# 1 Introduction

Dense matrix multiplication is used in a variety of applications and is one of the core components in many scientific computations. The standard way of multiplying two matrices of size $n \times n$ requires $O(n^3)$ floating point operations on a sequential machine. Since dense matrix multiplication is computationally expensive, the development of efficient algorithms for large distributed memory machines is of great interest. Matrix multiplication is a very regular computation and lends itself well to parallel implementation. One of the efficient approaches to design other parallel matrix or graph algorithms is to decompose them into a sequence of matrix multiplications [3, 9].

One of the earliest distributed algorithms proposed for matrix multiplication was by Cannon [2] in 1969 for 2-D meshes. Ho, Johnsson, and Edelman in [8] presented a variant of Cannon's algorithm which uses the full bandwidth of a 2-D grid embedded in a hypercube. Some other algorithms are by Dekel, Nassimi, and Sahni [3], Berntsen [1] and Fox, Otto, and Hey [4]. Gupta and Kumar in [5] discuss the scalability of these algorithms and their variants.

In this paper we propose two new algorithms for hypercubes. The algorithms proposed in this paper are better than all previously proposed algorithms for a wide range of matrix sizes and number of processors.

The rest of the paper is organized as follows. In Section 2 we state our assumptions and discuss the communication models used. In Section 3 we discuss the previously proposed algorithms. In Section 4 we present the new algorithms. Section 5 presents some optimality results. In Section 6, we analyze the performance of the algorithms on hypercubes for three different communication cost parameters. We present our conclusions in Section 7.

# 2 Communication Models

In this paper we analyze the performance of the various algorithms presented for hypercube architectures. Throughout this paper, we refer to a 2-ary n-cube as a hypercube and all the *logarithms* used are with respect to the base 2. We consider hypercube machines with *one-port* processor nodes as well as machines with *multi-port* processor nodes. In case of the one-port hypercube architectures, a processor node can use at most one communication link (to send *and* receive) at any given time while in the multi-port architectures a processor node can use all its communication links simultaneously.

The time required for a processor node to send a message of $m$ words to a neighboring processor node is modeled as $t_s + t_w m$, where $t_s$ is the message start-up cost and $t_w$ is the data transmission time per word. All the algorithms presented in this paper run on a virtual 2-D or 3-D grid of processors. Any collective communication pattern involved in an algorithm presented in this paper is along a one-dimensional chain of processors. In case of a virtual 2-D or 3-D grid embedded into a hypercube, each one-dimensional chain of processors is in itself a hypercube of smaller dimension [6].

Ho and Johnsson [7] present scheduling disciplines for performing *broadcasting* and *personalized communication* on Boolean $n$-cube configured ensemble architectures. They address four different communication patterns, viz. one-to-all broadcast, one-to-all personalized communication, all-to-all broadcast, and all-to-all personalized communication. In *one-to-all broadcast*, a data set of size $M$ words is copied from one node to all other nodes. *One-to-all personalized communication* involves a single node sending unique data of size $M$ words to all other nodes. In *all-to-all broadcast*, each node distributes its data of size $M$ to all other nodes while in *all-to-all personalized communication* each node sends a unique piece of information of size $M$ to every other node. Fundamentally, the difference between broadcasting and personalized communication is that in the former data replication helps in dissemination of data and hence, it incurs less communication overhead in comparison to personalized communication. The reverse of the broadcasting operation is *reduction*, in which the data set is reduced by applying operators such as addition/subtraction.

When the communication is restricted to one port at a time (as in one-port hypercubes), the spanning binomial tree (SBT) scheduling results in optimum communication time. In the case of multi-port hyper-

| Communication type | Hypercubes | | |
|---|---|---|---|
| | $t_s$ term | $t_w$ term | |
| | | One-port[1] | Multi-port[2]($M \geq \log N$) |
| One-to-All Broadcast | $\log N$ | $M \log N$ | M |
| One-to-All Personalized Communication | $\log N$ | $(N-1)M$ | $\frac{(N-1)M}{\log N}$ |
| All-to-All Broadcast | $\log N$ | $(N-1)M$ | $\frac{(N-1)M}{\log N}$ |
| All-to-All Personalized Communication | $\log N$ | $\frac{NM \log N}{2}$ | $\frac{NM}{2}$ |

Table 1: Optimal broadcasting and personalized communication on an $N$-processor hypercube. $M$ is the message length in words.

cubes, when each processor is able to communicate on all its ports concurrently, optimum communication time is achieved by using $\log N$ rotated spanning binomial trees simultaneously, where $N$ is the number of processors in the hypercube. In our analysis of communication overheads for various algorithms we use some of these results presented by Ho and Johnsson [7] for optimal broadcasting and personalized communication in hypercubes. Table 1 summarizes the results used in this paper. Note that the message length $M$ should be greater than or equal to $\log N$, the number of communication links incident on each processor, for communication patterns on multi-port hypercube so that all the $\log N$ spanning binomial trees can be used concurrently.

# 3 Distributed Matrix Multiplication Algorithms

In this section we present the well known distributed algorithms for multiplying two dense matrices A and B of size $n \times n$. The characteristics of the algorithms presented in this and the following section have been summarized in Table 2 and Table 3.

## 3.1 Algorithm Simple

Consider a hypercube of $p$ processors mapped onto a $\sqrt{p} \times \sqrt{p}$ 2-D mesh. Matrices A and B are block partitioned into $\sqrt{p}$ blocks along each dimension as shown in Figure 1. The sub-blocks $A_{ij}$ and $B_{ij}$ are mapped onto processor $p_{ij}$, the processor in the $i^{th}$ row and $j^{th}$ column, $0 \leq i, j < \sqrt{p}$, of the 2-D mesh. Thus, each processor initially has $\frac{n^2}{p}$ elements of each matrix.

The algorithm consists of two communication phases. In the first phase, all processors in each row independently engage in an *all-to-all broadcast* of the sub-blocks of matrix A among themselves. In the second phase, all processors in each column independently engage in an *all-to-all broadcast* of the sub-blocks of matrix B. At the end of these two phases, each processor $p_{ij}$ has all the required sub-blocks of matrices A and B to compute the block $C_{ij}$ of the result matrix.

Each phase of the algorithm involves an all-to-all broadcast of messages of size $\frac{n^2}{p}$ among $\sqrt{p}$ processors in each row or column and hence takes $t_s \log \sqrt{p} + t_w \frac{n^2}{\sqrt{p}}(1 - \frac{1}{\sqrt{p}})$ time on a one-port hypercube and $t_s \log \sqrt{p} + t_w \frac{n^2}{\sqrt{p} \log \sqrt{p}}(1 - \frac{1}{\sqrt{p}})$ on a multi-port hypercube (see Table 1). On a multi-port hypercube architecture, the two communication phases can occur in parallel. The size of the message ($\frac{n^2}{p}$) should be greater than or equal to $\log \sqrt{p}$, the number of communication channels along a dimension, for communication patterns on

[1] Using a Spanning Binomial Tree (SBT)
[2] Using $\log N$ trees concurrently

| | | | |
|---|---|---|---|
| $A_{00}$ | $A_{01}$ | $A_{02}$ | $A_{03}$ |
| $A_{10}$ | $A_{11}$ | $A_{12}$ | $A_{13}$ |
| $A_{20}$ | $A_{21}$ | $A_{22}$ | $A_{23}$ |
| $A_{30}$ | $A_{31}$ | $A_{32}$ | $A_{33}$ |

Figure 1: Matrix A partitioned into 4 $\times$ 4 blocks.

multi-port hypercubes. This algorithm is very inefficient with respect to space as each processor uses $\frac{2n^2}{\sqrt{p}}$ words of memory.

### 3.2 Cannon's Algorithm

This algorithm is designed for execution on a virtual 2-D grid of processors. Matrices A and B are mapped naturally onto the processors as in Algorithm Simple. Cannon's algorithm executes in two phases. The first phase essentially skews the matrices A and B to align them appropriately. In this phase sub-block $A_{ij}(B_{ij})$ is shifted left (up) circularly by some number of positions along the row (column) of processors such that the processor $p_{ij}$ receives $A_{i,(j+i) mod n}$ and $B_{(i+j) mod n, j}$. The second phase is a sequence of $(\sqrt{p}-1)$ shift-multiply-add operations. During each step $A_{ij}(B_{ij})$ is shifted left (up) circularly by one processor and each processor multiplies the newly acquired sub-blocks of A and B and adds the result to the sub-block $C_{ij}$ being maintained. Each processor requires $3\frac{n^2}{p}$ words of memory to store sub-blocks of A, B, and C matrices.

Consider a 2-D grid of processors embedded into a physical $p$-processor hypercube. The communication time required for the initial alignment on a one-port hypercube is $2\log\sqrt{p}(t_s + t_w\frac{n^2}{p})$ while the second phase takes $2(\sqrt{p}-1)t_s + 2\frac{n^2}{p}(\sqrt{p}-1)t_w$ time as each shift-multiply-add operation takes $2(t_s + t_w\frac{n^2}{p})$. In case of the multi-port hypercube architectures, both the A and B sub-blocks can be communicated in parallel, halving the time required. The greatest advantage of this algorithm is that it uses constant storage, independent of the number of processors.

### 3.3 Ho-Johnsson-Edelman Algorithm

The second phase of Cannon's algorithm has the same performance on 2-D tori and hypercubes. It can be further improved on hypercubes by using the full bandwidth available, provided the sub-blocks of matrices A and B are large enough. Such a variant was proposed by Ho, Johnsson, and Edelman [8]. This algorithm is different from Cannon's only for multi-port hypercubes. We present here only a brief sketch of the algorithm taken from [9]. See Algorithm 1. The reader is referred to the original paper for details.

On a virtual $\sqrt{p}\times\sqrt{p}$ 2-D grid embedded into a $p$-processor hypercube, the data transmission time for the shift-multiply-add phase of Cannon's algorithm is improved by a factor of $\log\sqrt{p}$, the total number of communication links on any processor along either grid dimension. This algorithm is applicable only when each processor has at least $\log\sqrt{p}$ rows and columns, i.e., when $\frac{n}{\sqrt{p}} \geq \log\sqrt{p}$. The space requirement is the same as that for Cannon's algorithm.

Consider the matrices A and B distributed evenly over a 2D grid of processors with equal dimensions. The result matrix C is said to be *accumulated in place* if the processor $p$, which eventually stores the sub-block $C_{ij}$, is responsible for calculating and accumulating the products $A_{ik}B_{kj}$ for all $k$. Ho et al. [8] prove that the data transfer time of Ho-Johnsson-Edelman Algorithm is optimal within a constant factor for matrix multiplication on hypercubes with the result matrix accumulated in place and the operands distributed uniformly over the processors configured as a 2D grid with equal dimensions.

**Algorithm 1: Ho-Johnsson-Edelman**

**Initial Distribution** Each processor $p_{i,j}$ contains $A_{ij}$ and $B_{ij}$.

Program of processor $p_{i,j}$

for $k = 1, \log \sqrt{p}$
  Let $j_k = (k^{th}$ bit of $j) \cdot 2^k$
  Let $i_k = (k^{th}$ bit of $i) \cdot 2^k$
  Send $A_{i,j}$ to $p_{i,j \otimes i_k}$
  Receive $A_{i,j}$ from $p_{i,j \otimes i_k}$
  /* $\otimes$ is the bit-wise *exclusive-or* operator */
  Send $B_{i,j}$ to $p_{j_k \otimes i,j}$
  Receive $B_{i,j}$ from $p_{j_k \otimes i,j}$
end for
Let $g_{l,k}$ be the bit position in which $\log \sqrt{p}$-bit gray codes, left shifted by $l$ bits, of the $k^{th}$ and $(k+1)^{th}$ numbers differ.
for $k = 1, \sqrt{p}$
  $C_{ij} = C_{ij} + A_{ij} \times B_{ij}$
  forall $l = 0, \log \sqrt{p} - 1$
    Send $A^l_{i,j}$ to $p_{i,j \otimes 2^{g_{l,k}}}$
    Receive $A^l_{i,j}$ from $p_{i,j \otimes 2^{g_{l,k}}}$
    /* where $A^l_{i,j}$ is the $l^{th}$ group of columns of $A_{i,j}$ */
    Send $B^l_{i,j}$ to $p_{i \otimes 2^{g_{l,k}},j}$
    Receive $B^l_{i,j}$ from $p_{i \otimes 2^{g_{l,k}},j}$
    /* where $B^l_{i,j}$ is the $l^{th}$ group of rows of $B_{i,j}$*/
  end forall
end for

Figure 2: Ho-Johnsson-Edelman Algorithm

### 3.4 Berntsen's Algorithm

In [1], Berntsen presents an algorithm for a hypercube. Consider a $p$-processor hypercube where $p \leq n^{3/2}$. Matrix A is split by columns and B by rows into $\sqrt[3]{p}$ sets. Each set contains $\frac{n}{\sqrt[3]{p}}$ rows or columns. The hypercube is divided into $\sqrt[3]{p}$ subcubes each consisting of $p^{2/3}$ processors. The $m^{th}$ subcube is delegated the task of calculating the outer product of the $m^{th}$ set of columns of A and the $m^{th}$ set of rows of B using Cannon's algorithm. Each set of rows (columns) of B (A) is block partitioned as shown in Figure 1 into $\sqrt[3]{p} \times \sqrt[3]{p}$ blocks for mapping onto the respective subcube processors. Each subcube calculates the outer product using Cannon's algorithm, with each processor performing a submatrix multiplication between submatrices of A of size $\frac{n}{\sqrt[3]{p}} \times \frac{n}{p^{2/3}}$ and submatrices of B of size $\frac{n}{p^{2/3}} \times \frac{n}{\sqrt[3]{p}}$. After computation of these $\sqrt[3]{p}$ outer products, an all-to-all reduction phase occurs among the corresponding processors from each subcube, which takes $t_s \log \sqrt[3]{p} + t_w \frac{n^2}{p^{2/3}}(1 - \frac{1}{\sqrt[3]{p}})$ time on a one-port hypercube. On a multi-port hypercube architecture the data transmission time can be reduced by a factor of $\log \sqrt[3]{p}$ as compared to a one-port hypercube by using the techniques presented in [7] (see Table 1) so that the time required is $t_s \log \sqrt[3]{p} + t_w \frac{n^2}{p^{2/3}} \frac{1}{\log \sqrt[3]{p}}(1 - \frac{1}{\sqrt[3]{p}})$. The size of each message being $\frac{n^2}{p}$ in the all-to-all reduction phase, $n^2$ should be greater than or equal to $p \log \sqrt[3]{p}$ for multi-port hypercubes. As each subcube of $p^{2/3}$ processors uses Cannon's algorithm to calculate the outer product, the space requirement for this algorithm is $2\frac{n^2}{p} + \frac{n^2}{p^{2/3}}$ words per processor to store the submatrices of A, B, and the outer product.

One of the drawbacks of this algorithm is that the algorithm starts with A and B distributed differently and the result obtained is not aligned in the same manner as A or B.

### 3.5 DNS Algorithm

Dekel, Nassimi and Sahni in [3] presented an algorithm for virtual 3-D meshes which uses $n^3$ processors. We consider here the more generalized version of the algorithm which can use *upto* $n^3$ processors by allowing a processor to store a sub-block rather than an element of a matrix. Consider a 3-D grid of dimensions $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$ embedded into a hypercube of $p$ processors. Initially matrices A and B are both mapped naturally, block partitioned, onto the $z = 0$ plane (the shaded region in Figure 3) such that processor $p_{i,j,0}$ contains the sub-blocks $A_{ij}$ and $B_{ij}$. The algorithm can be viewed as consisting of three phases. The first phase involves each processor $p_{i,j,0}$ transmitting $A_{ij}$ to $p_{i,j,j}$ and $B_{ij}$ to $p_{i,j,i}$. The second phase consists of two *one-to-all broadcasts* among sets of $\sqrt[3]{p}$ processors[3] with $p_{i,j,j}$ broadcasting $A_{ij}$ along the $y$-direction to $p_{i,*,j}$ and $p_{i,j,i}$ broadcasting $B_{ij}$ along the $x$-direction to $p_{*,j,i}$. At the end of this phase, each processor $p_{i,j,k}$ multiplies the sub-blocks $A_{ik}$ and $B_{kj}$ acquired during the first two phases. The last phase is an *all-to-one reduction* (by addition) which occurs along the $z$-direction. It is easy to see that the space required per processor is $2\frac{n^2}{p^{2/3}}$ words for this algorithm.

On a one-port hypercube architecture, each of the initial two phases takes $2 \log \sqrt[3]{p}(t_s + \frac{n^2}{p^{2/3}} t_w)$ time. The point-to-point communication of the sub-blocks of A and B in the first phase cannot be overlapped on a multi-port architecture as they both occur along the $z$-direction. However, in the second phase the two *one-to-all broadcasts* can occur in parallel. The reduction phase, being the inverse of a one-to-all broadcast of messages of size $\frac{n^2}{p^{2/3}}$, takes $\log \sqrt[3]{p}(t_s + \frac{n^2}{p^{2/3}} t_w)$ time on a one-port hypercube and $\log \sqrt[3]{p} t_s + \frac{n^2}{p^{2/3}} t_w$ time on a multi-port hypercube (see Table 1). Each phase occurs over sets of $\sqrt[3]{p}$ processors and involves messages of size $\frac{n^2}{p^{2/3}}$. Hence, $n^2$ should be greater than or equal to $p^{2/3} \log \sqrt[3]{p}$ for multi-port hypercubes.

In [3], Dekel, Nassimi, and Sahni also propose an algorithm, a combination of the above basic DNS algorithm and Cannon's algorithm, which calculates the product of the submatrices using Cannon's algorithm on a square sub-mesh of processors, saving overall space. More formally, the hypercube is visualized as a $\sqrt[3]{s} \times \sqrt[3]{s} \times \sqrt[3]{s}$ 3-D grid of supernodes where each supernode is a square mesh of $\sqrt{r} \times \sqrt{r}$ processor elements involved in computing the product of the submatrices of A and B using Cannon's algorithm. The two new algorithms presented in the next section have been shown to be better than the basic DNS algorithm in terms of the number of message start-ups as well as the data transmission time and hence the combination of any proposed new algorithm with Cannon's algorithm would yield an algorithm better than the combination algorithm of the DNS and Cannon. Hence, we present only the basic algorithms in this paper.

## 4 New Algorithms

In this section we present two new algorithms designed for hypercubes. In order to explain the rationale behind the algorithms, we present them in various stages. Throughout this section, $p_{i,j,k}$ is used to refer to the processor whose $x, y$, and $z$ co-ordinates are $i, j$, and $k$ respectively on the 3-D grid.

### 4.1 3-D Diagonal Approach

We first present a 2-D version of the 3-D Diagonal scheme and then extend it to the 3-D Diagonal algorithm in two stages.

[3] Each set is a one-dimensional row of processors forming a 2-ary subcube.

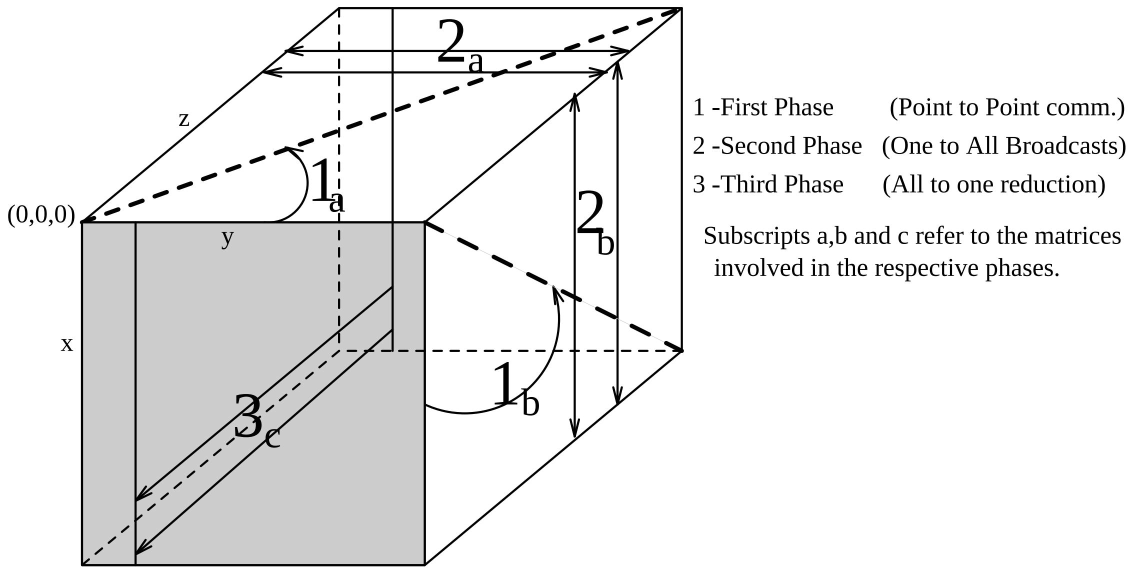


Figure 3: DNS Algorithm

#### 4.1.1 2-D Diagonal Algorithm

Consider a 2-D processor mesh of size $q \times q$, laid out on the $x$-$y$ plane. We use $p_{i,j}$ to refer to the processor whose $x$ and $y$ co-ordinates are $i$ and $j$ respectively. Matrix A is partitioned into $q$ groups of columns and matrix B is partitioned into $q$ groups of corresponding rows as shown in Figure 4. Initially, each processor $p_{j,j}$, on the diagonal of the mesh, contains the $j^{th}$ group of columns of A and the $j^{th}$ group of rows of B. The set of processors $p_{*,j}$ is delegated the task of computing the outer product of the columns of A and rows of B initially stored at $p_{j,j}$. This is achieved by having $p_{j,j}$ **scatter** (*one-to-all personalized communication*) the group of rows of B and **broadcast** (*one-to-all broadcast*) the group of columns of A along the $x$-direction. After computing the outer products, each processor doing equal amount of computation, the last stage consists of *reducing* the results by addition along the $y$-direction and the result matrix C is obtained along the diagonal processors, aligned in the same way as matrix A was initially distributed. See Algorithm 2.

The above algorithm can be easily extended to a 3-D mesh embedded in a hypercube with $A_{*,i}$ and $B_{i,*}$ being initially distributed along the third dimension, $z$, with processor $p_{i,i,k}$ holding the sub-blocks $A_{k,i}$ and $B_{i,k}$. The *one-to-all personalized communication* of $B_{i,*}$ is then replaced by point-to-point communication of $B_{i,k}$ from $p_{i,i,k}$ to $p_{k,i,k}$, followed by *one-to-all broadcast* of $B_{i,k}$ by $p_{k,i,k}$ along the $z$-direction to $p_{k,i,*}$. Apart from the initial communication of blocks of B, all other communication patterns along with their directions remain the same as in the 2-D diagonal scheme.

One of the problems with the above discussed 3-D extension of the 2-D diagonal approach is that the initial distribution assumed is not the same for matrices A and B. One obvious way to get around this problem is to first form the transpose of matrix B before executing the actual algorithm. In the next section, we present a variant of the above discussed 3-D diagonal scheme which computes the matrix product of matrices with identical initial distribution without any additional communication overhead.

#### 4.1.2 The 3-D Diagonal Algorithm

A hypercube consisting of $p$ processors can be visualized as a 3-D mesh of size $\sqrt[3]{p} \times \sqrt[3]{p} \times \sqrt[3]{p}$, where $p \leq n^3$. Matrices A and B are block partitioned into $p^{2/3}$ blocks with $\sqrt[3]{p}$ blocks along each dimension as shown in Figure 1. Initially, matrices A and B are assumed to be mapped on to the diagonal mesh corresponding to the 2-D plane $x = y$ (the shaded region in Figure 6), with processor $p_{i,i,k}$ containing the blocks $A_{k,i}$ and $B_{k,i}$.

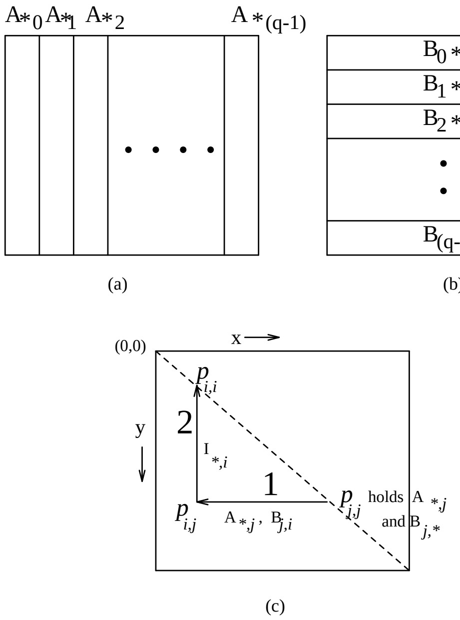


Figure 4: 2-D Diagonal Algorithm (a) Partitioning of A (b) Partitioning of B (c) The two phases of the algorithm (Here, $p_{i,j}$ refers to the processor whose $x$ and $y$ co-ordinates are $i$ and $j$ respectively).

**Algorithm 2:** 2-D Diagonal

**Initial Distribution:** Each diagonal processor $p_{i,i}$ holds the $i^{th}$ group of columns and rows of the matrices A and B respectively.

*Program of processor* $p_{i,j}$

If $(i = j)$ then
  Broadcast $A_{*,j}$ to all processors $p_{*,j}$
  /* $A_{*,j}$ is the $j^{th}$ group of columns of A, initially stored at $p_{j,j}$ */
  for $k = 0, q-1$
    Send $B_{jk}$ to $p_{k,j}$
    /* $B_{j,*}$ is the $j^{th}$ group of rows of B, initially stored at $p_{j,j}$ and $B_{jk}$ is the $k^{th}$ group of columns of $B_{j,*}$ */
  end for
endif
Receive $A_{*,j}$ and $B_{ji}$ from $p_{j,j}$
Calculate $I_{*,i} = A_{*,j} \times B_{ji}$
Send $I_{*,i}$ along the $y$-direction to $p_{i,i}$
If $(i = j)$ then
  for $k = 0,\ q-1$
    Receive $I_{*,i}$ from $p_{i,k}$
    $C_{*,i} = C_{*,i} + I_{*,i}$
  endfor
endif

Figure 5: 2-D Diagonal Algorithm

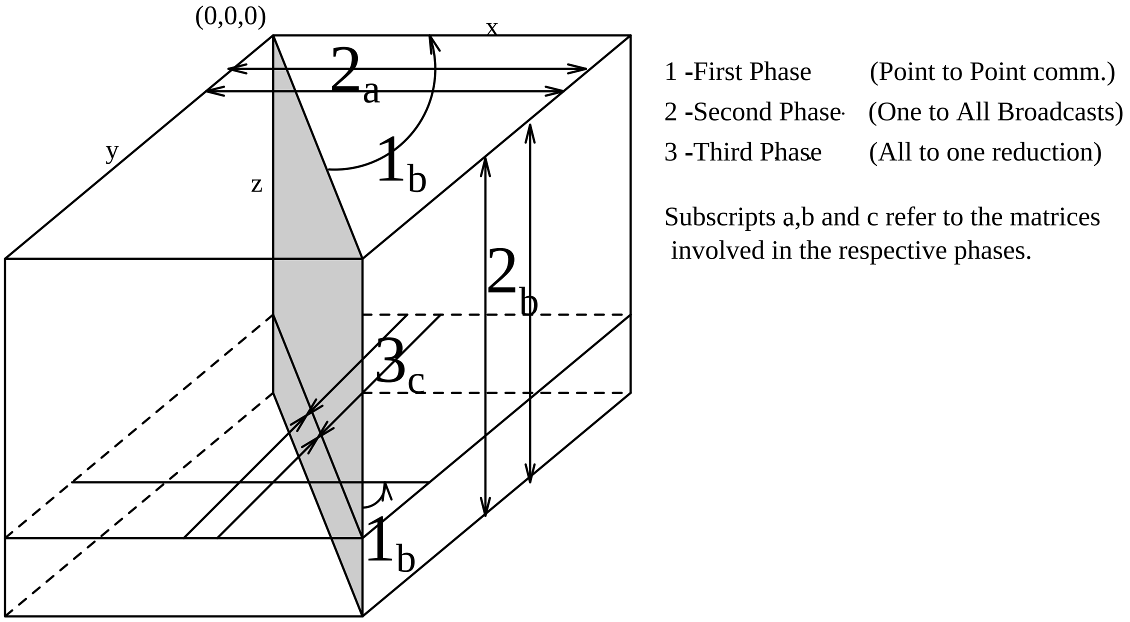


Figure 6: 3D Diagonal Algorithm

In this algorithm the 2-D plane $y = j$ has the responsibility of calculating the outer product of $A_{*,j}$, the set of columns initially stored at the processors $p_{j,j,*}$, and $B_{j,*}$, the corresponding set of rows of B. The algorithm consists of three phases. Point-to-point communication of $B_{k,i}$ by $p_{i,i,k}$ to $p_{i,k,k}$ forms the first phase of the algorithm. The second phase consists of *one-to-all broadcasts* of blocks of A along the $x$-direction and the newly acquired blocks of B along the $z$-direction. In other words, processor $p_{i,i,k}$ broadcasts $A_{k,i}$ to $p_{*,i,k}$ and every processor of the form $p_{i,k,k}$ broadcasts $B_{k,i}$ to $p_{i,k,*}$. At the end of the second phase, every processor $p_{i,j,k}$ has blocks $A_{k,j}$ and $B_{j,i}$. Each processor now calculates the product of the acquired blocks of A and B. After the computation stage, the *reduction* by addition of the result submatrices along the $y$-direction constitutes the third and the final phase. The result matrix C is obtained aligned in the same manner as the source matrices A and B. See Algorithm 3. The overall space used by this algorithm is $2n^2 \sqrt[3]{p}$.

The first phase of the 3DD algorithm, being a point-to-point communication phase of messages of size $\frac{n^2}{p^{2/3}}$, takes $\log \sqrt[3]{p}(t_s + t_w \frac{n^2}{p^{2/3}})$ time on a one-port hypercube architecture. On a one-port hypercube architecture the second phase, which consists of two one-to-all broadcasts, takes twice as much time as the first phase. On a one-port hypercube the third phase, an all-to-one reduction of messages of size $\frac{n^2}{p^{2/3}}$, can be completed in the same amount of time as the first phase. On a multi-port hypercube the one-to-all broadcasts of A and B blocks in the second phase can occur in parallel and the data transmission times of each communication pattern can be reduced by a factor of $\log \sqrt[3]{p}$ (see Table 1). Each phase occurs over sets of $\sqrt[3]{p}$ processors and involves messages of size $\frac{n^2}{p^{2/3}}$. Hence, $n^2$ should be greater than or equal to $p^{2/3} \log \sqrt[3]{p}$ for multi-port hypercubes.

## 4.2 3-D All Approach

In this section we present another new algorithm designed for hypercube architectures. The algorithm presented in the previous section forms the basis of this algorithm. First we present an algorithm which assumes different initial distributions for matrices A and B (transpose of B aligned with A) and then in the following subsection present the variant which works with identically aligned matrices.

**Algorithm 3:** 3-D Diagonal

**Initial Distribution:** Processor $p_{i,i,k}$ contains $A_{ki}$ and $B_{ki}$

*Program of processor* $p_{i,j,k}$

If $(i = j)$ then
    Send $B_{ki}$ to $p_{i,k,k}$
    Broadcast $A_{ki}$ to all processors $p_{*,j,k}$
endif
If $(j = k)$ then
    Receive $B_{ji}$ from $p_{i,i,k}$
    Broadcast $B_{ji}$ to all processors $p_{i,j,*}$
endif
Receive $A_{kj}$ from $p_{j,j,k}$ and $B_{ji}$ from $p_{i,j,j}$
Calculate $I_{ki} = A_{kj} \times B_{ji}$
Send $I_{ki}$ to $p_{i,i,k}$
If $(i = j)$ then
    for $l = 0,\ \sqrt[3]{p} - 1$
        Receive $I_{k,i}$ from $p_{i,l,k}$
        $C_{k,i} = C_{k,i} + I_{k,i}$
    endfor
endif

Figure 7: 3-D Diagonal Algorithm

#### 4.2.1 3-D All_Trans Algorithm

This algorithm is essentially the 2-D Diagonal algorithm extended to the third dimension, where the columns (rows) of A (B) are mapped onto each column of processors perpendicular to the $z = 0$ plane (as opposed to only the diagonal columns). Consider a 3-D grid having $\sqrt[3]{p}$ processors along each dimension embedded into a hypercube, where $p \leq n^{3/2}$. Matrix A is partitioned into $\sqrt[3]{p} \times p^{2/3}$ blocks as shown in Figure 8, while B is partitioned into $p^{2/3} \times \sqrt[3]{p}$ blocks as shown in Figure 9. Each processor $p_{i,j,k}$ contains sub-blocks $A_{k,f(i,j)}$ and $B_{f(i,j),k}$, where $f(i,j)$ is defined as $(i \cdot \sqrt[3]{p} + j)$. We present an algorithm which computes $A \times B$ given this initial distribution. In this algorithm, the transpose of matrix B is initially identically distributed as matrix A.

The algorithm consists of three phases. In the first phase, each processor $p_{i,j,k}$ sends $B_{f(i,j),k}$ to $p_{k,j,k}$, i.e., each row of B is scattered along the $x$-direction in the $x$-$z$ plane it initially belongs. In the second

| | | | |
|---|---|---|---|
| $A_{0,f(0,0)}$ | $A_{0,f(0,1)}$ | $A_{0,f(1,0)}$ | $A_{0,f(1,1)}$ |
| $A_{1,f(0,0)}$ | $A_{1,f(0,1)}$ | $A_{1,f(1,0)}$ | $A_{1,f(1,1)}$ |

Figure 8: Partitioning of matrix A for 3-D All_Trans when $p = 8$.

| | |
|---|---|
| $B_{f(0,0),0}$ | $B_{f(0,0),1}$ |
| $B_{f(0,1),0}$ | $B_{f(0,1),1}$ |
| $B_{f(1,0),0}$ | $B_{f(1,0),1}$ |
| $B_{f(1,1),0}$ | $B_{f(1,1),1}$ |

Figure 9: Partitioning of matrix B for 3-D All_Trans when $p = 8$.

phase, all processors engage in an *all-to-all broadcast* of the sub-blocks of matrix A they contain, along the $x$-direction and processor $p_{k,j,k}$ engages in a *one-to-all broadcast* of the sub-blocks $B_{f(*,j),k}$, acquired in the first phase, along the $z$-direction. During the first two phases, each processor acquires $\sqrt[3]{p}$ sub-blocks of both the matrices A and B. Specifically, each processor $p_{i,j,k}$ acquires $B_{f(*,j),i}$ and $A_{k,f(*,j)}$. Hence each processor $p_{i,j,k}$ can compute $I_{k,i}$ where matrix I, the outer product computed by the plane $y = j$, is assumed symmetrically partitioned along rows and columns into $\sqrt[3]{p} \times \sqrt[3]{p}$ blocks. The last phase ensures that the result matrix C is obtained aligned in the same way as the source matrix A by reducing the corresponding blocks of the outer products by addition along the $y$-direction. Hence the last phase involves an *all-to-all reduction* along the $y$-direction. The first and second phases involve accumulation of $\sqrt[3]{p}$ blocks of submatrices of B and A respectively where each submatrix is of size $\frac{n^2}{p}$. Hence, space required for each processor is $2n^2 \sqrt[3]{p}$ words.

The first phase, being an all-to-one communication, the inverse of one-to-all personalized communication, along the $x$-direction, takes $t_s \log \sqrt[3]{p} + t_w \frac{n^2}{p^{2/3}}(1 - \frac{1}{\sqrt[3]{p}})$ time on a one-port hypercube. The second phase consists of a one-to-all broadcast of sub-blocks of B containing $\frac{n^2}{p^{2/3}}$ data elements, which takes $\log \sqrt[3]{p}(t_s + t_w \frac{n^2}{p^{2/3}})$ time and an all-to-all broadcast of sub-blocks of A containing $\frac{n^2}{p}$ data elements, which takes $t_s \log \sqrt[3]{p} + t_w \frac{n^2}{p^{2/3}}(1 - \frac{1}{\sqrt[3]{p}})$ time on a one-port hypercube. The last phase is an all-to-all reduction phase, which is the inverse of an all-to-all broadcast of messages of size $\frac{n^2}{p}$, and takes $t_s \log \sqrt[3]{p} + t_w \frac{n^2}{p^{2/3}}(1 - \frac{1}{\sqrt[3]{p}})$ time on a one-port hypercube. On a multi-port hypercube architecture the two broadcasts in the second phase can occur in parallel and the data transmission times can be reduced by a factor of $\log \sqrt[3]{p}$, the total number of communication links on every node along a virtual grid dimension, by using the techniques presented in [7] (see Table 1). On multi-ports, for each of the communication patterns the message size $M$ should be greater than or equal to $\log \sqrt[3]{p}$, the number of processors in each set involved in the communication patterns. The message size $\frac{n^2}{p}$ is the least for the last phase and hence, it suffices for $n^2$ to be greater than or equal to $p \log \sqrt[3]{p}$ for multi-port hypercubes.

### 4.2.2 The 3-D All Algorithm

One possible drawback of the 3-D All_Trans algorithm is that the initial distributions required for the matrices A and B are not identical. In this subsection, we present the 3-D All algorithm, a variant of the 3-D All_Trans algorithm, which starts with identical initial distributions of the matrices A and B and computes the result matrix C with even lower communication overhead.

Following the same notations as in the previous subsection, in the 3-D All algorithm each processor $p_{i,j,k}$ initially contains sub-blocks $A_{k,f(i,j)}$ and $B_{k,f(i,j)}$, with matrices A and B being partitioned identically, as shown in Figure 8. The main difference between the 3-D All_Trans algorithm and the 3-D All algorithm is in the first phase of the algorithm which requires proper movement of the data elements of matrix B. The first phase of the 3-D All algorithm consists of an *all-to-all personalized* communication of sub-blocks of B along the $y$-direction, where each processor $p_{i,j,k}$ transmits $B^l_{k,f(i,j)}$, the $l^{th}$ group of rows of $B_{k,f(i,j)}$, $0 \leq l < \sqrt[3]{p}$, to processor $p_{i,l,k}$. The only other difference is that in the second phase the newly acquired sub-blocks of B are *all-to-all broadcast* along the $z$-direction, as opposed to the *one-to-all broadcast* in the 3-D All_Trans algorithm. All other communication *and* computation steps are exactly the same as in the 3-D All_Trans algorithm. See Algorithm 5. The space requirement for this algorithm is the same as that for

**Algorithm 4:** 3-D All_Trans

**Initial Distribution:** Each processor $p_{i,j,k}$ contains $A_{k,f(i,j)}$ and $B_{f(i,j),k}$. See Fig. 8 & 9.

*Program of processor* $p_{i,j,k}$

```
Send B_{f(i,j),k} to p_{k,j,k}
If (i = k) then
  for l = 0, ∛p − 1
    Receive B_{f(l,j),k} from p_{l,j,k}
  Broadcast B_{f(*,j),k} along the z-direction to all processors p_{i,j,*}
endif
Broadcast A_{k,f(i,j)} along the x-direction to all processors p_{*,j,k}
Receive B_{f(*,j),i} from p_{i,j,i}
for l = 0, ∛p − 1
  Receive A_{k,f(l,j)} from p_{l,j,k}
Calculate I_{k,i} = Σ_{l=0}^{l=∛p−1} (A_{k,f(l,j)} × B_{f(l,j),i})
for l = 0, ∛p − 1
  Send I^l_{k,i} to p_{i,l,k}
    /* I^l_{k,i} is the l^{th} group of columns of I_{k,i} when
    I_{k,i} is split into ∛p groups by columns */
for l = 0, ∛p − 1
  Receive I^j_{k,i} from p_{i,l,k}
  C_{k,f(i,j)} = C_{k,f(i,j)} + I^j_{k,i}
endfor
```

Figure 10: 3-D All_Trans Algorithm

**Algorithm 5:** 3-D All

**Initial Distribution:** Each processor $p_{i,j,k}$ contains $A_{k,f(i,j)}$ and $B_{k,f(i,j)}$. See Figure 8.

*Program of processor* $p_{i,j,k}$

```
for l = 0, ∛p − 1
  Send B^l_{k,f(i,j)} to p_{i,l,k}
  /* B^l_{k,f(i,j)} is the l^th group of rows of B_{k,f(i,j)} */
endfor
for l = 0, ∛p − 1
  Receive B^j_{k,f(i,l)} from p_{i,l,k}
endfor
Broadcast B^j_{k,f(i,*)} along the z-direction to all processors p_{i,j,*}
Broadcast A_{k,f(i,j)} along the x-direction to all processors p_{*,j,k}
for m = 0, ∛p − 1
  Receive A_{k,f(m,j)} from p_{m,j,k}
  Receive B^j_{m,f(i,*)} from p_{i,j,m}
  /* B^j_{m,f(i,*)} is essentially B_{f(m,j),i} if B is visualized to be
  partitioned as in Figure 9. */
endfor
Calculate I_{k,i} = Σ_{m=0}^{m=∛p−1} (A_{k,f(m,j)} × B_{f(m,j),i})
for l = 0, ∛p − 1
  Send I^l_{k,i} to p_{i,l,k}
  /* I^l_{k,i} is the l^th group of columns of I_{k,i} when I_{k,i} is split
  into ∛p groups by columns */
for l = 0, ∛p − 1
  Receive I^j_{k,i} from p_{i,l,k}
  C_{k,f(i,j)} = C_{k,f(i,j)} + I^j_{k,i}
endfor
```

Figure 11: 3-D All Algorithm

the 3D All_Trans algorithm.

## *Proof of correctness*

Starting with the initial distribution with each processor $p_{i,j,k}$ containing $A_{k,f(i,j)}$ and $B_{k,f(i,j)}$, the first phase ensures that each processor $p_{i,j,k}$ gets $B^j_{k,f(i,l)}$ for all $0 \leq l < \sqrt[3]{p}$, where $B^j_{k,f(i,l)}$, as defined earlier, is the $j^{th}$ group of rows of $B_{k,f(i,l)}$ when it is partitioned into $\sqrt[3]{p}$ groups of rows. If B is visualized as partitioned into $p$ blocks as in Figure 9, then the set of data elements $B^j_{k,f(i,*)}$ is essentially $B_{f(k,j),i}$. The newly acquired blocks of the matrix B and the initial blocks of the matrix A are all-to-all broadcast along the $z$ and $x$ directions respectively in the second phase. Hence, by the end of the second phase, each processor $p_{i,j,k}$ acquires $B_{f(*,j),i}$ and $A_{k,f(*,j)}$. During the computation stage, a 2-D plane, $y = j$, calculates in a distributed fashion one of the outer products, I, corresponding to $A_{*,f(*,j)}$ and $B_{f(*,j),*}$. A processor $p_{i,j,k}$ calculates $I_{ki}$ where I is assumed symmetrically partitioned into $\sqrt[3]{p} \times \sqrt[3]{p}$ blocks as in Figure 1. There are $\sqrt[3]{p}$ such outer products calculated, one by each $x$-$z$ plane. It is easy to see that the block $I_{k,i}$ is the same as the group of sub-blocks $I_{k,f(i,*)}$ if I is visualized as partitioned into $p$ sub-blocks similar to the initial

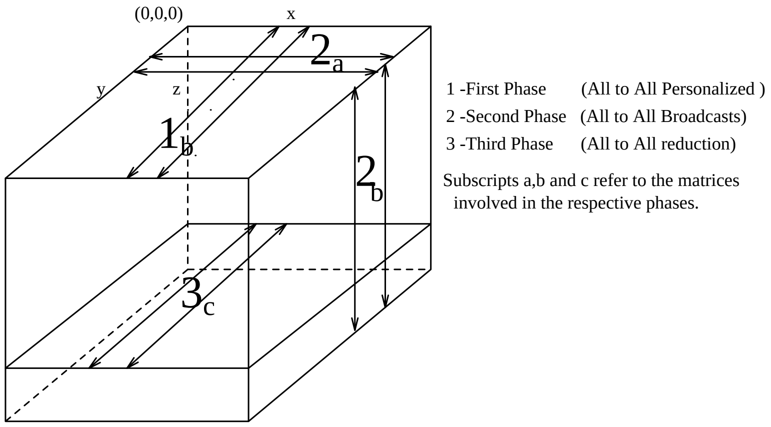


Figure 12: 3D All Algorithm

distribution of the matrix A (Figure 8). In the final reduction phase each processor $p_{i,j,k}$ needs to send $I_{k,f(i,l)}$ to processor $p_{i,l,k}$ for all $0 \le l < \sqrt[3]{p}$. Thus, each processor $p_{i,j,k}$ receives $I_{k,f(i,j)}$ from each $x$-$z$ plane and hence getting the required data elements from *all* of the $\sqrt[3]{p}$ outer products computed. □

The first phase of the 3-D All algorithm can be completed in $\log \sqrt[3]{p}(t_s + t_w \frac{n^2}{2p})$ time on a one-port hypercube and in $t_s \log \sqrt[3]{p} + t_w \frac{n^2}{2p}$ time (see Table 1) on a multi-port hypercube since it is an all-to-all personalized communication of messages of size $\frac{n^2}{p\sqrt[3]{p}}$ in a one-dimensional line of $\sqrt[3]{p}$ processors forming a subcube. The second phase now consists of two all-to-all broadcasts of messages of size $\frac{n^2}{p}$ along different dimensions, with each taking $t_s \log \sqrt[3]{p} + t_w \frac{n^2}{p^{2/3}}(1 - \frac{1}{\sqrt[3]{p}})$ time on a one-port hypercube. The third phase, being an all-to-all reduction phase, the reverse of all-to-all broadcasting of messages of size $\frac{n^2}{p}$, takes the same amount of time as an all-to-all broadcast in the second phase. On a multi-port hypercube the data transmission time can be reduced by a factor of $\log \sqrt[3]{p}$ by the techniques presented in [7] (see Table 1). Also, on a multi-port hypercube the two all-to-all broadcasts during the second phase can be overlapped. In this algorithm, the message size is the least for the first phase. Hence, it suffices for $\frac{n^2}{p\sqrt[3]{p}}$ to be greater than or equal to $\log \sqrt[3]{p}$ for full utilization of the bandwidth on multi-port hypercubes. When $\frac{n^2}{p\sqrt[3]{p}} < \log \sqrt[3]{p}$ but $\frac{n^2}{p} \ge \log \sqrt[3]{p}$, multiple ports can be used only for the second and third phases.

For a given matrix of size $n \times n$, the 3-D All algorithm can be applied on upto $n^{3/2}$ processors, since the maximum number of processors which can reside on an $x$-$y$ plane is $n$. A slight modification namely, mapping a 3-D grid of size $\sqrt[4]{p} \times \sqrt[4]{p} \times \sqrt{p}$ onto a $p$-processor hypercube, can allow us to use upto $n^2$ processors. Though the communication time reduces in terms of the number of start-ups required, the overall space requirement increases to $n^2\sqrt{p} + n^2\sqrt[3]{p}$.

# 5 Optimality Results

In this section, we state a lower bound result on the number of sequential data movement steps required to perform matrix multiplication on a machine. This result is a slight generalization of the lower bound result by Gentleman [10].

The function $f(k)$ is defined to be the maximum number of processors at which data originally available only at a single given processor can be made available in $k$ or fewer data movement steps. For a two-dimensional mesh, $f(k) = 2k^2 + 2k + 1$, while for a hypercube $f(k) = 2^k$.

| Algorithm | One-port Hypercubes | Multi-port Hypercubes | |
|---|---|---|---|
| | Communication overhead $(a,b)$ | Communication overhead $(a,b)$ | Conditions |
| Simple | $(\log p, 2\frac{n^2}{\sqrt{p}}(1-\frac{1}{\sqrt{p}})$ | $(\frac{1}{2}\log p, \frac{n^2}{\sqrt{p}\log\sqrt{p}}(1-\frac{1}{\sqrt{p}}))$ | $n^2 \geq p\log\sqrt{p}$ |
| Cannon | $(2(\sqrt{p}-1)+\log p,$ $\frac{n^2}{\sqrt{p}}(2-\frac{2}{\sqrt{p}}+\frac{\log p}{\sqrt{p}}))$ | $(\sqrt{p}-1+\frac{1}{2}\log p, \frac{n^2}{\sqrt{p}}(1-\frac{1}{\sqrt{p}}+\frac{\log p}{2\sqrt{p}}))$ | - |
| Ho *et al.* | - | $(\sqrt{p}-1+\frac{1}{2}\log p,$ $\frac{n^2}{\sqrt{p}}(\frac{2}{\log p}-\frac{2}{\sqrt{p}\log p}+\frac{\log p}{2\sqrt{p}}))$ | $n \geq \sqrt{p}\cdot\log\sqrt{p}$ |
| Berntsen | $(2(\sqrt[3]{p}-1)+\log p,$ $\frac{n^2}{p^{2/3}}(3(1-\frac{1}{\sqrt[3]{p}})+\frac{2\log p}{3\sqrt[3]{p}}))$ | $(\sqrt[3]{p}-1+\frac{2}{3}\log p,$ $\frac{n^2}{p^{2/3}}((1+\frac{3}{\log p})(1-\frac{1}{\sqrt[3]{p}})+\frac{\log p}{3\sqrt[3]{p}}))$ | $n^2 \geq p\log\sqrt[3]{p}$ |
| DNS | $(\frac{5}{3}\log p, \frac{n^2}{p^{2/3}}(\frac{5}{3}\log p))$ | $(\frac{4}{3}\log p, 4\frac{n^2}{p^{2/3}})$ | $n^2 \geq p^{2/3}\log\sqrt[3]{p}$ |
| 3DD | $(\frac{4}{3}\log p, \frac{n^2}{p^{2/3}}(\frac{4}{3}\log p))$ | $(\log p, 3\frac{n^2}{p^{2/3}})$ | $n^2 \geq p^{2/3}\log\sqrt[3]{p}$ |
| 3D All_Trans | $(\frac{4}{3}\log p, \frac{n^2}{p^{2/3}}(3(1-\frac{1}{\sqrt[3]{p}})+\frac{1}{3}\log p))$ | $(\log p, \frac{n^2}{p^{2/3}}(\frac{6}{\log p}(1-\frac{1}{\sqrt[3]{p}})+1))$ | $n^2 \geq p\log\sqrt[3]{p}$ |
| 3D All | $(\frac{4}{3}\log p, \frac{n^2}{p^{2/3}}(3(1-\frac{1}{\sqrt[3]{p}})+\frac{\log p}{6\sqrt[3]{p}}))$ | $(\log p, \frac{n^2}{p^{2/3}}(\frac{6}{\log p}(1-\frac{1}{\sqrt[3]{p}})+\frac{1}{2\sqrt[3]{p}}))$ | $n^2 \geq p^{4/3}\log\sqrt[3]{p}$ |
| | | $(\log p, \frac{n^2}{p^{2/3}}(\frac{6}{\log p}(1-\frac{1}{\sqrt[3]{p}})+\frac{\log p}{6\sqrt[3]{p}}))$ | $n^2 \geq p\log\sqrt[3]{p}$ |

Table 2: Communication overheads for various algorithms on hypercubes with one-port and multi-port architectures. Communication time for each entry is $t_s a + t_w b$.

| Algorithm | Conditions | Overall Space used |
|---|---|---|
| Simple | $p \leq n^2$ | $2n^2\sqrt{p}$ |
| Cannon | $p \leq n^2$ | $3n^2$ |
| Ho *et al.* | $p \leq n^2$ | $3n^2$ |
| Berntsen | $p \leq n^{3/2}$ | $2n^2 + n^2\sqrt[3]{p}$ |
| DNS | $p \leq n^3$ | $2n^2\sqrt[3]{p}$ |
| 3DD | $p \leq n^3$ | $2n^2\sqrt[3]{p}$ |
| 3D All_Trans | $p \leq n^{3/2}$ | $2n^2\sqrt[3]{p}$ |
| 3D All | $p \leq n^{3/2}$ | $2n^2\sqrt[3]{p}$ |

Table 3: Some **architecture independent** characteristics for various algorithms.

**Theorem 1**. *Under the assumption that each element of the given operand matrices is represented once and only once within the machine, and not using any broadcast facility, the multiplication of two $n \times n$ matrices A and B to produce the $n \times n$ product matrix C requires at least $k$ sequential data movement operations, where $f(2k)$ is greater than or equal to the number of processors initially storing the input matrices.*

*Proof.* Similar to the proof by Gentleman [10]. □

**Corollary 1**. *On a p-processor hypercube, the multiplication of two $n \times n$ matrices A and B requires at least $\frac{1}{2}\log(\frac{2n^2}{S})$ sequential message start-ups, where S is the maximum space available at each processor.*

*Proof.* Since $S$ is the maximum space available at each processor, the minimum number of processors initially storing the input matrices is $\frac{2n^2}{S}$. As $f(k) = 2^k$ for the hypercube, the minimum number of sequential data movement steps required to multiply A and B is $\frac{1}{2}\log\frac{2n^2}{S}$. □

As the function $f(k)$ is same for the one-port as well as multi-port hypercubes, Corollary 1 holds for both the hypercube architectures. When $S = 2\frac{n^2}{p^{2/3}}$, the minimum number of data movement steps required is $\frac{1}{3}\log p$. Hence, the DNS, 3DD, 3D All_Trans, and 3D All algorithms are optimal within a constant factor in the number of message start-ups, when the space available per processor is $O(\frac{n^2}{p^{2/3}})$. When $p = n^3$, DNS and 3DD algorithms have a data transmission overhead of $O(\log p)$, which is also optimal within a constant factor, under the space constraint, because each data movement step involves a data transfer of at least one word.

To our knowledge, there are no non-trivial lower bounds known for the data transmission overhead for distributed matrix multiplication algorithms. We derive here a lower bound of $O(max(\frac{n^3}{Sp}, \frac{n^2}{p}))$ for data transmission time on one-port distributed machines. This lower bound is independent of the network topology.

**Proposition 1:** *Consider a one-port distributed machine consisting of p processors, each having a local memory of S words. The data transmission time required for multiplying two $n \times n$ matrices on this machine is at least $O(max(\frac{n^3}{Sp}, \frac{n^2}{p}))$, under the assumption that the operand matrices have an identical initial distribution and each element of the operand matrices is represented once and only once within the machine.*

*Proof:* It is easy to derive this lower bound by observing that data transfer of one word can result in at most $min(S, n)$ multiplications, and matrix multiplication requires $O(n^3)$ multiplications. Initial distributions being identical, with no replications, can result in at most $((n/\sqrt{p})^3 p)$ multiplications without any data transfer. □

# 6 Analysis

In this section we analyze the performance of the algorithms presented in the previous two sections, for one-port hypercubes and multi-port hypercubes. The communication overheads and other characteristics of the algorithms have been summarized in Table 2 and Table 3. For multi-port hypercubes, the full bandwidth of the hypercube can be used by multi-port processors only if size of each message is greater than or equal to the number of communication links on any node along the dimension of the communication pattern. This imposes some conditions on the minimum size of the matrix required to be able to use all the links. Table 3 lists some architecture independent characteristics for the algorithms. We require there are no processors which are idle throughout the execution of the algorithm. For this purpose, the conditions listed in Table 3 need to be satisfied. The overall space used is the same for one-port hypercubes and multi-port hypercubes.

In our analysis, we compare the performances of Cannon, Berntsen, Ho-Johnsson-Edelman, 3DD and 3D All algorithms. Algorithm Simple has not been considered since it is the most inefficient algorithm with respect to the space requirement. From the tables, it can be easily seen that the 3DD and 3D All algorithms perform at least as well as the DNS and 3D All_Trans algorithms respectively, for both the architectures discussed, irrespective of the values of $n, p, t_s, t_w$. The results are based on analytical reasoning and statistics generated by a computer program on the basis of the expressions in Table 2. We present graphical results

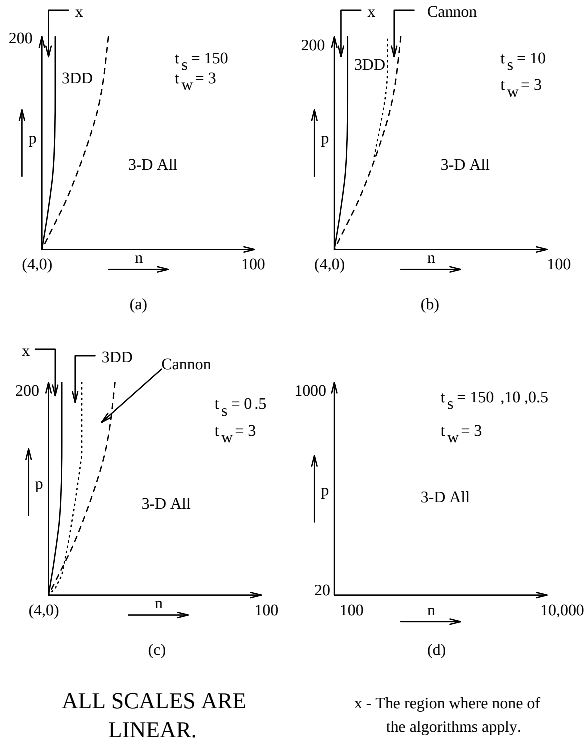


Figure 13: Performance analysis for one-port hypercubes

for three different sets of values of $t_s$ and $t_w$. In Fig. 13 and Fig. 14, each region of the parameter space is marked with the algorithm which performs the best in that range of $n$ and $p$.

## 6.1 Hypercubes with one-port processors

From the expressions of communication overheads for the various algorithms given in the Table 2, it is easy to see that the 3D All algorithm performs better than the 3DD, Berntsen's and Cannon's algorithms for all values of $p$ greater than or equal to 8, irrespective of the values of $n$, $t_s$ and $t_w$, wherever the 3D All algorithm is applicable. In the region $n^2 \geq p > n\sqrt{n}$, the 3DD algorithm should have less communication overhead than Cannon's Algorithm for large values of the ratio $\frac{t_s}{t_w}$.

The graphs in Figures 13 (a)–(d), generated by a computer program support our above analysis. The 3D All algorithm has the least communication overhead in the region $n^{3/2} \geq p$. In the region $n^2 \geq p > n^{3/2}$, the 3DD algorithm performs the best over the whole region for $t_s = 150, t_w = 3$ while for very small values of $t_s$, Cannon's algorithm performs better over most of the region. The 3DD is the only algorithm applicable in the region $n^3 \geq p > n^2$.

## 6.2 Hypercubes with multi-port processors

In case of multi-port hypercubes, the Ho-Johnsson-Edelman algorithm, wherever applicable, is better than Cannon's algorithm. From Table 2, we see that the 3D All algorithm will always performs better than the 3DD algorithm wherever both the algorithms are applicable. Similarly, the 3D All algorithm has better performance than Berntsen's algorithm for all values of $p$ greater than or equal to 8, independent of $n$, $t_s$ and $t_w$. The Ho-Johnsson-Edelman algorithm might perform better than the 3D All algorithm for very small

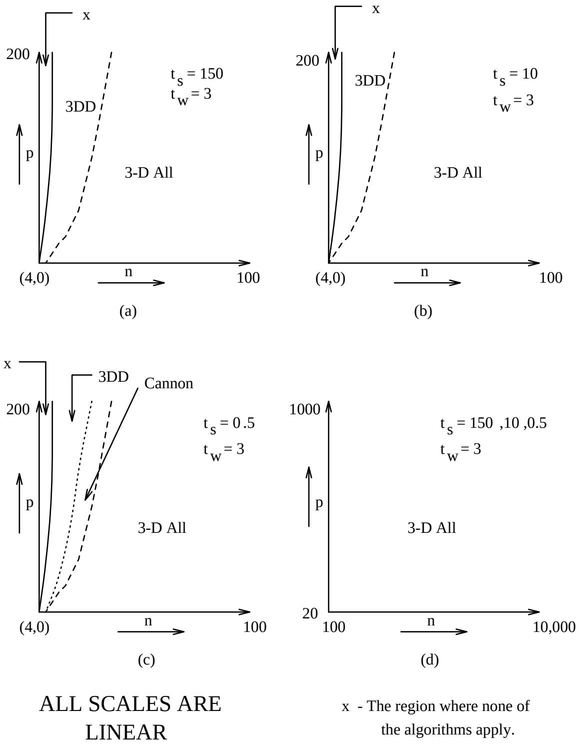


Figure 14: Performance analysis for multi-port hypercubes

values of $p$ when both are applicable, but 3D All should tend to be better for larger values of $p$ or $t_s$ because of the number of start-ups in the Ho-Johnson-Edelman algorithm being of O($\sqrt{p}$).

In Figures 14 (a)–(d) presented, we see that 3D All, wherever applicable, performs the best among the four algorithms. In the region $n^2 \geq p > n\sqrt{n}$, Cannon's algorithm has an edge over the 3DD algorithm for very small values of $t_s$.

# 7 Conclusion

In this paper we have analyzed most of the existing popular algorithms for dense matrix multiplication on hypercubes and designed two new algorithms. We showed that the newly proposed algorithms are optimal, within a constant factor, in the number of sequential data movement steps, when the space available per processor is $O(\frac{n^2}{p^{2/3}})$. We compared the communication overheads of the various algorithms on hypercubes with one-port processors and hypercubes with multi-port processors. One of the proposed algorithms, 3D ALL, has the least communication overhead whenever applicable for almost all values of $p$, $n$, $t_s$ and $t_w$ in the region $p \leq n\sqrt{n}$. In the region $n\sqrt{n} < p \leq n^3$ the other proposed algorithm, 3DD, performs the best for a major part of the region.

# References

[1] J. Berntsen, Communication efficient matrix multiplication on hypercubes, *Parallel Computing*, **12** (1989) 335–342.

[2] L. E. Cannon, A cellular computer to implement the Kalman Filter Algorithm, (Technical report, Ph.D. Thesis, Montana State University, 1969).

[3] E. Dekel, D. Nassimi, and S. Sahni, Parallel matrix and graph algorithms, *SIAM Journal of Computing*, **10** (1981) 657–673.

[4] G. C. Fox, S. W. Otto, and A. J. G. Hey, Matrix algorithms on a hypercube I: Matrix multiplication, *Parallel Computing*, **4** (1987) 17–31.

[5] A. Gupta and V. Kumar, Scalability of Parallel Algorithms for Matrix Multiplication, *Proceedings of the 1993 International Conference on Parallel Processing*, **3** 115–123.

[6] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation* (Prentice Hall, 1989).

[7] S. L. Johnsson and C. T. Ho, Optimum broadcasting and personalized communication in hypercubes, *IEEE Transactions on Computers*, **38**(9) (1989) 1249–1268.

[8] C. T. Ho, S. L. Johnsson and A. Edelman, Matrix multiplication on hypercubes using full bandwidth and constant storage, In *Proceeding of the Sixth Distributed Memory Computing Conference* (1991) 447–451.

[9] J. W. Demmel, M. T. Heath, and H. A. Van der Vorst, Parallel Linear Algebra, *Acta Numerica* **2** (Cambridge Press, New York, 1993).

[10] W. M. Gentleman, Some Complexity Results for Matrix Computations on Parallel Processors, *Journal of the ACM* **25** (1978) 112–115.

[11] H. Gupta and P. Sadayappan. Communication Efficient Matrix Multiplication on Hypercubes. In *Proceedings of the Sixth ACM Symposium on Parallel Algorithms and Architectures*, 320–329, 1994.