THE COMPLEXITY OF REFORMING ARRAYS ON BOOLEAN CUBES

THINKING MACHINES CORP.
CAMBRIDGE, MA

1990

U.S. DEPARTMENT OF COMMERCE
National Technical Information Service
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 axis splittings the exchanges in the different dimensions can be ordered arbitrarily. We present two algorithms that vary in complexity.

**Subject Terms:**
- Boolean algorithms
The Complexity of Reshaping Arrays on Boolean Cubes

S.L. Johnsson, Yale University
C.T. Ho, IBM Almaden Research Center
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 communications. The lower bound for the number of element transfers in sequence is \( \frac{K}{3} \) with \( K \) elements per processor. We present algorithms that are 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 processor.
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

$$\begin{align*}
(paddr^1 \cdot maddr^1)\| & = (g_0, g_1, \ldots, g_{n_1-1}, b_0, b_1, \ldots, b_{n_0-1}) \\
(paddr^2 \cdot maddr^2)\| & = (g_0, g_1, \ldots, g_{n_1-1}, b_0, b_1, \ldots, b_{n_0-1})
\end{align*}$$

The processor address for an element $(i, j)$ of the logic array is formed as $(paddr^1(i))\|paddr^2(j))$, and the local storage address is $(maddr^1(i))\|maddr^2(j))$, where $G_p(i) = (g_0, g_1, \ldots, g_{n_1-1})$ is the binary-reflected Gray code encoding of $i$, and $G_q(j) = (g_0, g_1, \ldots, g_{n_0-1})$ 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

$$\begin{align*}
(paddr^3 \cdot maddr^3)\| & = (g_0, g_1, \ldots, g_{n_1-1}, b_0, b_1, \ldots, b_{n_0-1}) \\
(paddr^4 \cdot maddr^4)\| & = (g_0, g_1, \ldots, g_{n_1-1}, b_0, b_1, \ldots, b_{n_0-1})
\end{align*}$$

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

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 is required. For example, consider the allocation of the form $[4, 13, 12, 15, 10, 5]$ 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 $p.ddr$, where $p.ddr$, $m.ddr$, and $b.ddr$ denote the processor, local storage, and binary encodings, respectively.

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 $i/N_1$ and $j/N_0$. The local memory address is determined from the binary codes of $i$ mod $N_1$ and $j$ mod $N_0$. The encoding of the address field is

$$\begin{align*}
(paddr^5 \cdot maddr^5)\| & = (g_0, g_1, \ldots, g_{n_1-1}, b_0, b_1, \ldots, b_{n_0-1}) \\
(paddr^6 \cdot maddr^6)\| & = (g_0, g_1, \ldots, g_{n_1-1}, b_0, b_1, \ldots, b_{n_0-1})
\end{align*}$$

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 by denning a processor array congruent to the logic array, as for instance in matrix multiplication on the Connection Machine, the combining of two axes into one, can be performed 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. 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 x 2 x ... x 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 \( B_n(i) = (b_{n-1}b_{n-2}...b_0) \) and its binary-reflected Gray code encoding is \( G_n(i) = (g_{n-1}g_{n-2}...g_0) \). \( Z_N = \{0,1,...,N-1\} \) 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 \( K \). \( \mathcal{G}_n \) is the sequence of \( n \)-bit binary-reflected Gray codes for \( Z_N \), i.e., \( \mathcal{G}_n = (G_n(0), G_n(1), \ldots, G_n(2^n - 1)) \).

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

\[
\mathcal{G}_1 = (G_1(0), G_1(1)), \text{ where } G_1(0) = 0, G_1(1) = 1.
\]

\[
\mathcal{G}_{n+1} = \begin{pmatrix}
0|G_n(0) \\
0|G_n(1) \\
0|G_n(2^n - 2) \\
1|G_n(2^n - 1) \\
1|G_n(2^n - 2) \\
\vdots \\
1|G_n(1) \\
1|G_n(0)
\end{pmatrix}
\]

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 \( i \in Z_{N/2} \) are defined by \( G_{n-1}((b_{n-2}b_{n-3}...b_0)) \). The remaining bits in the encoding of \( i \in Z_N - Z_{N/2} \) are defined by \( G_{n-1}((b_{n-2}b_{n-3}...b_0)) \).

Thus,

\[
G_n((b_{n-1}b_{n-2}...b_0)) = \begin{cases}
G_{n-1}((b_{n-2}b_{n-3}...b_0)), & \text{if } b_{n-1} = 0, \\
G_{n-1}((b_{n-2}b_{n-3}...b_0)), & \text{if } b_{n-1} = 1.
\end{cases}
\]

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 \( A[i], i \in Z_N \) into a n-cube, address \( G_n(i) \) contains \( A[i] \).

Lemma 1 [14] \( b_m = g_{m-1} \oplus g_{m-2} \oplus ... \oplus g_m, m \in Z_N \). Conversely, \( g_m = b_m \oplus b_{m+1}, m \in Z_N \) with \( b_m = 0 \).
IUI

nary-reflected Gray code encoded on a Boolean cube.

Figure 2: Reshaping between two arrays with bi-

shape A from IT to V.

\[ C^m = 2^m, m \in \mathbb{Z} (U_i, uU, 2, 2, \ldots, 2), \]

\[ \text{and } G_m(t) = G_m((b_m-1(t)b_m-2(t) \cdots b_0(t))) = \]

\[ (g_m-1(t)g_m-2(t) \cdots g_0(t)) \]

From the binary encoding \(b_j(k) = b_{j+1}(k), j \in \mathbb{Z}_{m-1, m}, b_j(t) = b_j(i), j \in \mathbb{Z}_m, \) by Lemma 1, \( b_j(k) = b_j(t) @ b_{j+1}(k) = b_{j+1}(i) @ b_{j+1}(i) = g_j(i) = g_j(t) \) \( \text{for all } j \in \mathbb{Z}_{m-1}, \) \( b_j(t) = b_j(i) @ b_{j+1}(i) = g_j(i) \) \( \text{for all } j \in \mathbb{Z}_0, \) \( b_j(t) = b_j(i) @ b_{j+1}(i) = \]

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

Hence, if \( b_m(i) = 0 \) then \( G_m(i) = G_{m-1}(i) || G_m(t) \)

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 \( (k, t) \)

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 x 2 x 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 \( (g_m-1g_m-2 \cdots g_0) \) 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 \( G_m(i) = (g_m-1g_m-2 \cdots g_0) \) is \( b_m, \) otherwise it is \( b_m. \)

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

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 \( G_m(i) \) contains element \( A[i], \) \( i \in \mathbb{Z}, \) initially. If all nodes \( i = (b_m-1b_m-2 \cdots b_0) \) such that \( b_m = 1 \) exchange data in dimension \( m-1 \) for any \( m \in \{1, 2, \ldots, n-1\}, \) then node \( G_m(m((b_m-1b_m-2 \cdots b_0)) || G_m((b_m-1b_m-2 \cdots b_0)) \) 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 \( G_m(i) = \)

\[ g_m-1(i)g_m-2(i) \cdots g_0(i) \]

initially contains element \( A[i]. \) Let \( i = k2^m + l, l \in \mathbb{Z}, k \in \mathbb{Z}_{N/2}, \) after the reshape operation element \( i \), now \( (k, t), \) should reside in address \( (G_{m-1}(k)||G_m(t)) \), where \( G_{m-1}(k) = G_m(m((b_m-1b_m-2 \cdots b_0(k))) = \)

\[ (g_{m-1}(k)g_{m-2}(k) \cdots g_0(k)) \]

and \( G_m(t) = G_m((b_m-1(t)b_m-2(t) \cdots b_0(t))) = \)

\[ (g_m-1(t)g_m-2(t) \cdots g_0(t)). \]

The current value of \( b_m \) that is assigned to a given address \( (g_m-1g_m-2 \cdots g_0) \) is easily determined from the address.
Figure 3: Reshaping an array of 8 elements into a $2 \times 2 \times 2$ array.

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.
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 \( g_j \), \( j \in \{m, m+1, \ldots, n-1\} \) 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 \( \left\lceil \frac{K}{n-1} \right\rceil + 25 - 1 \) element transfers in sequence, if for each dimension requiring communication there exists one more significant dimension not requiring communication and \( K \geq 25 \).

Proof:  
The problem is equivalent to sending \( K \) elements along a path of length \( \delta \) and each edge on the path is paired with a length-three path, disjoint with all other edges. If \( \delta \) is even two edge-disjoint paths of length 2\( \delta \) can be defined by combining length-three and length-one paths for different dimensions. If \( \delta \) 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 \( 2\delta \) and one of length 3\( \delta \), and routing \( \left\lceil \frac{K}{n-1} \right\rceil + \delta \) elements along the short route and \( \left\lceil \frac{K}{n-1} \right\lceil - \delta \) elements along the long route the same routing time is achieved if \( K \geq 25 \). For \( K < 26 \), the latter approach degenerates to using a single path of length \( \delta \) and the required time is \( K + \delta - 1 \), which is lower than if two paths of the same length were used. However, if \( K < 26 \) 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 \( 1 \times 16 \) 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 \( \left\lceil \frac{K}{n-1} \right\rceil \) element transfers in sequence.

Proof:  
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 \( K^{n-1} \).

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: \( \{0, 1, \ldots, n-2\} \).

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].

```bash
/* 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
endo
```

The control in the above algorithm is particularly simple, since the following corollary follows from Lemma 3.
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 $m$, $m \in \mathbb{Z}_{n-1}$ 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 $m$, $m \in \mathbb{Z}_{n-1}$ is subject to exchange in dimension $(n - 2 - m - t) \mod (n - 2)$ during step $t$, $t \in \mathbb{Z}_{n-1}$.

/* Converting Gray code to binary code starting from dimension $m$. Dimensions in decreasing order, cyclically*/
if $g_{m-1} \oplus g_{m-2} \oplus \cdots \oplus g_{m+1} = 1$ 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

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 $d$ dimensions requiring communication for the reshape operation. If $K$ is a multiple of $d$, then the number of element transfers in sequence is independent of $d$. An upper bound is $K$ and a lower bound is $K/d$. We present three algorithms: (i) one of communication complexity $d[K/d]$, (ii) one of complexity $d[K/d] + 1$ for reshape operations for which no two dimensions requiring communication are adjacent and $K > 2d$, and (iii) one of complexity $K + 2d - 1$, 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 $K + d - 1$ [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 $0, 1, \ldots, n - 2$ in arbitrary order with a complexity of at most $(n - 1)[K/n-1]$ element transfers in sequence.

References


in preparation.


