Optimal All-to-All Personalized Communication with Minimum Span on Boolean Cubes

The Harvard community has made this article openly available. Please share how this access benefits you. Your story matters

<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Citable link</td>
<td><a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:25680330">http://nrs.harvard.edu/urn-3:HUL.InstRepos:25680330</a></td>
</tr>
<tr>
<td>Terms of Use</td>
<td>This article was downloaded from Harvard University’s DASH repository, and is made available under the terms and conditions applicable to Other Posted Material, as set forth at <a href="http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA">http://nrs.harvard.edu/urn-3:HUL.InstRepos:dash.current.terms-of-use#LAA</a></td>
</tr>
</tbody>
</table>
Optimal All-to-All Personalized Communication with Minimum Span on Boolean Cubes

S. Lennart Johnsson
Ching-Tien Ho

TR-18-91
April 1991

Parallel Computing Research Group
Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

Optimal All-to-All Personalized Communication with Minimum Span on Boolean Cubes

S. Lennart Johnsson  
Harvard University and  
Thinking Machines Corp.  
Cambridge, MA  
johnsson@think.com

Ching-Tien Ho  
IBM Almaden Research Center  
650 Harry Road  
San Jose, CA 95120  
ho@ibm.com

Abstract

All-to-all personalized communication is a class of permutations in which each processor sends a unique message to every other processor. We present optimal algorithms for concurrent communication on all channels in Boolean cube networks, both for the case with a single permutation, and the case where multiple permutations shall be performed on the same local data set, but on different sets of processors. For \( K \) elements per processor our algorithms give the optimal number of elements transferred, \( K/2 \). For a succession of all-to-all personalized communications on disjoint subcubes of \( \beta \) dimensions each, our best algorithm yields \( \frac{K}{2} + \sigma - \beta \) element exchanges in sequence, where \( \sigma \) is the total number of processor dimensions in the permutation. An implementation on the Connection Machine of one of the algorithms offers a maximum speed-up of 50% compared to the previously best known algorithm.

1 Introduction

We give simple, yet optimal, schedules for all-to-all personalized communication on Boolean cubes with concurrent communication on all channels of every processor. An example of an architecture that allows for such communication is the Connection Machine. The schedules avoid indirect addressing in the data exchanges by performing a local alignment of the data in each processor prior to, and after, the data interchanges between processors.

In addition to optimal utilization of communication channels we are also concerned with the duration of the orbits of the elements, i.e., the time from the first motion of an element until it reaches its destination. The orbit length is important for pipelining several all-to-all personalized communications (AAPC). With \( K \) elements per processor and AAPC in \( \beta \)-cubes our first algorithm requires \( \frac{K}{2} + \beta - (\frac{K}{2} \mod \beta) \) element transfers in sequence for a single AAPC, except if \( \frac{K}{2} \mod \beta = 0 \). Then, the number of element transfers is optimal, \( \frac{K}{2} \). The orbit length for all pairs of elements is \( \beta \). One or the other element in a pair is always exchanged. Our second algorithm has the minimal number of element exchanges for any \( K \) and \( \beta \), but the maximum orbit length is \( \beta + (\frac{K}{2} \mod \beta) \). Our third algorithm illustrates how pipelining can be combined with exchange sequences starting in arbitrary dimensions to yield an optimal number of element exchanges in sequence. But, the orbit length is \( \frac{K}{2} \). Our last algorithm requires \( \frac{K}{2} \) element transfers in sequence for any \( \beta \), and has a maximum orbit length of \( \beta \).

All-to-all personalized communication is a frequently used class of permutations in multi-processor systems. Examples of all-to-all personalized communication are bit-reversal, vector reversal, matrix transposition, shuffle permutations, and conversion between cyclic and consecutive mapping [5] for allocations of the arrays such that a number of storage dimensions are exchanged with the same number of processor dimensions. All-to-all personalized communication can also be combined with code conversion, such as conversion between binary code, and binary-reflected Gray code [6, 10], which is often used for encoding of arrays on Boolean cubes. As an example of the use of a succession of AAPC’s consider the computation of high radix FFT (Fast Fourier Transform) on a large multi-processor with relatively few data points per processor. It can be performed through a sequence of local FFT. This requires a sequence of all-to-all personalized communications on the same local memory address bits, but on different processor address bits.

Saad and Schultz [12] have suggested a recursive AAPC algorithm based on \( 2^\beta \) translated binomial trees. The algorithm requires \( \beta \frac{K}{2} \) element transfers in sequence. In [7] we presented algorithms based on balanced trees, and rotated binomial trees, both attaining the minimal number of element transfers in sequence, \( \frac{K}{2} \), if \( K \) is a multiple of \( \beta \). For other values of \( K \) the algorithms are optimal within a small constant factor \( \leq 24\% \) for \( \beta > 4 \) [3]. Independently, Stout and Wagar [13] gave an algorithm with the same complexity. Later, Bertsekas et al. [1] presented an algorithm optimal for any \( \beta \). A detailed optimal scheduling algorithm has been presented by Edelman [2], who also implemented the algorithm on the Connection Machine. The algorithm uses indirect addressing, and has orbit lengths of order \( O(2^\beta) \). Varvarigos and Bertsekas
related AAPC on hypercubes and tori to a matrix decomposition problem.

2 All-to-all personalized communication on Boolean cubes

2.1 Preliminaries

A Boolean n-cube has $N = 2^n$ nodes. Each node has $n$ neighbors, which with the conventional binary addressing scheme correspond to the $n$ different single bit complementations of the bits in a node address. We use $\oplus$ to denote the bitwise exclusive-or operation. The local address space in each node is $K = \{0, 1, \ldots, 2^k-1\}$, and the global address is $(i|j)$ where $i$ denotes the processor address and $j$ is the memory location.

**Definition 1** Let $S_i \subset \{0, 1, \ldots, k-1\}$, $S_p \subset \{k, k+1, \ldots, k+n-1\}$, $|S_i| = |S_p|$. $f$ be a one-to-one mapping $S_i \rightarrow S_p$ and $g$ be a one-to-one mapping $S_p \rightarrow S_i$. An AAPC is a permutation defined by $a_i \rightarrow a_{f(i)}$ for all $i \in S_i$ and $a_j \rightarrow a_{g(j)}$ for all $j \in S_p$.

An example of the bit-reversal permutation is $(a_0a_1a_2a_3) \rightarrow (a_3a_2a_1a_0)$ and that of the matrix transposition is $(a_0a_1a_2a_3a_4a_5) \rightarrow (a_0a_2a_4a_1a_3a_5)$. Both are examples of AAPC.

Let $|S_i| = \beta \leq \min(k,n)$. If $\beta < k$ then the permutation is repeated $2^{k-\beta}$ times. For instance, if there are eight bits for the local memory, $k = 8$, and three are part of the permutation, $\beta = 3$, then the permutation is repeated $2^{5-3} = 32$ times. Similarly, if there are processor dimensions not included in the permutation, then the permutation consists in a number of permutations in disjoint subcubes. Each such subcube is identified by the address bits not included in the permutation.

The *relative address* of a local memory address $i$ in processor $j$ with respect to its destination is $j \oplus i$, where $\oplus$ is performed bit-wise. A *homogeneous scheduling* has communication schedules for each node that only depend on the relative addresses. This implies that node $i$ sends data to node $j \oplus i$ in the same dimension and step as node $0$ sends data to node $j$. The message path from node $i$ to node $i \oplus j$ is a translation with respect to node $i$ (i.e., exclusive-or) of the path from node $0$ to node $j$. We only consider homogeneous schedules. For such schedules it is sufficient to consider homogeneous schedules for node 0.

**Definition 2** The *span* for message $j$, $span(j)$, is the number of exchanges from the step during which the message starts its path through the cube until it arrives at its destination in a single AAPC. Formally, if the message leaves the source during step $t_1$ and arrives at the destination during step $t_2 > t_1$, then $span(j) = t_2 - t_1 + 1$. The span of the communication for a single AAPC is the maximum span for any message in the communication, i.e., $\overline{span} = \max_j span(j)$.

In the following, we consider AAPC of the form $(j|0) \rightarrow (j|i)$. Based on this, algorithms for the general AAPC $(j|0) \rightarrow (g(j)|f(i))$ can be derived.

2.2 Routing for the permutation $(j|i) \rightarrow (j|i)$

The permutation $(j|i) \rightarrow (j|i)$ is equivalent to the transposition of a matrix stored with one row per processor to the storage of one column per processor. The permutation can also be viewed as changing the allocation of a one-dimensional array from consecutive storage to cyclic storage [5].

The routing algorithms we present use only direct addressing in the data exchanges between processors. All processors access the same (local) address during the same step. To accomplish this properly a local alignment precedes the exchange phase. Depending on the relative times for the alignment and realignment phases and the exchanges with direct and indirect addressing, avoiding indirect addressing in the exchanges may be desirable. On the Connection Machine avoiding indirect addressing in the exchange phase results in a significant speed-up.

**Phase 1: Alignment.** Sort the local data by relative address, i.e., perform the operation $(j|0) \rightarrow (j|i) \oplus j$, where $i$ is the local memory address and $j$ is the processor address.

**Phase 2: Interprocessor exchange.** The interprocessor exchange phase implements the operation: $(j|0) \rightarrow (j|i) \oplus i$.

**Phase 3: Realignment.** The realignment to restore the local memory order is identical to phase 1.

The alignment is performed on the storage dimensions involved in the exchange. The treatment of other storage dimensions is arbitrary.

**Pairing**

A memory location is subject to an exchange operation in processor dimensions that correspond to non-zero bits in its (relative) address. Hence, local memory location zero retains its content throughout phase 2, whereas memory location $(11\ldots1)$ exchanges content with neighboring nodes in every dimension. Similarly, memory location $(00\ldots01)$ only exchanges its content with the neighboring node in the least significant processor dimension where local memory location $(11\ldots10)$ exchanges content with all neighboring nodes, except the neighbor in the least significant processor dimension.

For any local memory address $i$, either $i$ or $\overline{i}$ is subject to exchange in any step of the algorithm. In the following we will always consider pairs of memory locations defined by $(i, \overline{i})$. If there are local storage dimensions in addition to the dimensions involved in the AAPC ($K > 2^{\beta}$), then the pairing is simply repeated $K/2^n$ times. The pairing is always made on the dimensions included in the AAPC. Other dimensions are of
no consequence with respect to contention for communication, or correctness.

Permutation for a set of $\beta$ pairs will be considered together. The partitioning of all pairs to sets can be arbitrary. For instance, in a set of $\beta$ contiguous memory locations and their complements, pair $s$, $0 \leq s < \beta$, is exchanged in dimensions $s, (s + 1) \mod \beta, (s + 2) \mod \beta, \ldots, (s + \beta - 1) \mod \beta$. After exchanges the $\beta$ pairs have performed the necessary communications. Every channel is used in every step, and each item follows a minimum-length path. The procedure is repeated for the next $\beta$ pairs, etc. Algorithm 1 yields the following lemma.

**Lemma 1** All-to-all personalized communication in a $\beta$-cube for $K$ local memory locations requires at most $\left\lceil \frac{K}{\beta} \right\rceil$ element exchanges in sequence with concurrent exchanges on all channels.

The correctness of the above algorithm follows from the following consideration:

$$
\begin{align*}
\text{Phase 1:} & \quad (j|i)^{12} (j|i \oplus j), \\
\text{Phase 2:} & \quad (j|i)^{12} (i \oplus j|i), \\
\text{Phase 3:} & \quad (j|i)^{12} (j|i \oplus j),
\end{align*}
$$

or $(j|i)^{12} (j|i \oplus j)^{23} (i \oplus j \oplus j|i \oplus j) = (i|i \oplus j)^{23} (i|i \oplus j) = (i|i \oplus j)$.

Phase 2 of Algorithm 1 proceeds in $\left\lfloor \frac{K}{\beta} \right\rfloor$ subsphases with $\beta$ exchanges per subsphase. If $\beta$ does not divide $\frac{K}{\beta}$, then the last subsphase ($\beta$ communication steps) does not fully use all dimensions. This property causes the algorithm to be slightly non-optimal with respect to channel utilization. However, the schedule is easily modified to an optimal schedule by having one subsphase include $\beta + (\frac{K}{\beta} \mod \beta)$ pairs. Consider the case for $\beta = 3$. There are a total of eight memory locations, or four pairs. By scheduling the four pairs as shown in Table 1, the optimal number of element transfers is achieved. The scheme is easily generalized to arbitrary $\beta$. We refer to this algorithm as Algorithm 2.

**Lemma 2** All-to-all personalized communication in a $\beta$-cube requires $\frac{K}{\beta}$ element transfers in sequence by Algorithm 2 with concurrent exchanges on all channels, and $K$ elements per processor.

There are many schedules that yield an optimal utilization of communication channels. For instance, the schedule in Table 2 is also optimal with respect to channel utilization, Algorithm 3. But, the last $\beta - 1$ pairs are in orbit for $2^{\beta-1}$ exchanges for an AAPC on $\beta$ bits. The schedule in [2] has a similar property. The long orbits are an disadvantage in the case several AAPC’s on different processor address bits shall be performed.

### Table 1: Optimal scheduling of $\beta + \frac{K}{\beta} \mod \beta$ locations, $\beta = 3$, $K = 8$.

<table>
<thead>
<tr>
<th>Memory location pair</th>
<th>Exchange step</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0000, 1111)</td>
<td>0 1 2 3 - - -</td>
</tr>
<tr>
<td>(0001, 1110)</td>
<td>- 0 1 2 3 - -</td>
</tr>
<tr>
<td>(0010, 1101)</td>
<td>- - 0 1 2 3 -</td>
</tr>
<tr>
<td>(0011, 1100)</td>
<td>- - - 0 1 2 3</td>
</tr>
<tr>
<td>(0100, 1011)</td>
<td>- - - 0 1 2 3</td>
</tr>
<tr>
<td>(0101, 1010)</td>
<td>3 - - - 0 1 2</td>
</tr>
<tr>
<td>(0110, 1001)</td>
<td>2 3 - - - 0 1</td>
</tr>
<tr>
<td>(0111, 1000)</td>
<td>1 2 3 - - - 0</td>
</tr>
</tbody>
</table>

### Table 2: The schedule for Algorithm 3.

<table>
<thead>
<tr>
<th>Memory location pair</th>
<th>Exchange step</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0000, 1111)</td>
<td>0 1 2 3 - - -</td>
</tr>
<tr>
<td>(0001, 1110)</td>
<td>- 0 1 2 3 - -</td>
</tr>
<tr>
<td>(0010, 1101)</td>
<td>- - 0 1 2 3 -</td>
</tr>
<tr>
<td>(0011, 1100)</td>
<td>- - - 0 1 2 3</td>
</tr>
<tr>
<td>(0100, 1011)</td>
<td>- - - 0 1 2 3</td>
</tr>
<tr>
<td>(0101, 1010)</td>
<td>3 - - - 0 1 2</td>
</tr>
<tr>
<td>(0110, 1001)</td>
<td>2 3 - - - 0 1</td>
</tr>
<tr>
<td>(0111, 1000)</td>
<td>1 2 3 - - - 0</td>
</tr>
</tbody>
</table>

The characteristics of the above algorithms are summarized in Table 3. For a single AAPC Algorithms 2 or 3 have the fewest (optimal) number of element transfers in sequence. If several AAPC’s shall be performed on distinct sets of processor dimensions, then it is desirable that the maximum span is $\beta$ in order to minimize the pipeline delay. In section 3 we present an algorithm with a maximum span of $\beta$, and $\frac{K}{\beta}$ element transfers in sequence.

### 3 Pipelining many all-to-all personalized communications

#### 3.1 Preliminaries

Performing $\sigma$ AAPC’s in sequence by pipelining successive applications of Algorithm 1 requires $\beta \lceil \frac{K}{\beta} \rceil + (\sigma - 1)\beta$ exchanges. For Algorithm 2 pipelining yields

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Element transfers</th>
<th>Max. Span</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\beta \lceil \frac{K}{\beta} \rceil$</td>
<td>$\beta$</td>
</tr>
<tr>
<td>2</td>
<td>$\frac{K}{\beta}$</td>
<td>$\beta + (\frac{K}{\beta} \mod \beta)$</td>
</tr>
<tr>
<td>3</td>
<td>$\frac{K}{\beta}$</td>
<td>$\beta + (\frac{K}{\beta} \mod \beta)$</td>
</tr>
</tbody>
</table>

Table 3: The number of element transfers and the span in Algorithms 1–3 on a $\beta$-cube.
\[ \frac{K}{2} + (\sigma - 1)(\beta + \frac{K}{2} \mod \beta) \] exchanges. Using the schedules below (Algorithm 4) with a span of \( \beta \) and pipelining yields \( \frac{K}{2} + \sigma - \beta \) exchanges in sequence. Compared to the pipelined usage of Algorithm 1 the gain may be \( \beta - 1 \) exchanges, and compared to Algorithm 2 \( (\sigma - 1)(\frac{K}{2} \mod \beta) \) exchanges. This gain in efficiency is obtained at the expense of a more complex scheduling for some of the local elements. The pipeline delay can be further reduced as described in Section 3.3.4.

The schedules in Algorithms 1 through 3 grouped local storage addresses by complement pairs. In Algorithm 4 we will use this grouping strategy as well as one based on necklaces to accomplish a span of \( \beta \) and an optimal number of element transfers in sequence for a single AAPC. A necklace \([1]\) is a set of addresses derived from each other through rotations of some fixed bit string. A necklace is \textit{full} if it has \( \beta \) distinct addresses for a string of length \( \beta \). Otherwise, it is \textit{degenerated}. The period of a bit string is the minimum number of rotations (\( \geq 0 \)) required to generate a bit string identical to the unrotated string. An address in a degenerate necklace is \textit{cyclic}. Addresses in full necklaces are \textit{non-cyclic}. For instance, the address \((0000)\) is cyclic with period one and the address \((0101)\) is cyclic with period two; the addresses \((0001)\) and \((0111)\) are non-cyclic. The address with smallest value in the necklace is a \textit{distinguished address}. A \textit{q-necklace} is a necklace in which each address has \( q \) bits equal to 1.

**Non-cyclic addresses**

The exchanges for the three local memory locations \((001)\), \((010)\), and \((100)\) obviously can be done concurrently. Similarly, locations \((011)\), \((110)\), and \((101)\) can be scheduled concurrently with respect to the least significant bit of \((011)\), the second least significant bit of \((110)\), and the most significant bit of \((101)\). The communication for the remaining bit that is one in each of these addresses can also be scheduled concurrently. The basic idea should now be clear: sets of \( \beta \) memory locations with addresses being cyclic rotations of each other are scheduled concurrently.

**Lemma 3** Addresses forming a full \( q \)-necklace can be scheduled to complete the permutation in \( q \) exchange steps, which is optimal. The span is \( q \leq \beta \).

In any optimal scheduling according to the lemma all communication channels are used in every exchange. For instance, an optimal schedule for \([0111, 1110, 1101, 1011]\) is

\[
\begin{align*}
0111: & \ 0, 1, 2, 3, 4 \\
1110: & \ 1, 2, 3, 4 \\
1101: & \ 2, 3, 0, 1 \\
1011: & \ 3, 0, 1, 2
\end{align*}
\]

**Lemma 4** A \( q \)-necklace of addresses obtained through bit-complementation of a full \( (\beta - q) \)-necklace is also a full necklace, and distinct if \( q \neq \beta/2 \). For \( q = \beta/2 \) the complement necklace may be the same as the original necklace.

Any full necklace and its complement, if distinct, can be scheduled together by pairing of addresses as in Algorithms 1 through 3. Hence, addresses in full \( q \)-necklaces can either be scheduled as blocks of \( \beta \) addresses during \( q \) exchange steps, or as \( 2 \beta \) addresses during \( \beta \) exchange steps by combining a full \( q \)-necklace with a full \((\beta - q)\)-necklace. For \( q = \beta/2 \) not every \( q \)-necklace has a matching complement necklace. But, pairs of such \( \beta/2 \)-necklaces can be scheduled by pairing of addresses through complementation. Such complement addresses belong to the same necklace. Hence, with the exception of addresses in at most one \( \beta/2 \)-necklace not having a distinct complement necklace, all addresses in a full necklace can be scheduled optimally by pairing of addresses through complementation.

**Cyclic addresses**

The scheduling for non-cyclic addresses cannot be used for cyclic addresses. For instance, location \([11\ldots 1]\) has period one, and must be matched with other locations in order to achieve maximum utilization of the communication channels. For every cyclic address the address formed through bit-complementation is also cyclic. Hence, pairs of cyclic addresses can be scheduled through bit-complementation (as in Algorithms 1 through 3). If the number of such pairs is \( \beta \), or a multiple thereof, then all cyclic addresses can also be scheduled to fully use the communication channels by scheduling \( \beta \) pairs at a time. When the number of pairs of cyclic addresses is not a multiple of \( \beta \), then we combine \( \beta - q \) pairs of cyclic nodes with a full \( q \)-necklace.

The schedule for the cyclic addresses combined with addresses in a full necklace is constructed based on a \( \beta \times \beta \) generating table. The first row of this table consists of the numbers \(0, 1, \ldots, \beta - 1\). Each other row is a one step left cyclic rotation of the previous row. Figure 1 shows the generating table for \( \beta = 5 \). Columns of the generating table represent time steps. Complement pairs of cyclic addresses share a row. The table entries specify the exchange dimension for the addresses. One or the other address in a complement pair must be exchanged in every dimension. Every row contains all dimensions. Which element in a complement pair is exchanged depends upon which address in the pair has the address bit set to one for the considered dimension.

\[
\begin{array}{ccccccc}
0 & 1 & 2 & 3 & 4 \\
1 & 2 & 3 & 4 & 0 \\
2 & 3 & 4 & 0 & 1 \\
3 & 4 & 0 & 1 & 2 \\
4 & 0 & 1 & 2 & 3
\end{array}
\]

Figure 1: Generating tables for \( \beta = 5 \).
necklace are derived by drawing \( q \) lines each of length \( \beta \) through the first \( q \) rows of the scheduling table. The rules for drawing the \( q \) lines are described as follows.

- The first line is drawn from the top of the first column down to the \( q \)th row, then \( \beta - q \) columns to the right.
- The second line, if any, starts in the second column. goes through \( q - 1 \) rows, then turns right through \( \beta - q \) columns, followed by a vertical turn to include one more row. Thus, it is also of length \( \beta \).
- Subsequent lines are drawn in a similar way by proceeding vertically one row less than the previous line, then proceeding horizontally through \( \beta - q \) columns, then proceeding vertically to the last row of the generating table reserved for the full necklace.
- The last line starts out horizontally.

The construction should be apparent from Figure 2. The first entry on each line corresponds to the distinguished address \((0^\beta 1^q)\) where \(1^q\) denotes the \( q \) consecutive 1-bits. The second entry on each line corresponds to a one-step left cyclic rotation of the distinguished address. The \( p \)th entry refers to an address which is the \( (p - 1) \)-step left cyclic rotations of the distinguished address. For instance, for \( \beta = 5 \) and \( q = 4 \), the first through the \( 8 \)th entries refer to address \((01111), (11110), (11101), (11011)\) and \((10111)\), respectively. The value of the \( p \)th line entry defines the dimension in which the contents of the \( p \)th address is exchanged during the time step given by the column in which the \( p \)th entry occurs.

The number of times an address is scheduled is equal to the number of lines, i.e., \( q \). It is easy to see that the first time an address is scheduled, it is scheduled in the dimension that corresponds to dimension zero in the distinguished address. For each address the next higher dimension modulo the number of dimensions, is scheduled the next time the address is scheduled for an exchange.

![Figure 2: Scheduling lines in the generating tables for \( \beta = 5 \) (\( q = 4 \)).](image)

For \( \beta = 5 \) and \( q = 4 \) the distinguished address is \((01111)\). The addresses in the necklace are \((01111), (11110), (11101), (11011)\) and \((10111)\). The complete schedule is shown in Table 4, which is derived from

<table>
<thead>
<tr>
<th>Address</th>
<th>Time step</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111</td>
<td>0 1 2 3 4</td>
</tr>
<tr>
<td>11110</td>
<td>1 2 3 - 4</td>
</tr>
<tr>
<td>11011</td>
<td>- 2 3 - 4</td>
</tr>
<tr>
<td>10111</td>
<td>- - 4 0 1</td>
</tr>
<tr>
<td>(110000,11111)</td>
<td>4 0 1 2 3</td>
</tr>
</tbody>
</table>

Table 4: Exchange dimension for type-2 scheduling for \( \beta = 5 \) and \( q = 4 \).

Lemma 5 There exists a set of addresses forming a full \( q \)-necklace such that any \( \beta - q \) complement pairs of cyclic addresses can be scheduled together with the \( q \)-necklace to complete all communication in \( \beta \) steps, which is optimal.

Proof: The \( q \)-necklace defined by addresses with \( q \) consecutive one bits in cyclic order is non-cyclic for \( q < \beta \). Deriving the schedule from the table guarantees that all dimensions are scheduled every time step precisely one address is scheduled for a dimension every time step, each complement pair is scheduled in every dimension, and every address in the full necklace is scheduled in each of the \( q \) dimensions with a bit set, and in no other dimension.

In summary, memory addresses are first partitioned into cyclic and non-cyclic addresses. The number of cyclic addresses is two if \( \beta \) is prime, otherwise it is of order \( O(\sqrt{2^\beta}) \) [4]. Cyclic addresses are paired through bit-complementation and divided into blocks of \( \beta \) pairs. The remaining \( \beta - q \) pairs is scheduled together with \( \beta \) non-cyclic addresses forming a \( q \)-necklace. All non-cyclic addresses can be scheduled individually, except the addresses scheduled with the cyclic addresses. Non-cyclic addresses can also be scheduled as complement pairs, except if there are no complement pairs forming distinct full necklaces. There are at most two such sets, one set forming a \( \beta/2 \)-necklace, if \( \beta \) is even, and one set of addresses forming the complement of the addresses scheduled with cyclic addresses.

Theorem 1 An all-to-all personalized communication on \( \beta \) processor dimensions can be performed in \( K \)
element exchanges in sequence for concurrent communication on all β ports of every processor. The maximum span is β. The number of element exchanges in sequence for all-to-all personalized communications in succession on different sets of β processor dimensions and the same set of local storage dimensions is $\frac{K}{2} + (\sigma - 1)\beta = \frac{K}{2} + \sigma - \beta$.

For the case where the succession of AAPC’s shall be performed on non-overlapping subcubes of different order, refer to [8].

4 Summary and Discussion

We have presented four schedules for a single all-to-all personalized communication, three of which are simple to implement. One algorithm (Algorithm 4) requires the optimal number of element exchanges in sequence, $\frac{K}{2}$, with a span of $\beta$. The other three algorithms have either minimal maximal orbit lengths, or maximum channel utilization, but not both.

Algorithm I has been implemented on the Connection Machine model CM-2. The exchanges require 40 µsec per element compared to 62 µsec for the schedule in [2]. The total expense for alignment and realignment is about 0.9 µsec per element (four bytes). Hence, the simplified algorithms presented here may yield a speed-up of up to 50% for a single AAPC, and a considerably reduced pipeline delay for multiple AAPC.

It should be noted that for single AAPC on a β-cube, Algorithm 4 can be easily blocked to β communication steps with the optimal number of element transfers preserved [9]. The blocking procedure can also be used to generate optimal schedules for channel widths greater than the width of the data item [9].

Acknowledgment The Connection Machine implementation of Algorithm I was performed by Michel Jacquemin of Yale University, Department of Computer Science, in a collaborative effort between Thinking Machines Corporation and INRIA, Centre de Sophia-Antipolis. Valuable comments on a draft were given by Roland Sweet of the University of Colorado at Denver.

References


