EXPRESSING BOOLEAN CUBE MATRIX ALGORITHMS IN SHARED MEMORY PRIMITIVES

THINKING MACHINES CORP.
CAMBRIDGE, MA

1993
Matrix Multiplication on Boolean Cubes Using Shared Memory Primitives
C.T. Ho and S.L. Johnson, Yale University

Expressing Boolean Cube Matrix Algorithms in Shared Memory Primitives
Expressing boolean cube matrix algorithms in shared memory primitives

S.L. Johnsson and C. Ho

Thinking Machines Corp.
245 First Street.
Cambridge, MA 02142-1264

ONR -- The Pentagon
Washington DC 20350

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.
Expressing Boolean Cube Matrix Algorithms in Shared Memory Primitives

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

\textbf{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 \cite{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).

\section{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.

\textsuperscript{1}Also with Dept. of Electrical Engineering, Yale Univ., and Thinking Machines Corp., Cambridge, MA 02142.
Table 1: Various ways of assigning matrix elements into processors.

<table>
<thead>
<tr>
<th>Part.</th>
<th>Storage</th>
<th>Encoding</th>
<th>Processor address</th>
</tr>
</thead>
<tbody>
<tr>
<td>column</td>
<td>consec.</td>
<td>binary</td>
<td>(j_{q-1}j_{q-2} \ldots j_{q-n})</td>
</tr>
<tr>
<td></td>
<td>Gray</td>
<td>(G(j_{q-1}j_{q-2} \ldots j_{q-n}))</td>
<td></td>
</tr>
<tr>
<td>cyclic</td>
<td>binary</td>
<td>(j_{n-1}j_{n-2} \ldots j_0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gray</td>
<td>(G(j_{n-1}j_{n-2} \ldots j_0))</td>
<td></td>
</tr>
<tr>
<td>2-dim.</td>
<td>consec.</td>
<td>binary</td>
<td>(i_{p-1}i_{p-2} \ldots i_{p-n_2})</td>
</tr>
<tr>
<td></td>
<td>Gray</td>
<td>(G(i_{p-1}i_{p-2} \ldots i_{p-n_2}))</td>
<td></td>
</tr>
<tr>
<td>cyclic</td>
<td>binary</td>
<td>(i_{n_1-1}i_{n_1-2} \ldots i_0)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Gray</td>
<td>(G(i_{n_1-1}i_{n_1-2} \ldots i_0))</td>
<td></td>
</tr>
</tbody>
</table>

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 \(N_3^2\) block columns and matrix \(D\) is partitioned into \(N_3^3\) block rows. Each block column of \(C\) and each block row of \(D\) are further partitioned into \(N_1^2\) \(\times N_2^2\) blocks. The resulting matrix \(A\) can be partitioned into \(N_1^2\) \(\times N_2^2\) \(\times N_3^2\) blocks, or as \(N_1^3\) \(\times N_2^3\) 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 (nonminimum 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 \textit{all-to-all broadcasting} on a (sub)cube, \textit{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\(_2\) \( \times \) \( \ldots \) \( \times \) domain\(_n\)", where domain\(_i\) is the domain of the \( i \)th 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 \( \langle \) and \( \rangle \) form a conditional statement. For example,

\[
\langle \text{cond}\_1 \rightarrow \text{result}\_1, \\
\quad \text{cond}\_2 \rightarrow \text{result}\_2, \\
\quad \text{else} \rightarrow \text{result}\_3 \rangle,
\]

reads as "if \( \text{cond}\_1 \) then \( \text{result}\_1 \), else if \( \text{cond}\_2 \), then \( \text{result}\_2 \), else \( \text{result}\_3 \)". \( \forall [f(j)*g(j)]0 \leq j < x \) 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 \( i \)th row and \( j \)th 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. \( l_c, l_{d} \) and \( l_{a} \) are used to denote the local submatrices of \( C, D \), and \( A \), respectively.

We use \( \alpha \) and \( \alpha' \) to distinguish between binary encoding and Gray code encoding of the processor id \( (\text{pid}) \), i.e., \( \alpha = \text{pid} \) and \( \alpha' = G(\text{pid}) \), 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 \( t_r \), the time for the transmission of an element by \( t_e \), 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.
<table>
<thead>
<tr>
<th>Model</th>
<th>Algorithm</th>
<th>Element transfers</th>
<th>min start-ups</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-port</td>
<td>SBT</td>
<td>$(N-1)M$</td>
<td>$n$</td>
</tr>
<tr>
<td>n-port</td>
<td>nRSBT</td>
<td>$(N-1)M_n$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Algorithm</th>
<th>Element transfers</th>
<th>min start-ups</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-port</td>
<td>SBT</td>
<td>$\frac{(N-1)M}{N}$</td>
<td>$n$</td>
</tr>
<tr>
<td>n-port</td>
<td>nRSBT</td>
<td>$\frac{(N-1)M}{nN}$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Algorithm</th>
<th>Element transfers</th>
<th>min start-ups</th>
</tr>
</thead>
<tbody>
<tr>
<td>one-port</td>
<td>SBT</td>
<td>$\frac{nm}{2}$</td>
<td>$n$</td>
</tr>
<tr>
<td>n-port</td>
<td>nRSBT</td>
<td>$\frac{M}{2}$</td>
<td>$n$</td>
</tr>
</tbody>
</table>

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.brdf1(\( \alpha, i, j', t \)) over \([0 : N] \times [0 : P] \times [0 : 2^t \frac{Q}{N}] \times [0 : n] = \)
\( \ll t = 0 \rightarrow lc(\alpha, i, j'), \)
\( \text{else } \rightarrow \)
\( \ll 0 \leq j' < 2^{t-1} \frac{Q}{N} \rightarrow lc.brdf1(\alpha, i, j', t - 1), \)
/* Get from \((t - 1)\)th nbr and append. */
\( \text{else } \rightarrow lc.brdf1(\alpha \oplus 2^{t-1}, i, j' - 2^{t-1} \frac{Q}{N}, t - 1) \gg, \)
/* Order the \( N \) blocks by pid. */
lc.brdf(\( \alpha, i, j \)) over \([0 : N] \times [0 : P] \times [0 : Q] = lc.brdf1(\alpha, i, j \oplus \alpha \frac{Q}{N}, n) \)

For Gray code encoding, \( \alpha \) is replaced by \( \hat{\alpha} \) and \( lc.brdf \) is redefined as:

/* Order the \( N \) blocks by \( G(pid) \). */
lc.brdf(\( \hat{\alpha}, i, j \)) over \([0 : N] \times [0 : P] \times [0 : Q] = lc.brdf1(\hat{\alpha}, i, (G([\frac{iN}{Q}]) \oplus \hat{\alpha}) \frac{Q}{N} + j \bmod \frac{Q}{N}, n) \)

With \textit{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.brdf1(\( \alpha, u, i', j', t \)) over \([0 : N] \times [0 : n] \times [0 : \frac{P}{n}] \times [0 : 2^t \frac{Q}{N}] \times [0 : n] = \)
\( \ll t = 0 \rightarrow lc(\alpha, u \frac{P}{n} + i', j'), \)
\( \text{else } \rightarrow \)
\( \ll 0 \leq j' < 2^{t-1} \frac{Q}{N} \rightarrow lc.brdf1(\alpha, u, i', j', t - 1), \)
\( \text{else } \rightarrow lc.brdf1(\alpha \oplus 2^{(u+t-1) \bmod n}, u, i', j' - 2^{t-1} \frac{Q}{N}, t - 1) \gg, \)
lc.brdf(\( \alpha, i, j \)) over \([0 : N] \times [0 : P] \times [0 : Q] = \)
lc.brdf1(\( \alpha, [\frac{iP}{n}], i \bmod \frac{P}{n}, (sh([\frac{iP}{n}], [\frac{iN}{Q}]) \oplus \alpha) \frac{Q}{N} + j \bmod \frac{Q}{N}, n) \)

With \textit{one-port communication}, the code is described below. The code for \textit{n-port communication} is included in appendix A.
/* SBT transpose. */
/* Column partitioning, one-port, binary encoding. */
lc.txp1(a, i', j, t) over [0 : N) x [0 : P) x [0 : Q) x [0 : n] =
   \ll t = 0 \rightarrow lc(a, i', j),
   \lfloor \frac{n_t}{2} \rfloor \mod 2 = 0 \rightarrow
   \lfloor 0 \leq j < 2^{t-1} \frac{Q}{N} \rightarrow lc.txp1(a, i', j, t-1)
   \text{else } \rightarrow lc.txp1(a \oplus 2^{n-t}, i', j - 2^{t-1} \frac{Q}{N}, t-1) \rr,
   \text{else } \rightarrow
   \lfloor 0 \leq j < 2^{t-1} \frac{Q}{N} \rightarrow lc.txp1(a \oplus 2^{n-t}, i' + \frac{P}{N}, j, t-1)
   \text{else } \rightarrow lc.txp1(a, i' + \frac{P}{N}, j - 2^{t-1} \frac{Q}{N}, t-1) \rr,
   lc.txp(a, i', j) over [0 : N) x [0 : P) x [0 : Q) = lc.txp1(a, 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. */
lr.red1(a, i, k', t) over [0 : N) x [0 : P) x [0 : R) x [0 : n] =
   \ll t = 0 \rightarrow lr(a, i, k'),
   \lfloor \frac{n_t}{2} \rfloor \mod 2 = 0 \rightarrow lr.red1(a, i, k', t-1) + lr.red1(a \oplus 2^{n-t}, i, k', t-1),
   \text{else } \rightarrow lr.red1(a, i, k' + \frac{R}{N}, t-1) + lr.red1(a \oplus 2^{n-t}, i, k' + \frac{R}{N}, t-1) \rr,
   lr.red(a, i, k') over [0 : N) x [0 : P) x [0 : R) = lr.red1(a, 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 x Q matrix partitioned into N1 x N2 blocks, implies that the processor that holds block (i, j), 0 \leq i < N1, 0 \leq j < N2, 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}{N1} x \frac{Q}{N2} to \frac{P}{N2} x \frac{Q}{N1} (instead of changing to a \frac{Q}{N1} x \frac{P}{N2} 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(N1, N2) x min(N1, N2) blocks. In the second phase, there are 2^{n_2-n_1} 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 N1 x N2 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 → 2^{2 \min(n_1,n_2) - t} \gg,
\lcp(\alpha_1, \alpha_2, i', j', t) \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : f(t) N_1] \times [0 : \frac{Q}{N_2 f(j)}] \times [0 : n] =
\iff t = 0 \rightarrow \lcp(\alpha_1, \alpha_2, i', j'),
\iff t \leq 2 \min(n_1, n_2) →
\text{/* two-dimensional transpose. */}
\iff \frac{\alpha}{2^{n_1-t}} \text{ mod } 2 = \frac{\alpha}{2^{n_2-t}} \text{ mod } 2 →
\iff t \text{ mod } 2 = 1 \rightarrow \lcp(\alpha_1 \oplus 2^{n_1-t}, \alpha_2, i', j', t-1) \gg,
\text{else} →
\iff t \text{ mod } 2 = 0 \rightarrow \lcp(\alpha_1, \alpha_2 \oplus 2^{n_2-t}, i', j', t-1) \gg \gg,
\text{/* one-dimensional transpose. */}
\iff n_1 < n_2 →
\iff \frac{n_1}{2^{n_1-t}} \text{ mod } 2 = 0 →
\iff 0 \leq j' < 2^{t'-1} \frac{Q}{N_2} \rightarrow \lcp(\alpha_1, \alpha_2, i', j', t-1),
\text{else} → \lcp(\alpha_1, \alpha_2 \oplus 2^{n-t}, i', j'-2^{t'-1} \frac{Q}{N_2}, t-1) \gg,
\text{else} →
\iff 0 \leq j' < 2^{t'-1} \frac{Q}{N_2} \rightarrow \lcp(\alpha_1, \alpha_2 \oplus 2^{n-t}, i' + \frac{P}{2^{n_1+t}}, j', t-1),
\text{else} → \lcp(\alpha_1, \alpha_2, i' + \frac{P}{2^{n_1+t}}, j'-2^{t'-1} \frac{Q}{N_2}, t-1) \gg \gg,
\text{else} →
\iff \frac{n_2}{2^{n_1-t}} \text{ mod } 2 = 0 →
\iff 0 \leq i' < 2^{t'-1} \frac{P}{N_1} \rightarrow \lcp(\alpha_1, \alpha_2, i', j', t-1),
\text{else} → \lcp(\alpha_1 \oplus 2^{n-t}, \alpha_2, i' - 2^{t'-1} \frac{P}{N_1}, j', t-1) \gg,
\text{else} →
\iff 0 \leq i' < 2^{t'-1} \frac{P}{N_1} \rightarrow \lcp(\alpha_1 \oplus 2^{n-t}, \alpha_2, i', j' + \frac{Q}{2^{n_2+t}}, t-1),
\text{else} → \lcp(\alpha_1, \alpha_2, i' - 2^{t'-1} \frac{P}{N_1}, j' + \frac{Q}{2^{n_2+t}}, t-1) \gg \gg
\text{where } t' = t - 2 \min(n_1, n_2),
\lcp(\alpha_1, \alpha_2, i', j') \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_1}] \times [0 : \frac{Q}{N_1}] = \lcp(\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 \(A(-1,1,1)\). 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:
\[ A(\cdot, 1, 1) : \quad C_{\alpha}, D_{\alpha} \xrightarrow{\text{brd. } C_i} C_{\alpha}, D_{\alpha} \xrightarrow{\text{mpy. } [R]} A_{\alpha} \]
\[ A(\cdot, 1, 3) : \quad C_{\alpha}, D_{\alpha} \xrightarrow{\text{txp. } C_i} C_{\alpha}, D_{\alpha} \xrightarrow{\text{brd. } D_i} C_{\alpha}, D_{\alpha} \xrightarrow{\text{mpy. } [P]} A_{\alpha} \xrightarrow{\text{txp. } A_i} A_{\alpha} \]
\[ A(\cdot, 1, 4) : \quad C_{\alpha}, D_{\alpha} \xrightarrow{\text{txp. } D_i} C_{\alpha}, D_{\alpha} \xrightarrow{\text{mpy. } [Q]} A_{\alpha} \xrightarrow{\text{red. } A_i} A_{\alpha} \]

Row Partitioning:
\[ A(\cdot, 1, 1) : \quad C_{\alpha}, D_{\alpha} \xrightarrow{\text{brd. } D_i} C_{\alpha}, D_{\alpha} \xrightarrow{\text{mpy. } [P]} A_{\alpha} \]
\[ A(\cdot, 1, 3) : \quad C_{\alpha}, D_{\alpha} \xrightarrow{\text{txp. } D_i} C_{\alpha}, D_{\alpha} \xrightarrow{\text{brd. } C_i} C_{\alpha}, D_{\alpha} \xrightarrow{\text{mpy. } [R]} A_{\alpha} \xrightarrow{\text{txp. } A_i} A_{\alpha} \]
\[ A(\cdot, 1, 4) : \quad C_{\alpha}, D_{\alpha} \xrightarrow{\text{txp. } C_i} C_{\alpha}, D_{\alpha} \xrightarrow{\text{mpy. } [Q]} A_{\alpha} \xrightarrow{\text{red. } A_i} A_{\alpha} \]

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

computes \( CD(\cdot, [\frac{j}{N}]) \) for all \( j \) mapped to \( \alpha \), where \( \text{PID} \) is the allocation function as described in Table 1.

- Algorithm \( A(\cdot, 1, 2) \). 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 \( \alpha = \text{PID}(j) \) computes \( CD(\cdot, [\frac{j}{N}]) \) for all \( j \) mapped to \( \alpha \).

- Algorithm \( A(\cdot, 1, 3) \). 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 \( \alpha = \text{PID}(j) \) computes \( C([\frac{j}{N}], \cdot) D \).

- Algorithm \( A(\cdot, 1, 4) \). Compute \( A \) in-space by a transpose of \( D \), and reduction of partial inner products of \( A \).

The algorithms are identified by \( A(\text{number of ports used concurrently, number of loops parallelized, algorithm identifier}) \). Algorithm \( A(\cdot, 1, 2) \) is clearly inferior to algorithm \( A(\cdot, 1, 1) \) 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 (e.g. \( [R] \) in \( A(\cdot, 1, 1) \)) 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(\(\alpha, i, j', t\)) over \([0 : N] \times [0 : P] \times [0 : \frac{N}{P}] \times [0 : N] =
\[ t = 0 \rightarrow c(i, \alpha N) + j', \]
else \( \rightarrow l(a((\alpha + 1) \bmod N) i, j', t - 1) \) \]
\[ ld(\alpha, j', k') \text{ over } [0 : N] \times [0 : Q] \times [0 : F^k] = \text{d}(j, \alpha N) + k'), \]
\[ la(\alpha, i, k', t) \text{ over } [0 : N] \times [0 : P] \times [0 : F^k] \times [0 : N] = \]
\[ \ll t = 0 \rightarrow 0, \]
else \( \rightarrow la(\alpha, i, k', t - 1) + (\ll [l(\alpha, i, j', t - 1)
* ld(\alpha, ((\alpha + t - 1) \bmod N) F^k + j', k') 0 \leq j' < F^k]) \) \]
\[ a(i, k) \text{ over } [0 : P] \times [0 : R] = la([\frac{kN}{N}], i, k \bmod \frac{F}{N}, N) \]

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 \( \alpha \) and \((\alpha + 1) \bmod N\) are in adjacent processors. Since \( \alpha \) is the Gray code encoding of the processor \( id \), i.e., the \( j \)th block column is stored in processor \( pid \) with \( \alpha = G(pid) = j \), rotation of \( C \) implies nearest-neighbor communications. For binary encoding, i.e., the \( j \)th block column is stored in processor \( \alpha = j \), we redefine \( lc \) and \( la \) as follows:

\[ G(t) = t \oplus [\frac{t}{2}], \]
\[ G^{-1}(t) = \ll t = 0 \rightarrow 0, \]
else \( \rightarrow t \oplus G^{-1}(\frac{t}{2}) \) \]
\[ lc(\alpha, i, j', t) \text{ over } [0 : N] \times [0 : P] \times [0 : F^k] \times [0 : N] = \]
\[ \ll t = 0 \rightarrow c(i, \alpha N) + j', \]
else \( \rightarrow lc(G^{-1}((\alpha + 1) \bmod N), i, j', t - 1) \) \]
\[ la(\alpha, i, k', t) \text{ over } [0 : N] \times [0 : P] \times [0 : F^k] \times [0 : N] = \]
\[ \ll t = 0 \rightarrow 0, \]
else \( \rightarrow la(\alpha, i, k', t - 1) + (\ll [l(\alpha, i, j', t - 1)
* ld(\alpha, G((G^{-1}(\alpha) + t - 1) \bmod N) F^k + j', k') 0 \leq j' < F^k]) \) \]

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

\[ T(t) = \ll t \bmod 2 = 0 \rightarrow 0, \]
\[ \rightarrow 1 + T(\frac{t}{2}) \) \]
\[ lc(\alpha, i, j', t) \text{ over } [0 : N] \times [0 : P] \times [0 : F^k] \times [0 : N] = \]
\[ \ll t = 0 \rightarrow c(i, \alpha N) + j', \]

9
else \rightarrow \text{lc}(\alpha \oplus \beta^{2^T(t-1)}, i, j', t-1) \gg, \\
\text{ld}(\alpha, j, k') \text{ over } [0 : N] \times [0 : Q] \times \left[0 : \frac{Q}{N}\right] = d(j, \alpha \frac{Q}{N} + k'), \\
\text{la}(\alpha, i, k', t) \text{ over } [0 : N] \times [0 : P] \times \left[0 : \frac{Q}{N}\right] \times [0 : N] = \\
\ll t = 0 \rightarrow 0, \\
\text{else } \rightarrow \text{la}(\alpha, i, k', t-1) + (\ll [\text{lc}(\alpha, i, j', t-1) \\
\ast \text{ld}(\alpha, (\alpha \oplus G(t-1)) \frac{Q}{N} + j', k') | 0 \leq j' < \frac{Q}{N}]) \gg, \\
\text{a}(i, k) \text{ over } [0 : P] \times [0 : R] = \text{la}(\left[\frac{kN}{R}\right], i, k \mod \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 $(T(t) + u) \mod n$ during the same step.

/* A Gray code Exchange alg. A(n,1,1). */
/* Column partitioning, binary code encoding. */
\text{lc}(\alpha, u, i', j', t) \text{ 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'), \\
\text{else } \rightarrow \text{lc}(\alpha \oplus \beta^{2^{(T(t-1)+u)} \mod n}, u, i', j', t-1) \gg, \\
\text{ld}(\alpha, j, k') \text{ over } [0 : N] \times [0 : Q] \times \left[0 : \frac{Q}{N}\right] = d(j, \alpha \frac{Q}{N} + k'), \\
/* Shuffle (cyclic left-shift) $u$ steps of $t$. */
\text{sh}(u, t) \text{ over } [0 : n] \times [0 : N] = (t \mod 2^{n-u})2^u + \left[\frac{i}{2^{n-u}}\right], \\
\text{la}(\alpha, u, i', k', t) \text{ over } [0 : N] \times [0 : n] \times [0 : \frac{P}{n}] \times [0 : \frac{Q}{n}] \times [0 : N] = \\
\ll t = 0 \rightarrow 0, \\
\text{else } \rightarrow \text{la}(\alpha, u, i', k', t-1) + (\ll [\text{lc}(\alpha, u, i', j', t-1) \\
\ast \text{ld}(\alpha, (\alpha \oplus \text{sh}(u, G(t-1))) \frac{Q}{N} + j', k') | 0 \leq j' < \frac{Q}{N}]) \gg, \\
\text{a}(i, k) \text{ over } [0 : P] \times [0 : R] = \text{la}(\left[\frac{kN}{R}\right], i, k \mod \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(a, i, j) over [0 : N) x [0 : P) x [0 : Q) = c(i, a + j),
ld(a, j, k') over [0 : N) x [0 : Q) x [0 : R) = d(j, a + k'),
/* Final location of matrix A. */
a(i, k) over [0 : P) x [0 : R) = la(1 + k), i, k mod R)

/* A Doubling Algorithm A(-1,1): */
lc.br(a, i, j) over [0 : N) x [0 : P) x [0 : Q) = lc(1 + i, i, j mod Q),
la(a, i, k') over [0 : N) x [0 : P) x [0 : R) = \{ i, i + j, 0 \leq j < Q \}

/* Algorithm A(-1,3): */
lc.xp(a, i', j') over [0 : N) x [0 : P) x [0 : Q) = lc(i', i, j mod Q),
ld.br(a, j, k) over [0 : N) x [0 : Q) x [0 : R) = ld(j, j, k mod R),
la.xp(a, i', k') over [0 : N) x [0 : P) x [0 : R') = \{ i, i + j, 0 \leq j < Q \},
la(a, i, k') over [0 : N) x [0 : P) x [0 : R') = la.xp(i', i, j') mod R', a + k')

/* Algorithm A(-1,4): */
lc.xp(a, j', k) over [0 : N) x [0 : P) x [0 : R) = lc(i, i, j') mod R),
la(a, i, k) over [0 : N) x [0 : P) x [0 : R) = \{ la(i, i, j') * ld.xp(a, j', k) \}, 0 \leq j' < Q \}
la.red(a, i, k') over [0 : N) x [0 : P) x [0 : R') = \{ la(a', i, a + k') \}, 0 \leq a' < N \}

Table 5 shows the total number of arithmetic operations in sequence. If P, Q, and R 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 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 broadcasts 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 A(-1,1), A(-1,3) and A(-1,4) and assuming P, Q, R \geq N. Note that for R = \frac{P}{N} = \frac{Q}{N}, the communication complexity of A(-1,1) is less than that of A(-1,3), which in turn is less than that of A(-1,4). 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 A(-1,1) that computes A in-place by broadcasting C in its two-dimensional
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of arithmetic operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A(\cdot, 1, 1)$</td>
<td>$2PQ\lceil \frac{Q}{N} \rceil$</td>
</tr>
<tr>
<td>$A(\cdot, 1, 3)$</td>
<td>$2QR\lceil \frac{R}{N} \rceil$</td>
</tr>
<tr>
<td>$A(\cdot, 1, 4)$</td>
<td>$PR(2\lceil \frac{R}{N} \rceil - 1) + P(\lceil \frac{R}{N} \rceil + \sum_{i=1}^{n}\lceil \frac{i}{N} \rceil)$</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Element transfers</th>
<th>min start-ups</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A^e(1,1,1)$</td>
<td>$(N - 1)P\lceil \frac{Q}{N} \rceil$</td>
<td>$N - 1$</td>
</tr>
<tr>
<td>$A^e(1,1,3)$</td>
<td>$(N - 1)Q\lceil \frac{Q}{N} \rceil + \frac{N}{2} \lceil \frac{Q}{N} \rceil + \frac{N}{2} \lceil \frac{R}{N} \rceil$</td>
<td>$3n$</td>
</tr>
<tr>
<td>$A^e(1,1,4)$</td>
<td>$(N - 1)P\lceil \frac{Q}{N} \rceil + \frac{N}{2} \lceil \frac{Q}{N} \rceil + \frac{N}{2} \lceil \frac{R}{N} \rceil$</td>
<td>$2n$</td>
</tr>
<tr>
<td>$A^e(n,1,1)$</td>
<td>$\frac{1}{n}(N - 1)P\lceil \frac{Q}{N} \rceil$</td>
<td>$N - 1$</td>
</tr>
<tr>
<td>$A^e(n,1,3)$</td>
<td>$\frac{1}{n}(N - 1)Q\lceil \frac{Q}{N} \rceil + \frac{N}{2} \lceil \frac{Q}{N} \rceil + \frac{N}{2} \lceil \frac{R}{N} \rceil$</td>
<td>$3n$</td>
</tr>
<tr>
<td>$A^e(n,1,4)$</td>
<td>$\frac{1}{n}(N - 1)P\lceil \frac{Q}{N} \rceil + \frac{N}{2} \lceil \frac{Q}{N} \rceil + \frac{N}{2} \lceil \frac{R}{N} \rceil$</td>
<td>$2n$</td>
</tr>
</tbody>
</table>

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

12
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 \( (A(-,1,4) \) has two variations) are

- **Algorithm \( A(-,2,1) \).** 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 \( \alpha_1, \alpha_2 \) then computes \( C([\frac{i}{N_1}], \cdot)D(\cdot, [\frac{j}{N_2}]) \) for all \( i \) mapped to \( \alpha_1 \) and all \( j \) mapped to \( \alpha_2 \). The communication operations are broadcasting from multiple sources within rows and columns.

  **Algorithm \( A(-,2,2) \).** 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 \( \alpha_1, \alpha_2 \) receives \( C(\cdot, [\frac{i}{N_1}]) \) during the broadcasting operation, then computes the product \( C(\cdot, [\frac{i}{N_1}])D([\frac{i}{N_1}], [\frac{j}{N_2}]) \). The summation over index \( i \) is the reduction operation along columns.

- **Algorithm \( A(-,2,3) \).** Transpose \( C \), perform multiple source broadcasting of the elements of \( D \) within processor columns, 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 \( \frac{P}{N_2} \) 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 \( \alpha_1, \alpha_2 \) computes \( C([\frac{i}{N_1}], [\frac{j}{N_2}])D([\frac{i}{N_1}], \cdot) \) for all \( i, j \) such that \( [\frac{i}{N_1}] = \alpha_2 \) and \( [\frac{j}{N_2}] = \alpha_1 \).

- **Algorithm \( A(-,2,4) \).** 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 \( \frac{P}{N_2} \) columns are accumulated within a processor column. After the transposition and broadcasting processor \( \alpha_1, \alpha_2 \) has the elements \( C([\frac{i}{N_1}], [\frac{j}{N_2}])D([\frac{i}{N_1}], \cdot) \) for all \( i, j \) such that \( [\frac{i}{N_1}] = \alpha_1 \) and \( j \) such that \( [\frac{j}{N_2}] = \alpha_2 \).

  **Algorithm \( A(-,2,5) \).** 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 \( A^T \).
\[ \mathcal{A}(\cdot, 2, 1) : \quad C_{\alpha_1 \alpha_2}, D_{\alpha_1 \alpha_2} \xrightarrow{\text{brd. } C, \text{ un}} C_{\alpha_1 \alpha_2}, D_{\alpha_1 \alpha_2} \xrightarrow{\text{brd. } D, \text{ un}} C_{\alpha_1 \alpha_2}, D_{\alpha_1 \alpha_2} \]
\[ \xrightarrow{\text{mpv., } [PR]} \quad A_{\alpha_1 \alpha_2} \]

\[ \mathcal{A}(\cdot, 2, 2) : \quad C_{\alpha_1 \alpha_2}, D_{\alpha_1 \alpha_2} \xrightarrow{\text{txp. } C, \text{ un}} C_{\alpha_2 \alpha_1}, D_{\alpha_1 \alpha_2} \xrightarrow{\text{brd. } C, \text{ un}} C_{\alpha_1 \alpha_2}, D_{\alpha_1 \alpha_2} \]
\[ \xrightarrow{\text{mpv., } [QR]} \quad A_{\alpha_1 \alpha_2} \xrightarrow{\text{red. } A, \text{ un}} A_{\alpha_1 \alpha_2} \]

\[ \mathcal{A}(\cdot, 2, 3) : \quad C_{\alpha_1 \alpha_2}, D_{\alpha_1 \alpha_2} \xrightarrow{\text{txp. } C, \text{ un}} C_{\alpha_2 \alpha_1}, D_{\alpha_1 \alpha_2} \xrightarrow{\text{brd. } D, \text{ un}} C_{\alpha_2 \alpha_1}, D_{\alpha_1 \alpha_2} \]
\[ \xrightarrow{\text{mpv., } [PQ]} \quad A_{\alpha_1 \alpha_2} \xrightarrow{\text{red. } A, \text{ un}} A_{\alpha_2 \alpha_1} \xrightarrow{\text{txp. } A, \text{ un}} A_{\alpha_1 \alpha_2} \]

\[ \mathcal{A}(\cdot, 2, 4) : \quad C_{\alpha_1 \alpha_2}, D_{\alpha_1 \alpha_2} \xrightarrow{\text{txp. } D, \text{ un}} C_{\alpha_1 \alpha_2}, D_{\alpha_2 \alpha_1} \xrightarrow{\text{brd. } D, \text{ un}} C_{\alpha_1 \alpha_2}, D_{\alpha_2 \alpha_1} \]
\[ \xrightarrow{\text{mpv., } [PQ]} \quad A_{\alpha_1 \alpha_2} \xrightarrow{\text{red. } A, \text{ un}} A_{\alpha_2 \alpha_1} \xrightarrow{\text{txp. } A, \text{ un}} A_{\alpha_1 \alpha_2} \]

\[ \mathcal{A}(\cdot, 2, 5) : \quad C_{\alpha_1 \alpha_2}, D_{\alpha_1 \alpha_2} \xrightarrow{\text{txp. } D, \text{ un}} C_{\alpha_1 \alpha_2}, D_{\alpha_2 \alpha_1} \xrightarrow{\text{brd. } C, \text{ un}} C_{\alpha_1 \alpha_2}, D_{\alpha_2 \alpha_1} \]
\[ \xrightarrow{\text{mpv., } [QR]} \quad A_{\alpha_1 \alpha_2} \xrightarrow{\text{red. } A, \text{ un}} A_{\alpha_2 \alpha_1} \xrightarrow{\text{txp. } A, \text{ un}} A_{\alpha_1 \alpha_2} \]

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

for each of \( \frac{R}{N_1} \) columns of \( D \). Processor \( \alpha_1, \alpha_2 \) computes \( C(\cdot, \lfloor \frac{i}{N_2} \rfloor) D(\lfloor \frac{i}{N_1} \rfloor, \lfloor \frac{j}{N_1} \rfloor) \)
for all \( i \) such that \( \lfloor \frac{i}{N_1} \rfloor = \alpha_1 \) and \( j \) such that \( \lfloor \frac{j}{N_2} \rfloor = \alpha_2 \).

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 \( [0 : \sqrt{N}] \times [0 : \sqrt{N}] \times [0 : \frac{P}{\sqrt{N}}] \times [0 : \frac{Q}{\sqrt{N}}] \times [0 : \sqrt{N}] = \)
\[ \ll t = 0 \rightarrow c(\hat{\alpha}_1 \frac{P}{\sqrt{N}} + i', \hat{\alpha}_2 \frac{Q}{\sqrt{N}} + j'), \]
\[ \text{else} \rightarrow lc(\hat{\alpha}_1, (\hat{\alpha}_2 + 1) \mod \sqrt{N}, i', j', t - 1) \gg, \]
ld(\( \hat{\alpha}_1, \hat{\alpha}_2, j', k', t \)) over \( [0 : \sqrt{N}] \times [0 : \sqrt{N}] \times [0 : \frac{Q}{\sqrt{N}}] \times [0 : \frac{P}{\sqrt{N}}] \times [0 : \sqrt{N}] = \)
\[ \ll t = 0 \rightarrow d(\hat{\alpha}_1 \frac{Q}{\sqrt{N}} + j', \hat{\alpha}_2 \frac{P}{\sqrt{N}} + k'), \]

14
else \rightarrow \text{l}d((\hat{\alpha}_1 + 1) \mod \sqrt{N}, \hat{\alpha}_2, j', k', t - 1) \Rightarrow,
\text{l}a(\hat{\alpha}_1, \hat{\alpha}_2, i', k', t) \ \text{over} \ [0 : \sqrt{N}] \times [0 : \sqrt{N}] \times [0 : \frac{P}{N}] \times [0 : \frac{R}{N}] =
\begin{align*}
\leq t = 0 \\
\text{else} \rightarrow \text{l}a(\hat{\alpha}_1, \hat{\alpha}_2, i', k', t - 1) + (\downarrow [\text{l}c(\hat{\alpha}_1, \hat{\alpha}_2, i, j', t - 1) \\
\star \text{l}d(\hat{\alpha}_1, \hat{\alpha}_2, j', k', t - 1)]0 \leq j' < \frac{R}{N}) \Rightarrow,
\end{align*}
a(i, k) \ \text{over} \ [0 : P] \times [0 : R] = \text{la}_\text{red}(\frac{P}{N}, \frac{R}{N}, i \mod \frac{P}{N}, k \mod \frac{R}{N})

It is also possible to design a matrix multiplication algorithm based on the SBT, or the nRSBT communication algorithms. For Algorithm $A(\cdot, 2, 1)$, 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 $A(\cdot, 2, 2)$ and $A(\cdot, 2, 4)$, $\text{l}a$ is replaced by $\text{la}_\text{red}$.

/* Initial allocations of $C$ and $D$. */
\text{l}c(\alpha_1, \alpha_2, i', j') \ \text{over} \ [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N}] \times [0 : \frac{Q}{N}] = c(\alpha_1 \frac{P}{N} + i', \alpha_2 \frac{Q}{N} + j'),
\text{l}d(\alpha_1, \alpha_2, j', k') \ \text{over} \ [0 : N_1] \times [0 : N_2] \times [0 : \frac{Q}{N}] \times [0 : \frac{P}{N}] = d(\alpha_1 \frac{Q}{N} + j', \alpha_2 \frac{P}{N} + k'),
/* Final allocation of $A$. */
a(i, k) \ \text{over} \ [0 : P] \times [0 : R] = \text{la}(\frac{N_1}{N}, \frac{N_2}{R}, i \mod \frac{P}{N}, k \mod \frac{R}{N})

/* A Doubling Algorithm $A(\cdot, 2, 1)$: */
\text{l}_\text{row}(\alpha_1, \alpha_2, i', j) \ \text{over} \ [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N}] \times [0 : Q] = \text{l}c(\alpha_1, \frac{N_2}{N}, i', j \mod \frac{Q}{N}),
\text{l}_\text{col}(\alpha_1, \alpha_2, j, k') \ \text{over} \ [0 : N_1] \times [0 : N_2] \times [0 : Q] \times [0 : \frac{P}{N}] = \text{l}d(\frac{N_1}{Q}, \alpha_2, j \mod \frac{P}{N}, k'),
\text{l}a(\alpha_1, \alpha_2, i', k') \ \text{over} \ [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N}] \times [0 : \frac{Q}{N}] =
\begin{align*}
\downarrow [\text{l}_\text{row}(\alpha_1, \alpha_2, i', j) \star \text{l}_\text{col}(\alpha_1, \alpha_2, j, k')]0 \leq j < Q
\end{align*}

/* Algorithm $A(\cdot, 2, 2)$: */
\text{l}_\text{txp}(\alpha_1, \alpha_2, i', j') \ \text{over} \ [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N}] \times [0 : \frac{Q}{N}] =
\text{l}c((\alpha_1 \frac{P}{N} + i')/\frac{P}{N} \frac{Q}{N}, (\alpha_2 \frac{Q}{N} + j')/\frac{Q}{N}, (\alpha_1 \frac{P}{N} + i') \mod \frac{P}{N}, (\alpha_2 \frac{Q}{N} + j') \mod \frac{Q}{N}),
\text{l}_\text{txp}_\text{row}(\alpha_1, \alpha_2, i, j') \ \text{over} \ [0 : N_1] \times [0 : N_2] \times [0 : P] \times [0 : \frac{Q}{N}] =
\text{l}c(\frac{P}{N}, \frac{Q}{N}, i \mod \frac{P}{N}, j'),
\text{l}_\text{txp}(\alpha_1, \alpha_2, i, k') \ \text{over} \ [0 : N_1] \times [0 : N_2] \times [0 : P] \times [0 : \frac{Q}{N}] =
\begin{align*}
\downarrow [\text{l}_\text{txp}_\text{row}(\alpha_1, \alpha_2, i, j') \star \text{l}d(\alpha_1, \alpha_2, j', k')]0 \leq j' < \frac{Q}{N},
\end{align*}
\text{l}_\text{red}(\alpha_1, \alpha_2, i', k') \ \text{over} \ [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N}] \times [0 : \frac{Q}{N}] =
\begin{align*}
\downarrow [\text{l}_\text{txp}_\text{row}(\alpha_1, \alpha_2, i', k')0 \leq \alpha'_i < N_1
\end{align*}
Table 7: The communication complexity for optimum buffer sizes, two-dimensional partitioning, and one-port communication.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Number of arithmetic operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A^c(\cdot,2,1)$</td>
<td>$2Q \left( \frac{P}{N_1} \right) \left( \frac{P}{N_2} \right)$</td>
</tr>
<tr>
<td>$A^c(\cdot,2,2)$</td>
<td>$(2 \left( \frac{P}{N_1} \right) - 1) \left( \frac{P}{N_1} \right) P + \sum_{i=1}^{N_1} \left( \frac{P}{N_1} \right) + \left( \frac{P}{N_2} \right)$</td>
</tr>
<tr>
<td>$A^c(\cdot,2,3)$</td>
<td>$(2 \left( \frac{P}{N_1} \right) - 1) \left( \frac{P}{N_2} \right) R + \sum_{i=1}^{N_1} \left( \frac{P}{N_2} \right) + \left( \frac{R}{N_1} \right)$</td>
</tr>
<tr>
<td>$A^c(\cdot,2,4)$</td>
<td>$(2 \left( \frac{P}{N_1} \right) - 1) \left( \frac{R}{N_1} \right) R + \sum_{i=1}^{N_1} \left( \frac{R}{N_2} \right) + \left( \frac{R}{N_1} \right)$</td>
</tr>
<tr>
<td>$A^c(\cdot,2,5)$</td>
<td>$(2 \left( \frac{P}{N_1} \right) - 1) \left( \frac{R}{N_2} \right) P + \sum_{i=1}^{N_1} \left( \frac{R}{N_2} \right) + \left( \frac{R}{N_1} \right)$</td>
</tr>
</tbody>
</table>

/* Algorithm $A(\cdot,2,3)$: */

$lc.tzp(a_1, a_2, i', j')$ over $[0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_1}] \times [0 : \frac{Q}{N_2}] =
$\hspace{1cm}$lc([(a_1 \frac{P}{N_1} + i')/\frac{P}{N_1}], [(a_2 \frac{Q}{N_2} + j')/\frac{Q}{N_2}], (a_1 \frac{P}{N_1} + i') \mod \frac{P}{N_1}, (a_2 \frac{Q}{N_2} + j') \mod \frac{Q}{N_2})$,

$ld.row(a_1, a_2, j', k)$ over $[0 : N_1] \times [0 : N_2] \times [0 : \frac{Q}{N_1}] \times [0 : R] =
$\hspace{1cm}$ld(a_1, [\frac{N_2}{N_1}], j', k \mod \frac{R}{N_1})$,

$la.tzp(a_1, a_2, i', k)$ over $[0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_1}] \times [0 : R] =
$\hspace{1cm}$\{ [lc.tzp(a_1, a_2, i', j') \times ld.row(a_1, a_2, j', k) \mid 0 \leq j < \frac{Q}{N_1} \}$,

$la.tzp.red(a_1, a_2, i', k')$ over $[0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_1}] \times [0 : \frac{R}{N_2}] =
$\hspace{1cm}$\{ [la.tzp(a_1', a_2, i', \frac{R}{N_2} + k') \mid 0 \leq a_1' < N_1] \}$,

$la(a_1, a_2, i', k')$ over $[0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_1}] \times [0 : \frac{R}{N_2}] =
$\hspace{1cm}$la.tzp.red([(a_1 \frac{P}{N_1} + i')/\frac{P}{N_1}], [a_2 \frac{R}{N_2} + k'/\frac{R}{N_2}], (a_1 \frac{P}{N_1} + i') \mod \frac{R}{N_1}, (a_2 \frac{R}{N_2} + k') \mod \frac{R}{N_1})$.

Algorithms $A(\cdot,2,4)$ and $A(\cdot,2,5)$ 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 $Q : P : R : P$ from $A(1,2,1)$ to $A(1,2,5)$. It can be shown [7] that for $P$, $Q$ and $R$ being multiples of $N_1$ and $N_2$, the complexities of algorithms $A(\cdot,2,3)$ and $A(\cdot,2,5)$ are always higher than that of $\min(A(\cdot,2,2), A(\cdot,2,4))$, 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].
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Element transfers</th>
<th>min start-ups</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{A}^c(1,2,1)$</td>
<td>$(N_2 - 1)\left[\frac{F}{N_2}\right]n + \left[\frac{F}{N_2}\right](N_2 - 1)$</td>
<td>$n$</td>
</tr>
<tr>
<td>$+ (N_1 - 1)\left[\frac{P}{N_1}\right]n$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,2)$</td>
<td>$\left[\frac{F}{N_1}\right]n + \left[\frac{F}{N_2}\right](N_2 - 1)$</td>
<td>$2n$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,3)$</td>
<td>$\left[\frac{F}{N_1}\right]n + \left[\frac{F}{N_2}\right](N_2 - 1)$</td>
<td>$3n$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,4)$</td>
<td>$\left[\frac{F}{N_1}\right]n + \left[\frac{F}{N_2}\right](N_2 - 1)$</td>
<td>$2n$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,5)$</td>
<td>$\left[\frac{F}{N_1}\right]n + \left[\frac{F}{N_2}\right](N_2 - 1)$</td>
<td>$3n$</td>
</tr>
</tbody>
</table>

Table 8: The communication complexity using two-dimensional partitioning.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$N_1$</th>
<th>$N_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{A}^c(1,2,1)$</td>
<td>$\sqrt{PN}$</td>
<td>$\sqrt{RN}$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,2)$</td>
<td>$\sqrt{QN}$</td>
<td>$\sqrt{RN}$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,3)$</td>
<td>$\sqrt{QN}$</td>
<td>$\sqrt{RN}$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,4)$</td>
<td>$\sqrt{PN}$</td>
<td>$\sqrt{QN}$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,5)$</td>
<td>$\sqrt{RN}$</td>
<td>$\sqrt{QN}$</td>
</tr>
</tbody>
</table>

Table 9: The optimum values of $N_1$ and $N_2$ for $P$, $Q$ and $R$ being multiples of $N$ and one-port communication.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>$T_{\text{min}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{A}^c(1,2,1)$</td>
<td>$\frac{2PQR}{N}t_a + \frac{Q}{\sqrt{N}}(2\sqrt{PR} - \frac{P+R}{\sqrt{N}})t_c + n\tau$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,2)$</td>
<td>$\frac{2PQR}{N}t_a + \frac{P}{\sqrt{N}}(2\sqrt{QR} + \frac{RQ-(Q+R)}{\sqrt{N}})t_c + 2n\tau$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,3)$</td>
<td>$\frac{2PQR}{N}t_a + \frac{R}{\sqrt{N}}(2\sqrt{PQ} + \frac{nP(1+x)}{\sqrt{N}} + \frac{n(P+Q)}{\sqrt{N}})t_c + 3n\tau$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,4)$</td>
<td>$\frac{2PQR}{N}t_a + \frac{R}{\sqrt{N}}(2\sqrt{PQ} + \frac{RQ-(Q+R)}{\sqrt{N}})t_c + 2n\tau$</td>
</tr>
<tr>
<td>$\mathcal{A}^c(1,2,5)$</td>
<td>$\frac{2PQR}{N}t_a + \frac{P}{\sqrt{N}}(2\sqrt{QR} + \frac{nR(1+x)-(Q+R)}{\sqrt{N}})t_c + 3n\tau$</td>
</tr>
</tbody>
</table>

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.
<table>
<thead>
<tr>
<th>Algorithm</th>
<th>min communication time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_c(n,2,1)$</td>
<td>$\max\left(\frac{N_2-1}{n_2} [\frac{P}{N_1}][\frac{Q}{N_2}] t_c + n_2 \tau, \frac{N_2-1}{n_2} [\frac{R}{N_2}][\frac{P}{N_1}] t_c + n_2 \tau\right)$</td>
</tr>
<tr>
<td>$A_c(n,2,2)$</td>
<td>$n\tau + (\sqrt{\frac{P}{N_1}}[\frac{Q}{N_2}] t_c + \sqrt{(n-1)\tau})^2 + (\sqrt{\frac{P}{N_1}}[\frac{R}{N_2}] t_c + \sqrt{(n-1)\tau})^2$</td>
</tr>
<tr>
<td>$A_c(n,2,3)$</td>
<td>$n\tau + (\sqrt{\frac{P}{N_1}}[\frac{Q}{N_2}] t_c + \sqrt{(n-1)\tau})^2 + (\sqrt{\frac{P}{N_1}}[\frac{R}{N_2}] t_c + \sqrt{(n-1)\tau})^2$</td>
</tr>
<tr>
<td>$A_c(n,2,4)$</td>
<td>$n\tau + (\sqrt{\frac{P}{N_1}}[\frac{Q}{N_2}] t_c + \sqrt{(n-1)\tau})^2 + (\sqrt{\frac{P}{N_1}}[\frac{R}{N_2}] t_c + \sqrt{(n-1)\tau})^2$</td>
</tr>
<tr>
<td>$A_c(n,2,5)$</td>
<td>$n\tau + (\sqrt{\frac{P}{N_1}}[\frac{Q}{N_2}] t_c + \sqrt{(n-1)\tau})^2 + (\sqrt{\frac{P}{N_1}}[\frac{R}{N_2}] t_c + \sqrt{(n-1)\tau})^2$</td>
</tr>
</tbody>
</table>

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 $N_1 \times N_2 \times N_3$ subset of processors compute the product of a $P \times Q$ matrix and a $Q \times N_3$ matrix. If the matrices are initially allocated such that there are distinct submatrices $P \times Q$ and $Q \times N_3$ assigned to each set of $N_3$ processors then the multiplication in each subset is the same as in the two-dimensional partitioning, except that $Q$ is replaced by $Q \times N_3$. In addition, there is an accumulation phase at the end. The number of arithmetic operations for this part of the computation is $\left[\frac{P}{N_1}\right][\frac{R}{N_2}] \log N_3$ without any pipelining, and all partial products being accumulated in the same way. The matrix $A$ is allocated among $N_3$ 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 $\sum_{i=1}^{n_3} \left[\frac{P}{N_1}\right][\frac{R}{N_3}] t_a$. The communication complexity for the reduction is $\sum_{i=1}^{n_3} \left[\frac{P}{N_1}\right][\frac{R}{N_3}] t_c + \sum_{i=1}^{n_3} \left[\frac{P}{N_1}\right][\frac{R}{N_2}] t_c + \left(1 - \frac{1}{N_3}\right)[\frac{P}{N_1}][\frac{R}{N_3}] t_c + \sum_{i=1}^{n_3} \left[\frac{P}{N_1}\right][\frac{R}{N_2}] \tau$. When $\left[\frac{P}{N_1}\right][\frac{R}{N_2}] \geq N_3$, it is an all-to-all reduction, and the communication complexity of the reduction is approximately $\left(1 - \frac{1}{N_3}\right)[\frac{P}{N_1}][\frac{R}{N_3}] t_c + \sum_{i=1}^{n_3} \left[\frac{P}{N_1}\right][\frac{R}{N_2} \tau]$. For detailed complexity analysis, see [7]. The code is given below.
/* Algorithm A(1,3,1): */
/* Matrix C is partitioned as \( N_1' \times N_2'N_3' \). */
\[ l3(c_1, c_2, c_3, i', j'') \text{ over } [0 : N_1'] \times [0 : N_2'] \times [0 : N_3'] \times [0 : \frac{P}{N_1'}] \times [0 : \frac{R}{N_2'}] = \]
\[ c(c_1 \frac{P}{N_1'} + i', c_2 \frac{R}{N_2'} + j''), \]
/* Matrix D is partitioned as \( N_1'N_3' \times N_2'. */
\[ ld3(d_1, d_2, d_3, j', k') \text{ over } [0 : N_1'] \times [0 : N_2'] \times [0 : N_3'] \times [0 : \frac{P}{N_1'}] \times [0 : \frac{R}{N_2'}] = \]
\[ d(d_1 \frac{P}{N_1'} + d_2 \frac{R}{N_2'} + j', d_3 \frac{R}{N_2'} + k'), \]
/* Broadcast C along \( N_2' \) direction. */
\[ lc3(a_1, a_2, a_3, i', j') \text{ over } [0 : N_1'] \times [0 : N_2'] \times [0 : N_3'] \times [0 : \frac{P}{N_1'}] \times [0 : \frac{R}{N_2'}] = \]
\[ lc3(a_1, [\frac{i'N_1'N_3'}{Q'}], a_3, i', j' \mod \frac{N_3'}{N_2'}), \]
/* Broadcast D along \( N_1' \) direction. */
\[ ld3(b_1, b_2, b_3, j', k') \text{ over } [0 : N_1'] \times [0 : N_2'] \times [0 : N_3'] \times [0 : \frac{P}{N_1'}] \times [0 : \frac{R}{N_2'}] = \]
\[ ld3([\frac{j'N_1'N_3'}{Q'}], b_2, b_3, j' \mod \frac{N_3'}{N_2'}, k'), \]
/* Compute partial inner product locally. */
\[ la3(a_1, a_2, a_3, i', k') \text{ over } [0 : N_1'] \times [0 : N_2'] \times [0 : N_3'] \times [0 : \frac{P}{N_1'}] \times [0 : \frac{R}{N_2'}] = \]
\[ \{ l3(\text{row}(a_1, a_2, a_3, i', j') \ast ld3(\text{col}(a_1, a_2, a_3, j', k')) 0 \leq j' < \frac{Q'}{N_3'} \}, \]
/* Reduction along \( N_3' \) direction. */
\[ la3.red(a_1, a_2, a_3, i', k') \text{ over } [0 : N_1'] \times [0 : N_2'] \times [0 : N_3'] \times [0 : \frac{P}{N_1'}] \times [0 : \frac{R}{N_2'}] = \]
\[ \{ l3.red([\frac{i'N_1'N_3'}{Q'}], [\frac{a_3N_3'}{N_2'}], a_1, a_2, a_3, i', k') 0 \leq a_3 < \frac{N_3'}{N_2'} \}, \]
/* Relabeling processor indices as two-dimensional. */
\[ la2(a_1, a_2, i', k') \text{ over } [0 : N_1'] \times [0 : N_2'] \times [0 : \frac{P}{N_1'}] \times [0 : \frac{R}{N_2'}] = \]
\[ la2([\frac{i'N_1'}{P'}], [\frac{kN_3'}{R'}], i, k \mod \frac{P}{N_1'}, k \mod \frac{R}{N_2'}), \]
/* Resulting matrix A is partitioned as \( N_1 \times N_2. */
\[ a(i, k) \text{ over } [0 : P] \times [0 : R] = la2([\frac{iN_1'}{P'}], [\frac{kN_3'}{R'}], i \mod \frac{P}{N_1'}, k \mod \frac{R}{N_2'}) \]

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 \( N_1'N_2'N_3' \) 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 \( N_1' \times N_2' \times N_3' \) blocks, and \( N_1' \times N_3' \times N_2' \) blocks, respectively. The transformation can be specified as follows.

\[ lc3(c_1, c_2, c_3, i', j'') \text{ over } [0 : N_1'] \times [0 : N_2'] \times [0 : N_3'] \times [0 : \frac{P}{N_1'}] \times [0 : \frac{R}{N_2'}] = \]
\[ lc2(c_1, c_2, c_3, i', j', \text{mod } \frac{P}{N_1'}, c_2 \times \frac{N_3'}{N_2'}, \text{mod } \frac{P}{N_1'}, \text{mod } \frac{R}{N_2'}, \text{mod } \frac{R}{N_2'}) \]
\[ ld3(d_1, d_2, d_3, j', k') \text{ over } [0 : N_1'] \times [0 : N_2'] \times [0 : N_3'] \times [0 : \frac{P}{N_1'}] \times [0 : \frac{R}{N_2'}] = \]
\[ ld2(d_1, d_2, d_3, j', k', \text{mod } \frac{P}{N_1'}, \text{mod } \frac{R}{N_2'}) \]

19
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.tzp1(\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'),
\left[\frac{2^t\alpha - i'}{nN}\right] \mod 2 = 0 \rightarrow
\ll 0 \leq j < 2^{t-1} \frac{Q}{nN} \rightarrow lc.tzp1(\alpha, u, i', j', t - 1)
\quad else \rightarrow lc.tzp1(\alpha \oplus 2^{(u-t)\mod n}, u, i', j' - 2^{t-1} \frac{Q}{nN}, t - 1) \gg,
\quad else \rightarrow
\ll 0 \leq j < 2^{t-1} \frac{Q}{nN} \rightarrow lc.tzp1(\alpha \oplus 2^{(u-t)\mod n}, u, i' + \frac{P}{2^t}, j', t - 1)
\quad else \rightarrow lc.tzp1(\alpha, u, i' + \frac{P}{2^t}, j' - 2^{t-1} \frac{Q}{nN}, t - 1) \gggg,
lc.tzp(\alpha, i', j) over \([0 : N]\times[0 : \frac{P}{2^t}] = \times[0 : Q] =
lc.tzp1(\alpha, \left\lfloor \frac{\ln N}{Q} \right\rfloor \mod n, i', \left[\frac{\ln N}{Q} \right]\frac{Q}{nN} + j \mod \frac{Q}{nN}, n)

/* nRSBT reduction. */
/* Between columns, n-port, binary encoding. */
lr.red1(\alpha, u, i', k', t) over \([0 : N]\times[0 : n]\times[0 : \frac{P}{2^t}] \times[0 : \frac{R}{2^t}]\times[0 : n] =
\ll t = 0 \rightarrow lr(\alpha, u \frac{R}{2^t} + i', sh(u, \left\lfloor \frac{\ln N}{R} \right\rfloor \frac{R}{nN} + k' \mod \frac{R}{nN}),
\left[\frac{2^{t-1} \alpha - i'}{nN}\right] \mod 2 = 0 \rightarrow lr.red1(\alpha, u, i', k', t - 1) + lr.red1(\alpha \oplus 2^{(u-t)\mod n}, u, i', k', t - 1),
\quad else \rightarrow lr.red1(\alpha, u, i', k' + \frac{R}{2^t}, t - 1) + lr.red1(\alpha \oplus 2^{(u-t)\mod n}, u, i', k' + \frac{R}{2^t}, t - 1) \gg,
\( \text{la\_red}(\alpha, i, k') \) over \([0 : N) \times [0 : P) \times [0 : \frac{R}{N}) = \text{la\_red1}(\alpha, \frac{i}{n}, i \mod \frac{P}{n}, k', n) \)

### A.2 Two-dimensional partitioning

/* SBT broadcasting (row direction, one-port). */
\[ \text{lc\_row1}(\alpha, \alpha_1, \alpha_2, i', j', t) \] over \([0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : 2^t \frac{Q}{N_2^2}) \times [0 : n_2] = \]
\[
\leq t = 0 \rightarrow \text{lc}(\alpha_1, \alpha_2, i', j'), \\
\text{else} \rightarrow \leq 0 \leq j' < 2^{t-1} \frac{Q}{N_2^2} \rightarrow \text{lc\_row1}(\alpha_1, \alpha_2, i', j', t - 1), \\
\text{else} \rightarrow \text{lc\_row1}(\alpha_1, \alpha_2 \oplus 2^{t-1}, i', j' - 2^{t-1} \frac{Q}{N_2^2}, t - 1) \gg, \\
\]
\[ \text{lc\_row}(\alpha_1, \alpha_2, i', j) \] over \([0 : N_1) \times [0 : N_2) \times [0 : \frac{P}{N_1}) \times [0 : Q) = \text{lc\_row1}(\alpha_1, \alpha_2, i', j \oplus \alpha_2 \frac{Q}{N_2^2}, n_2) \]

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

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

/* nRSBT broadcasting (column direction, n-port). */
\[ \text{ld\_col1}(\alpha_1, \alpha_2, u, j', k', t) \] 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_1 N_2}) \times [0 : n_1) = \]
\[
\leq t = 0 \rightarrow \text{ld}(\alpha_1, \alpha_2, u \frac{R}{n_1 N_2} + j', k'), \\
\text{else} \rightarrow \leq 0 \leq j' < 2^{t-1} \frac{Q}{N_1} \rightarrow \text{ld\_col1}(\alpha_1, \alpha_2, u, j', k', t - 1), \\
\text{else} \rightarrow \text{ld\_col1}(\alpha_1 \oplus 2^{t-1} \text{mod} n_1, \alpha_2, u, j' - 2^{t-1} \frac{Q}{n_1 N_2}, k', t - 1) \gg, \\
\]
\[ \text{ld\_col}(\alpha_1, \alpha_2, u, j', k) \] over \([0 : N_1) \times [0 : N_2) \times [0 : n_1) \times [0 : \frac{Q}{N_1}) \times [0 : Q) = \text{ld\_col1}(\alpha_1, \alpha_2, [\frac{k N_1}{R}], k' \text{mod} \frac{R}{n_1 N_2}, (\frac{sh(u, [\frac{N_1}{Q}]) \oplus \alpha_1)}{n_1} \frac{Q}{N_1}, n_1) \]

21
B Matrix Multiplication

B.1 One-dimensional partitioning

/* A Gray code Exchange alg. A(1,1,1). */
/* Column partitioning, Gray code encoding. */
lc(α, i, j, t) over [0 : N) x [0 : P) x [0 : R/N] x [0 : N) =
  \langle t = 0 \rightarrow c(\alpha, \alpha \oplus N/J, j'),
  \text{else} \rightarrow lc(G^{-1}(\alpha) \oplus 2^{T(t-1)}), i, j', t - 1 \rangle \rangle,
ld(α, j, k') over [0 : N) x [0 : Q) x [0 : R/N] = d(j, \alpha \oplus P/N, k'),
la(α, i, k', t) over [0 : N) x [0 : P) x [0 : R/N] x [0 : N) =
  \langle t = 0 \rightarrow 0,
  \text{else} \rightarrow la(\alpha, i, k', t - 1) + (\langle [lc(\alpha, i, j', t - 1)
  * ld(\alpha, (\alpha \oplus (t - 1)) \oplus j', k') 0 \leq j' < \frac{R}{N} \rangle) \rangle,
a(i, k) over [0 : P) x [0 : R) = la(\lfloor \frac{kb}{N} \rfloor, i, k \mod \frac{R}{N}, N)

B.2 Two-dimensional partitioning

The index l in the following code denotes the rank of the \(\max(N_1, N_2)\) 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(α₁, α₂, l, i', j', t) over [0 : N₁) x [0 : N₂) x [0 : \frac{N_{max}}{N_{min}}) x [0 : \frac{P}{N_{max}}) x [0 : \frac{Q}{N_{max}}) x [0 : N_{max}] =
  \langle N₁ \geq N₂ \rightarrow \langle t = 0 \rightarrow c(\alpha₁ \frac{P}{N₁} + l', \alpha₂ \frac{Q}{N₂} + l \frac{R}{N₁} + j'),
  \text{/* Initial alignment. */}
  \text{t = 1 \rightarrow lc(α₁, \lfloor \frac{(\alpha₁ \frac{N₂}{N₁} + l + α₁) \mod N₂}{N₁} \rfloor, (l + α₁) \mod N₂, i', j', 0),}
  \text{/* The last block gets from next proc. */}
  \text{l = \frac{N_{max}}{N_{min}} - 1 \rightarrow lc(α₁, (α₂ + 1) \mod N₂, 0, i', j', t - 1),}
  \text{/* Other blocks get from right locally. */}
  \text{else \rightarrow lc(α₁, α₂, l + 1, i', j', t - 1) \rangle},
  \text{else \rightarrow lc(α₁, α₂, l + 1, i', j', t - 1) \rangle},
ld(α₁, α₂, l, j', k', t) over [0 : N₁) x [0 : N₂) x [0 : \frac{N_{max}}{N_{min}}) x [0 : \frac{Q}{N_{max}}) x [0 : \frac{R}{N_{max}}) x [0 : N_{max}] =
  \langle N₁ \leq N₂ \rightarrow \langle t = 0 \rightarrow d(\alpha₁ \frac{Q}{N₁} + l \frac{R}{N₁} + j', \alpha₂ \frac{R}{N₂} + k'),
  \text{t = 1 \rightarrow ld(\lfloor \frac{(α₁ \frac{N₂}{N₁} + l + α₁) \mod N₂}{N₁} \rfloor, α₂, \lfloor \frac{(l + α₂) N₂}{N₁} \rfloor \mod \frac{N₂}{N₁}, j', k', 0),

22
\[
\begin{align*}
l &= \frac{N_{\text{max}}}{N_{\text{min}}} - 1 \rightarrow \text{ld}((\alpha_1 + 1) \mod N_1, \alpha_2, 0, j', k', t - 1), \\
\text{else} & \rightarrow \text{ld}(\alpha_1, \alpha_2, l + 1, j', k', t - 1) \gg,
\end{align*}
\]

\[
\begin{align*}
&\ll t = 0 \rightarrow \text{ld}(\alpha_1 \frac{Q}{N_2} + j', \alpha_2 \frac{P}{N_2} + l \frac{R}{N_2} + k'), \\
&t = 1 \rightarrow \text{ld}((\alpha_2 \frac{N_2}{N_1} + l + \alpha_1) \mod N_1, \alpha_2, l, j', k', 0), \\
\text{else} & \rightarrow \text{ld}((\alpha_1 + 1) \mod N_1, \alpha_2, l, j', k', t - 1) \gg,
\end{align*}
\]

\[
\begin{align*}
l_a(\alpha_1, \alpha_2, l, i', k', t) & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_{\text{max}}}] \times [0 : \frac{R}{N_{\text{max}}}] \times [0 : N_{\text{max}}] = \\
&\ll t = 0 \rightarrow 0, \\
\text{else} & \rightarrow \text{la}(\alpha_1, \alpha_2, l, i', k', t - 1) + (\| [\text{lc}(\alpha_1, \alpha_2, l, i', j', t - 1) \\
* \text{ld}(\alpha_1, \alpha_2, l, j', k', t - 1)]0 \leq j' < \frac{Q}{N_{\text{max}}}] \gg,
\end{align*}
\]

\[
\begin{align*}
a(i, k) & \text{ over } [0 : P] \times [0 : R] = \\
&\ll N_1 \geq N_2 \rightarrow \text{la}([\frac{P}{N_1}], [\frac{R}{N_2}], [\frac{N_2}{N_1}], \text{mod } \frac{N_2}{N_1}, i \text{ mod } \frac{P}{N_1}, k \text{ mod } \frac{R}{N_2}, N_1), \\
\text{else} & \rightarrow \text{la}([\frac{P}{N_1}], [\frac{R}{N_2}], [\frac{P}{N_1}], \text{mod } \frac{N_2}{N_1}, i \text{ mod } \frac{P}{N_1}, k \text{ mod } \frac{R}{N_2}, N_2) \gg.
\end{align*}
\]

/* Algorithm A(-,2,4): */

\[
\begin{align*}
\text{ld}_\text{tsp}(\alpha_1, \alpha_2, j', k') & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : \frac{Q}{N_2}] \times [0 : \frac{R}{N_2}] = \\
&\text{ld}(((\alpha_1 \frac{Q}{N_2} + j')/\frac{Q}{N_2}, [(\alpha_2 \frac{P}{N_2} + k')/\frac{R}{N_2}]), (\alpha_1 \frac{Q}{N_2} + j') \mod \frac{Q}{N_2}, (\alpha_2 \frac{P}{N_2} + k') \mod \frac{R}{N_2}), \\
\text{ld}_\text{tsp}_\text{row}(\alpha_1, \alpha_2, i, j') & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : \frac{Q}{N_2}] \times [0 : R] = \\
&\text{ld}_\text{tsp}(\alpha_1, [\frac{N_2}{N_1}], j', k \text{ mod } \frac{P}{N_1}), \\
\text{la}(\alpha_1, \alpha_2, i', k) & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_1}] \times [0 : R] = \\
&\| [\text{lc}(\alpha_1, \alpha_2, i', j') * \text{ld}_\text{tsp}_\text{col}(\alpha_1, \alpha_2, j', k)]0 \leq j' < \frac{Q}{N_2}], \\
\text{la}_\text{red}(\alpha_1, \alpha_2, i', k') & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_1}] \times [0 : \frac{R}{N_2}] = \\
&\| [\text{la}(\alpha_1, \alpha_2, i', \alpha_2 \frac{P}{N_1} + k')0 \leq \alpha_2 < N_2]
\end{align*}
\]

/* Algorithm A(-,2,5): */

\[
\begin{align*}
\text{ld}_\text{tsp}(\alpha_1, \alpha_2, j', k') & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : \frac{Q}{N_2}] \times [0 : \frac{R}{N_2}] = \\
&\text{ld}(((\alpha_1 \frac{Q}{N_2} + j')/\frac{Q}{N_2}, [(\alpha_2 \frac{P}{N_2} + k')/\frac{R}{N_2}]), (\alpha_1 \frac{Q}{N_2} + j') \mod \frac{Q}{N_2}, (\alpha_2 \frac{P}{N_2} + k') \mod \frac{R}{N_2}), \\
\text{lc}_\text{col}(\alpha_1, \alpha_2, i, j') & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : P] \times [0 : \frac{Q}{N_2}] = \\
&\text{lc}([\frac{P}{N_1}], i \text{ mod } \frac{P}{N_1}, j'), \\
\text{la}_\text{tsp}(\alpha_1, \alpha_2, i, k') & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : P] \times [0 : \frac{R}{N_2}] = \\
&\| [\text{lc}_\text{col}(\alpha_1, \alpha_2, i, j') * \text{ld}_\text{tsp}(\alpha_1, \alpha_2, j', k')0 \leq j < \frac{Q}{N_2}], \\
\text{la}_\text{tsp}_\text{red}(\alpha_1, \alpha_2, i', k') & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_1}] \times [0 : \frac{R}{N_2}] = \\
&\| [\text{la}_\text{tsp}(\alpha_1, \alpha_2, i', \alpha_2 \frac{P}{N_1} + i')0 \leq \alpha_2 < N_2], \\
\text{la}(\alpha_1, \alpha_2, i', k') & \text{ over } [0 : N_1] \times [0 : N_2] \times [0 : \frac{P}{N_1}] \times [0 : \frac{R}{N_2}] = \\
&\text{la}_\text{tsp}_\text{red}([\alpha_1 \frac{P}{N_1} + i'/\frac{P}{N_1}], [\alpha_2 \frac{P}{N_2} + k'/\frac{R}{N_2}], (\alpha_1 \frac{P}{N_1} + i') \text{ mod } \frac{P}{N_1}, (\alpha_2 \frac{P}{N_1} + k') \text{ mod } \frac{R}{N_1})
\end{align*}
\]
Acknowledgement

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

References


