Analysis of Dense Matrix Algorithms

Pradeep Kumar Reddy Dasari Leela, Deepinder Sidhu[a*]

[a]Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County, Baltimore, MD 21250, USA

Abstract

Dense Matrix Algorithms, due to their regular structure, parallel computations involving matrices and vectors readily lend themselves to data-decomposition. Typical algorithms rely on input, output, or intermediate data decomposition. Discussed various algorithms and different dimensional matrices. In this paper I mentioned about communication models, algorithms and introduced a new algorithm. I would like to do performance analysis for one-port hypercubes and hypercubes with multi-port processors for dense matrix algorithms.

1. Introduction

Dense matrix multiplication is one of the central components of many scientific computations and is used in several applications. On a sequential computer, the standard method of multiplying two matrices of size n x n involves $O(n^3)$ floating point operations. The development of efficient algorithms for large distributed memory machines is of great interest because dense matrix multiplication is computationally costly. 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 matric multiplications [3,9].

One of the earliest distributed algorithms proposed for matrix multiplications was by Cannon [2] in 1969 for 2-D meshes. Ho, Johnsson, and Edleman 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 a number of processors.

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

2. Communication Models

The efficiency of the various algorithms proposed for hypercube architectures is examined in this paper. Throughout this paper, we ask a 2-array n-cube as a hypercube, and every one of the logarithms used are with reference to the bottom 2. Hypercube machines with one-port processor nodes and machines with multi-port processor nodes are also considered. A processor node in a one-port hypercube architecture can only use one communication connection (to send and receive) at a time, while in a multi-port architecture, a processor node can use all of its communication links at the same time.

The time it takes a processor node to transmit a message of m words to a neighboring processor node is calculated as $t_s + t_w m$, where t_s represents the message beginning cost and t_w represent the data transmission time per word. In this paper, many of the algorithms are implemented on a simulated 2-D or 3-D processor grid. Any collective communication pattern used in this paper's algorithm is along a one-dimensional chain of processors. Each one-dimensional chain of processors embedded in a simulated 2-D or 3-D grid is a smaller-dimensional hypercube [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 a one-to-all broadcast, a knowledge set of size M words is copied from one node to all or any other nodes. One-to-all personalized communication involves one node sending unique data of size M words to all or any 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 distinction between broadcasting and personalized communication is that data duplication aids information dissemination with the former, resulting in lower communication overhead than personalized communication. The reverse of the broadcasting operation is reduction, during which the info set is reduced by applying operators like addition/subtraction.

When communication is constrained to one port at a time (as in a one-port hypercube), spanning binomial tree (SBT) scheduling results in the shortest possible communication time. When using log N rotated spanning binomial trees simultaneously in multi-port hypercubes, optimal contact time is obtained by using log N rotated spanning binomial trees simultaneously, where N is the number of processors inside 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 utilized in this paper. Note that the message length M should be greater than or adequate to log N, the amount 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.

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.

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 x 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 pig has all the required sub-blocks of matrices A and B to compute the block $C_{i,j}$ 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 $t_s \log \sqrt{p} + t_w \frac{n^2}{\sqrt{p}}\left(1 - \frac{1}{\sqrt{p}}\right)$ time on a one-port hypercube and $t_s \log \sqrt{p} + t_w \frac{n^2}{\sqrt{p}\, log \sqrt{p}}\left(1 - \frac{1}{\sqrt{p}}\right)$ 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 multi-port hypercubes. This algorithm is very inefficient with respect to space as each processor uses $\frac{2n^2}{p}$ words of memory.

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 * 4 blocks.

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 $A_{i,(j+i)\text{mod}n}$ and $B_{(i+j)\text{mod}n,j}$. 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 $C_{i\,j}$ 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 $\sqrt{p}\left(t_s + t_w \frac{n^2}{p}\right)$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 most significant benefit of this algorithm is that it uses constant storage regardless of the number of processors.

3.3 Ho-Johnsson-Edelman Algorithm

On 2-D tori and hypercubes, the second step of Cannon's algorithm performs similarly. On hypercubes, it can be increased even further by using the whole bandwidth available, assuming that the sub-blocks of matrices A and B are large enough. Ho, Johnsson, and Edelman [8] suggested such a version. This algorithm is different from Cannon's only for multi-port hypercubes. Just a brief sketch of the algorithm is presented here, taken from [9]. Algorithm 1 can be found here.

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

On a virtual $\sqrt{p}$ x $\sqrt{p}$ 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 necessary amount of space is the same as 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.

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 n^{th} set of columns of A and the n^{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 $\sqrt[3]{p}$ x $\sqrt[3]{p}$ 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 $\frac{n}{p^{2/3}} x \frac{n}{\sqrt[3]{p}}$. 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 $t_s \log \sqrt[3]{p} + t_w \frac{n^2}{p^{2/3}} \left(1 - \frac{1}{\sqrt[3]{p}}\right)$ 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 $t_s \log \sqrt[3]{p} + t_w \frac{n^2}{p^{2/3}} \frac{1}{\log \sqrt[3]{p}} \left(1 - \frac{1}{\sqrt[3]{p}}\right)$. 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 Salmi 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 $\sqrt[3]{p} x \sqrt[3]{p} x \sqrt[3]{p}$ embedded into a hypercube of p processors. Initially matrices A and B are both mapped naturally, block partitioned, onto the z = 0 plane 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_{il} to $p_{i,j,j}$ and B_{ij}.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 $2 \log \sqrt[3]{p} \left(t_s + \frac{n^2}{p^{2/3}} t_w\right)$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 $\log \sqrt[3]{p} \left(t_s + \frac{n^2}{p^{2/3}} t_w\right)$ time on a one-port hypercube and $\log \sqrt[3]{p} \left(t_s + \frac{n^2}{p^{2}/3} t_w\right)$ 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 $\sqrt[3]{s}$ x $\sqrt[3]{s}$ x $\sqrt[3]{s}$ 3-D grid of supernodes where each supernode is a square mesh of $\sqrt{r}$ x $\sqrt{r}$ 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 Algorithm

A new algorithm designed for hypercubes have been presented in this section. We present them in various stages to present the rationale behind the algorithms. 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 3D grid.

4.1 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 × B_ji
Send I_ki to p_{i,i,k}
If (i = j) then
    for l = 0, ∛p − 1
        Receive I_{k,i} from p_{i,l,k}
        C_{k,i} = C_{k,i} + I_{k,i}
    endfor
endif
```

Figure 3: 3-D Diagonal Algorithm

4.1.1 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 $\sqrt[3]{p}$ x $p^{2/3}$ blocks as in the Figure 4, while B is partitioned into $p^{2/3}$ x $\sqrt[3]{p}$ blocks as in figure 5. Each processor $p_{i,j,k}$ contains subblocks $A_{k,f(i,j)}$ and $B_{f(i,j),k,}$ where f(i,j) is defined as $(i + \sqrt[3]{p} + j)$. We present an algorithm which computes A x 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 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 $p_{i,j,k}$ acquired $B_{f(*,j),I}$ and $A_{k,f(*,j)}$. Hence each processor $p_{i,j,k}$ can compute $I_{k,I}$ where matrix I, the other product computed by the plane y = j, is assumed symmetrically partitioned along rows and columns into $\sqrt[3]{p}\ x\ \sqrt[3]{p}$ blocks. The last phase ensures that the result matrix C is obtained aligned in the same way as the source matric A by reducing the corresponding blocks of the outer products by addition along the y-direction. Hence the last phase involves an al-to-all reduction along the y-direction. The first and second phases involve accumulation 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.

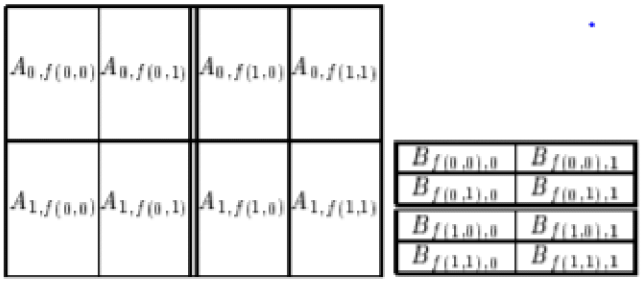


Figure 4: Partitioning of matrix A for 3-D All_Trans when p = 8

Figure 5: Partitioning of matrix B for 3-D All_Trans when p = 8

4.1.2 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 4. 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 $B_{k,i(i,j)}$, $0 \leq l < \sqrt[3]{p}$, 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 the 3-D All_Trans algorithm.

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 $l = 0, \sqrt[3]{p} - 1$
    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 $l = 0, \sqrt[3]{p} - 1$
  Receive $A_{k,f(l,j)}$ from $p_{l,j,k}$
Calculate $I_{k,i} = \sum_{l=0}^{l=\sqrt[3]{p}-1} (A_{k,f(l,j)} \times B_{f(l,j),i})$
for $l = 0, \sqrt[3]{p} - 1$
  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 $\sqrt[3]{p}$ groups by columns */
for $l = 0, \sqrt[3]{p} - 1$
  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 6: 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 $l = 0, \sqrt[3]{p} - 1$
  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 $l = 0, \sqrt[3]{p} - 1$
  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 $m = 0, \sqrt[3]{p} - 1$
  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 $I_{k,i} = \sum_{m=0}^{m=\sqrt[3]{p}-1} (A_{k,f(m,j)} \times B_{f(m,j),i})$
for $l = 0, \sqrt[3]{p} - 1$
  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 $\sqrt[3]{p}$ groups by columns */
for $l = 0, \sqrt[3]{p} - 1$
  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 7: 3-D All Algorithm

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 hound result by Gentleman [10].

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.

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.

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 um 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 my analysis. we compare the performances of Cannon, Berntsen. Ho-Johnson-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 for three different sets of values of t_s and t_w. In figure 6 and figure 7, 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 1 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 value 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 6 (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 perform better than the 3DD algorithm wherever both the algorithms arc 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 values of p when both are applicable, but 3D All should tend to be better for larger values of p or t, because of the number of start-ups in the Ho-Johnson-Edelman algorithm being of $O(\sqrt{p})$.

In Figures 7 (a) (d) praN4ented, 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.

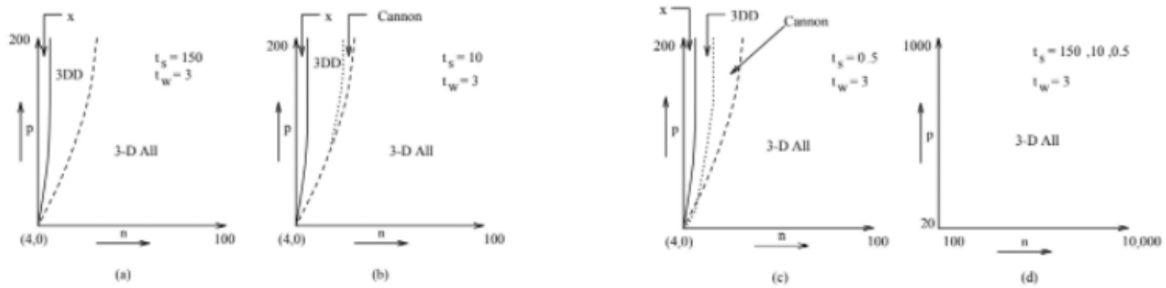
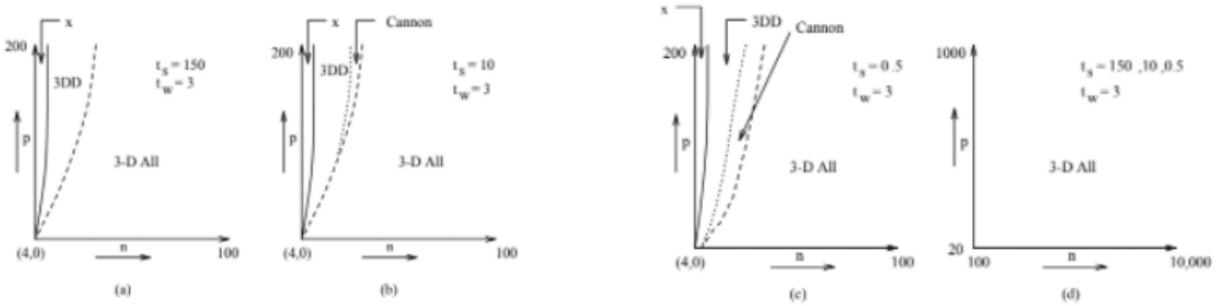


Figure 6: Performance analysis for one-port hypercubes



ALL SCALES ARE LINEAR x- The region where none of the algorithms apply.

Figure 8: Performance analysis for multi-port hypercubes

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 $O\left(\frac{n^2}{p^{2/3}}\right)$. 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 $p < n\sqrt{n}$. 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.

8. 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 Kahnan 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 1: Matrix multiplication. Parallel Computing, 4 (1987) 17-31.

[5] A. Gupta and V. Kmnar, 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 mid constant storage, in Proceeding of the Sixth Distributed Memory Computing Conference (1991) 447-451.

[9] J. W. Demme', M. T. Heath, and H. A. Van der Vonit. Parallel Linear Algebra, Acta Numerics 2 (Cambridge Press. New York, 1993).

[10] W. M. Gentleman. Some Complexity Results for Matrix Computations on Parallel Processors, Journal of the ACM25 (1978) 112 115.

[11] H. Gupta and P. Sadayappati. Communication Efficient Matrix Multiplication on Hypercubes. in Proceedings of the Sixth ACM Symposium. on Parallel Algorithms and Architectures, 320-329. 1994.