SIMD Algorithms for Matrix Multiplication on the Hypercube

C. A. Alonso Sanches* and S. W. Song†

University of São Paulo - IME
Department of Computer Science
C.P. 20570 - São Paulo, SP – 01452-990 – Brazil

Abstract

In this paper we present a SIMD algorithm for $n \times n$ matrix multiplication on a hypercube of p processors, with time complexity of $O(\frac{n^2}{p^{2/3}} \log p + \frac{n^\lambda}{p^{\lambda/3}})$, $1 \leq p \leq n^3$ and $2 \leq \lambda < 3$. For $1 \leq p \leq n^3/\log^3 n$ the new algorithm is better than the algorithm of Dekel, Nassimi and Sahni. The algorithm is derived by using the matricial visualization of the hypercube, suggested by Nassimi and Sahni.

1 Introduction

In this paper we consider SIMD algorithms for parallel machines with processors interconnected as a hypercube of dimension d. The hypercube interconnection network is used in many parallel architectures. It is not easy to conceive parallel algorithms for such distributed memory machines that fully exploit their parallelism. In [4, 7, 8] the hypercube is viewed as a matrix whose rows and columns are formed by subhypercubes. This *matricial visualization*, together with a series of properties derived from the recursive definition of the hypercube, constitutes a useful instrument to design parallel algorithms for machines of such architecture.

For the problem of multiplication of $n \times n$ matrices on a hypercube of p processors, Dekel, Nassimi and Sahni [4] present algorithms with time complexity of $O(n^\lambda/p^{\frac{\lambda-1}{2}})$, where $2 \leq \lambda < 3$ and $1 \leq p \leq n^2$, and $O(\log \frac{p}{n^2} + \frac{n^3}{p})$, with $n^2 \leq p \leq n^3$. By using the matricial visualization we develop a new algorithm, called MMM_1, which is $O(\frac{n^2}{p^{2/3}} \log p + \frac{n^\lambda}{p^{\lambda/3}})$, with $1 \leq p \leq n^3$. It can be shown that MMM_1 is better for $1 \leq p \leq n^3/\log^3 n$.

In this section we present some basic communication operations, and propose a notation to simplify the algorithm description. In Section 2, we review the matricial visualization of the hypercube and give some of its properties. In [12] some algorithms presented in [7, 8] are described using the basic operations and the properties of the matricial visualization. It can be seen that such descriptions are simpler than the original ones. In Section 3, we present the SIMD algorithms for the basic operations on the hypercube, together with their time complexities. Section 4 is the main section of this paper. We first review some known algorithms for matrix multiplication on the hypercube. We propose a new algorithm and derive its complexity. Finally in Section 5 we give concluding remarks.

1.1 Basic communication operations

Communication between processors in a distributed memory system obviously depends on the interconnection topology. Some basic operations appear frequently in the design of parallel algorithms. Here we will adopt the terminology of [2]. We need the following operations in this paper.

Single-node Broadcast: One message is sent from one node to all the other nodes of the network. From now on we use the name BROADCAST for this operation.

Multi-node Broadcast: Each node sends a same message to all the other nodes. We will refer to this operation by the name MULTI-BR.

Single-node Accumulation: A certain node receives a message from each one of the other nodes. In each intermediate node along the route, the

*Supported by FAPESP - Proc. 91/1646-0, CNPq and currently CAPES.

†Supported by FAPESP - Proc. No. 93/0603-1 and CNPq - Proc. No. 306063/88-3 and PROTEM-CC/SP.

received messages are combined, giving rise to a single message of the same type and size. The combining time is not superior than the transmission time between two adjacent nodes. This operation corresponds to the inverse of the broadcast operation and will be referred to as ACCUM.

1.1.1 Notation for SIMD algorithms

A hypercube $H(d)$ of dimension d possesses $p = 2^d$ nodes. Each node is denoted by a binary address of d bits. Node i is adjacent to all nodes whose addresses differ from i in exactly one bit position. Each node of a hypercube of p nodes is adjacent to $d = \log p$ other nodes. We use the following notation based on Nassimi and Sahni [7, 8].

- $R(i)$, $0 \leq i < p$, denotes a register R at node i of $H(d)$. When register R of node i is empty, $R(i)$ = null. If R is a vector of registers, $R[j](i)$ is the j-th position of vector R of node i.

- i_b denotes the b-th bit of the binary representation of i.

- $i_{b_2:b_1}$ corresponds to bits $b_2, \ldots, b_1$ of the binary representation of i.

- Let $i_{d-1} \ldots i_0$ be the binary representation of i, for $i \in [0, p-1]$. $i^{(b)}$ is the number whose binary representation is $i_{d-1} \ldots i_{b+1} \bar{i_b} i_{b-1} \ldots i_0$, where $\bar{i_b}$ is the complement of i_b, $0 \leq b < d$. In the hypercube, node i is connected to $i^{(b)}$, $0 \leq b < d$.

An assignment statement involving two registers on the hypercube assigns the value of one register to another register. We consider the following types of assignment.

$:=$ Used between registers of a same node.

$\leftarrow$ This assignment envolves two adjacent nodes. For example, $R(i) \leftarrow R(i^{(b)})$ is a valid assignment on the hypercube.

$\leftrightarrow$ Exchange operation between two adjacent nodes, for example, $R(i) \leftrightarrow R(i^{(b)})$.

For SIMD machines, the algorithm is the same in each processor and a command can be executed or not by any given processor. A condition expressed between parenthesis enables execution of any given command. For example:

$R(i) \leftarrow R(i^{(b)}) + 1, \ (i_4 = 1)$

where the assignment is only executed for nodes whose addresses have a 1 in bit 4.

2 Visualizing a hypercube

In [7, 8, 4], the hypercube is viewed as a number of sub-hypercubes arranged in a certain array. In the 2-dimensional matricial visualization, the nodes of $H(d)$ are considered as a matrix of $L = 2^m$ rows and $C = 2^n$ columns, with $p = 2^d = 2^{m+n} = LC$. As an example, Figure 1 illustrates $H(6)$, with $m = n = 3$. The node addresses of this example are shown in the following table.

000 000	000 001	000 010	000 011	000 100	000 101	000 110	000 111
001 000	001 001	001 010	001 011	001 100	001 101	001 110	001 111
010 000	010 001	010 010	010 011	010 100	010 101	010 110	010 111
011 000	011 001	011 010	011 011	011 100	011 101	011 110	011 111
100 000	100 001	100 010	100 011	100 100	100 101	100 110	100 111
101 000	101 001	101 010	101 011	101 100	101 101	101 110	101 111
110 000	110 001	110 010	110 011	110 100	110 101	110 110	110 111
111 000	111 001	111 010	111 011	111 100	111 101	111 110	111 111

Considering the hypercube as a matrix $P = (p_{i,j})$, $0 \leq i < L$ and $0 \leq j < C$, the node corresponding to position $p_{0,0}$ of the matrix has address 0, $p_{0,C-1}$ is $C-1$, $p_{1,0}$ is C, and so on. The address of each node has $d = n + m$ bits.

In each row, the most significant m bits remain constant, while the n less significant ones increase from 0 to $C-1$. Each row can thus be considered as a $H(n)$ arranged in a linear fashion.

On the other hand, the same occurs in each column, where the n least significant bits are constant and the m most significant bits increase from 0 to $L-1$. Therefore, each column can also be considered as a $H(m)$ arranged linearly.

Furthermore, we can consider the matrix P as formed by sub-matrices $P_{i,j}$ of order $2^{m-m'} \times 2^{n-n'}$, where $0 \leq m' \leq m$ and $0 \leq n' \leq n$, for $0 \leq i < 2^{m'}$ and $0 \leq j < 2^{n'}$, as shown in Figure 2.

Given k matrices $M_0, M_1, \ldots, M_{k-1}$ of the same order, we denote by *corresponding elements* each group of k elements, one in each matrix, that possess the same indices. The sub-matrices of P on a same row will be named $P_{i,0}$, $P_{i,1}$, $\ldots$, $P_{i,2^{n'}-1}$. The sub-matrices of P on a same column will be named $P_{0,j}, P_{1,j}, \ldots, P_{2^{m'}-1,j}$.

- Each sub-matrix $P_{i,j}$ corresponds to a $H(n+m-n'-m')$ also visualized in matricial form.

- Each group of corresponding elements of the sub-matrices of P of a same row form a $H(n')$. Figure 3 illustrates this property, where the arcs connect nodes of the same group.

- Similarly, each group of corresponding elements of the sub-matrices of P of a same column form a $H(m')$.

3 Basic communication operations

We consider only the linear dependence of communication time on the message size. In other words, we ignore the setup time for communication. The difference between *half-duplex* or *full-duplex* channels is not important, as far as order of magnitude is concerned, since this corresponds to a factor of 2 in time. On the other hand, the difference between d-port and 1-port is significant. For machines with the hypercube architecture $H(d)$, this corresponds to a factor of $d = \log p$ in the communication time. We describe some SIMD algorithms that use the 1-port hypothesis. Other assumptions made include the following.

- Messages sent from one node to another are of unit size.
- The communication time is the same between any pair of adjacent processors.
- Message transmissions between processors are error-free.
- The time of combining messages, considered in accumulation operations, is not superior to the communication time between the two nodes, and the resulting message has the same size of the initial messages.

4 Matrix multiplication on the hypercube

Our objective is to describe the matrix multiplication algorithms in a systematic way, by using the several visualizations. The number of distinct algorithms is large, due to the following reasons: there are several matricial visualizations of the hypercube that allow the development of algorithms; we can assume different initial storage of the matrix elements; the majority of the algorithms are based on sub-matrix multiplication. For this purpose, we use algorithms that solve the same problem. This gives rise to many possible combinations.

We group the algorithms that originate from the same visualzation into classes. The name of each class will enable us to associate with the respective visualization, as well as the number of processors used.

The usage of basic communication operations simplifies the algorithm description without affecting the time complexity. A parameter indicating a set of bits is used to determine the sub-cubes considered in each case, according to Section 2. This parameter is represented by the vector *BITS*. This allows the usage of these algorithms as subroutines of other algorihtms. In order to avoid overloading of the notation, we omit the node index used in the parameters and variables.

The usage of a basic operation of $O(x)$ with messages of unit size will spend time $O(tx)$ with messages of size t. The subscritp of each processor p will always refer to the visualization being considered. Thus two subscripts are used (for example $p_{i,j}$) in bi-dimensional matricial visualization.

4.1 Matrix multiplication

Given two $n \times n$ matrices $\mathcal{A}$ and $\mathcal{B}$ where $\mathcal{A}= (a_{i,j})$ and $\mathcal{B}= (b_{i,j})$, $0 \leq i,j < n$, conveniently stored in a hypercube $H(d)$ with $p = 2^d$ processors, we wish to compute the product $\mathcal{C} = \mathcal{AB}$. The final layout of matrix $\mathcal{C}$ in $H(d)$ will be analogous to the initial layout of $\mathcal{A}$ and $\mathcal{B}$.

Assume that each element of the matrices has unit size. The transmission of one element of $\mathcal{A}$, $\mathcal{B}$ or $\mathcal{C}$ takes constant time and the transmission of t elements takes $O(t)$ time.

There exist several sequential algorithms for matrix multiplication [3, 9, 14] that take $o(n^3)$ time[1], that is $O(n^\lambda)$ with $2 \leq \lambda < 3$.

4.2 Basic parallel algorithms on the hypercube

There exist in the literature several parallel algorithms for multiplication of $n \times n$ matrices on a hypercube of p processors, with $p = n$, n^2 or n^3. These algorithms have time complexity of $O(n^2)$, $O(n)$ and $O(\log n)$, respectively, and will be denoted *basic* algorithms. They belong to the classes named as N, NN and NNN. Indices will be used to denote different algorithms of a same class. For example, NN_1 and NN_2 belong to class NN.

4.2.1 Class N: with $p = n$ processors

The hypercube is viewed as a $1 \times n$ matrix. Two initial storage schemes are possible: by row or by column. We suppose that each processor p_j, $0 \leq j < n$, stores a column of matrix $\mathcal{A}$ and matrix $\mathcal{B}$, that is,

[1]Given two functions $f(n)$ and $g(n)$, we say that $f(n) = o(g(n))$ if $\lim_{n\to+\infty} f(n)/g(n)$ exists and is equal to 0.

$a_{0,j}, \ldots, a_{n-1,j}$ and $b_{0,j}, \ldots, b_{n-1,j}$ are in p_j. The following algorithm, called N_1, calculates the product $\mathcal{C}$, storing each column in a processor. Each processor p_j, $0 \leq j < n$, executes:

for $i := 0$ to $n-1$ do
  MULTI-BR of $a_{i,j}$
  p_j calculates the value of $c_{i,j}$

Both steps of the loop spend $O(n)$ time and the algorithm has time complexity of $O(n^2)$. The needed space in each processor is also $O(n)$.

Each processor contains two vectors A and B with n entries. Initially, in each p_j, $0 \leq j < n$, $A[i] = a_{i,j}$ and $B[i] = b_{i,j}$, with $0 \leq i < n$. As seen before, $A[i](j)$ denotes element $A[i]$ of processor p_j. The result will stay in a third vector C, also with n entries. F is also a vector of size n, and $BITS = \{0, \ldots, \log_2 n - 1\}$. We can now formalize algorithm N_1 in each processor p_j.

N$_1$(*BITS, A, B, C)*
  for i:=0 to $n-1$ do
    $F(j) := \emptyset$
    MULTI-BR($BITS, A[i](j), F(j)$)
    $C[i](j) := \sum_{i=0}^{n-1} F[i](j) * B[i](j)$

There exists an analogous N_2 algorithm for the case in which the matrices are stored by row.

4.2.2 Class NN: with $p = n^2$ processors

Algorithm NN_1 [2] assumes that each processor stores an element of each matrix. Thus $a_{i,j}$ and $b_{i,j}$ are in $p_{i,j}$, $0 \leq i, j < n$.

This arrangement takes advantage of one of the properties of matricial visualization: the rows and columns correspond to sub-hypercubes of n processors. The algorithm consists of only three steps:

- MULTI-BR of $a_{i,j}$ in each row i;
- MULTI-BR of $b_{i,j}$ in each column j;
- $p_{i,j}$ computes the value of $c_{i,j}$

Each step takes $O(n)$ time, which is the final complexity. Each element of matrix $\mathcal{C}$ stays in one processor, in the same way as matrices $\mathcal{A}$ and $\mathcal{B}$. The space needed in each processor is $O(n)$.

In the formalization of this algorithm, registers *COL* and *LIN* store the elements of *BITS* that correspond respectively to the sub-hypercubes of the columns and rows. Each processor has also two registers A and B. Initially, in each $p_{i,j}$, $A = a_{i,j}$ and $B = b_{i,j}$. The result will stay in a third register C. In this algorithm, $BITS = \{0, \ldots, 2\log_2 n - 1\}$. We show below the algorithm in each processor p_j.

NN$_1$(*BITS, A, B, C)*
  $q := |BITS|/2$
  $LIN(j) := \{BITS[0], \ldots, BITS[q-1]\}$
  $COL(j) := \{BITS[q], \ldots, BITS[2q-1]\}$
  $F_1(j) := \emptyset$
  MULTI-BR($LIN(j), A(j), F_1(j)$)
  $F_2(j) := \emptyset$
  MULTI-BR($COL(j), B(j), F_2(j)$)
  $C(j) := \sum_{i=0}^{n-1} F_1[i](j) * F_2[i](j)$

In [4] there is another algorithm (which we call NN_2) which also uses n^2 processors and has the same complexity, taking however $O(\log n)$ space in each processor.

4.2.3 Class NNN: with $p = n^3$ processors

Algorithm NNN_1 is described in [4]. The processors are arranged in an $n \times n \times n$ layout, and the matrices $\mathcal{A}$ and $\mathcal{B}$ are stored on face $k = 0$, one element per processor. The initial arrangement on this face is identical to that of n^2 processors of class NN.

Each processor $p_{i,j,k}$ has registers A, B, A', B' and C. Denoting by $R_{i,j,k}$ the value of register R of $p_{i,j,k}$, we have initially $A_{i,j,0} = a_{i,j}$ and $B_{i,j,0} = b_{i,j}$. We have the following steps:

- From $A_{i,j,0} = a_{i,j}$, BROADCAST along axis k resulting in $A_{i,j,k} = a_{i,j}$, $\forall k$;
- From $B_{i,j,0} = b_{i,j}$, BROADCAST along axis k resulting in $B_{i,j,k} = b_{i,j}$, $\forall k$;
- From $A_{i,k,k} = a_{i,k}$, BROADCAST along axis j resulting in $A'_{i,j,k} = a_{i,k}$, $\forall j$;
- From $B_{k,j,k} = b_{k,j}$, BROADCAST along axis i resulting in $B'_{i,j,k} = b_{k,j}$, $\forall i$;
- Each $p_{i,j,k}$ computes $A'_{i,j,k} B'_{i,j,k} = a_{i,k} b_{k,j}$;
- ACCUM along axis k for $p_{i,j,0}$ of the sum of the computed values, obtaining $c_{i,j}$.

With the exception of the last step which takes constant time, all the others utilize operations of $O(\log n)$, which is the final complexity of the algorithm. Constant space is needed in each processor.

Let us formalize the algorithm: in each $p_{i,j,0}$, we have initially $A = a_{i,j}$ and $B = b_{i,j}$. In the other processors, $A = B =$ null. In all the processors, we have initially $A' = B' =$ null. In this algorithm, $BITS = \{0, \ldots, 3\log_2 n - 1\}$, and I, J and K are vectors that correspond to the bits determining the sub-cubes in each axis.

NNN$_1$(*BITS, A, B, C)*
$q := |BITS|/3$
$I(j) := \{BITS[0], \ldots, BITS[q-1]\}$
$J(j) := \{BITS[q], \ldots, BITS[2q-1]\}$
$K(j) := \{BITS[2q], \ldots, BITS[3q-1]\}$
$BROADCAST(K(j), A(j))$
$BROADCAST(K(j), B(j))$
$A'(j) := A(j), \ (j_{q:2q-1} = j_{2q:3q-1})$
$BROADCAST(J(j), A'(j))$
$B'(j) := B(j), \ (j_{0:q-1} = j_{2q:3q-1})$
$BROADCAST(I(j), B'(j))$
$A'(j) := A'(j) * B'(j)$
$SUM(K(j), A'(j), C(j))$

4.3 Algorithms of Dekel, Nassimi and Sahni

Algorithms from NN and NNN are used as sub-algorithms in [4]. They propose two algorithms covering the cases where the number of processors varies from 1 to n^3. These algorithms belong to classes NNM and MM, in our terminology.

4.3.1 Class NNM: with $n^2 \leq p \leq n^3$ processors

In [4] $p = n^2m$ processors are arranged in a $n \times n \times m$ matrix, where n and m are powers of 2 such that $1 \leq m \leq n$. Initially, the matrices $\mathcal{A}$ and $\mathcal{B}$ are stored on the first $n \times n$ face, as in class NNN.

The basic idea of the algorithm is to treat $\mathcal{A}$ and $\mathcal{B}$ as $m \times m$ matrices, whose elements are $\frac{n}{m} \times \frac{n}{m}$ sub-matrices:

$$\mathcal{A} = (\mathcal{A}_{i,j}) \text{ and } \mathcal{B} = (\mathcal{B}_{i,j}),$$

where $\mathcal{A}_{i,j}$ and $\mathcal{B}_{i,j}$ are sub-matrices of order n/m, with $0 \leq i, j < m$.

In this way, $\mathcal{C} = (\mathcal{C}_{i,j})$, where $\mathcal{C}_{i,j} = \sum_{k=0}^{m-1} \mathcal{A}_{i,k}\mathcal{B}_{k,j}$. An important fact is that, as m is a power of 2, $\mathcal{A}_{i,j}$ and $\mathcal{B}_{i,j}$ are stored in a sub-cube with $(n/m)^2$ processors, by one of the properties of the matricial visualization.

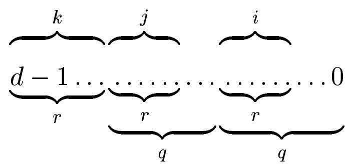
Figure 4 visualizes the hypercube under the $n \times n \times m$ arrangement, and shows the initial storage of the sub-matrices. The dotted squares correspond to processors.

Algorithm NNM_1 presented in [4] consists of the steps of NNN_1, and utilize NN_2 as sub-algorithm to obtain the sub-matrix product:

- Apply the four BROADCAST operations described in NNN_1, considering however the sub-matrices $\mathcal{A}_{i,j}$ and $\mathcal{B}_{i,j}$ as elements. The time needed is $O(\log m)$, due to the property of the *corresponding elements* of the matricial visualization.
- At the end of the previous step, each pair of sub-matrices $\mathcal{A}_{i,k}$ and $\mathcal{B}_{k,j}$ will be in a same group of $(n/m)^2$ processors. The product of these sub-matrices can be done in $O(n/m)$ time using NN_2. (As a matter of fact, NN_1 can equally be used.)
- The needed summation to compute each sub-matrix $\mathcal{C}_{i,j}$ is done by a ACCUM operation on the k axis, according to the description of NNN_1. The needed time is also $O(\log m)$.

Therefore, algorithm NNM_1 spends the total time of $O(\log m + n/m)$, or, considering $p = n^2m$, $O(\log \frac{p}{n^2} + \frac{n^3}{p})$.

To formalize this algorithm, it is necessary to see which of the d bits of the address of each processor are related to the directions i, j and k. Considering $q = \log_2 n$ and $r = \log_2 m$, the following figure shows which ones vary along the three direction, among the d bits of the processors of each group.



Therefore, it is necessary to pass the value of $r = \log_2 m$ as parameter, in order to divide the bits. Furthermore, we have $BITS = \{0, \ldots, 2\log_2 n + \log_2 m - 1\}$.

NNM$_1$(*BITS, r, A, B, C)*
$q := (|BITS| - r)/2$
$I(j) := \{BITS[q-r], \ldots, BITS[q-1]\}$
$J(j) := \{BITS[2q-r], \ldots, BITS[2q-1]\}$
$K(j) := \{BITS[2q], \ldots, BITS[2q+r-1]\}$
$BROADCAST(K(j), A(j))$
$BROADCAST(K(j), B(j))$
$A'(j) := A(j), \ (j_{2q-r:2q-1} = j_{2q:2q+r-1})$
$BROADCAST(J(j), A'(j))$
$B'(j) := B(j), \ (j_{q-r:q-1} = j_{2q:2q+r-1})$

$BROADCAST(I(j), B'(j))$
$BITS2 := \{0, \ldots, q-r-1, q, \ldots, 2q-r-1\}$
$NN_2(BITS2, A', B', C)$
$A'(j) := C(j)$
$SUM(K(j), A'(j), C(j))$

If we use NN_1 as sub-algorithm, we would have a new algorithm NNM_2 with the same time complexity.

4.3.2 Class MM: with $1 \leq p \leq n^2$ processors

The same idea described in the previous class is again used here, considering however $p = m^2$, where $1 \leq m \leq n$, and applying the steps of an algorithm of class NN and treating the sub-matrices as elements.

Initially, each processor $p_{i,j}$ stores the sub-matrices $\mathcal{A}_{i,j}$ and $\mathcal{B}_{i,j}$, $0 \leq i, j < m$, and thus contains $2(n/m)^2$ elements (Figure 5).

Based on NN_1, we can apply its steps the the sub-matrices of $\mathcal{A}$ and $\mathcal{B}$, obtaining thus algorithm MM_1:

- MULTI-BR of $\mathcal{A}_{i,j}$ along each row i. The needed time is $O(m\frac{n^2}{m^2}) = O(n^2/m)$.
- MULTI-BR of $\mathcal{B}_{i,j}$ along each column j. The needed time is the same as the previous step, $O(n^2/m)$.
- m multiplications and $m-1$ additions of $\frac{n}{m} \times \frac{n}{m}$ sub-matrices in each $p_{i,j}$ to compute $\mathcal{C}_{i,j}$. In this step we can use the best sequential algorithm to multiply matrices of order n, with complexity $O(n^\lambda) = o(n^3)$. With this, the needed time becomes $O(m(\frac{n}{m})^\lambda + m(\frac{n}{m})^2) = O(m(\frac{n}{m})^\lambda)$, as $\lambda \geq 2$.

Hence the total time of MM_1 is $O(n^\lambda/m^{\lambda-1}) = O(n^\lambda/p^{\frac{\lambda-1}{2}})$.

To simplify the formalization of this algorithm, we assume that:

1. A, B and C are registers (more precisely, set of registers) that store square matrices of $(n/m)^2$ elements each;
2. $MULTSEQ$ is a function that receives two of these registers as argument and performs the sequential multiplication of square matrices of order n/m in time $O((n/m)^\lambda)$ and returns the result matrix in the same format;
3. the elements of the vectors F_1 and F_2 are square matrices of order n/m.

MM_1 is called with $BITS = \{0, \ldots, 2\log_2 m - 1\}$.

MM$_1$($BITS$, A, B, C)
$q := |BITS|/2$
$LIN(j) := \{BITS[0], \ldots, BITS[q-1]\}$
$COL(j) := \{BITS[q], \ldots, BITS[2q-1]\}$
$F_1(j) := \emptyset$
MULTI-BR$(LIN(j), A(j), F_1(j))$
$F_2(j) := \emptyset$
MULTI-BR$(COL(j), B(j), F_2(j))$
$C(j) := \sum_{i=0}^{n-1} MULTSEQ(F_1[i](j), F_2[i](j))$

The algorithm suggested in [4] is based on NN_2 rather than NN_1, and has the same time complexity. We name that algorithm as MM_2, which is based on NN_2, and has the same time complexity.

4.4 A global view

The results presented in the previous sections are the best known to perform the multiplication of two square matrices of order n on the hypercube with p processors, where $1 \leq p \leq n^3$. The following table resumes the situation. (For simplicity, we put only one representative algorithm for each complexity.)

PROCESSORS	ALGORITHM	COMPLEXITY
$1 \leq p \leq n^2$	MM_1	$O(n^\lambda/p^{\frac{\lambda-1}{2}})$
$n^2 \leq p \leq n^3$	NNM_1	$O(\log\frac{p}{n^2} + \frac{n^3}{p})$

A global analysis of what have been presented shows that basically bi-dimensional and tri-dimensional arrangements of the processors are used. Based on these two arrangements we have some well-known algorithms, as N_1, NN_1, NN_2 and NNN_1, among others. If sub-matrices are considered as elements, these algorithms give origin to several classes of algorithms.

4.4.1 Bi-dimensional arrangement

Again we assume matrices of order n, and m is such that $1 \leq m \leq n$. In this first case (Figure 6), the hypercube is viewed as rs processors arranged in a matricial form of two dimensions, where r is the number of columns and s the number of rows.

$$p = rs \begin{cases} r = s \left\{ r = s = m: \ m^2 \text{ processors} \rightarrow MM \right. \\ r > s \begin{cases} r = n: \ nm \text{ processors} \rightarrow NM \text{ (new)} \\ r = m \text{ and } s = 1: \ m \text{ processors} \rightarrow M \text{ (new)} \end{cases} \\ r < s \left\{ \text{Can be considered as analogous.} \right. \end{cases}$$

4.4.2 Tri-dimensional arrangement

In the second case (Figure 7), r^2s processors of the hypercube are considered as a cubic matrix with r processors along the first two dimensions and s on the third.

$$p = r^2 s \begin{cases} r = s \,\{\, r = s = m: \ m^3 \text{ processors} \rightarrow MMM \text{ (new)} \\ r > s \begin{cases} r = n: \ n^2 m \text{ processors} \rightarrow NNM \\ r = m \text{ and } s = 1: \ m^2 \text{ processors} \rightarrow MM \end{cases} \\ r < s \,\{\, \text{Not interesting.} \end{cases}$$

4.5 New classes of algorithms

Three new classes of algorithms can be investigated: MMM, NM and M. We will examine only the MMM class and omit the other two (described in [12]), for not improving the existing algorithms. We can have algorithms in a same class with distinct complexities. For this reason, we have also a new algorithm for the class NNM, to be called algorithm NNM_3. Its complexity, though better than the existing algorithmns, is equivalent to MMM. It is described in [12]. Though we omit the details of several algorithms ([12]) their complexities are reproduced in this section, to give an overall view.

4.5.1 Class MMM: with $1 \leq p \leq n^3$ processors

The $p = m^3$ processors, where n and m are powers of two such that $1 \leq m \leq n$, are arranged as a cube of side m (Figure 8). Initially, each processor $p_{i,j,0}$ contains the sub-matrices $\mathcal{A}_{i,j}$ and $\mathcal{B}_{i,j}$, $0 \leq i, j < m$, of order n/m, corresponding to a total of $2(n/m)^2$ elements. The initial storage is only on the face of the cube formed by processors $p_{i,j,0}$, as in algorithms MM_1 and MM_2.

As we have m^3 processors, we will adapt one of the algorithms of the class NNN, namely NNN_1. This new algorithm will be named MMM_1. We have the following steps:

- Apply the four BROADCAST operations described in NNN_1, considering however the sub-matrices as elements. The time spent here is $O(\frac{n^2}{m^2}\log m)$.
- In each processor $p_{i,j,k}$ compute the product of the sub-matrices $\mathcal{A}_{i,k}$ and $\mathcal{B}_{k,j}$, using the best sequential algorithm. This takes $O((\frac{n}{m})^\lambda)$ time.
- Perform the ACCUM operation to compute the summation. This takes the same time as the first step, $O(\frac{n^2}{m^2}\log m)$.

The total time of MMM_1 is $O(\frac{n^2}{m^2}\log m + (\frac{n}{m})^\lambda) = O(\frac{n^2}{p^{2/3}}\log p + \frac{n^\lambda}{p^{\lambda/3}})$.

We make the same assumptions as in MM_1, relative to the registers A, B, C, A' and B' (each storing $(n/m)^2$ elements). In the following, *SUM2* is the same algorithm as *SUM*, with the only difference that we add the $(n/m)^2$ corresponding elements stored in each register. MMM_1 is called with $BITS = \{0, \ldots, 3\log_2 m - 1\}$.

MMM$_1$(*BITS, A, B, C*)

$q := |BITS|/3$
$I(j) := \{BITS[0], \ldots, BITS[q-1]\}$
$J(j) := \{BITS[q], \ldots, BITS[2q-1]\}$
$K(j) := \{BITS[2q], \ldots, BITS[3q-1]\}$
$BROADCAST(K(j), A(j))$
$BROADCAST(K(j), B(j))$
$A'(j) := A(j), \ (j_{q:2q-1} = j_{2q:3q-1})$
$BROADCAST(J(j), A'(j))$
$B'(j) := B(j), \ (j_{0:q-1} = j_{2q:3q-1})$
$BROADCAST(I(j), B'(j))$
$A'(j) := MULTSEQ(A'(j), B'(j))$
$SUM2(K(j), A'(j), C(j))$

4.6 Main results

The complexities of the various algorithms are given in Figure 9. All but the first two [4] are original.

In Figure 10 we group the algorithms into intervals and obtain their complexities in the cases where the number of processors is the same as the basic algorithms. We now have a better view for comparison.

The algorithms we described cover several intervals on the number of processors p of the hypercube. These intervals depend on the order n of the matrices $\mathcal{A}$ and $\mathcal{B}$. Therefore, we consider p as a function of n to present the comparisons that follow. We will assume the following. When we say $g(n) \leq p \leq h(n)$, this means $p = f(n)$ such that $g(n) \leq f(n) \leq h(n)$ for all positive n. Therefore, $f(n) = \Omega(g(n))$ and $f(n) = O(h(n))$.

With these considerations, we can establish a criterium of comparison. Given two algorithms X and Y defined in a same interval on the number of processors, of complexities $O(f_x(n))$ and $O(f_y(n))$, respectively, X will be considered better than Y if:

- $f_x(n) = O(f_y(n))$ in this interval;
- for at least some number of processors in this interval we have $f_x(n) = o(f_y(n))$.

The comparison results thus obtained can be grouped into the table shown in Figure 11.

We can thus conclude which ones are the best algorithms in each interval on the number of processors, that varies from 1 to n^3, with their respective complexities, as shown in Figure 12.

We choose some special values for the number of processors, and compare the complexities of the algorithms of [4] (MM_1 and NNM_1) with those of the two new algorithms MMM_1 and NNM_3 in the cases in which the latter ones are better.

A fictitious value for λ is assumed ($\lambda = 8/3$) to facilitate the comparison. Obviously, we could have used any other value for λ such that $2 \leq \lambda < 3$. We have Figure 13.

5 Conclusions

We have presented a new SIMD algorithm for $n \times n$ matrix multiplication on a hypercube of p processors with complexity $O(\frac{n^2}{p^{2/3}} \log p + \frac{n^\lambda}{p^{\lambda/3}})$, $1 \leq p \leq n^3$. This algorithm is superior in complexity than that of Dekel, Nassimi and Sahni, in the interval $1 \leq p \leq n^3/\log^3 n$. To derive the proposed algorithm, we have used the same matricial visualization proposed by the latter two authors. The matricial visualization of the hypercube constitutes an important approach to conceive new parallel algorithms. It also helps to make many algorithms easier to understand.

We describe the algorithms in terms of basic communication operations on the hypercube, and thus gain clarity, simplicity and concision. Usage of basic communication operations to describe algorithms was motivated by the work of Bertsekas and Tsitsiklis [2]. Once the reader becomes familiar with such operations, the algorithms described using them can be simpler and more concise.

One difficulty we experience was to compare the different algorithms. The complexities depend on the number of processors p and on the size of the matrices. We have also suggested a comparison criterium to enable this comparison.

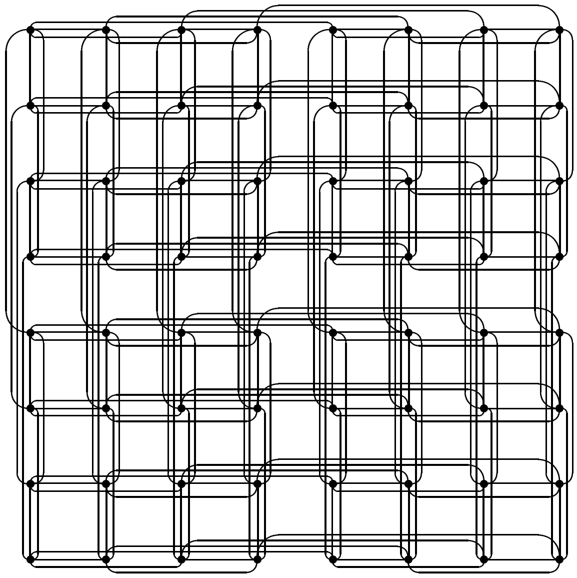


Figure 1: Matricial visualization of $H(6)$

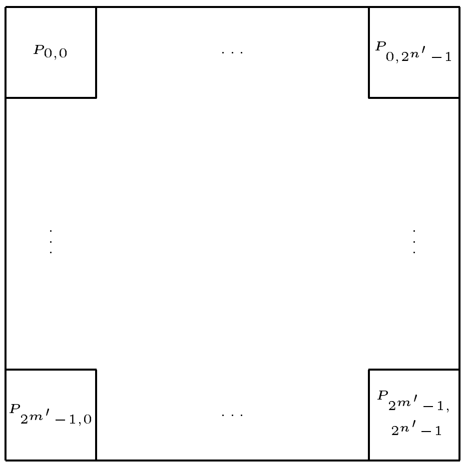


Figure 2: Hypercube visualized as sub-matrices

Acknowledgments

We thank the referees for their helpful comments and suggestions.

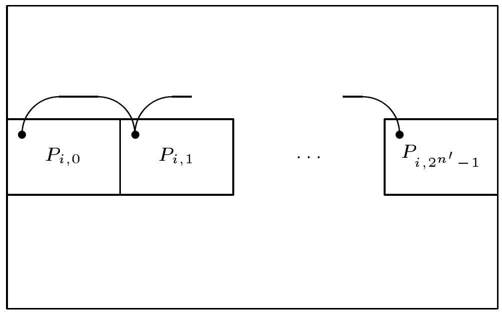


Figure 3: Hypercube formed by *corresponding elements*

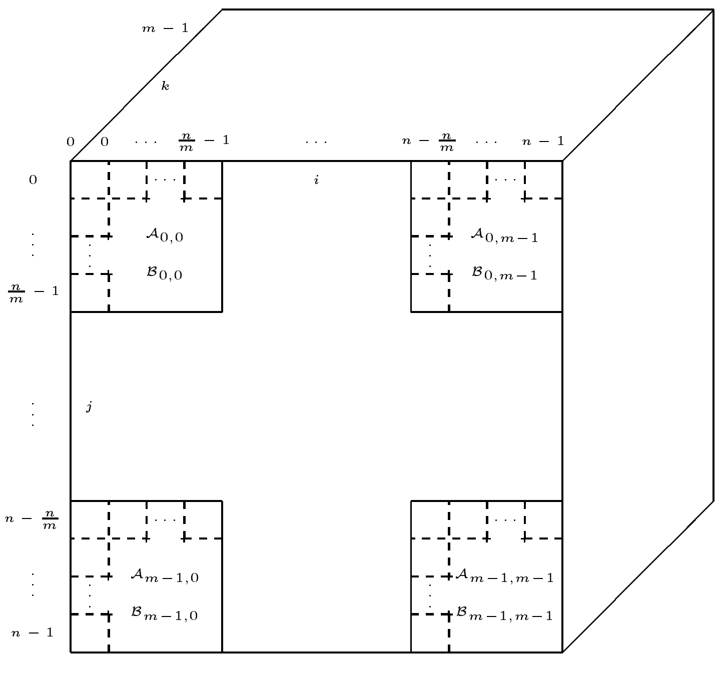


Figure 4: $n \times n \times m$ visualization

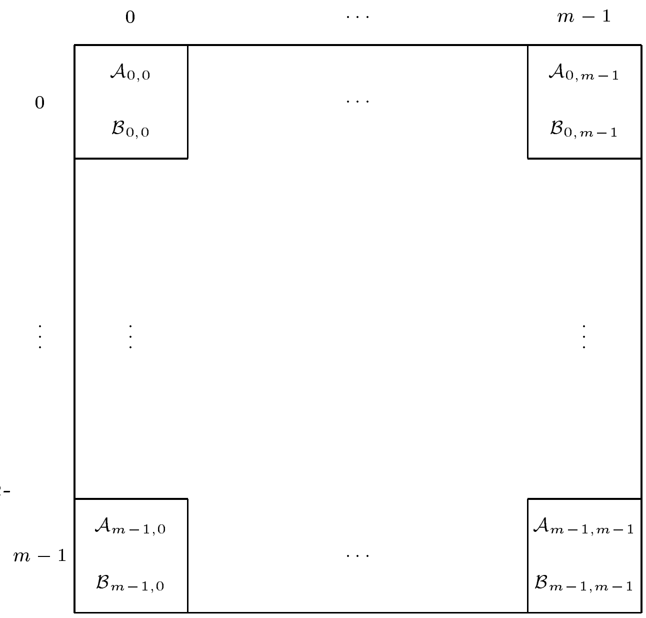


Figure 5: $m \times m$ visualization

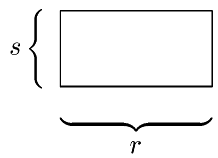


Figure 6: Algorithms suggested by bi-dimensional visualization

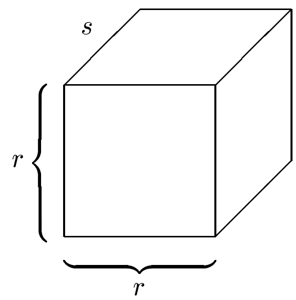


Figure 7: Algorithms suggested by tri-dimensional visualization

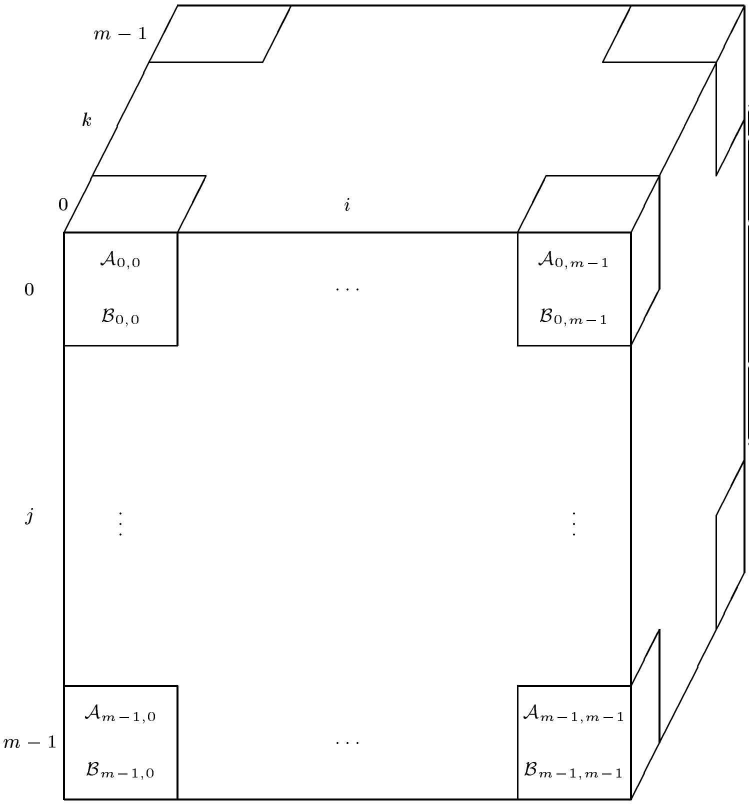


Figure 8: $m \times m \times m$ visualization

PROCESSORS	ALGORITHM	$p = n$	$p = n^2$	$p = n^3$
$1 \le p \le n$	M_1	$O(n^2)$		
	MM_1	$O(n^{\frac{\lambda+1}{2}})$		
	MMM_1	$O(n^{4/3} \log n + n^{2\lambda/3})$		
$n \le p \le n^2$	MM_1	$O(n^{\frac{\lambda+1}{2}})$	$O(n)$	
	NM_1	$O(n^2)$	$O(n)$	
	NM_2	$O(n^{\frac{\lambda+1}{2}})$	$O(n)$	
	NM_3	$O(n^{4/3} \log n + n^{2\lambda/3})$	$O(n)$	
	MMM_1	$O(n^{4/3} \log n + n^{2\lambda/3})$	$O(n^{2/3} \log n + n^{\lambda/3})$	
$n^2 \le p \le n^3$	NNM_1		$O(n)$	$O(\log n)$
	NNM_3		$O(n^{2/3} \log n + n^{\lambda/3})$	$O(\log n)$
	MMM_1		$O(n^{2/3} \log n + n^{\lambda/3})$	$O(\log n)$

Figure 10: Complexity results grouped in intervals

$1 \le p \le n$	M_1: worse than MM_1. MM_1: worse than MMM_1. MMM_1: the best.
$n \le p \le n^2$	MM_1: worse than NM_3. NM_1: worse than NM_2. NM_2: worse than NM_3. NM_3: worse than MMM_1. MMM_1: the best.
$n^2 \le p \le n^3$	NNM_1: possesses boundary with NNM_3. NNM_3: possesses boundary with NNM_1. MMM_1: equivalent to NNM_3.

Figure 11: Comparison results

ALGORITHM	PROCESSORS	COMPLEXITY
NNM_1	$n^2 \le p \le n^3$	$O(\log \frac{p}{n^2} + \frac{n^3}{p})$
MM_1	$1 \le p \le n^2$	$O(n^\lambda / p^{\frac{\lambda-1}{2}})$
MMM_1	$1 \le p \le n^3$	$O(\frac{n^2}{p^{2/3}} \log p + \frac{n^\lambda}{p^{\lambda/3}})$
NM_1	$n \le p \le n^2$	$O(n^3/p)$
NM_2	$n \le p \le n^2$	$O(n^\lambda / p^{\frac{\lambda-1}{2}})$
NM_3	$n \le p \le n^2$	$O(\frac{n^{5/3}}{p^{1/3}} \log \frac{n^2}{p} + \frac{n^\lambda}{p^{\lambda/3}} (\frac{p}{n})^{1-\lambda/3})$
M_1	$1 \le p \le n$	$O(n^\lambda / p^{\lambda-2})$
NNM_3	$n^2 \le p \le n^3$	$O(\frac{n^2}{p^{2/3}} \log n + \frac{n^\lambda}{p^{\lambda/3}})$

Figure 9: Complexity results of the various algorithm classes

PROCESSORS	ALGORITHMS	COMPLEXITIES
$1 \le p \le n^2$	MMM_1	$O(\frac{n^2}{p^{2/3}} \log p + \frac{n^\lambda}{p^{\lambda/3}})$
$n^2 \le p \le n^3 / \log^3 n$	MMM_1 or NNM_3	$O(\frac{n^2}{p^{2/3}} \log n + \frac{n^\lambda}{p^{\lambda/3}})$
$p = \Theta(n^3 / \log^3 n)$	MMM_1 or NNM_3 or NNM_1	$O(\log^3 n)$
$n^3 / \log^3 n \le p \le n^3$	NNM_1	$O(\log \frac{p}{n^2} + \frac{n^3}{p})$
$p = \Theta(n^3)$	MMM_1 or NNM_3 or NNM_1	$O(\log n)$

Figure 12: The best algorithms

PROCESSORS	COMPLEXITIES	COMPLEXITIES with $\lambda = 8/3$
$p = \log n$	$MM_1: \quad O(n^{\lambda}/\log^{\frac{\lambda-1}{2}} n)$ $MMM_1: \quad O(\frac{n^2}{\log^{2/3} n}\log\log n + \frac{n^{\lambda}}{\log^{\lambda/3} n})$	$MM_1: \quad O(\frac{n^{8/3}}{\log^{5/6} n})$ $MMM_1: \quad O(\frac{n^{8/3}}{\log^{8/9} n})$
$p = n$	$MM_1: \quad O(n^{\frac{\lambda+1}{2}})$ $MMM_1: \quad O(n^{4/3}\log n + n^{2\lambda/3})$	$MM_1: \quad O(n^{11/6})$ $MMM_1: \quad O(n^{16/9})$
$p = n\log n$	$MM_1: \quad O(n^{\frac{\lambda+1}{2}}/\log^{\frac{\lambda-1}{2}} n)$ $MMM_1: \quad O(n^{4/3}\log^{1/3} n + (\frac{n^2}{\log n})^{\lambda/3})$	$MM_1: \quad O(\frac{n^{11/6}}{\log^{5/6} n})$ $MMM_1: \quad O(\frac{n^{16/9}}{\log^{8/9} n})$
$p = n^{3/2}$	$MM_1: \quad O(n^{\frac{\lambda+3}{4}})$ $MMM_1: \quad O(n\log n + n^{\lambda/2})$	$MM_1: \quad O(n^{17/12})$ $MMM_1: \quad O(n^{4/3})$
$p = n^2$	$NNM_1: \quad O(n)$ $MMM_1: \quad O(n^{2/3}\log n + n^{\lambda/3})$	$NNM_1: \quad O(n)$ $MMM_1: \quad O(n^{8/9})$
$p = n^2\log n$	$NNM_1: \quad O(n/\log n)$ $NNM_3: \quad O(n^{2/3}\log^{1/3} n + (\frac{n}{\log n})^{\lambda/3})$	$NNM_1: \quad O(\frac{n}{\log n})$ $NNM_3: \quad O((\frac{n}{\log n})^{8/9})$
$p = n^{5/2}$	$NNM_1: \quad O(n^{1/2})$ $MMM_1: \quad O(n^{1/3}\log n + n^{\lambda/6})$	$NNM_1: \quad O(n^{1/2})$ $MMM_1: \quad O(n^{4/9})$

Figure 13: Special cases

References

[1] Bertsekas, D.P.; zveren, C.; Stamoulis, G.D.; Tseng, P. and Tsitsiklis, J.N., "Optimal Communication Algorithms for Hypercubes", *Journal of Parallel and Distributed Computing*, Vol. 11, p. 263-275, 1991.

[2] Bertsekas, D.P. and Tsitsiklis, J.N., *Parallel and Distributed Computation - Numerical Methods*, Prentice-Hall, 1989.

[3] Coppersmith, D. and Winograd, S., "Matrix Multiplication via Arithmetic Progression", *Proc. 19th Annual ACM Symp. on the Theory of Computing*, p. 1-6, 1987.

[4] Dekel, E.; Nassimi, D. and Sahni, S., "Parallel Matrix and Graph Algorithms", *SIAM J. Comp.*, Vol. 10, N. 10, p. 657-675, November 1981.

[5] Fraigniaud, P. and Lazard, E., *Methods and Problems of Communication in Usual Networks*, Technical Report LIP N. 91-33, October 1991.

[6] Hillis, W.D., *The Connection Machine*, MIT Press, Cambridge, Mass., 1985.

[7] Nassimi, D. and Sahni, S., "Data Broadcasting in SIMD Computers", *IEEE Trans. Comput.*, Vol. C-30, N. 2, p. 101-107, February 1981.

[8] Nassimi, D. and Sahni, S., "Parallel Permutation and Sorting Algorithms and a New Generalized Connection Network", *Journal of the ACM*, p. 642-667, July 1982.

[9] Pan, V. Ya., "New Fast Algorithms for Matrix Operations", *SIAM J. Comp.*, Vol. 9, N. 2, p. 321-342, May 1980

[10] Saad, Y. and Schultz, M.H., "Topological Properties of Hypercubes", *IEEE Trans. Comput.*, Vol. 37, N. 7, p. 867-872, July 1988.

[11] Saad, Y. and Schultz, M.H., "Data Communication in Hypercubes", *Journal of Parallel and Distributed Computing*, Vol. 6, p. 115-135, 1989.

[12] Alonso Sanches C.A and Song, S.W., "Revisiting SIMD algorithms for matrix multiplication on the hypercube". Technical Report, Department of Computer Science, IME, University of São Paulo, 1993. (In preparation.)

[13] Siegel, H.J., "Interconnection networks for SIMD machines", *Computer*, p. 57-65, June 1979.

[14] Strassen, V., "Gaussian Elimination is not Optimal", *Numer. Math.*, 13, p. 354-356, 1969.