Yale University
Department of Computer Science

Matrix Multiplication on Boolean Cubes Using Shared Memory Primitives

Ching-Tien Ho and S. Lennart Johnsson

YALEU/DCS/TR-636
July 1988

This work has in part been supported by the Office of Naval Research under contract N00014-86-K-0564. Approved for public release: distribution is unlimited.

† A preliminary version of this paper was presented at the Third Conference on Hypercube Concurrent Computers and Applications, January, 1988, Pasadena, CA. A revised edition of TR623.

Matrix Multiplication on Boolean Cubes Using Shared Memory Primitives

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

Abstract. Generic communication primitives can be used for many algorithms on Boolean cubes. Here we focus on expressing such primitives and algorithms for matrix multiplication in terms of shared memory type programming primitives. All processors share the same global address space. The communication primitives realize nearest-neighbor communication and global operations such as broadcasting from one processor to a set of processors, the reverse operation of plus-reduction, and matrix transposition (dimension permutation). We consider both the case where communication is restricted to one processor port at a time and the case of concurrent communication on all processor ports. The communication algorithms are provably optimal within a factor of two. We describe both constant storage algorithms and algorithms with reduced communication time but with a storage need proportional to the number of processors and the matrix sizes (for a one-dimensional partitioning of the matrices). The choice of the described matrix multiplication algorithms depends on machine size relative to the matrix sizes, the matrix shapes, and the architectural parameters of the machine.

1 Introduction

Making good use of the locality that exists in many computations is a necessity in high performance computation and current technologies. The communications bandwidth at the chip boundary is one to two orders of magnitude less than the on-chip communications capability in submicron MOS technologies. Similarly, the on-board communications capabilities are an order of magnitude higher than the bandwidth at the board edge. The communications characteristics of current technologies suggest multiprocessor architectures in which processors with their own storage modules are interconnected by a network of some type. Currently, the dominating interconnection networks are two-dimensional meshes, n-dimensional Boolean cubes, and butterfly networks. In a mesh of two or more dimensions, and Boolean n-cubes, there is also a locality in the network. For instance, in a Boolean cube there are $\binom{n}{l}$ nodes at distance l from any node. The number of local references per remote reference for optimum use of the locality of reference for many computations scales as $\alpha\frac{1}{d}(\frac{M}{\beta})^{\frac{1}{d}}$, or $\alpha \log \frac{M}{\beta}$, where α is a constant, β is the number of variables per object (assuming that all objects require the same number of variables), and d is the dimensionality of the data structure. The range of values for β is typically 1 - 1000. Examples of computations of the former type, are many matrix operations and relaxation. Sorting and Fast Fourier Transform computations have the latter behavior [7]. The significance of

[1]Also with Dept. of Electrical Engineering, Yale Univ., and Thinking Machines Corp., Cambridge, MA 02142.

Function	Local Storage					
	Min.	1k	32k	1M	32M	1G
Matrix multiplication	$\frac{1}{2}$	10	50	300	1600	10000
3-D relaxation, $\beta = 8$	$\frac{1}{6}$	0.8	$\frac{8}{3}$	8	$\frac{28}{5}$	$\frac{256}{3}$
FFT	1	10	16	22.5	29	35

Table 1: The number of local references per remote reference as a function of local storage.

locality is illustrated in Table 1. For a chip, the reduction in bandwidth requirement at the chip boundary is one to two orders of magnitude for optimum use of locality of reference, and at the board level the reduction is up to three orders of magnitude.

In order to take full advantage of locality in an architecture with a network like a Boolean n-cube, it is important to map the data into the storage of the individual processors such that not only are a majority of the references local, but also when references are made to the storage of other processors the communication time is minimized. Clearly, it is desirable to minimize contention in the network. If all communication between processors is at unit distance, then there will be no contention. For some computations there exist algorithms and data allocation schemes such that the computation only implies nearest neighbor communication, even though the complete computation requires global communication. For communication of large data volumes in networks with multiple paths between source and destination it is also of interest to make efficient use of this capability.

For architectures with the storage distributed among the processors it is of considerable interest to find generic global communication primitives that allow algorithms to be expressed concisely, yet have efficient implementations. The generic communication primitives can be viewed as part of the architecture, and implemented efficiently as part of the communication system. The programmer does not need to know about the topological properties of the network. Examples of generic global communication primitives are:

- Pairwise exchange;
- Rotation;
- Copy/Reduction;
- Permutation.

A global copy/reduction can be implemented using either of the first two primitives.

In this paper we focus on expressing generic communication primitives in programming primitives for a shared address space. Optimal routing and scheduling disciplines for the implementation of the primitives on Boolean n-cubes are given in [11,10,5] for two different communication capabilities of Boolean cube configured multiprocessors, namely communication on one port at a time for each processor, or all ports concurrently for every processor. The computation we have selected for illustration of a global programming model for distributed memory multiprocessors

is the multiplication of two matrices by algorithms of arithmetic complexity $O(PQR)$ for a $P \times Q$ matrix and a $Q \times R$ matrix. In [9] we showed how the choice of algorithm with respect to total execution time (arithmetic and communication) depends on the shapes of the matrices and the architectural parameters, such as arithmetic and communication speeds, communications overhead, and the number of processors relative to the matrix sizes.

The outline of the paper is as follows. In the next section we introduce the notation used throughout the paper, define the address map, and give a brief overview of the algorithms described in the paper. Section 3 introduces the generic communication primitives encapsulating the network features. In section 4, which is the main section of this paper, we show examples of algorithms based on emulating linear arrays or two-dimensional arrays in the Boolean cube. Algorithms making use of the unique properties of the Boolean cube network are also seen here. Our conclusions are summarized in Section 5.

2 Preliminaries

We consider the matrix operation $A \leftarrow C*D$, where all matrices are dense, C a $P \times Q$ matrix, D a $Q \times R$ matrix, and A a $P \times R$ matrix. Some of the parallel algorithms for the matrix multiplication that we consider are based on the emulation of a one dimensional array of processors. Others are based on the emulation of a two-dimensional array, some on the emulation of three dimensional arrays, and some are recursive in nature making direct use of the Boolean cube topology. For the emulation of two-dimensional meshes, the dimensions of the Boolean n-cube of $2^n = N$ nodes are factored into two groups such that n_1 dimensions are used for the encoding of row addresses, and $n_2 = n - n_1$ dimensions are used for the encoding of column addresses. The number of processors in a column is $N_1 = 2^{n_1}$ and the number of processors in a row is $N_2 = 2^{n_2}$. A three dimensional partitioning of the cube is made such that $N = N_1' \times N_2' \times N_3'$ ($2^{n_1'} \times 2^{n_2'} \times 2^{n_3'}$). It is well known that a lattice of lower dimensionality than n can be emulated preserving proximity on a Boolean n-cube, see for instance [8].

The matrices C and D, and the product matrix A are assumed to be distributed among all the processors in the same manner, except in the three-dimensional case. We assume $P = 2^p$, $Q = 2^q$ and $R = 2^r$. The row and column indices of the matrices require $p + q$ bits for C, $q + r$ bits for D, and $p + r$ bits for A. We further assume that the address field is partitioned as follows:

$$(row\ address | column\ address).$$

For algorithms based on the emulation of a *linear array*, the rows, or columns, are assigned to processors either *cyclicly*, or *consecutively* [8]. The address map is easily defined in terms of the address dimensions as follows:

$$Cyclic\ Row : (row\ \underbrace{address}_{rp} | column\ address),$$

$$Consecutive\ Row : (\underbrace{row\ address}_{rp} | column\ address),$$

$$Cyclic\ Column : (row\ address | column\ \underbrace{address}_{rp}),$$

Part.	Storage	Encoding	Processor address
column	consec.	binary	$(j_{q-1}j_{q-2}\cdots j_{q-n})$
		Gray	$(G(j_{q-1}j_{q-2}\cdots j_{q-n}))$
	cyclic	binary	$(j_{n-1}j_{n-2}\cdots j_0)$
		Gray	$(G(j_{n-1}j_{n-2}\cdots j_0))$
2-dim.	consec.	binary	$(i_{p-1}i_{p-2}\cdots i_{p-n_1}$ $j_{q-1}j_{q-2}\cdots j_{q-n_2})$
		Gray	$(G(i_{p-1}i_{p-2}\cdots i_{p-n_1})\|\|$ $G(j_{q-1}j_{q-2}\cdots j_{q-n_2}))$
	cyclic	binary	$(i_{n_1-1}i_{n_1-2}\cdots i_0$ $j_{n_2-1}j_{n_2-2}\cdots j_0)$
		Gray	$(G(i_{n_1-1}i_{n_1-2}\cdots i_0)\|\|$ $G(j_{n_2-1}j_{n_2-2}\cdots j_0))$

Table 2: Various ways of assigning matrix elements into processors.

$$\textit{Consecutive Column} : (\textit{row address}|\underbrace{\textit{column address}}_{rp}),$$

where rp denotes real processor addresses. The remaining dimensions of the address field define local memory addresses. For algorithms based on the emulation of a *two-dimensional array* the address map for *consecutive* partitioning of both rows and columns is

$$\textit{Consecutive Row/Column} : (\underbrace{\textit{row address}}_{rp}|\underbrace{\textit{column address}}_{rp}).$$

For the emulation of lattices the real processors addresses are encoded in a Gray code. Initially, an element of C and D is assigned to only one processor. Table 2 summarizes the address maps for one- or two-dimensional partitionings, *consecutive* or *cyclic* storage [8], and binary or Gray code encoding [12,8]. In the two-dimensional partitioning, each column (row) is assigned to N_1 (N_2) different processors. The row partitioning is obtained by replacing (j,q) by (i,p). By replacing (i,j,p,q) by (j,k,q,r) or (i,k,p,r), the processor assignment for the matrix element d_{jk} (of D) or a_{ik} (of A) is obtained.

For an algorithm based on a *three-dimensional array*, matrix C is partitioned as N_3' block columns and matrix D is partitioned into N_3' block rows. Each block column of C and each block row of D are further partitioned into $N_1' \times N_2'$ blocks. The resulting matrix A can be partitioned into a form of $N_1'2^x \times N_2'2^{n_3'-x}$ blocks, where x is an integer, $0 \le x \le n_3'$, with the same communication complexity. The address maps for the matrices C and D are

$$C : (\underbrace{\textit{row}}_{N_1'}\ \textit{address}|\underbrace{\textit{col}}_{N_3'}\underbrace{\textit{umn}}_{N_2'}\ \textit{address}),$$

$$D : (\underbrace{\textit{row}}_{N_3'}\ \underbrace{\textit{add}}_{N_1'}\textit{ress}|\underbrace{\textit{col}}_{N_2'}\textit{umn address}).$$

If the matrices C and D are initially assigned to an $N_1 \times N_2$ processor array, then some communications in the form of dimension permutations [5] are required to rearrange the data allocation for a matrix multiplication in which all three nested loops in a matrix multiplication algorithm (expressed in a conventional language) are parallelized. This communication has a data communication time that is of lower order than that for the matrix multiplication, except if there are very few elements per processor.

In matrix multiplication each element of C in column j interacts with every element of D in row j, and every element of D in row j interacts with every element of C in column j. This communication need is apparent, if the multiplication is expressed as a sequence of rank-1 updates. *All-to-all broadcasting*, or copy, is required in the subspaces defined by the column dimensions for rows of C, and in the subspaces defined by the row dimensions for the columns of D. Reduction is required for the inner products. The broadcasting operation can be performed as a sequence of rotations, which yields a linear time, constant storage algorithm. It can also be performed as a sequence of exchange operations in the different dimensions of the cube. For an exchange sequence in the form of a *binary-reflected* Gray code, a linear time, constant storage algorithm, is again obtained. For a one-dimensional partitioning of the matrices, the former is a one-dimensional version of Cannon's algorithm [1], and the latter a one-dimensional version of Dekel's algorithm [4]. By combining a doubling algorithm with the exchange algorithm, n_1 and n_2 communication steps suffice for *all-to-all broadcasting*. However, the storage requirement per processor (buffer size) doubles for every step. These two techniques can be combined [9]. In the two- and three-dimensional cases it is necessary to align the operands C and D in order to make full use of the processing capability. The address map defined above does not provide the necessary alignment. The alignment can be based on rotations, or more general, but still restricted permutations, such as dimension permutations. Matrix transposition is a particular form of dimension permutation that is used in some of the algorithms described in Section 4.

In the case that communication can take place concurrently on several ports of every processor the algorithms should be constructed to take advantage of this fact. We describe several such algorithms. One frequently used technique to create several concurrent communication operations for each processor, is to generate several communication sequences by performing the communication for different sequences in the same order of cube dimensions, cyclicly, but with different starting dimensions.

In the following, all algorithms are described in a Crystal-like notation [2,3]. Each instruction is defined as a function. By interpreting the first one, two, or three parameters as processor identifier(s) in the one-, two-, or three-dimensional partitioning cases, parallel codes for the algorithms are obtained. The communications are specified assuming a global address space. The processor indices are part of the global address. For a naive implementation of the communications, for instance by using a noncombining router, and without using multiple paths between pairs of processors, efficiency may be lost due to poor scheduling (collisions), or poor path selection (non-minimum path lengths, single paths). We expand the communication primitives (specification) into a sequence of nearest-neighbor communications, also described in the Crystal-like notation. Execution of the communication code replaces the high-level communication specification. The communication primitives we use are *all-to-all broadcasting* on a (sub)cube, *all-to-all reduction* (in a divide-and-conquer manner) [11], and matrix transposition (dimension permutation).

In the Crystal-like codes each function of l parameters may be optionally followed by an expression " **over** $\text{domain}_1 \times \text{domain}_2 \times \cdots \times \text{domain}_l$", where domain_i is the domain of the ith parameter. $[x, y]$, $y > x$, denotes the set of integers $\{x, x+1, \ldots, y\}$, and $[x, y)$ denotes $\{x, x+1, \ldots, y-1\}$. The statements enclosed between $\ll$ and $\gg$ form a conditional statement. For example,

$\ll cond_1 \rightarrow result_1,$
$\quad cond_2 \rightarrow result_2,$
$\quad$ **else** $\rightarrow result_3 \gg,$

reads as "if $cond_1$ then $result_1$, else if $cond_2$, then $result_2$, else $result_3$". $\backslash\!\!+[f(j)*g(j)|0 \leq j < x]$ denotes $\sum_{j=0}^{x-1}(f(j) * g(j))$. We use $c(i,j)$, $0 \leq i < P$, $0 \leq j < Q$, to denote the matrix element at the ith row and jth column of C. $d(j,k)$ and $a(i,k)$ are similarly defined. For matrices distributed over a set of processors, in our case a Boolean cube, it is more convenient to identify a matrix element by a processor address, and the relative indices of the local submatrix. lc, ld and la are used to denote the local submatrices of C, D, and A, respectively. The suffixes brd, row, col, txp and red in the following, are used to denote broadcasting, broadcasting along row, broadcasting along column, transposition and reduction, respectively. We use α and $\hat{\alpha}$ to distinguish between binary encoding and Gray code encoding of the processor id (pid), i.e., $\alpha = pid$ and $G(\hat{\alpha}) = pid$ or $\hat{\alpha} = G^{-1}(pid) = G^{-1}(\alpha)$, where G is the binary-reflected Gray code encoding function. For instance, if $pid = (100)$ then $\alpha = 4$ and $\hat{\alpha} = 7$. Note, that for a Gray code encoding of column blocks, the jth block column is in processor $pid = G(j) = G(\hat{\alpha})$. Some primitive functions used in the paper are defined below.

/* $nbr(\alpha, i)$ is the neighbor pid of processor α along cube dimension i. */
$nbr(\alpha, i) = \alpha \oplus 2^i,$
/* Shuffle (cyclic left-shift) u steps of t. */
$sh(u,t)$ **over** $[0:n) \times [0:N) = (t \bmod 2^{n-u})2^u + \lfloor \frac{t}{2^{n-u}} \rfloor,$
/* Unshuffle (cyclic right-shift) u steps of t. */
$ush(u,t)$ **over** $[0:n) \times [0:N) = sh(n-u, t)$
/* $G(t)$ is the binary-reflected Gray code of t. */
$G(t) = t \oplus \lfloor \frac{t}{2} \rfloor,$
/* G^{-1} is the inverse function of G. */
$G^{-1}(t) = \ll t = 0 \rightarrow 0,$
$\qquad$ **else** $\rightarrow t \oplus G^{-1}(\lfloor \frac{t}{2} \rfloor) \gg,$
/* $T(t)$ is the index of the tth transition bit in the Gray code
$\quad$ = the number of trailing 1's of t. */
$T(t) = \ll t \bmod 2 = 0 \rightarrow 0,$
$\qquad$ **else** $\rightarrow 1 + T(\lfloor \frac{t}{2} \rfloor) \gg .$

In the arithmetic and communication complexity expressions presented, we denote the communication packet size by B, the communication start-up time with τ, the time for the transmission of an element by t_c, and the time for an arithmetic operation by t_a. For the communication system we consider *one-port communication*, where communication can take place on only one port at a time, and *n-port communication*, for all ports on each processor can be used concurrently.

Model	Algorithm	Element transfers	min start-ups
one-port	SBT	$(N-1)M$	n
n-port	nRSBT	$\frac{(N-1)M}{n}$	n

Table 3: Communication complexity of all-to-all broadcasting on an n-cube with M elements per processor initially.

Model	Algorithm	Element transfers	min start-ups
one-port	SBT	$\frac{(N-1)M}{N}$	n
n-port	nRSBT	$\frac{(N-1)M}{nN}$	n

Table 4: Communication complexity of all-to-all reduction on an n-cube with M elements per processor initially.

3 Communication primitives

The communication routines we use for matrix multiplication on the Boolean cube are *all-to-all broadcasting*, *all-to-all reduction* and matrix transposition. All-to-all broadcasting and the reversed operation all-to-all reduction are described in detail in [11,13]. Matrix transposition with one-dimensional partitioning has the same communication pattern as *all-to-all personalized communication* [11], also known as a *complete exchange* [13]. With a two-dimensional square partitioning into $\sqrt{N} \times \sqrt{N}$ blocks, optimal algorithms are described in [10,13]. For the transposition of a matrix partitioned into $N_1 \times N_2$ blocks, one can combine the one-dimensional matrix transposition algorithm with the algorithm for the transposition of a two-dimensionally square-partitioned matrix. The communication complexities of various algorithms are summarized in Tables 3, 4 and 5. Note, that the complexity of the all-to-all reduction is the same as that of all-to-all broadcasting, if the number of elements per processor before the reduction is the same as the number of elements per processor after the broadcasting.

Model	Algorithm	Element transfers	min start-ups
one-port	SBT	$\frac{nM}{2}$	n
n-port	nRSBT	$\frac{M}{2}$	n

Table 5: Communication complexity of all-to-all personalized communication with M elements per processor initially.

3.1 One-dimensional Matrix Partitionings

All-to-all broadcasting in a linear array can be performed by rotation, or an exchange sequence. Such broadcasting algorithms are described directly in the multiplication algorithms presented in the next section. Two broadcasting algorithms using a doubling technique to reduce the number of communication steps from N to n are presented. All-to-all broadcasting based on N translated Spanning Binomial Trees (SBT's) [11] with *one-port communication* is defined as follows:

/* SBT broadcasting. */
/* Row direction, one-port, binary encoding. */
$lc\_brd1(\alpha, i, j', t)$ **over** $[0:N) \times [0:P) \times [0:2^t\frac{Q}{N}) \times [0:n] =$
$\ll t = 0 \rightarrow lc(\alpha, i, j'),$
$0 \leq j' < 2^{t-1}\frac{Q}{N} \rightarrow lc\_brd1(\alpha, i, j', t-1),$
/* Get from $(t-1)$th nbr and append. */
else $\rightarrow lc\_brd1(nbr(\alpha, t-1), i, j' - 2^{t-1}\frac{Q}{N}, t-1) \gg,$
/* Order the N blocks by *pid*. */
$lc\_brd(\alpha, i, j)$ **over** $[0:N) \times [0:P) \times [0:Q) = lc\_brd1(\alpha, i, j \oplus \alpha\frac{Q}{N}, n).$

Within the if-clause the first statement is the initial condition, the second statement defines the left half of the $P \times 2^t\frac{Q}{N}$ local submatrix as the local submatrix of the previous iteration, and the third statement defines the right half of the local submatrix as the local submatrix of the previous iteration of the neighbor processor along cube dimension $t-1$. All communications implied are nearest neighbor. Since the number of block columns doubles for each iteration (described by t), the domain of the parameter j' grows exponentially from $\frac{Q}{N}$ initially to Q at the end. The last function orders the N blocks by processor id. The "1" in the function name "*lc_brd*1" is introduced as an internal function for convenience. It describes the n nearest-neighbor communication steps in sequence and has a parameter t denoting the iteration. *lc_brd* is the broadcasting primitive and differs from the result of the internal function by a permutation of the N column blocks to the right order. In the function *lc_brd*1 the blocks received from a neighbor processor are appended to the end. A 1 appended to a function name in the following also denotes an internal function differing from the external function in a way similar to the above case.

For Gray code encoding, α is replaced by $\hat{\alpha}$, $nbr(\alpha, t-1)$ is replaced by $nbr(G(\hat{\alpha}), t-1)$ and *lc_brd* is redefined as:

/* Order the N blocks by $G^{-1}(pid)$. */
$lc\_brd(\hat{\alpha}, i, j)$ **over** $[0:N) \times [0:P) \times [0:Q) = lc\_brd1(\hat{\alpha}, i, (G(\lfloor\frac{jN}{Q}\rfloor) \oplus \hat{\alpha})\frac{Q}{N} + j \bmod \frac{Q}{N}, n).$

The converse operation to broadcasting is reduction. For an *all-to-all reduction*, different parts of the matrix are reduced to different processors. An *all-to-all reduction* algorithm for a Spanning Binomial Tree routing and *one-port communication* is

/* SBT reduction. */
/* Between columns, one-port, binary encoding. */
$la\_red1(\alpha, i, k', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{R}{2^t}) \times [0:n] =$
$\ll t = 0 \rightarrow la(\alpha, i, k'),$
/* Procs. in subcube 0 get sums of two left half submatrices. */
$\lfloor \frac{\alpha}{2^{n-t}} \rfloor \bmod 2 = 0 \rightarrow la\_red1(\alpha, i, k', t-1) + la\_red1(nbr(\alpha, n-t), i, k', t-1),$
/* Procs. in subcube 1 get sums of two right half submatrices. */
else $\rightarrow la\_red1(\alpha, i, k' + \frac{R}{2^t}, t-1) + la\_red1(nbr(\alpha, n-t), i, k' + \frac{R}{2^t}, t-1) \gg,$
$la\_red(\alpha, i, k')$ **over** $[0:N) \times [0:P) \times [0:\frac{R}{N}) = la\_red1(\alpha, i, k', n).$

The second statement of the if-clause defines the reduction for the processors in subcube 0 (with respect to cube dimension $n-t$). It sums up the left half of the local submatrix and the left half of the $(n-t)$th neighbor processor's local submatrix. The third statement of the if-clause defines the reduction for the processors in subcube 1. It sums up the two corresponding right half submatrices.

With *n-port communication*, all-to-all broadcasting based on N distinct translations of n Rotated Spanning Binomial Trees (nRSBT), Spanning Balanced n-Trees (SBnT) and n Edge-disjoint Spanning Binomial Trees (nESBT) [11] are all optimal within constant factors. Translation of a directed graph rooted at node 0 to a new root s is defined by an exclusive-or operation on all node addresses with the address of node s. Rotation of a graph is defined by the same rotation of the dimensions of the address field for all nodes in the graph [11]. The algorithm for nRSBT broadcasting is

/* nRSBT broadcasting. */
/* Row direction, n-port, binary encoding. */
$lc\_brd1(\alpha, u, i', j', t)$ **over** $[0:N) \times [0:n) \times [0:\frac{P}{n}) \times [0:2^t\frac{Q}{N}) \times [0:n] =$
$\ll t = 0 \rightarrow lc(\alpha, u\frac{P}{n} + i', j'),$
$0 \le j' < 2^{t-1}\frac{Q}{N} \rightarrow lc\_brd1(\alpha, u, i', j', t-1),$
else $\rightarrow lc\_brd1(nbr(\alpha, (u+t-1) \bmod n), u, i', j' - 2^{t-1}\frac{Q}{N}, t-1) \gg,$
$lc\_brd(\alpha, i, j)$ **over** $[0:N) \times [0:P) \times [0:Q) =$
$lc\_brd1(\alpha, \lfloor \frac{in}{P} \rfloor, i \bmod \frac{P}{n}, (ush(\lfloor \frac{in}{P} \rfloor, \lfloor \frac{jN}{Q} \rfloor) \oplus \alpha)\frac{Q}{N} + j \bmod \frac{Q}{N}, n).$

The local submatrix $P \times \frac{Q}{N}$ initially is partitioned into n block rows identified by the parameter u, $0 \le u < n$. The function lc_brd1 has similar structure and meaning as in the SBT broadcasting. The changes from $nbr(\alpha, t-1)$ in SBT to $nbr(\alpha, (u+t-1) \bmod n)$ in nRSBT is because the uth block row communicates along the sequence of cube dimensions u, $(u+1) \bmod n$, $(u+2) \bmod n$, ..., $(u-1) \bmod n$. In the function lc_brd, the n block columns in block row $u = \lfloor \frac{in}{P} \rfloor$ need to be shuffled u steps, i.e., block column $x = \lfloor \frac{jN}{Q} \rfloor$ gets data from block column $ush(u, x)$. The shuffle operation is an unshuffle operation viewed from the receiver's point of view. An nRSBT algorithm for all-to-all reduction and *n-port communication* is included in appendix A.

Matrix transposition for a one-dimensionally partitioned matrix can also be performed by communication algorithms based on Spanning Binomial Tree routing. An algorithm for *one-port*

communication is

/* SBT transpose. */
/* Column partitioning, one-port, binary encoding. */
$lc\_txp1(\alpha, i', j, t)$ **over** $[0:N) \times [0:\frac{P}{2^t}) \times [0:2^t\frac{Q}{N}) \times [0:n] =$
$\ll t = 0 \rightarrow lc(\alpha, i', j),$
/* Processors in subcube 0 w.r.t. cube dimension $n-t$. */
$\lfloor \frac{\alpha}{2^{n-t}} \rfloor \bmod 2 = 0 \rightarrow$
$\ll 0 \leq j < 2^{t-1}\frac{Q}{N} \rightarrow lc\_txp1(\alpha, i', j, t-1)$
else $\rightarrow lc\_txp1(nbr(\alpha, n-t), i', j - 2^{t-1}\frac{Q}{N}, t-1) \gg,$
/* Processors in subcube 1 w.r.t. cube dimension $n-t$. */
else $\rightarrow$
$\ll 0 \leq j < 2^{t-1}\frac{Q}{N} \rightarrow lc\_txp1(nbr(\alpha, n-t), i' + \frac{P}{2^t}, j, t-1)$
else $\rightarrow lc\_txp1(\alpha, i' + \frac{P}{2^t}, j - 2^{t-1}\frac{Q}{N}, t-1) \gg\gg,$
$lc\_txp(\alpha, i', j)$ **over** $[0:N) \times [0:\frac{P}{N}) \times [0:Q) = lc\_txp1(\alpha, i', j, n).$

The goal of the algorithm is to change processor α from storing the αth block column to storing the αth block row. The shape of the local submatrix is $\frac{P}{2^t} \times \frac{2^tQ}{N}$ before iteration t. After iteration t, the local submatrix of processor α is the upper (lower) half of the local submatrix of the previous iteration abutted in the column direction with the upper (lower) half of the submatrix of the $(n-t)$th neighbor processor's of the previous iteration, if the processor is in subcube 0 (1) with respect to cube dimension $n-t$; the former submatrix is put on the left (right) of the latter submatrix if the processor is in subcube 0 (1). Therefore, after each iteration, the number of rows is halved and the number of columns is doubled of the local submatrix. An nRSBT algorithm for *n-port communication* is included in appendix A.

3.2 Two-dimensional Matrix Partitionings

All-to-all broadcasting based on the SBT and nRSBT routings within a column or row subcube are the same as in the one-dimensional case, see appendix A.

Transposing a $P \times Q$ matrix partitioned into $N_1 \times N_2$ blocks, implies that the processor that holds block (i,j), $0 \leq i < N_1$, $0 \leq j < N_2$, will hold block (j,i) after the transposition. For convenience, we assume that the shape of the submatrix defined by a block changes from $\frac{P}{N_1} \times \frac{Q}{N_2}$ to $\frac{P}{N_2} \times \frac{Q}{N_1}$ (instead of changing to a $\frac{Q}{N_1} \times \frac{P}{N_2}$ submatrix). The transposition can be decomposed into two phases. In the first phase, there are $2^{|n_2-n_1|}$ subcubes, such that each subcube executes a transposition of $\min(N_1,N_2) \times \min(N_1,N_2)$ blocks. In the second phase, there are $2^{n-2\min(n_1,n_2)}$ subcubes, such that each subcube executes a one-dimensional transposition. The communication complexity is derived in [9]. The algorithm for *one-port communication* is given below.

/* SBT transpose algorithm for an $N_1 \times N_2$ block matrix. */
$f(t)$ **over** $[0:n] =$

$\ll t \le 2\min(n_1, n_2) \rightarrow 1,$
$n_1 > n_2 \rightarrow 2^{t-2\min(n_1,n_2)},$
else $\rightarrow 2^{2\min(n_1,n_2)-t} \gg,$
$lc\_txp1(\alpha_1, \alpha_2, i', j', t)$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : f(t)\frac{P}{N_1}) \times [0 : \frac{Q}{N_2 f(t)}) \times [0 : n] =$
$\ll t = 0 \rightarrow lc(\alpha_1, \alpha_2, i', j'),$
$t \le 2\min(n_1, n_2) \rightarrow$
/* two-dimensional transpose. */
$\ll \lfloor \frac{\alpha_1}{2^{n_1-\lceil t/2 \rceil}} \rfloor \bmod 2 = \lfloor \frac{\alpha_2}{2^{n_2-\lceil t/2 \rceil}} \rfloor \bmod 2 \rightarrow$
$\ll t \bmod 2 = 1 \rightarrow lc\_txp1(nbr(\alpha_1, n_1 - \lceil t/2 \rceil), \alpha_2, i', j', t-1) \gg,$
else $\rightarrow$
$\ll t \bmod 2 = 0 \rightarrow lc\_txp1(\alpha_1, nbr(\alpha_2, n_2 - \lceil t/2 \rceil), i', j', t-1) \gg\gg,$
/* one-dimensional transpose. */
$n_1 < n_2 \rightarrow$
$\ll \frac{\alpha_2}{\lfloor 2^{n-t} \rfloor} \bmod 2 = 0 \rightarrow$
$\ll 0 \le j' < 2^{t'-1}\frac{Q}{N_2} \rightarrow lc\_txp1(\alpha_1, \alpha_2, i', j', t-1),$
else $\rightarrow lc\_txp1(\alpha_1, nbr(\alpha_2, n-t), i', j' - 2^{t'-1}\frac{Q}{N_2}, t-1) \gg,$
else $\rightarrow$
$\ll 0 \le j' < 2^{t'-1}\frac{Q}{N_2} \rightarrow lc\_txp1(\alpha_1, nbr(\alpha_2, n-t), i' + \frac{P}{2^{n_1+t'}}, j', t-1),$
else $\rightarrow lc\_txp1(\alpha_1, \alpha_2, i' + \frac{P}{2^{n_1+t'}}, j' - 2^{t'-1}\frac{Q}{N_2}, t-1) \gg\gg,$
else $\rightarrow$
$\ll \frac{\alpha_1}{\lfloor 2^{n-t} \rfloor} \bmod 2 = 0 \rightarrow$
$\ll 0 \le i' < 2^{t'-1}\frac{P}{N_1} \rightarrow lc\_txp1(\alpha_1, \alpha_2, i', j', t-1),$
else $\rightarrow lc\_txp1(nbr(\alpha_1, n-t), \alpha_2, i' - 2^{t'-1}\frac{P}{N_1}, j', t-1) \gg,$
else $\rightarrow$
$\ll 0 \le i' < 2^{t'-1}\frac{P}{N_1} \rightarrow lc\_txp1(nbr(\alpha_1, n-t), \alpha_2, i', j' + \frac{Q}{2^{n_2+t'}}, t-1),$
else $\rightarrow lc\_txp1(\alpha_1, \alpha_2, i' - 2^{t'-1}\frac{P}{N_1}, j' + \frac{Q}{2^{n_2+t'}}, t-1) \gg\gg\gg$
where $t' = t - 2\min(n_1, n_2),$
$lc\_txp(\alpha_1, \alpha_2, i', j')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_2}) \times [0 : \frac{Q}{N_1}) = lc\_txp1(\alpha_1, \alpha_2, i', j', n).$

The algorithm used for the two-dimensional transpose part is called SPT (Single Path Transpose algorithm) described in [10], in which a matrix to be transposed is partitioned into a 2×2 block matrix and exchange is performed between the upper right submatrix and the lower left submatrix. This procedure is applied to the 4 submatrices recursively and concurrently.

The reason the function $f(t)$ defined above appears complicated is that the shape of local submatrix is preserved in the two-dimensional transpose phase, but is changed in the one-dimensional transpose phase, as described before. In the second if-clause of lc_txp1 above, we logically partition the cube into 4 subcubes with respect to cube dimensions $n_1 - \lceil \frac{t}{2} \rceil$ and $n_2 - \lceil \frac{t}{2} \rceil$, called subcube 00, 01, 10 and 11. Subcubes 01 and 10 serve as intermediate nodes and get submatrices from subcube 00 and 11, respectively, for odd t. Subcubes 00 and 11 get submatrices from intermediate nodes for even t. The one-dimensional transpose part is similar to the codes described before, except that the role of α is replaced by α_1 or α_2 and the processor id is identified by (α_1, α_2).

With *n-port communication*, one can either run the *n-port* version for the two phases separately, or pipeline the two phases. However, by treating the transposition as a stable dimension permutation [5,6] and employing one of those algorithms in [5,6], a communication complexity lower than that of the above algorithm can be obtained. The dimension permutation algorithm is based on the fact that the two phases can be reversed, or mixed, preserving the permutation.

4 Matrix multiplication

4.1 One-dimensional partitioning

We consider column partitioning. For row partitioning, similar algorithms can be derived.

- **Algorithm $\mathcal{A}(\cdot,1,1)$.** Compute A *in-place* by *broadcasting* C from every processor that has elements of C to every processor that has elements of D. Processor $\alpha = PID(j)$ computes $CD(*, \lfloor \frac{j}{\lceil \frac{R}{N} \rceil} \rfloor)$ for all j mapped to α, where PID is the allocation function as described in Table 2.
- **Algorithm $\mathcal{A}(\cdot,1,2)$.** Compute A by transposing C and *broadcasting* C^T from every processor that has elements of C^T to every processor that has elements of D. Processor $\alpha = PID(j)$ computes $CD(*, \lfloor \frac{j}{\lceil \frac{R}{N} \rceil} \rfloor)$ for all j mapped to α.
- **Algorithm $\mathcal{A}(\cdot,1,3)$.** Compute A by transposing C, *broadcasting* D from every processor that has elements of D to every processor that has elements of C^T and transposing A^T. Processor $\alpha = PID(j)$ computes $C(\lfloor \frac{j}{\lceil \frac{P}{N} \rceil} \rfloor, *)D$.
- **Algorithm $\mathcal{A}(\cdot,1,4)$.** Compute A *in-space* by transposing D and *reduction* of partial inner products of A.

The algorithms are identified by $\mathcal{A}$(*number of ports used concurrently, number of loops parallelized, algorithm identifier*). Algorithm $\mathcal{A}(\cdot,1,2)$ is clearly inferior to algorithm $\mathcal{A}(\cdot,1,1)$ with respect to communication complexity. It is not further considered for the one-dimensional partitioning, however, it will be considered for the two-dimensional partitioning. For row partitioning the roles of C and D are interchanged. Figure 1 characterizes the basic algorithms. The corresponding algorithms for row partitioning are also included for comparison. The two subscripts denote the ordinal number of block rows and block columns. The superscript denotes the ordinal number of the partial inner product result. The number in the square brackets (eg. [R] in $\mathcal{A}(\cdot,1,1)$) is the number of processors that minimizes the arithmetic time for each algorithm.

A complete matrix multiplication algorithm based on *rotations* of the matrix C is given below:

/* A Rotation Algorithm A(1,1,1). */
/* Column partitioning, Gray code encoding. */
$lc(\hat{\alpha}, i, j', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{Q}{N}) \times [0:N) =$

Column Partitioning:

$$\mathcal{A}(\cdot,1,1): \quad C_{*\alpha}, D_{*\alpha} \xrightarrow{\text{brd. C, } \leftrightarrow} C_{**}, D_{*\alpha} \xrightarrow{\text{mpy., [R]}} A_{*\alpha}$$

$$\mathcal{A}(\cdot,1,3): \quad C_{*\alpha}, D_{*\alpha} \xrightarrow{\text{txp. C, } \swarrow\nearrow} C_{\alpha*}, D_{*\alpha} \xrightarrow{\text{brd. D, } \leftrightarrow} C_{\alpha*}, D_{**} \xrightarrow{\text{mpy., [P]}} A_{\alpha*} \xrightarrow{\text{txp. A, } \swarrow\nearrow} A_{*\alpha}$$

$$\mathcal{A}(\cdot,1,4): \quad C_{*\alpha}, D_{*\alpha} \xrightarrow{\text{txp. D, } \swarrow\nearrow} C_{*\alpha}, D_{\alpha*} \xrightarrow{\text{mpy., [Q]}} A^{\alpha}_{**} \xrightarrow{\text{red. A, } \leftrightarrow} A_{*\alpha}$$

Row Partitioning:

$$\mathcal{A}(\cdot,1,1): \quad C_{\alpha*}, D_{\alpha*} \xrightarrow{\text{brd. D, } \updownarrow} C_{\alpha*}, D_{**} \xrightarrow{\text{mpy., [P]}} A_{\alpha*}$$

$$\mathcal{A}(\cdot,1,3): \quad C_{\alpha*}, D_{\alpha*} \xrightarrow{\text{txp. D, } \swarrow\nearrow} C_{\alpha*}, D_{*\alpha} \xrightarrow{\text{brd. C, } \updownarrow} C_{**}, D_{*\alpha} \xrightarrow{\text{mpy., [R]}} A_{*\alpha} \xrightarrow{\text{txp. A, } \swarrow\nearrow} A_{\alpha*}$$

$$\mathcal{A}(\cdot,1,4): \quad C_{\alpha*}, D_{\alpha*} \xrightarrow{\text{txp. C, } \swarrow\nearrow} C_{*\alpha}, D_{\alpha*} \xrightarrow{\text{mpy., [Q]}} A^{\alpha}_{**} \xrightarrow{\text{red. A, } \updownarrow} A_{\alpha*}$$

Figure 1: Notation summary of algorithms for one-dimensional partitioning.

$$\begin{aligned}
&\ll t = 0 \rightarrow c(i, \hat{\alpha}\tfrac{Q}{N} + j'),\\
&\quad \textbf{else} \ \rightarrow lc((\hat{\alpha}+1) \bmod N, i, j', t-1) \gg,\\
&ld(\hat{\alpha}, j, k') \textbf{ over } [0:N) \times [0:Q) \times [0:\tfrac{R}{N}) = d(j, \hat{\alpha}\tfrac{R}{N} + k'),\\
&la(\hat{\alpha}, i, k', t) \textbf{ over } [0:N) \times [0:P) \times [0:\tfrac{R}{N}) \times [0:N] =\\
&\quad \ll t = 0 \rightarrow 0,\\
&\quad\quad \textbf{else} \ \rightarrow la(\hat{\alpha}, i, k', t-1) + (\backslash+ [lc(\hat{\alpha}, i, j', t-1)\\
&\quad\quad\quad\quad * ld(\hat{\alpha}, ((\hat{\alpha}+t-1) \bmod N)\tfrac{Q}{N} + j', k') | 0 \le j' < \tfrac{Q}{N}]) \gg,\\
&a(i,k) \textbf{ over } [0:P) \times [0:R) = la(\lfloor \tfrac{kN}{R} \rfloor, i, k \bmod \tfrac{R}{N}, N).
\end{aligned}$$

For the first statement of the if-clause in lc, the column index of the global matrix C is transformed into the processor id ($\hat{\alpha}$), and the column index of the local submatrix C (j'). With $Q = 2^q$ the highest order n bits of the global column index are interpreted as $\hat{\alpha}$, and the lowest order $q - n$ bits are interpreted as local column index. The transformation between the global and the local index of D and A are similarly defined. The second statement of the if-clause in lc accounts for the rotation from processor $(\hat{\alpha}+1) \bmod N$ to $\hat{\alpha}$. The rotation operation implies nearest-neighbor communication if $\hat{\alpha}$ and $(\hat{\alpha}+1) \bmod N$ are in adjacent processors. Since $\hat{\alpha}$ is the Gray code encoding of the processor id, i.e., the jth block column is stored in processor pid with $G(\hat{\alpha}) = pid = G(j)$, rotation of C implies nearest-neighbor communications.

The second statement of the if-clause in la accounts for the cumulated sums of products at iteration t. The order of the local N block rows of submatrix D which interact with the current submatrix C is important. Processor $\hat{\alpha}$ holds the $((\hat{\alpha}+t) \bmod N)$th block columns of C during iteration t. This column must multiply the corresponding block row of the local submatrix of D. A naive implementation of the above code may use more storage than necessary. For instance, each processor needs to store all the N column blocks of C. However, a reasonable compiler can resolve this problem by deallocating unused space, or by using shared variables.

For binary encoding, i.e., the jth block column is stored in processor $\alpha = j$, we redefine lc and la as follows:

$lc(\alpha, i, j', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{Q}{N}) \times [0:N) =$
$\ll t = 0 \rightarrow c(i, \alpha\frac{Q}{N} + j'),$
else $\rightarrow lc(G((G^{-1}(\alpha) + 1) \bmod N), i, j', t-1) \gg,$
$la(\alpha, i, k', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{R}{N}) \times [0:N] =$
$\ll t = 0 \rightarrow 0,$
else $\rightarrow la(\alpha, i, k', t-1) + (\backslash + [lc(\alpha, i, j', t-1)$
$* ld(\alpha, G((G^{-1}(\alpha) + t - 1) \bmod N)\frac{Q}{N} + j', k') | 0 \leq j' < \frac{Q}{N}]) \gg.$

The difference is that we interpret the first parameter α as a *pid*. Here, $G((G^{-1}(\alpha) + 1) \bmod N)$ computes the pid of its next neighbor in the loop defined by the binary-reflected Gray code traversal.

Instead of *all-to-all broadcasting* through rotations, a *Gray code exchange* algorithm can be used:

/* A Gray code Exchange algorithm A(1,1,1). */
/* Column partitioning, binary code encoding. */
$lc(\alpha, i, j', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{Q}{N}) \times [0:N) =$
$\ll t = 0 \rightarrow c(i, \alpha\frac{Q}{N} + j'),$
else $\rightarrow lc(nbr(\alpha, T(t-1)), i, j', t-1) \gg,$
$ld(\alpha, j, k')$ **over** $[0:N) \times [0:Q) \times [0:\frac{R}{N}) = d(j, \alpha\frac{R}{N} + k'),$
$la(\alpha, i, k', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{R}{N}) \times [0:N] =$
$\ll t = 0 \rightarrow 0,$
else $\rightarrow la(\alpha, i, k', t-1) + (\backslash + [lc(\alpha, i, j', t-1)$
$* ld(\alpha, (\alpha \oplus G(t-1))\frac{Q}{N} + j', k') | 0 \leq j' < \frac{Q}{N}]) \gg,$
$a(i, k)$ **over** $[0:P) \times [0:R) = la(\lfloor\frac{kN}{R}\rfloor, i, k \bmod \frac{R}{N}, N).$

The $nbr(\alpha, T(t-1))$ describes that processors communicate through the sequence of dimensions 0, 1, 0, 2, 0, 1, 0, 3, etc. Note, that all processors communicate along the same cube dimensions at the same step, and the cube dimension changes with time. In the previous rotation algorithm, different processors may communicate along different cube dimensions and the cube dimension does not change with time. During iteration t, processor α holds the $(\alpha \oplus G(t))$th block column of C. For Gray code encoding, the Gray code exchange algorithm can also be used. The code is similar and is included in appendix B. Note, that the rotation algorithm and the Gray code exchange algorithm can be viewed as one-dimensional versions of Cannon's [1] and Dekel's [4] algorithms, respectively. The encodings only affects the order the N block rows of D within each processor where interaction occurs with the current block column of C.

If communication can take place concurrently on all the ports of a processor, the data set for the matrix C is partitioned into n equal pieces (n block rows in the code below). Each piece is broadcasted through a unique path. In the case of the Gray code exchange algorithm, the paths are obtained through rotation of the cube dimensions, such that if the edges in cube dimension $T(t)$ are used by path 0 during step t, then path u uses the edges in cube dimension $(T(t) + u) \bmod n$ during the same step.

/* A Gray code Exchange alg. A(n,1,1). */
/* Column partitioning, binary code encoding. */
$lc(\alpha, u, i', j', t)$ **over** $[0:N) \times [0:n) \times [0:\frac{P}{n}) \times [0:\frac{Q}{N}) \times [0:N) =$
$\ll t = 0 \rightarrow c(u\frac{P}{n} + i', \alpha\frac{Q}{N} + j'),$
else $\rightarrow lc(nbr(\alpha, (T(t-1)+u) \bmod n), u, i', j', t-1) \gg,$
$ld(\alpha, j, k')$ **over** $[0:N) \times [0:Q) \times [0:\frac{R}{N}) = d(j, \alpha\frac{R}{N} + k'),$
$la(\alpha, u, i', k', t)$ **over** $[0:N) \times [0:n) \times [0:\frac{P}{n}) \times [0:\frac{R}{N}) \times [0:N] =$
$\ll t = 0 \rightarrow 0,$
else $\rightarrow la(\alpha, u, i', k', t-1) + (\backslash + [lc(\alpha, u, i', j', t-1)$
$* ld(\alpha, (\alpha \oplus (sh(u, G(t-1))))\frac{Q}{N} + j', k') | 0 \leq j' < \frac{Q}{N}]) \gg,$
$a(i,k)$ **over** $[0:P) \times [0:R) = la(\lfloor\frac{kN}{R}\rfloor, \lfloor\frac{in}{P}\rfloor, i \bmod \frac{P}{n}, k \bmod \frac{R}{N}, N).$

During iteration t, the uth block row of processor α is the uth block row and $\alpha \oplus sh(u, G(t))$th block column of C. The code is essentially the same as in the *one-port* case except for the parameters of nbr in lc and that of ld in la for the reasons just described.

Both the previous algorithms operate with constant storage requirements. The number of communication actions is linear in the number of processors, but can be reduced if there exists sufficient storage to employ a doubling algorithm. Note, that by using a high-level specification for communication the code below is independent of how the communication is realized, and hence independent of, for instance, network topology and low level communication primitives.

The initial allocation of C and D, and the final allocation of A are the same for all the algorithms for column partitioning that we consider. The allocations are shown below, and omitted in the following.

/* Initial allocation of C and D. */
$lc(\alpha, i, j')$ **over** $[0:N) \times [0:P) \times [0:\frac{Q}{N}) = c(i, \alpha\frac{Q}{N} + j'),$
$ld(\alpha, j, k')$ **over** $[0:N) \times [0:Q) \times [0:\frac{R}{N}) = d(j, \alpha\frac{R}{N} + k'),$
/* Final location of matrix A. */
/* For Algorithm A(·,1,4), la is replaced by la_red below. */
$a(i,k)$ **over** $[0:P) \times [0:R) = la(\lfloor\frac{kN}{R}\rfloor, i, k \bmod \frac{R}{N}).$

/* A Doubling Algorithm A(·,1,1): */
$lc\_brd(\alpha, i, j)$ **over** $[0:N) \times [0:P) \times [0:Q) = lc(\lfloor\frac{jN}{Q}\rfloor, i, j \bmod \frac{Q}{N}),$
$la(\alpha, i, k')$ **over** $[0:N) \times [0:P) \times [0:\frac{R}{N}) = \backslash + [lc\_brd(\alpha, i, j) * ld(\alpha, j, k') | 0 \leq j < Q].$

/* Algorithm A(·,1,3): */
$lc\_txp(\alpha, i', j)$ **over** $[0:N) \times [0:\frac{P}{N}) \times [0:Q) = lc(\lfloor\frac{jN}{Q}\rfloor, \alpha\frac{P}{N} + i', j \bmod \frac{Q}{N}),$
$ld\_brd(\alpha, j, k)$ **over** $[0:N) \times [0:Q) \times [0:R) = ld(\lfloor\frac{kN}{R}\rfloor, j, k \bmod \frac{R}{N}),$
$la\_txp(\alpha, i', k)$ **over** $[0:N) \times [0:\frac{P}{N}) \times [0:R) = \backslash + [lc\_txp(\alpha, i', j) * ld\_brd(\alpha, j, k) | 0 \leq j < Q],$
$la(\alpha, i, k')$ **over** $[0:N) \times [0:P) \times [0:\frac{R}{N}) = la\_txp(\lfloor\frac{iN}{P}\rfloor, i \bmod \frac{P}{N}, \alpha\frac{R}{N} + k').$

Algorithm	Number of arithmetic operations
$\mathcal{A}(\cdot,1,1)$	$2PQ\lceil\frac{R}{N}\rceil$
$\mathcal{A}(\cdot,1,3)$	$2QR\lceil\frac{P}{N}\rceil$
$\mathcal{A}(\cdot,1,4)$	$PR(2\lceil\frac{Q}{N}\rceil-1)+P(\lceil\frac{R}{N}\rceil+\sum_{i=1}^{n}\lceil\frac{R}{2^i}\rceil)$

Table 6: The arithmetic time for one-dimensional column partitioning.

/* Algorithm A(·,1,4): */
$ld\_txp(\alpha,j',k)$ **over** $[0:N)\times[0:\frac{Q}{N})\times[0:R) = ld(\lfloor\frac{kN}{R}\rfloor,\alpha\frac{Q}{N}+j',k \bmod \frac{R}{N})$,
$la(\alpha,i,k)$ **over** $[0:N)\times[0:P)\times[0:R) = \backslash+[lc(\alpha,i,j')*ld\_txp(\alpha,j',k)|0\le j'<\frac{Q}{N}]$,
$la\_red(\alpha,i,k')$ **over** $[0:N)\times[0:P)\times[0:\frac{R}{N}) = \backslash+[la(\alpha',i,\alpha\frac{R}{N}+k')|0\le\alpha'<N]$.

Algorithm $\mathcal{A}(\cdot,1,1)$ broadcasts C and then performs multiplication of $P\times Q$ and $Q\times\frac{R}{N}$ matrices locally. Algorithm $\mathcal{A}(\cdot,1,3)$ transposes C, broadcasts D, performs multiplication of $\frac{P}{N}\times Q$ and $Q\times R$ matrices locally, and transposes A. Algorithm $\mathcal{A}(\cdot,1,4)$ transposes D, performs multiplication of $P\times\frac{Q}{N}$ and $\frac{Q}{N}\times R$ matrices locally, and performs (all-to-all) reduction of the partial inner product. By calling the communication primitives such as broadcasting, transpose and reduction, the implementation details are hidden.

Table 6 shows the total number of arithmetic operations in sequence. If P,Q, and R are all multiples of N, then all three algorithms have the same arithmetic complexity. For $P,Q,R\ge N$, the differences of the arithmetic complexities are within constant factors. Table 7 shows the total number of elements transferred in sequence and the minimum number of start-ups for $P,Q,R\ge N$. The superscript l on $\mathcal{A}$ denotes a linear array algorithm, and superscript c a Boolean cube algorithm. For some values of P, Q, and R less than N, the communication complexity can be smaller than what is given in the table, because some of the broadcastings and personalized communications may complete earlier. The communication complexity for the general case is complicated and is described in [9]. The data transfer time compares as $PQ:QR:PR$, approximately, by considering the highest-order term of $\mathcal{A}(\cdot,1,1)$, $\mathcal{A}(\cdot,1,3)$ and $\mathcal{A}(\cdot,1,4)$ and assuming $P,Q,R\ge N$. Note that for $\frac{P}{N}=\frac{Q}{N}=\frac{R}{N}$, the communication complexity of $\mathcal{A}(\cdot,1,1)$ is less than that of $\mathcal{A}(\cdot,1,4)$, which in turn is less than that of $\mathcal{A}(\cdot,1,3)$. For a detailed analysis, see [9].

4.2 Two-dimensional partitioning

The algorithms described for the one-dimensional case have analogues in the two dimensional case. Algorithm $\mathcal{A}(\cdot,1,1)$ that computes A *in-place* by broadcasting C in its two-dimensional form, requires broadcasting elements of C along rows, and broadcasting elements of D along columns. The two broadcasting operations need to be synchronized in order to conserve storage. Cannon [1] has described such an algorithm for mesh configured multiprocessors (that can be emulated on Boolean cubes) and Dekel et al. [4] described such an algorithm making use of

Algorithm	Element transfers	min start-ups
$\mathcal{A}^l$(1,1,1)	$(N-1)P\lceil\frac{Q}{N}\rceil$	$N-1$
$\mathcal{A}^c$(1,1,1)	$(N-1)P\lceil\frac{Q}{N}\rceil$	n
$\mathcal{A}^c$(1,1,3)	$(N-1)Q\lceil\frac{R}{N}\rceil + \frac{nP}{2}\lceil\frac{R}{N}\rceil + \frac{nP}{2}\lceil\frac{Q}{N}\rceil$	$3n$
$\mathcal{A}^c$(1,1,4)	$(N-1)P\lceil\frac{R}{N}\rceil + \frac{nQ}{2}\lceil\frac{R}{N}\rceil$	$2n$
$\mathcal{A}^l$(n,1,1)	$\frac{1}{n}(N-1)P\lceil\frac{Q}{N}\rceil$	$N-1$
$\mathcal{A}^c$(n,1,1)	$\frac{1}{n}(N-1)P\lceil\frac{Q}{N}\rceil$	n
$\mathcal{A}^c$(n,1,3)	$\frac{1}{n}(N-1)Q\lceil\frac{R}{N}\rceil + \frac{P}{2}\lceil\frac{R}{N}\rceil + \frac{P}{2}\lceil\frac{Q}{N}\rceil$	$3n$
$\mathcal{A}^c$(n,1,4)	$\frac{1}{n}(N-1)P\lceil\frac{R}{N}\rceil + \frac{Q}{2}\lceil\frac{R}{N}\rceil$	$2n$

Table 7: The communication complexity using one-dimensional column partitioning, assuming $P, Q, R \geq N$.

the Boolean cube topology. These algorithms are special cases of matrix multiplication using broadcasting algorithms that preserve storage requirements.

The algorithms corresponding to the four one-dimensional algorithms ($\mathcal{A}(\cdot,1,4)$ has two variations) are as follows:

- **Algorithm $\mathcal{A}(\cdot,2,1)$.** Compute A *in-place* by broadcasting C in the row direction and D in the column direction, such that each processor receives all elements of the rows of C mapped into that processor row and all elements of D mapped into the corresponding column of processors. Processor α_1, α_2 then computes $C(\lfloor\frac{i}{\lceil\frac{P}{N_1}\rceil}\rfloor, *)D(*, \lfloor\frac{j}{\lceil\frac{R}{N_2}\rceil}\rfloor)$ for all i mapped to α_1 and all j mapped to α_2. The communication operations are *broadcasting* from multiple sources within rows and columns.

  Algorithm $\mathcal{A}(\cdot,2,2)$. Transpose C, perform a multiple source *broadcast* along processor rows for the elements of C^T in that processor row, and accumulate inner products for A through multiple sink *reduction* in the column direction (of the processors). The accumulation can be made such that $\frac{P}{N_1}$ elements for each column of D are accumulated in each processor by *all-to-all reduction*. A processor α_1, α_2 receives $C(*, \lfloor\frac{i}{\lceil\frac{Q}{N_1}\rceil}\rfloor)$ during the broadcasting operation, then computes the product $C(*, \lfloor\frac{i}{\lceil\frac{Q}{N_1}\rceil}\rfloor)D(\lfloor\frac{i}{\lceil\frac{Q}{N_1}\rceil}\rfloor, \lfloor\frac{j}{\lceil\frac{R}{N_2}\rceil}\rfloor)$. The summation over index i is the reduction operation along columns.

- **Algorithm $\mathcal{A}(\cdot,2,3)$.** Transpose C, perform multiple source *broadcasting* of the elements of D within processor rows and accumulate inner products in the column direction. The multiple sink *reduction* is performed such that each processor receives all $\frac{P}{N_2}$ elements of $\frac{R}{N_1}$ distinct columns of D such that A^T is computed. (Alternatively, the accumulation can be made such that $\frac{P}{\max(N_1,N_2)}$ elements for each column are accumulated in a processor selected such that the proper allocation of A is obtained through a *some-to-all personalized communication* within rows.) Processor α_1, α_2 computes $C(\lfloor\frac{i}{\lceil\frac{P}{N_2}\rceil}\rfloor, \lfloor\frac{j}{\lceil\frac{Q}{N_1}\rceil}\rfloor)D(\lfloor\frac{j}{\lceil\frac{Q}{N_1}\rceil}\rfloor, *)$

$$\mathcal{A}(\cdot,2,1): \quad C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xrightarrow{\text{brd. C, } \leftrightarrow} C_{\alpha_1 *}, D_{\alpha_1\alpha_2} \xrightarrow{\text{brd. D, } \updownarrow} C_{\alpha_1 *}, D_{*\alpha_2} \xrightarrow{\text{mpy., [PR]}} A_{\alpha_1\alpha_2}$$

$$\mathcal{A}(\cdot,2,2): \quad C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xrightarrow{\text{txp. C, } \swarrow\nearrow} C_{\alpha_2\alpha_1}, D_{\alpha_1\alpha_2} \xrightarrow{\text{brd. C, } \leftrightarrow} C_{*\alpha_1}, D_{\alpha_1\alpha_2} \xrightarrow{\text{mpy., [QR]}} A^{\alpha_1}_{*\alpha_2} \xrightarrow{\text{red. A, } \updownarrow} A_{\alpha_1\alpha_2}$$

$$\mathcal{A}(\cdot,2,3): \quad C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xrightarrow{\text{txp. C, } \swarrow\nearrow} C_{\alpha_2\alpha_1}, D_{\alpha_1\alpha_2} \xrightarrow{\text{brd. D, } \leftrightarrow} C_{\alpha_2\alpha_1}, D_{\alpha_1 *} \xrightarrow{\text{mpy., [PQ]}} A^{\alpha_1}_{\alpha_2 *} \xrightarrow{\text{red. A, } \updownarrow} A_{\alpha_2\alpha_1} \xrightarrow{\text{txp. A, } \swarrow\nearrow} A_{\alpha_1\alpha_2}$$

$$\mathcal{A}(\cdot,2,4): \quad C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xrightarrow{\text{txp. D, } \swarrow\nearrow} C_{\alpha_1\alpha_2}, D_{\alpha_2\alpha_1} \xrightarrow{\text{brd. D, } \updownarrow} C_{\alpha_1\alpha_2}, D_{\alpha_2 *} \xrightarrow{\text{mpy., [PQ]}} A^{\alpha_2}_{\alpha_1 *} \xrightarrow{\text{red. A, } \leftrightarrow} A_{\alpha_1\alpha_2}$$

$$\mathcal{A}(\cdot,2,5): \quad C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xrightarrow{\text{txp. D, } \swarrow\nearrow} C_{\alpha_1\alpha_2}, D_{\alpha_2\alpha_1} \xrightarrow{\text{brd. C, } \updownarrow} C_{*\alpha_2}, D_{\alpha_2\alpha_1} \xrightarrow{\text{mpy., [QR]}} A^{\alpha_2}_{*\alpha_1} \xrightarrow{\text{red. A, } \leftrightarrow} A_{\alpha_2\alpha_1} \xrightarrow{\text{txp. A, } \swarrow\nearrow} A_{\alpha_1\alpha_2}$$

Figure 2: Notation summary of the algorithms for two-dimensional partitioning.

for all i, j such that $\lfloor \frac{i}{\lceil \frac{P}{N_2} \rceil} \rfloor = \alpha_2$ and $\lfloor \frac{j}{\lceil \frac{Q}{N_1} \rceil} \rfloor = \alpha_1$.

- **Algorithm $\mathcal{A}(\cdot,2,4)$.** Transpose D, perform a multiple source *broadcasting* of the elements of D^T within processor columns, and accumulate the partial inner products for elements of A by multiple sink *reduction* along processor rows such that the elements of at most $\lceil \frac{R}{N_2} \rceil$ columns are accumulated within a processor column. After transposition and broadcasting, processor α_1, α_2 has the elements $C(\lfloor \frac{i}{\lceil \frac{P}{N_1} \rceil} \rfloor, \lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor) D(\lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor, *)$ for all i such that $\lfloor \frac{i}{\lceil \frac{P}{N_1} \rceil} \rfloor = \alpha_1$ and j such that $\lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor = \alpha_2$.

- **Algorithm $\mathcal{A}(\cdot,2,5)$.** Transpose D, perform a multiple source *broadcasting* of the elements of C within processor columns and accumulate inner products for elements of A by multiple sink *reduction* along processor rows, such that each processor receives $\frac{P}{N_2}$ elements of A^T for each of $\frac{R}{N_1}$ columns of D. Processor α_1, α_2 computes $C(*, \lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor) D(\lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor, \lfloor \frac{i}{\lceil \frac{R}{N_1} \rceil} \rfloor)$ for all i such that $\lfloor \frac{i}{\lceil \frac{R}{N_1} \rceil} \rfloor = \alpha_1$ and j such that $\lfloor \frac{j}{\lceil \frac{Q}{N_2} \rceil} \rfloor = \alpha_2$.

Figure 2 characterizes the 5 algorithms. The two subscripts in sequence are used to denote the ordinal number of block rows and block columns of the $N_1 \times N_2$ blocks. The "$*$" sign means union of all the block rows (or columns). The superscript denotes the ordinal number of the partial inner product result. The number in the square brackets (eg. [PR] in $\mathcal{A}(\cdot,2,1)$) is the minimum maximum number of processors to minimize the arithmetic time for each algorithm. Algorithm $\mathcal{A}(\cdot,2,2)$ has a matrix transpose in addition to the communication of C as in algorithm $\mathcal{A}(\cdot,2,1)$. However, unlike the one-dimensional case, algorithm $\mathcal{A}(\cdot,2,2)$ may have a higher processor utilization than algorithm $\mathcal{A}(\cdot,2,1)$.

Broadcasting in $\mathcal{A}(1,2,1)$ can be realized by a rotation algorithm, which yields Cannon's algorithm [1]. Unlike the one-dimensional case, an initial alignment is required in order to

synchronize between the rotations of C and D. For $N_1 = N_2 = \sqrt{N}$, the code is shown below. For $N_1 \neq N_2$, say $N_1 > N_2$, we further partition the submatrix C in each processor into $\frac{N_1}{N_2}$ blocks and simulate the algorithm for $N_1 \times N_1$ blocks. Each processor simulates $\frac{N_1}{N_2}$ processors. The code is included in appendix B.

/* Cannon's Algorithm A(1,2,1): */

/* Assume $N_1 = N_2 = \sqrt{N}$, Gray code encoding. */

$lc(\hat{\alpha}_1, \hat{\alpha}_2, i', j', t)$ **over** $[0:\sqrt{N}) \times [0:\sqrt{N}) \times [0:\frac{P}{\sqrt{N}}) \times [0:\frac{Q}{\sqrt{N}}) \times [0:\sqrt{N}) =$

$\ll t = 0 \rightarrow c(\hat{\alpha}_1 \frac{P}{\sqrt{N}} + i', \hat{\alpha}_2 \frac{Q}{\sqrt{N}} + j'),$

/* Initial alignment step. */

$t = 1 \rightarrow lc(\hat{\alpha}_1, (\hat{\alpha}_1 + \hat{\alpha}_2) \bmod \sqrt{N}, i', j', 0),$

/* Get from east neighbor. */

else $\rightarrow lc(\hat{\alpha}_1, (\hat{\alpha}_2 + 1) \bmod \sqrt{N}, i', j', t-1) \gg,$

$ld(\hat{\alpha}_1, \hat{\alpha}_2, j', k', t)$ **over** $[0:\sqrt{N}) \times [0:\sqrt{N}) \times [0:\frac{Q}{\sqrt{N}}) \times [0:\frac{R}{\sqrt{N}}) \times [0:\sqrt{N}) =$

$\ll t = 0 \rightarrow d(\hat{\alpha}_1 \frac{Q}{\sqrt{N}} + j', \hat{\alpha}_2 \frac{R}{\sqrt{N}} + k'),$

/* Initial alignment step. */

$t = 1 \rightarrow ld((\hat{\alpha}_1 + \hat{\alpha}_2) \bmod \sqrt{N}, \hat{\alpha}_2, i', j', 0),$

/* Get from south neighbor. */

else $\rightarrow ld((\hat{\alpha}_1 + 1) \bmod \sqrt{N}, \hat{\alpha}_2, j', k', t-1) \gg,$

$la(\hat{\alpha}_1, \hat{\alpha}_2, i', k', t)$ **over** $[0:\sqrt{N}) \times [0:\sqrt{N}) \times [0:\frac{P}{\sqrt{N}}) \times [0:\frac{R}{\sqrt{N}}) \times [0:\sqrt{N}] =$

$\ll t = 0 \rightarrow 0,$

else $\rightarrow la(\hat{\alpha}_1, \hat{\alpha}_2, i', k', t-1) + (\backslash + [lc(\hat{\alpha}_1, \hat{\alpha}_2, i', j', t)$

$* ld(\hat{\alpha}_1, \hat{\alpha}_2, j', k', t) | 0 \leq j' < \frac{Q}{\sqrt{N}}]) \gg,$

$a(i,k)$ **over** $[0:P) \times [0:R) = la\_red(\lfloor \frac{i\sqrt{N}}{P} \rfloor, \lfloor \frac{k\sqrt{N}}{R} \rfloor, i \bmod \frac{P}{\sqrt{N}}, k \bmod \frac{R}{\sqrt{N}}).$

Processor $(\hat{\alpha}_1, \hat{\alpha}_2)$ gets the local submatrix of C from processor $(\hat{\alpha}_1, (\hat{\alpha}_1 + \hat{\alpha}_2) \bmod \sqrt{N})$ and the local submatrix of D from processor $((\hat{\alpha}_1 + \hat{\alpha}_2) \bmod \sqrt{N}, \hat{\alpha}_2)$ for the initial alignment. The rest of the code is similar to the Rotation Algorithm in the one-dimensional case. During each iteration, processor $(\hat{\alpha}_1, \hat{\alpha}_2)$ gets the local submatrix of C from its east neighbor, $(\hat{\alpha}_1, (\hat{\alpha}_2 + 1) \bmod \sqrt{N})$, and the local submatrix of D from its south neighbor, $((\hat{\alpha}_1 + 1) \bmod \sqrt{N}, \hat{\alpha}_2)$.

It is also possible to design a matrix multiplication algorithm based on the SBT, or the nRSBT communication algorithms. For Algorithm $\mathcal{A}(\cdot,2,1)$, the temporary storage for each processor becomes $\frac{PQ}{N_1}$ for C and $\frac{QR}{N_2}$ for D, instead of $\frac{PQ}{N}$ and $\frac{QR}{N}$ for Cannon's or Dekel's algorithms. However, the number of start-ups is reduced to $O(n_1 + n_2)$, instead of $O(N_1 + N_2)$. Note, that the initial alignment steps can be eliminated. It is possible to interleave the communication and multiplication steps to save half of the storage. However, an initial alignment is required for such an algorithm.

The initial allocations of C and D, and final allocation of A for the five algorithms below are the same, and is described once and for all. For Algorithms $\mathcal{A}(\cdot,2,2)$ and $\mathcal{A}(\cdot,2,4)$, la is replaced by *la_red.*

/* Initial allocations of C and D. */

$lc(\alpha_1, \alpha_2, i', j')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : \frac{Q}{N_2}) = c(\alpha_1 \frac{P}{N_1} + i', \alpha_2 \frac{Q}{N_2} + j')$,
$ld(\alpha_1, \alpha_2, j', k')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{Q}{N_1}) \times [0 : \frac{R}{N_2}) = d(\alpha_1 \frac{Q}{N_1} + j', \alpha_2 \frac{R}{N_2} + k')$,
/* Final allocation of A. */
$a(i, k)$ **over** $[0 : P) \times [0 : R) = la(\lfloor \frac{iN_1}{P} \rfloor, \lfloor \frac{kN_2}{R} \rfloor, i \bmod \frac{P}{N_1}, k \bmod \frac{R}{N_2})$.

/* A Doubling Algorithm A(·,2,1): */
$lc\_row(\alpha_1, \alpha_2, i', j)$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : Q) = lc(\alpha_1, \lfloor \frac{jN_2}{Q} \rfloor, i', j \bmod \frac{Q}{N_2})$,
$ld\_col(\alpha_1, \alpha_2, j, k')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : Q) \times [0 : \frac{R}{N_2}) = ld(\lfloor \frac{jN_1}{Q} \rfloor, \alpha_2, j \bmod \frac{Q}{N_1}, k')$,
$la(\alpha_1, \alpha_2, i', k')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
$\backslash + [lc\_row(\alpha_1, \alpha_2, i', j) * ld\_col(\alpha_1, \alpha_2, j, k') | 0 \le j < Q]$.

/* Algorithm A(·,2,2): */
$lc\_txp(\alpha_1, \alpha_2, i', j')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_2}) \times [0 : \frac{Q}{N_1}) =$
$lc(\lfloor (\alpha_1 \frac{P}{N_2} + i')/\frac{P}{N_1} \rfloor, \lfloor (\alpha_2 \frac{Q}{N_1} + j')/\frac{Q}{N_2} \rfloor, (\alpha_1 \frac{P}{N_2} + i') \bmod \frac{P}{N_1}, (\alpha_2 \frac{Q}{N_1} + j') \bmod \frac{Q}{N_2})$,
$lc\_txp\_row(\alpha_1, \alpha_2, i, j')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : P) \times [0 : \frac{Q}{N_1}) =$
$lc\_txp(\alpha_1, \lfloor \frac{iN_2}{P} \rfloor, i \bmod \frac{P}{N_2}, j')$,
$la(\alpha_1, \alpha_2, i, k')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : P) \times [0 : \frac{R}{N_2}) =$
$\backslash + [lc\_txp\_row(\alpha_1, \alpha_2, i, j') * ld(\alpha_1, \alpha_2, j', k') | 0 \le j' < \frac{Q}{N_1}]$,
$la\_red(\alpha_1, \alpha_2, i', k')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
$\backslash + [la(\alpha_1', \alpha_2, \alpha_1 \frac{P}{N_2} + i', k') | 0 \le \alpha_1' < N_1]$.

/* Algorithm A(·,2,3): */
$lc\_txp(\alpha_1, \alpha_2, i', j')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_2}) \times [0 : \frac{Q}{N_1}) =$
$lc(\lfloor (\alpha_1 \frac{P}{N_2} + i')/\frac{P}{N_1} \rfloor, \lfloor (\alpha_2 \frac{Q}{N_1} + j')/\frac{Q}{N_2} \rfloor, (\alpha_1 \frac{P}{N_2} + i') \bmod \frac{P}{N_1}, (\alpha_2 \frac{Q}{N_1} + j') \bmod \frac{Q}{N_2})$,
$ld\_row(\alpha_1, \alpha_2, j', k)$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{Q}{N_1}) \times [0 : R) =$
$ld(\alpha_1, \lfloor \frac{kN_2}{R} \rfloor, j', k \bmod \frac{R}{N_2})$,
$la\_txp(\alpha_1, \alpha_2, i', k)$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_2}) \times [0 : R) =$
$\backslash + [lc\_txp(\alpha_1, \alpha_2, i', j') * ld\_row(\alpha_1, \alpha_2, j', k) | 0 \le j < \frac{Q}{N_1}]$,
$la\_txp\_red(\alpha_1, \alpha_2, i', k')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_2}) \times [0 : \frac{R}{N_1}) =$
$\backslash + [la\_txp(\alpha_1', \alpha_2, i', \alpha_1 \frac{R}{N_1} + k') | 0 \le \alpha_1' < N_1]$,
$la(\alpha_1, \alpha_2, i', k')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
$la\_txp\_red(\lfloor (\alpha_1 \frac{P}{N_1} + i')/\frac{P}{N_2} \rfloor, \lfloor (\alpha_2 \frac{R}{N_2} + k')/\frac{R}{N_1} \rfloor, (\alpha_1 \frac{P}{N_1} + i') \bmod \frac{P}{N_2}, (\alpha_2 \frac{R}{N_2} + k') \bmod \frac{R}{N_1})$.

Algorithm $\mathcal{A}(\cdot,2,1)$ broadcasts C along rows, broadcasts D along columns and locally multiplies matrices of the forms $\frac{P}{N_1} \times Q$ and $Q \times \frac{R}{N_2}$. Algorithm $\mathcal{A}(\cdot,2,2)$ transposes C, broadcasts C along rows, locally multiplies matrices of the forms $P \times \frac{Q}{N_1}$ and $\frac{Q}{N_1} \times \frac{R}{N_2}$ and performs reduction of the partial inner product along columns. Algorithm $\mathcal{A}(\cdot,2,3)$ transposes C, broadcasts D along rows, locally multiplies matrices of the forms $\frac{P}{N_2} \times \frac{Q}{N_1}$ and $\frac{Q}{N_1} \times R$, performs reduction

Algorithm	Number of arithmetic operations
$\mathcal{A}^c(\cdot,2,1)$	$2Q\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$
$\mathcal{A}^c(\cdot,2,2)$	$(2\lceil\frac{Q}{N_1}\rceil-1)\lceil\frac{R}{N_2}\rceil P+\sum_{i=1}^{n_1}\lceil\frac{R}{N_2}\rceil\lceil\frac{P}{2^i}\rceil+\lceil\frac{R}{N_2}\rceil\lceil\frac{P}{N_1}\rceil$
$\mathcal{A}^c(\cdot,2,3)$	$(2\lceil\frac{Q}{N_1}\rceil-1)\lceil\frac{P}{N_2}\rceil R+\sum_{i=1}^{n_1}\lceil\frac{P}{N_2}\rceil\lceil\frac{R}{2^i}\rceil+\lceil\frac{P}{N_2}\rceil\lceil\frac{R}{N_1}\rceil$
$\mathcal{A}^c(\cdot,2,4)$	$(2\lceil\frac{Q}{N_2}\rceil-1)\lceil\frac{P}{N_1}\rceil R+\sum_{i=1}^{n_2}\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{2^i}\rceil+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$
$\mathcal{A}^c(\cdot,2,5)$	$(2\lceil\frac{Q}{N_2}\rceil-1)\lceil\frac{R}{N_1}\rceil P+\sum_{i=1}^{n_2}\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{2^i}\rceil+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil$

Table 8: The communication complexity for *optimum* buffer sizes, two-dimensional partitioning, and *one-port* communication.

on A along rows of matrix (R), which is column of processor arrays (N_1), and then transposes A. Algorithms $\mathcal{A}(\cdot,2,4)$ and $\mathcal{A}(\cdot,2,5)$ are included in appendix B and is similar to Algorithms $\mathcal{A}(\cdot,2,2)$ and $\mathcal{A}(\cdot,2,3)$, respectively.

Table 8 shows the total number of arithmetic operations in sequence. Note, that if P, Q and R are multiples of N_1 and N_2, then the arithmetic complexities of the algorithms are the same, and indeed the same as for a one-dimensional partitioning. Table 9 shows the total number of elements transferred in sequence and the minimum number of start-ups with *one-port communication.* By using some approximations, the values of N_1 and N_2 that minimize the number of elements transferred for different algorithms are shown in Table 10. The resulting total complexities are shown in Table 11. By considering the highest-order term, the data transfer times compare as $\sqrt{Q}:\sqrt{P}:\sqrt{R}:\sqrt{R}:\sqrt{P}$ from $\mathcal{A}(1,2,1)$ to $\mathcal{A}(1,2,5)$. It can be shown [9] that for P, Q and R being multiples of N_1 and N_2, the complexities of algorithms $\mathcal{A}(\cdot,2,3)$ and $\mathcal{A}(\cdot,2,5)$ are always higher than that of $\min(\mathcal{A}(\cdot,2,2), \mathcal{A}(\cdot,2,4))$, if the optimum values of N_1 and N_2 are chosen for each algorithm. Table 12 shows the communication complexity with *n-port communication* and optimum packet size. For a detailed analysis and optimum choice of algorithms, see [9].

Algorithm	Element transfers	min start-ups
$\mathcal{A}^c(1,2,1)$	$(N_2-1)\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil$ $+(N_1-1)\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil$	n
$\mathcal{A}^c(1,2,2)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil n+\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_2-1)$ $+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_1-1)$	$2n$
$\mathcal{A}^c(1,2,3)$	$\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil n+\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_2-1)$ $+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_1-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil n$	$3n$
$\mathcal{A}^c(1,2,4)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil n+\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil(N_1-1)$ $+\lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil(N_2-1)$	$2n$
$\mathcal{A}^c(1,2,5)$	$\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil n+\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil(N_1-1)$ $+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil(N_2-1)+\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil n$	$3n$

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

Algorithm	N_1	N_2
$\mathcal{A}^c(1,2,1)$	$\sqrt{\frac{PN}{R}}$	$\sqrt{\frac{RN}{P}}$
$\mathcal{A}^c(1,2,2)$	$\sqrt{\frac{QN}{R}}$	$\sqrt{\frac{RN}{Q}}$
$\mathcal{A}^c(1,2,3)$	$\sqrt{\frac{QN}{P}}$	$\sqrt{\frac{PN}{Q}}$
$\mathcal{A}^c(1,2,4)$	$\sqrt{\frac{PN}{Q}}$	$\sqrt{\frac{QN}{P}}$
$\mathcal{A}^c(1,2,5)$	$\sqrt{\frac{RN}{Q}}$	$\sqrt{\frac{QN}{R}}$

Table 10: The *optimum* values of N_1 and N_2 for P, Q and R being multiples of N and *one-port communication.*

Algorithm	T_{min}
$\mathcal{A}^c(1,2,1)$	$\frac{2PQR}{N}t_a+\frac{Q}{\sqrt{N}}(2\sqrt{PR}-\frac{P+R}{\sqrt{N}})t_c+n\tau$
$\mathcal{A}^c(1,2,2)$	$\frac{2PQR}{N}t_a+\frac{P}{\sqrt{N}}(2\sqrt{QR}+\frac{nQ-(Q+R)}{\sqrt{N}})t_c+2n\tau$
$\mathcal{A}^c(1,2,3)$	$\frac{2PQR}{N}t_a+\frac{R}{\sqrt{N}}(2\sqrt{PQ}+\frac{nP(1+\frac{Q}{R})-(P+Q)}{\sqrt{N}})t_c+3n\tau$
$\mathcal{A}^c(1,2,4)$	$\frac{2PQR}{N}t_a+\frac{R}{\sqrt{N}}(2\sqrt{PQ}+\frac{nQ-(P+Q)}{\sqrt{N}})t_c+2n\tau$
$\mathcal{A}^c(1,2,5)$	$\frac{2PQR}{N}t_a+\frac{P}{\sqrt{N}}(2\sqrt{QR}+\frac{nR(1+\frac{Q}{P})-(Q+R)}{\sqrt{N}})t_c+3n\tau$

Table 11: The total complexity with *optimum* values of N_1 and N_2 for P, Q and R being multiples of N and *one-port communication.*

Algorithm	min communication time
$\mathcal{A}^c$(n,2,1)	$\max(\frac{N_2-1}{n_2}\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil t_c + n_2\tau,$ $\frac{N_1-1}{n_1}\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_c + n_1\tau)$
$\mathcal{A}^c$(n,2,2)	$n\tau + (\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{Q}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{N_2-1}{n_2} + \lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{N_1-1}{n_1})t_c$
$\mathcal{A}^c$(n,2,3)	$n\tau + (\sqrt{\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{N_2-1}{n_2} + \lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{N_1-1}{n_1})t_c$
$\mathcal{A}^c$(n,2,4)	$n\tau + (\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{R}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{N_1-1}{n_1} + \lceil\frac{P}{N_1}\rceil\lceil\frac{R}{N_2}\rceil\frac{N_2-1}{n_2})t_c$
$\mathcal{A}^c$(n,2,5)	$n\tau + (\sqrt{\lceil\frac{Q}{N_1}\rceil\lceil\frac{R}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\sqrt{\lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil t_c} + \sqrt{(n-1)\tau})^2$ $+(\lceil\frac{P}{N_1}\rceil\lceil\frac{Q}{N_2}\rceil\frac{N_1-1}{n_1} + \lceil\frac{R}{N_1}\rceil\lceil\frac{P}{N_2}\rceil\frac{N_2-1}{n_2})t_c$

Table 12: The communication complexity for *optimum* buffer sizes, two-dimensional partitioning, and *n-port* communication.

4.3 Three-dimensional partitioning

In the case of a three dimensional partitioning of the Boolean cube, each $N_1' \times N_2'$ subset of processors compute the product of a $P \times \frac{Q}{N_3'}$ matrix and a $\frac{Q}{N_3'} \times R$ matrix. If the matrices are initially allocated such that there are distinct submatrices $P \times \frac{Q}{N_3'}$ and $\frac{Q}{N_3'} \times R$ assigned to each set of $\frac{N}{N_3'}$ processors, then the multiplication in each subset is the same as in the two dimensional partitioning, except that Q is replaced by $\frac{Q}{N_3'}$. In addition, there is an accumulation phase at the end. The number of arithmetic operations for this part of the computation is $\lceil \frac{P}{N_1'} \rceil \lceil \frac{R}{N_2'} \rceil \log N_3'$ without any pipelining, and all partial products being accumulated in the same way. Matrix A is allocated among $\frac{N}{N_3'}$ processors. If there are several elements of the matrix A that are stored in the same processor, then the accumulation can be made faster by using *all-to-all reduction*. The arithmetic complexity becomes $\sum_{i=1}^{n_3'} \lceil \frac{\lceil \frac{P}{N_1'} \rceil \lceil \frac{R}{N_2'} \rceil}{2^i} \rceil t_a$. The communication complexity for the reduction is $\sum_{i=1}^{n_3'} \lceil \frac{\lceil \frac{P}{N_1'} \rceil \lceil \frac{R}{N_2'} \rceil}{2^i} \rceil t_c + \sum_{i=1}^{n_3'} \lceil \frac{\lceil \frac{P}{N_1'} \rceil \lceil \frac{R}{N_2'} \rceil}{2^i B} \rceil \tau$. When $\lceil \frac{P}{N_1'} \rceil \lceil \frac{R}{N_2'} \rceil \geq N_3'$, it is an *all-to-all reduction*, and the communication complexity of the reduction is approximately $(1 - \frac{1}{N_3'}) \lceil \frac{P}{N_1'} \rceil \lceil \frac{R}{N_2'} \rceil t_c + \sum_{i=1}^{n_3'} \lceil \frac{\lceil \frac{P}{N_1'} \rceil \lceil \frac{R}{N_2'} \rceil}{2^i B} \rceil \tau$. For a detailed complexity analysis, see [9]. The code is given below.

/* Algorithm A(·,3,1): */
/* Matrix C is partitioned as $N_1' \times N_2' N_3'$. */
$lc3(\alpha_1, \alpha_2, \alpha_3, i', j'')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{P}{N_1'}) \times [0 : \frac{Q}{N_2' N_3'}) =$
$\quad c(\alpha_1 \frac{P}{N_1'} + i', \alpha_3 \frac{Q}{N_3'} + \alpha_2 \frac{Q}{N_2' N_3'} + j'')$,
/* Matrix D is partitioned as $N_1' N_3' \times N_2'$. */
$ld3(\alpha_1, \alpha_2, \alpha_3, j', k'')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{Q}{N_1' N_3'}) \times [0 : \frac{R}{N_2'}) =$
$\quad d(\alpha_3 \frac{Q}{N_3'} + \alpha_1 \frac{Q}{N_1' N_3'} + j'', \alpha_2 \frac{R}{N_2'} + k')$,
/* Broadcast C along N_2' direction. */
$lc3\_row(\alpha_1, \alpha_2, \alpha_3, i', j')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{P}{N_1'}) \times [0 : \frac{Q}{N_3'}) =$
$\quad lc3(\alpha_1, \lfloor \frac{j' N_2' N_3'}{Q} \rfloor, \alpha_3, i', j' \bmod \frac{Q}{N_2' N_3'})$,
/* Broadcast D along N_1' direction. */
$ld3\_col(\alpha_1, \alpha_2, \alpha_3, j', k')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{Q}{N_3'}) \times [0 : \frac{R}{N_2'}) =$
$\quad ld3(\lfloor \frac{j' N_1' N_3'}{Q} \rfloor, \alpha_2, \alpha_3, j' \bmod \frac{Q}{N_1' N_3'}, k')$,
/* Compute partial inner product locally. */
$la3(\alpha_1, \alpha_2, \alpha_3, i', k')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{P}{N_1'}) \times [0 : \frac{R}{N_2'}) =$
$\quad \backslash + [lc3\_row(\alpha_1, \alpha_2, \alpha_3, i', j') * ld3\_col(\alpha_1, \alpha_2, \alpha_3, j', k') | 0 \leq j' < \frac{Q}{N_3'}]$,
/* Reduction along N_3' direction. */
$la3\_red(\alpha_1, \alpha_2, \alpha_3, i', k')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
$\quad \backslash + [la3(\alpha_1, \alpha_2, \alpha_3', \lfloor \frac{\alpha_3 N_2'}{N_2} \rfloor \frac{P}{N_1} + i', (\alpha_3 \bmod \frac{N_2}{N_2'}) \frac{R}{N_2} + k') | 0 \leq \alpha_3' < N_3']$,
/* Relabeling processor indices as two-dimensional. */
$la2(\alpha_1, \alpha_2, i', k')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
$\quad la3\_red(\lfloor \frac{\alpha_1 N_1'}{N_1} \rfloor, \lfloor \frac{\alpha_2 N_2'}{N_2} \rfloor, (\alpha_1 \bmod \frac{N_1}{N_1'}) \frac{N_2}{N_2'} + \alpha_2 \bmod \frac{N_2}{N_2'}, i', k')$,

/* Resulting matrix A is partitioned as $N_1 \times N_2$. */
$a(i,k)$ **over** $[0:P) \times [0:R) = la2(\lfloor \frac{iN_1'}{P} \rfloor, \lfloor \frac{kN_2'}{R} \rfloor, i \bmod \frac{P}{N_1'}, k \bmod \frac{R}{N_2'})$.

Functions $lc3$ and $ld3$ assign the initial matrix elements of C and D, respectively, to the emulated three-dimensional array $N_1' \times N_2' \times N_3'$. *lc3_row* broadcasts C along the second direction (N_2). *ld3_col* broadcasts D along the first direction (N_1). As a result, each processor can compute a partial inner product locally as defined by $la3$, followed by a reduction along the third direction (N_3') as defined by *la3_red*. $la2$ is introduced as an intermediate data structure to relabel the processors as a two-dimensional array, followed by the function a that transforms the local index to a global index using the processor addresses. Note, that $la2$ and a can be combined into one function, but the resulting function looks complicated.

In the above algorithm, the matrix A is partitioned into $N_1 \times N_2$ blocks with no extra communication after the reduction step. Depending on how the data set is divided during the reduction steps, the resulting matrix A can be partitioned into a form of $N_1' 2^x \times N_2' 2^{n_3'-x}$ blocks, where $0 \leq x \leq n_3'$, with the same communication complexity.

If the matrices C and D initially are partitioned into $N_1 \times N_2$ blocks, then transformations are required to change the allocation into $N_1' \times N_2' N_3'$ blocks and $N_1' N_3' \times N_2'$ blocks, respectively. The transformation can be specified as follows:

$lc3(\alpha_1, \alpha_2, \alpha_3, i', j'')$ **over** $[0:N_1') \times [0:N_2') \times [0:N_3') \times [0:\frac{P}{N_1'}) \times [0:\frac{Q}{N_2'N_3'}) =$
$\quad lc2(\alpha_1 \frac{N_1}{N_1'} + \lfloor \frac{i'N_1}{P} \rfloor, \alpha_3 \frac{N_2}{N_3'} + \lfloor \frac{\alpha_2 N_2}{N_2'N_3'} \rfloor, i' \bmod \frac{P}{N_1}, (\alpha_2 \bmod \frac{N_2'N_3'}{N_2}) \frac{Q}{N_2'N_3'} + j'')$,
$ld3(\alpha_1, \alpha_2, \alpha_3, j'', k')$ **over** $[0:N_1') \times [0:N_2') \times [0:N_3') \times [0:\frac{Q}{N_1'N_3'}) \times [0:\frac{R}{N_2'}) =$
$\quad ld2(\alpha_3 \frac{N_1}{N_3'} + \lfloor \frac{\alpha_1 N_1}{N_1'N_3'} \rfloor, \alpha_2 \frac{N_2}{N_2'} + \lfloor \frac{k'N_2}{R} \rfloor, (\alpha_1 \bmod \frac{N_1'N_3'}{N_1}) \frac{Q}{N_1'N_3'} + j'', k' \bmod \frac{R}{N_2})$.

The transformations can be implemented using algorithms that realize an arbitrary dimension permutation [6].

5 Conclusion

We have shown how algorithms for distributed architectures, such as a Boolean cube, can be expressed in terms of a shared global address space, and how the translation between local and global addresses can be carried out. We have also shown how the network and low level communication features of the architecture can be encapsulated into generic global communication primitives, such as *all-to-all broadcasting* within a (sub)cube, *all-to-all reduction*, and matrix transposition (dimension permutation). These primitives can either be integrated into compilers, or incorporated into the communication system by providing different communication modes. The communications would be transparent to the user. The architectural dependence is hidden in the communication primitives. Algorithms for matrix multiplication that we have used for illustration cover algorithms that parallelize one, two, or all three loops of a matrix

multiplication, and algorithms that are optimal for different matrix shapes and architectural parameters for each degree of parallelization.

Acknowledgement

We thank Eileen Connolly and Joe Rodrigue for proofreading the manuscript. The generous support of the Office of Naval Research under Contract No. N00014-86-K-0564 is gratefully acknowledged.

Appendix

A Communication primitives

A.1 One-dimensional partitioning

/* An nRSBT transpose algorithm (column part., n-port). */
$lc\_txp1(\alpha, u, i', j', t)$ **over** $[0:N) \times [0:n) \times [0:\frac{P}{2^t}) = \times [0:2^t\frac{Q}{nN}) \times [0:n]$
$\ll t = 0 \rightarrow lc(\alpha, i', u\frac{Q}{nN} + j'),$
$\lfloor \frac{\alpha}{2(u-t) \bmod n} \rfloor \bmod 2 = 0 \rightarrow$
$\ll 0 \le j < 2^{t-1}\frac{Q}{nN} \rightarrow lc\_txp1(\alpha, u, i', j', t-1)$
else $\rightarrow lc\_txp1(nbr(\alpha, (u-t) \bmod n), u, i', j' - 2^{t-1}\frac{Q}{nN}, t-1) \gg,$
else $\rightarrow$
$\ll 0 \le j < 2^{t-1}\frac{Q}{nN} \rightarrow lc\_txp1(nbr(\alpha, (u-t) \bmod n), u, i' + \frac{P}{2^t}, j', t-1)$
else $\rightarrow lc\_txp1(\alpha, u, i' + \frac{P}{2^t}, j' - 2^{t-1}\frac{Q}{nN}, t-1) \gg\gg,$
$lc\_txp(\alpha, i', j)$ **over** $[0:N) \times [0:\frac{P}{N}) \times [0:Q) =$
$lc\_txp1(\alpha, \lfloor \frac{jnN}{Q} \rfloor \bmod n, i', \lfloor \frac{jN}{Q} \rfloor \frac{Q}{nN} + j \bmod \frac{Q}{nN}, n).$

/* nRSBT reduction. */
/* Between columns, n-port, binary encoding. */
$la\_red1(\alpha, u, i', k', t)$ **over** $[0:N) \times [0:n) \times [0:\frac{P}{n}) \times [0:\frac{R}{2^t}) \times [0:n] =$
$\ll t = 0 \rightarrow la(\alpha, u\frac{P}{n} + i', sh(u, \lfloor \frac{k'N}{R} \rfloor)\frac{R}{N} + k' \bmod \frac{R}{N}),$
$\lfloor \frac{\alpha}{2(u-t) \bmod n} \rfloor \bmod 2 = 0 \rightarrow la\_red1(\alpha, u, i', k', t-1) + la\_red1(nbr(\alpha, (u-t) \bmod n), u, i', k', t-1),$
else $\rightarrow la\_red1(\alpha, u, i', k' + \frac{R}{2^t}, t-1) + la\_red1(nbr(\alpha, (u-t) \bmod n), u, i', k' + \frac{R}{2^t}, t-1) \gg,$
$la\_red(\alpha, i, k')$ **over** $[0:N) \times [0:P) \times [0:\frac{R}{N}) = la\_red1(\alpha, \lfloor \frac{in}{P} \rfloor, i \bmod \frac{P}{n}, k', n).$

A.2 Two-dimensional partitioning

/* SBT broadcasting (row direction, one-port). */

$lc\_row1(\alpha_1, \alpha_2, i', j', t)$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{P}{N_1}) \times [0:2^t\frac{Q}{N_2}) \times [0:n_2] =$
$\ll t = 0 \rightarrow lc(\alpha_1, \alpha_2, i', j'),$
$0 \le j' < 2^{t-1}\frac{Q}{N_2} \rightarrow lc\_row1(\alpha_1, \alpha_2, i', j', t-1),$
else $\rightarrow lc\_row1(\alpha_1, nbr(\alpha_2, t-1), i', j' - 2^{t-1}\frac{Q}{N_2}, t-1) \gg,$
$lc\_row(\alpha_1, \alpha_2, i', j)$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{P}{N_1}) \times [0:Q) = lc\_row1(\alpha_1, \alpha_2, i', j \oplus \alpha_2\frac{Q}{N_2}, n_2).$

/* nRSBT broadcasting (row direction, n-port). */
$lc\_row1(\alpha_1, \alpha_2, u, i', j', t)$ **over** $[0:N_1) \times [0:N_2) \times [0:n_2) \times [0:\frac{P}{n_2N_1}) \times [0:2^t\frac{Q}{N_2}) \times [0:n_2] =$
$\ll t = 0 \rightarrow lc(\alpha_1, \alpha_2, u\frac{P}{n_2N_1} + i', j'),$
$0 \le j' < 2^{t-1}\frac{Q}{N_2} \rightarrow lc\_row1(\alpha_1, \alpha_2, u, i', j', t-1),$
else $\rightarrow lc\_row1(\alpha_1, nbr(\alpha_2, (u+t-1) \bmod n_2), u, i', j' - 2^{t-1}\frac{Q}{N_2}, t-1) \gg,$
$lc\_row(\alpha_1, \alpha_2, u, i', j)$ **over** $[0:N_1) \times [0:N_2) \times [0:n_2) \times [0:\frac{P}{n_2N_1}) \times [0:Q) =$
$lc\_row1(\alpha_1, \alpha_2, \lfloor\frac{i'n_2N_1}{P}\rfloor, i' \bmod \frac{P}{n_2N_1}, (sh(u, \lfloor\frac{jN_2}{Q}\rfloor) \oplus \alpha_2)\frac{Q}{N_2}, n_2).$

/* SBT broadcasting (column direction, one-port). */
$ld\_col1(\alpha_1, \alpha_2, j', k', t)$ **over** $[0:N_1) \times [0:N_2) \times [0:2^t\frac{Q}{N_1}) \times [0:\frac{R}{N_2}) \times [0:n_1] =$
$\ll t = 0 \rightarrow ld(\alpha_1, \alpha_2, j', k'),$
$0 \le j' < 2^{t-1}\frac{Q}{N_1} \rightarrow ld\_col1(\alpha_1, \alpha_2, j', k', t-1),$
else $\rightarrow ld\_col1(nbr(\alpha_1, t-1), \alpha_2, j' - 2^{t-1}\frac{Q}{N_1}, k', t-1) \gg,$
$ld\_col(\alpha_1, \alpha_2, j, k')$ **over** $[0:N_1) \times [0:N_2) \times [0:Q) \times [0:\frac{R}{N_2}) = ld\_col1(\alpha_1, \alpha_2, j \oplus \alpha_1\frac{Q}{N_1}, k', n_1).$

/* nRSBT broadcasting (column direction, n-port). */
$ld\_col1(\alpha_1, \alpha_2, u, j', k', t)$ **over** $[0:N_1) \times [0:N_2) \times [0:n_1) \times [0:2^t\frac{Q}{N_1}) \times [0:\frac{R}{n_1N_2}) \times [0:n_1] =$
$\ll t = 0 \rightarrow ld(\alpha_1, \alpha_2, j', u\frac{R}{n_1N_2} + k'),$
$0 \le j' < 2^{t-1}\frac{Q}{N_1} \rightarrow ld\_col1(\alpha_1, \alpha_2, u, j', k', t-1),$
else $\rightarrow ld\_col1(nbr(\alpha_1, (u+t-1) \bmod n_1), \alpha_2, u, j' - 2^{t-1}\frac{Q}{N_1}, k', t-1) \gg,$
$ld\_col(\alpha_1, \alpha_2, u, j', k)$ **over** $[0:N_1) \times [0:N_2) \times [0:n_1) \times [0:Q) \times [0:\frac{R}{n_1N_2}) =$
$ld\_col1(\alpha_1, \alpha_2, \lfloor\frac{k'n_1N_2}{R}\rfloor, k' \bmod \frac{R}{n_1N_2}, (sh(u, \lfloor\frac{jN_1}{Q}\rfloor) \oplus \alpha_1)\frac{Q}{N_1}, n_1).$

B Matrix Multiplication

B.1 One-dimensional partitioning

/* A Gray code Exchange alg. A(1,1,1). */
/* Column partitioning, Gray code encoding. */
$lc(\hat{\alpha}, i, j', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{Q}{N}) \times [0:N) =$
$\ll t = 0 \rightarrow c(i, \hat{\alpha}\frac{Q}{N} + j'),$

else $\rightarrow lc(G^{-1}(nbr(G(\hat{\alpha}), T(t-1))), i, j', t-1) \gg,$
$ld(\hat{\alpha}, j, k')$ **over** $[0:N) \times [0:Q) \times [0:\frac{R}{N}) = d(j, \hat{\alpha}\frac{R}{N} + k'),$
$la(\hat{\alpha}, i, k', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{R}{N}) \times [0:N] =$
$\ll t = 0 \rightarrow 0,$
else $\rightarrow la(\hat{\alpha}, i, k', t-1) + (\backslash + [lc(\hat{\alpha}, i, j', t-1)$
$* ld(\hat{\alpha}, (\hat{\alpha} \oplus (t-1))\frac{Q}{N} + j', k') | 0 \leq j' < \frac{Q}{N}]) \gg,$
$a(i,k)$ **over** $[0:P) \times [0:R) = la(\lfloor \frac{kN}{R} \rfloor, i, k \bmod \frac{R}{N}, N).$

B.2 Two-dimensional partitioning

The index l in the following code denotes the rank of the $\frac{\max(N_1,N_2)}{\min(N_1,N_2)}$ blocks within each processor. The number of the communication steps after the initial alignment is $2\max(N_1, N_2) - 2$ in the code. It is possible to reduce it to $N_1 + N_2 - 2$ by a more complicated code.

/* Cannon's Algorithm A(1,2,1): */
/* $N_{max} = \max(N_1, N_2)$ and $N_{min} = \min(N_1, N_2)$. */
$lc(\hat{\alpha}_1, \hat{\alpha}_2, l, i', j', t)$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{N_{max}}{N_{min}}) \times [0:\frac{P}{N_{max}}) \times [0:\frac{Q}{N_{max}}) \times [0:N_{max}] =$
$\ll N_1 \geq N_2 \rightarrow$
$\ll t = 0 \rightarrow c(\hat{\alpha}_1\frac{P}{N_1} + i', \hat{\alpha}_2\frac{Q}{N_2} + l\frac{Q}{N_1} + j'),$
/* Initial alignment. */
$t = 1 \rightarrow lc(\hat{\alpha}_1, \lfloor \frac{((\hat{\alpha}_2\frac{N_1}{N_2} + l + \hat{\alpha}_1) \bmod N_1)N_2}{N_1} \rfloor, (l + \hat{\alpha}_1) \bmod \frac{N_1}{N_2}, i', j', 0),$
/* The last block gets from next proc. */
$l = \frac{N_{max}}{N_{min}} - 1 \rightarrow lc(\hat{\alpha}_1, (\hat{\alpha}_2 + 1) \bmod N_2, 0, i', j', t-1),$
/* Other blocks get from right locally. */
else $\rightarrow lc(\hat{\alpha}_1, \hat{\alpha}_2, l+1, i', j', t-1) \gg,$
else $\rightarrow$
$\ll t = 0 \rightarrow c(\hat{\alpha}_1\frac{P}{N_1} + l\frac{P}{N_2} + i', \hat{\alpha}_2\frac{Q}{N_2} + j'),$
$t = 1 \rightarrow lc(\hat{\alpha}_1, (\hat{\alpha}_1\frac{N_2}{N_1} + l + \hat{\alpha}_2) \bmod N_2, l, i', j', 0),$
else $\rightarrow lc(\hat{\alpha}_1, (\hat{\alpha}_2 + 1) \bmod N_2, l, i', j', t-1) \gg\gg,$
$ld(\hat{\alpha}_1, \hat{\alpha}_2, l, j', k', t)$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{N_{max}}{N_{min}}) \times [0:\frac{Q}{N_{max}}) \times [0:\frac{R}{N_{max}}) \times [0:N_{max}] =$
$\ll N_1 \leq N_2 \rightarrow$
$\ll t = 0 \rightarrow d(\hat{\alpha}_1\frac{Q}{N_1} + l\frac{Q}{N_2} + j', \hat{\alpha}_2\frac{R}{N_2} + k'),$
$t = 1 \rightarrow ld(\lfloor \frac{((\hat{\alpha}_1\frac{N_2}{N_1} + l + \hat{\alpha}_2) \bmod N_2)N_1}{N_2} \rfloor, \hat{\alpha}_2, \lfloor \frac{(l+\hat{\alpha}_2)N_2}{N_1} \rfloor \bmod \frac{N_2}{N_1}, j', k', 0),$
$l = \frac{N_{max}}{N_{min}} - 1 \rightarrow ld((\hat{\alpha}_1 + 1) \bmod N_1, \hat{\alpha}_2, 0, j', k', t-1),$
else $\rightarrow ld(\hat{\alpha}_1, \hat{\alpha}_2, l+1, j', k', t-1) \gg,$
else $\rightarrow$
$\ll t = 0 \rightarrow d(\hat{\alpha}_1\frac{Q}{N_1} + j', \hat{\alpha}_2\frac{R}{N_2} + l\frac{R}{N_1} + k'),$
$t = 1 \rightarrow ld((\hat{\alpha}_2\frac{N_1}{N_2} + l + \hat{\alpha}_1) \bmod N_1, \hat{\alpha}_2, l, j', k', 0),$
else $\rightarrow ld((\hat{\alpha}_1 + 1) \bmod N_1, \hat{\alpha}_2, l, j', k', t-1) \gg\gg,$
$la(\hat{\alpha}_1, \hat{\alpha}_2, l, i', k', t)$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{N_{max}}{N_{min}}) \times [0:\frac{P}{N_{max}}) \times [0:\frac{R}{N_{max}}) \times [0:N_{max}] =$
$\ll t = 0 \rightarrow 0,$

else $\rightarrow la(\hat{\alpha}_1, \hat{\alpha}_2, l, i', k', t-1) + (\backslash + [lc(\hat{\alpha}_1, \hat{\alpha}_2, l, i', j', t) * ld(\hat{\alpha}_1, \hat{\alpha}_2, l, j', k', t) | 0 \leq j' < \frac{Q}{N_{max}}]) \gg,$

$a(i,k)$ **over** $[0:P) \times [0:R) =$

$\ll N_1 \geq N_2 \rightarrow la(\lfloor \frac{iN_1}{P} \rfloor, \lfloor \frac{kN_2}{R} \rfloor, \lfloor \frac{kN_1}{R} \rfloor \bmod \frac{N_1}{N_2}, i \bmod \frac{P}{N_1}, k \bmod \frac{R}{N_1}, N_1),$

else $\rightarrow la(\lfloor \frac{iN_1}{P} \rfloor, \lfloor \frac{kN_2}{R} \rfloor, \lfloor \frac{iN_2}{P} \rfloor \bmod \frac{N_2}{N_1}, i \bmod \frac{P}{N_2}, k \bmod \frac{R}{N_2}, N_2) \gg.$

/* Algorithm A(·,2,4): */

$ld\_txp(\alpha_1, \alpha_2, j', k')$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{Q}{N_2}) \times [0:\frac{R}{N_1}) =$

$ld(\lfloor (\alpha_1 \frac{Q}{N_2} + j')/\frac{Q}{N_1} \rfloor, \lfloor (\alpha_2 \frac{R}{N_1} + k')/\frac{R}{N_2} \rfloor, (\alpha_1 \frac{Q}{N_2} + j') \bmod \frac{Q}{N_1}, (\alpha_2 \frac{R}{N_1} + k') \bmod \frac{R}{N_2}),$

$ld\_txp\_row(\alpha_1, \alpha_2, i, j')$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{Q}{N_2}) \times [0:R) =$

$ld\_txp(\alpha_1, \lfloor \frac{kN_1}{R} \rfloor, j', k \bmod \frac{R}{N_1}),$

$la(\alpha_1, \alpha_2, i', k)$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{P}{N_1}) \times [0:R) =$

$\backslash + [lc(\alpha_1, \alpha_2, i', j') * ld\_txp\_col(\alpha_1, \alpha_2, j', k) | 0 \leq j' < \frac{Q}{N_2}],$

$la\_red(\alpha_1, \alpha_2, i', k')$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{P}{N_1}) \times [0:\frac{R}{N_2}) =$

$\backslash + [la(\alpha_1, \alpha_2', i', \alpha_2 \frac{R}{N_1} + k') | 0 \leq \alpha_2' < N_2].$

/* Algorithm A(·,2,5): */

$ld\_txp(\alpha_1, \alpha_2, j', k')$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{Q}{N_2}) \times [0:\frac{R}{N_1}) =$

$ld(\lfloor (\alpha_1 \frac{Q}{N_2} + j')/\frac{Q}{N_1} \rfloor, \lfloor (\alpha_2 \frac{R}{N_1} + k')/\frac{R}{N_2} \rfloor,$
$(\alpha_1 \frac{Q}{N_2} + j') \bmod \frac{Q}{N_1}, (\alpha_2 \frac{R}{N_1} + k') \bmod \frac{R}{N_2}),$

$lc\_col(\alpha_1, \alpha_2, i, j')$ **over** $[0:N_1) \times [0:N_2) \times [0:P) \times [0:\frac{Q}{N_2}) =$

$lc(\lfloor \frac{iN_1}{P} \rfloor, \alpha_2, i \bmod \frac{P}{N_1}, j'),$

$la\_txp(\alpha_1, \alpha_2, i, k')$ **over** $[0:N_1) \times [0:N_2) \times [0:P) \times [0:\frac{R}{N_1}) =$

$\backslash + [lc\_col(\alpha_1, \alpha_2, i, j') * ld\_txp(\alpha_1, \alpha_2, j', k') | 0 \leq j < \frac{Q}{N_2}],$

$la\_txp\_red(\alpha_1, \alpha_2, i', k')$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{P}{N_2}) \times [0:\frac{R}{N_1}) =$

$\backslash + [la\_txp(\alpha_1, \alpha_2', \alpha_2 \frac{P}{N_2} + i', k') | 0 \leq \alpha_2' < N_2],$

$la(\alpha_1, \alpha_2, i', k')$ **over** $[0:N_1) \times [0:N_2) \times [0:\frac{P}{N_1}) \times [0:\frac{R}{N_2}) =$

$la\_txp\_red(\lfloor (\alpha_1 \frac{P}{N_1} + i')/\frac{P}{N_2} \rfloor, \lfloor (\alpha_2 \frac{R}{N_2} + k')/\frac{R}{N_1} \rfloor, (\alpha_1 \frac{P}{N_1} + i') \bmod \frac{P}{N_2}, (\alpha_2 \frac{R}{N_2} + k') \bmod \frac{R}{N_1}).$

References

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

[2] Marina C. Chen. Very-high-level parallel programming in crystal. In Michael T. Heath, editor, *Hypercube Multiprocessors 1987*, Society for Industrial and Applied Mathematics, Philadelphia, PA, 1987.

[3] Marina C. Chen, Young-il Choo, and Jingke Li. Crystal: from functional description to efficient parallel code. In *Proc. of the Third Conf. on Hypercube Concurrent Computers and Applications, 1988*, ACM, 1988.

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

[5] Ching-Tien Ho and S. Lennart Johnsson. Algorithms for dimension permutations on boolean cubes. In *The Third Hypercube Conference*, ACM, 1988. YALEU/DCS/RR-620.

[6] Ching-Tien Ho and S. Lennart Johnsson. *Stable dimension permutation on Boolean cubes.* Technical Report YALEU/DCS/RR-617, Dept. of Computer Science, Yale Univ., New Haven, CT, March 1988.

[7] J.W. Hong and H.T. Kung. I/O complexity: the red-blue pebble game. In *Proc. of the 13th ACM Symposium on the Theory of Computation*, pages 326–333, ACM, 1981.

[8] 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).

[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. Matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. Matrix Anal. Appl.*, 9(3):419–454, July 1988. YALE/DCS/RR-572, September 1987. (Revised edition of YALEU/DCS/RR-494 November 1986.).

[11] 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. To appear in IEEE Trans. Computers.

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

[13] 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.