# Boolean Cube Emulation of Butterfly Networks Encoded by Gray Code

S. Lennart Johnsson and Ching-Tien Ho

YALEU/DCS/TR-764
February 1990

This work has been supported in part by AFOSR-89-0382
and by NSF/DARPA grant CCR-8908285.

# Boolean Cube Emulation of Butterfly Networks Encoded by Gray Code

S. Lennart Johnsson and Ching-Tien Ho
Department of Computer Science
Yale University

February 1990

**Abstract**

We present algorithms for butterfly emulation on binary-reflected Gray coded data that require the same number of element transfers in sequence in a Boolean cube network as for a binary encoding. The required code conversion is either performed in local memories, or though concurrent exchanges not effecting the number of element transfers in sequence. The emulation of a butterfly network with one or two elements per processor requires $n$ communication cycles on an $n$-cube. For more than two elements per processor one additional communication cycle is required for every pair of elements. The encoding on completion can be either binary, or binary-reflected Gray code, or any combination thereof without affecting the communication complexity.

## 1 Introduction

In this paper we show that butterfly emulation on binary-reflected Gray coded data embedded in a Boolean cube multiprocessor can be performed with the same communication complexity as if the data had been encoded in binary code. The order on completion is shuffled with respect to the input ordering, and the encoding can be in either binary code, or binary-reflected Gray code. A straightforward emulation on Gray coded data results in a communication complexity twice that of binary encoding for both packet and circuit switched communication. The communication complexity is affected by the scheme for assigning multiple elements to a processor when there are more elements than processors. We show that cyclic allocation yields the lowest communication complexity.

Boolean cube networks are versatile interconnection networks for distributed memory multiprocessors. Several other networks can be emulated with no loss of efficiency. The butterfly network is ideal for many important computations, such as Fast Fourier Transforms, parallel prefix operations and sorting. In distributed memory architectures the efficient utilization of the communication system is important with respect to performance. This problem consists of two components: data placement to minimize the need for communication, and data routing and scheduling to maximize the use of the available bandwidth (ideally for minimum length paths) while minimizing the congestion.

The number of data elements often exceeds the number of available processors. Two common schemes for allocating elements evenly to the memory units are *consecutive* and *cyclic* [4]. With the first type of allocation data elements with successive indices are allocated to the same memory

unit. In cyclic allocation consecutive elements are allocated to successive processors. For a Boolean $n$-cube and a data set with $P = 2^p$ elements, $p \geq n$ the $n$ least significant bits of the data index determine the processor address in *cyclic* allocation. In *consecutive* allocation the $n$ most significant bits of the data index determine the processor addresses.

A Boolean $n$-cube contains meshes of up to $n$ dimensions as subgraphs. Hence, the adjacency defined by a multi-dimensional mesh can be preserved by an appropriate addressing scheme, or data encoding. This fact has made the binary-reflected Gray code [14] a frequently used addressing scheme for Boolean cubes [11, 4]. The Gray code encoding can be applied to the entire address field, only to the processor address field, or independently to the processor and local memory address fields. In combination with the different schemes for allocating multiple data elements to the memories of the processors the following combinations arise.

| Consecutive allocation | Cyclic allocation |
|---|---|
| $(\underbrace{Gray}_{paddr}\ \underbrace{code}_{maddr})$ | $(\underbrace{Gray}_{maddr}\ \underbrace{code}_{paddr})$ |
| $(\underbrace{Gray\ code}_{paddr} \| \underbrace{Gray\ code}_{maddr})$ | $(\underbrace{Gray\ code}_{maddr} \| \underbrace{Gray\ code}_{paddr})$ |
| $(\underbrace{Gray\ code}_{paddr} \| \underbrace{Binary\ code}_{maddr})$ allocation | $(\underbrace{Gray\ code}_{maddr} \| \underbrace{Binary\ code}_{paddr})$ |
| $(\underbrace{Binary}_{paddr}\ \underbrace{code}_{maddr})$ | $(\underbrace{Binary}_{maddr}\ \underbrace{code}_{paddr})$ |

A Boolean $n$-cube has $N = 2^n$ nodes, and node $i$ is connected to nodes $i \oplus 2^j$, $j \in \{0 \ldots n-1\}$. A butterfly network with $P = 2^p$ rows has $p+1$ columns. A node in row $i, i \in [0, P-1]$ and column $j, j \in [0, p]$ is connected to nodes in rows $i$ and $i \oplus 2^j$ in column $j+1$, or to nodes in rows $i$ and $i \oplus 2^{p-1-j}$ in column $j+1$. The two definitions correspond to networks that are mirror images of each other. In a mapping of a butterfly network to a Boolean cube such that all butterfly nodes in a row are mapped to the same cube node, and rows assigned to cube nodes acccording to the binary encoding of the row index, the required communication for emulating the butterfly network corresponds exactly to the connections in the Boolean cube. The different stages of the butterfly network corresponds to communication in different dimensions of the Boolean cube. All communication steps are nearest neighbor communications.

All elements in a processor must be communicated in the same dimension for a given butterfly stage. Pipelining [8, 10], or multi-sectioning [8, 9] can be used to increase the utilization of the communication system. In cyclic data allocation the first $p-n$ stages are local for a data set of size $P = 2^p$, with the emulation proceeding from the most significant dimension to the least significant dimension. The last $n$ stages require inter-processor communication. In consecutive data allocation the first $n$ stages require inter-processor communication and the last $p-n$ stages are local. Each of the $\frac{P}{N}$ local memory addresses defines a butterfly network emulation with one data point per processor. These emulations are independent. For a binary encoding pipelining the computations yields $\frac{P}{N} + n - 1$ element transfers in sequence [8]. Multi-sectioning by $R = 2^r$, $r \leq n$, together with pipelining of the emulation on blocks of $R$ memory locations yield a communication complexity of $\frac{P}{2N} + (\frac{n}{r} - 1)\frac{R}{2}$ [9], which for $\frac{P}{N} \gg n$ is approximately half of what is required for the pipelining approach.

With the data encoded in a binary-reflected Gray code $G$ the addresses of $G(i)$ and $G(i \oplus 2^j)$ differ in two bits, implying distance two communication. Those paths for distinct pairs of elements

can be made edge-disjoint [3]. However, in a butterfly network emulation every dimension, except the most significant, is used twice. Hence, for the emulation of multiple butterfly networks each with one element per processor the emulation of different networks can only be initiated every other cycle. The contention for communication channels results in a doubling of the communication time compared to a binary encoding. Converting the binary-reflected Gray code encoding to binary code, and then performing the butterfly emulation also results in a time complexity proportional to $2\frac{P}{N}$ for $\frac{P}{N} \gg n$.

The communication in one of the two dimensions required for butterfly emulation on binary-reflected Gray coded data is performing code conversion. By using bi-sectioning the code conversion can be made in local memory, if there is at least four local elements. The element pairs required for a butterfly computation are local as well. With more than four elements per processor, the emulation for different sets of four elements can be pipelined. For two elements per processor both bi-sectioning and code conversion require communication under the assumption of a balanced load. But, these two communications can be performed concurrently by organizing the data motion such that every element is only communicated in one dimension. Every dimension is used twice, except the most significant dimension. In the case of a single data point per processor a data replication/reduction is performed in every emulation stage such that distance one communication suffices in every stage. (The total amount of communciation is twice that of the binary encoding). The final data encoding for any of the algorithms can be either binary, or binary-reflected Gray code. The control for all algorithms is derived from the local memory addresses, the processor address, and the butterfly stage being emulated. No tags are needed, and the control is completely distibuted.

The outline of this paper is as follows. Section two introduces some properties of the binary code and the binary-reflected Gray code. Section three presents the results for the case with processor addresses in a binary-reflected Gray code and local memory addresses in binary code. Section four treats the case with the entire address space in Gray code. Cyclic and consecutive allocation is considered in both sections three and four. The case with one data element per processor is discussed in section five. A summary and discussion is given in section six.

## 2 Preliminaries

A Boolean $n$-cube has $N = 2^n$ nodes, and every node has $n$ neighbors. The nodes model the processors and the edges model the communication channels of the network. $n$ address bits, or *dimensions*, are required for the *processor address field* and $p - n$ bits for the *local memory address field.* Each processor can concurrently communicate with all $n$ neighbors. The total machine address field is denoted $(a_{p-1}a_{p-2}\ldots a_n||a_{n-1}\ldots a_0)$, where $||$ denotes concatenation of two binary strings. We arbitrarily assume that the memory address field defines the high order bits of the machine address. Subcube $0_j$ contains all processors whose $j$th bit is 0. Subcube $1_j$ is similarly defined. The symbol $*$ denotes 0 or 1, and $\oplus$ the bit-wise *exclusive-or* operation. Bit 0 is the least significant bit. $(1^j)$ is a string of $j$ instances of a bit with value one, with the least significant bit being bit zero. The binary encoding of $i$ is $B(i) = (b_{n-1}b_{n-2}\ldots b_0)$ and the binary-reflected Gray code of $i$ is $G_n(i) = (g_{n-1}g_{n-2}\ldots g_0)$. When the encoding of $i$ is of no particular interest we write $i = (i_{n-1}i_{n-2}\ldots i_0)$. $|i| = \sum_{j=0}^{n-1} i_j$ is the number of bits in $i$ with value one. $i^{\underline{k}} = (i_{k-1}i_{k-2}\ldots i_0)$ is the integer defined by the $k$ least significant bits in the encoding of $i$. $i^{\overline{k}} = (i_{p-1}i_{p-2}\ldots i_{p-k})$ is the integer defined by the $k$ most significant bits in the encoding of $i$. $\mathcal{Z}_N = \{0, 1, \cdots, N-1\}$ is

| Integer | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Binary | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |
| Gray | 0000 | 0001 | 0011 | 0010 | 0110 | 0111 | 0101 | 0100 | 1100 | 1101 | 1111 | 1110 | 1010 | 1011 | 1001 | 1000 |

Figure 1: A binary-reflected Gray code and Binary code.

the set of natural numbers less than $N$. $\widehat{G_n}$ is the sequence of $n$-bit binary-reflected Gray codes for $\mathcal{Z}_N$, i.e., $\widehat{G_n} = (G_n(0), G_n(1), \cdots, G_n(2^n - 1))$.

A butterfly network on $P = 2^p$ rows has $p + 1$ columns. We assume that a node in row $i$ and column $j$ is connected to nodes in rows $i$ and $i \oplus 2^{p-j-1}$ in column $j + 1$ for $j = \{0, 1, \ldots, p - 1\}$. This order is used for computation of the Fast Fourier Transform with data in normal order.

**Definition 1** [14] The *binary-reflected Gray code* for all integers in $\mathcal{Z}_N$ is defined recursively.

$$\text{Let } \widehat{G_1} = (G_1(0), G_1(1)), \text{where } G_1(0) = 0, G_1(1) = 1.$$

and

$$\widehat{G_n} = \begin{pmatrix} G_n(0) \\ G_n(1) \\ \vdots \\ G_n(2^n - 2) \\ G_n(2^n - 1) \end{pmatrix}.$$

$$\text{Then} \quad \widehat{G_{n+1}} = \begin{pmatrix} 0||G_n(0) \\ 0||G_n(1) \\ \vdots \\ 0||G_n(2^n - 2) \\ 0||G_n(2^n - 1) \\ 1||G_n(2^n - 1) \\ 1||G_n(2^n - 2) \\ \vdots \\ 1||G_n(1) \\ 1||G_n(0) \end{pmatrix}, \quad \text{or alternatively,} \quad \widehat{G_{n+1}} = \begin{pmatrix} G_n(0)||0 \\ G_n(0)||1 \\ G_n(1)||1 \\ G_n(1)||0 \\ G_n(2)||0 \\ G_n(2)||1 \\ \vdots \\ G_n(2^n - 1)||1 \\ G_n(2^n - 1)||0 \end{pmatrix}.$$

In the following we always refer to the binary-reflected Gray code defined above. The inverse of $G_n(i)$ is $G_n^{-1}$, where $G_n^{-1}(G_n(i)) = i$. Figure 1 shows a four bit code.

**Corollary 1** *The value of the most significant bit is the same in binary code and a binary-reflected Gray code. The remaining bits in $G_n(i), i \in \mathcal{Z}_{\frac{N}{2}}$ are defined by $G_{n-1}(i)$. The remaining bits in $G_n(i), i \in \mathcal{Z}_N - \mathcal{Z}_{\frac{N}{2}}$ are defined by $G_{n-1}(\bar{i})$, where $\bar{i}$ is the bit-wise complement of $i$, or*

$$G_n((i_{n-1}i_{n-2}\cdots i_0)) = \begin{cases} i_{n-1}||G_{n-1}((i_{n-2}i_{n-3}\cdots i_0)), & \text{if } i_{n-1} = 0, \\ i_{n-1}||G_{n-1}((\bar{i}_{n-2}\bar{i}_{n-3}\cdots \bar{i}_0)), & \text{otherwise.} \end{cases}$$

*Conversely, processor* $(a_{n-1}a_{n-2}\ldots a_0)$ *encodes the integer* $i = G_n^{-1}((a_{n-1}a_{n-2}\ldots a_0))$, *where*

$$G_n^{-1}((a_{n-1}a_{n-2}\cdots a_0)) = \begin{cases} a_{n-1}||G_{n-1}^{-1}((a_{n-2}a_{n-3}\cdots a_0)), & \text{if } a_{n-1} = 0, \\ a_{n-1}||\overline{G_{n-1}^{-1}((a_{n-2}a_{n-3}\cdots a_0))}, & \text{otherwise.} \end{cases}$$

The proof of corollary 1 follows from Definition 1. Corollary 2 is also immediate.

**Corollary 2** *The integer encoded at Hamming distance one of* $G_n(i)$ *in dimension* $j$ *is* $G_n(i \oplus (1^{j+1}))$, *i.e.,* $G_n(i) \oplus 2^j = G_n(i \oplus (1^{j+1}))$.

With a binary-reflected Gray code encoding of $i$ the two inputs to a butterfly computation differ in precisely two bits, since $g_i = b_{i+1} \oplus b_i$.

**Lemma 1**

$$G_n(i) \oplus G_n(i \oplus 2^j) = \begin{cases} 2^j + 2^{j-1}, & \text{if } j > 0, \\ 1, & \text{if } j = 0. \end{cases}$$

The implication of lemma 1 is that with the emulation proceeding from the most significant dimension to the least significant dimension the $k$th butterfly stage requires communication in dimensions $p-k-1$ and $p-k-2$ for a $p$ bit code, and the first stage being stage zero.

**Lemma 2** *With the entire address space of* $p$ *bits encoded in a binary-reflected Gray code and a cyclic mapping to an* $n$*-cube,* $p > n$, *memory location* $m$ *of processor* $G_n(i^{\underline{n}})$ *contains the integer* $G_{p-n}(m)||i^{\underline{n}}$, *if* $|G_{p-n}(m)|$ *is even, and* $G_{p-n}(m)||\overline{i^{\underline{n}}}$ *if* $|G_{p-n}(m)|$ *is odd.*

The lemma follows from the construction of the binary-reflected Gray code [14]. The address computation for the allocation of the data element with index $i$ for the different data allocation and encoding schemes are summarized in Table 1.

| Allocation | Encoding | Memory address | Processor address |
|---|---|---|---|
| Consecutive | Gray | $G(i \bmod \frac{P}{N})$ if $\lvert G(\lfloor \frac{iN}{P} \rfloor)\rvert$ is even<br>$\overline{G(i \bmod \frac{P}{N})}$ if $\lvert G(\lfloor \frac{iN}{P} \rfloor)\rvert$ is odd | $G(\lfloor \frac{iN}{P} \rfloor)$ |
| | Gray\|\|Gray | $G(i \bmod \frac{P}{N})$ | $G(\lfloor \frac{iN}{P} \rfloor)$ |
| | Binary\|\|Gray | $B(i \bmod \frac{P}{N})$ | $G(\lfloor \frac{iN}{P} \rfloor)$ |
| | Binary | $B(i \bmod \frac{P}{N})$ | $B(\lfloor \frac{iN}{P} \rfloor)$ |
| Cyclic | Gray | $G(\lfloor \frac{i}{N} \rfloor)$ | $G(i \bmod N)$ if $\lvert G(\lfloor \frac{i}{N} \rfloor)\rvert$ is even<br>$\overline{G(i \bmod N)}$ if $\lvert G(\lfloor \frac{i}{N} \rfloor)\rvert$ is odd |
| | Gray\|\|Gray | $G(\lfloor \frac{i}{N} \rfloor)$ | $G(i \bmod N)$ |
| | Binary\|\|Gray | $B(\lfloor \frac{i}{N} \rfloor)$ | $G(i \bmod N)$ |
| | Binary | $B(\lfloor \frac{i}{N} \rfloor)$ | $B(i \bmod N)$ |

Table 1: Address of element $i$ in a data set of size $P = 2^p$ allocated to an $n$-cube, $n \leq p$.

All our algorithms combine code conversion with butterfly emulation. The conversion from a binary-reflected Gray code to binary code is defined by $b_i = (\sum_{j=i}^{n-1} g_j) \bmod 2$, and conversely,

| paddr | 0000 | 0001 | 0011 | 0010 | 0110 | 0111 | 0101 | 0100 | 1100 | 1101 | 1111 | 1110 | 1010 | 1011 | 1001 | 1000 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| integer | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| step 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |
| step 2 | 0 | 1 | 2 | 3 | 7 | 6 | 5 | 4 | 12 | 13 | 14 | 15 | 11 | 10 | 9 | 8 |
| step 3 | 0 | 1 | 3 | 2 | 6 | 7 | 5 | 4 | 12 | 13 | 15 | 14 | 10 | 11 | 9 | 8 |

Figure 2: Conversion of a binary-reflected Gray code to binary code.

$g_i = b_{i+1} \oplus b_i, 0 \leq i < n-1$, and $g_{n-1} = b_{n-1}$. Depending upon whether the emulation starts from the most or least significant bit, and what the final encoding shall be, one of the following four code conversion algorithms is used.

Converting binary-reflected Gray code to binary code

**for** $k := n-2$ **downto** 0 **do**
  **if** $a_{k+1} = 1$ **then**
msb→lsb  $(a_{n-1}a_{n-2}\ldots a_k \ldots a_0) \rightarrow (a_{n-1}a_{n-2}\ldots \overline{a_k} \ldots a_0)$
  **endif**
**enddo**

**for** $k := 0$ **to** $n-2$ **do**
  **if** $a_{n-1} \oplus a_{n-2} \oplus \cdots \oplus a_{k+1} = 1$ **then**
lsb→ msb  $(a_{n-1}a_{n-2}\ldots a_k \ldots a_0) \rightarrow (a_{n-1}a_{n-2}\ldots \overline{a_k} \ldots a_0)$
  **endif**
**enddo**

Converting binary code to binary-reflected Gray code

**for** $k := n-2$ **downto** 0 **do**
  **if** $a_{n-1} \oplus a_{n-2} \oplus \cdots \oplus a_{k+1} = 1$ **then**
msb→lsb  $(a_{n-1}a_{n-2}\ldots a_k \ldots a_0) \rightarrow (a_{n-1}a_{n-2}\ldots \overline{a_k} \ldots a_0)$
  **endif**
**enddo**

**for** $k := 0$ **to** $n-2$ **do**
  **if** $a_{k+1} = 1$ **then**
lsb→msb  $(a_{n-1}a_{n-2}\ldots a_k \ldots a_0) \rightarrow (a_{n-1}a_{n-2}\ldots \overline{a_k} \ldots a_0)$
  **endif**
**enddo**

The data motion for the first algorithm is depicted in Figure 2. Initially, index $i$ is assigned to processor $G_n(i)$. After conversion, processor $G_n(i)$ contains data with index $B_n^{-1}(G_n(i))$. Elements subject to an exchange operation are in boldface.

# 3 Binary coded memory addresses, Gray coded processor addresses.

## 3.1 Cyclic allocation

With the butterfly emulation starting from the most significant bit the first several steps are local with cyclic data allocation. Then, there are in effect a number of independent butterfly emulations, each with one data point per processor. The data organization for all emulations is the same. We first treat the case with at least four elements per processor, then the case with two elements per processor.

### 3.1.1 Four or more elements per processor

The algorithms below perform the conversion from binary-reflected Gray code to binary code in local memory. First an *all-to-all personalized communication* is performed in all 2-cubes defined by the two most significant processor dimenssions. It allows for the local emulation of the first butterfly stage. Then, each $(n-2)$-cube contains two independent sequences, each with two elements per processor. Each sequence of length $\frac{N}{2}$ is allocated within the $(n-2)$-cubes as if it was encoded in a binary-reflected Gray code, and the data allocation cyclic. By performing a bi-section on the appropriate memory locations and cube dimension each sequence is split into two sequences each with two elements per processor in cubes of one dimension less. The allocation of each new sequence within its subcube is similar to the allocation of the original sequence within the $(n-2)$-cube. Hence, for each communication step one additional butterfly stage can be emulated. The communications for different sequences can be pipelined. The main result is

**Theorem 1** *A butterfly emulation on $P = 2^p$ elements distributed cyclicly among $N = 2^n$ processors, $p \geq n+2$, with binary encoded memory addresses and binary-reflected Gray code encoded processor addresses can be performed in at most $\frac{P}{2N} + n - 2$ element transfers in sequence. The encoding of the processor addresses on completion can be either binary, or binary-reflected Gray code.*

Below we provide the insight and the algorithms for a proof of the theorem by induction. We represent the data allocation by the assignment of logic dimensions to machine dimensions. The full address space is considered for scheduling of communications, and for some details of the proof.

**Emulation by dimension exchange sequences**

The initial data assignment is:

$$\begin{array}{lccccccccccc} \text{Machine addr. field:} & (a_{p-1} & a_{p-2} & \cdots & a_n & \| & a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0) \\ & \uparrow & \uparrow & \uparrow & \uparrow & & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \text{Logic address field:} & (b_{p-1} & b_{p-2} & \cdots & b_n & \| & b_{n-1} & g_{n-2} & g_{n-3} & \cdots & g_0) \end{array}$$

where $b_i$ denotes a bit generated with a binary encoding and $g_i$ a bit generated by a binary-reflected Gray code. The most significant bit in a binary code and a binary-reflected Gray code are

the same, i.e. $g_{n-1} = b_{n-1}$. The first step in the emulation is to perform an *all-to-all personalized communication* [7], or four-sectioning in all 2-cubes defined by the two most significant processor dimensions. Performing this operation such that it is equivalent to the following two exchanges (refered to as $i$-cycles in [15])

**if** $a_{n+1} \oplus a_{n-1} = 1$ **then**
$(a_{p-1}a_{p-2}\ldots a_{n+1}a_n||a_{n-1}a_{n-2}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots \overline{a_{n+1}}a_n||\overline{a_{n-1}}a_{n-2}\ldots a_0)$
**endif**
**if** $a_n \oplus a_{n-2} = 1$ **then**
$(a_{p-1}a_{p-2}\ldots a_{n+1}a_n||a_{n-1}a_{n-2}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots a_{n+1}\overline{a_n}||a_{n-1}\overline{a_{n-2}}\ldots a_0)$
**endif**

yields the data allocation in the second row of Figure 3. With the two most significant dimensions of the data subject to butterfly emulation in local memory the first rank of the emulation can be carried out locally, and concurrently for all processors. For the next emulation step dimensions $n-2$ and $n-3$ are required by Lemma 1. By performing the exchange between dimensions $n+1$ and $n-3$ the two lowest order local memory dimensions are assigned $g_{n-3}$ and $g_{n-2}$, respectively. The next butterfly emulation step can be performed locally, and concurrently for all processors.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Initial | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_{n+1}$ | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| alloc. | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | $\ldots$ | $b_{n+1}$ | $b_n$ | $\|\|$ | $b_{n-1}$ | $g_{n-2}$ | $g_{n-3}$ | $\ldots$ | $g_0)$ |
| First | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_{n-1}$ | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| emul. | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| step | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | $\ldots$ | $b_{n-1}$ | $g_{n-2}$ | $\|\|$ | $b_{n+1}$ | $b_n$ | $g_{n-3}$ | $\ldots$ | $g_0)$ |
| 2nd | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_{n+1}$ | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| emul. | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| step | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | $\ldots$ | $g_{n-3}$ | $g_{n-2}$ | $\|\|$ | $b_{n+1}$ | $b_n$ | $b_{n-1}$ | $\ldots$ | $g_0)$ |

Figure 3: The dimension exchanges for the first two emulation steps.

The local memory strides are somewhat complicated. The stride becomes two for the first emulation stage, if $g_{n-2}$ is converted to binary code. The code conversion is accomplished by the local memory exchange

**if** $a_{n+1} = 1$ **then**
$(a_{p-1}a_{p-2}\ldots a_{n+1}a_n||a_{n-1}a_{n-2}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots a_{n+1}\overline{a_n}||a_{n-1}a_{n-2}\ldots a_0)$
**endif**

which implements the operation $b_{n-2} = b_{n-1} \oplus g_{n-2}$. At the next bi-section step $g_{n-3}$ is brought into local memory. Using the algorithm converting a binary-reflected Gray code to binary code starting from the most significant dimension, the required operation is $b_{n-3} = b_{n-2} \oplus g_{n-3}$, or

**if** $a_n = 1$ **then**
$(a_{p-1}a_{p-2}\ldots a_{n+1}a_n||a_{n-1}a_{n-2}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots \overline{a_{n+1}}a_n||a_{n-1}a_{n-2}\ldots a_0)$
**endif**

Note that the stride for the butterfly computations corresponding to dimension $n-2$ is one, since this dimension is assigned to the least significant local memory address. If the dimensions in local memory are changed to binary code before the butterfly computation, and then moved to the processor address field through exchange operations, then the processor address field will be encoded in binary code on completion. For a binary-reflected Gray code encoding on completion the encoding should be changed in memory before the dimensions are exchanged with processor dimensions. Using the apropriate code conversion algorithm the local exchange is controlled by the leading processor address bits:

**for** $k := n-2$ **downto** 0 **do**
  **if** $a_{n-1} \oplus a_{n-2} \oplus \cdots \oplus a_{k+1} = 1$ **then**
    **if** $n-2-k \bmod 2 = 0$ **then**
      $(a_{p-1}a_{p-2}\ldots a_{n+1}a_n||a_{n-1}a_{n-2}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots a_{n+1}\overline{a_n}||a_{n-1}a_{n-2}\ldots a_0)$
    **else**
      $(a_{p-1}a_{p-2}\ldots a_{n+1}a_n||a_{n-1}a_{n-2}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots \overline{a_{n+1}}a_n||a_{n-1}a_{n-2}\ldots a_0)$
  **endif**
**enddo**

### Detailed exchange schedules

The code conversion can be combined with the multi-sectioning [5]. Consider the first four memory locations in a 2-cube. In the algorithm

**forall** $(a_1a_0)$ **do**
  **if** $a_3 \oplus a_1||a_2 \oplus a_0 = 11$ **then**
    $(a_3a_2||a_1a_0) \rightarrow (\overline{a_3}a_2||\overline{a_1}a_0)$
  **if** $a_3 \oplus a_1||a_2 \oplus a_0 = 01$ **then**
    $(a_3a_2||a_1a_0) \rightarrow (a_3\overline{a_2}||a_1\overline{a_0})$
**endforall**
**forall** $(a_1a_0)$ **do**
  **if** $a_3 \oplus a_1||a_2 \oplus a_0 = 11$ **then**
    $(a_3a_2||a_1a_0) \rightarrow (a_3\overline{a_2}||a_1\overline{a_0})$
  **if** $a_3 \oplus a_1||a_2 \oplus a_0 = 10$ **then**
    $(a_3a_2||a_1a_0) \rightarrow (\overline{a_3}a_2||\overline{a_1}a_0)$
**endforall**

all processors perform exchanges in both dimensions of the 2-cube in each of the two **forall** statements. Figure 4 shows the data motion and final encoding for a 2-cube with memory addresses in binary code and processor addresses in binary-reflected Gray code. All four subcubes have the same encoding on termination. The final processor encoding is in binary code, and the memory encoding is in binary-reflected Gray code. The encoding of each axis is preserved during the transposition. The elements subject to exchange in a step of the algorithm is highlighted in boldface. The exchange dimension is indicated by a superscript. In each of the two steps every processor communicates in both of its inter-processor dimensions.

Changing the memory encoding to a binary encoding requires that a code conversion is performed on the least significant memory dimension. A local exchange is required, if the most

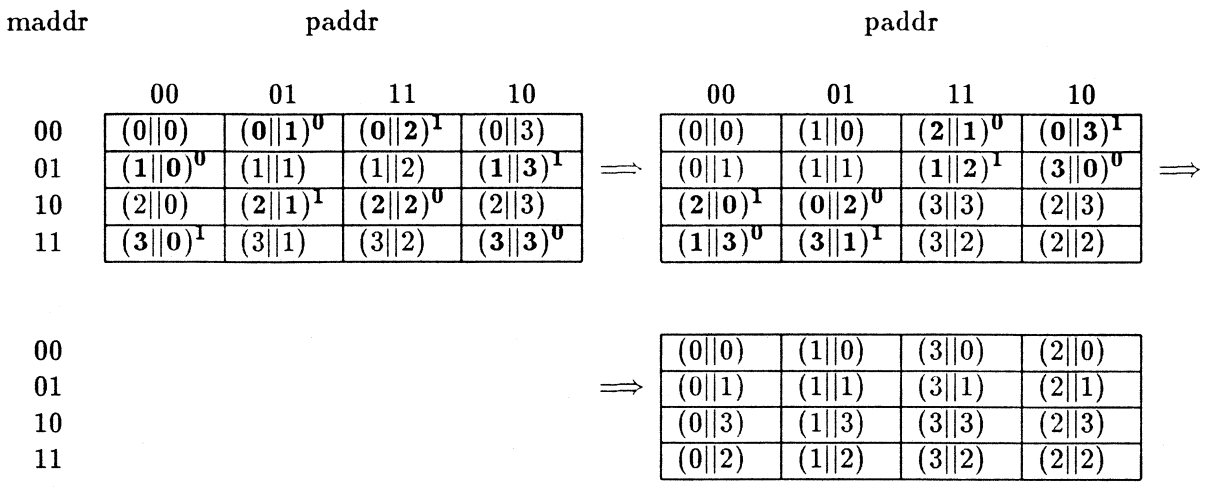


Figure 4: All-to-all personalized communication in 2-cubes without code conversion.

significant dimension is one. An exchange algorithm that changes the encoding of the axes such that the memory encoding is in binary code, and the processor encoding is in binary-reflected Gray code both *before* and *after* the transposition, is depicted in Figure 5. As in the previous algorithm each processor communicates in both of its dimensions in both cycles.

We now consider the data organization that results when the algorithms depicted in Figures 4 and 5 are applied to $n$-cubes. Memory location zero in the four processors with the same $n-2$ least significant bits stores data elements with the following four indices according to corollary 1.

| Processor | Index |
|---|---|
| $0_{n-1}0_{n-2}a_{n-3}a_{n-4}\ldots a_0$ | $G_2^{-1}(00)\|\|\underline{G_{n-2}^{-1}((a_{n-3}a_{n-4}\ldots a_0))} = \underline{G_{n-2}^{-1}((a_{n-3}a_{n-4}\ldots a_0))}$ |
| $0_{n-1}1_{n-2}a_{n-3}a_{n-4}\ldots a_0$ | $G_2^{-1}(01)\|\|\underline{G_{n-2}^{-1}((a_{n-3}a_{n-4}\ldots a_0))} = \frac{N}{4} + \underline{G_{n-2}^{-1}((a_{n-3}a_{n-4}\ldots a_0))}$ |
| $1_{n-1}1_{n-2}a_{n-3}a_{n-4}\ldots a_0$ | $G_2^{-1}(11)\|\|\underline{G_{n-2}^{-1}((a_{n-3}a_{n-4}\ldots a_0))} = \frac{2N}{4} + \underline{G_{n-2}^{-1}((a_{n-3}a_{n-4}\ldots a_0))}$ |
| $1_{n-1}0_{n-2}a_{n-3}a_{n-4}\ldots a_0$ | $G_2^{-1}(10)\|\|\underline{G_{n-2}^{-1}((a_{n-3}a_{n-4}\ldots a_0))} = \frac{3N}{4} + \underline{G_{n-2}^{-1}((a_{n-3}a_{n-4}\ldots a_0))}$ |

The rearrangement defined by the all-to-all personalized communication moves all four data elements into the same processor. The algorithm depicted in Figure 4 results in the data organization shown in Figure 6, and the algorithm in Figure 5 results in the organization in Figure 7, where $\underline{i^{n-2}} = G_{n-2}^{-1}(a_{n-3}a_{n-4}\ldots a_0)$. The encoding of the data after the all-to-all personalized communication allows for a butterfly emulation on the most significant memory dimension (originally assigned to processor dimension $n-1$), but not on the least significant memory dimension. After the emulation step on the most significant memory dimension, the remaining emulation consists of two independent emulations. The data set for each such emulation has two elements per processor in an $(n-2)$-cube, with binary-reflected Gray code encoding of data allocated cyclicly.

With the data allocation within each $(n-2)$-cube being the same after the all-to-all personalized communication, it suffices to consider subcube $0_{n-1}0_{n-2}$. Moreover, since the two most significant bits in the data index is independent of memory location we exclude them. The indices stored in the first four memory locations of the subcubes follows from corollary 1. They are shown in Figure 8, where $\underline{i^{n-3}} = G_{n-3}^{-1}(a_{n-4}a_{n-5}\ldots a_0)$. The memory encoding is in binary code. An exchange

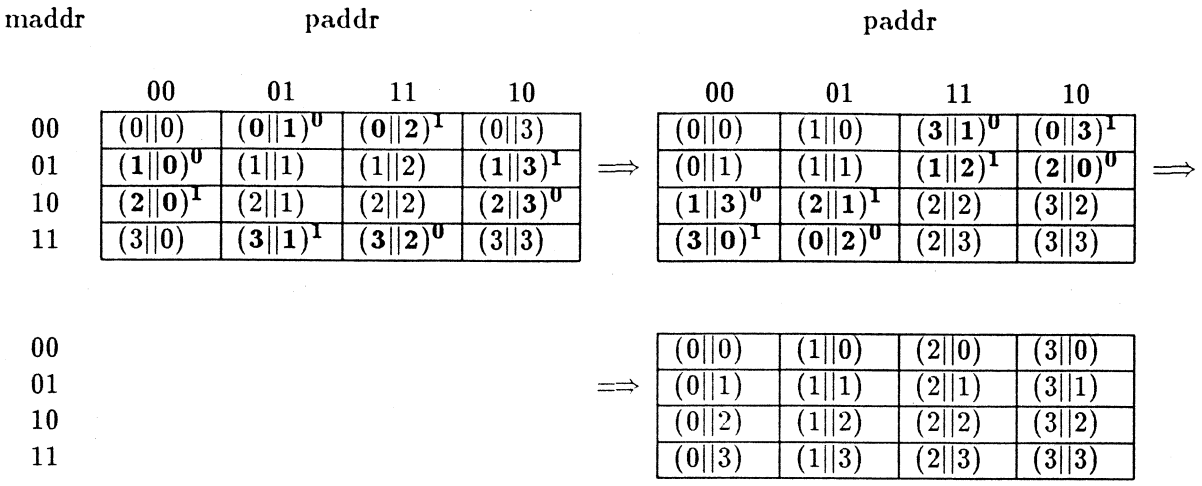


Figure 5: All-to-all personalized communication in 2-cubes with code conversion of both axes.

| maddr | paddr | | | |
|---|---|---|---|---|
| | $00a_{n-3}\ldots a_0$ | $01a_{n-3}\ldots a_0$ | $11a_{n-3}\ldots a_0$ | $10a_{n-3}\ldots a_0$ |
| 00 | $(0\|\|0\|\|i^{n-2})$ | $(1\|\|0\|\|i^{n-2})$ | $(3\|\|0\|\|i^{n-2})$ | $(2\|\|0\|\|i^{n-2})$ |
| 01 | $(0\|\|1\|\|i^{n-2})$ | $(1\|\|1\|\|i^{n-2})$ | $(3\|\|1\|\|i^{n-2})$ | $(2\|\|1\|\|i^{n-2})$ |
| 10 | $(0\|\|3\|\|i^{n-2})$ | $(1\|\|3\|\|i^{n-2})$ | $(3\|\|3\|\|i^{n-2})$ | $(2\|\|3\|\|i^{n-2})$ |
| 11 | $(0\|\|2\|\|i^{n-2})$ | $(1\|\|2\|\|i^{n-2})$ | $(3\|\|2\|\|i^{n-2})$ | $(2\|\|2\|\|i^{n-2})$ |

Figure 6: Data indices after all-to-all personalized communication in 2-cubes without code conversion.

| maddr | paddr | | | |
|---|---|---|---|---|
| | $00a_{n-3}\ldots a_0$ | $01a_{n-3}\ldots a_0$ | $11a_{n-3}\ldots a_0$ | $10a_{n-3}\ldots a_0$ |
| 00 | $(0\|\|0\|\|i^{n-2})$ | $(1\|\|0\|\|i^{n-2})$ | $(2\|\|0\|\|i^{n-2})$ | $(3\|\|0\|\|i^{n-2})$ |
| 01 | $(0\|\|1\|\|i^{n-2})$ | $(1\|\|1\|\|i^{n-2})$ | $(2\|\|1\|\|i^{n-2})$ | $(3\|\|1\|\|i^{n-2})$ |
| 10 | $(0\|\|2\|\|i^{n-2})$ | $(1\|\|2\|\|i^{n-2})$ | $(3\|\|2\|\|i^{n-2})$ | $(2\|\|2\|\|i^{n-2})$ |
| 11 | $(0\|\|3\|\|i^{n-2})$ | $(1\|\|3\|\|i^{n-2})$ | $(3\|\|3\|\|i^{n-2})$ | $(2\|\|3\|\|i^{n-2})$ |

Figure 7: Data indices after all-to-all personalized communication in 2-cubes with code conversion of both axes.

| maddr | paddr | |
|---|---|---|
| | $(0_{n-3}a_{n-4}\ldots a_0)$ | $(1_{n-3}a_{n-4}\ldots a_0)$ |
| 00 | $(00\|\|0\|\|i^{\underline{n-3}})$ | $(00\|\|1\|\|\overline{i^{\underline{n-3}}})$ |
| 01 | $(01\|\|1\|\|\overline{i^{\underline{n-3}}})$ | $(01\|\|0\|\|i^{\underline{n-3}})$ |
| 10 | $(10\|\|0\|\|i^{\underline{n-3}})$ | $(10\|\|1\|\|\overline{i^{\underline{n-3}}})$ |
| 11 | $(11\|\|1\|\|\overline{i^{\underline{n-3}}})$ | $(11\|\|0\|\|i^{\underline{n-3}})$ |

Figure 8: Data indices within $(n-2)$-cubes after all-to-all personalized communication in 2-cubes.

| maddr | paddr | |
|---|---|---|
| | $(0_{n-3}a_{n-4}\ldots a_0)$ | $(1_{n-3}a_{n-4}\ldots a_0)$ |
| 00 | $(00\|\|0\|\|i^{\underline{n-3}})$ | $(10\|\|0\|\|i^{\underline{n-3}})$ |
| 01 | $(01\|\|1\|\|i^{\underline{n-3}})$ | $(11\|\|1\|\|\overline{i^{\underline{n-3}}})$ |
| 10 | $(00\|\|1\|\|\overline{i^{\underline{n-3}}})$ | $(10\|\|1\|\|\overline{i^{\underline{n-3}}})$ |
| 11 | $(01\|\|0\|\|i^{\underline{n-3}})$ | $(11\|\|0\|\|i^{\underline{n-3}})$ |

Figure 9: Data indices within $(n-2)$-cubes after all-to-all personalized communication in 2-cubes, and an exchange in dimension $n-3$.

operation on the most significant memory dimension and processor dimension $n-3$ yields the index allocation of Figure 9. The leading dimension of the data index is equal to $a_{n-3}$ in both subcubes. By performing a cyclic shift on the last three memory locations the data allocation within all $(n-3)$-cubes defined by the three leading dimensions are as given in Figure 10 (the three leading dimensions in both the processor address and data index are omitted). An induction argument now follows easily. Two communications are required for the first two dimensions, and two for each subsequent dimension. Sets of four elements can be pipelined, with a new set initiated every two cycles. Some local moves can be avoided by combining the memory reordering with the exchange. This detail is omitted for simplicity.

The communication time of the algorithm can be improved by a factor of two by extending the pipelining to every sequence, not only to pairs of sequences. After the first exchange in a dimension, $k$, a processor contains three half length subsequences of one sequence, and one half subsequence of the other sequence. The latter is yet to be communicated in dimension $k$. Each processor can perform a butterfly emulation step on half a sequence. Since each emulation step recursively splits a sequence into two new sequences, the butterfly computation after communication in dimension $k$ creates one half of two new sequences. An exchange can be performed in dimension $k-1$ for these two new sequences concurrently with the remaining exchange in dimension $k$. The first few steps are illustrated in Figure 11. The butterfly stage being emulated is indicated by a subscript, and the dimension of exchange by a superscript.

With the pipeling of individual sequences Theorem 1 follows.

| maddr | paddr |
|---|---|
| | $(*a_{n-4}\ldots a_0)$ |
| 00 | $(00\|\|i^{\overleftarrow{n-3}})$ |
| 01 | $(01\|\|i^{\overleftarrow{n-3}})$ |
| 10 | $(10\|\|i^{\overleftarrow{n-3}})$ |
| 11 | $(11\|\|i^{\overleftarrow{n-3}})$ |

Figure 10: Data indices within $(n-3)$-cubes after all-to-all personalized communication in 2-cubes, exchange in dimension $n-3$ and memory reordering.

| maddr | paddr | | | |
|---|---|---|---|---|
| | $(0_{n-3}0_{n-4}a_{n-5}\ldots a_0)$ | $(0_{n-3}1_{n-4}a_{n-5}\ldots a_0)$ | $(1_{n-3}1_{n-4}a_{n-5}\ldots a_0)$ | $(1_{n-3}0_{n-4}a_{n-5}\ldots a_0)$ |
| 00 | $(00\|\|00\|\|i^{\overleftarrow{n-4}})$ | $(00\|\|01\|\|i^{\overleftarrow{n-4}})$ | $(00\|\|10\|\|i^{\overleftarrow{n-4}})^{n-3}$ | $(00\|\|11\|\|i^{\overleftarrow{n-4}})^{n-3}$ |
| 01 | $(01\|\|11\|\|i^{\overleftarrow{n-4}})$ | $(01\|\|10\|\|i^{\overleftarrow{n-4}})$ | $(01\|\|01\|\|i^{\overleftarrow{n-4}})$ | $(01\|\|00\|\|i^{\overleftarrow{n-4}})$ |
| 10 | $(10\|\|00\|\|i^{\overleftarrow{n-4}})^{n-3}$ | $(10\|\|01\|\|i^{\overleftarrow{n-4}})^{n-3}$ | $(10\|\|10\|\|i^{\overleftarrow{n-4}})$ | $(10\|\|11\|\|i^{\overleftarrow{n-4}})$ |
| 11 | $(11\|\|11\|\|i^{\overleftarrow{n-4}})$ | $(11\|\|10\|\|i^{\overleftarrow{n-4}})$ | $(11\|\|01\|\|i^{\overleftarrow{n-4}})$ | $(11\|\|00\|\|i^{\overleftarrow{n-4}})$ |
| | | | $\Downarrow$ | |
| 00 | $(00\|\|00\|\|i^{\overleftarrow{n-4}})$ | $(00\|\|01\|\|i^{\overleftarrow{n-4}})$ | $(10\|\|01\|\|i^{\overleftarrow{n-4}})^{n-4}_{n-2}$ | $(10\|\|00\|\|i^{\overleftarrow{n-4}})_{n-2}$ |
| 01 | $(01\|\|11\|\|i^{\overleftarrow{n-4}})^{n-4}_{n-2}$ | $(01\|\|10\|\|i^{\overleftarrow{n-4}})_{n-2}$ | $(01\|\|01\|\|i^{\overleftarrow{n-4}})^{n-3}$ | $(01\|\|00\|\|i^{\overleftarrow{n-4}})^{n-3}$ |
| 10 | $(00\|\|11\|\|i^{\overleftarrow{n-4}})_{n-2}$ | $(00\|\|10\|\|i^{\overleftarrow{n-4}})^{n-4}_{n-2}$ | $(10\|\|10\|\|i^{\overleftarrow{n-4}})$ | $(10\|\|11\|\|i^{\overleftarrow{n-4}})$ |
| 11 | $(11\|\|11\|\|i^{\overleftarrow{n-4}})^{n-3}$ | $(11\|\|10\|\|i^{\overleftarrow{n-4}})^{n-3}$ | $(11\|\|01\|\|i^{\overleftarrow{n-4}})_{n-2}$ | $(11\|\|00\|\|i^{\overleftarrow{n-4}})^{n-4}_{n-2}$ |
| | | | $\Downarrow$ | |
| 00 | $(00\|\|00\|\|i^{\overleftarrow{n-4}})_{n-2,n-3}$ | $(00\|\|01\|\|i^{\overleftarrow{n-4}})^{n-4}_{n-2}$ | $(11\|\|00\|\|i^{\overleftarrow{n-4}})_{n-3}$ | $(10\|\|00\|\|i^{\overleftarrow{n-4}})$ |
| 01 | $(00\|\|10\|\|i^{\overleftarrow{n-4}})_{n-3}$ | $(01\|\|10\|\|i^{\overleftarrow{n-4}})$ | $(11\|\|10\|\|i^{\overleftarrow{n-4}})_{n-2,n-3}$ | $(11\|\|11\|\|i^{\overleftarrow{n-4}})^{n-4}_{n-2}$ |
| 10 | $(00\|\|11\|\|i^{\overleftarrow{n-4}})$ | $(01\|\|11\|\|i^{\overleftarrow{n-4}})_{n-3}$ | $(10\|\|10\|\|i^{\overleftarrow{n-4}})^{n-4}_{n-2}$ | $(10\|\|11\|\|i^{\overleftarrow{n-4}})_{n-2,n-3}$ |
| 11 | $(01\|\|00\|\|i^{\overleftarrow{n-4}})^{n-4}_{n-2}$ | $(01\|\|01\|\|i^{\overleftarrow{n-4}})_{n-2,n-3}$ | $(11\|\|01\|\|i^{\overleftarrow{n-4}})$ | $(10\|\|01\|\|i^{\overleftarrow{n-4}})_{n-3}$ |

Figure 11: Data indices within $(n-2)$-cubes after all-to-all personalized communication in 2-cubes.

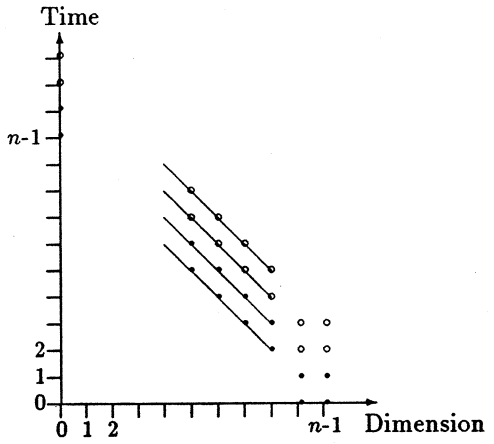


Figure 12: Scheduling of communications for bi-sectioning with local code conversion.

**Remark:** After the initial all-to-all personalized communication a communication in dimension $n-3$ for half of the local data converts the memory encoding from Gray code to binary code, and an additional butterfly stage can be emulated. Then, there are four independent emulation problems each with the same data encoding as in the original problem. However, the communication complexity of such an algorithm is $\frac{P}{N} + 4\lceil\frac{n}{2}\rceil - 6$, i.e., twice that of the algorithm above. Four cycles are required for every pair of dimensions and set of four elements.

### 3.1.2 Two or more elements per processor

The algorithms described in this section consistently operate on two independent sequences allocated to cubes of successively decreasing dimension. The algorithms rely on bi-section in one cube dimension concurrent with code conversion in the next lower cube dimension. Half of the data for one of the two butterfly emulations is communicated in one of the two required dimensions, and the other half of the data for the same butterfly emulation in the other cube dimension. The data for a butterfly stage meet in a processor adjacent to the two initially holding the data. In every stage each data set that interact in the remaining emulation is allocated to a subcube defined by the processor dimensions on which bi-section have been performed. Figures 13 and 14 show the data motion for a 3-cube with two data points per processor. The initial data allocation is cyclic and the encoding is by a binary-reflected Gray code. The output ordering is by binary code, and binary-reflected Gray code respectively. The final allocation is consecutive. The communication channels in every 2-cube are fully utilized, and the load is balanced. After one communication step butterfly computations for both emulations can be performed. The main result is

**Theorem 2** *The emulation of butterfly networks on $P = 2^p$ elements allocated cyclicly to $N = 2^n$ processors, $p \geq n+1$, with binary memory encoding and binary-reflected Gray code encoding of the processor address field can be performed in at most $\frac{P}{N} + n - 2$ element transfers in sequence. The processor encoding on output may be either binary or binary-reflected Gray code.*

Below we provide the arguments for a proof by induction. The algorithms described below can be used for more than two elements per processor, for instance by pipelining successive pairs. However, for four or more elements the previous algorithms yield a lower time complexity by approximately a factor of two.

**Binary coded processor addresses on termination.**

The address space and its encoding on input is

$$\begin{array}{lccccccccccc} \text{Machine addr. field:} & (a_{p-1} & a_{p-2} & \ldots & a_n & \| & a_{n-1} & a_{n-2} & a_{n-3} & \ldots & a_0) \\ & \uparrow & \uparrow & \uparrow & \uparrow & & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \text{Logic address field:} & (b_{p-1} & b_{p-2} & \ldots & b_n & \| & b_{n-1} & g_{n-2} & g_{n-3} & \ldots & g_0) \end{array}$$

For a binary processor encoding and a binary local memory encoding the following algorithm can be used. It uses the least significant local memory dimension for all bi-sections.

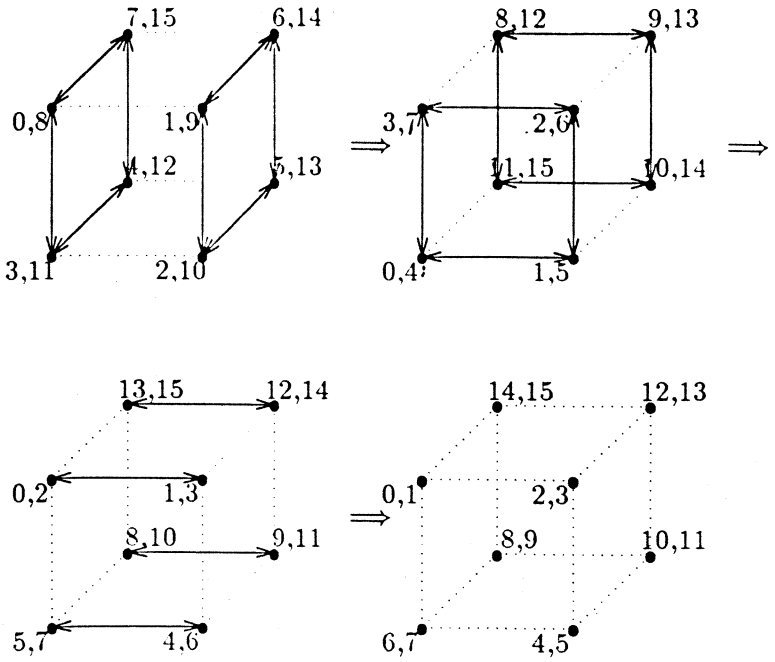


Figure 13: Bi-sectioning with inter-processor code conversion for cyclic data allocation, binary coded memory addresses and binary-reflected Gray code encoded processor addresses. The processor addresses are encoded in a binary-reflected Gray code on completion.

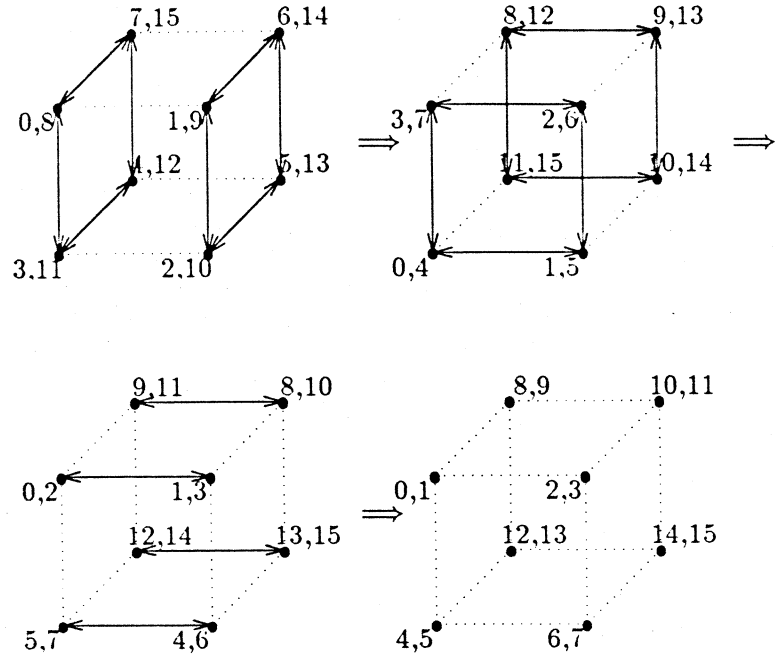


Figure 14: Bi-sectioning with inter-processor code conversion for cyclic data allocation, binary coded memory addresses, and binary-reflected Gray code encoded processor addresses. The processor addresses are in binary code on completion.

| maddr | paddr | | | |
|---|---|---|---|---|
| | $(00a_{n-3}a_{n-4}\ldots a_0)$ | $(01a_{n-3}a_{n-4}\ldots a_0)$ | $(11a_{n-3}a_{n-4}\ldots a_0)$ | $(10a_{n-3}a_{n-4}\ldots a_0)$ |
| 0 | $(0\|\|00\|\|i^{\underline{n-2}})$ | $(0\|\|01\|\|\overline{i^{\underline{n-2}}})$ | $(0\|\|10\|\|i^{\underline{n-2}})$ | $(0\|\|11\|\|\overline{i^{\underline{n-2}}})$ |
| 1 | $(1\|\|00\|\|i^{\underline{n-2}})$ | $(1\|\|01\|\|\overline{i^{\underline{n-2}}})$ | $(1\|\|10\|\|i^{\underline{n-2}})$ | $(1\|\|11\|\|\overline{i^{\underline{n-2}}})$ |
| | | $\Downarrow$ | | |
| 0 | $(0\|\|01\|\|\overline{i^{\underline{n-2}}})$ | $(0\|\|00\|\|i^{\underline{n-2}})$ | $(1\|\|01\|\|\overline{i^{\underline{n-2}}})$ | $(1\|\|00\|\|i^{\underline{n-2}})$ |
| 1 | $(0\|\|11\|\|\overline{i^{\underline{n-2}}})$ | $(0\|\|10\|\|i^{\underline{n-2}})$ | $(1\|\|11\|\|\overline{i^{\underline{n-2}}})$ | $(1\|\|10\|\|i^{\underline{n-2}})$ |

Figure 15: Data allocation *before* and *after* bi-section and inter-processor communication for code conversion.

**for** $k = n-1$ **downto** 0 **do**
  **if** $a_n \oplus a_k = 1$ **then**
    $(a_{p-1}a_{p-2}\ldots a_n||a_{n-1}a_{n-2}\ldots a_k\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots \overline{a_n}||a_{n-1}a_{n-2}\ldots \overline{a_k}\ldots a_0)$
  **else**
    $(a_{p-1}a_{p-2}\ldots a_n||a_{n-1}a_{n-2}\ldots a_{k-1}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots a_n||a_{n-1}a_{n-2}\ldots \overline{a_{k-1}}\ldots a_0)$
  **endif**
  **if** $\overline{a_{n-1}} \oplus \overline{a_{n-2}} \oplus \ldots \oplus \overline{a_{k+1}} = 1$ **then**
    $(a_{p-1}a_{p-2}\ldots a_n||a_{n-1}a_{n-2}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots \overline{a_n}||a_{n-1}a_{n-2}\ldots a_0)$
  **endif**
**endfor**

The first assignment statement performs the bi-section. The second assignment statement performs the inter-processor communication required for code conversion. The last assignment performs a local memory reordering when required. The local memory moves for reordering can be avoided by writing the result of a butterfly computation back in inverted order, when a move would otherwise be required. The local memory reordering assures that the memory dimension used in the next exchange is in binary order. With the memory dimension used for the bi-sectioning always being in binary code, the final processor encoding is binary.

To prove the correctness of the algorithm we begin by considering the first iteration. In the first assignment dimensions $n$ and $n-1$ are exchanged. Both dimensions are in binary code. The second assignment converts processor dimension $n-2$ from binary-reflected Gray code to binary code in subcube $1_{n-1}$ for the data set in odd memory locations. The same assignment statement causes dimension $n-2$ of the data set in even memory locations in subcube $0_{n-1}$ to be complemented as well. The data allocation before and after the first **if** statement in the first iteration of the algorithm is given in Figure 15, where $i^{\underline{n-2}} = G_{n-2}^{-1}((a_{n-3}a_{n-4}\ldots a_0))$. The data allocation follows from corollaries 1 and 2. After the first iteration sequences originally in even memory locations are allocated to subcube $0_{n-1}$, and sequences in odd locations are allocated to subcube $1_{n-1}$. Furthermore, within each subcube the first half of a sequence is in an even memory location, and the second half in the next odd memory location. With respect to the subsequent emulation steps each such half is an independent sequence. The data index assigned to processor $(a_{n-2}a_{n-3}\ldots a_0)$ in subcube $1_{n-1}$ is $i^{\underline{n-1}} = G_{n-1}^{-1}((a_{n-2}a_{n-3}\ldots a_0))$. The same processor within subcube $0_{n-1}$ is

| maddr | paddr | | | |
|---|---|---|---|---|
| | $(00a_{n-3}a_{n-4}\ldots a_0)$ | $(01a_{n-3}a_{n-4}\ldots a_0)$ | $(11a_{n-3}a_{n-4}\ldots a_0)$ | $(10a_{n-3}a_{n-4}\ldots a_0)$ |
| 0 | $(0\|\|11\|\|\overline{i^{n-2}})$ | $(0\|\|10\|\|i^{n-2})$ | $(0\|\|01\|\|\overline{i^{n-2}})$ | $(0\|\|00\|\|i^{n-2})$ |
| 1 | $(1\|\|11\|\|i^{n-2})$ | $(1\|\|10\|\|\overline{i^{n-2}})$ | $(1\|\|01\|\|i^{n-2})$ | $(1\|\|00\|\|\overline{i^{n-2}})$ |
| | | $\Downarrow$ | | |
| 0 | $(0\|\|10\|\|i^{n-2})$ | $(0\|\|11\|\|\overline{i^{n-2}})$ | $(1\|\|10\|\|i^{n-2})$ | $(1\|\|11\|\|\overline{i^{n-2}})$ |
| 1 | $(0\|\|00\|\|\overline{i^{n-2}})$ | $(0\|\|01\|\|i^{n-2})$ | $(1\|\|00\|\|\overline{i^{n-2}})$ | $(1\|\|01\|\|i^{n-2})$ |
| | | $\Downarrow$ | | |
| 0 | $(0\|\|00\|\|i^{n-2})$ | $(0\|\|01\|\|\overline{i^{n-2}})$ | $(1\|\|00\|\|i^{n-2})$ | $(1\|\|01\|\|\overline{i^{n-2}})$ |
| 1 | $(0\|\|10\|\|\overline{i^{n-2}})$ | $(0\|\|11\|\|i^{n-2})$ | $(1\|\|10\|\|\overline{i^{n-2}})$ | $(1\|\|11\|\|i^{n-2})$ |

Figure 16: Data allocation *before* and *after* bi-section and inter-processor communication for code conversion. Reflected sequences.

assigned index $\overline{i^{n-1}}$. Clearly, in subcube $1_{n-1}$ the data allocation after the first step is cyclic, the memory address encoding binary and the processor address encoding a binary-reflected Gray code. In this subcube the remaining emulation problem is identical to the original problem, except it is half the size.

Now, consider an emulation on two sequences allocated in reverse order in an $n$-cube. The initial data allocation, the allocation after the first execution of the first **if** statement, and the allocation after the first execution of the second **if** statement are given in Figure 16. After the first **if** statement the first subsequence is allocated to subcube $0_{n-1}$ and the second to subcube $1_{n-1}$. However, the first half of each sequence is in the second memory location and the second half in the first memory location. The second **if** statement reorders the local memories such that the first half is in the first memory location, and the second half in the second location. Subcube $0_{n-1}$ encodes $i^{n-1}$, and subcube $1_{n-1}$ encodes $\overline{i^{n-1}}$.

The number of reflections required for the conversion of bit $k$ from a binary-reflected Gray code to binary code is $|g_{n-1}g_{n-2}\ldots g_{k+1}|$. But, in the butterfly emulation algorithm a reflection is performed in all dimensions, except the most significant. Hence, the number of excessive reflections is $|\overline{g_{n-1}g_{n-2}}\ldots\overline{g_{k+1}}|$. If the excessive number of reflections is even, then the encoding is indeed binary, but if it is odd the encoding is inverted.

The first two emulation steps are shown in Figure 17. The second exchange in a butterfly emulation step does not affect the local memory encoding, only the processor encoding. Local memory reordering is required in subcube $0_{n-1}$ in order to convert the dimension used for bi-section to binary code. In subcube $1_{n-1}$ the memory encoding is binary, and no change of the encoding is required. Since $g_{n-2} = b_{n-2}$ if $b_{n-1} = 0$ the data encoding of the memory is binary after the two first emulation steps, including the memory reordering. The next butterfly emulation step moves $b_{n-2}$ into the processor address field. This dimension is correctly encoded for all processors. The communication in dimension $n-3$ converts the binary-reflected Gray code to binary code in

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Initial alloc. | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | ... | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | ... | $a_0)$ |
| | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | ... | $b_n$ | $\|\|$ | $b_{n-1}$ | $g_{n-2}$ | $g_{n-3}$ | ... | $g_0)$ |
| First | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | ... | $a_n$ | $\|\|$ | 0 | $a_{n-2}$ | $a_{n-3}$ | ... | $a_0)$ |
| | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| emul. | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | ... | $b_{n-1}$ | $\|\|$ | $b_n$ | $\overline{g_{n-2}}$ | $g_{n-3}$ | ... | $g_0)$ |
| | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | ... | $a_n$ | $\|\|$ | 1 | $a_{n-2}$ | $a_{n-3}$ | ... | $a_0)$ |
| step | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | ... | $b_{n-1}$ | $\|\|$ | $b_n$ | $b_{n-2}$ | $g_{n-3}$ | ... | $g_0)$ |
| First | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | ... | $a_n$ | $\|\|$ | 0 | $a_{n-2}$ | $a_{n-3}$ | ... | $a_0)$ |
| exch. | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| in | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | ... | $\overline{g_{n-2}}$ | $\|\|$ | $b_n$ | $b_{n-1}$ | $g_{n-3}$ | ... | $g_0)$ |
| 2nd | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | ... | $a_n$ | $\|\|$ | 1 | $a_{n-2}$ | $a_{n-3}$ | ... | $a_0)$ |
| emul. | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| step | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | ... | $b_{n-2}$ | $\|\|$ | $b_n$ | $b_{n-1}$ | $g_{n-3}$ | ... | $g_0)$ |
| 2nd | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | ... | $a_n$ | $\|\|$ | 0 | $a_{n-2}$ | $a_{n-3}$ | ... | $a_0)$ |
| | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| emul. | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | ... | $\overline{g_{n-2}}$ | $\|\|$ | $b_n$ | $b_{n-1}$ | $\overline{g_{n-3}}$ | ... | $g_0)$ |
| | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | ... | $a_n$ | $\|\|$ | 1 | $a_{n-2}$ | $a_{n-3}$ | ... | $a_0)$ |
| step | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | ... | $b_{n-2}$ | $\|\|$ | $b_n$ | $b_{n-1}$ | $\overline{g_{n-3}}$ | ... | $g_0)$ |
| Local | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | ... | $a_n$ | $\|\|$ | 0 | $a_{n-2}$ | $a_{n-3}$ | ... | $a_0)$ |
| mem. | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| reord. | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | ... | $g_{n-2}$ | $\|\|$ | $b_n$ | $b_{n-1}$ | $\overline{g_{n-3}}$ | ... | $g_0)$ |
| Before | Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | ... | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | ... | $a_0)$ |
| 3rd | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| step | Logic addr. field: | $(b_{p-1}$ | $b_{p-2}$ | ... | $b_{n-2}$ | $\|\|$ | $b_n$ | $b_{n-1}$ | $\overline{g_{n-3}}$ | ... | $g_0)$ |

Figure 17: The two first emulation steps for bi-sectioning with inter-processor code conversion. Binary output order.

subcube $1_{n-1}1_{n-2}$, i.e., in this subcube $\overline{g_{n-3}} = b_{n-3}$. In subcube $1_{n-1}0_{n-2}$ the complementation should not take place with respect to code conversion, and hence $\overline{g_{n-3}} \neq b_{n-3}$. In subcube $0_{n-1}$ the ordering is inverted with respect to dimension $n-2$ from the first emulation step. To restore the ordering $\overline{a_{n-1}}$ should be used for the control of the conversion of the next dimension. Hence, in subcube $0_{n-1}0_{n-2}$ $\overline{g_{n-3}} = b_{n-3}$, but in subcube $0_{n-1}1_{n-2}$ $\overline{g_{n-3}} \neq b_{n-3}$.

**Binary-reflected Gray code encoding of processor addresses on termination.**

The local memory reordering in the algorithm above ensures that the memory dimension used for the exchange is in binary code. By instead reordering the memory such that the dimension is encoded in a binary-reflected Gray code the processor encoding on completion is in a binary-reflected Gray code. A sample pseudo code is given below. On termination processor dimension $k$ is assigned logic dimension $k+1$, $k \in \{0, 1, \ldots, n-1\}$. By using the least significant memory dimension for all exchanges the assignment of logic dimensions to memory dimensions is the same as initially, except the least significant dimension $(n)$ is assigned logic dimension zero.

**for** $k = n-1$ **downto** 0 **do**
  **if** $a_n \oplus a_k = 1$ **then**
    $(a_{p-1}a_{p-2}\ldots a_n||a_{n-1}a_{n-2}\ldots a_k\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots \overline{a_n}||a_{n-1}a_{n-2}\ldots \overline{a_k}\ldots a_0)$

**else**

$(a_{p-1}a_{p-2}\ldots a_n||a_{n-1}a_{n-2}\ldots a_{k-1}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots a_n||a_{n-1}a_{n-2}\ldots \overline{a_{k-1}}\ldots a_0)$

**endif**

**if** $\underbrace{1 \oplus 1 \oplus \ldots \oplus 1}_{n-k-2} \oplus a_k = 1$ **then**

$(a_{p-1}a_{p-2}\ldots a_n||a_{n-1}a_{n-2}\ldots a_0) \rightarrow (a_{p-1}a_{p-2}\ldots \overline{a_n}||a_{n-1}a_{n-2}\ldots a_0)$

**endif**

**endfor**

Note that the only difference between the two algorithms for binary encoded or binary-reflected Gray code encoded output is in the local memory reordering. To produce a binary-reflected Gray code encoding of the processor address field the dimension that is moved into the processor dimension $k$ shall be inverted if $\alpha = g_{n-1} \oplus g_{n-2} \oplus \ldots \oplus g_{k+1} = 1$. The memory dimension used for the exchange is inverted if $\beta = \overline{g_{n-1}} \oplus \overline{g_{n-2}} \oplus \ldots \oplus \overline{g_{k+2}} = 1$, since the dimension in memory is processor dimension $k+1$. Local memory reordering is required if $\alpha = 1$ and $\beta = 0$, or $\alpha = 0$ and $\beta = 1$ which reduces to $\underbrace{1 \oplus 1 \oplus \ldots \oplus 1}_{n-k-2} \oplus a_k = 1$. For instance, a memory reordering is performed in subcube $1_{n-1}$ before the first exchange of the second butterfly emulation step. The first emulation step is independent of output order, and the result is

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_n$ | $\|\|$ | $0$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| Logic address field: | $(b_{p-1}$ | $b_{p-2}$ | $\ldots$ | $b_{n-1}$ | $\|\|$ | $b_n$ | $\overline{g_{n-2}}$ | $g_{n-3}$ | $\ldots$ | $g_0)$ |

and

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Machine addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_n$ | $\|\|$ | $1$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| Logic address field: | $(b_{p-1}$ | $b_{p-2}$ | $\ldots$ | $b_{n-1}$ | $\|\|$ | $b_n$ | $b_{n-2}$ | $g_{n-3}$ | $\ldots$ | $g_0)$ |

But, since if $b_{n-1} = 1$ $g_{n-2} = \overline{b_{n-2}}$ a memory reordring is required in subcube $1_{n-1}$.

With all partitioning steps using the same local dimension the result on completion is an unshuffle on this storage dimension and the processor dimensions. By using successively lower storage dimensions starting from the most significant dimension, and restarting from the most significant dimension cyclicly, the $n$ most significant logic dimensions will be assigned to the processor address field. The final data allocation is consecutive, with the possible exception that the local memory may need to be reordered.

## 3.2 Consecutive mapping

**Theorem 3** *The emulation of a butterfly network on $P = 2^p$ elements allocated consecutively and evenly to $N = 2^n$ processors, $p \geq n+1$, with memory addresses in binary code and processor addresses in binary-reflected Gray code can be performed in at most $\frac{P}{2N} + \max(n, \frac{P}{2N})$ element transfers using a bi-sectioning algorithm with local code conversion. The required number of element transfers in sequence for the bi-sectioning algorithm with code conversion through inter-processor communication is $\frac{P}{2N} + \max(n, \frac{P}{N})$. The encoding of the processor address field on output can be either binary or binary-reflected Gray code.*

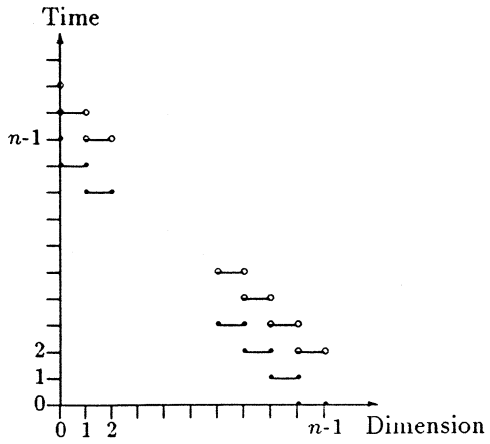


Figure 18: Scheduling of communications for the bi-sectioning algorithm with inter-processor code conversion.

The communication complexity for consecutive allocation, binary encoded memory addresses and binary-reflected Gray code encoding of the processor address field may be as much as twice that of cyclic allocation.

With a consecutive mapping and a binary-reflected Gray code encoding of the processor addresses and binary encoding of local memory addresses the initial data allocation is

$$\begin{array}{lccccccccccc} \text{Machine addr. field:} & (a_{p-1} & a_{p-2} & a_{p-3} & \cdots & a_n & \| & a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0) \\ & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \text{Logic address field:} & (b_{p-n-1} & b_{p-n-2} & b_{p-n-3} & \cdots & b_0 & \| & b_{p-1} & g_{p-2} & g_{p-3} & \cdots & g_{p-n}) \end{array}$$

If there is at least four data points per processor and the two most significant local memory dimensions are used for the first two exchange operations, then the data allocation becomes

$$\begin{array}{lccccccccccc} \text{Machine addr. field:} & (a_{p-1} & a_{p-2} & a_{p-3} & \cdots & a_n & \| & a_{n-1} & a_{n-2} & a_{n-3} & \cdots & a_0) \\ & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ \text{Logic address field:} & (b_{p-1} & g_{p-2} & b_{p-n-3} & \cdots & b_0 & \| & b_{p-n-1} & b_{p-n-2} & g_{p-3} & \cdots & g_{p-n}) \end{array}$$

The first $n$ emulation steps are identical to the last $n$ steps for cyclic data allocation. Emulation steps $n+1$ and $n+2$ require communication as well, since the first all-to-all personalized communication moved these dimensions into the processor address field. The two most significant processor dimensions are used twice in the algorithm for four or more data elements per processor. After these emulation steps all subsequent steps are local. If some memory dimensions other than the two most significant are used for the first two exchange steps, then there may be some local emulation steps before the second communication in the two most significant processor dimensions.

By moving only one processor dimension to local memory the algorithm for two data points per processor can be used in a manner analogous to the case for cyclic data allocation. The most significant processor dimension is used twice.

The scheduling of processor dimensions for local and inter-processor code conversion is shown in Figure 19.

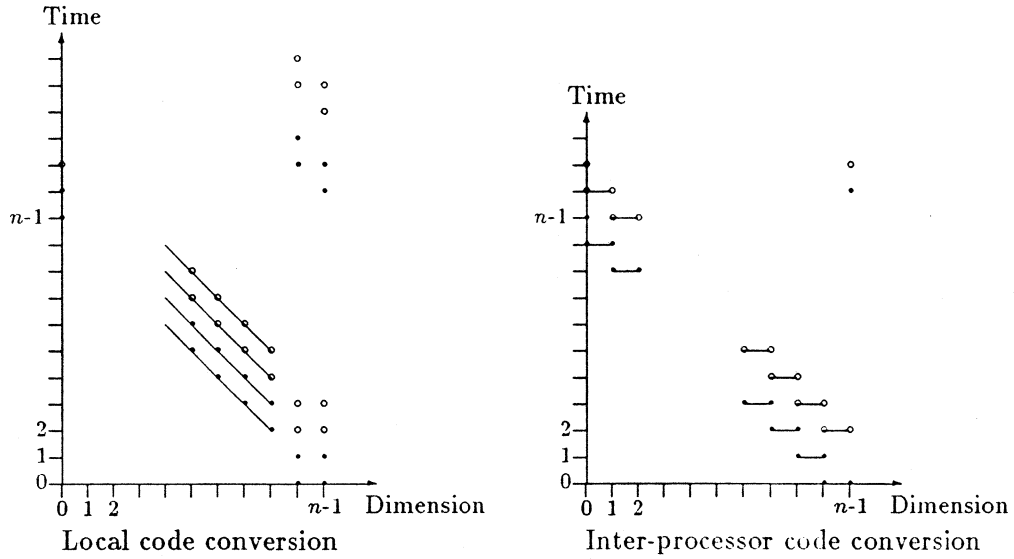


Figure 19: Scheduling of communications for bi-sectioning, consecutive allocation.

# 4 The entire address space in Gray code

## 4.1 Cyclic mapping

**Theorem 4** *The emulation of butterfly networks on $P = 2^p$ elements allocated cyclicly to $N = 2^n$ processors, $p \geq n + 1$, with the entire address space encoded in a binary-reflected Gray code can be carried out in at most $\frac{P}{2N} + n - 1$ element transfers in sequence by using the bi-sectioning algorithm with local code conversion. Bi-sectioning with inter-processor code conversion results in $\frac{P}{N} + n - 1$ element transfers in sequence. The output encoding can be either binary or binary-reflected Gray code.*

With the entire address space encoded in a binary-reflected Gray code, and cyclic data allocation butterfly stages 0 though $p - n - 1$ are entirely local. For the butterfly stage corresponding to dimension $n$ in the binary encoding communication in dimension $n - 1$ is required by lemma 1. By lemma 2 memory location $m$ is assigned $G(m)||i^n$, if $|G(m)| \bmod 2 = 0$, otherwise $G(m)||\overline{i^n}$. With four data points per processor (or multiples thereof) the data allocation is identical to the allocation within the four $(n - 2)$-cubes after the all-to-all personalized communication stage in the case of cyclic allocation and binary encoded memory, with the exception of the local memory ordering.

Table 2 gives an example of the emulation of a five stage butterfly network on a 3-cube, assuming cyclic data allocation with the entire address space encoded in a binary-reflected Gray code. The sequence of dimension assignments are shown in Table 3. The scheduling of communications are shown in Figure 20.

For the bi-sectioning algorithm with inter-processor code conversion an exchange in dimension $n - 1$ is required for memory locations with $|G(m)| \bmod 2 = 1$. Such a communication changes the allocation to cyclic with binary encoded memory addresses. All processors encode $G(m)||i^n$. Examples of the data motion is shown in Figures 21 and 22. The scheduling of communications are shown in Figure 20.

| paddr $G(i)$ | 000 | 001 | 011 | 010 | 110 | 111 | 101 | 100 |
|---|---|---|---|---|---|---|---|---|
| $B(i)$ | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| Initial | 00000 | 00001 | 00010 | 00011 | 00100 | 00101 | 00110 | 00111 |
| | 01111 | 01110 | 01101 | 01100 | 01011 | 01010 | 01001 | 01000 |
| alloc. | 11111 | 11110 | 11101 | 11100 | 11011 | 11010 | 11001 | 11000 |
| | 10000 | 10001 | 10010 | 10011 | 10100 | 10101 | 10110 | 10111 |
| After | 00000 | 00001 | 00010 | 00011 | 11100 | 11101 | 11110 | 11111 |
| exch. | 01111 | 01110 | 01101 | 01100 | 10011 | 10010 | 10001 | 10000 |
| dim. 2 | 00111 | 00110 | 00101 | 00100 | 11011 | 11010 | 11001 | 11000 |
| and 4 | 01000 | 01001 | 01010 | 01011 | 10100 | 10101 | 10110 | 10111 |
| After | 00000 | 00001 | 00010 | 00011 | 10011 | 10010 | 10001 | 10000 |
| local | 01111 | 01110 | 01101 | 01100 | 11100 | 11101 | 11110 | 11111 |
| conv. | 00111 | 00110 | 00101 | 00100 | 10100 | 10101 | 10110 | 10111 |
| dim 3 | 01000 | 01001 | 01010 | 01011 | 11011 | 11010 | 11001 | 11000 |
| After | 00000 | 00001 | 01110 | 01111 | 11111 | 11110 | 10001 | 10000 |
| exch. | 00011 | 00010 | 01101 | 01100 | 11100 | 11101 | 10010 | 10011 |
| dim 1. | 00111 | 00110 | 01001 | 01000 | 11000 | 11001 | 10110 | 10111 |
| and 3 | 00100 | 00101 | 01010 | 01011 | 11011 | 11010 | 10101 | 10100 |
| After | 00000 | 00001 | 01001 | 01000 | 11000 | 11001 | 10001 | 10000 |
| local | 00011 | 00010 | 01010 | 01011 | 11011 | 11010 | 10010 | 10011 |
| conv. | 00111 | 00110 | 01110 | 01111 | 11111 | 11110 | 10110 | 10111 |
| dim 4 | 00100 | 00101 | 01101 | 01100 | 11100 | 11101 | 10101 | 10100 |
| After | 00000 | 00111 | 01111 | 01000 | 11000 | 11111 | 10111 | 10000 |
| exch. | 00011 | 00100 | 01100 | 01011 | 11011 | 11100 | 10100 | 10011 |
| dim. 0 | 00001 | 00110 | 01110 | 01001 | 11001 | 11110 | 10110 | 10001 |
| and 4 | 00010 | 00101 | 01101 | 01010 | 11010 | 11101 | 10101 | 10010 |

Table 2: Bi-sectioning with local code conversion applied to a 32 node butterfly network emulation on a 3-cube with cyclic data allocation and the entire address space in a binary-reflected Gray code on input. The processor addresses are encoded in binary code on output.

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Initial alloc. | Proc. addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_{n+1}$ | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| | Logic addr. field: | $(b_{p-1}$ | $g_{p-2}$ | $\ldots$ | $g_{n+1}$ | $g_n$ | $\|\|$ | $g_{n-1}$ | $g_{n-2}$ | $g_{n-3}$ | $\ldots$ | $g_0)$ |
| Local reord. $n+1$ | Proc. addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_{n+1}$ | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| | Logic addr. field: | $(g_{p-1}$ | $g_{p-2}$ | $\ldots$ | $b_{n+1}$ | $g_n$ | $\|\|$ | $g_{n-1}$ | $g_{n-2}$ | $g_{n-3}$ | $\ldots$ | $g_0)$ |
| Exch. dim. $n-1$ | Proc. addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_{n+1}$ | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| | Logic addr. field: | $(g_{p-1}$ | $g_{p-2}$ | $\ldots$ | $g_{n-1}$ | $g_n$ | $\|\|$ | $b_{n+1}$ | $g_{n-2}$ | $g_{n-3}$ | $\ldots$ | $g_0)$ |
| Local reord. dim $n$ | Proc. addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_{n+1}$ | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| | Logic addr. field: | $(g_{p-1}$ | $g_{p-2}$ | $\ldots$ | $g_{n-1}$ | $b_n$ | $\|\|$ | $b_{n+1}$ | $g_{n-2}$ | $g_{n-3}$ | $\ldots$ | $g_0)$ |
| Exch. dim. $n-2$ | Proc. addr. field: | $(a_{p-1}$ | $a_{p-2}$ | $\ldots$ | $a_{n-1}$ | $a_n$ | $\|\|$ | $a_{n-1}$ | $a_{n-2}$ | $a_{n-3}$ | $\ldots$ | $a_0)$ |
| | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ | $\uparrow$ |
| | Logic addr. field: | $(g_{p-1}$ | $g_{p-2}$ | $\ldots$ | $g_{n-1}$ | $g_{n-2}$ | $\|\|$ | $b_{n+1}$ | $b_n$ | $g_{n-3}$ | $\ldots$ | $g_0)$ |

Table 3: The first two dimension exchanges for bi-sectioning with local code conversion applied to cyclicly allocated data encoded in a binary-reflected Gray code. Processor addresses in binary code on output.

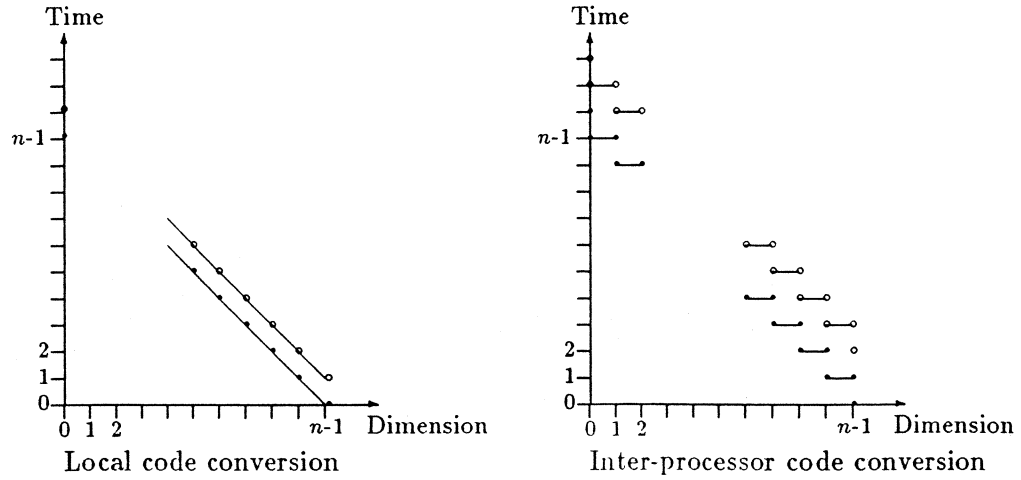


Figure 20: Scheduling of communications for the bi-sectioning algorithm. The entire address space in binary-reflected Gray code, cyclic data allocation.

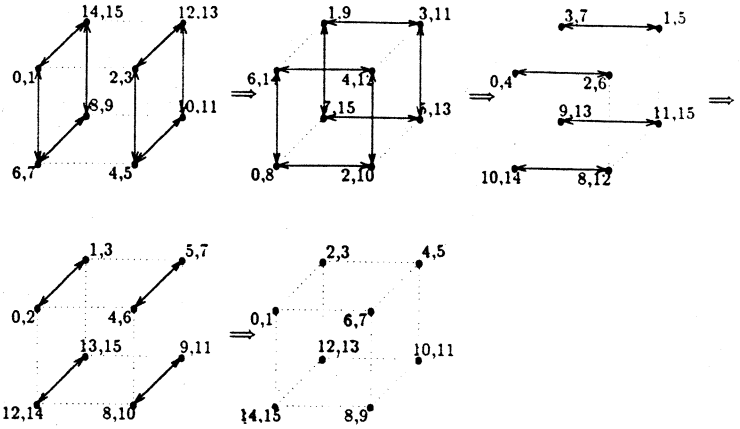


Figure 21: Bi-sectioning with inter-processor code conversion applied to binary-reflected Gray code encoded data indices, consecutive data allocation. The encoding on termination is in a binary-reflected Gray code with the dimensions cyclicly shifted one step to the right.

## 4.2 Consecutive mapping

The first $n-1$ emulation steps are identical for the entire address space encoded in a binary-reflected Gray code, and only the processor address field in such a code. The emulation step on the least significant processor dimension differs compared to binary encoded memory addresses only in the local memory addressing scheme. All emulation steps corresponding to memory dimensions in the binary encoding requires no inter-processor communication, with the exception of the dimension(s) moved to the processor address field in the first exchange(s). However, the local memory accesses differ compared to a binary encoded memory. The communication complexity is given by theorem 3. Examples of the data motion are shown in Figures 21 and 22.

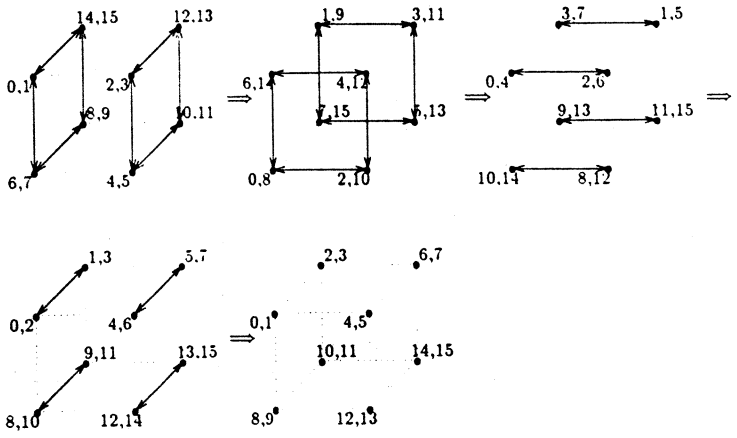


Figure 22: Bi-sectioning with inter-processor code conversion applied to binary-reflected Gray code encoded data indices, consecutive data allocation. The encoding on termination is in binary code with the dimensions cyclicly shifted one step to the right.

# 5 One data point per processor.

With one data point per processor the lower bound for the number of element transfers in sequence is $n$. An obvious way to perform the emulation is to apply the algorithm using a single memory dimension. The second memory location is initially empty and the communication in the most significant memory dimension is a send operation, not an exchange. After the initial step there are two elements per processor in an $(n-1)$-cube. A final splitting is required to restore one element per processor. The total number of communications is $n+1$. We will now prove that $n$ communication steps suffice.

**Theorem 5** *An emulation of a butterfly network of $N = 2^n$ points allocated to a Boolean n-cube with one point per processor by a binary-reflected Gray code can be performed with n element transfers in sequences. The output can be encoded in either binary code or binary-reflected Gray code.*

The idea is that each node sends a copy of its data to both its neighbors in the 2-cube. Hence, both neighbors can compute the complete butterfly. But, each node only computes one output of the butterfly. Which output is computed determines the final data ordering. The rule for deciding which output to compute is that after stage $k$ the $k+1$ most significant bits of the output must agree with the final data encoding. With the emulation proceeding from the most significant bit to the least significant bit the encoding is generated in the same order. The communication in each step of the algorithm is the same as in the algorithm using a single memory dimension. The second memory dimension is a copy of the first location.

It follows from the discussion of the bi-section algorithm with inter-processor code conversion that after the first communication step the $n-1$ least significant bits of the original data index in processor $(a_{n-2}a_{n-3}\dots a_0)$ is $i^{\underline{n-1}} = G_{n-1}^{-1}((a_{n-2}a_{n-3}\dots a_0))$ in subcube $1_{n-1}$ and $\overline{i^{\underline{n-1}}} = G_{n-1}^{-1}((a_{n-2}a_{n-3}\dots a_0))$ in subcube $0_{n-1}$. Subcube $0_{n-1}$ computes the "top" of a butterfly, and subcube $1_{n-1}$ the "bottom", with indices increasing from top to bottom. The index assigned to a processor is that originally assigned to the neighbor in dimension $n-2$. Figure 23 illustrates the data motion in a 3-cube with input and output in a binary-reflected Gray code.

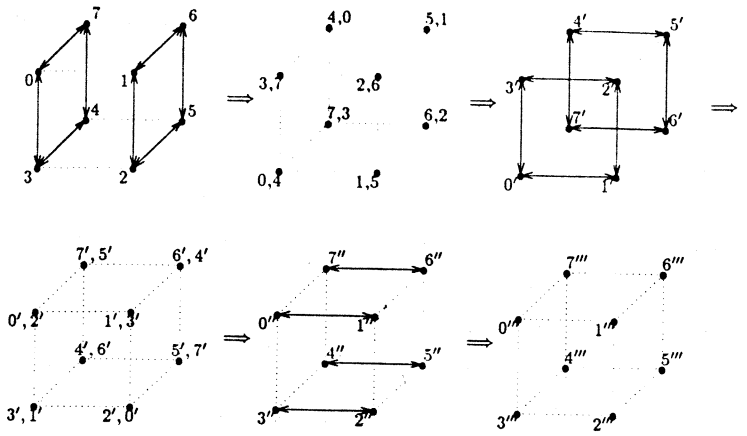

Figure 23: Emulation of a $2^n$ node butterfly network on an $n$-cube with binary-reflected Gray code encoded addresses.

In the seond stage with the final order being binary-reflected Gray code a node computes the butterfly output that corresponds to the current index of the neighbor across dimension $n-2$. Hence, dimension $n-2$ is complemented twice, and the initial ordering restored. For each pair of steps the Gray code encoding is restored. If $n$ is odd, then the last stage is an exchange step, and a computation of the butterfly output corresponding to the local index.

In the sample code below for input and output in a binary-reflected Gray code order, the butterfly computation in processor $i$ and stage $k$ is denoted $f(x, y, i, j)$. $x$ and $y$ are the butterfly inputs. $x$ has the smaller index in a pair, i.e. $x_{n-k-1} = 0$ and $y_{n-k-1} = 1$. $x \leftarrow (j, y)$ denotes a communication from variable $y$ of processor $j$ to variable $x$ of the current processor. Local assignment is denoted by the symbol :=.

```
/* Code for processor i: */
bstage := 0;
for j := n − 1 downto mod(n, 2) do
    x ← (i ⊕ 2^j, m[0]);
    y ← (i ⊕ 2^(j−1), m[0]);
    if (even(bstage)) then
        m[0] := f(x, y, G^(−1)(i ⊕ (1^j)), j);
    else
        m[0] := f(x, y, G^(−1)(i), j);
    endif
    bstage := bstage + 1;
endfor
if (odd(n)) then
    if (even(G^(−1)(i))) then
        x := m[0];
        y ← (i ⊕ 1, m[0]);
    else
        x ← (i ⊕ 1, m[0]);
        y := m[0];
    endif
```

| Initial Allocation | Encoding memory\|\|processor | Bi-section interpr. code conv. | Bi-sectioning with local code conv. |
|---|---|---|---|
| Consecutive | Gray | $\frac{P}{2N} + \max(n, \frac{P}{N})$ | $\frac{P}{2N} + \max(n, \frac{P}{2N})$ |
| | Gray\|\|Gray<br>Binary\|\|Gray | $\frac{P}{2N} + \max(n, \frac{P}{N})$ | $\frac{P}{2N} + \max(n, \frac{P}{2N})$ |
| Cyclic | Gray | $\frac{P}{N} + n - 1$ | $\frac{P}{2N} + n - 1$ |
| | Gray\|\|Gray<br>Binary\|\|Gray | $\frac{P}{N} + n - 2$ | $\frac{P}{2N} + n - 1$ |

Table 4: Data transfer times for the bi-sectioning algorithms for binary-reflected Gray code encoded data.

**endif**
$m[0] := f(x, y, G^{-1}(i), 0);$

For a binary coded output the only difference compared to the above algorithm is the butterfly output a node computes in a given stage. If the data allocation on input to emulation step $k$ is normal ($\overline{i^{n-k}}$), then a processor with $a_{n-1-k} = 1$ computes "bottom" and the other processors compute "top". If the allocation is reversed ($\overline{i^{n-k}}$), then for a binary ordered output processor $a_{n-1-k} = 1$ instead computes "top".

# 6 Summary and discussion

Table 6 summarizes the communication complexities of the bi-sectioning algorithms with local code conversion, or inter-processor code conversion. The emulation for one or two elements per processor requires $n$ communication cycles, each of which sends data in two dimensions. For more than two elements per processor one additional cycle is required for every pair of elements. The encoding of the data on termination of the algorithms can be either in binary code, or binary-reflected Gray code. The control of the data motion is a function of the encoding of data before and after the emulation, the allocation (*cyclic* or *consecutive*), and the emulation step. The control can be completely distributed. It is derived from local memory addresses, the processor address, the emulation step, and the original final data encoding. All communication channels of the Boolean cube can be used through pipelining, or multi-sectioning.

The bi-sectioning algorithms results in a permutation of the assignment of logic dimensions to machine dimensions. To restore the original dimension assignment a dimension permutation is required. Concurrent algorithms for dimension permutation are given in [2, 6]. (Sequential algorithms are given in [12, 13, 1, 15]). The dimension permutation algorithms can be pipelined with the emulation algorithms, but for $\frac{P}{N} \gg n$ the communication time for emulation with restoration of dimension assignment is approximately twice that of emulation alone.

Throughout the paper we have assumed that the emulation is starting from the most significant dimension. If the emulation instead is starting from the least significant dimension, then the algorithms for code conversion starting from the least significant dimension should be used. The communication complexity remains the same as if the emulation starts from the most significant dimension.

In $2^r$-sectioning $r$ processor dimensions are used for partitioning of the data set. This partitioning is an all-to-all personalized communication with code conversion within $r$-cubes. The number of element transfers in sequence for this operation is $\frac{R}{2}$ with at least $r$ communication start-ups [7]. The first multi-sectioning allows for local emulation of $r-1$ butterfly stages by lemma 1. For the emulation on dimension $n-r$ half of the data in each processor needs to be exchanged in dimension $n-r-1$. For $\frac{P}{N} > R$ pipelining can be used for the emulation of different sets of $R$ butterfly networks. For cyclic allocation with binary coded memory addresses and binary-reflected Gray code encoded processor addresses the total number of element transfers in sequence is $\frac{P}{2N} + (\frac{n}{r} - 1)\frac{R}{2}$, $r < n$. The second term can be reduced further if pipelining is applied to the $n-r$ dimensions remaining after the first $2^r$-sectioning. The pipeline delay is entirely avoided for $r = n$. Pipelined bi-sectioning, or four-sectioning, are second best to $N$-sectioning.

**Acknowledgement**

This work has been supported in part by AFOSR grant AFOSR-89-0382 and by NSF/DARPA grant CCR-8908285.

## References

[1] Peter M. Flanders. A unified approach to a class of data movements on an array processor. *IEEE Trans. Computers*, 31(9):809–819, September 1982.

[2] Ching-Tien Ho and S. Lennart Johnsson. Stable dimension permutations on Boolean cubes. Technical Report YALEU/DCS/RR-617, Department of Computer Science, Yale University, October 1988.

[3] S. Lennart Johnsson. Odd-even cyclic reduction on ensemble architectures and the solution of tridiagonal systems of equations. Technical Report YALE/DCS/RR-339, Dept. of Computer Science, Yale University, October 1984.

[4] S. Lennart Johnsson. Communication efficient basic linear algebra computations on hypercube architectures. *J. Parallel Distributed Comput.*, 4(2):133–172, April 1987.

[5] S. Lennart Johnsson and Ching-Tien Ho. Matrix transposition on Boolean n-cube configured ensemble architectures. *SIAM J. Matrix Anal. Appl.*, 9(3):419–454, July 1988.

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

[7] S. Lennart Johnsson and Ching-Tien Ho. Spanning graphs for optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Computers*, 38(9):1249–1268, September 1989.

[8] S. Lennart Johnsson, Ching-Tien Ho, Michel Jacquemin, and Alan Ruttenberg. Computing fast Fourier transforms on Boolean cubes and related networks. In *Advanced Algorithms and Architectures for Signal Processing II*, volume 826, pages 223–231. Society of Photo-Optical Instrumentation Engineers, 1987.

[9] S. Lennart Johnsson, Michel Jacquemin, and Ching-Tien Ho. High radix FFT on Boolean cube networks. Technical Report Department of Computer Science, Yale University, Technical Report YALEU/DCS/RR-751, November 1989, Thinking Machines Corp., November 1989.

[10] S. Lennart Johnsson, Robert L. Krawitz, Douglas MacDonald, and Roger Frye. A radix-2 FFT on the Connection Machine. In *Supercomputing 89*, pages 809–819. ACM, November 1989.

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

[12] David Nassimi and Sartaj Sahni. An optimal routing algorithm for mesh-connected parallel computers. *JACM*, 27(1):6–29, January 1980.

[13] David Nassimi and Sartaj Sahni. Optimal bpc permutations on a cube connected simd computer. *IEEE Trans. Computers*, C-31(4):338–341, April 1982.

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

[15] Paul N. Swarztrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197–210, 1987.