{PQ}{N}$)
          ELSE
                    u = the position of the rightmost 0-bit of i-1
                    SWAP C[$j$] with C[$j \oplus 2^u$] locally for $j \in \{1, 2, ..., k\}$.
          ENDIF
          CALL GCEA (i, D, $\frac{QR}{N}$)
          DO i = 1, k
                    A [i] = A [i] + C [i] * D [i]
          ENDDO
ENDDO
```

It is also possible to design a matrix multiplication algorithm based on the SBT-b$(1, N)$ 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.

```
/* Matrix multiplication using the SBT-b$(1, N)$ Algorithm: */
CC [cid] = C
DD [rid] = D
CALL sub_SBT (0, $n_2 - 1$, CC, $\frac{PQ}{N}$) /* within subcube of dimensions 0 to $n_2 - 1$ */
CALL sub_SBT ($n_2$, $n - 1$, DD, $\frac{QR}{N}$)
A = CC [0] * DD [0]
```

```
DO i = 1, $N_1 - 1$
      A = A + CC [i] * DD [i]
ENDDO
```

For *n-port* communication we only consider *all-to-all broadcasting* and the SBnT-b(n, N) and nRSBT-b(n, N) algorithms. 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 $B < \frac{2}{n} \lceil \frac{PQ}{N} \rceil$.

5.2.2 Algorithm 2

For the transposition of C we use a transpose algorithm as described in [10,8]. The *all-to-all broadcasting* of C^T can be made with algorithm CRA$(1, N_2)$, GCEA$(1, N_2)$, or SBT-b$(1, N_2)$ for *one-port* communication. The reduction can be carried out by the same algorithms, but in the other direction.

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 $\frac{P}{N_{max}} \frac{Q}{N_{max}}$.

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

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

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

5.2.3 Algorithm 3

For version 1 of algorithm 3 two matrix transpose operations are needed, one on the matrix C, and one on the matrix A. For these operations we use the transpose algorithms in [10]. For

Model	Algorithm	Communication operations
one-port	$A1(1)$	SBT-b$(1, N_2)[C]$ + SBT-b$(1, N_1)[D]$
	$A2(1)$	TXP$(1, N)[C]$ + SBT-b$(1, N_2)[C]$ + SBT-b$(1, N_1)[A]$
	$A3(1)$	TXP$(1, N)[C]$ + SBT-b$(1, N_2)[D]$ + SBT-b$(1, N_1)[A]$ + TXP$(1, N)[A]$
	$A4(1)$	TXP$(1, N)[D]$ + SBT-b$(1, N_1)[D]$ + SBT-b$(1, N_2)[A]$
	$A5(1)$	TXP$(1, N)[D]$ + SBT-b$(1, N_1)[C]$ + SBT-b$(1, N_2)[A]$ + TXP$(1, N)[A]$
n-port	$A1(n)$	SBnT-b$(n_2, N_2)[C]$ + SBnT-b$(n_1, N_1)[D]$
	$A2(n)$	TXP$(n, N)[C]$ + SBnT-b$(n_2, N_2)[C]$ + SBnT-b$(n_1, N_1)[A]$
	$A3(n)$	TXP$(n, N)[C]$ + SBnT-b$(n_2, N_2)[D]$ + SBnT-b$(n_1, N_1)[A]$ + TXP$(n, N)[A]$
	$A4(n)$	TXP$(n, N)[D]$ + SBnT-b$(n_1, N_1)[D]$ + SBnT-b$(n_2, N_2)[A]$
	$A5(n)$	TXP$(n, N)[D]$ + SBnT-b$(n_1, N_1)[C]$ + SBnT-b$(n_2, N_2)[A]$ + TXP$(n, N)[A]$

Table 8: Algorithm classification of two-dimensional partitioning.

the *all-to-all broadcasting* of D within rows algorithm SBT-b$(1, N_2)$ can be used. For the *all-to-all reduction* SBT-b$(1, N_1)$ is a possible choice. Recall that the complexity of the SBT-b$(1, *)$ is the same as that of the CRA$(1, *)$ or GCEA$(1, *)$ 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$(n, *)$ or the nRSBT-b$(n, *)$ routings are used instead.

For version 2 of algorithm 3 the *all-to-all 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$(1, *)$ algorithm, and for *n-port* communication either the SBnT-p$(n, *)$ or nRSBT-p$(n, *)$ algorithm.

5.2.4 Summary and Comparison

Tables 8 to 11 summarize the communication and arithmetic complexities. The optimum values of N_1 and N_2 depend on the algorithm we choose and the values of P, Q and R. In deriving the optimum values of N_1 and N_2, we assume that P is a multiple of N_1, Q a multiple of N_1 and N_2 and R a multiple of N_2. Table 12 lists the optimum values of N_1 and N_2 for the five algorithms we consider. Figure 18 shows the optimum values of N_1 for the $A1$ algorithm as a function of $\frac{P}{N_1}$, $\frac{R}{N_2}$ and N. The choice of N_1 does not depend on Q. The circle symbol denotes the boundary location, which has two minimum values of N_1. Note that the ratio of the slopes of the successive boundary lines is 4. The same Figure also applies to other algorithms by the appropriate relabeling of the axis. Figure 19 shows the measured and communication times of the $A1(1)$ algorithm 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.

Figures 20 to 23 show the best algorithm of the above five algorithms with respected to $\frac{\tau}{t_c}$, $\frac{P}{N}$, $\frac{Q}{N}$, $\frac{R}{N}$ and N. The comparison is made based on the times of each algorithm with its own optimum N_1 and N_2. The plot does not depend on the arithmetic rate by assuming that P, Q, R are all multiples of N. All Figures contains four plots. Plot (a), the upper left plot, shows the

Algorithm	Element transfers	start-ups
$A1(1)$	$(N_2-1)\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil$ $+(N_1-1)\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$\sum_{i=0}^{n_2-1}\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{2^i}{B_m}\rceil$
$A2(1)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil n+\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_2-1)$ $+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_1-1)$	$\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{1}{B_m}\rceil n+\sum_{i=0}^{n_2-1}\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{2^i}{B_m}\rceil$
$A3(1)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil n+\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_2-1)$ $+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_1-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil n$	$\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{1}{B_m}\rceil n+\sum_{i=0}^{n_2-1}\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{2^i}{B_m}\rceil+\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{1}{B_m}\rceil n$
$A4(1)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil n+\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil(N_1-1)$ $+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_2-1)$	$\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{1}{B_m}\rceil n+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_2-1}\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{2^i}{B_m}\rceil$
$A5(1)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil n+\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil(N_1-1)$ $+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_2-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil n$	$\lceil\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{1}{B_m}\rceil n+\sum_{i=0}^{n_1-1}\lceil\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{2^i}{B_m}\rceil$ $+\sum_{i=0}^{n_2-1}\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{2^i}{B_m}\rceil+\lceil\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{1}{B_m}\rceil n$

Table 9: The communication complexity of matrix multiplication using two-dimensional partitioning.

Algorithm	B_{opt}	*min* start-ups	Arithmetic
$A1(1)$	$\frac{N_2}{2}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil, \frac{N_1}{2}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	n	$2Q\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$
$A2(1)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil, \frac{N_2}{2}\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil,$ $\frac{N_1}{2}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$2n$	$(2\lceil\frac{Q}{N_1}\rceil-1)\lceil\frac{R}{N_2}\rceil P$ $+\sum_{i=1}^{n_1}\lceil\frac{R}{N_2}\rceil\lceil\frac{P}{2^i}\rceil+\lceil\frac{R}{N_2}\rceil\lceil\frac{P}{N_1}\rceil$
$A3(1)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil, \frac{N_2}{2}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil,$ $\frac{N_1}{2}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil, \lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil$	$3n$	$(2\lceil\frac{Q}{N_1}\rceil-1)\lceil\frac{P}{N_2}\rceil R$ $+\sum_{i=1}^{n_1}\lceil\frac{P}{N_2}\rceil\lceil\frac{R}{2^i}\rceil+\lceil\frac{P}{N_2}\rceil\lceil\frac{R}{N_1}\rceil)$
$A4(1)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil, \frac{N_1}{2}\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil,$ $\frac{N_2}{2}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	$2n$	$(2\lceil\frac{Q}{N_2}\rceil-1)\lceil\frac{P}{N_1}\rceil R$ $+\sum_{i=1}^{n_2}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{2^i}\rceil+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$
$A5(1)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil, \frac{N_1}{2}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil,$ $\frac{N_2}{2}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil, \lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil$	$3n$	$(2\lceil\frac{Q}{N_2}\rceil-1)\lceil\frac{R}{N_1}\rceil P$ $+\sum_{i=1}^{n_2}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{2^i}\rceil+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil)$

Table 10: The *optimum* communication complexity of matrix multiplication using two-dimensional partitioning with *one-port* communication. For the amount of element transfers, see Table 9.

Algorithm	B_{opt}	$T_{min}-$ arith. time
$A1(n)$	$\sqrt{\frac{2}{\pi}\frac{N_2}{{n_2}^{3/2}}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil}$, $\sqrt{\frac{2}{\pi}\frac{N_1}{{n_1}^{3/2}}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil}$	$\max(\frac{N_2-1}{n_2}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil t_c + n_2\tau,$ $\frac{N_1-1}{n_1}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_c + n_1\tau)$
$A2(n)$	$\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\frac{2}{\pi}\frac{N_2}{{n_2}^{3/2}}\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil}$, $\sqrt{\frac{2}{\pi}\frac{N_1}{{n_1}^{3/2}}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil}$	$n\tau + (\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{N_2-1}{n_2} + \lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{N_1-1}{n_1})t_c$
$A3(n)$	$\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\frac{2}{\pi}\frac{N_2}{{n_2}^{3/2}}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil}$, $\sqrt{\frac{2}{\pi}\frac{N_1}{{n_1}^{3/2}}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil}$	$n\tau + (\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{N_2-1}{n_2} + \lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{N_1-1}{n_1})t_c$
$A4(n)$	$\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\frac{2}{\pi}\frac{N_1}{{n_1}^{3/2}}\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil}$, $\sqrt{\frac{2}{\pi}\frac{N_2}{{n_2}^{3/2}}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil}$	$n\tau + (\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{N_1-1}{n_1} + \lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{N_2-1}{n_2})t_c$
$A5(n)$	$\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{\tau}{(n-1)t_c}}$, $\sqrt{\frac{2}{\pi}\frac{N_1}{{n_1}^{3/2}}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil}$, $\sqrt{\frac{2}{\pi}\frac{N_2}{{n_2}^{3/2}}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil}$	$n\tau + (\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{N_1-1}{n_1} + \lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{N_2-1}{n_2})t_c$

Table 11: The *optimum* communication complexity of matrix multiplication using two-dimensional partitioning with *n-port* communication. The arithmetic time is the same as in the *one-port* case and is omitted from the last column.

Algorithm	N_1	N_2	$T_{min}, \quad B_m \geq B_{opt}$
$A1(1)$	$\sqrt{\frac{PN}{R}}$	$\sqrt{\frac{RN}{P}}$	$\frac{2PQR}{N}t_a + \frac{Q}{\sqrt{N}}(2\sqrt{PR} - \frac{P+R}{\sqrt{N}})t_c + n\tau$
$A2(1)$	$\sqrt{\frac{QN}{R}}$	$\sqrt{\frac{RN}{Q}}$	$\frac{2PQR}{N}t_a + \frac{P}{\sqrt{N}}(2\sqrt{QR} + \frac{nQ-(Q+R)}{\sqrt{N}})t_c + 2n\tau$
$A3(1)$	$\sqrt{\frac{QN}{P}}$	$\sqrt{\frac{PN}{Q}}$	$\frac{2PQR}{N}t_a + \frac{R}{\sqrt{N}}(2\sqrt{PQ} + \frac{nP(1+\frac{Q}{R})-(P+Q)}{\sqrt{N}})t_c + 3n\tau$
$A4(1)$	$\sqrt{\frac{PN}{Q}}$	$\sqrt{\frac{QN}{P}}$	$\frac{2PQR}{N}t_a + \frac{R}{\sqrt{N}}(2\sqrt{PQ} + \frac{nQ-(P+Q)}{\sqrt{N}})t_c + 2n\tau$
$A5(1)$	$\sqrt{\frac{RN}{Q}}$	$\sqrt{\frac{QN}{R}}$	$\frac{2PQR}{N}t_a + \frac{P}{\sqrt{N}}(2\sqrt{QR} + \frac{nR(1+\frac{Q}{P})-(Q+R)}{\sqrt{N}})t_c + 3n\tau$

Table 12: The *optimum* values of N_1 and N_2 for P, Q and R being multiples of N and *one-port* communication.

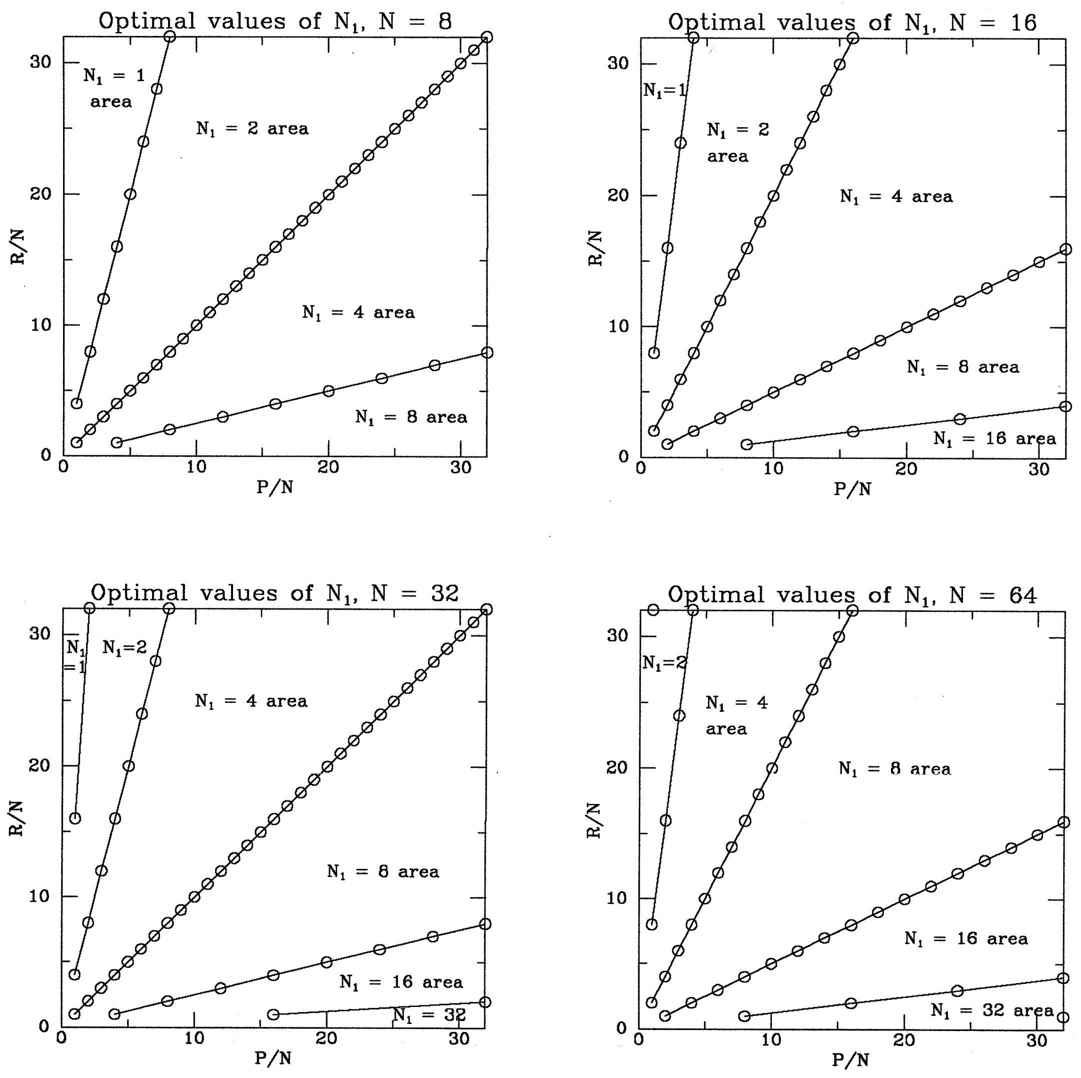


Figure 18: Optimal values of N_1 for the $A1$ algorithm with *one-port* communication.

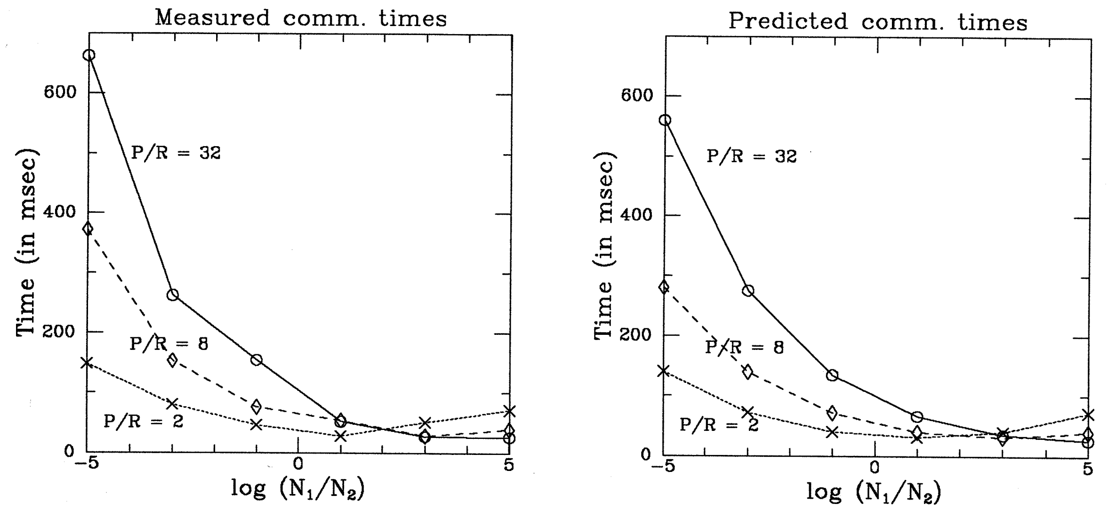


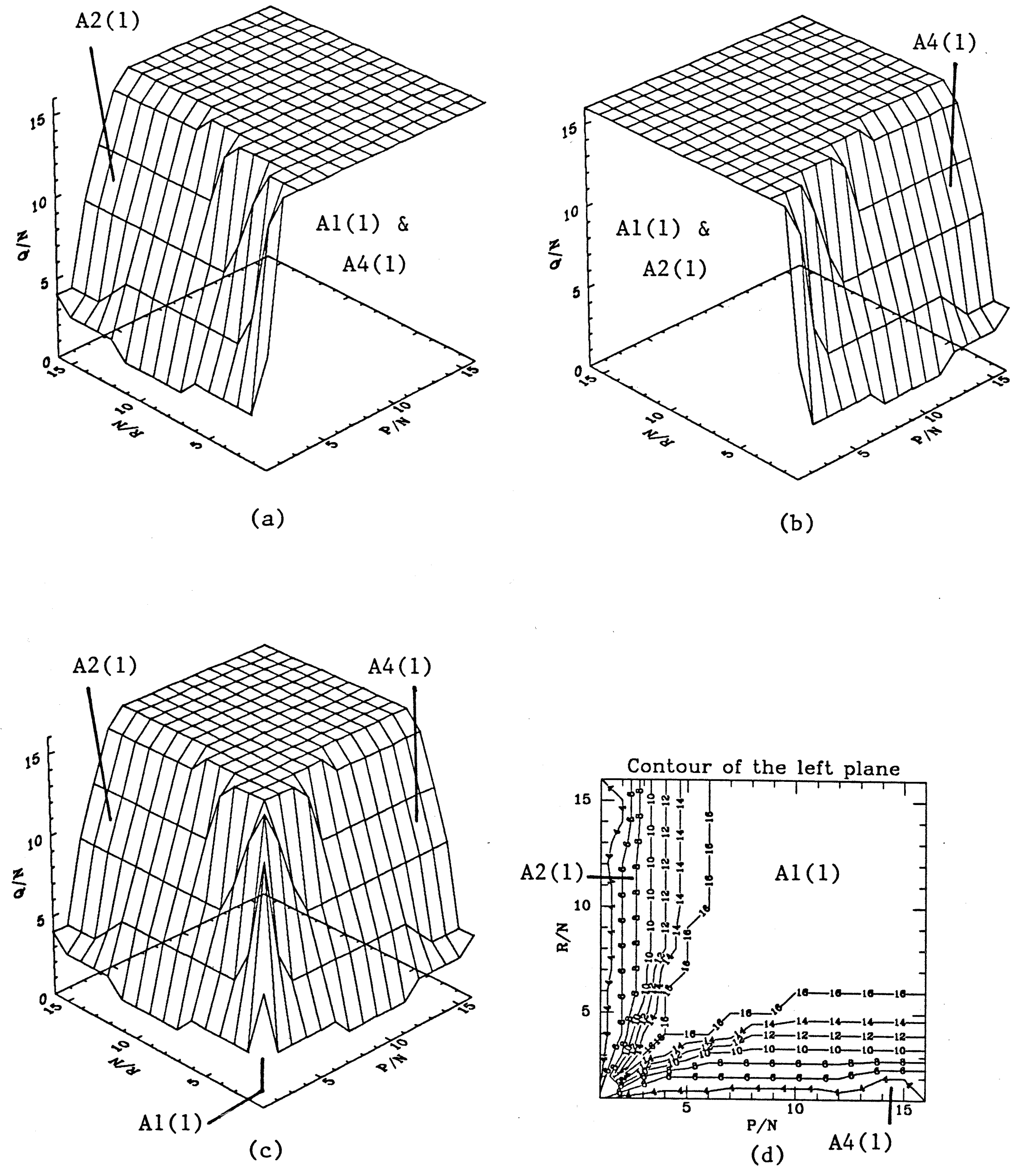
Figure 19: Measured and predicted communication times for the $A1(1)$ algorithm on an iPSC as a function of $\frac{N_1}{N_2}$ with $N = Q = 32$ and $PR = 1024N$.

boundary between $A2(1)$ (above the boundary) and $A1(1) + A4(1)$. Plot (b), the upper right plot, shows the boundary between $A4(1)$ (above the boundary) and $A1(1)+A2(1)$. Plot (c), the lower left plot, shows the boundary between $A1(1)$ (below the boundary) and $A2(1) + A4(1)$. Plot (d), the lower right plot, shows the contour of the plot (c).

For $N = 16$ and $\tau = t_c$, the volume within the considered domain is mostly in the region of the $A1$ algorithm, Figure 20. With the increasing aspect ratio of $\frac{\tau}{t_c}$, the region of the $A1$ algorithm also increases. As the number of processors increases, such as from Figure 20 to 23, the region in which algorithms $A2(1)$ or $A4(1)$ 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 23. For $\frac{P}{N} = \frac{Q}{N} = \frac{R}{N}$, the communication complexity of algorithm $A1(1)$ is less than that of algorithms $A2(1)$ and $A4(1)$; the complexities of $A2(1)$ and $A4(1)$ are the same. The volume for which $A1(1)$ is optimum is always at least $\frac{1}{3}$ of the whole volume. The volumes for $A2(1)$ and $A4(1)$ 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 $A1(1)$, the shape of its optimum volume is a pyramid with a square base at $\frac{Q}{N} = 0$ plane.

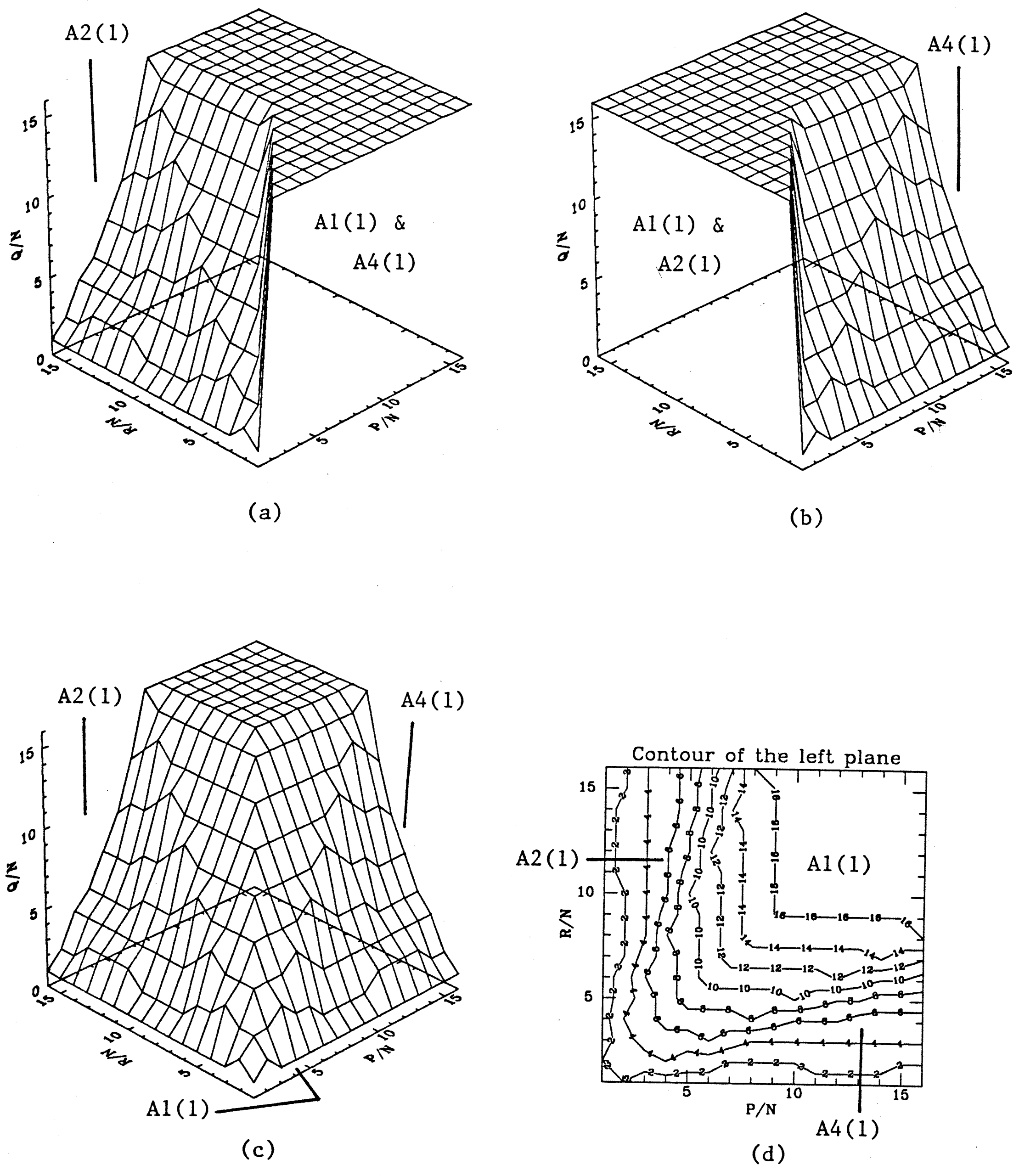
With *n-port* communication, the optimum value of N_1 for the $A2(n)$ algorithm has to minimize $R\frac{(N_1-1)}{n_1} + Q\frac{(N_2-1)}{n_2}$ if P, Q, R are multiples of N. Figure 24 shows the optimum values of N_1 for the $A2(n)$ algorithm. For algorithms $A3(n)$, $A4(n)$ and $A5(n)$, (Q, R) are replaced by (Q, P), (P, Q) and (R, Q) respectively. For algorithm $A1(n)$, Q is replaced by P, approximately.

Figures 25 and 26 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 $A1(n)$ is preferable with respect to execution time for most values of P, Q, and R. Algorithm $A1(n)$ shall be chosen if P, $R \geq Q$. Algorithm $A2(n)$ shall be chosen only if Q, $R > P$, and $A4(n)$ only if Q, $P > R$. The volume for $A2(n)$ ($A4(n)$) is significantly smaller than that for $A2(1)$



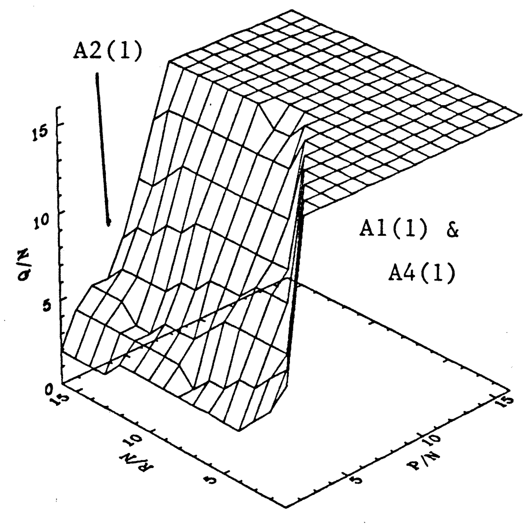
(a) The boundary between $A2(1)$ and $A1(1) + A4(1)$. (b) The boundary between $A4(1)$ and $A1(1) + A2(1)$. (c) The boundary between $A1(1)$ and $A2(1) + A4(1)$. (d) Contour plot of (c).

Figure 20: Lowest complexity algorithm as a function of matrix shape; $N = 16$, $\tau = t_c$, $B_m = \infty$, two-dimensional partitioning, *one-port* communication.

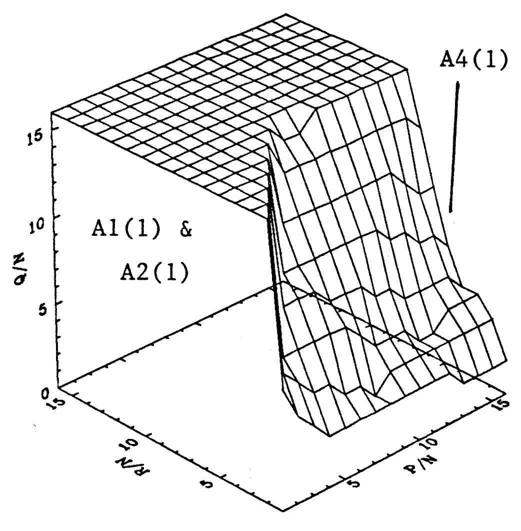


(a) The boundary between $A2(1)$ and $A1(1) + A4(1)$. (b) The boundary between $A4(1)$ and $A1(1) + A2(1)$. (c) The boundary between $A1(1)$ and $A2(1) + A4(1)$. (d) Contour plot of (c).

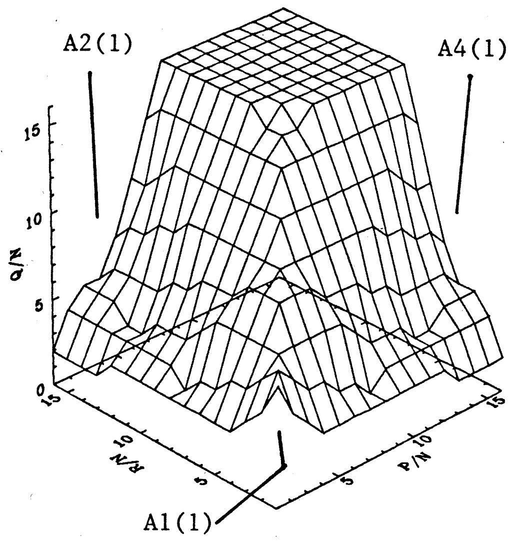
Figure 21: Lowest complexity algorithm as a function of matrix shape; $N = 128$, $\tau = t_c$, $B_m = \infty$, two-dimensional partitioning, *one-port* communication.



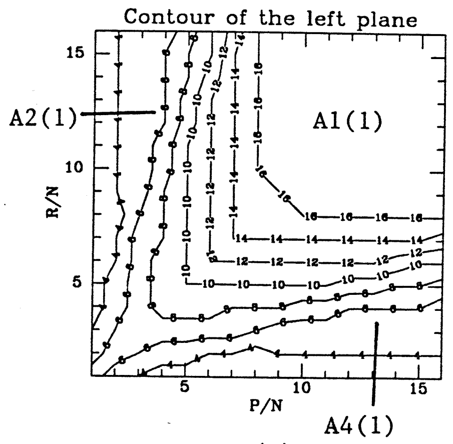
(a)



(b)



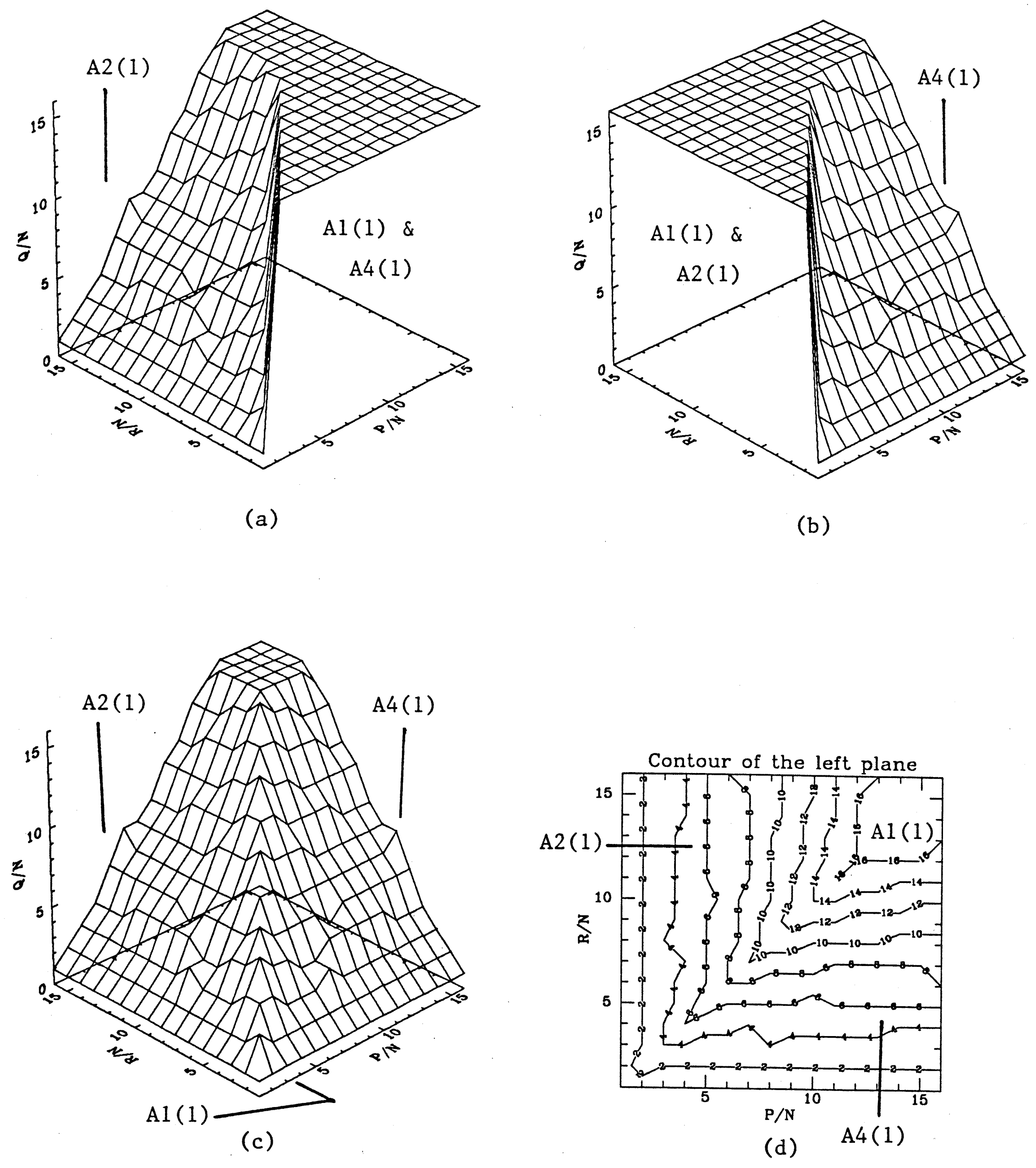
(c)



(d)

(a) The boundary between $A2(1)$ and $A1(1) + A4(1)$. (b) The boundary between $A4(1)$ and $A1(1) + A2(1)$. (c) The boundary between $A1(1)$ and $A2(1) + A4(1)$. (d) Contour plot of (c).

Figure 22: Lowest complexity algorithm as a function of matrix shape; $N = 128$, $\tau = 1000t_c$, $B_m = \infty$, two-dimensional partitioning, *one-port* communication.



(a) The boundary between $A2(1)$ and $A1(1) + A4(1)$. (b) The boundary between $A4(1)$ and $A1(1) + A2(1)$. (c) The boundary between $A1(1)$ and $A2(1) + A4(1)$. (d) Contour plot of (c).

Figure 23: Lowest complexity algorithm as a function of matrix shape; $N = 1024$, $\frac{\tau}{t_c} = 1$ to 1000, two-dimensional partitioning, *one-port* communication.

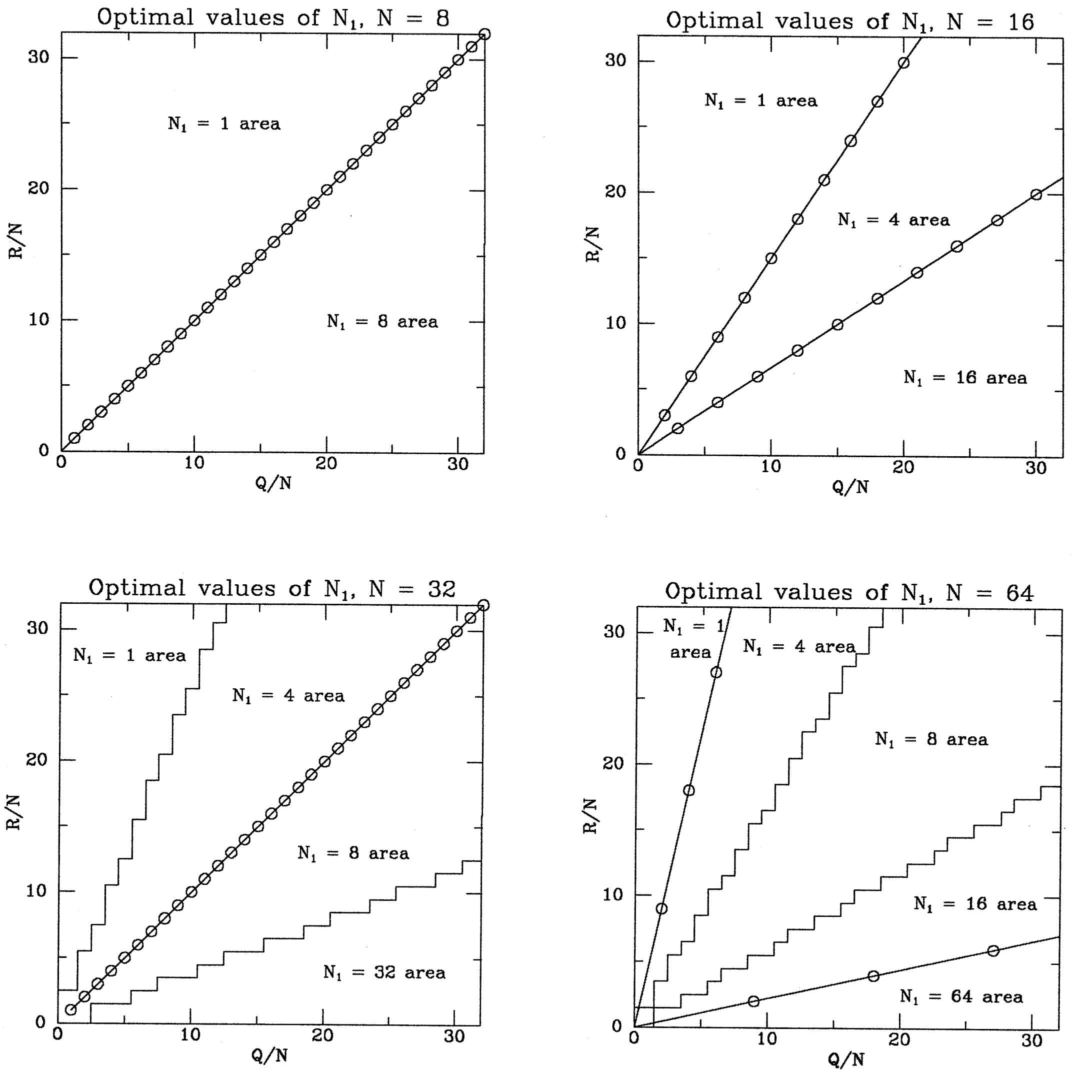


Figure 24: Optimal values of N_1 for algorithm $A2(n)$ with *n-port* communication.

($A4(1)$) of the corresponding *one-port* case.

6 Conclusions

A two-dimensional partitioning yields a complexity that is at most the same as that of a one-dimensional partitioning. In the one-dimensional partitioning the processors shall be aligned with the axis with the largest number of elements. In the two dimensional partitioning the processing plane shall be aligned with the matrix plane with the maximum number of elements. The aspect ratio of the processing array shall be the same as that of the matrix plane.

For the Boolean cube a reduction in the number of start-ups is possible in exchange for an increase in buffer sizes, and temporary storage. With a temporary storage that is equal to the size of the partitioned matrices, $O(N)$ communication steps are required for the one-dimensional partitioning while only $O(N_1, N_2)$ communication steps are required for the two-dimensional partitioning. The two-dimensional partitioning offers a reduction in element transfer time and number of start-ups by a factor of approximately $\frac{2}{\sqrt{N}}$ in the case of *one-port* communication and optimum aspect ratio for the partitioning $(\frac{N_1}{N_2})$. The number of start-ups is also reduced by a factor of approximately $\frac{2}{\sqrt{N}}$ for the optimum aspect ratio if $B_m < \frac{PQ}{N}$. For sufficiently large values of B_m the number of start-ups is the same in both the one- and two-dimensional partitioning, but the value of B_m at which the minimum number of start-ups is achieved is lower by a factor of approximately $\sqrt{N}$ in the two-dimensional case.

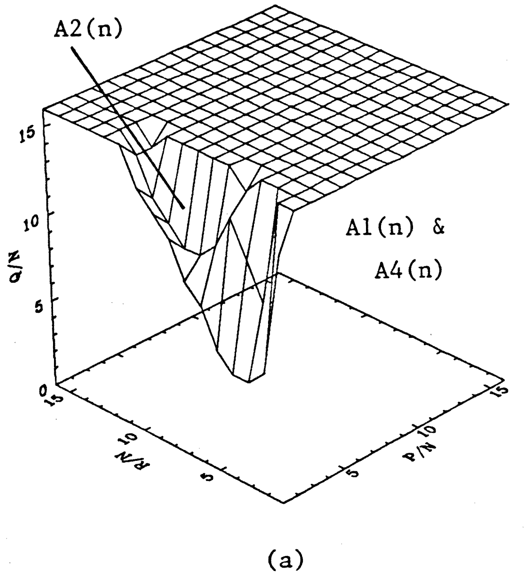
The CRA and the GCEA algorithms in one and two dimensions preserves storage requirements. The Spanning Binomial Tree algorithm only requires $\log N$ communication steps at the expense of larger temporary storage and optimum buffer size. With limited temporary storage (and maximum buffer size), a hybrid 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.

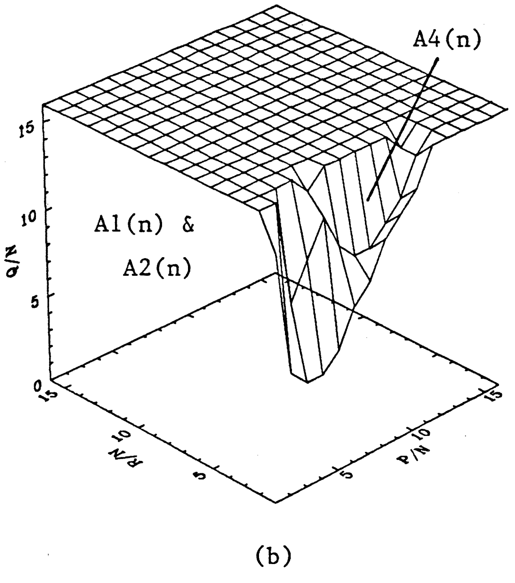
We have also devised algorithms with three-dimensional partitioning of the matrices as described in [16]. It shows that if the initial matrices are properly located, or if the maximum packet size is small relative to the matrix size, or if the communication start-up times are small relative to the data transmission times, three-dimensional partitioning is always preferable.

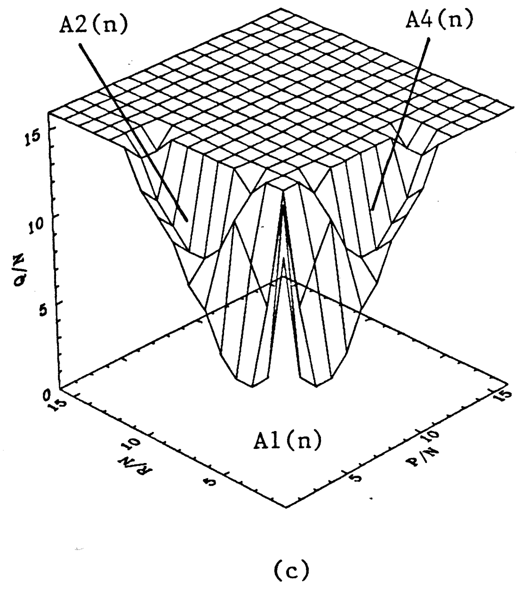
Throughout the paper it is implicitly assumed that the matrices are stored in the processors local storage by a consecutive strategy [14]. The complexity results also hold for cyclic storage. All the algorithms described work for binary code and Gray code encodings. The communication patterns are the same; only the local data accessing schemes differ.

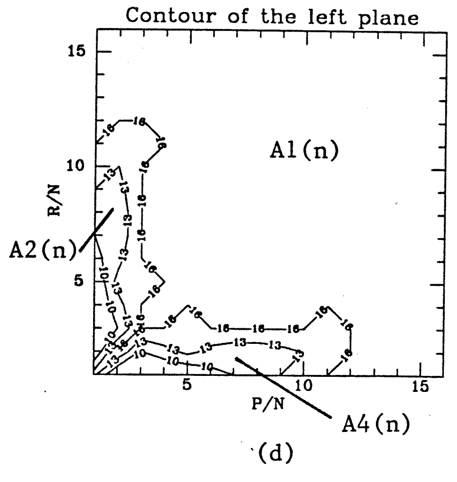
Acknowledgement

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



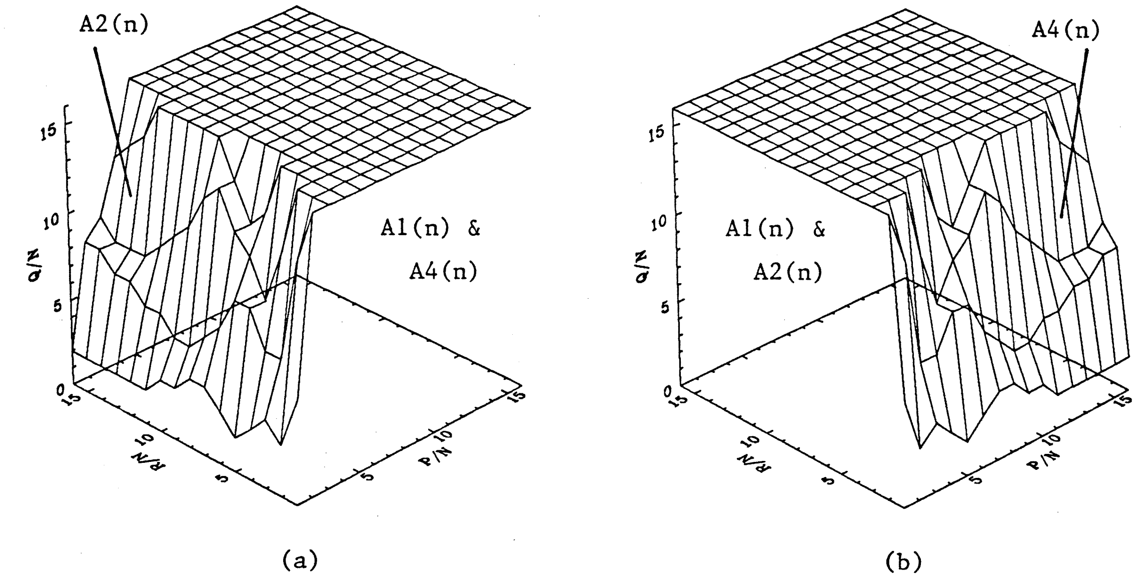


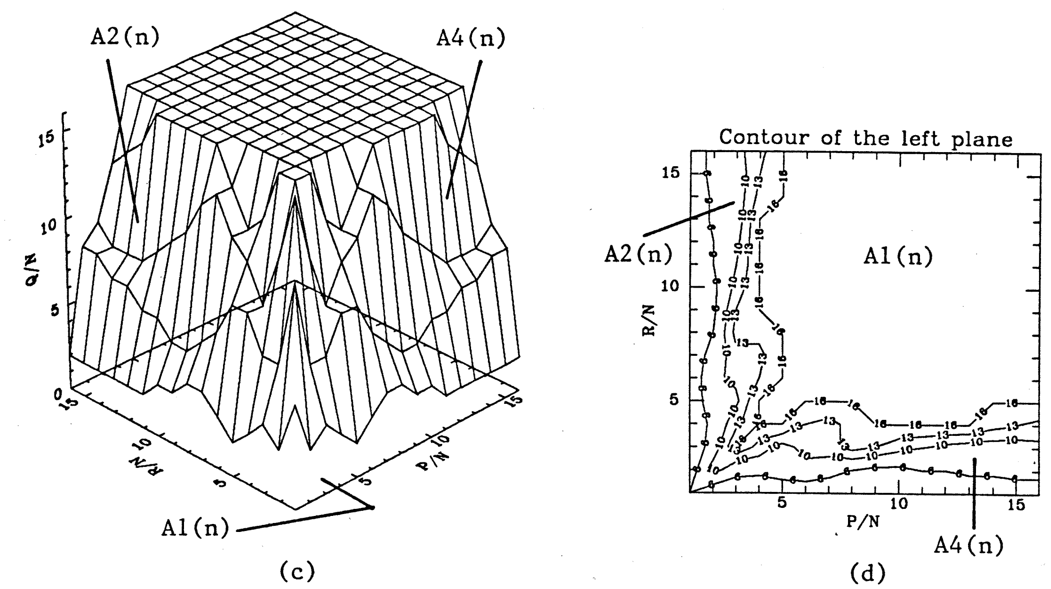




(a) The boundary between $A2(1)$ and $A1(1) + A4(1)$. (b) The boundary between $A4(1)$ and $A1(1) + A2(1)$. (c) The boundary between $A1(1)$ and $A2(1) + A4(1)$. (d) Contour plot of (c).

Figure 25: 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 $A2(1)$ and $A1(1) + A4(1)$. (b) The boundary between $A4(1)$ and $A1(1) + A2(1)$. (c) The boundary between $A1(1)$ and $A2(1) + A4(1)$. (d) Contour plot of (c).

Figure 26: Lowest complexity algorithm as a function of matrix shape; $N = 1024$, $\tau = t_c$, $B_m = \infty$, two-dimensional partitioning, *n-port* communication.

References

[1] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

[2] Romas Aleliunas and Arnold L. Rosenberg. On embedding rectangular grids in square grids. *IEEE Trans. Computers,* C-31(9):907–913, September 1982.

[3] Sandeep N. Bhatt and Ilse I.F. Ipsen. *How to Embed Trees in Hypercubes.* Technical Report YALEU/CSD/RR-443, Yale University, Dept. of Computer Science, December 1985.

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

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

[6] Sanjay R. Deshpande and Roy M. Jenevin. Scaleability of a binary tree on a hypercube. In *International Conference on Parallel Processing,* pages 661–668, IEEE Computer Society, 1986. TR-86-01, Univ. Texas at Austin.

[7] Michael J. Fischer. *Efficiency of Equivalence Algorithms,* pages 153–167. Plenum Press, 1972.

[8] 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.

[9] Ching-Tien Ho and S. Lennart Johnsson. Distributed routing algorithms for broadcasting and personalized communication in hypercubes. In *1986 Int. Conf. Parallel Processing,* pages 640–648, IEEE Computer Society, 1986. Tech. report YALEU/CSD/RR-483.

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

[11] Ching-Tien Ho and S. Lennart Johnsson. On the embedding of arbitrary meshes in Boolean cubes with expansion two dilation two. In *Int. Conf. on Parallel Processing,* pages 188–191, IEEE Computer Society, 1987.

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

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

[14] 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/CSD/RR-361, January 1985).

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

[16] S. Lennart Johnsson and Ching-Tien Ho. *Algorithms for Multiplying Matrices of Arbitrary Shapes Using Shared Memory Primitives on a Boolean Cube.* Technical Report YALEU/CSD/RR-, Department of Computer Science, Yale University, October 1987.

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

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

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

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

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

[22] Charles L. Seitz. Ensemble architectures for VLSI – a survey and taxonomy. In P. Penfield Jr., editor, *1982 Conf on Advanced Research in VLSI*, pages 130 – 135, Artech House, January 1982.

[23] Angela Y. Wu. Embedding of tree networks in hypercubes. *Journal of Parallel and Distributed Computing*, 2(3):238–249, 1985.