Yale University
Department of Computer Science

Multiplication of Arbitrarily Shaped Matrices on Boolean Cubes Using the Full Communications Bandwidth

S. Lennart Johnsson and Ching-Tien Ho

YALEU/DCS/TR-721
July 1989

This work was supported by the Office of Naval Research under Grant N00014-89-J-1906. Approved for public release: distribution is unlimited.

Multiplication of Arbitrarily Shaped Matrices On Boolean Cubes Using the Full Communications Bandwidth

S. Lennart Johnsson[1] and Ching-Tien Ho
Department of Computer Science
Yale University
New Haven, CT 06520
Johnsson@cs.yale.edu, Johnsson@think.com, Ho@cs.yale.edu

Abstract. Algorithms for the multiplication of matrices of arbitrary shapes using the full communications bandwidth of multiprocessors configured as Boolean cubes are presented. The main new feature of the algorithms is the organization of the data motion. The communication is optimal for algorithms accumulating the inner-products for the product matrix *in-place*, assuming a traditional matrix multiplication algorithm requiring $2PQR$ arithmetic operations for the operation $C \leftarrow C + A \times B$, where A is a $P \times Q$ matrix and B is a $Q \times R$ matrix. For matrices with at least $\frac{n}{2}$ elements of the common dimension per processor in a Boolean n-cube all communication channels are used in every multiplication step of the algorithm. The presented algorithms are valid regardless of matrix size relative to the number of processors. For the multiplication of "small" matrices relative to the number of processors the common dimension is parallelized to the extent possible.

1 Introduction

The multiplication of matrices of arbitrary shapes is an important operation in many computationally intensive scientific applications. The number of floating-point processors in todays supercomputers range from a few to a few thousand. The next generation supercomputers with performance of a trillion (floating-point) operations per second are expected to have thousands to tens of thousands of processors interconnected by some form of network. With the current technologies the communication capabilities are often the limiting factor with respect to performance, especially where different technologies meet, as at the chip and board boundaries. Effective use of the communication bandwidth is critical for maximum performance. The communication needs are minimized by a good choice of address map, i.e., data placement, and routing algorithms that minimize path lengths and congestion once an address map is given.

Several different networks are used for interconnection of processors in highly concurrent systems, with two-dimensional meshes, butterfly networks, and Boolean cubes being the most common. We focus on Boolean cube networks. In such a network of N nodes, each node has $\log_2 N$ neighbors. With sufficiently high data motion capability at each node, communication may be performed on all ports concurrently, and the full communications bandwidth of the network, and the technology, used.

Cannon [2] has given an algorithm for the multiplication of square matrices on two-dimensional meshes. Since a two-dimensional mesh is a subgraph of a Boolean cube [11],

[1]Also with the Department of Electrical Engineering and Thinking Machines Corp.

[5], it is possible to use Cannon's algorithm on a Boolean cube by emulating a mesh [6]. Dekel *et al.* [3] have described an algorithm for multiplication of square matrices on a Boolean cube with an even number of dimensions. Both Dekel's and Cannon's algorithms assume that the number of matrix elements is equal to the number of processors. A generalization of Cannon's algorithm to matrices of arbitrary shapes and sizes is given in [7] and [9]. Cannon's algorithm may use up to four communication (unidirectional) channels per processor concurrently. Dekel's algorithm only use two (bidirectional) channels at a time. The algorithm presented below concurrently use all n (bidirectional) channels in a Boolean n-cube.

The paper is organized as follows. In the next section, we introduce a few basic results regarding the specific communication operations used in the multiplication algorithms. In Section 3, we generalize Dekel's algorithm to the multiplication of $P \times Q$ and $Q \times R$ matrices on a Boolean n-cube, configured as a product cube of $\sqrt{N} \times \sqrt{N}$ processors, and show how the Boolean cube bandwidth can be fully utilized, assuming $P, Q, R \geq \sqrt{N}$. Section 4 generalizes the algorithm to Boolean cubes configured as a product cube with $N_0 \times N_1$ processors, $N_0 N_1 = N$. Section 5 generalizes the algorithm further to matrices of arbitrary shapes and sizes multiplied on Boolean cubes of arbitrary configuration. Conclusions are found in Section 6.

2 Preliminaries

The focus of this paper is on communication efficient algorithms. In this section we introduce some of the notation used throughout the paper, and some basic results regarding communication in Boolean cubes. These results are needed for the proof of the optimality of the communication in the matrix multiplication algorithms that are the main contribution of this paper.

In the following log denotes $\log_2$. The bit-wise exclusive-or operation is denoted "$\oplus$" and $\mathcal{Z}_n = \{0, 1, \cdots, n-1\}$. The number of 1-bits in the binary representation of i is $||i||$. $*^n$ denotes a string of n instances of $*$, where $*$ is either 0 or 1. $\mathcal{D}(i)$ is the ordered set of dimensions (in an increasing order) for which the corresponding bits of the binary representation of i are one. The dimension of the least significant bit is zero. It is the rightmost bit in a field. For example, $\mathcal{D}((10110)) = \{1, 2, 4\}$. $|\mathcal{D}(i)|$ denotes the cardinality of the set $\mathcal{D}(i)$. $|\mathcal{D}(i)| = ||i||$. Let $e(\mathcal{D}(i), x)$ be the xth element in the set $\mathcal{D}(i)$, $0 \leq x \leq ||i|| - 1$. A is a $P \times Q$ matrix, B a $Q \times R$ matrix, and C a $P \times R$ matrix, where $P = 2^p$, $Q = 2^q$, and $R = 2^r$. The element in row i and column j of matrix A is $a(i, j)$, $i \in \mathcal{Z}_P$, $j \in \mathcal{Z}_Q$. $b(i, j)$ and $c(i, j)$ are similarly defined. The axis encoding the rows is axis zero, and the axis encoding columns is axis one. Processors along axis zero are identified by k and along axis one by ℓ. The matrices are distributed over the nodes of a Boolean n-cube with $N = 2^n$ nodes. The cube is factored with n_0 dimensions along axis zero and n_1 dimensions along axis one, $n_0 + n_1 = n$. The length of processor axis zero is $N_0 = 2^{n_0}$ and of axis one $N_1 = 2^{n_1}$.

Let $\mathcal{S}(1, 0) = (0)$ and $\mathcal{S}(n, 0) = \mathcal{S}(n-1, 0)|n-1|\mathcal{S}(n-1, 0)$ for $n > 0$, where "|" is the concatenation operator of two sequences. For instance, $\mathcal{S}(3, 0) = (0, 1, 0, 2, 0, 1, 0)$. $\mathcal{S}(n, 0)$ is the transition sequence in a *binary-reflected* n-bit Gray code [13]. If $\mathcal{S}(n, 0) = (x_1, x_2, \cdots, x_{(2^n-1)})$, then

$$\mathcal{S}(n, s) = ((x_1 + s) \bmod n, (x_2 + s) \bmod n, \cdots, (x_{(2^n-1)} + s) \bmod n), \quad 0 \leq s < n.$$

For instance, $\mathcal{S}(3,1) = (1,2,1,0,1,2,1)$. Let $\alpha(t,n,s)$ be the tth element of the sequence $\mathcal{S}(n,s)$, $1 \leq t \leq 2^n - 1$.

The communication times are measured by the number of elements transferred in sequence. Concurrent communication on all ports of all processors is assumed possible, unless stated otherwise. All communications links are bidirectional.

A particular communication pattern that is used for one phase of the matrix multiplication algorithm is *bit-inversion* [14]. A bit-inversion in an n-cube implies that processor i sends its data to processor $i \oplus (1^n)$ for all i's.

Lemma 1 [14] *A tight bound for bit-inversion on the complete processor address by every processor in a Boolean n-cube is K with K elements per processor.*

Proof: The required bandwidth is nNK and the available bandwidth is nN, which gives the lower bound K. An upper bound equal to the lower bound is given by the following algorithm. Divide the local data set into n parts, and exchange part i, $0 \leq i \leq n-1$, according to the sequence of dimensions $i, (i+1) \bmod n, \cdots, (i+n-1) \bmod n$. All n data sets can be exchanged concurrently without edge conflict. ∎

In general, with *bit-inversion* on only a subset of the processor address bits of every processor, the communications requirements are reduced, but not the lower bound.

Lemma 2 [10] *Any tight bound for communication in a Boolean n-cube is also a tight bound for the same communication in all disjoint n dimensional subcubes of an n' dimensional cube, when the subcubes are identified by the same n dimensions, $n' > n$.*

The significance of this lemma is that even though only a fraction $\frac{n}{n'}$ of the total bandwidth of the n'-cube is used, the communication time cannot be reduced when the communication in each subcube is optimal. Hence, in the case of the same *bit-inversion* on a subset of the bits of the address space the tight lower bound is still K for K elements per processor. The bits subject to inversion define a subcube, and the bits not inverted define the disjoint instances of the subcubes in which inversion is performed.

When the Boolean cube is factored as an $N_0 \times N_1$ cube, then the alignment of one of the operands implies a combination of bit-inversion and *all-to-all personalized communication* [8] within subcubes. In all-to-all personalized communication every processor sends a unique set of data to every other processor. For instance, if $n_1 > n_0$ then the alignment of row $2^{n_0} - 1$ of A requires a bit-inversion on n_0-dimensional subcubes and all-to-all personalized communication within $(n_1 - n_0)$-dimensional subcubes. The following lemma gives lower and upper bounds for the alignment confined to row/column subcubes for Boolean cubes configured as an $N_0 \times N_1$ cube.

Lemma 3 *If every processor k in an n_1-cube sends unique sets of $\frac{K}{2^{n_1-n_0}}$ elements to each of $2^{n_1-n_0}$ processors, $n_1 \geq n_0$, defined by $k \oplus (1^{n_0} *^{n_1-n_0})$ then a tight bound for the communication complexity is K if $n_0 > 0$ and $\frac{K}{2}$ for $n_0 = 0$.*

Proof: For $n_0 = 0$, the communication is *all-to-all personalized communication* in an n_1-cube. A tight bound of complexity $\frac{K}{2}$ is given in [8]. For $n_0 > 0$, the lower bound can be derived from the bit-inversion alone. The entire data set K in a processor is subject to the same bit-inversion on n_0 bits. The required bandwidth is $Kn_0 2^{n_1}$. To determine the available bandwidth the n_1 dimensional cube is considered as an n_0 cube of "supernodes" consisting of $2^{n_1-n_0}$ cube nodes, and $2^{n_1-n_0}$ communication channels between every pair of "supernodes". The available bandwidth is $n_0 2^{n_1-n_0} 2^{n_0}$, and the lower bound K follows.

For the upper bound for $n_0 > 0$ partition each of the $\frac{K}{2^{n_1-n_0}}$ local data sets further into n_1 sets. Set s of size $\frac{K}{n_1 2^{n_1-n_0}}$ is assigned an exchange sequence $s, (s+1) \bmod n_1, \cdots, (s+n_1-1) \bmod n_1$. The exchange on a dimension is conditional, and determined by the dimensions involved in the personalized communication of the set of size $\frac{K}{2^{n_1-n_0}}$, and the bit-inversion. All n_1 different exchange sequences can be performed concurrently, and all blocks of size $\frac{K}{n_1 2^{n_1-n_0}}$ that needs to be exchanged in a dimension can be exchanged as one transfer operation should that be advantageous. The permutation can be completed in n_1 exchanges. Each set of $\frac{K}{2^{n_1-n_0}}$ elements require $\frac{K}{2^{n_1-n_0}}$ element transfers in sequence by this algorithm, and the total number of element transfers in sequence is K. ▮

3 A block algorithm

In this section we first generalize Dekel's algorithm for the operation $C \leftarrow A \times B + C$ to the multiplication of $P \times P$ matrices distributed uniformly over the processors of a Boolean n-cube, $2p \gg n$, n even. We assume that the processors of the Boolean cube are factored such that $n_0 = \frac{n}{2}$ dimensions are assigned to axis zero and $n_1 = \frac{n}{2}$ dimensions are assigned to axis one. Then we generalize the algorithm to the multiplication of a $P \times Q$ matrix by a $Q \times R$ matrix, $p, q \geq n_0$ and $q, r \geq n_1$. Finally, we present an algorithm that use all communication channels of a Boolean n-cube. The case $n_0 \neq n_1$ is treated in the next section.

3.1 Dekel's algorithm

Dekel's algorithm [3] assumes that A and B are $P \times P$ matrices, $P = 2^p$, and that the number of Boolean cube processors is P^2. The algorithm consists of two phases: alignment and multiplication.

Alignment:
$$a(i,j) \leftarrow a(i, i \oplus j), \forall i,j \in \mathcal{Z}_P,$$
$$b(i,j) \leftarrow b(i \oplus j, j), \forall i,j \in \mathcal{Z}_P.$$
Multiplication, step t, $0 \leq t \leq P-1$:
$$a(i,j) \leftarrow a(i, j \oplus 2^{\alpha(t,p,0)}), \text{ if } t \neq 0, \forall i,j \in \mathcal{Z}_P,$$
$$b(i,j) \leftarrow b(i \oplus 2^{\alpha(t,p,0)}, j), \text{ if } t \neq 0, \forall i,j \in \mathcal{Z}_P,$$
$$c(i,j) \leftarrow a(i,j) * b(i,j) + c(i,j), \forall i,j \in \mathcal{Z}_P.$$

With one element per processor and (i,j) being a processor address, the column index of an element of A is the same as the row index of an element of B for every processor (i,j) after the alignment phase, and for each step of the multiplication phase. Moreover, for any integer, complementing the bits of its binary encoding according to the transition sequence in a *binary-reflected* Gray code, such as the sequence $\mathcal{S}(p,0)$ for a p-bit number, produces every integer that can be encoded in p bits precisely once. Hence, during the course of the algorithm, processor (i,j) receives all the elements of row i of matrix A and column j of matrix B appropriately synchronized.

Replacing $\alpha(t,p,0)$ by $\alpha(t,p,s)$, $1 \le s \le p-1$, yields a matrix multiplication algorithm that for each t performs an exchange in dimension $(\alpha(t,p,0)+s) \bmod p$ instead of dimension $\alpha(t,p,0)$. This observation is the basis for defining an algorithm that fully uses the communications bandwidth of the Boolean cube.

3.2 Naive extension

3.2.1 Square matrices

Each processor holds a $2^{p-\frac{n}{2}} \times 2^{p-\frac{n}{2}}$ submatrix (consecutive assignment [6]). The data assignment is defined by the address map

$$(\underbrace{w^r_{p-1}w^r_{p-2}\cdots w^r_{p-\frac{n}{2}}}_{rp^r}\underbrace{w^r_{p-\frac{n}{2}-1}\cdots w^r_0}_{vp^r} \mid \underbrace{w^c_{p-1}w^c_{p-2}\cdots w^c_{p-\frac{n}{2}}}_{rp^c}\underbrace{w^c_{p-\frac{n}{2}-1}\cdots w^c_0}_{vp^c}).$$

All operands have corresponding address maps. *Virtual processor* address bits (labeled vp) define local storage addresses, whereas the *real processor* address bits (labeled rp) define different physical processors. The superscripts "r" and "c" denote "row" and "column", respectively. The exchange operation defined by the exclusive-or operation on the virtual processor address bits reorders data in the local storage of *all* processors, but there is no exchange between real processors. An exclusive-or operation on bits in the real processor field implies an exchange of all data between pairs of processors. The local address map is preserved.

Lemma 4 *The alignment on the bits in the virtual processor address field, and the steps of the multiplication phase corresponding to bits in this address field defines a complete matrix multiplication on blocks of size* $2^{p-\frac{n}{2}} \times 2^{p-\frac{n}{2}}$.

Lemma 4 follows from the recursive nature of the binary-reflected Gray code. This block matrix multiplication can be replaced by any suitable matrix multiplication algorithm in each node, without affecting the part of the algorithm requiring interprocessor communication. For instance, a block, matrix-vector, or SAXPY [12] based algorithm may be used depending on the architecture of each node.

Theorem 1 *The multiplication of two square matrices of size $P \times P$ on an n-cube, $2p \gg n$, can be performed by applying the algorithm by Dekel et al.* [3] *to the real processor address field, and by employing any suitable matrix multiplication algorithm for the local blocks of size* $2^{p-\frac{n}{2}} \times 2^{p-\frac{n}{2}}$, *assuming consecutive assignment of matrix elements to real processors.*

The time complexity of the algorithm is,

1. Communication:
   - Alignment: $\frac{n}{2}\frac{P^2}{N}$.

- Multiplication: $(\sqrt{N}-1)\frac{P^2}{N}$.

2. Arithmetic: $\frac{2P^3}{N}$.

The alignments of the matrices A and B are assumed to take place concurrently in the above estimates. The arithmetic time is reduced in proportion to the number of processors, but the largest communication term only in proportion to the square root of the number of processors. The data motion for the matrix A only uses one cube dimension per processor, and so does the data motion for B. A total of two cube dimensions are used for each processor, in each step of the multiplication algorithm. The communications capability of Boolean cubes of many dimensions is poorly utilized.

3.2.2 Rectangular matrices

In the block algorithm in the preceding section, the critical assumption for the alignment is that the address fields for P, Q, and R are of the same length. Hence, by extending each of these address fields to $\max(p,q,r)$ bits the alignment will be correct regardless of matrix shape. The extension can be made by adding high order bits (unused). In the multiplication phase, an *all-to-all broadcasting* is performed within rows of A and columns of B. Every element of a column of B must visit every processor row to which elements of A are allocated. This requirement implies that the exchange sequence is of length $\max(\min(P,N_0),\min(Q,N_0))$. Similarly, every element of a row of A must visit every processor column to which elements of B are allocated. This requirement implies that the length of the exchange sequence must be $\max(\min(R,N_1),\min(Q,N_1))$. With $n_0 = n_1 = \frac{n}{2}$ we have:

Corollary 1 *The multiplication of a $P \times Q$ matrix by a $Q \times R$ matrix on a Boolean n-cube factored as two $\frac{n}{2}$-cubes, can be performed by a block version of Dekel's algorithm requiring $2\lceil\frac{P}{\sqrt{N}}\rceil\lceil\frac{R}{\sqrt{N}}\rceil\lceil\frac{Q}{\sqrt{N}}\rceil \min\{\max(P,Q,R),\sqrt{N}\}$ arithmetic operations, and $\max(\lceil\frac{P}{\sqrt{N}}\rceil,\lceil\frac{R}{\sqrt{N}}\rceil)\lceil\frac{Q}{\sqrt{N}}\rceil\{\min(\max(p,q,r),\frac{n}{2})+\min(\max(P,Q,R),\sqrt{N})-1\}$ element transfers in sequence with concurrent communication of A and B.*

The number of processors performing arithmetic operations is $\min(P,\sqrt{N})\min(R,\sqrt{N})$. In this set a processor is active $\min(Q,T)$ out of $T = \min(\max(P,Q,R),\sqrt{N})$ steps. The number of processors involved in communication during the multiplication phase is $\min(P,\sqrt{N})T$ for matrix A. For the matrix B $\min(R,\sqrt{N})T$ processors participates in the communication. Every processor performing arithmetic operations also performs communications, but the converse is not necessarily true. Techniques for increased processor utilization for $P,Q,R < \sqrt{N}$ are given in section 5.

3.3 A block algorithm using all cube dimensions

By partitioning the matrix A into $1 \times \lambda$ blocks and the matrix B into $\lambda \times 1$ blocks the matrix multiplication is transformed into λ rank $\frac{Q}{\lambda}$ updates. The idea in the algorithm below is to perform the communication for the different high rank updates concurrently. If $Q \geq \frac{n}{2}\sqrt{N}$, then $\lambda = \frac{n}{2}$ and $\frac{n}{2}$ communication channels per processor are used for both A

and B. The full communications bandwidth is used. We refer to the $P \times \frac{Q}{\lambda}$ blocks of A and the $\frac{Q}{\lambda} \times R$ blocks of B as level-one blocks in order to distinguish this blocking from the blocks assigned to individual processors, the level-zero blocks. The naive block algorithm modified as described below is used for the multiplication of each pair of level-one blocks.

3.3.1 Data allocation

Each level-one block is allocated to the processors with consecutive assignment [6] (as in the preceding section), and different level-one blocks assigned cyclically. The address map for A is

$$(\underbrace{w^r_{p-1} w^r_{p-2} \cdots w^r_{p-\frac{n}{2}}}_{rp^r} \underbrace{w^r_{p-\frac{n}{2}-1} \cdots w^r_0}_{vp0^r} |$$

$$\underbrace{w^c_{q-1} w^c_{q-2} \cdots w^c_{q-\mu}}_{vp1^c} \underbrace{w^c_{q-\mu-1} \cdots w^c_{q-\mu-\frac{n}{2}}}_{rp^c} \underbrace{w^c_{q-\mu-\frac{n}{2}-1} \cdots w^c_0}_{vp0^c})$$

and for B it is

$$(\underbrace{w^r_{q-1} w^r_{q-2} \cdots w^r_{q-\mu}}_{vp1^r} \underbrace{w^r_{q-\mu-1} \cdots w^r_{q-\mu-\frac{n}{2}}}_{rp^r} \underbrace{w^r_{q-\mu-\frac{n}{2}-1} \cdots w^r_0}_{vp0^r} |$$

$$\underbrace{w^c_{r-1} w^c_{r-2} \cdots w^c_{r-\frac{n}{2}}}_{rp^c} \underbrace{w^c_{r-\frac{n}{2}-1} \cdots w^c_0}_{vp0^c})$$

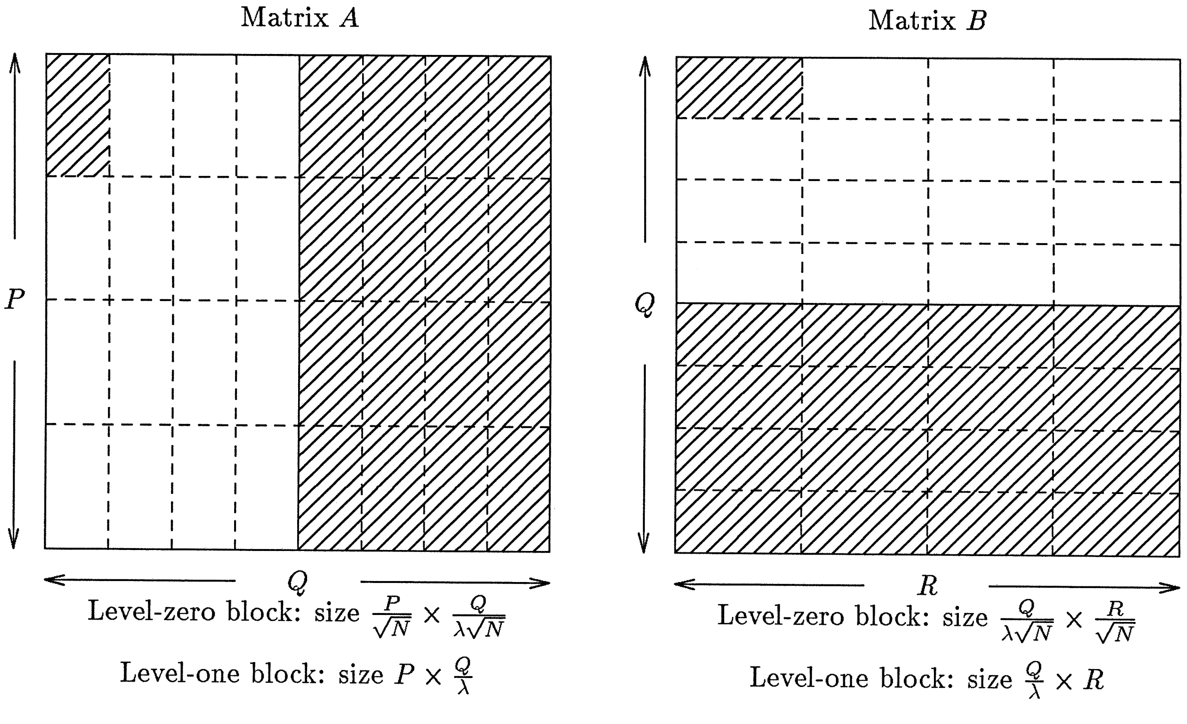
assuming that $\lambda \leq \frac{n}{2}$ is a power of two and $\mu = \log \lambda$. This assumption is only made for notational convenience in the address map. The blocks at level zero are defined by the fields labeled $vp0$. The level-zero block size for A is $\lceil \frac{P}{\sqrt{N}} \rceil \times \lceil \frac{Q}{\lambda\sqrt{N}} \rceil$, and for B it is $\lceil \frac{Q}{\lambda\sqrt{N}} \rceil \times \lceil \frac{R}{\sqrt{N}} \rceil$. Level-one blocks are identified by the field labeled $vp1$, and labeled with $m, 0 \leq m \leq \lambda - 1$. The concatenated rp and $vp0$ fields define the level-one blocks. Each such block is distributed uniformly over all $\sqrt{N} \times \sqrt{N}$ processors. The partitioning of the matrices is illustrated in Figure 1. Solid lines define boundaries of level-one blocks, and dashed lines boundaries of level-zero blocks. Typical level-one and level-zero blocks are represented by large and small shaded areas, respectively.

By Lemma 4 the alignment and subsequent exchange and multiplication operations related to the $vp0^c$ field of A and $vp0^r$ field of B define a block matrix multiplication local to every processor. Note that since $n_0 = n_1$ the lengths of the two fields, $vp0^c$ and $vp0^r$ are the same. The exchange on the $vp1$ field is a local memory move. This exchange implies that in the next several steps a new pair of level-one blocks will be multiplied. Local memory moves can be avoided by a suitable address calculation during the multiplication phase.

3.3.2 Alignment

The alignment is performed on the processor address fields alone, i.e., after the alignment processor (k, ℓ) has column indices

$$(\underbrace{* * \cdots *}_{vp1^c} \underbrace{k \oplus \ell}_{rp^c} \underbrace{* * \cdots *}_{vp0^c})$$



Matrix A
Matrix B
P
Q
Q
R
Level-zero block: size $\frac{P}{\sqrt{N}} \times \frac{Q}{\lambda\sqrt{N}}$
Level-one block: size $P \times \frac{Q}{\lambda}$
Level-zero block: size $\frac{Q}{\lambda\sqrt{N}} \times \frac{R}{\sqrt{N}}$
Level-one block: size $\frac{Q}{\lambda} \times R$

Figure 1: Partitioning of the matrices.

of the matrix A, and row indices

$$(\underbrace{**\cdots*}_{vp1^r}\underbrace{k\oplus\ell}_{rp^r}\underbrace{**\cdots*}_{vp0^r})$$

of the matrix B, where $**\cdots*$ denotes all numbers that can be represented by that bit-field. The matrices are properly aligned. For a processor in row k and column ℓ, the alignment of A involves the set of cube dimensions $\mathcal{D}(k\sqrt{N})$ (the higher-order $\frac{n}{2}$ cube dimensions are used for the encoding of rows) and the alignment of B involves the set of cube dimensions $\mathcal{D}(\ell)$. Clearly, $\mathcal{D}(k\sqrt{N}) \cap \mathcal{D}(\ell) = \phi$. Note that the set of dimensions involved in the alignment operation does not depend on the level-one block index m.

The number of processor dimensions involved in the alignment of A is $|\mathcal{D}(k\sqrt{N})| = ||k||$ for processor row k. The number of dimensions involved in the alignment of B is $||\ell||$ for processor column ℓ. The number of processor dimensions involved for A ranges from zero to $\min(p, \frac{n}{2})$ and for B from zero to $\min(r, \frac{n}{2})$. The data volume that needs to be communicated per processor is $\lceil \frac{P}{\sqrt{N}} \rceil \lceil \frac{Q}{\sqrt{N}} \rceil$ for A and $\lceil \frac{R}{\sqrt{N}} \rceil \lceil \frac{Q}{\sqrt{N}} \rceil$ for B. The naive block algorithm does not fully use the communication bandwidth of the Boolean cube.

We first constrain the alignment of a row to be confined to its row subcube, and the alignment of a column to be confined to its column subcube. By Lemma 1 and Lemma 2 the minimum number of element transfers in sequence under this constraint is $\max(\lceil \frac{P}{\sqrt{N}} \rceil \lceil \frac{Q}{\sqrt{N}} \rceil, \frac{n}{2})$ for A and $\max(\lceil \frac{R}{\sqrt{N}} \rceil \lceil \frac{Q}{\sqrt{N}} \rceil, \frac{n}{2})$ for B. It can be shown that the complexity still holds even when $\lceil \frac{P}{\sqrt{N}} \rceil \lceil \frac{Q}{\sqrt{N}} \rceil$ is not a multiple of $\frac{n}{2}$. The time for the alignment of both operands is $\max(\lceil \frac{P}{\sqrt{N}} \rceil \lceil \frac{Q}{\sqrt{N}} \rceil, \lceil \frac{R}{\sqrt{N}} \rceil \lceil \frac{Q}{\sqrt{N}} \rceil, \frac{n}{2})$. The alignment of each operand is sped up by a factor of $\frac{n}{2}$ by concurrent communication within subcubes, compared to the algorithm in the preceding section.

Lemma 5 *A lower bound for the alignment of A and B on a Boolean n-cube configured with $n_0 = n_1 = \frac{n}{2} \geq 1$ and $P, Q, R \geq \sqrt{N}$ is $\frac{(P+R)Q}{2N}$.*

Proof: Consider the $\frac{N}{4}$ processors in rows $\{1*^{\frac{n}{2}-1}\}$ and columns $\{1*^{\frac{n}{2}-1}\}$, i.e., the processors to which the lower right quarter submatrix of each operand is allocated. These $\frac{N}{4}$ processors form a $(n-2)$-dimensional subcube. Each processor in the subcube needs to exchange $\frac{PQ}{N}$ elements with the subcube storing the lower left quarter submatrix of A, and $\frac{QR}{N}$ elements with the subcube storing the upper right quarter submatrix of B. The total number of elements that must be sent out of the subcube is $\frac{(P+R)Q}{4}$. The total number of links that connect to processors outside the subcube is $2\frac{N}{4}$. ∎

Corollary 2 *Restricting the alignment to row/column subcubes is optimal if $P = R$, and suboptimal by at most a factor of two for $P, Q, R \geq \sqrt{N}$.*

The worst case is $P \gg R$ or $P \ll R$. We will now present an algorithm that has a lower complexity than the subcube alignment algorithm for this case.

The alignment of A is a *bit-inversion* on the column address field. The *bit-inversion* is dependent on the row address. Lemma 2 applies to each subcube, but *not* to the complete alignment operation. Different row subcubes execute different *bit-inversions*. Similar arguments apply to the alignment of B with respect to column subcubes. If column dimension ℓ_s must be routed for the alignment within processor row k of A, then $k_s = 1$. The key observation for the algorithm outlined here is that in the row complimentary to row k with respect to bit s, $k \oplus 2^{k_s}$, column dimension ℓ_s is not included in the alignment, since in that row $k_s = 0$. Hence, if the alignment of A is performed one column dimension at a time, then part of the data can be exchanged within the subcube with $k_s = 1$, and part of the data exchanged by sending it to the complimentary row with respect to the column dimension being routed, exchanging it there, and sending it back to the original row subcube. With δ elements per processor and δ_c elements exchanged in the complimentary row subcube the time for the alignment on one column dimension is $\max(\delta - \delta_c, \delta_c + 2)$, which is minimized for $\delta_c = \lfloor \frac{\delta}{2} \rfloor - 1$. The optimal time for the alignment on one column dimension can be derived as $\lceil \frac{\delta}{2} \rceil + 1$ if $\delta \geq 2$, or 1 if $\delta = 1$. Note that the communication between a row and its complimentary row is in the $1 \rightarrow 0$ direction before the exchange and in the $0 \rightarrow 1$ direction after the exchange. The elements can be pipelined. Figure 2 shows the three steps for the alignment on a 6-cube with the alignment performed one dimension at a time. The shaded area represents the rows for which alignment is required during the next step. To complete the description of the algorithm we notice that for each alignment (column) dimension, there is a unique row dimension. The cube dimensions for row and column encoding are disjoint, and it follows that the alignment can be performed on all dimensions concurrently. The data set is divided into $\frac{n}{2}$ pieces, and set u, $0 \leq u < \frac{n}{2}$, subject to alignment, if necessary, in column dimension $u, (u+1) \bmod \frac{n}{2}, \cdots, (u + \frac{n}{2} - 1) \bmod \frac{n}{2}$. The total time for the alignment of A is $\frac{n}{2}(\lceil \lceil \frac{P}{\sqrt{N}} \rceil \lceil \frac{Q}{\sqrt{N}} \rceil \frac{1}{n} \rceil + 1)$ for $\lceil \frac{P}{\sqrt{N}} \rceil \lceil \frac{Q}{\sqrt{N}} \rceil > \frac{n}{2}$, and $\frac{n}{2}$ otherwise.

Lemma 6 *An upper bound for the alignments of A and B is* $\min\{\frac{n}{2}(\lceil \frac{PQ}{nN} \rceil + \lceil \frac{RQ}{nN} \rceil) + n, \max(\frac{PQ}{N}, \frac{RQ}{N})\}$ *with $P, Q, R \geq \sqrt{N}$, $\frac{PQ}{N} \geq \frac{n}{2}$ and $\frac{RQ}{N} \geq \frac{n}{2}$.*

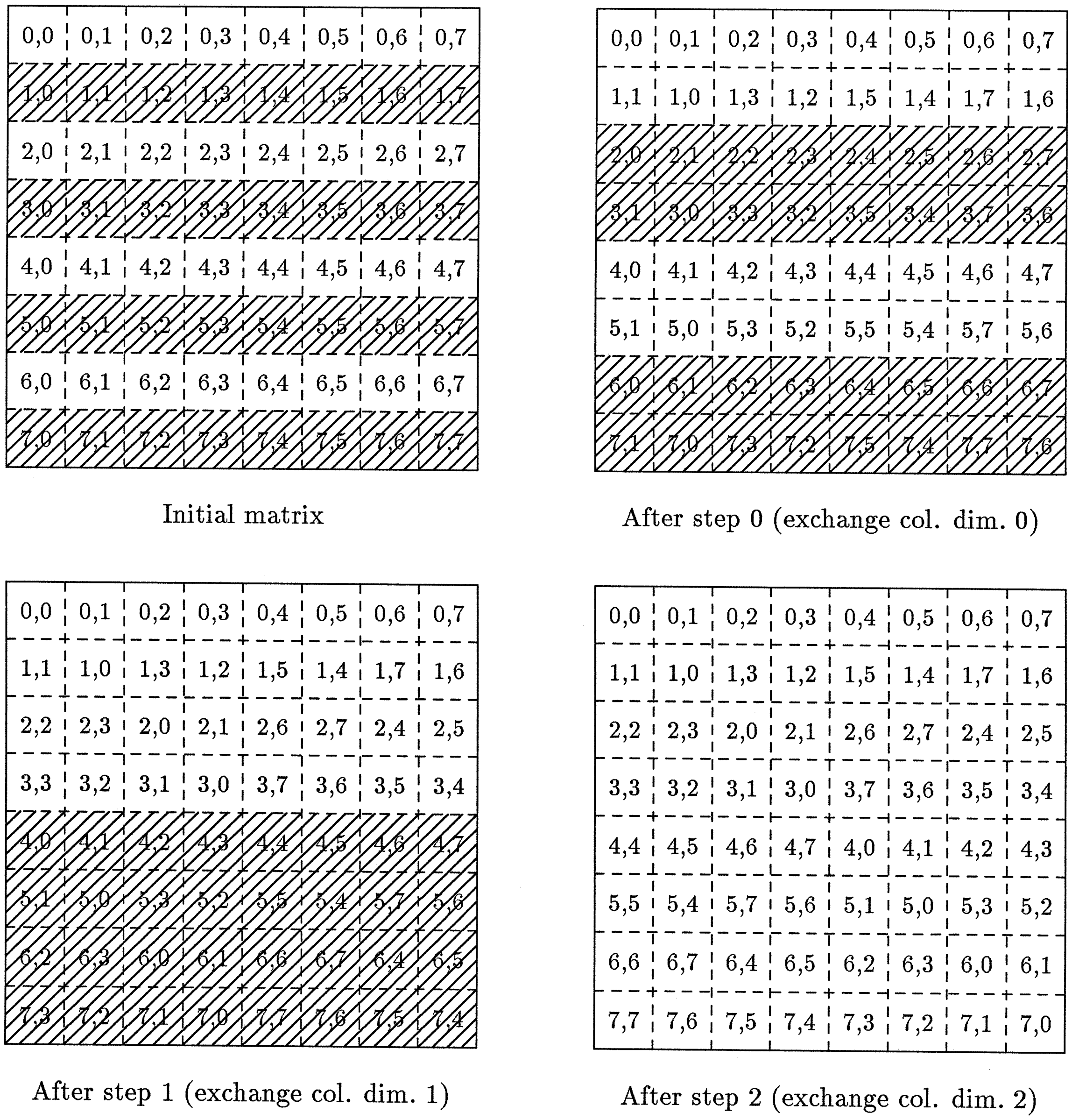


Figure 2: The three steps for the alignment of matrix A on a 6-cube.

The conditions of $\frac{PQ}{N} \geq \frac{n}{2}$ and $\frac{RQ}{N} \geq \frac{n}{2}$ are required, since both algorithms described above assume that the data set can be split into at least $\frac{n}{2}$ parts. A tight bound for the alignment when $P, Q, R < \sqrt{N}$ is $\max(p, r)$.

3.3.3 Multiplication

Lemma 7 [8] *A lower bound for the data transfer time of the matrices A and B during multiplication is $\lceil \frac{\max(P,R)Q}{\frac{n}{2}N} \rceil (\sqrt{N}-1)$ for $P, Q, R \geq \sqrt{N}$.*

Proof: Every processor needs to receive $\frac{PQ}{N}$ elements of A from each of $(\sqrt{N}-1)$ processors. The lower bound for this *all-to-all broadcasting* within row subcubes is $\lceil \frac{PQ}{\frac{n}{2}N} \rceil (\sqrt{N}-1)$ [8]. But, since all row subcubes perform the same communication and are fully utilized for this lower bound the subcube lower bound is also the total lower bound by Lemma 2. The bound for B is derived similarly, and since the set of dimensions used for the broadcasting of A and B are disjoint the lemma follows. ▮

For the multiplication phase the binary-reflected Gray code exchange sequence accomplishes an *all-to-all broadcasting* [8] within columns for B, and within rows for A. Any sequence with this property applied to both A and B in the same order is acceptable. The exchange sequence $\mathcal{S}(\frac{n}{2}, s)$, $1 \leq s \leq \frac{n}{2}-1$, is as appropriate as $\mathcal{S}(\frac{n}{2}, 0)$. It follows that $\frac{n}{2}$ pairs of blocks can be exchanged concurrently, with $\frac{n}{2}$ cube dimensions along each axis of the matrices. The total data transfer time for the multiplication phase is $\lceil \frac{P}{\sqrt{N}} \rceil \lceil \frac{Q}{\frac{n}{2}\sqrt{N}} \rceil (\sqrt{N}-1)$ for the matrix A and $\lceil \frac{R}{\sqrt{N}} \rceil \lceil \frac{Q}{\frac{n}{2}\sqrt{N}} \rceil (\sqrt{N}-1)$ for B, for multiplication performed on a $\sqrt{N} \times \sqrt{N}$ processor Boolean cube. When $P, R \geq \sqrt{N}$ and Q is a multiple of $\frac{n}{2}\sqrt{N}$, the bound is tight.

For the rank $\frac{Q}{\lambda}$ algorithm level-one block column m of A multiplies block row m of B. All level-zero blocks of level-one block m of A and B are subject to the exchange sequence $\mathcal{S}(\frac{n}{2}, m)$. Let $A(k, \ell, m)$ be the level-zero block assigned to processor (k, ℓ) of level-one block m of matrix A. The multiplication phase for level-one block column m of A and block row m of B involves the data motion defined by

$$A(k, \ell, m) \leftarrow A(k, \ell \oplus 2^{\alpha(t, \frac{n}{2}, m)}, m), \ \forall m \in \mathcal{Z}_\lambda, \forall k, \ell \in \mathcal{Z}_{\sqrt{N}} \text{ concurrently,}$$

$$B(k, \ell, m) \leftarrow B(k \oplus 2^{\alpha(t, \frac{n}{2}, m)}, \ell, m), \ \forall m \in \mathcal{Z}_\lambda, \forall k, \ell \in \mathcal{Z}_{\sqrt{N}} \text{ concurrently.}$$

The index for time, t, ranges from 1 to $\sqrt{N}-1$. Note that the communication for exchanges of A and B can be performed concurrently. Moreover, since $\alpha(t, \frac{n}{2}, m_1) \neq \alpha(t, \frac{n}{2}, m_2), m_1 \neq m_2$ for all t, the communication can be performed concurrently also for all $m \in \mathcal{Z}_\lambda$. In any communication step all cube dimensions are used for $\lambda = \frac{n}{2}$.

3.3.4 The Algorithm

In the pseudo code below we express the concurrency in the communication during alignment as well as multiplication. The algorithm MACC (Multiplication using All Channels Concurrently) restricts the communication to be confined to within subcubes. For the concurrent alignment the local data set is divided into as many sets as there are dimensions in the alignment operation. In the algorithm below this division is based on the level-one blocks. This division may yield too few sets if $\lambda < \frac{n}{2}$, and a different strategy for partitioning the local data sets preferred. The data sets that are permuted according to the different exchange sequences during the alignment define clusters of level-one blocks. All blocks in a cluster are exchanged concurrently. Clusters are treated sequentially. The number of blocks in a cluster is equal to $||k||$ for A, except for the last cluster that has $\lambda \bmod ||k||$ blocks if $\lambda \bmod ||k|| \neq 0$. Hence, if $||k|| = 1$ there are λ clusters with a single level-one block in each. One exchange operation suffices for each block. If $\lambda = ||k|| = \frac{n}{2}$ (i.e., $k = \sqrt{N} - 1$), then there is only one cluster of $\frac{n}{2}$ blocks, each of which requires $\frac{n}{2}$ exchange operations. Each block in a cluster has its unique exchange sequence.

Algorithm MACC

```
forall $k, \ell \in \mathcal{Z}_{\sqrt{N}}$ do
/* Alignment phase: */
   do concurrently for (1) and (2)
      (1) for $\gamma^r := 0$ to $||k|| - 1$ do    /* Loop through all the $||k||$ aligned dimensions. */
            for $k^r := 0$ to $\lceil \frac{\lambda}{||k||} \rceil - 1$ do    /* Loop through all clusters. */
               forall $\epsilon \in \mathcal{Z}_{||k||}$ do    /* Concurrently for all blocks of the same cluster. */
                  if $\epsilon + k^r||k|| \in \mathcal{Z}_\lambda$ then    /* Special care for the last cluster. */
                     $A(k, \ell, \epsilon + k^r||k||) \leftarrow A(k, \ell \oplus 2^{e(\mathcal{D}(k),(\epsilon+\gamma^r) \bmod ||k||)}, \epsilon + k^r||k||)$
                  endif
               endforall $\epsilon$
            endfor $k^r$
         endfor $\gamma^r$
      (2) for $\gamma^c := 0$ to $||\ell|| - 1$
            for $k^c := 0$ to $\lceil \frac{\lambda}{||\ell||} \rceil - 1$ do
               forall $\epsilon \in \mathcal{Z}_{||\ell||}$ do
                  if $\epsilon + k^c||\ell|| \in \mathcal{Z}_\lambda$ then
                     $B(k, \ell, \epsilon + k^c||\ell||) \leftarrow B(k \oplus 2^{e(\mathcal{D}(\ell),(\epsilon+\gamma^c) \bmod ||\ell||)}, \ell, \epsilon + k^c||\ell||)$
                  endif
               endforall $\epsilon$
            endfor $k^c$
         endfor $\gamma^c$
   enddo
/* Multiplication phase: */
   for $t := 0$ to $\sqrt{N} - 1$
      if $t \neq 0$ then
         /* Concurrent communication for all $\lambda$ pairs. */
         forall $m \in \mathcal{Z}_\lambda$
            do concurrently for (1) and (2)
               (1) $A(k, \ell, m) \leftarrow A(k, \ell \oplus 2^{\alpha(t, \frac{n}{2}, m)}, m)$
               (2) $B(k, \ell, m) \leftarrow B(k \oplus 2^{\alpha(t, \frac{n}{2}, m)}, \ell, m)$
            enddo
         endforall $m$
      endif
      for $m := 0$ to $\lambda - 1$ /* Block inner-product. */
         $C(k, \ell, m) \leftarrow C(k, \ell, m) + A(k, \ell, m) * B(k, \ell, m)$
      endfor $m$
```

```
    endfor $t$
endforall $k, \ell$
```

In the alignment phase, the loop indexed by k^r enumerates the clusters for processor row k. The loop indexed by γ^r scans through the dimensions included in the alignment for processor row k. The concurrency for all the blocks of the same cluster is explored by the loop indexed by ϵ. The if-statement following it makes sure that $\epsilon + k^r||k||$ is a valid level-one block. It is required when $\lambda \bmod ||k|| \neq 0$. For the multiplication phase it is important that the loop over the level-one blocks is inside the loop for the level-zero blocks (t). Excess data motion is otherwise required. The algorithm above is optimal (within a factor of two for the alignment phase) for $P, R \geq \sqrt{N}$ and Q being multiple of $\frac{n}{2}\sqrt{N}$ given the data assignment, and offers a reduction in communication complexity by a factor of $\frac{n}{2}$ compared to the naive block algorithm.

Theorem 2 *The data transfer time for algorithm MACC with concurrent communication on all ports of a Boolean n-cube is* $\lceil \frac{\max(P,R)}{\sqrt{N}} \rceil (\frac{n}{2} + (\sqrt{N} - 1)) \lceil \frac{Q}{\frac{n}{2}\sqrt{N}} \rceil$, *which is optimal within a small constant factor with the operands distributed uniformly over the processors configured as a product of two* $\frac{n}{2}$*-cubes,* $P = R$ *and* $P, Q, R \geq \sqrt{N}$.

For the case $P, Q, R < \sqrt{N}$ algorithm MACC has an alignment complexity of $\frac{n}{2}$, which can be modified to $\max(p, r)$ by performing the alignment only on the rows/columns to which data is allocated. The communication complexity during multiplication can be reduced to $\max(P, Q, R) - 1$ by limiting the exchange sequence to the set of processors to which data is allocated. The modification consists in replacing the upper loop bound for t by $T = \max(\min(P, \sqrt{N}), \min(Q, \sqrt{N}), \min(R, \sqrt{N}))$.

4 Non-square machines

In the previous section we assumed that the number of cube dimensions along both matrix axes was the same ($n_0 = n_1 = \frac{n}{2}$). With n_0 cube dimensions along axis zero, and n_1 dimensions along axis one, a cube with $2\max(n_0, n_1)$ dimensions is emulated by introducing $\max(n_0, n_1)$ virtual dimensions along the axis with $\min(n_0, n_1)$ cube dimensions. Let $|n_0 - n_1| = \beta$, then 2^β virtual processors are assigned to each processor along the axis with the fewest number of processors [9]. For each communication between processors along the axis with the fewest processors, there are 2^β exchange steps between processors along the other axis. Communication along virtual processor dimensions are local memory moves, and can be avoided with suitable address calculation.

We illustrate the multiplication of square matrices on a machine with $n_0 \neq n_1$ for the case $n_0 = 2$ and $n_1 = 1$. For the illustration one level-zero block per processor is assumed. Hence, $vp0^c = 1 + vp0^r$. Figure 3 illustrates the initial level-zero block assignment. Dashed lines define physical processor boundaries, and dotted lines virtual processor boundaries for the emulation of a 4-cube with four processors along each axis.

The alignment of the matrix A is along axis one. It is performed as if n_1 dimensions were assigned to each axis. If $n_0 > n_1$ then blocks of 2^β processor rows are aligned in the

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Initial allocation of matrix A

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

Initial allocation of matrix B

Figure 3: Initial allocation of level-zero square blocks for $P \times P$ matrices on a 3-cube.

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,2	2,3	2,0	2,1
3,2	3,3	3,0	3,1

Allocation of matrix A
after alignment

0,0	1,1	2,2	3,3
1,0	0,1	3,2	2,3
2,0	3,1	0,2	1,3
3,0	2,1	1,2	0,3

Allocation of matrix B
after alignment

Figure 4: Square level-zero block allocation after alignment for $P \times P$ matrices on a 3-cube.

same way, i.e., the alignment is performed using $k'' = 2^{\beta}\lfloor \frac{k}{2^{\beta}} \rfloor$ as the row index for alignment when the column index refers to the virtual processor (block) level. If $n_1 > n_0$, then the row index $k'' = k$ is used for the alignment of A at the virtual processor level. The alignment of the matrix B is performed as if there were n_0 dimensions assigned to each axis. The alignment for the case where $n_0 = 2$ and $n_1 = 1$ is shown in Figure 4.

The lower bound for the alignment of matrices A and B can be shown to be the same as in Lemma 5 for $n_0, n_1 \geq 1$ and $P, Q, R \geq \sqrt{N}$. The optimal time for alignment confined to row/column subcubes is $\lceil \lceil \frac{P}{N_0} \rceil \lceil \frac{Q}{N_1} \rceil / n_1 \rceil n_1$ for A and $\lceil \lceil \frac{R}{N_1} \rceil \lceil \frac{Q}{N_0} \rceil / n_0 \rceil n_0$ for B by Lemma 3, or Lemma 2. Hence, the subcube alignment is sub-optimal by at most a factor of two also in the case $n_0 \neq n_1$.

The matrix multiplication phase is shown in Figure 5 for $\lambda = 1$. The blocks being multiplied have the same shading. Note that in the multiplication phase the blocks being exchanged are always of the shape $2^{vp0^r} \times 2^{vp0^c}$ for both A and B.

To use all the cube dimensions during multiplication the matrices A and B are divided into λ blocks, as in the case $n_0 = n_1$, with $\lambda = \max(n_0, n_1)$. Without loss of generality, we assume $n_0 \geq n_1$. For the mth level-one block row of B, the exchange sequence follows $S(n_0, m)$, $m \in \mathcal{Z}_{n_0}$. Since there are n_0 different cube dimensions assigned to axis zero, all the λ exchange sequences can be executed concurrently. For the mth level-one block column of A, the exchange sequence is also $S(n_0, m)$. The dimensions $0, 1, \cdots, n_0 - n_1 - 1$ are inter-

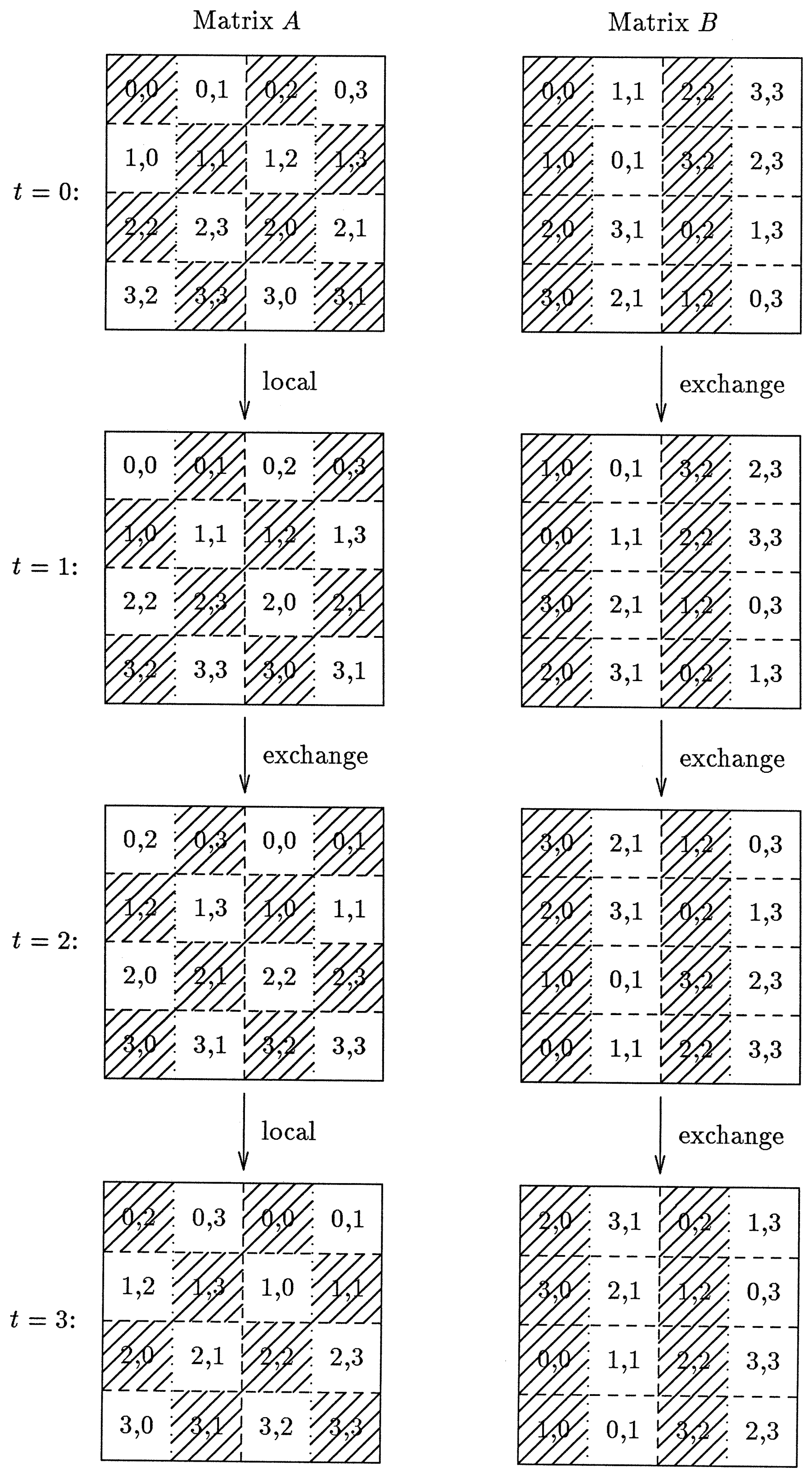


Figure 5: Square level-zero block multiplication for $P \times P$ matrices on a 3-cube.

preted as the virtual dimensions and the remaining n_1 dimensions as physical dimensions. For every step of these n_0 sequences of exchanges, there are n_1 exchanges between physical processors using all the n_1 cube dimensions, and $n_0 - n_1$ local data movements. As before, the local data movements can be avoided by address calculation.

We express the algorithm in terms of three subroutines: matrix-align, matrix-mpy-and-add, and matrix-exchange. The routines apply to matrices of arbitrary shapes, and arbitrarily factored Boolean cubes. Each processor stores 2^β blocks of each operand in the emulation of the virtual cube of $2\max(n_0, n_1)$ dimensions. v selects one of these 2^β blocks. For convenience, level-zero blocks are defined for virtual processors indexed by k', ℓ'. For instance, $A(k', \ell', m)$, $k', \ell' \in \mathcal{Z}_{\max(N_0,N_1)}$, is the level-zero block in virtual processor row k' and virtual processor column ℓ' of level-one block m. $G(t)$ is the binary-reflected Gray code of t. The subroutines are complicated by the address calculation for the local data movements avoided.

```
**Subroutine** matrix-align $(A, B, n_0, n_1, \lambda)$
  **forall** $k \in \mathcal{Z}_{N_0}, \ell \in \mathcal{Z}_{N_1}$ **do**
    $\beta \leftarrow |n_0 - n_1|$
    **for** $v := 0$ **to** $2^\beta - 1$ **do**
      **if** $n_0 \geq n_1$ **then**
        $k' \leftarrow k$
        $\ell' \leftarrow \ell 2^\beta + v$
        $k'' \leftarrow 2^\beta \lfloor \frac{k}{2^\beta} \rfloor$
        $\ell'' \leftarrow \ell$
      **else**
        $k' \leftarrow k 2^\beta + v$
        $\ell' \leftarrow \ell$
        $k'' \leftarrow k$
        $\ell'' \leftarrow 2^\beta \lfloor \frac{\ell}{2^\beta} \rfloor$
      **endif**
      **do** concurrently for (1) and (2)
        (1) **for** $\gamma^r := 0$ **to** $||k''|| - 1$ **do**
              **for** $k^r := 0$ **to** $\lceil \frac{\lambda}{||k''||} \rceil - 1$ **do**
                **forall** $\epsilon \in \mathcal{Z}_{||k''||}$ **do**
                  **if** $\epsilon + k^r||k''|| \in \mathcal{Z}_\lambda$ **then**
                    $A(k', \ell', \epsilon + k^r||k''||) \leftarrow A(k', \ell' \oplus 2^{e(\mathcal{D}(k''), (\epsilon+\gamma^r) \bmod ||k''||)}, \epsilon + k^r||k''||)$
                  **endif**
                **endforall** $\epsilon$
              **endfor** $k^r$
            **endfor** $\gamma^r$
        (2) **for** $\gamma^c := 0$ **to** $||\ell''|| - 1$
              **for** $k^c := 0$ **to** $\lceil \frac{\lambda}{||\ell''||} \rceil - 1$ **do**
                **forall** $\epsilon \in \mathcal{Z}_{||\ell''||}$ **do**
                  **if** $\epsilon + k^c||\ell''|| \in \mathcal{Z}_\lambda$ **then**
                    $B(k', \ell', \epsilon + k^c||\ell''||) \leftarrow B(k' \oplus 2^{e(\mathcal{D}(\ell''), (\epsilon+\gamma^c) \bmod ||\ell''||)}, \ell', \epsilon + k^c||\ell''||)$
                  **endif**
                **endforall** $\epsilon$
              **endfor** $k^c$
            **endfor** $\gamma^c$
            enddo
        endfor $v$
    endforall $i, j$
return
```

In the subroutine above, real processors are indexed by k, ℓ while virtual processors are indexed by k', ℓ'. For $n_0 > n_1$, the alignment of row k is determined by $k'' = 2^\beta \lfloor \frac{k}{2^\beta} \rfloor$, i.e., the least significant β bits of k are set to 0-bits.

```
Subroutine matrix-mpy-and-add $(A, B, C, n_0, n_1, \lambda, t)$
    forall $k \in \mathcal{Z}_{N_0}, \ell \in \mathcal{Z}_{N_1}$ do
        $\beta \leftarrow |n_0 - n_1|$
        for $m := 0$ to $\lambda - 1$ /* Block inner-product. */
            for $v := 0$ to $2^\beta - 1$ do
                $x \leftarrow 2^m G(t \bmod 2^{\beta - m}) + G(\lfloor \frac{t}{2^{\max(n_0, n_1) - m}} \rfloor)$
                if $n_0 \geq n_1$ then
                    $k' \leftarrow k$
                    $\ell' \leftarrow \ell 2^\beta + v$
                    $C(k', \ell', m) \leftarrow C(k', \ell', m) + A(k', \ell' \oplus (k' \bmod 2^\beta) \oplus x, m) * B(k', \ell', m)$
                else
                    $k' \leftarrow k 2^\beta + v$
                    $\ell' \leftarrow \ell$
                    $C(k', \ell', m) \leftarrow C(k', \ell', m) + A(k', \ell', m) * B(k' \oplus (\ell' \bmod 2^\beta) \oplus x, \ell', m)$
                endif
            endfor $v$
        endfor $m$
    endforall $k, \ell$
return
```

In the subroutine above, x keeps track of all the local exchanges. For $m = 0$, x at time t can be derived as

$$\bigoplus \left\{ 2^{\alpha(t', \max(n_0, n_1), 0)} \mid 1 \leq t' \leq t, \alpha(t', \max(n_0, n_1), 0) < \beta \right\} = G(t).$$

where $\bigoplus$ applied to the set of numbers means an exclusive-or operation on all the elements in the set. For an arbitrary m, x at time t becomes

$$\bigoplus \left\{ 2^{\alpha(t', \max(n_0, n_1), m)} \mid 1 \leq t' \leq t, \alpha(t', \max(n_0, n_1), m) < \beta \right\}$$
$$= 2^m G(t \bmod 2^{\beta - m}) + G\left(\left\lfloor \frac{t}{2^{\max(n_0, n_1) - m}} \right\rfloor \right).$$

The local alignment for row k' is $k' \bmod 2^\beta$. For $n_0 > n_1$, an exclusive-or of x and $k' \bmod 2^\beta$ with the column index of A guarantees the correct interaction between the corresponding blocks of A and B in the same real processor.

Subroutine matrix-exchange $(A, B, n_0, n_1, \lambda, t)$

```
    forall $k \in \mathcal{Z}_{N_0}, \ell \in \mathcal{Z}_{N_1}$ do
        $\beta \leftarrow |n_0 - n_1|$
        for $v := 0$ to $2^\beta - 1$ do
            if $n_0 \geq n_1$ then
                $k' \leftarrow k$
                $\ell' \leftarrow \ell 2^\beta + v$
                forall $m \in \mathcal{Z}_\lambda$ concurrently for (1) and (2) do
                    (1) if $\alpha(t, n_1, m) \geq \beta$ then
                            $A(k', \ell', m) \leftarrow A(k', \ell' \oplus 2^{\alpha(t,n_1,m)}, m)$
                        endif
                    (2) $B(k', \ell', m) \leftarrow B(k' \oplus 2^{\alpha(t,n_0,m)}, \ell', m)$
                endforall $m$, (1), (2)
            else
                $\ell' \leftarrow \ell$
                $k' \leftarrow k 2^\beta + v$
                forall $m \in \mathcal{Z}_\lambda$ concurrently for (1) and (2) do
                    (1) if $\alpha(t, n_0, m) \geq \beta$ then
                            $B(k', \ell', m) \leftarrow B(k' \oplus 2^{\alpha(t,n_0,m)}, \ell', m)$
                        endif
                    (2) $A(k', \ell', m) \leftarrow A(k', \ell' \oplus 2^{\alpha(t,n_1,m)}, m)$
                endforall $m$, (1), (2)
            endif
        endfor $v$
    endforall $k, \ell$
return
```

The complete matrix multiplication algorithm for arbitrary n_0 and n_1 is:

Algorithm MACCG

```
call matrix-align($A, B, n_0, n_1, \max(n_0, n_1)$)
call matrix-mpy-and-add($A, B, C, n_0, n_1, \max(n_0, n_1), 0$)
for $t := 1$ to $2^{\max(n_1,n_0)} - 1$ do
    call matrix-exchange($A, B, n_0, n_1, \max(n_0, n_1), t$)
    call matrix-mpy-and-add($A, B, C, n_0, n_1, \max(n_0, n_1), t$)
endfor $t$
```

The configuration of the set of processors has no effect on the number of element transfers in sequence for the alignment, considering only higher order terms. The blocks being exchanged in the multiplication phase are of size $\lceil \frac{P}{N_0} \rceil \times \lceil \frac{Q}{\lambda N_1} \rceil$ for the matrix A and $\lceil \frac{Q}{\lambda N_0} \rceil \times \lceil \frac{R}{N_1} \rceil$ for B. The total data transfer time during the multiplication phase is $\lceil \frac{P}{N_0} \rceil \times \lceil \frac{Q}{\lambda N_1} \rceil (N_1 - 1) \approx \frac{PQ}{\lambda N_0}$ for A and $\lceil \frac{Q}{\lambda N_0} \rceil \times \lceil \frac{R}{N_1} \rceil (N_0 - 1) \approx \frac{QR}{\lambda N_1}$ for B.

Theorem 3 *The communication complexity for the multiplication of a $P \times Q$ matrix A and a $Q \times R$ matrix B on a Boolean n-cube configured as an $N_0 \times N_1$ cube is at most* $\max(\lceil \lceil \frac{P}{N_0} \rceil \lceil \frac{Q}{N_1} \rceil / n_1 \rceil n_1, \lceil \lceil \frac{Q}{N_0} \rceil \lceil \frac{R}{N_1} \rceil / n_0 \rceil n_0) + \max(\lceil \frac{P}{N_0} \rceil \lceil \frac{Q}{n_1 N_1} \rceil (N_1 - 1), \lceil \frac{Q}{n_0 N_0} \rceil \lceil \frac{R}{N_1} \rceil (N_0 - 1))$.

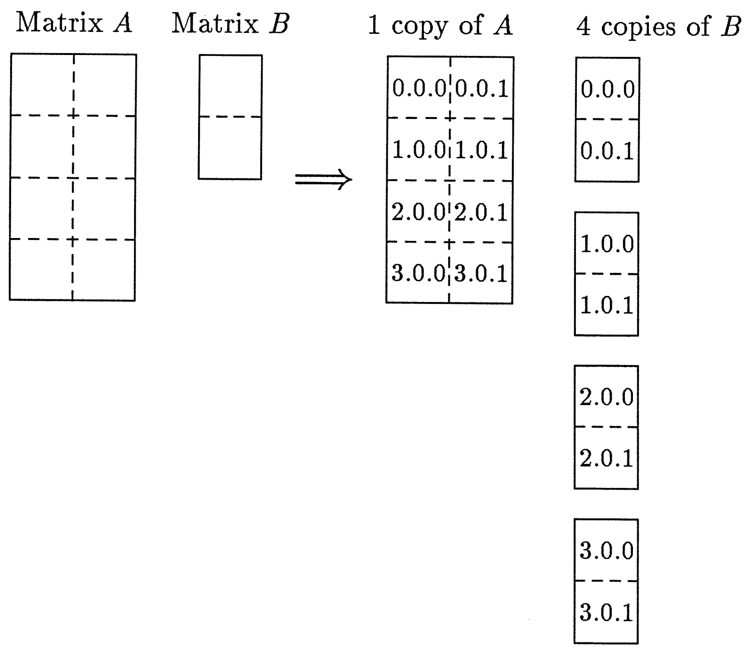


Figure 6: Partitioning of rectangular matrices with copy/reduction, $R \le Q \le P$

For the multiplication phase with $N = N_0 \times N_1$ constant, the optimum values of (N_0, N_1) should be chosen such that $\frac{N_0 n_0}{N_1 n_1} \approx \frac{P}{R}$ is satisfied for $P \geq N_0$ and $R \geq N_1$. If $P < N_0$ and $R < N_1$, then the optimum aspect ratio is 1. This aspect ratio minimizes the length of the longest processor axis. If N_0 and N_1 can be chosen independently, then the optimum value of $N_0 = \max(P, Q)$ and of $N_1 = \max(Q, R)$.

5 Matrices of arbitrary sizes

If $P < N_0$ and $R < N_1$, then the processor utilization during multiplication can be improved by parallelizing the loop on Q (index t). The number of matrix kernels that can be executed concurrently is $\gamma = \min(Q, \lceil \frac{N}{PR} \rceil)$. The multiplication is represented as rank $\frac{Q}{\gamma}$ updates. A third axis is introduced for the encoding of the different, concurrent matrix multiplications. The length of this axis is γ. A distribution of the different block columns of A and block rows of B to distinct subcubes precedes the multiplication, and a reduction succeeds it. For each rank $\frac{Q}{\gamma}$ multiplication a copy operation as described in the previous paragraph may be necessary. Figure 6 gives an example where both copy and reduction is required. Blocks are identified by $x.y.z$. The third index labels the third axis. Blocks with identical labels are multiplied together. Reduction is required for all blocks of same x and y and different z's. The number of copies of blocks of size $\frac{Q}{\gamma}$ is $\lceil \frac{R\gamma}{Q} \rceil$ for A and $\lceil \frac{P\gamma}{Q} \rceil$ for B. The number of processors performing arithmetic operations is $PR\gamma$. The matrix product requires $2\frac{Q}{\gamma}$ local arithmetic operations plus PR independent reduction operations on sets of size γ. The arithmetic time is $2\frac{Q}{\gamma} + \log \gamma$.

If $P > Q$ and $Q < N_0$, then only Q out of $\min(P, N_0)$ processors along axis zero perform arithmetic operations at any time, and $\min(P, N_0) > Q$ steps are required for the multiplication of A with a column of B. Similarly, if $R > Q$ and $Q < N_1$, then only Q out

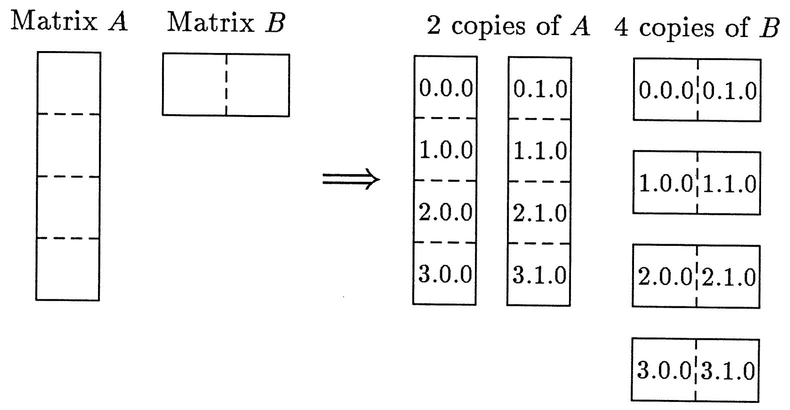


Figure 7: Partitioning of rectangular matrices with copy, $Q \leq R \leq P$

of $\min(R, N_1)$ processors along axis one perform arithmetic operations at any given time. To increase the processor utilization and shorten the length of the loop on t the matrix A is replicated such that the number of instances along axis one is $\lceil \frac{R}{Q} \rceil$, and the matrix B is replicated such that the number of instances along axis zero is $\lceil \frac{P}{Q} \rceil$. The range for the loop on t is $0 - Q - 1$. The replication of the operands assures that $\min(P, N_0) \times \min(R, N_1)$ processors are used effectively for multiplication and communication. Figure 7 illustrates the replication of A and B for the case $\frac{P}{Q} = 4$ and $\frac{R}{Q} = 2$.

6 Conclusion

We have presented a matrix multiplication algorithm with linear speedup of the arithmetic and optimal use of the communication capacity of Boolean cube networks for $\max(n_0, n_1)$ elements per processor of the common axis, where n_0 and n_1 are the number of cube dimensions allocated to the two matrix axes. The communication complexity for the multiplication of a $P \times Q$ matrix A and a $Q \times R$ matrix B on a Boolean n-cube configured as a $N_0 \times N_1$ cube is $\max(\lceil \lceil \frac{P}{N_0} \rceil \lceil \frac{Q}{N_1} \rceil / n_1 \rceil n_1, \lceil \lceil \frac{Q}{N_0} \rceil \lceil \frac{R}{N_1} \rceil / n_0 \rceil n_0) + \max(\lceil \frac{P}{N_0} \rceil \lceil \frac{Q}{n_1 N_1} \rceil (N_1 - 1), \lceil \frac{Q}{n_0 N_0} \rceil \lceil \frac{R}{N_1} \rceil (N_0 - 1))$. The arithmetic complexity is $\lceil \frac{P}{N_0} \rceil \lceil \frac{R}{N_1} \rceil \max(\min(P, N_0), \min(R, N_1), Q)$. We also give algorithms with arithmetic complexity $\lceil \frac{P}{N_0} \rceil \lceil \frac{R}{N_1} \rceil Q$ For $P < N_0$ and/or $R < N_1$ at an increased communication cost. Parallelization of the common axis (Q) is also described.

The algorithms presented here use constant storage. If the storage is sufficiently large to allow all-to-all broadcasting within rows and columns to be performed by spanning tree algorithms then $\log N$ steps suffice, and the communications bandwidth can be fully utilized [8]. But, the storage requirement per processor is proportional to $\frac{P+R}{\sqrt{N}} Q$, i.e., a factor of $\sqrt{N}$ higher than for the algorithms presented here.

It is also possible to generalize Cannon's algorithm such that the full communications bandwidth of the cube is used. The generalization can made since there exists n edge-disjoint Hamiltonian cycles in a $2n$-cube, [4], [1], [15]. The matrices are assigned to the processors by a two-level partitioning, as in the algorithms described here. The storage per processor is the same as for our algorithms. However, the local control at each processor is more complicated, because the known method for constructing the n edge-disjoint Hamiltonian cycles is quite complex (double recursion), and the path encoding complicated.

Acknowledgement We thank John Smagula for proofreading the manuscript. This work has been supported in part by the Office of Naval Research under Grant N00014-89-J-1906.

References

[1] Jacques Aubert and Bernadette Schneider. Decomposition de la somme cartesienne d'un cycle et de l'union de deux cycles hamiltoniens en cycles hamiltoniens. *Discrete Mathematics*, 38:7–16, 1982.

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

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

[4] Marsha Foregger. Hamiltonian decompositions of products of cycles. *Discrete Mathematics*, 24:251–260, 1978.

[5] Geoffrey C. Fox, S.W. Otto, and A.J.G. Hey. Matrix algorithms on a hypercube i: Matrix multiplication. Technical Report Caltech Concurrent Computation Project Memo 206, California Institute of Technology, dept. of Theoretical Physics, October 1985.

[6] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987. (Tech. Rep. YALEU/DCS/RR-361, Yale Univ., New Haven, CT, January 1985).

[7] S. Lennart Johnsson. Data parallel programming and basic linear algebra subroutines. In John R. Rice, editor, *Proceedings of the IMA Workshop on Mathematical Aspects of Scientific Software*, pages 183–196. Springer Verlag, 1987. YALE/DCS/RR-584, September 1987.

[8] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. Technical Report YALEU/DCS/RR-500, Dept. of Computer Science, Yale Univ., New Haven, CT, November 1986. Revised November 1987, YALEU/DCS/RR-610. To appear in IEEE Trans. Computers.

[9] S. Lennart Johnsson and Ching-Tien Ho. Algorithms for multiplying matrices of arbitrary shapes using shared memory primitives on a Boolean cube. Technical Report YALEU/DCS/RR-569, Dept. of Computer Science, Yale Univ., New Haven, CT, October 1987. Revision of YALE/DCS/RR-530. Presented at the ARMY Workshop on Medium Scale Parallel Processors, Stanford Univ., January 1986.

[10] S. Lennart Johnsson and Ching-Tien Ho. Shuffle permutations on Boolean cubes. Technical Report YALEU/DCS/RR-653, Department of Computer Science, Yale University, October 1988.

[11] S. Lennart Johnsson and Peggy Li. Solutionset for AMA/CS 146. Technical Report 5085:DF:83, California Institute of Technology, May 1983.

[12] C. L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM TOMS*, 5(3):308–323, September 1979.

[13] E M. Reingold, J Nievergelt, and N Deo. *Combinatorial Algorithms.* Prentice-Hall, Englewood Cliffs. NJ, 1977.

[14] Quentin F. Stout and Bruce Wager. Passing messages in link-bound hypercubes. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987.* Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[15] Alan Wagner, 1988. Personal communication.