Optimal Self-Routing of Linear-Complement Permutations in Hypercubes[1]

by

Rajendra Boppana and C. S. Raghavendra

Technical Report No. CENG 89-37
12 December 1989

e-mail: raghu@surya.usc.edu
phone: (213) 743-5532
Dept. of Electrical Engineering–Systems
University of Southern California
Los Angeles, CA 90089–0781

[1]This research is supported by the NSF grant No. MIP 8452003, a grant from AT&T, DARPA/ARO Contract No. DAAG29-84-K-0066, ONR Contract No. N00014-86-K-0602.

Optimal Self–Routing of Linear-Complement Permutations in Hypercubes

Abstract

In this paper we describe an algorithm to route the class of linear-complement permutations on Hypercube SIMD computers. The proposed algorithm is self-routing and optimal, that is, the path established by the algorithm between each pair of source and destination PE's is via a minimal path using only the destination PE address. Furthermore, the algorithm requires only $(\log N)$ steps to realize any linear-complement permutation. The best known previous routing algorithms for Hypercubes are for the class of bit-permute-complement permutations. Those algorithms are either non–optimal or not self–routing. The algorithm presented is self–routing, optimal, and it routes a larger class of permutations. Also, this algorithm can route the class of linear-complement permutations in multi-dimensional meshes in optimal number of steps.

Key words: hypercube, interconnection network, linear permutations, minimal routing, self-routing.

Contents

List of Figures

1 Introduction

A parallel computer consists of a large number of processors (called Processing Elements, or in short PE's), and an interconnection network to exchange information between them. This interconnection network can be characterized as either a multistage interconnection network, e.g., shuffle exchange and Beneš networks, or a static interconnection network, e.g., topological connections of hypercube. In a static interconnection network, each PE is directly connected with other PE's which are termed adjacent PE's. Communication between PE's, that are not directly connected, goes through intermediate PE's.

In this paper, we are interested in SIMD computers with static interconnection network, specifically the hypercube computer. In such parallel computers, efficient communication schemes are necessary to obtain fast and efficient parallel algorithms. The problem of moving data from one PE to another is called the 'data routing problem'. Various schemes have been developed for the general case of data routing [8]. Permutation is a special case of data routing in which each PE sends a message to a PE, and each PE receives a message from a PE.

In a hypercube [12, 13], there are $N = 2^n$, $n > 0$, PE's; each PE is given a unique index (also called address) from $\{0, 1, \ldots, N-1\}$. PE i is connected to PE j if and only if the binary representations of the indices i and j differ in exactly one bit. Hence each PE has $\log N$ adjacent PE's (neighbors). A hypercube of 8 nodes is shown in figure 1. The index of each PE is indicated in binary form.

PE i (that is, PE with index i) can send a message to a non-adjacent PE k by sending it to an adjacent PE j such that the number of bits that j and k differ is one less than that of i and k, and repeating this until PE k is reached. It is clear that a shortest path of communication between any two PE's goes through minimal number of intermediate PE's, which is one less than the hamming distance of the indices of the two PE's.

An efficient method to realize arbitrary permutations in hypercubes is to use Batcher's

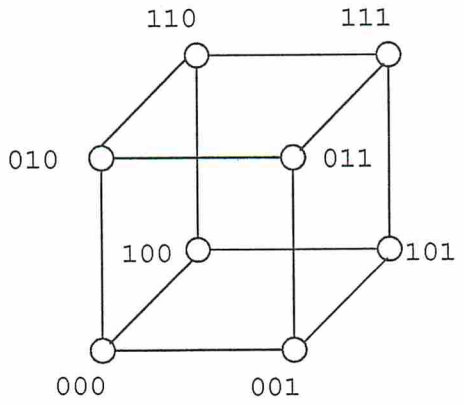


Figure 1: An 8 PE Hypercube.

sorting technique [1, 14]; this takes $O(\log^2 N)$ time. However, for certain classes of permutations, e.g. bit-permute-complement permutations ($\mathcal{BPC}$), more efficient and, perhaps, optimal routing schemes are possible [9, 10]. A routing scheme is called optimal (minimal) if the communication path for each pair of source and destination PE's goes through the minimum number of intermediate nodes.

Since each stage of a Beneš network of size $N = 2^n$ can be simulated in a hypercube of N PE's without any loss of time, any algorithm that routes a permutation in a Beneš network would route the same in hypercube, with the same time complexity [9]. A linear-complement permutation can be routed in a hypercube computer using the algorithm given in [2]. However this routing is not optimal, since communication paths will be of length $2 \log N - 1$, in the worst case, where as, the length of a path obtained minimally is, at most, of length $\log N$.

Nassimi and Sahni [10] developed an algorithm for minimal routing of bit-permute-complement permutations in hypercubes. However, their algorithm requires the structure of the permutation to be routed, hence is not self-routing. A routing algorithm is termed self-routing, if moving of data for each source-destination PE pair is done using only some local information, mostly, source and destination information. In this paper, we present a self-routing and optimal algorithm to route the class of linear-complement permutations.

An example of routing bit reversal permutation in an 8 PE hypercube is shown in figure 2. For each PE, the index and its initial tag (in parenthesis) are shown in binary form. The tags in PE's $000, 010, 101, 111$ are in the correct places before routing by the algorithm. Therefore the algorithm does not move these tags in the routing process. However the tags in PE's $001, 011, 100, 110$ differ from their respective host PE indices in the least significant and most significant bits. So tags in these PE's are first moved to the PE's with indices differing in the least significant bit position, and then moved to destinations PE's; for example the tag 100 in PE 001 is moved to PE 000, and then to PE 100, whereas the tag 001 in PE 100 is moved to PE 101 first, and then to PE 001. The path traced by each tag during routing is indicated by arrows between consecutive intermediate PE's in the tag's path.

In the next section, we give the algorithm and discuss its working. Also, we prove that the algorithm does minimal routing for linear-complement permutations.

2 A self-routing algorithm

First, we define the linear-complement permutation class and explain notation and expressions used in this paper. Later, we give the algorithm and discuss it with an example. We conclude this section with a proof of correctness of the algorithm. Throughout this paper it is assumed that there are $N = 2^n$ PE's in the hypercube computer, unless otherwise noted.

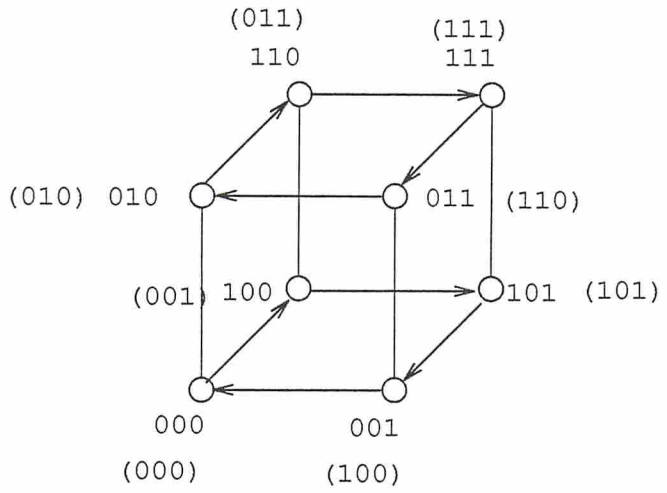


Figure 2: Routing bit reversal permutation in 8 PE hypercube using the algorithm.

2.1 Notation and definitions

In what follows we define the classes of bit-permute-complement permutations and linear-complement permutations of $N = 2^n$ numbers. Let $S = \{0, 1, \ldots, N-1\}$, and $T = \{1, 2, \ldots, n\}$. Also, let I (with binary representation $(I_n\ I_{n-1}\ \ldots\ I_1)$) be any number and O (with binary representation $(O_n\ O_{n-1}\ \ldots\ O_1)$) be its corresponding image under some mapping.

Definition 1 *A mapping $f : S \rightarrow S$ is called a permutation, if f is a bijection.*

Definition 2 *A permutation $f : S \rightarrow S$ is called a bit-permute permutation ($\mathcal{BP}$) if there exists a permutation $g : T \rightarrow T$ such that for each $I \in S$, its image $O = f(I)$ is given by permuting the bits $I_n, I_{n-1}, \ldots, I_1$ as specified by g.*

Definition 3 *Bit-permute-complement permutation ($\mathcal{BPC}$) is a bit-permute permutation that allows bits to be complemented after permuting.*

Definition 4 *A permutation is said to be a linear permutation* [6], *if there exists a non singular binary matrix $Q_{n \times n}$ such that every (I,O) pair satisfies equation 1.*

$$O^T = Q \times I^T \tag{1}$$

Definition 5 *Let $I' = (I_n\ I_{n-1}\ \ldots\ I_1, 1)$. A permutation is a linear-complement permutation ($\mathcal{LC}$) if there exists a binary matrix $P = (Q \mid k)$, where Q is as defined above and k is some n-bit binary vector, such that every (I,O) pair satisfies equation 2.*

$$O^T = P \times I'^T \tag{2}$$

A bit-permute-complement permutation is a linear-complement permutation with a permutation matrix (each row and column of the matrix has exactly one 1) as Q.

In the literature [3, 5, 11], the linear permutations are termed as the non-singular linear transformations of the n-dimensional vector space over the field $F = GF(2^n)$. (There are exactly 2^n elements in this vector space, and each element corresponds to a PE index in binary form.) There is one-one correspondence between the boolean matrices of size $n \times n$ and linear transformations on n-dimensional boolean vector space [5]. A linear transformation is invertible, if and only if the corresponding boolean matrix is invertible. A linear transformation can also be viewed as a homomorphism on the group underlying the vector space.

Index of a PE is represented as an n-bit vector. Therefore, in the context of a hypercube, a permutation is called a linear-complement permutation if there exists a P, as defined above, satisfying equation (2) for every pair of source and destination PE indices. Hence, if a given permutation is a linear-complement permutation, each bit of the destination PE index is described by a linear combination of the bits of the source PE index, and a constant term that could be either 0 or 1.

A hypercube of 2^n processors is denoted as $\mathcal{Q}_n$; and the set of linear-complement permutations on 2^n numbers is represented as $\mathcal{LC}(n)$. In this paper, all the additions are bitwise and modulo 2 (that is, bitwise exclusive-or operation), and all the logarithms are to the base 2. If PE's i and j are connected by a link, then PE j is termed adjacent to PE i in dimension r, where i and j differ in bit r.

In the beginning, each Processing Element (PE) in the hypercube, contains the index of the destination PE and a message. We use the word *tag* to indicate the packet containing the destination PE index and the message. For the purpose of the routing algorithm the message is unimportant. Hence, we assume that the tags contain only the the destination PE index, an n-bit vector. If PE x has tag y, then it is called the host PE of the tag y. The goal of the algorithm is to route tags, so that, at the end of the routing process, each tag matches with the address of its host PE.

When a PE sends a tag to its neighbor, it loses a tag; and when it receives a tag from neighbor, it gains a tag. So, during routing, the number of tags in a PE may vary. However, the algorithm assures that there are only two cases: one tag per PE, and two tags per PE for $N/2$ PE's and no tags for the remaining PE's. We say that the hypercube is in **state A** when each PE has one tag. Before routing the hypercube is in state A. When there are 2 tags for each of $N/2$ PE's and no tags for the remaining $N/2$ PE's then the hypercube is said to be in **state B**. Throughout the routing process, the hypercube is, as proved later, in one of the two states.

During routing, the effect of sending a tag from a PE to another PE with index differing in bit position i is succinctly stated that the tag is routed along the dimension

i. We use the expression 'a tag is routed to correct bit i' to mean that the PE having that tag routes the same along the dmension i. if the tag's bit i differs from that of PE's index in binary form. Note that it makes sense to say that a tag is routed to correct bit i even when bit i of the tag and that of the PE address match; however, in this case the tag remains in the PE itself during that routing step. In each routing step, all the tags are examined and all of them are routed to correct a particular bit, say i; an alternate way of saying this is 'bit i is corrected'. After a bit is corrected, all the tags are in the correct PE's with respect to that bit.

2.2 Statement and discussion of the algorithm

Algorithm **HL**:

If the hypercube is in state A, each PE routes its tag to correct the least significant bit that has not been used in earlier routing steps. If the hypercube is in state B, PE's with no tags do nothing, and PE's with two tags compare the two tags and route one of them to correct the least significant bit in which they differ. This is repeated $(\log N)$ times. ∎

An example showing the routing of a linear-complement permutation, specified by the following set of linear equations, is shown in figure 3.

$$\begin{aligned} y_3 &= x_2 + x_1, \\ y_2 &= x_3 + x_1 + 1, \\ y_1 &= x_2 + 1. \end{aligned}$$

The index of each PE and its initial tag (in parenthesis) are shown in figure 3(a); this indicates the allocation of tags to PE's in the hypercube. The effect of correcting bit 1 in routing step 1 is shown in figure 3(b). Here PE's with indices $000, 011, 100$, and 111 send their tags the PE's adjacent to them in dimension 1. It can be seen that the tags in these PE's differ from their respective host PE indices in the least significant bit. The paths traced by the tags are shown by arrows; and the tags in a PE (if present) are given in parenthesis.

After the routing step 1, half the PE's of the hypercube have four tags, and the remaining PE's have none. From figure 3(b), we can see that in each of the four PE's with two tags, the tags differ in both bit positions. Each of these four PE's pick to correct bit 2, as specified by the algorithm, independently. Figure 3(c) shows the effect of correcting bit 2 in routing step 2. The path traced by the tags that are routed are shown by arrows.

After correcting bits 1 and 2, each PE again has one tag. So, each PE picks to correct bit 3, as required by the algorithm, independently. In this routing step, PE's 001 and 010

exchange tags with PE's 101 and 110 respectively. This is shown by two parallel lines, pointing in opposite directions arrow heads, between the exchanging PE's to indicate the path traced by each tag. After the third routing step, all the tags are at their correct destination.

In the next section, we prove that the algorithm routes any linear-complement permutation correctly in $\log N$ steps.

2.3 Proof of correctness

Let $y = (y_n, \ldots, y_1)$ represent a tag and $x = (x_n, \ldots, x_1)$ represent its host PE address. Also, let the tags be distributed among PE's according to some affine linear transform f; that is, the mapping from tags to PE's is of the form: $f : y \rightarrow x$, $x = f(y) = T(y) + k$, where, T is some linear transformation and k an n-bit vector. In other words, for each tag, the address of its host PE can be specified by the following bit equations.

$$\left.\begin{array}{rcl} x_n & = & \sum_1^n \lambda_{n,i} \cdot y_i + k_n \\ & \vdots & \\ x_p & = & \sum_1^n \lambda_{p,i} \cdot y_i + k_p \\ & \vdots & \\ x_1 & = & \sum_1^n \lambda_{1,i} \cdot y_i + k_1 \end{array}\right\} \qquad (3)$$

Here, the $n \times n$ matrix $(\lambda_{i,j})$, $\lambda_{ij} \in \{0, 1\}$, represents the transformation matrix of the linear transform T. It is clear that if the matrix $(\lambda_{i,j})$ is non-singular, the linear transform T is invertible, and T is a linear permutation. A non-zero vector for k indicates translation of the linear transform, and it does not affect the invertibility of T.

Now, suppose that the tags are moved among the PE's using the following rule: "each PE moves the tags that do not agree with its index bit p to its neighbor PE in the dimension p." Then, we claim the following.

Claim 1 *When tags are moved as specified above, the mapping between tags and their host PE addresses is still an affine linear transform.*

Proof: After the routing step, each tag agrees with its host PE in bit p. Hence, for all the tags the equation for bit p is, simply, $x_p = y_p$.

Now, if a tag is moved during the routing step, then it is moved to a new host PE that differs from the old host PE only in the bit p. Hence, irrespective of whether the tag is moved or not, the other bit equations are unchanged. ∎

If the mapping of tags to PE's is an affine linear transform of the form, $x = T(y) + k$, Then, $x + k = T(y)$ is a linear transform. That is, if each PE address is translated by k, then the tag distribution is simply a linear transform T. Now, $\ker T = \{y \mid T(y) = 0\}$,

that is, ker T is the set of all tags assigned to PE with address 0, under the linear transform T.

For any linear transform T, $T(0) = 0$; so, $\mid \ker T \mid \geq 1$. Also, T can be treated as a homomorphism from the group underlying the vector space of PE indices to itself. From the first isomorphism theorem [11], we get that each PE with a tag will have the same number of tags that PE 0 has under the linear transform T. Therefore, we have the following lemma.

Lemma 2 *If the tags are distributed among the PE's such that some PE's have one tag, some other PE's have two tags, while the remaining PE's have none, then the mapping between tags and PE addresses is not an affine linear transform.* ∎

Lemma 3 *Suppose the tags are distributed among PE's, according to some affine linear transform $x = T(y) + k$, such that half of the PE's have two tags and the remaining PE's have no tags. Then, the two tags in a PE, if present, differ in the same bit positions.*

Proof: The mapping between tags and PE's after translating the PE's addresses by k is a linear transform, T. Under T, PE 0 has two tags, namely, 0 and a, for some $a \neq 0$. Now, take any PE x that has two tags b and c. To prove the lemma, it is sufficient to show that $b + c = a$. But, it is true in view of the following.

$$
\begin{aligned}
& T(0) = T(a) = 0 \\
& T(b) = T(c) = x \\
\Rightarrow\ & T(b + c) = T(b) + T(c) = x + x = 0 \\
\Rightarrow\ & (b + c) = 0, \text{ or } a \\
\Rightarrow\ & b + c = a, \text{ since } b \neq c .
\end{aligned}
$$

∎

Lemma 4 *The algorithm HL routes tags such that the following are always true: (a) after each routing step, the tag distribution is given by some affine linear transformation, (b) the hypercube is in either state A or state B, and (c) in any routing step, a PE moves at most one tag.*

Proof: Whenever the hypercube is in state A, each PE has exactly once tag, hence, the tag movement specified by the algorithm HL and the rule used in the claim 1 is the same. So, (a),(c) are true. Since, the tag distribution should be according to some affine linear transform, by lemma 2, either each PE has one tag (state A), or half the PE's have two tags (state B). So (b) is also true after the routing step. Now, let us assume that they are true for the first $m \geq 1$ routing steps. We need to show that they are true after routing step $(m + 1)$.

After routing step m, if the hypercube is in state A, then the conditions (a)-(c) are true after the next routing step. However, if the hypercube is in state B, then lemma 3,

tells that the two tags in a PE (if present) differ in the same bits. But, in that case, the algorithm HL chooses one dimension unambiguously. Hence, for each PE with two tags, exactly one tag matches with its index in the bit chosen for correction in the next routing step. So, in the routing step $(m+1)$, each PE with two tags routes exactly one tag, so that the tags that differed from the host PE index in the routing bit will now match with the new host PE index. This shows that (c) is true for the routing step $(m+1)$. The above argument and claim 1 tell that (a) is true after the routing step $(m+1)$. Since, a PE with two tags moves one tag in the $(m+1)$ routing step, after the routing step, it has 1 or 2 tags. Using lemma 2, we conclude that (b) is also true. ∎

Corollary 1 *Any routing step given by the algorithm HL, and the rule given for the claim 1 are equivalent, provided each time the bits chosen for the rule is same as the routing bit chosen by the algorithm.*

Proof: This follows directly from the above lemma. ∎

Corollary 2 *The algorithm HL routes tags such that each bit is chosen for routing exactly once. And, after correcting a bit, each tag matches with its host PE index in that bit.*

proof: The above corollary tells that when a bit is used for routing step, all the tags match with the host PE index, after completing the routing step. The fact that each bit is chosen as routing bit exactly once, is easy to see. ∎

Theorem 1 *The algorithm, HL, routes any linear-complement permutation, in the set $\mathcal{LC}(n)$, in a hypercube, $\mathcal{Q}_n$, in n routing steps, such that each PE needs to route at most one tag in a routing step.*

Proof: The proof follows from the lemma 4, and the corollaries following the lemma. ∎

3 Scope and use of the algorithm HL

In this section, we describe other aspects of the algorithm HL in routing permutations in various types of hypercubes. First, we describe how to use this algorithm to route linear-complement permutations in multi-dimensional meshes in optimal number of steps. Next, we describe how the algorithm HL can be used to route messages in a circuit switched mode of transmission. Then, we characterize a larger class of permutations that are routed by the algorithm HL in hypercubes with the constraint of choosing of same dimension links by all PE's, in a routing step, is relaxed.

3.1 Routing linear-complement permutations in a q-mesh

The algorithm HL can be used for routing the linear-complement permutations in multi-dimensional meshes (q-mesh, $q \geq 2$), by direct simulation of the hypercube connections.

Let us consider a 2-mesh with PE's indexed in row major order. Let n be an even number. Then the number of PE's in a row or column is $2^{n/2}$. Dimension i, $0 \leq i < n$, connection of n-cube can be simulated on the 2-mesh, in 2^x routing steps, where $x = i$ or $i - n/2$ depending on $i < n/2$ or $\geq n/2$.

The time required to route linear-complement permutations on a 2-mesh is:

$$2(2(2^0 + 2^1 + \cdots + 2^{\frac{n}{2}-1})) \;=\; 4(2^{\frac{n}{2}} - 1).$$

This is the optimal number of steps to route the class of linear-complement permutations on 2-mesh [7]. It can be shown that for an SIMD q-mesh, the time taken by the algorithm is $2q(2^{\frac{n}{q}} - 1)$, which is optimal.

Theorem 2 *The algorithm HL can route linear-complement permutations in a multi-dimensional mesh, in optimal number of steps.* ∎

3.2 Routing linear-complement permutations in a circuit switched hypercube

So far, we have assumed that a message is attached to the tag, hence moved with it to the destination. However, when the message is long, it is faster to send it using circuit switching scheme. In this scheme, a path is established between each source-destination pair of PE's, and then messages are transferred at high rates. At the end of transmission of the messages, the paths are released. To transfer messages in circuit switching mode in a hypercube, we will assume that each PE has necessary hardware to establish temporary physical paths between the input and output links used by the tags that passed through the PE. Also, we impose the restriction that a link between any two PE's, say i and j, can be used by each of the PE's i and j at most once, in circuit switching. The lemma 5 and the following discussion show that this constraint is satisfied.

Lemma 5 *During the routing process, any two adjacent PE's communicate at most once.*

Proof: Any two adjacent PE's have their index differ in only one bit say x. So if at all they communicate, they do so only in the routing step to correct bit x. ∎

We have shown that in a routing step any PE will move at most one tag. Hence, it immediately follows that a link between any two PE's will be used at most once.

To send messages by circuit switching scheme, first the tags (without messages) are routed. Then, each PE uses the path traced by its tag to send message to the destination PE. Note that this can not be done using the self-routing algorithms of Beneš network given in the papers [2, 9].

3.3 On routing a larger set of permutations

So far, a hypercube is assumed to be operating in SIMD mode, and it was proved that the algorithm HL can route linear-complement permutations. In the following discussion, we use a stronger mode of operation for hypercube, and show that the algorithm HL routes a larger set of permutations.

An n-cube can be partitioned into 2^k, for some $k \leq n$, $(n-k)$-cubes. In a routing step, if all the PE's in a subcube choose the same dimension for routing, but PE's in different subcubes may choose different dimensions, then it is clear that each subcube is operating in SIMD mode but the n-cube consisting of these subcubes is not in SIMD mode. Such a mode of operation is called Multiple-SIMD or M-SIMD. In general, each subcube can again be partitioned and partitioning of a subcube may be different from the partitioning of another subcube, etc. For the following, we assume that a hypercube can operate in M-SIMD mode when partitioning of PE's, as discussed above, is considered.

After the first routing step, for routing purposes, the $\mathcal{Q}_n$ can be viewed as two subcubes, $\mathcal{Q}_{n-1}$, with each subcube having 2^{n-1} tags. In the remaining steps of the algorithm, each subcube routes tags among its PE's, and does not send any tag to the other subcube. This observation can be applied for the remaining routing steps too. This gives us the motivation to relax the constraint of SIMD mode of operation to route a larger class of permutations by the algorithm HL. For the following discussion, we assume that after each routing step, the subcubes may choose different dimensions for the next routing step. We call this mode of operation as Multiple-SIMD or M-SIMD mode.

Let the set $A = \{n, n-1, \ldots, 1\}$ be partitioned into two subsets $B = \{n, n-1, \ldots, n-k+1\}$ and $C = \{n-k, n-k-1, \ldots, 1\}$, where $1 \leq k \leq n$. B can be used to partition the set of numbers $\{0, 1, \ldots, 2^n - 1\}$ such that if i, j are in the same partition, then $i_x = j_x$, $\forall\, n-k+1 \leq x \leq n$. This idea can be used to partition the PE's in a hypercube such that there are 2^k partitions, 2^{n-k} PE's in each subcube.

Suppose PE's in a hypercube are partitioned as given above. Define a permutation π that permutes partitions of PE's by some permutation in $\mathcal{LC}(k)$, and PE's in each partition by some permutation in $\mathcal{LC}(n-k)$ (permutations of PE's in different partitions can be different).

Lemma 6 *The algorithm HL routes the set of permutations as discussed above in a hypercube operating in M-SIMD mode with partitions as described above.*

Proof: In the first $n-k$ routing steps, in each sub-cube a linear-complement permutation is realized by the algorithm HL. This preserves the SIMD mode of operation for

each subcube. Once the PE's in each partition are permuted, the permutation of partitions is achieved as follows. Partitions are rearranged so that, bits $n-k, n-k-1, \ldots, 1$ are used in partitioning the hypercube. So each subcube will have 2^k PE's and there are 2^{n-k} such subcubes. Now each subcube has to route a permutation in $\mathcal{LC}(k)$, the permutation that is defined on the bits $n, n-1, \ldots, n-k+1$ of the n-cube, to complete the original routing task. Since this can be done by the algorithm HL, the statement holds for all n. ∎

This idea of partitioned permutations can be used recursively on each partition, and on the permutation of partitions itself. We call such permutations as partitioned linear-complement permutations ($\mathcal{PLC}$).

Definition 6 *For $n = 1, 2$, $\mathcal{PLC}(n)$=$\mathcal{LC}(n)$. For $n \geq 3$, $\mathcal{PLC}(n)$ is defined as follows. A permutation is in the set $\mathcal{PLC}(n)$, if it permutes the partitions of PE's by a permutation in $\mathcal{PLC}(k)$, and PE's in each partition are permuted by some permutation in the set $\mathcal{PLC}(n-k)$, where $1 \leq k \leq n$ and the partitioning of PE's is as defined above.*

Lemma 7 *The algorithm HL can route any $\mathcal{PLC}$ permutation in a hypercube operating in M-SIMD mode.*

Proof: By considering the partitioning of the hypercube to be the same as the partitioning of the $\mathcal{PLC}$ permutation being realized, it can be shown that proof for this lemma is a simple generalization of that of the previous lemma. ∎

3.3.1 The set $\mathcal{PLC}$

In what follows, we give an estimate on the size of the the set of partitioned linear-complement permutations.

A decomposable linear permutation [4] is a linear permutation whose matrix Q can be partitioned as follows.

$$Q = \left(\begin{array}{c|c} Q_1 & 0 \\ \hline 0 & Q_2 \end{array}\right)$$

Q_1 and Q_2 are square boolean matrices, and define linear permutations on smaller size set of numbers. Let us define a subset of $\mathcal{LC}(k)$, that contains the permutations that can not be decomposed such that rows 1 and k of the original Q matrix are in different partitions. With complement of bits considered, this set is denoted as $\mathcal{LC}'(k)$.

We now give an alternate definition of partitioned linear-complement permutations. It can be shown that the following definition and the definition given earlier are equivalent.

Definition 7 *A permutation is in the set of partitioned linear-complement permutations, if there is a partition of most significant k bits, for some $1 \leq k \leq n$, such that partitions of the PE's are permuted by some permutation in $\mathcal{LC}'(k)$, and the PE's in each partition are permuted by some permutation in $\mathcal{PLC}(n-k)$.*

From this, we get

$$|PLC(n)| = \sum_{k=1}^{n} |LC'(k)| \cdot |(PLC(n-k))|^{2^k} \quad (4)$$

Since, $|LC'(k)| \leq |LC(k)|$, and $|LC(k)| = 2^k \times |L(k)|$, where $L(k)$ is the set of linear permutations, $|L(k)| = 2^{\frac{k(k-1)}{2}}(2^k-1)(2^{k-1}-1)\cdots(2-1)$. Hence, the upper bound given below can be used to approximate the size of $\mathcal{PLC}(n)$.

$$|PLC(n)| \leq \sum_{k=1}^{n} |LC(k)| \cdot |(PLC(n-k))|^{2^k}.$$

Since, $PLC(k) \geq LC(k) \geq 2^{\frac{k^2}{2}}$, the size of the set $\mathcal{PLC}(n)$ is at least $\Omega\left(N^{\sqrt{N}\log N/8}\right)$, which is the lower bound on the term with $k = n/2$, in the summation of the right side of the identity 4. Thus the size of the set grows exponentially with N.

The idea of operating a hypercube in M-SIMD mode gives a method to construct exponential number of rearrangeable Beneš-like networks. A link between two PE's in the hypercube is assumed to be replaced by a switch such that tag routing between the two PE's is simulated by the switch. Each routing step in the hypercube is equivalent to the operation of a stage of switches in a corresponding multistage interconnection network constructed as below. Consider routing a partitioned linear-complement permutation in a M-SIMD hypercube. In the first routing step, all PE's in the cube use dimension 1 links. So, switches in the first stage have inputs lines with addresses that differ only in bit 1. In the next routing step, each subcube chooses dimension links depending on the partitioned linear-complement permutation being realized. For different partitioned linear-complement permutations, different dimensions are chosen in each subcube. The switches simulating the links used in the second routing step form the second stage of switches. The interconnection pattern between the first and second stage of switches is such that, in the top half of the switches will have input lines with even addresses, and the lower half of switches will have input lines with odd addresses, or vice versa. Furthermore, input lines to a switch will differ in only the bit that is the dimension of the link simulated by the switch. Now, top half of switches are considered to form a sub-network, and bottom half of switches are considered to form another sub-network. This is repeated recursively in each routing step. The next routing steps can be used in this

manner to obtain the first n stages of a network. After n routing steps, the first n stages, of a Beneš-like multistage network, are formed. The complete network of $(2n-1)$ stages is such that stages $n+1,\ldots,2n-1$ are mirror images of stages $n-1,\ldots,1$ respectively. Beneš network is obtained when the same dimension is chosen in each routing step in each subcube, i.e., when SIMD mode of operation is enforced on hypercube. For M-SIMD mode of operation, different partitioned linear-complement permutations give different Beneš like networks, which by construction are rearrangeable. The algorithm given in [2] with appropriate modifications can be used to route the class of linear-complement permutations in these networks.

4 Conclusions

In this paper, we presented an algorithm to the realize the the class of linear-complement permutations in a hypercube. The algorithm is simple, self-routing, and optimal. It routes any linear-complement permutation in $(\log N)$ routing steps, with each step requiring a constant time. In message passing scheme, time required for a routing step is proportional to the length of the message.

Since, the algorithm picks to correct bit 1 in routing step 1, and whenever the hypercube is in state A, the least significant bit that is not yet corrected is picked in the next routing step, it routes inverse omega (Ω^{-1}) permutations trivially. If the algorithm picks to correct the most significant bit, whenever there is a choice of bits that can be picked to correct in a routing step, then it is clear that it still routes linear-complement permutations and also the class of omega (Ω) permutations.

All the permutations of order 4 are routed by this algorithm, since they all are in $\mathcal{LC}(2)$. It is shown that the class of permutations realizable by this algorithm on a hypercube M-SIMD computer is larger than the class of permutations realizable on a hypercube SIMD computer. When the hypercube is allowed to operate in the MIMD mode, an even larger class of permutations can be routed by the algorithm HL. For example, in an 8 PE hypercube, any arbitrary permutation can be routed if the mode of operation is MIMD; the same is not true if SIMD mode of operation is used.

The algorithm routes many permutations that are not in the linear-complement permutation class. An interesting and useful problem would be to study the characterization of the class of permutations realizable by the algorithm. Knowing this characterization, one can perform a pre-processing step to convert an arbitrary permutation to a permutation realizable by this algorithm.

References

[1] K. E. Batcher. Sorting networks and their applications. In *AFIPS Spring Joint Computer Conference*, pages 307–314, 1968.

[2] R. Boppana and C. S. Raghavendra. On self–routing in Beneš and $(2n - 1)$-stage shuffle exchange networks. In *Int'l Conf. on Parallel Processing*, pages 196–200, 1988.

[3] G. Birhkoff and S. MacLane. *A survey of modern algebra.* Macmillan, 4 edition, 1977.

[4] I. N. Herstein. *Topics in Algebra.* John-Wiley and Sons, 2 edition, 1975.

[5] K. Hoffman and R. Kunze. *Linear Algebra.* Prentice-Hall, 2 edition, 1974.

[6] M. C. Pease, III. The indirect binary n–cube microprocessor array. *IEEE Trans. on Computers*, c–26(5), 1977.

[7] D. Nassimi and S. Sahni. An optimal routing algorithm for mesh-connected parallel computers. *J. Assoc. Comput. for Mach.*, 27(1), 1980.

[8] D. Nassimi and S. Sahni. Data broadcasting in simd computers. *IEEE Trans. on Computers*, c–30(2), 1981.

[9] D. Nassimi and S. Sahni. A self–routing Beneš network and parallel permutation algorithms. *IEEE Trans. on Computers*, c–30(5), 1981.

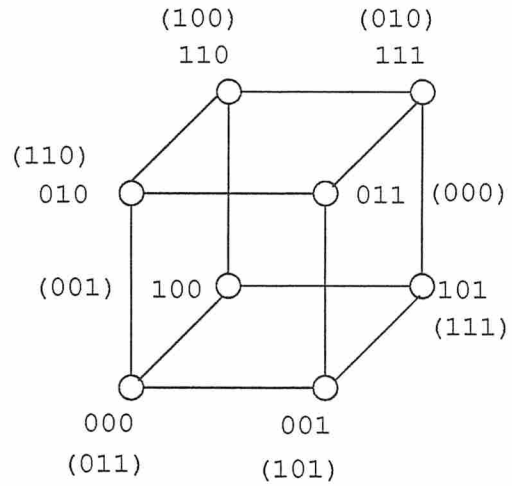
[10] D. Nassimi and S. Sahni. Optimal BPC Permutations on a Cube Connected SIMD Computer. *IEEE Trans. on Computers*, c–31(4), 1982.

[11] J. J. Rotman. *An introduction to the theory of groups.* Wm. C. Brown Publishers, 3 edition, 1988.

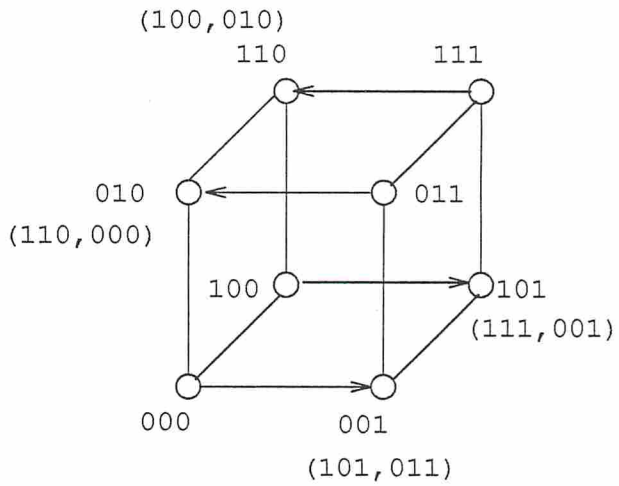
[12] C. L. Seitz. The cosmic cube. *Comm. of Assoc. for Comput. Mach.*, 28(1):22—33, 1985.

[13] J. S. Squire and S. M. Palais. Programming and Design Considerations for a Highly Parallel Computer. In *Proc. Spring Joint Computer Conf.*, 1963.

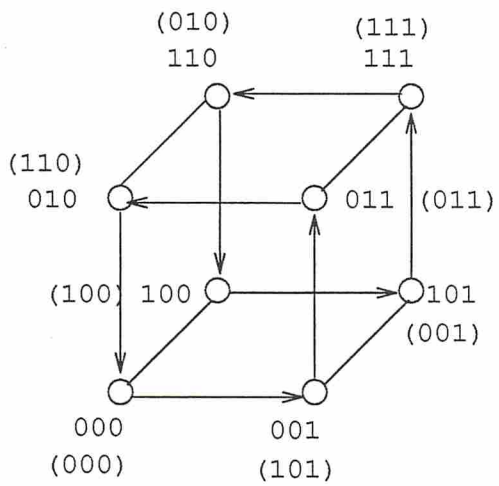
[14] H. S. Stone. Parallel processing with the perfect shuffle. *IEEE Trans. on Computers*, c–20(2), 1971.



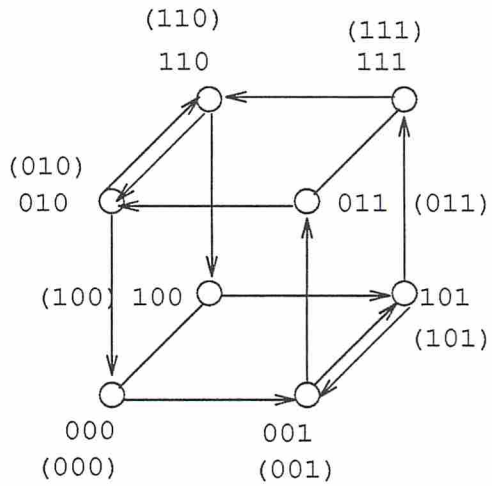
(a) Before routing



(b) After correcting bit 1



(c) After correcting bit 2



(d) After correcting bit 3

Figure 3: Routing a linear-complement permutation on $\mathcal{Q}_3$.