An Efficient Algorithm for Gray-to-Binary Permutation on Hypercubes

An Efficient Algorithm for Gray–to–Binary Permutation on Hypercubes

Ching-Tien Ho
S. Lennart Johnsson
M.T. Raghunath

TR-20-92

September 1992

Parallel Computing Research Group

Center for Research in Computing Technology
Harvard University
Cambridge, Massachusetts

An Efficient Algorithm for Gray–to–Binary Permutation on Hypercubes

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

M.T. Raghunath
Computer Science Division
University of California at Berkeley
Berkeley, CA 94720
mtr@cs.berkeley.edu

S. Lennart Johnsson
Harvard University and
Thinking Machines Corp.
Cambridge, MA
johnsson@harvard.edu

Abstract

Both Gray code and binary code are frequently used in mapping arrays into hypercube architectures. While the former is preferred when communication between adjacent array elements is needed, the latter is preferred for FFT-type communication. When different phases of computations have different types of communication patterns, the need arises to remap the data. We give a nearly optimal algorithm for permuting data from a Gray code mapping to a binary code mapping on a hypercube with communication restricted to one input and one output channel per node at a time. Our algorithm improves over the best previously known algorithm [6] by nearly a factor of two and is optimal to within a factor of $n/(n-1)$ with respect to data transfer time on an n-cube. The expected speedup is confirmed by measurements on an Intel iPSC/2 hypercube.

1 Introduction

The availability of run-time systems and libraries supporting a shared address space, such as Express (by ParaSoft) [5], or a shared memory programming model, such as Linda [2], for programming distributed memory architectures, makes parallel programming transparent with respect to the physically distributed memory. The utility of these programming systems depends critically on the efficient implementation of the underlying communication primitives [1, 4, 7, 13] on each individual architecture.

Consider a one–dimensional array partitioned into N blocks allocated evenly to the nodes of an $N = 2^n$ node hypercube. For the binary–code mapping of the N blocks, block

i, where $0 \leq i < N$, is allocated to node i. For the Gray code mapping, block i is allocated to node $G(i)$, the i-th code in the Gray code. For instance, for a binary–reflected Gray code [11] mapping on a 3–cube, blocks 0 through 7 are allocated to nodes $0, 1, 3, 2, 6, 7, 5$ and 4, respectively. Thus, block i is adjacent to blocks $i \oplus 2^j$ for all $0 \leq j < n$ with the binary–code mapping, and is adjacent to blocks $(i-1) \bmod N$ and $(i+1) \bmod N$ with the Gray code mapping. While the adjacency offered by the binary–code mapping is preferred for FFT–type communications, the adjacency offered by the Gray code mapping is preferred for communications between neighboring blocks. The need to change between these two types of mappings arises when different phases of a computation exhibit different data reference patterns.

In this paper, we give two practical algorithms for the permutation of a one–dimensional array from a binary–reflected Gray code mapping to binary code mapping (termed Gray–to–binary permutation). The new algorithms apply to communication systems restricted to communication on one channel per node at a time, known as *one–port* communication. The fastest of the two new algorithms improves upon the data transfer time of the *one–port* algorithm in [6] by a factor of $2(n-1)/n$ at the expense of one additional communication step. Implementation results on a 64 node iPSC/2 confirm our complexity analysis.

2 Preliminaries

2.1 Notation and definitions

$N = 2^n$ is the size of the hypercube. The least significant bit is the 0th bit. Subcube 0_j is the set of nodes whose jth bit is zero. Subcube 1_j is similarly defined. Let $i = (i_{n-1} i_{n-2} \cdots i_0)$ and $j = (j_{n-1} j_{n-2} \cdots j_0)$, then the Hamming distance between i and j is $Hamming(i,j) = \sum_{m=0}^{n-1} (i_m \oplus j_m)$, where "$\oplus$" is the bitwise exclusive–or operator. The concatenation symbol is "||".

2.2 Communication model

In a distributed memory multiprocessor, nodes communicate with each other by sending and receiving messages. We denote the overhead associated with each internode communication (send or receive) by τ, and the data transfer time per byte by t_c. We assume that each communication channel between a pair of nodes can transmit data in both directions *at the same time*. In an n–dimensional hypercube, each node has n output and n input ports. We assume the *one–port* communication model, in which only one output port *and* one input port per node can be active at a given time. In the *all–port* communication model each node can send and receive messages concurrently on all its ports. All–port Gray–to–binary permutation algorithms are given in [6, 10]. The communication time for sending an m byte message to a nearest neighbor is $T = \tau + mt_c$. We refer to the term associated with t_c as the "data transfer time", and to the term associated with τ

as the "startup time". The number of bytes per node subject to the Gray–to–binary permutation is K.

We do not make any assumption as to whether the router of the hypercube supports store–and–forward, circuit–switched, wormhole, or virtual cut–through routings. Our algorithms use only nearest–neighbor (in fact, one–dimension–at–a–time) communications. However, the complexity comparisons and the derivation of the lower bound on the time complexity are based on store–and–forward routing.

2.3 Gray–to–binary permutation

Let $\hat{G}_n$ be the sequence of n–bit binary–reflected Gray codes, i.e., $\hat{G}_n = (G_n(0), G_n(1), \cdots, G_n(2^n - 1))$.

Definition 1 [11] The *binary–reflected Gray code* can be defined recursively as follows:

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

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

For convenience, we refer to the binary–reflected Gray code just defined simply as Gray code. From the above definition, the following well-known lemma can be derived.

Lemma 1 [11] *Let* $i = (i_{n-1} i_{n-2} \cdots i_0)$, $i_n = 0$ *and* $G(i) = (g_{n-1} g_{n-2} \cdots g_0)$. *Then,* $i_m = g_{n-1} \oplus g_{n-2} \oplus \cdots \oplus g_m$ *and* $g_m = i_{m+1} \oplus i_m$ *for all* $0 \leq m < n$.

2.4 Previous algorithms

In [6], Johnsson gives an algorithm for Gray–to–binary permutation consisting of $n - 1$ exchanges along the sequence of cube dimensions $n-2, n-3, \cdots, 0$. Johnsson also shows that for *all–port* communication, pipelining of the communication steps can be used to reduce the communication complexity by a factor of n. In [8] Johnsson gives algorithms for Gray–to–binary and binary–to–Gray permutation with exchanges proceeding in both ascending and descending order of dimensions. In [9], Johnsson and Ho show that the

permutation can be realized by exchanges in cube dimensions $\{0, 1, \cdots, n-2\}$ in arbitrary order. Later, this was also proved by Edelman, Heller and Johnsson using an algebraic framework [3].

Algorithm GB1 [6] performs $n-1$ exchanges in ascending order of cube dimensions [8], i.e., along the sequence of cube dimensions $0, 1, \cdots, n-2$. We refer to the $n-1$ exchange steps as step 0 through step $n-2$, where during step i exchange operations are performed for a subset of edges in dimension i. The subset is determined as follows. Let g_i^{-1} be the i-th bit of $G^{-1}(pid)$, the inverse Gray code of the node address, pid. Then, during step i, an exchange is performed for nodes whose g_{i+1}^{-1} value is 1.

Figure 1 shows the three exchange steps of Gray–to–binary permutation on a 4–cube. In the figure, a 4–cube annotated with "step i, dimension i" is the scenario right before exchange step i. The arrows show the subset of edges in dimension i that are subject to an exchange operation across dimension i during the next step. The number at a node is the rank of the block allocated to the node initially (by Gray code mapping). The communication complexity for Algorithm GB1 [6] is

$$T_{\mathrm{GB1}} = (n-1)(\tau + Kt_c). \tag{1}$$

3 Gray–to–binary permutation

3.1 Lower bound

We now derive the lower bound for the Gray–to–binary permutation with respect to a store–and–forward routing. We first compute the (Hamming) distance distribution between all n–bit binary strings and their corresponding Gray codes. From the distribution, the required communication bandwidth can be computed.

Let $S(n, j)$ be the number of n–bit strings for which the binary and Gray code mapping is a Hamming distance j apart, i.e., $S(n, j)$ is the cardinality of the set $\{i | Hamming(i, G(i)) = j, 0 \leq i < 2^n\}$.

Define $\binom{n}{j} = 0$ if $j < 0$ or $j > n$.

Lemma 2 *$S(n, j) = 2\binom{n-1}{j}$ for all $n \geq 2$ and $0 \leq j \leq n$.*

Proof: There are 2^n different n–bit strings. Thus, for $n \geq 2$, if we show that $S(n, j) = 2\binom{n-1}{j}$ for $0 \leq j < n$, then it follows that $S(n, n) = 0$ (because $\sum_{j=0}^{n-1} 2\binom{n-1}{j} = 2^n$).

We prove that $S(n, j) = 2\binom{n-1}{j}$ by induction on n. For the basis $n = 2$, we have $S(2, 0) = S(2, 1) = 2$. For the induction hypothesis, we assume $S(k, j) = 2\binom{k-1}{j}$ for all $0 \leq j \leq k-1$. From Definition 1, we have $S(k+1, j) = S(k, j) + S(k, j-1)$, which, by the induction hypothesis, yields $2\binom{k-1}{j} + 2\binom{k-1}{j-1} = 2\binom{k}{j}$. This observation completes the proof. ∎

From Lemmas 1 and 2, $i = G(i)$ if and only if $i = 0$ or $i = 1$. Also, if $Hamming(i, G(i)) = n-1$, then $i = (11\cdots1)$ and $G(i) = (100\cdots0)$, or $i = (11\cdots10)$ and $G(i) = (100\cdots01)$. Thus, each of these two nodes must send their data to a node at a distance of $n-1$ in the Gray–to–binary permutation. It is easy to derive a lower bound from the preceding lemma as follows.

Lemma 3 [9] *The lower bound for Gray–to–binary permutation with respect to a store–and–forward routing and a one–port communication model is* $\max((n-1)\tau, (n-1)\frac{K}{2}t_c)$.

Proof: Clearly, the start–up time is at least $(n-1)\tau$, because the maximum Hamming distance between i and $G(i)$ is $n-1$. The total number of element transfers required by the permutation is

$$\begin{aligned}\sum_{j=0}^{n} j * S(n,j) * K &= \sum_{j=0}^{n} 2j\binom{n-1}{j}K \\ &= 2(n-1)K\sum_{j=0}^{n-2}\binom{n-2}{j} \\ &= (n-1)2^{n-1}K.\end{aligned}$$

In each step up to 2^n directed links can be used in the *one–port* communication model. Thus, at least a time of $(n-1)\frac{K}{2}t_c$ is needed for transferring the data. ∎

3.2 Algorithm GB2

We now introduce two new algorithms for Gray–to–binary permutation for *one–port* communication. The algorithms exchange half of the local data set per step at the expense of two additional communication steps for Algorithm GB2 compared to Algorithm GB1, and only one additional step for Algorithm GB3. Although Algorithm GB2 is always inferior to Algorithm GB3, the description of GB2 facilitates the understanding of GB3.

Both Algorithm GB2 and GB3 are based on the observation that Algorithm GB1 uses only half of the communication channels in dimensions $0, 1, \ldots, n-2$. In Figure 1, consider any two adjacent nodes i and j in subcube 0_3, and the corresponding two nodes i' and j' in subcube 1_3. Let i and j differ in the dth bit, where $0 \le d \le 2$. If an exchange is performed between nodes i and j in step d, then there is no exchange between nodes i' and j'. Similarly, if an exchange is needed between nodes i' and j' during step d, then there is no exchange between nodes i and j during the same step. Thus, exactly one of the two pairs (i, j) and (i', j') performs an exchange along dimension d during step d [10]. The property can be stated as follows:

Lemma 4 [10] *In Algorithm GB1 for the Gray–to–binary permutation, if two nodes exchange their data in subcube* 0_{n-1} *(respectively,* 1_{n-1}*), then the corresponding two nodes in subcube* 1_{n-1} *(respectively,* 0_{n-1}*) do not exchange data during the same step.*

Proof: In Algorithm GB1, the value of bit $g_{j+1}^{-1}(i)$ determines if node i must exchange its data with node $i \oplus 2^j$ during step j, for all $0 \leq j \leq n-2$. Thus, we only need to show that $g_{j+1}^{-1}(i) \oplus g_{j+1}^{-1}(i \oplus 2^{n-1}) = 1$ for all $0 \leq j \leq n-2$. Let $g_{j+1}^{-1}(i) = x$ and $g_{j+1}^{-1}(i \oplus 2^{n-1}) = y$. Then, by Lemma 1, $x = i_{n-1} \oplus i_{n-2} \oplus \cdots \oplus i_{j+1}$ and $y = \bar{i}_{n-1} \oplus i_{n-2} \oplus \cdots \oplus i_{j+1}$. Thus, $x \oplus y = 1$. ∎

Lemma 4 implies that only half of the edges in dimension $0, 1, \ldots, n-2$ are used for the permutation. This lemma was used in [10] to devise an *all–port* algorithm requiring $\frac{2}{3}K$ element transfers, an improvement over the algorithm in [6] by a factor of $\frac{1}{3}K$.

In the following, we refer to the $n-1$ exchanges in Algorithm GB1 as *normal–exchanges*. In Algorithm GB2 each node first exchanges half its data along dimension $n-1$. Then, normal–exchanges are performed along dimensions $0, 1, \cdots, n-2$ as in Algorithm GB1, except that when a node in subcube 0_{n-1} would not have exchanged data in Algorithm GB1, it exchanges the data it received from its neighbor in cube 1_{n-1} and vice versa. Hence, each subcube not only performs the Gray–to–binary permutation for half of its data, but also performs the permutation for half of the data of the other subcube. Finally, an exchange along dimension $n-1$ undoes the first exchange. We refer to the exchanges before and after the normal–exchanges as *pre–exchange* and *post–exchange*, respectively. These exchanges allow all edges of the hypercube in dimension $0, 1, \ldots, n-2$ to be used in each step. Further, during each step only half of the local data set is exchanged. Thus, the communication complexity is

$$T_{\mathrm{GB2}} = (n+1)(\tau + \frac{K}{2}t_c). \tag{2}$$

Figure 2 shows an example of Algorithm GB2 on a 4–cube. In the two 4–cubes annotated with pre–exchange and post–exchange, an arrow between subcube 0_3 and subcube 1_3 represents exchanges across all edges in dimension 3. Each data block, say of rank i, of the global array is partitioned into two subblocks of size $K/2$ each, denoted i and i', respectively, in the figure.

3.3 Algorithm GB3

Algorithm GB3 is similar to Algorithm GB2, except that the pre–exchange and post–exchange steps are performed in dimension $n-2$, i.e., the exchanges are made in dimensions $n-2, 0, 1, \cdots, n-2$. We show later that, by doing the pre– and post–exchanges in dimension $n-2$, the post–exchange step can be combined with the last step of the normal–exchanges. The size of the combined data set remains $K/2$.

Formally, consider the four subcubes $00, 01, 10$ and 11 defined by dimensions $n-1$ and $n-2$. For each of the first $n-2$ normal–exchange steps of GB1 (i.e., on dimension $0, 1, \cdots, n-3$), if there is an exchange between nodes i and j in subcube 00 (01, 10 and 11, respectively), then there is no exchange between their corresponding nodes in subcube 01 (00, 11 and 10, respectively). By exchanging half of the data across dimension $n-2$ before and after these $n-2$ exchanges, only $K/2$ elements need to be exchanged for

each of these $n-2$ normal–exchanges. To complete the Gray–to–binary permutation, the normal–exchange step of Algorithm GB1 in dimension $n-2$ must also be performed. We now show that the post–exchange and the normal–exchange in dimension $n-2$ can be combined into one exchange of $K/2$ elements. For subcube 0_{n-1}, there is no normal–exchange needed in dimension $n-2$ for Algorithm GB1 (i.e., if no pre–exchange had been made). For subcube 1_{n-1}, the post–exchange requires that $K/2$ elements be exchanged in dimension $n-2$, while for Algorithm GB1 (no pre–exchange), K elements must be exchanged in the normal–exchange in dimension $n-2$. Thus, in Algorithm GB3, due to the pre–exchange in dimension $n-2$, only $K/2$ elements are subject to exchange in dimension $n-2$ in both the 0_{n-1} and 1_{n-1} subcubes, in the last exchange step.

Figure 3 shows an example of the algorithm (GB3) on a 4–cube. The complexity of the algorithm is

$$T_{\mathrm{GB3}} = n(\tau + \frac{K}{2}t_c). \tag{3}$$

3.4 Complexity comparison

From Figure 4-(a) it can be seen that the estimated performance of Algorithm GB3 is better than that of Algorithm GB1 for large data sets. The estimated performance of Algorithm GB3 is always better than that of Algorithm GB2. To demonstrate the feasibility of our algorithms, we implemented all three on a 64 node Intel iPSC/2. Although the Intel iPSC/2 uses circuit–switched routing, we only use nearest–neighbor communication for our implementation. Figure 4-(b) compares the measured times on the Intel iPSC/2 for Algorithms GB1, GB2 and GB3. All measured times on the Intel iPSC/2 are averaged over at least 100 runs. The improvement of Algorithm GB3 over Algorithm GB1 increases as the message size increases and approaches $2(n-1)/n$, asymptotically.

For small data sets, however, Algorithm GB1 performs better than Algorithm GB3 because it needs $n-1$ communication steps instead of n steps. The measured break–even point for the 64 node iPSC/2 is at about $K = 485$ bytes, while the complexity estimates predict a break–even around $K = 1.4$ kbytes.

The speedup of Algorithm GB3 over Algorithm GB1 is expected to increase as the cube dimension increases. Figure 5 compares the measured times of the three algorithms on the iPSC/2 as a function of cube dimensions. The measured speedup of Algorithm GB3 over Algorithm GB1 increases from 1.63, 1.67 to 1.79 as the dimension increases from 4 to 6.

3.5 Circuit–switched routing

All the Gray–to–binary permutation algorithms discussed so far only use nearest–neighbor communication and thus do not assume a particular routing policy. From Algorithm GB1, one can easily observe that the paths from all nodes i to their corresponding destinations $G^{-1}(i)$ are edge–disjoint (in the directed edge sense). Thus, on a circuit–switched hypercube, such as the Intel iPSC/2, all data can be sent directly to their corresponding

destinations with one communication startup without congestion. (In fact, by [9] and [12], the congestion–free property remains true for any routing with any fixed order of hypercube dimensions.)

For circuit–switched routing, where each node can send and receive at least two messages at a time, it is possible to apply our technique to achieve a better complexity than for a direct–route. The complexity of sending a K byte message without congestion is $\tau + Kt_c$ in a circuit–switched routing, regardless of the number of hops that must be traversed. We partition each local data set into 3 blocks of equal size, called blocks 0, 1 and 2. In the first step, each node i sends block 0 to node $G^{-1}(i)$ and concurrently sends block 1 to node i', where i' is the neighbor of i across dimension $n-1$. In the second step, node i sends block 2 to node $G^{-1}(i)$, and concurrently node i' forwards block 1 of node i to node $G^{-1}(i)$. (The description is made from the viewpoint of the data originating in node i.) The same idea was used in [10] for an all–port algorithm. It can be easily shown that for both steps there is no congestion between the blocks of size $K/3$ each. Thus, the total time is $2\tau + \frac{2K}{3}t_c$ (compared to $\tau + Kt_c$ for the direct–route alternative).

4 Concluding remarks

We have presented two algorithms for the permutation between binary–reflected Gray code mapping and binary code mapping on hypercubes. Algorithm GB3 improves upon the data transfer time of the previous algorithm for *one–port* communication [6] (Algorithm GB1) by a factor of $2(n-1)/n$ on an n–cube, at the expense of one additional startup. The data transfer time of the new algorithm is optimal within a factor of $n/(n-1)$ on an n–cube with store–and–forward routing. The break–even point is measured at about 485 bytes on a 64 node iPSC/2. The improvement increases as the message size increases, the cube size increases, or the startup cost decreases. For a 64 kbyte message and a 64 node iPSC/2, the measured time of the improved algorithm is about 56% of the previous algorithm. For a circuit–switched hypercube, where each node can send and receive at least two messages at a time, our technique can be applied to achieve a communication complexity of $2\tau + \frac{2K}{3}t_c$ (compared to $\tau + Kt_c$ for a naive routing). With all–port communication (such as on the Connection Machine systems CM–2 and CM–200 [14]), the algorithms discussed in [10] can be used. Finally, it should be noted that although all algorithms were described for Gray–to–binary permutation, corresponding algorithms of the same complexities for binary–to–Gray permutation can be easily derived (including the model of circuit–switched, 2–port and routing in ascending order of cube dimensions (e–cube routing)).

Acknowledgments

The authors wish to acknowledge that an algorithm similar to our Algorithm GB2 has independently been proposed by Steve Heller of Thinking Machines Corp. We also wish to thank Oak Ridge National Laboratories and Michael Leuze, Director of the Parallel Computation Facility, for providing access to a 64 node Intel iPSC/2. The work by

Lennart Johnsson has, in part, been supported by AFOSR grant AFOSR-89-0382 with Yale and Harvard Universities, and in part by NSF and DARPA under contract CCR-8908285 also with Yale and Harvard Universities.

References

[1] D. P. Bertsekas, C. Ozveren, G.D. Stamoulis, P. Tseng, and J.N. Tsitsiklis. Optimal communication algorithms for hypercubes. *Journal of Parallel and Distributed Computing*, 11:263–275, 1991.

[2] Nicholas Carriero and David Gelernter. Linda in context. *Comm. of ACM*, 32(4):444–458, April 1989.

[3] Alan Edelman, Steve Heller and S. Lennart Johnsson. Index Transformation Algorithms in a Linear Algebra Framework. Thinking Machines Corp. Technical Report TMC-223, March 1992.

[4] Geoffrey C. Fox and Wojtek Furmanski. Optimal communication algorithms for regular decompositions on the hypercube. In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 648–713. ACM, 1988.

[5] K. Ikudome, G.C. Fox, A. Kolawa, and J.W. Flower. An automatic and symbolic parallelization system for distributed memory parallel computers. In *The Fifth Distributed Memory Computing Conference*, pages 1105–1114. IEEE Computer Society, April, 1990.

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

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

[8] S. Lennart Johnsson. Optimal Communication in Distributed and Shared Memory Models of Computation on Network Architectures. In *Models of Massively Parallel Computation*. Morgan Kaufmann, San Mateo, CA, 1990, pages 223-389.

[9] 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.

[10] S. Lennart Johnsson and Ching-Tien Ho. On the conversion between binary code and binary–reflected Gray code. Technical Report TR-20-91, Harvard University, July 1991.

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

[12] Arch D. Robison. A group of permutations with edge-disjoint paths on hypercubes. In *The Third IEEE Symp. on Parallel and Distributed Processing*, pages 746–749. IEEE, 1991.

[13] Quentin F. Stout and Bruce Wagar. Intensive hypercube communication: Prearranged communication in link-bound machines. *Journal of Parallel and Distributed Computing*, 10:167–181, 1990.

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

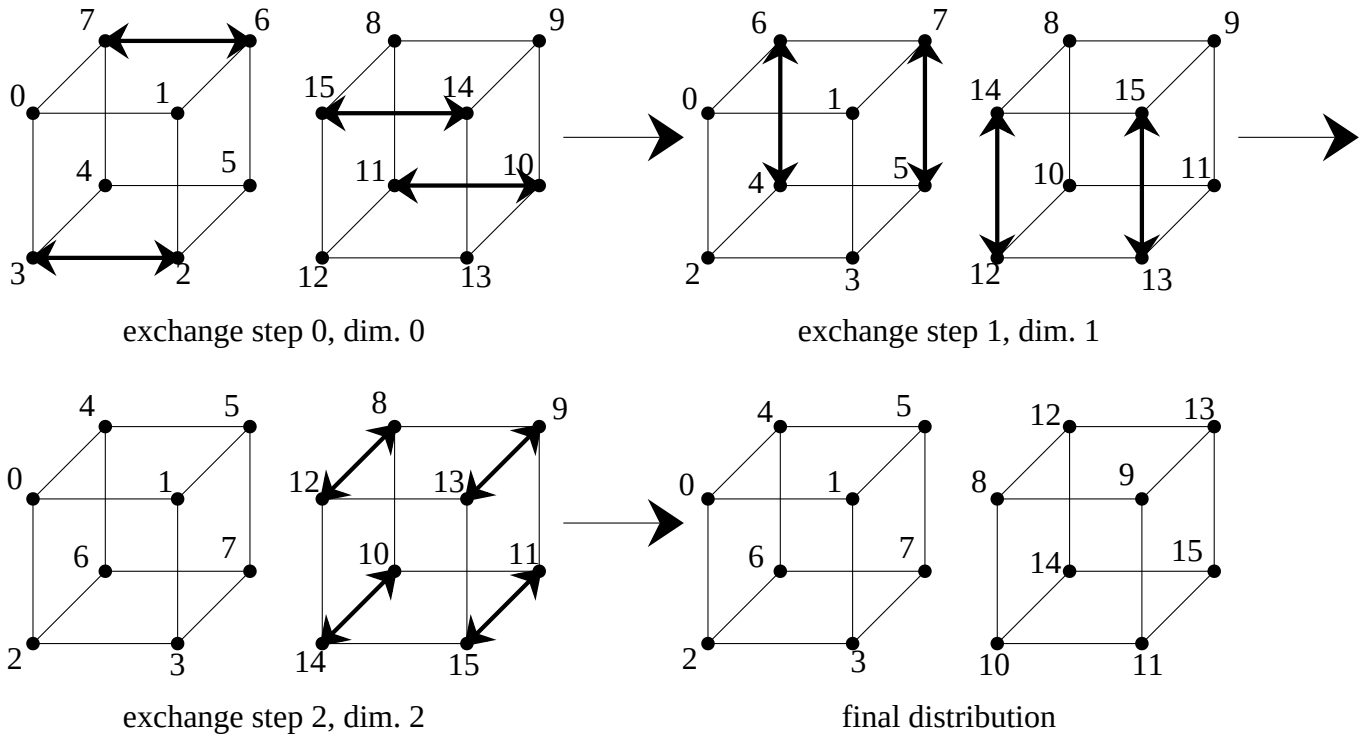

Figure 1: An example of Algorithm GB1 on a 4–cube.

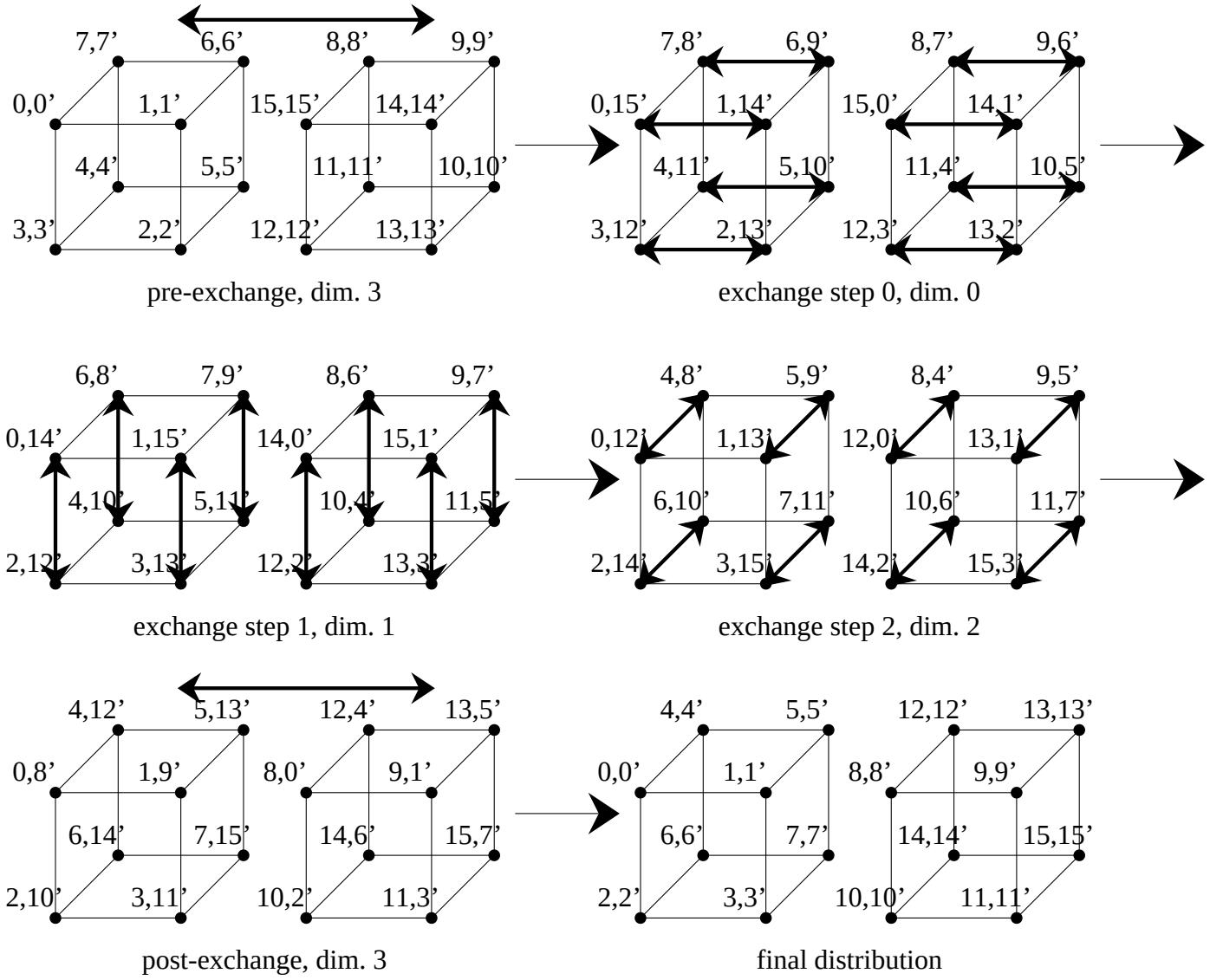

Figure 2: An example of Algorithm GB2 on a 4–cube.

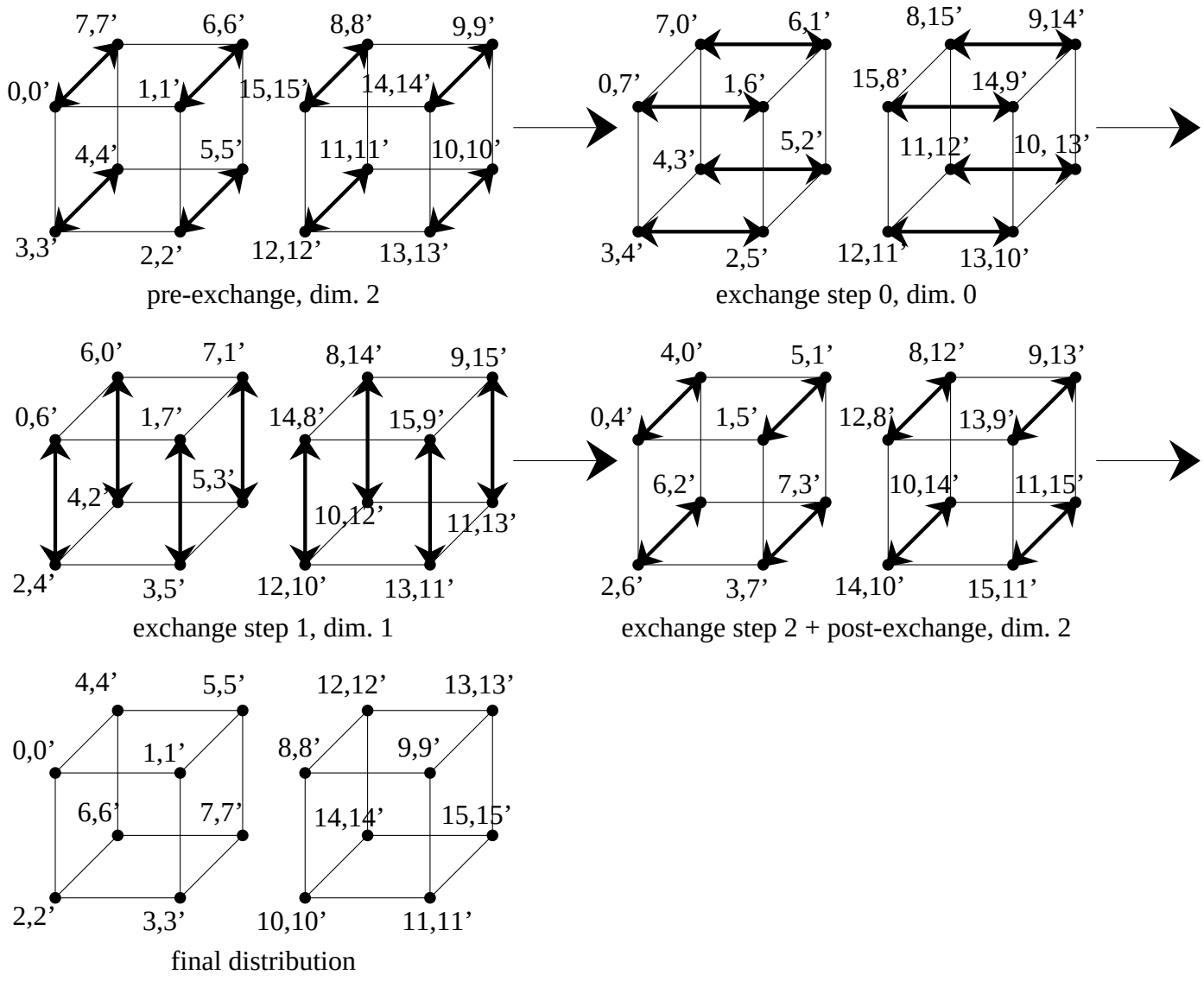

Figure 3: An example of Algorithm GB3 on a 4–cube.

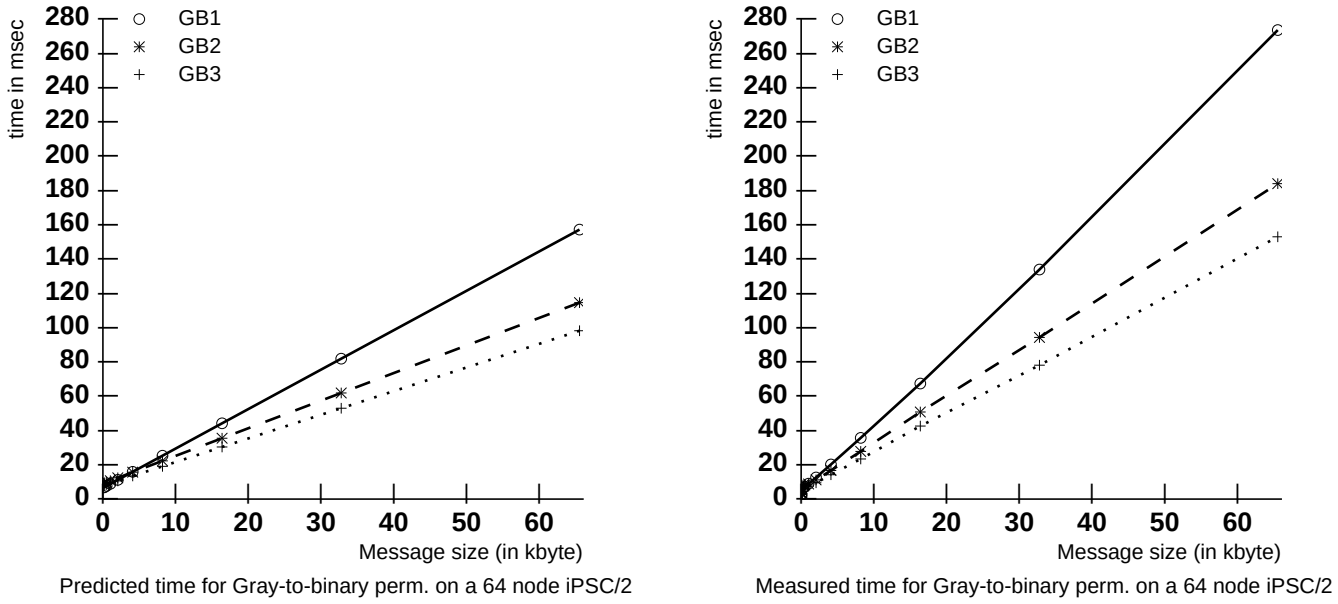

Figure 4: Comparison of the predicted (a) and measured (b) communication times for the three Gray–to–binary algorithms as a function of message sizes.

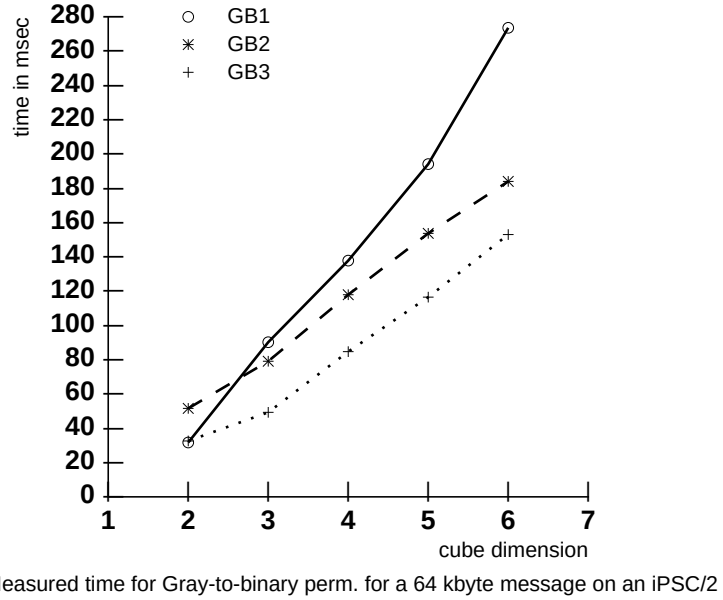

Figure 5: Comparison of the measured communication times for the three Gray–to–binary algorithms as a function of cube dimensions.