The Complexity of Reshaping Arrays on Boolean Cubes

S. Lennart Johnsson and Ching-Tien Ho

YALEU/DCS/TR-787
March 1990

To appear in the Proceedings of the Distributed Memory Computing Conference. Charleston, S.C. April 1990.

The Complexity of Reshaping Arrays on Boolean Cubes

S. Lennart Johnsson*
Department of Computer Science
Yale University
New Haven, CT 06520
Johnsson@cs.yale.edu, Johnsson@think.com

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

Abstract

Reshaping of arrays is a convenient programming primitive. For arrays encoded in a binary-reflected Gray code reshaping implies code change. We show that an axis splitting, or combining of two axes, requires communication in exactly one dimension, and that for multiple axes splittings the exchanges in the different dimensions can be ordered arbitrarily. The number of element transfers in sequence is independent of the number of dimensions requiring communication for large local data sets, and concurrent communication. The lower bound for the number of element transfers in sequence is $\frac{K}{2}$ with K elements per processor. We present algorithms that is of this complexity for some cases, and of complexity K in the worst case. Conversion between binary code and binary-reflected Gray code is a special case of reshaping.

1 Introduction

In computer systems locality of reference has had a significant impact on performance ever since memory hierarchies were introduced. In modern computer systems small memories in MOS technologies may be designed for higher speeds than larger memories. In multiprocessor systems with processors and memory modules interconnected via a network, the access time for non-local information is typically considerably longer than local access. Moreover, the access time depends upon the network topology, congestion and bandwidth of the communications network. The reference pattern has a significant impact on the optimal data allocation in networks that have a non-uniform distance between pairs of nodes, such as Boolean cube networks.

In well structured computations the data is conveniently represented by arrays. Many algorithms require local references in a Cartesian space corresponding to the array. Explicit methods for the solution of partial differential equations are examples thereof. Preserving the locality in the Cartesian space when mapped to the processor network is important with respect to performance. The binary-reflected Gray code is often used to accomplish this task in Boolean cube networks. Successive integers in the decimal encoding differ by one bit in their Gray code encoding. This property is used in CM-Fortran [1], Thinking Machines Corp. version of Fortran 8X [11] for the Connection Machine. In this language implementation, array axes are by default encoded in a binary-reflected Gray code.

Some important algorithms with a regular communication pattern depend on local references in a Boolean space. For instance, the Fast Fourier Transform requires communication in the form of a butterfly network, which implies communication between adjacent nodes in a Boolean space with corresponding nodes in different ranks mapped to the same processor. In many scientific and engineering applications algorithms that depend upon both types of access patterns may be used, and conversion between the two storage forms may be important.

Many recursive algorithms make use of axis splitting, or combining. An example is the data parallel implementation [2] of the divide-and-conquer algorithm by Dongarra and Sorensen [3] for computing eigenvalues of symmetric tridiagonal systems. Array manipulation through operations such as **RESHAPE** in Fortran 8X and APL, impacts the encoding for binary-reflected Gray coded axes. The encoding of binary coded axes is unaffected.

Different axes may have different encoding. For instance, if butterfly computations are performed along one axis, and nearest-neighbor communications in a Cartesian space along the other axis of a two-dimensional array, then binary encoding of the first axis and binary-reflected Gray code encoding of the second axis is desirable. Furthermore, the encoding of a single axis may be mixed. Typically the number of array elements along an axis exceeds the number of processors allocated to the axis, forcing several elements along an axis to be allocated to the memory of each proces-

*The author is also with Thinking Machines Corp., 245 First Street, Cambridge, MA 02142. This work was supported in part by AFOSR-89-0382 and ONR Contract No. N00014-86-K-0310.

†Part of the work was done while the author was with the Department of Computer Science, Yale University.

sor with the array elements being allocated as evenly as possible. *Cyclic* and *consecutive* [6] allocation are two common schemes for assigning multiple elements to processors. With local random access memories distance is not an issue in determining the encoding for the local memories. Binary encoding is typically used for the local part of an axis, and binary-reflected Gray code for the processor part.

As an example consider a two-dimensional *logic array* A of shape $P \times Q$ allocated to an $N_1 \times N_0$ *physical array* of processors, where $P = 2^p$, $Q = 2^q$, $N_1 = 2^{n_1}$, $N_0 = 2^{n_0}$, $p \geq n_1$ and $q \geq n_0$. The data allocation is consecutive, and each array axis is encoded in a binary-reflected Gray code. Bit m in the address space is denoted g_m if encoded in a binary-reflected Gray code, and b_m if encoded in binary code. Bit zero, or dimension zero, is the least significant, and the rightmost dimension in our expressions. The symbol $||$ denotes concatenation of two fields. Axes are also labeled right to left. We illustrate the allocation as follows

$$(\underbrace{g^1_{p-1}g^1_{p-2}\cdots g^1_{p-n_1}}_{\text{paddr}^1}\underbrace{g^1_{p-n_1-1}g^1_{p-n_1-2}\cdots g^1_0}_{\text{maddr}^1}||$$

$$\underbrace{g^0_{q-1}g^0_{q-2}\cdots g^0_{q-n_0}}_{\text{paddr}^0}\underbrace{g^0_{q-n_0-1}g^0_{q-n_0-2}\cdots g^0_0}_{\text{maddr}^0}).$$

The processor address for an element (i,j) of the logic array is formed as $(\text{paddr}^1(i)||\text{paddr}^0(j))$, and the local storage address is $(\text{maddr}^1(i)||\text{maddr}^0(j))$, where $G_p(i) = (g^1_{p-1}g^1_{p-2}\cdots g^1_0)$ is the binary-reflected Gray code encoding of i, and $G_q(j) = (g^0_{q-1}g^0_{q-2}\cdots g^0_0)$ is the binary-reflected Gray code encoding of j. Reshaping the logic array into a one-dimensional array such that $(i,j) \rightarrow iQ + j$ preserving the assignment of bits in the logic array to bits in the physical address space implies a code conversion for axis zero if i is odd, and data motion within n_0 dimensional subcubes. The result is an allocation of the form

$$(\underbrace{g_{p+q-1}g_{p+q-2}\cdots g_{p+q-n_1}}_{\text{paddr}^1}\underbrace{g_{p+q-n_1-1}g_{p+q-n_1-2}\cdots g_q}_{\text{maddr}^1}||$$

$$\underbrace{g_{q-1}g_{q-2}\cdots g_{q-n_0}}_{\text{paddr}^0}\underbrace{g_{q-n_0-1}g_{q-n_0-2}\cdots g_0}_{\text{maddr}^0})$$

where, as shown later, $g_{m+q} = g^1_m$, $m \in \{0,\cdots,p-1\}$ and $g_m = g^0_m$, $m \in \{0,\cdots,q-2\}$. The value of g_{q-1} depends upon the value of g_q. Figure 1 illustrates the data motion.

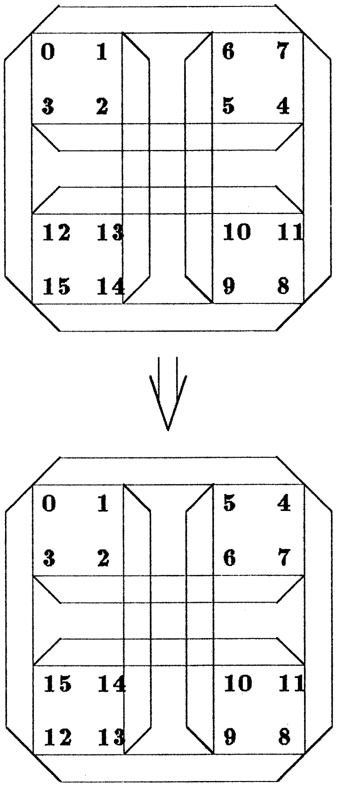

Figure 1: Reshaping an 1 × 16 array to a 4 × 4 array.

Note that whereas the initial data allocation was consecutive the data allocation after reshaping is not. If a *consistent* data allocation is desired, i.e., the same data allocation scheme before and after reshaping, then it is in general necessary to change the assignment of dimensions in the *logic address space* to dimensions in the *physical address space*. A *dimension permutation* [4,13,12,15,10,5] in the form of an n_0 step right cyclic shift, or $p - n_1$ steps left cyclic shift on the dimensions in the field $(\text{maddr}^1||\text{paddr}^0)$ is required, in combination with code conversion.

With consecutive allocation of A and a binary encoding of local addresses, and a binary-reflected Gray code encoding of processor addresses, the processor address of element (i,j) is formed by computing the address from the binary-reflected Gray codes of $\lfloor i/N_1 \rfloor$ and $\lfloor j/N_0 \rfloor$. The local memory address is determined from the binary codes of $i \bmod N_1$ and $j \bmod N_0$. The encoding of the address field is

$$(\underbrace{g^1_{p-1}g^1_{p-2}\cdots g^1_{p-n_1}}_{\text{paddr}^1}\underbrace{b^1_{p-n_1-1}b^1_{p-n_1-2}\cdots b^1_0}_{\text{maddr}^1}||$$

$$\underbrace{g^0_{q-1}g^0_{q-2}\cdots g^0_{q-n_0}}_{\text{paddr}^0}\underbrace{b^0_{q-n_0-1}b^0_{q-n_0-2}\cdots b^0_0}_{\text{maddr}^0}).$$

Reconfiguration of the processor array is equivalent to changing the assignment of dimensions in the logic address space to dimensions in the physical address space. A dimension permutation is required. If the encoding of the local address field is different from the processor address field, then a code conversion is required in combination with the dimension permutation. Reconfiguration of a processor array may be required to assure that all operands use the same physical machine configuration, as for instance in matrix multiplication on the Connection Machine [8]. The Connection Machine Fortran compiler allocates logic arrays to the processors by defining a processor array congruent to the logic array for each array. Hence, in the matrix multiplication $C \leftarrow A \times B$ all three matrices may assume a different shape of the processor array.

In this paper, we show how an axis splitting, or the combining of two axes into one, can be performed by a single exchange operation. For multiple axes split/merge operations, the number of element transfers in sequence is independent of the number of axes created or merged, if the communication system allows concurrent communication in all required dimensions. The number of element transfers in sequence is only a function of the size of the local data set, if there is a large local data set. The minimum number of element transfers in sequence is equal to the number of dimensions requiring communication. The conversion between binary-reflected Gray code and binary code is equivalent to reshaping between a one-dimensional array and a $2 \times 2 \times \cdots \times 2$ array of dimension n.

The algorithms we give for reshaping and code conversion are either asymptotically optimal, or optimal within a factor of two with respect to data transfer time. The control information can be computed locally from the node address. The code conversion can start in any dimension, and the required exchanges can be carried out in dimensions ordered arbitrarily. This property allows reshaping by concurrent communication in all required dimensions, if the size of the local data set exceeds the number of dimensions requiring communication. Compared to the algorithms in [6,7] the new algorithms avoid the pipeline delay. Here we only treat the case with an entire axis encoded in either binary code, or binary-reflected Gray code. Furthermore, we assume a fixed assignment of dimensions in the logic address space to dimensions in the physical address space. Reshaping combined with dimension permutations is considered in [9].

The paper is organized as follows. Notation and definitions are introduced next. Array reshaping is discussed in Section 3. The conversion between binary-reflected Gray code and binary code is discussed in Section 4, followed by summary in Section 5.

2 Preliminaries

A Boolean n-cube has $N = 2^n$ nodes. Two nodes are adjacent if and only if their addresses differ in exactly one bit. The binary encoding of i is $B_n(i) = (b_{n-1}b_{n-2}\cdots b_0)$ and its binary-reflected Gray code encoding is $G_n(i) = (g_{n-1}g_{n-2}\cdots g_0)$. $\mathcal{Z}_N = \{0, 1, \cdots, N-1\}$ and (1^j) is a string of j instances of a bit with value one. "$||$" is the concatenation symbol. For the complexity estimates we assume bi-directional channels and concurrent communication on all channels. The number of elements per node is K. $\hat{G}_n$ is the sequence of n-bit binary-reflected Gray codes for $\mathcal{Z}_N$, i.e., $\hat{G}_n = (G_n(0), G_n(1), \cdots, G_n(2^n-1))$.

Definition 1 [14] The *binary-reflected Gray code* is defined recursively as follows.

$$\hat{G}_1 = (G_1(0), G_1(1)), \text{where } G_1(0) = 0, G_1(1) = 1.$$

$$\hat{G}_{n+1} = \begin{pmatrix} 0||G_n(0) \\ 0||G_n(1) \\ \vdots \\ 0||G_n(2^n-2) \\ 0||G_n(2^n-1) \\ 1||G_n(2^n-1) \\ 1||G_n(2^n-2) \\ \vdots \\ 1||G_n(1) \\ 1||G_n(0) \end{pmatrix}.$$

In the following we always refer to the binary-reflected Gray code defined above.

Corollary 1 *The highest order bit is the same in the binary code and the binary-reflected Gray code. The remaining bits in the encoding of $i \in \mathcal{Z}_{N/2}$ are defined by $G_{n-1}((b_{n-2}b_{n-3}\cdots b_0))$. The remaining bits in the encoding of $i \in \mathcal{Z}_N - \mathcal{Z}_{N/2}$ are defined by $G_{n-1}((\bar{b}_{n-2}\bar{b}_{n-3}\cdots\bar{b}_0))$. Thus,*

$$G_n((b_{n-1}b_{n-2}\cdots b_0)) = \begin{cases} b_{n-1}||G_{n-1}((b_{n-2}b_{n-3}\cdots b_0)), \\ \quad \text{if } b_{n-1} = 0, \\ b_{n-1}||G_{n-1}((\bar{b}_{n-2}\bar{b}_{n-3}\cdots\bar{b}_0)), \\ \quad \text{if } b_{n-1} = 1. \end{cases}$$

Proof: From Definition 1. ▮

Corollary 2 *The integer encoded in the neighbor of node $G_n(i)$ in cube dimension j is $G_n(i \oplus (1^{j+1}))$, i.e., $G_n(i) \oplus 2^j = G_n(i \oplus (1^{j+1}))$.*

Proof: It follows from Corollary 1. ▮

Definition 2 With binary-reflected Gray code encoding of an N-element one-dimensional array $A[i], i \in \mathcal{Z}_N$ into an n-cube, address $G_n(i)$ contains $A[i]$.

Lemma 1 [14] $b_m = g_{n-1} \oplus g_{n-2} \oplus \cdots \oplus g_m$, $m \in \mathcal{Z}_n$. *Conversely,* $g_m = b_m \oplus b_{m+1}$, $m \in \mathcal{Z}_n$ *with* $b_n = 0$.

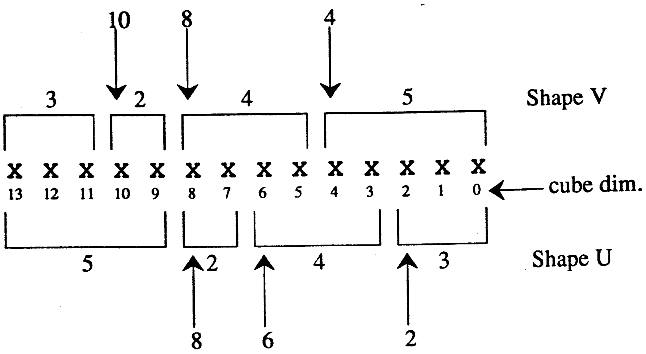

Figure 2: Reshaping between two arrays with binary-reflected Gray code encoded on a Boolean cube.

Definition 3 Let A be an array of shape $U_{d-1} \times U_{d-2} \times \cdots \times U_0$, $U = (U_{d-1}, U_{d-2}, \cdots, U_0)$, $U_m = 2^{u_m}$, $m \in \mathcal{Z}_d$, $V = (V_{d'-1}, V_{d'-2}, \cdots, V_0)$, $V_m = 2^{v_m}$, $m \in \mathcal{Z}_{d'}$ and $\prod_{m=0}^{d-1} U_m = \prod_{m=0}^{d'-1} V_m$. The reshape function $\rho(U, V)$ transforms the *shape* of the array A from U to V.

Let $\tilde{u}_k = (\sum_{m=0}^{k} u_m) - 1$, $\tilde{v}_k = (\sum_{m=0}^{k} v_m) - 1$, $\tilde{\mathcal{U}} = \{\tilde{u}_k \mid 0 \leq k < d-1\}$, $\tilde{\mathcal{V}} = \{\tilde{v}_k \mid 0 \leq k < d'-1\}$ and $\mathcal{D} = (\tilde{\mathcal{U}} \cup \tilde{\mathcal{V}}) - (\tilde{\mathcal{U}} \cap \tilde{\mathcal{V}})$. The sets $\tilde{\mathcal{U}}$ and $\tilde{\mathcal{V}}$ are the sets of most significant dimensions for the axes of the shapes U and V, with the most significant axes excluded. For instance, if $U = (2^5, 2^2, 2^4, 2^3)$ and $V = (2^3, 2^2, 2^4, 2^5)$, then $\tilde{\mathcal{U}} = \{8, 6, 2\}$, $\tilde{\mathcal{V}} = \{10, 8, 4\}$ and $\mathcal{D} = \{10, 6, 4, 2\}$, Figure 2. To form the shape V from U communication is required in the set of dimensions defined by $\tilde{\mathcal{U}} - \tilde{\mathcal{V}}$ for axes being combined into one, and the set of dimensions defined by $\tilde{\mathcal{V}} - \tilde{\mathcal{U}}$ for axes being split. $\mathcal{D}$ is the set of dimensions for which communication is required for changing the shape U into V. $\mathcal{D}_{\text{paddr}}$ is the subset of dimensions in $\mathcal{D}$ assigned to processor dimensions in the physical address space. $\mathcal{D}_{\text{maddr}} = \mathcal{D} - \mathcal{D}_{\text{paddr}}$ is the set of dimensions in $\mathcal{D}$ assigned to local memory dimensions in the physical address space.

3 Reshaping Arrays

Lemma 2 below states the fact that splitting an axis into two, or merging two axes into one, requires a code change in precisely one dimension.

Lemma 2 *Assume node $G_n(i)$ contains element $A[i]$, $i \in \mathcal{Z}_N$, initially. If all nodes $i = (b_{n-1}b_{n-2} \cdots b_0)$ such that $b_m = 1$ exchange data in dimension $m - 1$ for any $m \in \{1, 2, \cdots, n-1\}$, then node $G_{n-m}((b_{n-1}b_{n-2} \cdots b_m)) || G_m((b_{m-1}b_{m-2} \cdots b_0))$ contains element $A[i]$ after the exchange.*

Proof: Assume that the reshape operation is $U = (2^n) \rightarrow V = (2^{n-m}, 2^m)$, and that address $G_n(i) = (g_{n-1}(i)g_{n-2}(i) \cdots g_0(i))$ initially contains element $A[i]$. Let $i = k2^m + \ell$, $\ell \in \mathcal{Z}_{2^m}$, $k \in \mathcal{Z}_{N/2^m}$. After the reshape operation element i, now (k, ℓ), should reside in address $(G_{n-m}(k) || G_m(\ell))$, where $G_{n-m}(k) = G_{n-m}((b_{n-m-1}(k)b_{n-m-2}(k) \cdots b_0(k))) = (g_{n-m-1}(k)g_{n-m-2}(k) \cdots g_0(k))$
and $G_m(\ell) = G_m((b_{m-1}(\ell)b_{m-2}(\ell) \cdots b_0(\ell))) = (g_{m-1}(\ell)g_{m-2}(\ell) \cdots g_0(\ell))$.

¿From the binary encoding $b_j(k) = b_{m+j}(i)$, $j \in \mathcal{Z}_{n-m}$, and $b_j(\ell) = b_j(i)$, $j \in \mathcal{Z}_m$. By Lemma 1, $g_j(k) = b_j(k) \oplus b_{j+1}(k) = b_{m+j}(i) \oplus b_{m+j+1}(i) = g_{m+j}(i)$, for all $j \in \mathcal{Z}_{n-m}$, and $g_j(\ell) = b_j(\ell) \oplus b_{j+1}(\ell) = b_j(i) \oplus b_{j+1}(i) = g_j(i)$ for all $j \in \mathcal{Z}_{m-1}$. But, $g_{m-1}(\ell) = b_{m-1}(\ell) \oplus b_m(\ell) = b_{m-1}(i)$ and $g_{m-1}(i) = b_{m-1}(i) \oplus b_m(i)$, i.e.,

$$g_{m-1}(\ell) = \begin{cases} g_{m-1}(i), & \text{if } b_m(i) = 0, \\ \overline{g_{m-1}(i)}, & \text{if } b_m(i) = 1. \end{cases}$$

Hence, if $b_m(i) = 0$ then $G_n(i) = G_{n-m}(k) || G_m(\ell)$ and no data motion is necessary for reshaping. But, if $b_m(i) = 1$ then an exchange is required in dimension $m - 1$, and only in dimension $m - 1$, since this dimension is the only dimension in which the code for i and (k, ℓ) differs. ∎

The change in the binary-reflected Gray code caused by an axis splitting, or the merging of two axes, is limited to the most significant dimension of the lower ordered axis in the created pair of axes. The pairs of addresses exchanging content in a given dimension depend upon the order of exchanges in the case of multiple axes splittings. The control of the exchange is derived from b_m in the encoding of i. The index i assigned to an address changes if a more significant controlling dimension is one. For example, consider the reshaping of an array of 8 elements encoded in a binary-reflected Gray code to an array of $2 \times 2 \times 2$ elements (which is equivalent to conversion to binary code). Figure 3 shows exchanged data in boldface, and two exchange orders: dimension one then zero, or zero then one. As is apparent from Figure 3, an exchange is carried out in dimension one between addresses 110 and 111 if the dimensions are treated in the order one first then zero, but not if the order is dimension zero first, then dimension one.

The current value of b_m that is assigned to a given address $(g_{n-1}g_{n-2} \cdots g_0)$ is easily determined from the address.

Lemma 3 *If the number of exchanges in dimensions more significant than m is even, then the current value of logic dimension m assigned to an address $G_n(i) = (g_{n-1}g_{n-2} \cdots g_0)$ is b_m, otherwise it is $\bar{b}_m$.*

The lemma follows directly from Corollary 2.

Half of the total number of elements need to be exchanged for any split/merge operation. Hence, the number of exchanges in which an element participates falls in

Gray code assignment		Exch. dim. 1		Exch. dim. 0		Exch. dim. 0		Exch. dim. 1	
i	paddr	b_2	i	b_1	i	b_1	i	b_2	i
0	000	0	0	0	0	0	0	0	0
1	001	0	1	0	1	0	1	0	1
2	011	0	2	1	**3**	1	**3**	0	3
3	010	0	3	1	**2**	1	**2**	0	2
4	110	1	**7**	1	**6**	0	4	1	**6**
5	111	1	**6**	1	**7**	0	5	1	**7**
6	101	1	**5**	0	5	1	**7**	1	**5**
7	100	1	**4**	0	4	1	**6**	1	**4**

Figure 3: Reshaping an array of 8 elements into a $2\times2\times2$ array.

the range $0 - |\mathcal{D}|$, depending upon its binary encoding. The total number of element exchanges is $|\mathcal{D}|\frac{\Pi_{m=0}^{d-1}U_m}{2}$ for changing shape U to shape V. We will now determine the number of element exchanges in sequence when the logic array is allocated to an n-cube, with $\frac{\Pi_{m=0}^{d-1}U_m}{N} = K$ elements per processor.

Theorem 1 *A lower bound for the number of element transfers in sequence for array reshaping affecting the encoding of processor dimensions is $K/2$ with K elements per processor.*

Proof: Pick a dimension $d \in \mathcal{D}_{\text{paddr}}$. There are $N/2$ processors that need to transfer data across dimension d. There are K elements in each processor, and all elements need to be exchanged. The available bandwidth per dimension is N. ▮

In the following, let $\delta = |\mathcal{D}_{\text{paddr}}|$.

Theorem 2 *Changing the shape U to shape V preserving the assignment of logic dimensions to physical dimensions requires at most $\delta\lceil\frac{K}{\delta}\rceil$ element transfers in sequence with concurrent communication.*

Proof: Let $\mathcal{D}_{\text{paddr}} = \{d_{\delta-1}, d_{\delta-2}, \cdots, d_0\}$. Partition the local data set of size K into δ sets of size at most $\lceil\frac{K}{\delta}\rceil$ each. Label the data sets from 0 to $\delta - 1$. Each such set is assigned a sequence of dimensions including all dimensions in $\mathcal{D}_{\text{paddr}}$ once. Different sets are assigned different sequences such that no two sets have the same first, second, third, etc., dimension. For instance, let data in set m be assigned the sequence of dimensions $d_m, d_{(m+1)\bmod\delta}, \cdots, d_{(m-1)\bmod\delta}$. ▮

The upper bound in Theorem 2 differs from the lower bound by a factor of two. The upper bound can be improved in some cases. We give upper bounds that are almost identical to the lower bounds for two cases.

Theorem 3 *Changing the shape U to shape V preserving the assignment of logic dimensions to physical dimensions requires at most $\delta\lceil\frac{K}{2\delta}\rceil + 1$ element transfers in sequence with concurrent communication, if no two elements of $\mathcal{D}_{\text{paddr}}$ differ by one and $K > 2\delta$.*

Proof: Consider the merging of a single pair of axes, or splitting of an axis. Assume the communication occurs in dimension $m-1$. Consider a 2-cube formed by dimensions m and $m-1$. Label the four nodes according to $b_m b_{m-1}$. By Lemma 2, communication is only required between nodes 10 and 11. There exist two edge-disjoint paths between these two nodes of lengths one and three, respectively. By assigning $\lceil\frac{K}{2\delta}\rceil + 1$ elements to the path of length one and the remaining elements to the path of length three, $(\lceil\frac{K}{2\delta}\rceil + 1)$ element transfers in sequence are required.

If no two elements in $\mathcal{D}_{\text{paddr}}$ differ by one, then the 2-cubes used for different data sets are disjoint. Thus, $\delta(\lceil\frac{K}{2\delta}\rceil + 1)$ element transfers in sequence are required. To reduce the communication complexity to $\delta\lceil\frac{K}{2\delta}\rceil + 1$, we slightly overlap the communications on the successive 2-cubes of a given data set. Without this overlap no data is sent along the length three path during the last two cycles of the routing of a data set. By sending two elements that have been routed with respect to the first 2-cube to the length-three path of the second 2-cube during the last two cycles of the routing phase of the first 2-cube (with one cycle each), the communication delay due to the length-three path is only paid once. Sending elements along the length-three path during the last two cycles of the first 2-cube will not interfere with the communication of the data set exchanged in the second 2-cube. The reduced complexity is valid if $\lceil\frac{K}{\delta}\rceil > 2$, i.e., some data set has at least three elements. ▮

In the routing used for the proof of the bound, the number of elements routed along the length-one path and the length-three path differ by two only for the first 2-cube. For subsequent 2-cubes, the same number of elements are routed along each path, with the length-three path starting two cycles earlier. The first element on both paths arrives at the same time within the 2-cube except for the first 2-cube. If 2δ divides K and $K > 2\delta$, then the complexity is $\frac{K}{2} + 1$, which is only one element transfer above the lower bound. For $K \leq 2\delta$, there is no advantage of using the length-three paths over the algorithm used in the proof of Theorem 2.

If the reshape operation requires communication in dimensions $m-1$ and m (by creating an axis of length 2 encoded in dimension m), then dimension m cannot be used for rerouting to access unused communication links in dimension $m-1$. Unused links in dimensions lower than $m-1$ cannot be used either, since they do not connect to processors with unused links in dimension $m-1$. However, the following observation can be used to reduce the number of element transfers in sequence.

Lemma 4 *For a reshape operation requiring communication in dimension $m-1$ none of the links in dimension $m-1$ is used in $m-1$ dimensional subcubes obtained through complementing any of the address dimensions that are more significant than $m-1$.*

Proof: We need to show that in any $m-1$ dimensional subcube defined by dimensions m and higher, $b_m = 0$ if the address defining the subcube is obtained by complementing a single dimension of significance m or higher. But, by Lemma 1 complementing a single dimension g_j, $j \in \{m, m+1, \cdots, n-1\}$ complements b_m. ▮

By using a pipelined algorithm instead of the non-pipelined maximally concurrent algorithm used for the upper bound in Theorem 3, the properties in Lemma 4 can be exploited to establish the following bound.

Theorem 4 *Changing the shape U to shape V requires at most $\lceil \frac{K}{2} \rceil + 2\delta - 1$ element transfers in sequence, if for each dimension requiring communication there exists one more significant dimension not requiring communication and $K \geq 2\delta$.*

Proof: The problem is equivalent to sending K elements along a path of length δ and each edge on the path is paired with a length-three path, disjoint with all other edges. If δ is even two edge-disjoint paths of length 2δ can be defined by combining length-three and length-one paths for different dimensions. If δ is odd, then two paths of length $2\delta - 1$ and $2\delta + 1$ can be defined in a similar way. ▮

Several routing schemes yield the same complexity as the scheme used in the proof. For instance, by creating one path of length δ and one of length 3δ, and routing $\lceil \frac{K}{2} \rceil + \delta$ elements along the short route and $\lfloor \frac{K}{2} \rfloor - \delta$ elements along the long route the same routing time is achieved if $K \geq 2\delta$. For $K < 2\delta$, the latter approach degenerates to using a single path of length δ and the required time is $K + \delta - 1$, which is lower than if two paths of the same length were used. However, if $K < 2\delta$ then the time for reshaping by pipelining along one path is higher than, or at best the same as if the concurrent exchange algorithm in the proof of Theorem 2 is used.

Lemma 4 cannot be exploited directly for concurrent exchange sequences because an exchange in one dimension affects the set of edges being used in a subcube. This property follows from Lemma 3. For instance, if a 1×16 array is reshaped into a $4 \times 2 \times 2$ array, then if an exchange in dimension one is performed first the required exchanges in dimension zero are all on corresponding links in different subcubes instead of complimentary links.

4 Conversion between Gray code and binary code

Theorem 5 *The conversion between a binary-reflected Gray code and binary code in either direction requires communication in $n-1$ dimensions, and at most $(n-1)\lceil \frac{K}{n-1} \rceil$ element transfers in sequence.*

Theorem 5 follows from Theorem 2 and the observation that conversion from binary-reflected Gray code to binary code in an n-cube is equivalent to reshaping a one-dimensional array of size 2^n to an n-dimensional array of shape $2 \times 2 \times \cdots \times 2$.

In any algorithm according to Lemma 2 and Theorem 5 only half of the communications links in each of the $n-1$ dimensions are used in every step of the algorithm. Every path is of minimum length, and all minimum length paths are used evenly. The load on the communications network is minimal.

Conjecture 1 *For the conversion between binary-reflected Gray code and binary code encodings of K elements per processor in an n-cube, a lower bound is $K\frac{n-1}{n}$.*

For $n = 2$, the conjecture follows from Theorem 3. For $n > 2$ only the most significant dimension requires no communication.

Corollary 3 *The conversion between binary-reflected Gray code and binary code encoding in an n-cube can be performed as an arbitrary sequence of communications in dimensions: $\{0, 1, \cdots, n-2\}$.*

The corollary follows from the observation that the control is completely determined by the binary encoding of i.

An algorithm proceeding from dimension $n-2$ to dimension 0 is depicted in Figure 4. Initially, processor $G_4(i)$ contains data of index i. After the conversion, i is assigned to processor $B_4(i)$. The algorithm is described below. Several other algorithms are given in [7].

```
/* Converting Gray code to binary code
starting from the most significant dimension */

for $d := n-2$ downto 0 do
    if $g_{d+1} = 1$ then
        exch. content with the neighbor in dim. $d$
    endif
enddo
```

The control in the above algorithm is particularly simple, since the following corollary follows from Lemma 3.

Gray code		Exchange dim. 2		Exchange dim. 1		Exchange dim. 0	
data	paddr	b_3	data	b_2	data	b_1	data
00	0000	0	00	0	00	0	00
01	0001	0	01	0	01	0	01
02	0011	0	02	0	02	1	**03**
03	0010	0	03	0	03	1	**02**
04	0110	0	04	1	**07**	1	**06**
05	0111	0	05	1	**06**	1	**07**
06	0101	0	06	1	**05**	0	05
07	0100	0	07	1	**04**	0	04
08	1100	1	**15**	1	**12**	0	12
09	1101	1	**14**	1	**13**	0	13
10	1111	1	**13**	1	**14**	1	**15**
11	1110	1	**12**	1	**15**	1	**14**
12	1010	1	**11**	0	11	1	**10**
13	1011	1	**10**	0	10	1	**11**
14	1001	1	**09**	0	09	0	09
15	1000	1	**08**	0	08	0	08

Figure 4: Conversion of a binary-reflected Gray code to binary code

Corollary 4 *If the conversion from binary-reflected Gray code to binary code proceeds from the most significant dimension to the least significant dimension, then the current value of b_m assigned to an address is equal to g_m, where m is the controlling dimension.*

The algorithm is easy to generalize to an arbitrary starting dimension m, $m \in \mathcal{Z}_{n-1}$ with exchanges in successive dimensions of decreasing order in a cyclic fashion. The first exchange requires the computation of b_m. Figure 5 gives an example. Sequence 2 is the same as in Figure 4. The figure shows the location of i for each step of the algorithm for each sequence. For concurrent exchanges the local data set K is divided into $n-1$ sets, and set m, $m \in \mathcal{Z}_{n-1}$ is subject to exchange in dimension $(n-2-m-t) \bmod (n-2)$ during step t, $t \in \mathcal{Z}_{n-1}$.

```
/* Converting Gray code to binary code starting from
dimension $m$. Dimensions in decreasing order, cyclically*/
if $g_{n-1} \oplus g_{n-2} \oplus \cdots \oplus g_{m+1} = 1$ then
    exch. content with the neighbor in dim. $m$
endif
for $d := m-1$ downto 0 do
    if $g_{d+1} = 1$ then
        exch. content with the neighbor in dim. $d$
    endif
enddo
for $d := n-2$ downto $m+1$ do
    if $g_{d+1} = 1$ then
        exch. content with the neighbor in dim. $d$
    endif
enddo
```

Gray code assignment		Seq 2 Exchange dim.			Seq 1 Exchange dim.			Seq 0 Exchange dim.		
Data	paddr	2	1	0	1	0	2	0	2	1
0	0000	0	0	0	0	0	0	0	0	0
1	0001	1	1	1	1	1	1	1	1	1
2	0011	2	2	3	2	3	3	3	3	3
3	0010	3	3	2	3	2	2	2	2	2
4	0110	4	7	6	7	6	6	4	4	6
5	0111	5	6	7	6	7	7	5	5	7
6	0101	6	5	5	5	5	5	7	7	5
7	0100	7	4	4	4	4	4	6	6	4
8	1100	15	12	12	8	8	12	8	14	12
9	1101	14	13	13	9	9	13	9	15	13
10	1111	13	14	15	10	11	15	11	13	15
11	1110	12	15	14	11	10	14	10	12	14
12	1010	11	11	10	15	14	10	12	10	10
13	1011	10	10	11	14	15	11	13	11	11
14	1001	9	9	9	13	13	9	15	9	9
15	1000	8	8	8	12	12	8	14	8	8

Figure 5: Concurrent conversion of a binary-reflected Gray code to binary code.

5 Summary

We have shown that the splitting of a binary-reflected Gray code encoded axis into two binary-reflected Gray coded axes only requires an exchange in the most significant dimension of the lower order axis. The exchanges required for multiple axis splittings can be performed in arbitrary order.

Assume concurrent communication on all ports, K elements per processor, and δ dimensions requiring communication for the reshape operation. If K is a multiple of δ, then the number of element transfers in sequence is independent of δ. An upper bound is K and a lower bound is $\frac{K}{2}$. We present three algorithms: (i) one of communication complexity $\delta\lceil\frac{K}{\delta}\rceil$, (ii) one of complexity $\delta\lceil\frac{K}{2\delta}\rceil + 1$ for reshape operations for which no two dimensions requiring communication are adjacent and $K > 2\delta$, and (iii) and one of complexity $\frac{K}{2} + 2\delta - 1$, if there is one unused processor dimension of higher order for every processor dimension requiring communication. The previously best known algorithm has a complexity of $K + \delta - 1$ [6].

The conversion between binary-reflected Gray code and binary code encodings is a special case of reshaping an array, and can be carried out on an n-cube by $n-1$ exchanges in dimensions $0, 1, \cdots, n-2$ in arbitrary order with a complexity of at most $(n-1)\lceil\frac{K}{n-1}\rceil$ element transfers in sequence.

References

[1] *CM-Fortran Release Notes.* Thinking Machines Corp., 1989.

[2] Jean-Philippe Brunet, Danny C. Sorensen, and S. Lennart Johnsson. *A Data Parallel Implementation of the Divide-And-Conquer Algorithm*

for Computing Eigenvalues of Tridiagonal Systems. Technical Report , Thinking Machines Corp., 1989. in preparation.

[3] J.J. Dongarra and D.C. Sorensen. A fully parallel algorithm for the symmetric eigenvalue problem. *SIAM J. Scientific and Statistical Computing,* 8(2):s139–s153, 1987.

[4] 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.

[5] 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.

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

[7] S. Lennart Johnsson. *Optimal Communication in Distributed and Shared Memory Models of Computation on Network Architectures*, page . Morgan Kaufman, 1989.

[8] S. Lennart Johnsson, Tim Harris, and Kapil K. Mathur. Matrix multiplication on the connection machine. In *Supercomputing 89*, page , ACM, November 1989. Department of Computer Science, Yale University, Technical Report YALEU/DCS/RR-736.

[9] S. Lennart Johnsson and Ching-Tien Ho. *Reshaping of Arrays on Boolean Cubes.* Technical Report, Department of Computer Science, Yale University, 1990. in Preparation.

[10] 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.

[11] Michael Metcalf and John Reid. *Fortran 8X Explained.* Oxford Scientific Publications, 1987.

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

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

[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.