Expressing Boolean Cube Matrix Algorithms in Shared Memory Primitives

S. Lennart Johnsson and Ching-Tien Ho

YALEU/DCS/TR-623
April 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.

† To appear in the Proceedings for the Third Conference on Hypercube Concurrent Computers and Applications, January, 1988, Pasadena, CA.

Expressing Boolean Cube Matrix Algorithms in Shared Memory Primitives

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

Abstract. The multiplication of (large) matrices allocated evenly on Boolean cube configured multiprocessors poses several interesting trade-offs with respect to communication time, processor utilization, and storage requirement. In [7] we investigated several algorithms for different degrees of parallelization, and showed how the choice of algorithm with respect to performance depends on the matrix shape, and the multiprocessor parameters, and how processors should be allocated optimally to the different loops.

In this paper the focus is on expressing the algorithms in shared memory type primitives. We assume that all processors share the same global address space, and present communication primitives both for 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, or 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 a storage need proportional to the number of processors and the matrix sizes (for a one-dimensional partitioning of the matrices).

1 Preliminaries

Throughout the paper, $N = 2^n$ denotes the number of processors of an n-dimensional Boolean cube, or n-cube. With respect to algorithms and data structures we factor N as $N_1 \times N_2$ ($2^{n_1} \times 2^{n_2}$) for two-dimensional partitionings of matrices, or as $N_1' \times N_2' \times N_3'$ ($2^{n_1'} \times 2^{n_2'} \times 2^{n_3'}$) for three-dimensional partitionings (defined later). We consider the matrix operation $A \leftarrow C \times D$, where all matrices are dense, C a $P \times Q$ matrix, D a $Q \times R$ matrix, and A a $P \times R$ matrix. We present algorithms for different initial and final allocations of the matrices: one-dimensional (column or row), two-dimensional (block), and three-dimensional partitionings. The two initial matrices C and D and resulting matrix A are assumed to be distributed among all the processors in the same manner, except in the three-dimensional case.

A matrix element is assigned to only one processor initially. With P, Q, and R being powers of two, $P = 2^p$, $Q = 2^q$, and $R = 2^r$, matrix element c_{ij} of matrix C, $0 \leq i < P$, $0 \leq j < Q$, is assigned to a processor as shown in Table 1 for one- or two-dimensional partitionings,

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

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	$(G(i_{p-1}i_{p-2}\cdots i_{p-n_1})\Vert$ $G(j_{q-1}j_{q-2}\cdots j_{q-n_2}))$
	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	$(G(i_{n_1-1}i_{n_1-2}\cdots i_0)\Vert$ $G(j_{n_2-1}j_{n_2-2}\cdots j_0))$

Table 1: Various ways of assigning matrix elements into processors.

consecutive or *cyclic* storage [6], and binary or Gray code encoding [10,6]. 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 matrix element d_{jk} (of D) or a_{ik} (of A) is obtained.

For a three-dimensional partitioning, 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 $N_1' \times N_2' N_3'$ blocks, or as $N_1' N_3' \times N_2'$ blocks, or into a form in-between these forms with the same communication complexity. If the matrices C and D are initially partitioned into a $N_1 \times N_2$ processor array, then some communications 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 the data communication for the matrix multiplication, except if there are very few elements per processor.

In the following, all algorithms are described in a Crystal-like notation [2]. 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) [9], and matrix transposition (dimension permutation).

In the Crystal-like codes each function of l parameters may be optionally followed by an expression " **over** domain$_1\times$ domain$_2\times\cdots\times$ 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*$_1$,
cond$_2\rightarrow$ *result*$_2$,
else $\rightarrow$ *result*$_3\gg$,

reads as "if *cond*$_1$ then *result*$_1$, else if *cond*$_2$, then *result*$_2$, else *result*$_3$". $\backslash\!\!+[f(j)*g(j)|0\le j<x]$ denotes $\sum_{j=0}^{x-1}(f(j)*g(j))$. We use $c(i,j)$, $0\le i<P$, $0\le j<Q$, 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 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.

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 $\hat{\alpha}=G(pid)$, where G is the binary-reflected Gray code encoding function.

For the analysis 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*, for which communication only can take place on one port at a time, and *n-port communication*, for which all ports on each processor can be used concurrently.

2 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 [9,11]. Matrix transposition with one-dimensional partitioning has the same communication pattern as *all-to-all personalized communication* [9], also known as *complete exchange* [11]. With a two-dimensional square partitioning into $\sqrt{N}\times\sqrt{N}$ blocks, optimal algorithms are described in [8,11]. 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 2, 3 and 4. 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	$(N-1)M$	n
n-port	nRSBT	$\frac{(N-1)M}{n}$	n

Table 2: 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 3: Communication complexity of all-to-all reduction on an n-cube with M elements per processor initially.

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

Table 4: Communication complexity of all-to-all personalized communication with M elements per processor initially.

2.1 One-dimensional Matrix Partitionings

The code for all-to-all broadcasting based on N translated Spanning Binomial Trees (SBT's) [9] with *one-port communication* is described below.

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

For Gray code encoding, α is replaced by $\hat{\alpha}$ and lc_brd is redefined as:

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

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) [9] are all optimal within constant factors. The algorithm for nRSBT broadcasting is:

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

With *one-port communication*, the code is described below. The code for *n-port communication* is included in appendix A.

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

With *one-port communication*, the code is described below. The code for *n-port communication* is included in appendix A.

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

2.2 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 [7]. The code for *one-port communication* is given below.

```
/* SBT tranpose alg. with $N_1 \times N_2$ partitioning. */
$f(t)$ over $[0:n] =$
    $\ll t \leq 2\min(n_1, n_2) \rightarrow 1,$
      $n_1 > n_2 \rightarrow 2^{t - 2\min(n_1, n_2)},$
```

```
        else $\to 2^{2\min(n_1,n_2)-t} \gg$,
$lc\_txp1(\alpha_1,\alpha_2,i',j',t)$ over $[0:N_1)\times[0:N_2)\times[0:f(t)\frac{P}{N_1})\times[0:\frac{Q}{N_2f(t)})\times[0:n] =$
    $\ll t=0 \to lc(\alpha_1,\alpha_2,i',j')$,
        $t \le 2\min(n_1,n_2) \to$
            /* two-dimensional transpose. */
            $\ll \lfloor\frac{\alpha_1}{2^{n_1-\lceil t/2\rceil}}\rfloor \bmod 2 = \lfloor\frac{\alpha_2}{2^{n_2-\lceil t/2\rceil}}\rfloor \bmod 2 \to$
                $\ll t \bmod 2 = 1 \to lc\_txp1(\alpha_1 \oplus 2^{n_1-\lceil t/2\rceil},\alpha_2,i',j',t-1) \gg$,
            else $\to$
                $\ll t \bmod 2 = 0 \to lc\_txp1(\alpha_1,\alpha_2 \oplus 2^{n_2-\lceil t/2\rceil},i',j',t-1) \gg\gg$,
        /* one-dimensional transpose. */
        $n_1 < n_2 \to$
            $\ll \frac{\alpha_2}{\lceil 2^{n-t}\rceil} \bmod 2 = 0 \to$
                $\ll 0 \le j' < 2^{t'-1}\frac{Q}{N_2} \to lc\_txp1(\alpha_1,\alpha_2,i',j',t-1)$,
                  else $\to lc\_txp1(\alpha_1,\alpha_2 \oplus 2^{n-t},i',j'-2^{t'-1}\frac{Q}{N_2},t-1) \gg$,
            else $\to$
                $\ll 0 \le j' < 2^{t'-1}\frac{Q}{N_2} \to lc\_txp1(\alpha_1,\alpha_2 \oplus 2^{n-t},i'+\frac{P}{2^{n_1+t'}},j',t-1)$,
                  else $\to lc\_txp1(\alpha_1,\alpha_2,i'+\frac{P}{2^{n_1+t'}},j'-2^{t'-1}\frac{Q}{N_2},t-1) \gg\gg$,
        else $\to$
            $\ll \frac{\alpha_1}{\lceil 2^{n-t}\rceil} \bmod 2 = 0 \to$
                $\ll 0 \le i' < 2^{t'-1}\frac{P}{N_1} \to lc\_txp1(\alpha_1,\alpha_2,i',j',t-1)$,
                  else $\to lc\_txp1(\alpha_1 \oplus 2^{n-t},\alpha_2,i'-2^{t'-1}\frac{P}{N_1},j',t-1) \gg$,
            else $\to$
                $\ll 0 \le i' < 2^{t'-1}\frac{P}{N_1} \to lc\_txp1(\alpha_1 \oplus 2^{n-t},\alpha_2,i',j'+\frac{Q}{2^{n_2+t'}},t-1)$,
                  else $\to lc\_txp1(\alpha_1,\alpha_2,i'-2^{t'-1}\frac{P}{N_1},j'+\frac{Q}{2^{n_2+t'}},t-1) \gg\gg\gg$
        where $t' = t - 2\min(n_1,n_2)$,
$lc\_txp(\alpha_1,\alpha_2,i',j')$ over $[0:N_1)\times[0:N_2)\times[0:\frac{P}{N_2})\times[0:\frac{Q}{N_1}) = lc\_txp1(\alpha_1,\alpha_2,i',j',n)$
```

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 [4,5] and employing one of those algorithms 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.

3 Matrix multiplication

3.1 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* of C from every processor that has elements of C to every processor that has elements of D. Processor $\alpha = PID(j)$

Column Partitioning:

$$\begin{aligned}
\mathcal{A}(\cdot,1,1): &\quad C_{*\alpha}, D_{*\alpha} \xRightarrow{\text{brd. C}, \leftrightarrow} C_{**}, D_{*\alpha} \xRightarrow{\text{mpy., [R]}} A_{*\alpha} \\
\mathcal{A}(\cdot,1,3): &\quad C_{*\alpha}, D_{*\alpha} \xRightarrow{\text{txp. C}, \nearrow} C_{\alpha*}, D_{*\alpha} \xRightarrow{\text{brd. D}, \leftrightarrow} C_{\alpha*}, D_{**} \xRightarrow{\text{mpy., [P]}} A_{\alpha*} \xRightarrow{\text{txp. A}, \nearrow} A_{*\alpha} \\
\mathcal{A}(\cdot,1,4): &\quad C_{*\alpha}, D_{*\alpha} \xRightarrow{\text{txp. D}, \nearrow} C_{*\alpha}, D_{\alpha*} \xRightarrow{\text{mpy., [Q]}} A^{\alpha}_{**} \xRightarrow{\text{red. A}, \leftrightarrow} A_{*\alpha}
\end{aligned}$$

Row Partitioning:

$$\begin{aligned}
\mathcal{A}(\cdot,1,1): &\quad C_{\alpha*}, D_{\alpha*} \xRightarrow{\text{brd. D}, \updownarrow} C_{\alpha*}, D_{**} \xRightarrow{\text{mpy., [P]}} A_{\alpha*} \\
\mathcal{A}(\cdot,1,3): &\quad C_{\alpha*}, D_{\alpha*} \xRightarrow{\text{txp. D}, \nearrow} C_{\alpha*}, D_{*\alpha} \xRightarrow{\text{brd. C}, \updownarrow} C_{**}, D_{*\alpha} \xRightarrow{\text{mpy., [R]}} A_{*\alpha} \xRightarrow{\text{txp. A}, \nearrow} A_{\alpha*} \\
\mathcal{A}(\cdot,1,4): &\quad C_{\alpha*}, D_{\alpha*} \xRightarrow{\text{txp. C}, \nearrow} C_{*\alpha}, D_{\alpha*} \xRightarrow{\text{mpy., [Q]}} A^{\alpha}_{**} \xRightarrow{\text{red. A}, \updownarrow} A_{\alpha*}
\end{aligned}$$

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

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 1.

- **Algorithm $\mathcal{A}$(·,1,2).** Compute A by a transpose of C and *broadcasting* of 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 $\mathcal{A}$(·,1,3).** Compute A by a transpose of C, *broadcasting* of D from every processor that has elements of D to every processor that has elements of C^T, and transpose A^T. Processor $\alpha = PID(j)$ computes $C(\lfloor \frac{j}{\lceil \frac{P}{N} \rceil} \rfloor, *)D$.

- **Algorithm $\mathcal{A}$(·,1,4).** Compute A *in-space* by a transpose of 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 $\mathcal{A}$(·,1,2) is clearly inferior to algorithm $\mathcal{A}$(·,1,1) with respect to communication complexity, and is not further considered for the one-dimensional partitioning, but 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 is also included for comparison. The two subscripts denote the ordinal numbers 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 $\mathcal{A}$(·,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 $[0:N) \times [0:P) \times [0:\frac{Q}{N}) \times [0:N) =$
```

```
    $\ll t = 0 \to c(i, \hat{\alpha}\frac{Q}{N} + j'),$
        **else** $\to lc((\hat{\alpha}+1) \bmod N, i, j', t-1) \gg,$
$ld(\hat{\alpha}, j, k')$ **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** $[0:N) \times [0:P) \times [0:\frac{R}{N}) \times [0:N] =$
    $\ll t = 0 \to 0,$
        **else** $\to la(\hat{\alpha}, i, k', t-1) + (\backslash\!+ [lc(\hat{\alpha}, i, j', t-1)$
                $* ld(\hat{\alpha}, ((\hat{\alpha}+t-1) \bmod N)\frac{Q}{N} + j', k') | 0 \le j' < \frac{Q}{N}]) \gg,$
$a(i,k)$ **over** $[0:P) \times [0:R) = la(\lfloor \frac{kN}{R} \rfloor, i, k \bmod \frac{R}{N}, N)$
```

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.

Note that 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 $\hat{\alpha} = G(pid) = j$, rotation of C implies nearest-neighbor communications. For binary encoding, i.e., the jth block column is stored in processor $\alpha = j$, we redefine lc and la as follows:

```
/* $G(t)$ is the binary-reflected Gray code of $t$. */
$G(t) = t \oplus \lfloor \frac{t}{2} \rfloor,$
/* $G^{-1}$ is the inverse function of $G$. */
$G^{-1}(t) = \ll t = 0 \to 0,$
            **else** $\to t \oplus G^{-1}(\lfloor \frac{t}{2} \rfloor) \gg,$
$lc(\alpha, i, j', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{Q}{N}) \times [0:N) =$
    $\ll t = 0 \to c(i, \alpha\frac{Q}{N} + j'),$
        **else** $\to lc(G^{-1}((G(\alpha)+1) \bmod N), i, j', t-1) \gg,$
$la(\alpha, i, k', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{R}{N}) \times [0:N] =$
    $\ll t = 0 \to 0,$
        **else** $\to la(\alpha, i, k', t-1) + (\backslash\!+ [lc(\alpha, i, j', t-1)$
                $* ld(\alpha, G((G^{-1}(\alpha)+t-1) \bmod N)\frac{Q}{N} + j', k') | 0 \le j' < \frac{Q}{N}]) \gg$
```

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

```
**/* A Gray code Exchange alg. A(1,1,1). */**
/* Column partitioning, binary code encoding. */
/* $T(t)$ is the index of the $t$th transition bit in the Gray code
   on $n$ bits = the number of trailing 1's. */
$T(t) = \ll t \bmod 2 = 0 \to 0,$
           **else** $\to 1 + T(\lfloor \frac{t}{2} \rfloor) \gg,$
$lc(\alpha, i, j', t)$ **over** $[0:N) \times [0:P) \times [0:\frac{Q}{N}) \times [0:N) =$
    $\ll t = 0 \to c(i, \alpha\frac{Q}{N} + j'),$
```

```
        else $\rightarrow lc(\alpha \oplus 2^{T(t-1)}, i, j', t-1) \gg$,
$ld(\alpha, j, k')$ 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 $\rightarrow la(\alpha, i, k', t-1) + (\backslash + [lc(\alpha, i, j', t-1)$
                $* ld(\alpha, (\alpha \oplus G(t-1))\frac{Q}{N} + j', k') | 0 \le j' < \frac{Q}{N}]) \gg$,
$a(i,k)$ over $[0:P) \times [0:R) = la(\lfloor \frac{kN}{R} \rfloor, i, k \bmod \frac{R}{N}, N)$
```

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 [3] algorithms, respectively. The encodings only affect which N block rows of D within each processor interact with the current block column of C.

In the case 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. Each such piece is broadcast through a unique path. In the case of the Gray code exchange algorithm the paths are obtained through rotation of the dimensions, such that if the edges in dimension $T(t)$ are used by path 0 during step t, then path u uses the edges in 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 $\rightarrow lc(\alpha \oplus 2^{(T(t-1)+u) \bmod n}, u, i', j', t-1) \gg$,
$ld(\alpha, j, k')$ over $[0:N) \times [0:Q) \times [0:\frac{R}{N}) = d(j, \alpha\frac{R}{N} + k')$,
/* Shuffle (cyclic left-shift) u steps of t. */
$sh(u,t)$ over $[0:n) \times [0:N) = (t \bmod 2^{n-u})2^u + \lfloor \frac{t}{2^{n-u}} \rfloor$,
$la(\alpha, u, i', k', t)$ 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 $\rightarrow la(\alpha, u, i', k', t-1) + (\backslash + [lc(\alpha, u, i', j', t-1)$
                $* ld(\alpha, (\alpha \oplus (sh(u, G(t-1))))\frac{Q}{N} + j', k') | 0 \le j' < \frac{Q}{N}]) \gg$,
$a(i,k)$ over $[0:P) \times [0:R) = la(\lfloor \frac{kN}{R} \rfloor, \lfloor \frac{in}{P} \rfloor, i \bmod \frac{P}{n}, k \bmod \frac{R}{N}, N)$
```

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 the 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 $A$. */
$a(i, k)$ 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 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 $[0:N) \times [0:P) \times [0:R) = \backslash+ [lc(\alpha, i, j') * ld\_txp(\alpha, j', k) | 0 \leq j' < \frac{Q}{N}]$,
$la\_red(\alpha, i, k')$ over $[0:N) \times [0:P) \times [0:\frac{R}{N}) = \backslash+ [la(\alpha', i, \alpha\frac{R}{N} + k') | 0 \leq \alpha' < N]$
```

Table 5 shows the total number of arithmetic operations in sequence. If P, Q, and R all are multiples of N, then all three algorithms have the same arithmetic complexity. For $P, Q, R \geq N$, the differences of the arithmetic complexities are within constant factors. Table 6 shows the total number of elements transferred in sequence and the minimum number of start-ups for $P, Q, R \geq N$. 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 described in [7]. 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 $P, Q, R \geq N$. 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 [7].

3.2 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

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 5: The arithmetic time for one-dimensional column partitioning.

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

form requires broadcasting of elements of C along rows and broadcasting of 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. [3] described such an algorithm making use of 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

- **Algorithm $\mathcal{A}(\cdot,2,1)$.** Compute A *in-place* by broadcasting of 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, 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 $C(\lfloor \frac{i}{\lceil \frac{P}{N_2} \rceil} \rfloor, \lfloor \frac{j}{\lceil \frac{Q}{N_1} \rceil} \rfloor)D(\lfloor \frac{j}{\lceil \frac{Q}{N_1} \rceil} \rfloor, *)$ 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, 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 the 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, 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

$$
\begin{array}{ll}
\mathcal{A}(\cdot,2,1): & C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xRightarrow{\text{brd. C, }\leftrightarrow} C_{\alpha_1 *}, D_{\alpha_1\alpha_2} \xRightarrow{\text{brd. D, }\updownarrow} C_{\alpha_1 *}, D_{*\alpha_2} \\
& \xRightarrow{\text{mpy., [PR]}} A_{\alpha_1\alpha_2} \\
\mathcal{A}(\cdot,2,2): & C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xRightarrow{\text{txp. C, }\nearrow} C_{\alpha_2\alpha_1}, D_{\alpha_1\alpha_2} \xRightarrow{\text{brd. C, }\leftrightarrow} C_{*\alpha_1}, D_{\alpha_1\alpha_2} \\
& \xRightarrow{\text{mpy., [QR]}} A^{\alpha_1}_{*\alpha_2} \xRightarrow{\text{red. A, }\updownarrow} A_{\alpha_1\alpha_2} \\
\mathcal{A}(\cdot,2,3): & C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xRightarrow{\text{txp. C, }\nearrow} C_{\alpha_2\alpha_1}, D_{\alpha_1\alpha_2} \xRightarrow{\text{brd. D, }\leftrightarrow} C_{\alpha_2\alpha_1}, D_{\alpha_1 *} \\
& \xRightarrow{\text{mpy., [PQ]}} A^{\alpha_1}_{\alpha_2 *} \xRightarrow{\text{red. A, }\updownarrow} A_{\alpha_2\alpha_1} \xRightarrow{\text{txp. A, }\nearrow} A_{\alpha_1\alpha_2} \\
\mathcal{A}(\cdot,2,4): & C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xRightarrow{\text{txp. D, }\nearrow} C_{\alpha_1\alpha_2}, D_{\alpha_2\alpha_1} \xRightarrow{\text{brd. D, }\updownarrow} C_{\alpha_1\alpha_2}, D_{\alpha_2 *} \\
& \xRightarrow{\text{mpy., [PQ]}} A^{\alpha_2}_{\alpha_1 *} \xRightarrow{\text{red. A, }\leftrightarrow} A_{\alpha_1\alpha_2} \\
\mathcal{A}(\cdot,2,5): & C_{\alpha_1\alpha_2}, D_{\alpha_1\alpha_2} \xRightarrow{\text{txp. D, }\nearrow} C_{\alpha_1\alpha_2}, D_{\alpha_2\alpha_1} \xRightarrow{\text{brd. C, }\updownarrow} C_{*\alpha_2}, D_{\alpha_2\alpha_1} \\
& \xRightarrow{\text{mpy., [QR]}} A^{\alpha_2}_{*\alpha_1} \xRightarrow{\text{red. A, }\leftrightarrow} A_{\alpha_2\alpha_1} \xRightarrow{\text{txp. A, }\nearrow} A_{\alpha_1\alpha_2}
\end{array}
$$

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

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 numbers 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 $\mathcal{A}(\cdot,2,1)$. But, unlike in the one-dimensional case algorithm $\mathcal{A}(\cdot,2,2)$ may have a higher processor utilization than algorithm $\mathcal{A}(\cdot,2,1)$.

The 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 enc. */
$lc(\hat{\alpha}_1, \hat{\alpha}_2, i', j', t)$ over $[0 : \sqrt{N}) \times [0 : \sqrt{N}) \times [0 : \frac{P}{\sqrt{N}}) \times [0 : \frac{Q}{\sqrt{N}}) \times [0 : \sqrt{N}) =$
    $\ll t = 0 \rightarrow c(\hat{\alpha}_1 \frac{P}{\sqrt{N}} + i', \hat{\alpha}_2 \frac{Q}{\sqrt{N}} + j'),$
        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 $[0 : \sqrt{N}) \times [0 : \sqrt{N}) \times [0 : \frac{Q}{\sqrt{N}}) \times [0 : \frac{R}{\sqrt{N}}) \times [0 : \sqrt{N}) =$
    $\ll t = 0 \rightarrow d(\hat{\alpha}_1 \frac{Q}{\sqrt{N}} + j', \hat{\alpha}_2 \frac{R}{\sqrt{N}} + k'),$
```

$\quad\quad \textbf{else} \rightarrow ld((\hat{\alpha}_1 + 1) \bmod \sqrt{N}, \hat{\alpha}_2, j', k', t-1) \gg,$
$la(\hat{\alpha}_1, \hat{\alpha}_2, i', k', t)$ **over** $[0 : \sqrt{N}) \times [0 : \sqrt{N}) \times [0 : \frac{P}{\sqrt{N}}) \times [0 : \frac{R}{\sqrt{N}}) \times [0 : \sqrt{N}] =$
$\quad \ll t = 0 \rightarrow 0,$
$\quad\quad \textbf{else} \rightarrow la(\hat{\alpha}_1, \hat{\alpha}_2, i', k', t-1) + (\backslash + [lc(\hat{\alpha}_1, \hat{\alpha}_2, i', j', t-1)$
$\quad\quad\quad\quad * ld(\hat{\alpha}_1, \hat{\alpha}_2, j', k', t-1) | 0 \leq j' < \frac{Q}{\sqrt{N}}]) \gg,$
$a(i,k)$ **over** $[0 : P) \times [0 : R) = la\_red(\lfloor \frac{i\sqrt{N}}{P} \rfloor, \lfloor \frac{k\sqrt{N}}{R} \rfloor, i \bmod \frac{P}{\sqrt{N}}, k \bmod \frac{R}{\sqrt{N}})$

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** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
$\quad \backslash + [lc\_row(\alpha_1, \alpha_2, i', j) * ld\_col(\alpha_1, \alpha_2, j, k') | 0 \leq j < Q]$

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

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

```
/* 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 $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_2}) \times [0 : R) =$
    \+ $[lc\_txp(\alpha_1, \alpha_2, i', j') * ld\_row(\alpha_1, \alpha_2, j', k) | 0 \le j < \frac{Q}{N_1}]$,
$la\_txp\_red(\alpha_1, \alpha_2, i', k')$ over $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_2}) \times [0 : \frac{R}{N_1}) =$
    \+ $[la\_txp(\alpha_1', \alpha_2, i', \alpha_1 \frac{R}{N_1} + k') | 0 \le \alpha_1' < N_1]$,
$la(\alpha_1, \alpha_2, i', k')$ over $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
    $la\_txp\_red(\lfloor\alpha_1 \frac{P}{N_1} + i'/\frac{P}{N_2}\rfloor, \lfloor\alpha_2 \frac{R}{N_2} + k'/\frac{R}{N_1}\rfloor, (\alpha_1 \frac{P}{N_1} + i') \bmod \frac{P}{N_2}, (\alpha_2 \frac{R}{2_1} + k') \bmod \frac{R}{N_1})$
```

Algorithms A(·,2,4) and A(·,2,5) are included in appendix B.

Table 7 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 8 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 9. The resulting total complexities are shown in Table 10. By considering the highest-order term, the data transfer times compare as $Q : P : R : R : P$ from $\mathcal{A}(1,2,1)$ to $\mathcal{A}(1,2,5)$. It can be shown [7] 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 11 shows the communication complexity with *n-port communication* and optimum packet size. For a detailed analysis and optimum choice of algorithms, see [7].

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 8: 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 9: 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 10: 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 11: The communication complexity for *optimum* buffer sizes, two-dimensional partitioning, and *n-port* communication.

3.3 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. The 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 detailed complexity analysis, see [7]. 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** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{P}{N_1'}) \times [0 : \frac{Q}{N_2'N_3'}) =$
    $c(\alpha_1 \frac{P}{N_1'} + i', \alpha_3 \frac{Q}{N_3'} + \alpha_2 \frac{Q}{N_2'N_3'} + j''),$
/* Matrix D is partitioned as $N_1'N_3' \times N_2'$. */
$ld3(\alpha_1, \alpha_2, \alpha_3, j', k'')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{Q}{N_1'N_3'}) \times [0 : \frac{R}{N_2'}) =$
    $d(\alpha_3 \frac{Q}{N_3'} + \alpha_1 \frac{Q}{N_1'N_3'} + j'', \alpha_2 \frac{R}{N_2'} + k'),$
/* Broadcast C along $N_2'$ direction. */
$lc3\_row(\alpha_1, \alpha_2, \alpha_3, i', j')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{P}{N_1'}) \times [0 : \frac{Q}{N_3'}) =$
    $lc3(\alpha_1, \lfloor \frac{j'N_2'N_3'}{Q} \rfloor, \alpha_3, i', j' \bmod \frac{Q}{N_2'N_3'}),$
/* Broadcast D along $N_1'$ direction. */
$ld3\_col(\alpha_1, \alpha_2, \alpha_3, j', k')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{Q}{N_3'}) \times [0 : \frac{R}{N_2'}) =$
    $ld3(\lfloor \frac{j'N_1'N_3'}{Q} \rfloor, \alpha_2, \alpha_3, j' \bmod \frac{Q}{N_1'N_3'}, k'),$
/* Compute partial inner product locally. */
$la3(\alpha_1, \alpha_2, \alpha_3, i', k')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{P}{N_1'}) \times [0 : \frac{R}{N_2'}) =$
    $\backslash + [lc3\_row(\alpha_1, \alpha_2, \alpha_3, i', j') * ld3\_col(\alpha_1, \alpha_2, \alpha_3, j', k') | 0 \le j' < \frac{Q}{N_3'}],$
/* Reduction along $N_3'$ direction. */
$la3\_red(\alpha_1, \alpha_2, \alpha_3, i', k')$ **over** $[0 : N_1') \times [0 : N_2') \times [0 : N_3') \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
    $\backslash + [la3(\alpha_1, \alpha_2, \alpha_3', \lfloor \frac{\alpha_3 N_2'}{N_2} \rfloor \frac{P}{N_1} + i', (\alpha_3 \bmod \frac{N_2}{N_2'}) \frac{R}{N_2} + k') | 0 \le \alpha_3' < N_3'],$
/* Relabeling processor indices as two-dimensional. */
$la2(\alpha_1, \alpha_2, i', k')$ **over** $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
    $la3\_red(\lfloor \frac{\alpha_1 N_1'}{N_1} \rfloor, \lfloor \frac{\alpha_2 N_2'}{N_2} \rfloor, (\alpha_1 \bmod \frac{N_1}{N_1'}) \frac{N_2}{N_2'} + \alpha_2 \bmod \frac{N_2}{N_2'}, i', k'),$
/* Resulting matrix A is partitioned as $N_1 \times N_2$. */
$a(i, k)$ **over** $[0 : P) \times [0 : R) = la2(\lfloor \frac{iN_1'}{P} \rfloor, \lfloor \frac{kN_2'}{R} \rfloor, i \bmod \frac{P}{N_1'}, k \bmod \frac{R}{N_2'})$
```

Note that 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 $N_1' \times N_2'N_3'$ blocks, $N_1'N_3' \times N_2'$ blocks, or some blocking scheme in-between those two.

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

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

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 $[0:N) \times [0:n) \times [0:\frac{P}{2^t}) = \times[0:2^t\frac{Q}{nN}) \times [0:n]$
    $\ll t = 0 \rightarrow lc(\alpha, i', u\frac{Q}{nN} + j')$,
      $\lfloor \frac{\alpha}{2^{(u-t)\bmod n}} \rfloor \bmod 2 = 0 \rightarrow$
          $\ll 0 \leq j < 2^{t-1}\frac{Q}{nN} \rightarrow lc\_txp1(\alpha, u, i', j', t-1)$
            else $\rightarrow lc\_txp1(\alpha \oplus 2^{(u-t)\bmod n}, u, i', j' - 2^{t-1}\frac{Q}{nN}, t-1) \gg$,
      else $\rightarrow$
          $\ll 0 \leq j < 2^{t-1}\frac{Q}{nN} \rightarrow lc\_txp1(\alpha \oplus 2^{(u-t)\bmod n}, u, i' + \frac{P}{2^t}, j', t-1)$
            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 $[0:N) \times [0:n) \times [0:\frac{P}{n}) \times [0:\frac{R}{2^t}) \times [0:n] =$
    $\ll t = 0 \rightarrow la(\alpha, u\frac{P}{n} + i', sh(u, \lfloor \frac{k'N}{R} \rfloor)\frac{R}{N} + k' \bmod \frac{R}{N})$,
      $\lfloor \frac{\alpha}{2^{(u-t)\bmod n}} \rfloor \bmod 2 = 0 \rightarrow la\_red1(\alpha, u, i', k', t-1) + la\_red1(\alpha \oplus 2^{(u-t)\bmod n}, u, i', k', t-1)$,
      else $\rightarrow la\_red1(\alpha, u, i', k' + \frac{R}{2^t}, t-1) + la\_red1(\alpha \oplus 2^{(u-t)\bmod n}, u, i', k' + \frac{R}{2^t}, t-1) \gg$,
```

$la\_red(\alpha,i,k')$ **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 $[0:N_1)\times[0:N_2)\times[0:\frac{P}{N_1})\times[0:2^t\frac{Q}{N_2})\times[0:n_2] =$
    $\ll t=0 \rightarrow lc(\alpha_1,\alpha_2,i',j'),$
      else $\rightarrow \ll 0\le j'<2^{t-1}\frac{Q}{N_2} \rightarrow lc\_row1(\alpha_1,\alpha_2,i',j',t-1),$
               else $\rightarrow lc\_row1(\alpha_1,\alpha_2\oplus 2^{t-1},i',j'-2^{t-1}\frac{Q}{N_2},t-1)\gg\gg,$
$lc\_row(\alpha_1,\alpha_2,i',j)$ 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 $[0:N_1)\times[0:N_2)\times[0:n_2)\times[0:\frac{P}{n_2N_1})\times[0:2^t\frac{Q}{N_2})\times[0:n_2] =$
    $\ll t=0 \rightarrow lc(\alpha_1,\alpha_2,u\frac{P}{n_2N_1}+i',j'),$
      else $\rightarrow \ll 0\le j'<2^{t-1}\frac{Q}{N_2} \rightarrow lc\_row1(\alpha_1,\alpha_2,u,i',j',t-1),$
               else $\rightarrow lc\_row1(\alpha_1,\alpha_2\oplus 2^{(u+t-1)\bmod n_2},u,i',j'-2^{t-1}\frac{Q}{N_2},t-1)\gg\gg,$
$lc\_row(\alpha_1,\alpha_2,u,i',j)$ 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 $[0:N_1)\times[0:N_2)\times[0:2^t\frac{Q}{N_1})\times[0:\frac{R}{N_2})\times[0:n_1] =$
    $\ll t=0 \rightarrow ld(\alpha_1,\alpha_2,j',k'),$
      else $\rightarrow \ll 0\le j'<2^{t-1}\frac{Q}{N_1} \rightarrow ld\_col1(\alpha_1,\alpha_2,j',k',t-1),$
               else $\rightarrow ld\_col1(\alpha_1\oplus 2^{t-1},\alpha_2,j'-2^{t-1}\frac{Q}{N_1},k',t-1)\gg\gg,$
$ld\_col(\alpha_1,\alpha_2,j,k')$ 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 $[0:N_1)\times[0:N_2)\times[0:n_1)\times[0:2^t\frac{Q}{N_1})\times[0:\frac{R}{n_1N_2})\times[0:n_1] =$
    $\ll t=0 \rightarrow ld(\alpha_1,\alpha_2,j',u\frac{R}{n_1N_2}+k'),$
      else $\rightarrow \ll 0\le j'<2^{t-1}\frac{Q}{N_1} \rightarrow ld\_col1(\alpha_1,\alpha_2,u,j',k',t-1),$
               else $\rightarrow ld\_col1(\alpha_1\oplus 2^{(u+t-1)\bmod n_1},\alpha_2,u,j'-2^{t-1}\frac{Q}{N_1},k',t-1)\gg\gg,$
$ld\_col(\alpha_1,\alpha_2,u,j',k)$ 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 $\rightarrow lc(G(G^{-1}(\hat{\alpha}) \oplus 2^{T(t-1)}), i, j', t-1) \gg,$
$ld(\hat{\alpha}, j, k')$ 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 $\rightarrow la(\hat{\alpha}, i, k', t-1) + (\backslash + [lc(\hat{\alpha}, i, j', t-1)$
                $* ld(\hat{\alpha}, (\hat{\alpha} \oplus (t-1))\frac{Q}{N} + j', k') | 0 \le j' < \frac{Q}{N}]) \gg,$
$a(i, k)$ 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 $N_{min} = \min(N_1, N_2)$. */
$lc(\alpha_1, \alpha_2, l, i', j', t)$ over $[0 : N_1) \times [0 : N_2) \times [0 : \frac{N_{max}}{N_{min}}) \times [0 : \frac{P}{N_{max}}) \times [0 : \frac{Q}{N_{max}}) \times [0 : N_{max}] =$
    $\ll N_1 \ge N_2 \rightarrow$
        $\ll t = 0 \rightarrow c(\alpha_1\frac{P}{N_1} + i', \alpha_2\frac{Q}{N_2} + l\frac{Q}{N_1} + j'),$
          /* Initial alignment. */
          $t = 1 \rightarrow lc(\alpha_1, \lfloor \frac{((\alpha_2\frac{N_1}{N_2} + l + \alpha_1) \bmod N_1)N_2}{N_1} \rfloor, (l + \alpha_1) \bmod \frac{N_1}{N_2}, i', j', 0),$
          /* The last block gets from next proc. */
          $l = \frac{N_{max}}{N_{min}} - 1 \rightarrow lc(\alpha_1, (\alpha_2 + 1) \bmod N_2, 0, i', j', t-1),$
          /* Other blocks get from right locally. */
          else $\rightarrow lc(\alpha_1, \alpha_2, l+1, i', j', t-1) \gg,$
    else $\rightarrow$
        $\ll t = 0 \rightarrow c(\alpha_1\frac{P}{N_1} + l\frac{P}{N_2} + i', \alpha_2\frac{Q}{N_2} + j'),$
          $t = 1 \rightarrow lc(\alpha_1, (\alpha_1\frac{N_2}{N_1} + l + \alpha_2) \bmod N_2, l, i', j', 0),$
          else $\rightarrow lc(\alpha_1, (\alpha_2 + 1) \bmod N_2, l, i', j', t-1) \gg\gg,$
$ld(\alpha_1, \alpha_2, l, j', k', t)$ over $[0 : N_1) \times [0 : N_2) \times [0 : \frac{N_{max}}{N_{min}}) \times [0 : \frac{Q}{N_{max}}) \times [0 : \frac{R}{N_{max}}) \times [0 : N_{max}] =$
    $\ll N_1 \le N_2 \rightarrow$
        $\ll t = 0 \rightarrow d(\alpha_1\frac{Q}{N_1} + l\frac{Q}{N_2} + j', \alpha_2\frac{R}{N_2} + k'),$
          $t = 1 \rightarrow ld(\lfloor \frac{((\alpha_1\frac{N_2}{N_1} + l + \alpha_2) \bmod N_2)N_1}{N_2} \rfloor, \alpha_2, \lfloor \frac{(l + \alpha_2)N_2}{N_1} \rfloor \bmod \frac{N_2}{N_1}, j', k', 0),$
        $l = \frac{N_{max}}{N_{min}} - 1 \rightarrow ld((\alpha_1 + 1) \bmod N_1, \alpha_2, 0, j', k', t-1),$
        $\textbf{else} \rightarrow ld(\alpha_1, \alpha_2, l+1, j', k', t-1) \gg,$
    $\textbf{else} \rightarrow$
      $\ll t = 0 \rightarrow d(\alpha_1 \frac{Q}{N_1} + j', \alpha_2 \frac{R}{N_2} + l \frac{R}{N_1} + k'),$
        $t = 1 \rightarrow ld((\alpha_2 \frac{N_1}{N_2} + l + \alpha_1) \bmod N_1, \alpha_2, l, j', k', 0),$
        $\textbf{else} \rightarrow ld((\alpha_1 + 1) \bmod N_1, \alpha_2, l, j', k', t-1) \gg\gg,$
$la(\alpha_1, \alpha_2, l, i', k', t)$ over $[0 : N_1) \times [0 : N_2) \times [0 : \frac{N_{max}}{N_{min}}) \times [0 : \frac{P}{N_{max}}) \times [0 : \frac{R}{N_{max}}) \times [0 : N_{max}] =$
  $\ll t = 0 \rightarrow 0,$
    $\textbf{else} \rightarrow la(\alpha_1, \alpha_2, l, i', k', t-1) + (\backslash{+}\ [lc(\alpha_1, \alpha_2, l, i', j', t-1)$
          $* ld(\alpha_1, \alpha_2, l, j', k', t-1) | 0 \leq j' < \frac{Q}{N_{max}}]) \gg,$
$a(i, k)$ over $[0 : P) \times [0 : R) =$
  $\ll N_1 \geq N_2 \rightarrow la(\lfloor \frac{iN_1}{P} \rfloor, \lfloor \frac{kN_2}{R} \rfloor, \lfloor \frac{kN_1}{R} \rfloor \bmod \frac{N_1}{N_2}, i \bmod \frac{P}{N_1}, k \bmod \frac{R}{N_1}, N_1),$
    $\textbf{else} \rightarrow la(\lfloor \frac{iN_1}{P} \rfloor, \lfloor \frac{kN_2}{R} \rfloor, \lfloor \frac{iN_2}{P} \rfloor \bmod \frac{N_2}{N_1}, i \bmod \frac{P}{N_2}, k \bmod \frac{R}{N_2}, N_2) \gg$
```

/* 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 $[0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : \frac{R}{N_2}) =$
    $la\_txp\_red(\lfloor \alpha_1 \frac{P}{N_1} + i' / \frac{P}{N_2} \rfloor, \lfloor \alpha_2 \frac{R}{N_2} + k' / \frac{R}{N_1} \rfloor, (\alpha_1 \frac{P}{N_1} + i') \bmod \frac{P}{N_2}, (\alpha_2 \frac{R}{2_1} + k') \bmod \frac{R}{N_1})$
```

Acknowledgement

The work has been supported by the Office of Naval Research under contracts N00014-86-K-0564.

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

[4] 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.

[5] 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.

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

[7] S. Lennart Johnsson 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.

[8] S. Lennart Johnsson and Ching-Tien Ho. Matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. on Algebraic and Discrete Methods*, . To appear. YALE/DCS/RR-572, September 1987. (Revised edition of YALEU/DCS/RR-494 November 1986.).

[9] 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.

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

[11] 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.