On the Conversion between Binary Code and Binary-Reflected Gray Code on Boolean Cubes

On the Conversion between Binary Code and Binary–Reflected Gray Code on Boolean Cubes

S. Lennart Johnsson
Ching-Tien Ho

TR-20-91

July 1991

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

This work has in part been supported by the Air Force Office of Scientific Research under contract AFOSR-89-0382, and in part by NSF and DARPA under contract CCR-8908285.

On the Conversion between Binary Code and Binary–Reflected Gray Code on Binary Cubes

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

Ching-Tien Ho
IBM Research Division
Almaden Research Center
San Jose, CA 95120
Ho@almaden.ibm.com

Abstract

We present a new algorithm for conversion between binary code and binary–reflected Gray code that requires approximately $\frac{2K}{3}$ element transfers in sequence for K elements per node, compared to K element transfers for previously known algorithms. For a binary cube of $n = 2$ dimensions the new algorithm degenerates to yield a complexity of $\frac{K}{2} + 1$ element transfers, which is optimal. The new algorithm is optimal within a factor of $\frac{1}{3}$ with respect to the best known lower bound for any routing strategy. We show that the minimum number of element transfers for minimum path length routing is K with concurrent communication on all channels of every node of a binary cube.

1 Introduction.

Minimizing the required data motion in memory hierarchies has been crucial in achieving high performance almost since the beginning of modern computer technology. In conventional memory hierarchies, minimizing data motion takes the form of preserving temporal and spatial locality of reference in scheduling operations. Massively parallel processors with thousands of processing nodes are currently the only alternative to extreme performance, i.e., a performance of a trillion operations per second and beyond. Scalability of the design, as well as performance, dictates that each processing node has its own memory system, possibly extended with a physically shared memory. In such distributed memory hierarchies, the bandwidth between a processor and its local memory hierarchy is often considerably higher than the bandwidth to the memory units of other processors or shared memory if it exists. The latency is another important performance issue. Our focus is on efficient use of the communications bandwidth.

The communication system in distributed memory hierarchies often represents a considerable fraction of the total system cost. Nevertheless, due to the limited network bandwidth, it is often a bottleneck for performance in many computations. Minimizing the demands on the communication system through proper data placement, such that data frequently used together reside in the same memory unit or adjacent memory units, is important. Once the data placement has been made, it is important to select paths and schedule the data motion such that the routing time (contention) is minimized. Placing the data among the memory units for optimum performance is a very hard problem. Often different allocations are preferable during different phases of the computations. High Performance Fortran [2] is an extension of Fortran-90 in which directives have been added for user control over data placement among memory units. In this paper we address the routing issues in changing the data placement from one common placement in binary cubes, *binary–reflected Gray code* [8], to another common placement, *binary code*, or the converse.

Computations on regular grids constitute an important class of computations in science and engineering. So called explicit methods for finite difference approximations of partial differential equations and many signal and image processing tasks fall into this class. The computations in explicit methods for finite difference approximations are often dominated by the evaluation of the difference approximation, known as difference stencil or difference molecule. The operation is the same as what is typically known as convolution in signal and image processing. Data references are local in Cartesian space, which is used to represent the problem domain for most regular discretizations. However, the fast Fourier transform, FFT, is a powerful algorithm that is used frequently both for the solution of partial differential equations and in signal and image processing. The Cooley-Tukey FFT [1] references data that are adjacent in binary space. Both techniques may be used on the same data set. Thus, if a distributed memory hierarchy allows for an efficient placement with respect to references both in Cartesian and binary space, such a placement may be used. If the memory hierarchy is such that either one or the other reference pattern can be supported, but not both simultaneously, then conversion between the two placements become important. This is the issue addressed in this paper for a collection of memory units interconnected by a binary cube network.

Cartesian grids with axes' lengths being powers of two are subgraphs of binary cubes if the total number of elements is less than, or equal to, the number of nodes in the cube. Binary–reflected Gray codes [8] are often used for the embedding of arrays in binary cubes, since such codes preserve adjacency in Cartesian space. For multidimensional arrays, the Gray code encoding is typically applied to the different axes independently, thereby preserving adjacency along each axis. In this case, distinct subsets of address bits are assigned to different axes. We refer to each such subset as an address field. For arrays with axes' lengths not being powers of two, preserving adjacency forces an expansion of the number of required cube nodes [3]. But, even if the adjacency requirement is relaxed for multidimensional arrays with arbitrary axes' lengths in order to limit the expansion, binary–reflected Gray codes can be used for the embedding of subsections of arrays [5].

A conventional binary encoding of array axes is suitable for many divide–and–conquer algorithms, such as the FFT. In the binary code, adjacency is preserved in the binary space. The binary cube nodes can be labeled such that adjacent nodes differ in precisely one bit.

This paper focuses on the optimal routing of data for conversion between data placements based on binary–reflected Gray codes and binary codes in binary cube networks, in particular in such networks allowing concurrent communication on all channels of every node, *all–port* communication. The Connection Machine models CM–2 and CM–200 [4, 9] are examples of architectures using a binary cube network allowing all–port communication. Our results can easily be modified to systems allowing concurrent communication on some, but not all, channels of every processor. The results also provide interesting insights into the properties of binary–reflected Gray codes. The algorithms require very small temporary storage and the control can be made distributed, i.e., each node can determine its own actions based on its address and the local history of events.

The main new result is an algorithm that for K elements per node of a binary n–cube requires $\lceil \frac{2K-(n-2)}{3} \rceil + (n-2)$ element transfers in sequence with all–port communication. The algorithm is based on pipelining of the element transfers. It yields an improvement of about $\frac{K}{3}$ element transfers in sequence over previous algorithms [6, 7]. The new algorithm is within a factor of $\frac{1}{3}$ of the best known lower bound. It is optimal for $n = 2$. We also present an optimal minimum path length routing algorithm. Though the number of element transfers in sequence is higher than for the nonminimum path length routing algorithm, the minimum path length algorithm may yield fewer communication startups in a packet–switched system with packet sizes of the same order as the local data set. We give lower bounds for two routing strategies: one for using any set of routing paths, one for using only minimum length paths.

The outline of the paper is as follows. We first give some properties of binary cubes and binary–reflected Gray codes, state our assumptions of the communication system, and our objectives for the routing algorithms. Then, we present an optimal minimum path length routing algorithm, followed by the nonminimum path length routing algorithm that is the main result of the paper. We conclude with a summary.

2 Preliminaries.

A binary n–cube has $N = 2^n$ nodes. Two nodes are adjacent iff their addresses differ in exactly one bit. There exist n disjoint paths between any pair of nodes in a binary n–cube. For nodes at distance d, d of those paths are of length d, and $n-d$ paths are of length $d+2$. 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 $G_n(i) = (g_{n-1}g_{n-2}\cdots g_0)$. The processor address bits are $(a_{n-1}a_{n-2}\ldots a_0)$. $\mathcal{Z}_N = \{0, 1, \cdots, N-1\}$ 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 bidirectional channels and concurrent communication on all channels, all–port communication. 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 [8] 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 refer to the binary–reflected Gray code just defined as Gray code. The highest order bit is the same in the binary code and in the Gray code. The remaining bits in the

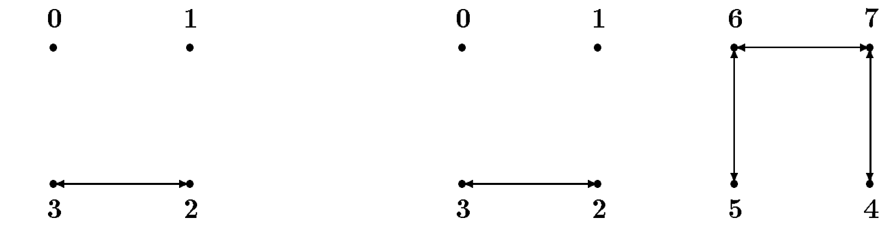

Figure 1: Routing by exchanges in a 2-cube and a 3-cube.

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 $G_{n-1}((\overline{b}_{n-2}\overline{b}_{n-3}\cdots\overline{b}_0))$.

Lemma 1 [8] *$b_m = g_{n-1} \oplus g_{n-2} \oplus \cdots \oplus g_m$, $m \in \mathcal{Z}_n$. Conversely, $g_m = b_m \oplus b_{m+1}$, $m \in \mathcal{Z}_n$ with $b_n = 0$.*

We present all of our results for Gray–to–binary conversion, i.e., conversion from Gray code encoding to binary code encoding. The adaptations of the results to binary–to–Gray conversion is straightforward. We assume that each address field subject to the Gray–to–binary conversion is of length at least two. This assumption avoids the trivial case of a one bit code for which the Gray code encoding is the same as the binary encoding.

Bidirectional communication is assumed for our complexity estimates. All communication complexities are stated in terms of the number of element transfers in sequence. Given $B_n(i)$ and $G_n(i)$, an exclusive–or operation on the two fields determines the cube dimensions through which the data from each node must be routed. The issues we address are:

- routing with constant storage (permutation routing)
- minimizing the congestion
- routing without tags

A routing that can be viewed as a sequence of permutations between neighboring nodes is highly desirable since such a routing conserves memory requirements. Limiting the problem size due to a demand for buffer space is often met with severe criticism from users. The Gray–to–binary conversion has the property that the communication can be performed as a sequence of exchanges, thereby realizing the first objective. This property of the two codes is illustrated in Figure 1 for the cases $n = 2$ and $n = 3$.

Our minimal path length algorithm is based on Theorem 1 below. Before proving the theorem, we prove one critical property of the binary–reflected Gray code.

Lemma 2 *Let node a in a binary n-cube initially contain the element of index $G^{-1}(a)$. For any $m \in \{1, 2, \cdots, n-1\}$, if each node a for which $a_{n-1} \oplus a_{n-2} \oplus \cdots \oplus a_m = 1$ exchanges its element with its neighbor across dimension $m - 1$, then node a contains the element of index $G^{-1}(a_{n-1}\cdots a_m)||G^{-1}(a_{m-1}\cdots a_0)$ after the exchange.*

Proof: It suffices to show that

$$G^{-1}(a_{n-1}\cdots a_m)||G^{-1}(a_{m-1}\cdots a_0) = \begin{cases} G^{-1}(a_{n-1}\cdots a_m \cdots a_0), & \text{if } a_{n-1}\oplus\cdots\oplus a_m = 0, \\ G^{-1}(a_{n-1}\cdots \overline{a_m} \cdots a_0), & \text{otherwise.} \end{cases}$$

Let $G^{-1}(a_{n-1}\cdots a_m)||G^{-1}(a_{m-1}\cdots a_0) = (a'_{n-1}\cdots a'_0)$, let $G^{-1}(a_{n-1}\cdots a_0) = (a''_{n-1}\cdots a''_0)$ and let $G^{-1}(a_{n-1}\cdots \overline{a_m}\cdots a_0) = (a'''_{n-1}\cdots a'''_0)$. It follows from Lemma 1 that $a'_j = a_{n-1}\oplus\cdots\oplus a_j = a''_j = a'''_j$ for $m \le j \le n-1$. For j being in the range $0 \le j \le m-1$, consider two cases.

(i) Case 1: $a_{n-1}\oplus\cdots\oplus a_m = 0$. Then, $a''_j = a_{n-1}\oplus\cdots\oplus a_j = a_{m-1}\oplus\cdots\oplus a_j = a'_j$.

(ii) Case 2: $a_{n-1}\oplus\cdots\oplus a_m = 1$. Then, $a'''_j = a_{n-1}\oplus\cdots\oplus\overline{a_m}\oplus\cdots\oplus a_j = 1\oplus a_{n-1}\oplus\cdots\oplus a_j = a_{m-1}\oplus\cdots\oplus a_j = a'_j$. ▮

Lemma 2 is easily generalized to the splitting of one out of several address fields by observing that the splitting of one address field is independent of other address fields and their encoding.

Theorem 1 *The Gray-to-binary conversion can be performed as a sequence of exchanges in dimensions* $\{0, 1, \cdots, n-2\}$ *taken in arbitrary order.*

Proof: Performing an exchange as in Lemma 2 creates two independently Gray coded address fields. The same splitting procedure can be applied to the created subfields in arbitrary order until there are $n-1$ address fields of one bit each. Since the Gray code and binary code for a one bit field is identical, the proof is complete. ▮

The proof of Lemma 2 provides a way of deriving the exchange control locally. Let m be the current exchange dimension, and let x be the next higher dimension which has already appeared in the exchange sequence. If no such dimension exists, then let $x = n-1$. Then, the current data of node a should be exchanged if and only if $a_{x-1}\oplus\cdots\oplus a_{m+1} = 1$.

Figure 2 gives an example of using an algorithm proceeding from dimension $n-2$ to dimension 0. Initially, processor $G_4(i)$ contains data of index i. After the conversion, i is assigned to processor $B_4(i)$. A pseudocode for the algorithm is given below. Initially, $g_m(i) \rightarrow a_m$, $m \in \mathcal{Z}_{n-1}$ and on termination $b_m(i) \rightarrow a_m$, $m \in \mathcal{Z}_{n-1}$, where $a = (a_{n-1}a_{n-2}\ldots a_0)$, is the processor address as before.

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

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

For the optimal minimum path length routing algorithm we present later, any dimension may be used as a starting dimension. For an arbitrary starting dimension m, $m \in \mathcal{Z}_{n-1}$ with

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 2: Conversion of a Gray code to binary code.

exchanges in successive dimensions of decreasing order, cyclicly, the first exchange requires the computation of the mth bit of $G^{-1}(a)$, which is $a_{n-1} \oplus \cdots \oplus a_{m+1}$. The subsequent steps are similar to the algorithm above. Figure 3 gives an example. Sequence 2 is the same as in Figure 2. The figure shows the location of i for each step of the algorithm for each sequence.

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

3 Lower bounds.

We first consider the case where Gray–to–binary conversion shall be performed on a single processor address field. We then consider the case for multiple processor address fields, where

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 3: Concurrent conversion of a Gray code to binary code.

each field is subject to a Gray–to–binary conversion.

3.1 A single field.

Theorem 2 *With the communication for Gray–to–binary conversion restricted to minimum length paths only, a lower bound is* $\max(K, n-1)$ *element transfers in sequence for a code field of n bits on an n–cube with all–port communication.*

Proof: With K elements per node either all or no elements must be communicated to some other node for code conversion. For any $n \geq 2$ there always exists a pair of nodes at distance one that must exchange data (nodes 2 and 3). Restricting the communication to minimum length paths implies that the code conversion requires at least K element transfers in sequence. ∎

Theorem 3 *A lower bound for the number of element transfers in sequence required for Gray–to–binary conversion on an n–cube with all–port communication is* $\max((1-\frac{1}{n})\frac{K}{2}, n-1)$.

Proof: The total amount of communication required for Gray–to–binary conversion using only minimum length paths is $2\sum_{i=0}^{n-1}\binom{n-1}{i}i = (n-1)2^{n-1}K$. (It follows from the definition of the Gray code that the number of paths of length i is $2\binom{n-1}{i}$.) The number of available edges in an n–cube is $n2^n$. Hence, the lower bound follows. ∎

Theorem 3 does not give a tight lower bound, since the only way in which the full bandwidth of the binary cube can be used is by using nonminimum length paths. For instance, for $n = 2$ Theorem 3 yields the bound $\frac{K}{4}$. A bound based on the number of disjoint paths is $\frac{K}{2}$. An upper bound for $n = 2$ is $\frac{K}{2}+1$. Simply route $\frac{K}{2}+1$ elements along the length one path and $\frac{K}{2}-1$ elements over the length-3 path. Figure 4 illustrates the routing of the data. The bound $\frac{K}{2}+1$ is also a lower bound for $K \geq 2$, since using two paths implies the use of nonminimum path length routing.

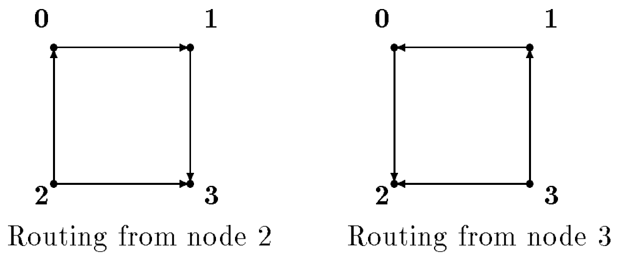

Figure 4: Routing paths in a 2-cube.

3.2 Multiple independent fields.

For the embedding of multidimensional arrays a separate encoding is often used for each address field. With d separately encoded address fields and a total of n dimensions used for the encoding, Theorem 2 is modified as follows.

Theorem 4 *With the communication for Gray–to–binary conversion restricted to paths of minimum length only, a lower bound for the number of element transfers in sequence is* $\max(K, n-d)$ *for all–port communication with a code field of n bits distributed among d axes.*

Proof: There exists a pair of nodes (say the nodes with indices 2 or 3 for one axis, and all other axis indices 0) at distance one that must exchange data. Thus, a time K is required using only minimum length paths. ▮

Theorem 5 *A lower bound for the number of element transfers in sequence for Gray–to–binary conversion on an n–cube with all–port communication is* $\max((1-\frac{d}{n})\frac{K}{2}, n-d)$ *for d address fields separately encoded in n bits.*

Proof: The routing requirements for axis i encoded in n_i bits is $(n_i-1)2^{n_i-1}K$. There are 2^{n-n_i} such subcubes. Hence, the total routing need is $\sum_{i=0}^{d-1}(n_i-1)2^{n-1}K = (n-d)2^{n-1}K$. ▮

4 Algorithms using only minimum length paths.

In this section we use the property that the code conversion can start in an arbitrary dimension to generate several concurrent exchange sequences for all–port communication. The routing algorithm below uses only minimum length paths. It may be advantageous compared to the nonminimum path length algorithm in the next section when there are other simultaneous routing needs. The total load on the communications network is minimal for the minimum path length routing, but the contention is not. The minimum path length routing may also be preferable when there is a significant communications overhead in a packet–switched communications system and the maximum allowable packet size is relatively large. The minimum path length routing described below may yield up to 40% fewer startups than the nonminimum path length routing for cubes of high dimension and a maximum packet size of at least $\frac{K}{n-1}$ elements.

4.1 Conversion of a single address field.

We will first consider code conversion of a single address field. Consecutive and cyclic data allocation [6] of one–dimensional arrays satisfy this constraint. The data allocation can be depicted as follows in terms of the address field:

$$(\underbrace{g_{p-1}g_{p-2}\cdots g_{p-n}}_{\text{paddr}}\underbrace{b_{p-n-1}b_{p-n-2}\cdots b_0}_{\text{maddr}}) \qquad \text{Consecutive}$$

$$(\underbrace{b_{p-1}b_{p-2}\cdots b_n}_{\text{maddr}}\underbrace{g_{n-1}g_{n-2}\cdots g_0}_{\text{paddr}}) \qquad \text{Cyclic}$$

In the consecutive allocation, the processor address field is assigned to the most significant bits of the array index domain. In the cyclic allocation, the processor address field is instead assigned to the least significant bits of the array index domain. In the illustration, it is in both cases assumed that the local memory addresses are encoded in binary code and that the processor address field is encoded in Gray code.

Theorem 6 *The Gray–to–binary conversion for a field of n bits can be attained with* $\max(n-1, K)$ *element transfers in sequence using only minimum length paths and* $(n-1)$*-port communication.*

Proof: The main idea is to generate several independent concurrent exchange sequences that creates a uniform load on all channels used by a minimum path length routing strategy. For the proof we consider three different sets of values of K, the number of local elements. Case 1 is used for most elements for large values of K and is the only relevant case when K is a multiple of $n-1$. Case 2 is used to assure optimal transmission for the remainder of elements when K is large but not a multiple of $n-1$.

Case 1. $K \bmod n-1 = 0$: Create $n-1$ exchange sequences that are different rotations of the dimensions $n-2, n-3, \cdots, 0$. Partition the data set in each processor into $n-1$ sets of the same size and assign one sequence to each data set. If the data set in a node does not require communication in a dimension, then the communication is simply not performed. The number of element transfers in sequence is K.

Case 2. $n-1 < K < 2n-2$: Let $x = K-(n-1)$. Create K exchange sequences that are distinct rotations of the dimensions $n-2, n-3, \cdots, 0, \phi_1, \phi_2, \cdots, \phi_x$, where ϕ_i's are dummy dimensions. No exchange is performed in a dummy dimension. During each step, no dimension in $\mathcal{Z}_{n-1}$ is used by more than one sequence. The number of element exchanges in sequence is K.

Case 3. $K < n-1$: It is easy to see that $n-1$ element transfers in sequence are necessary and sufficient.

For arbitrary $K \geq n-1$, define the routing by partitioning the data set such that cases 1 and 2 apply. ∎

In implementing the algorithm, a table can be set up in each processor such that for each memory partition there is a table entry for each dimension. The entry indicates whether or not a communication shall be performed.

4.2 Conversion of multiple address fields.

For d address fields encoding a total of n bits there are $n-d$ dimensions for which an exchange is required. For $K \bmod (n-d) = 0$, $n-d$ exchange sequences can be generated by applying $n-d$ different rotations to some basic sequence in a manner analogous to the single address field case. The local data set can be divided into $n-d$ partitions with each partition assigned one of the exchange sequences. The cases for $K \bmod (n-d) \neq 0$ can also be handled in a way similar to the case with a single address field to attain the lower bound of $\max(K, n-d)$ element transfers in sequence.

5 Algorithms using nonminimum length paths.

5.1 A single address field.

In order to achieve a time complexity of less than K element transfers in sequence, it is necessary to use nonminimum length paths for the conversion. In the following, we refer to the two subcubes with respect to the most significant dimension as the zero and one subcube, depending upon whether or not the leading dimension is zero or one.

For $n = 2$ a lower bound is $\frac{K}{2} + 1$, and we have already given an optimal algorithm, Figure 4. The critical observation for the algorithm was that of the two edges in dimension 0, only one being used for the minimum path length routing and none of the edges in dimension 1 being used. Hence, routing part of the data to the subcube containing the unused edge in dimension 0 (the zero subcube), performing the code conversion for part of the data in that subcube, then routing the converted data back allows the time to be reduced, by reducing the contention.

By considering the routing paths in Figure 1, it can be observed that if the 3–cube is collapsed into a 2–cube along dimension 2 by identifying nodes 0 and 7, 1 and 6, 2 and 5 and 3 and 4, then all bidirectional links in the resulting 2–cube would be used evenly. Hence, by exchanging data between the zero and one subcubes, all edges in both subcubes can be used for the code conversion. For the case $n = 2$ there is no exchange between the zero and one subcubes, since there is no routing requirement in the zero subcube. For $n = 3$ the edges between nodes 2 and 5, and 3 and 4, respectively, are used in both directions for data from both the zero and the one subcubes. All other edges are used for "single" exchanges.

We will now prove that the routing paths for Gray–to–binary conversion always can be determined such that if an n–cube is collapsed into a $(n-1)$–cube along the most significant dimension, all edges are used evenly in the collapsed cube.

Theorem 7 *If the routing paths for the Gray–to–binary conversion traverse the cube dimensions in ascending order, then an edge is used for an exchange in subcube zero, iff the corresponding edge in subcube one is not used.*

Proof: Let the exchange sequence for the Gray–to–binary conversion proceed through the cube dimensions in the order $0, 1, \cdots, n-2$. Let the first exchange step be step 0. From Lemma 2, an exchange in step 0 is performed for all nodes such that $a_{n-1} \oplus \cdots \oplus a_1 = 1$. It is easy to show that the exchange in step i, $0 \leq i \leq n-2$, is determined by $a_{n-1} \oplus \cdots \oplus a_{i+1}$. Since

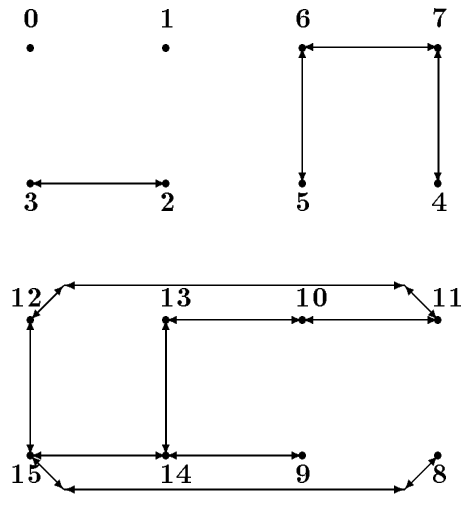

Figure 5: Routing for code conversion in a 4–cube.

a_{n-1} is always in the expression determining whether or not a node shall exchange data, only one node in a pair differing in their most significant bit exchange data in any of the dimensions $0, 1, \ldots, n-2$. Thus, an edge is used for an exchange in subcube zero iff the corresponding edge in subcube one is not used. ▮

Figure 5 shows the edges used for Gray–to–binary conversion with routing paths defined by routing in dimensions of ascending order. Note that by folding the 4–cube onto a 3–cube, all edges in the 3–cube are used evenly in the same way as all edges are used evenly if a 3–cube is folded onto a 2–cube. For binary to Gray code conversion, the same property holds for an exchange sequence using the cube dimensions in descending order.

The basic idea of our algorithm requiring approximately $\frac{2K}{3}$ element transfers in sequence is as follows.

1. Divide the local data set into two sets: the first set S_1 of size $M \approx \frac{2K}{3}$ and the second set S_2 of size $M' = K - M \approx \frac{K}{3}$. The larger data sets S_1 are routed over minimum length routes, *short routes*. The smaller data sets S_2 are routed over nonminimum length paths, *long routes*, by first routing the data to the node obtained by complementing the most significant dimension.

2. Perform the following two algorithms concurrently for each node i in communicating the data sets S_1 and S_2 to their destination node $G^{-1}(i)$.

   (a) Route each element in S_1 along an ascending order of cube dimensions $0, 1, \cdots, n-2$, as needed. Pipelining the M elements results in a total of $M + n - 2$ element transfers in sequence.

   (b) Route each element in S_2 along cube dimensions $n-1, 0, 1, \cdots, n-2, n-1$. Note that dimension $n-1$ needs to be routed twice for each element since routing in this dimension is not part of the code conversion. Pipelining is applied in the following manner, assuming $M' \geq n$.

- Pipelining the M' elements along dimensions $n-1, 0, 1, \cdots, n-2$ results in a total of $M' + n - 1$ element transfers in sequence.
- Initiating the final exchange along dimension $n-1$ for the M' elements after M' time steps of the previous procedure results in M' time steps with $n-1$ steps overlapped with the previous procedure.

Note that the set of short routes is edge–disjoint from the set of long routes, thus allowing for the element transfers to be pipelined without contention. Before determining the optimum size of the small and large data sets, we state the result.

Theorem 8 *The Gray–to–binary conversion requires at most* $\lceil \frac{2K-(n-2)}{3} \rceil + (n-2)$ *element transfers in sequence with all–port communication for* $K > n+3$*. The number of element transfers in sequence for* $K \leq n+2$ *is* n*.*

For a precise optimization of the number of element transfers in sequence, we note that with M elements assigned to each short route, the time for these routes, $T_s(M)$, satisfies the relation

$$T_s(M) \leq M + n - 2.$$

One simple scheduling for the long routes, which are subject to contention in dimension $n-1$, is to pipeline the transfers of the elements across dimension $n-1$ with the routing required for code conversion in the complemented subcube. The converted data remains in the complemented subcube until the edges in dimension $n-1$ become free, when the converted data is brought back to the originating subcube. Thus, for M' elements assigned to long routes,

$$T_l(M') \leq M' + \max(M', n)$$

The optimal partition of the data set K, is determined by $T_s(M) = T_l(K-M)$. This equality yields $M = \lceil \frac{2K-(n-2)}{3} \rceil$ for $K > n+2$, and $M = 2$ for $K \leq n+2$.

$$T_{min} = \begin{cases} \lceil \frac{2K-(n-2)}{3} \rceil + (n-2) & K > n+2 \\ n & K \leq n+2 \end{cases}$$

5.2 Multiple address fields.

With Gray–to–binary conversion required on multiple address fields, the number of element transfers in sequence is almost identical to code conversion for a single address field encoded in the same number of bits. For the routing, the local data sets can be divided into as many partitions as there are axes. The code conversion for the different partitions can be performed concurrently. All partitions must undergo code conversion for each axis.

5.3 Binary–to–Gray conversion.

A binary–to–Gray conversion is obtained by running our algorithms backwards. Thus,

Theorem 9 *The binary–to–Gray conversion requires at most* $\lceil \frac{2K-(n-2)}{3} \rceil + (n-2)$ *element transfers in sequence with all–port communication for* $K > n+3$. *The number of element transfers in sequence for* $K \leq n+2$ *is* n.

The theorem follows from Theorem 8 by using descending order routing for the code conversion and combining routing along short and long routes as in the Gray–to–binary conversion case.

6 Summary.

We have given an algorithm for the conversion between binary–reflected Gray code and binary code that requires approximately $\frac{2K}{3}$ element transfers in sequence. The algorithm offers a reduction in the number of element transfers in sequence by about $\frac{K}{3}$ compared to previously known algorithms. For $n = 2$, the new algorithm is optimal for any routing strategy. For larger values of n, it is nonoptimal by at most a factor of $\frac{1}{3}$ compared to the best known lower bound, $\frac{K}{2} + 1$.

We have also given an optimal minimum path length routing algorithm that for K elements per node and concurrent communication on all channels of every processor in a binary n-cube requires $\max(K, n-d)$ element transfers in sequence. The algorithm has $n-2$ element transfers less than a pipelined algorithm [6], when $K \geq n-1$.

The Connection Machine models CM–2 and CM–200 are distributed memory architectures which allow concurrent communication on all channels of every node, with the nodes interconnected as a binary cube. The code conversion routine on the Connection Machine was implemented before the nonminimum path length routing algorithm was discovered. The existing routine use the pipelined algorithm in [6]. The results here show that the code conversion time on the Connection Machine can be sped up by as much as 50%.

For a packet–switched communication system with a maximum packet size B, the minimum path length routing requires $\lceil \frac{K}{(n-1)B} \rceil (n-1)$ startups. The nonminimum length path routing requires $\max(\lceil \frac{2K}{3B} \rceil + n - 2, 2\lceil \frac{K}{3B} \rceil + n - 1)$ startups. For $B = \lceil \frac{K}{n-1} \rceil$, the minimum length path routing requires $n-1$ startups, while the nonminimum path length routing requires approximately $2\lceil \frac{2(n-1)}{3} \rceil + n - 1$ startups. For large n, the ratio of the number of startups between minimum path length routing and nonminimum path length routing approaches 0.6. For $\frac{K}{B} \gg 1$, the ratio instead approaches 1.5.

Acknowledgment

This work has in part been supported by the Air Force Office of Scientific Research under contract AFOSR-89-0382, and in part by NSF and ARPA under contract CCR-8908285.

References

[1] James C. Cooley and J.W. Tukey. An algorithm for the machine computation of complex fourier series. *Math. Comp*, 19:291–301, 1965.

[2] High Performance Fortran Forum. High performance fortran language specification, version 0.4. Technical report, Department of Computer Science, Rice University, November 1992.

[3] I. Havel and J. Móravek. B-valuations of graphs. *Czech. Math. J.*, 22:338–351, 1972.

[4] W. Daniel Hillis. *The Connection Machine.* MIT Press, Cambridge, MA, 1985.

[5] Ching-Tien Ho and S. Lennart Johnsson. Embedding meshes in Boolean cubes by graph decomposition. *J. of Parallel and Distributed Computing*, 8(4):325–339, April 1990.

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

[7] S. Lennart Johnsson and Ching-Tien Ho. The complexity of reshaping arrays on Boolean cubes. In *The Fifth Distributed Memory Computing Conference*, pages 370–377. IEEE Computer Society, April 1990.

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

[9] Thinking Machines Corp. *CM–200 Technical Summary*, 1991.