Optimal Processor Mapping for Linear-Complement Communication on Hypercubes

Yomin Hou, Chien-Min Wang, *Member*, *IEEE Computer Society*, Chiu-Yu Ku, and Lih-Hsing Hsu

Abstract—In this paper, we address the problem of minimizing channel contention of linear-complement communication on wormhole-routed hypercubes. Our research reveals that, for traditional routing algorithms, the degree of channel contention of a linear-complement communication can be quite large. To solve this problem, we propose an alternative approach, which applies processor reordering mapping at compile time. In this compiler approach, processors are logically reordered according to the given communication(s) so that the new communication(s) can be efficiently realized on the hypercube network. It is proved that, for any linear-complement communication, there exists a reordering mapping such that the new communication has minimum channel contention. An $O(n^3)$ algorithm is proposed to find such a mapping for an n-dimensional hypercube. An algorithm based on dynamic programming is also proposed to find an optimal reordering mapping for a set of linear-complement communications. Several computer simulations have been conducted and the results clearly show the advantage of the proposed approach.

Index Terms—Hypercubes, linear-complement communication, channel contention, processor mapping, wormhole routing.

1 Introduction

In a message-passing multicomputer, efficient schemes to move messages among the processors are required for obtaining fast and efficient parallel algorithms. Many studies have been based on *store-and-forward routing*, where the message latency is proportional to the product of the message length and the number of routing steps. Hence, most of them have concentrated on minimizing the number of routing steps in moving messages among processors [4], [16], [17], [18], [20], [21]. On the other hand, wormhole routing has been widely adopted recently due to its effectiveness in interprocessor communication [1], [2], [22]. With wormhole routing, each message is divided into a number of flits. The header flit(s) carries the address information and governs the route while the remaining flits of the message follow in a pipeline fashion. The pipelined nature of wormhole routing provides two attractions. First, in the absence of channel contention, the network latency would be relatively insensitive to the path length [9], [22]. Second, the large message buffers for each router are obviated. Only small flit buffers are required [22].

However, channel contention can have a severe impact on the network latency of wormhole routing. Channel contention happens when multiple messages simultaneously use the same channel in their routes. If k messages are contending for the same channel, only one of them can reserve the channel and be forwarded through it. The other messages have to wait until the channel is released and then contend for it again. In the worst case, the network latency will become k times. Hence, an important issue on wormhole-routed parallel computers is to minimize channel contention between messages. Many studies have tried to improve the adaptability of routing algorithms to solve the contention problem [5], [7], [10], [11], [15], [22]. However, this approach requires extra hardware support, such as buffer space and control logics. Moreover, the complexity of adaptive routers significantly increases their interrouter setup delay and flow control cycle times [6]. Consequently, the claims of performance advantages in channel utilization may not be able to be balanced against losses on achievable implementation speed. For these reasons, we try to solve the contention problem by compiler approaches rather than routing algorithms at runtime.

In this paper, we focus on the problem of minimizing the channel contention of *linear-complement communication* (LCC) on wormhole-routed hypercubes [14], [23]. Linear-complement communication is a class of communications where the address bits of the destination of each message are linear combinations of the address bits of its source and their complements. Many important problems, like fast Fourier transform, matrix transposition, polynomial evaluation, etc., can be effectively solved on parallel computers which have an efficient scheme to support this type of communications. To minimize the channel contention of LCC, we adopt a new approach which applies *processor reordering mapping* at compile time. In the compiler approach, processors are logically rearranged according to

- *Y. Hou and L.-H. Hsu are with the Department of Computer and Information Science, National Chiao-Tung University, Hsinchu, Taiwan, ROC. E-mail: {ymhou, lhhsu}@cis.nctu.edu.tw.*
- *C.-M. Wang is with the Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC. E-mail: cmwang@iis.sinica.edu.tw.*
- *C.-Y. Ku is with Avanti Technology Inc., Taipei, Taiwan, ROC. E-mail: cyku@avanticorp.com.*

Manuscript received 15 July 1998; revised 27 Sept. 1999; accepted 1 Sept. 2000.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number 107149.

the given communications before the executable code of the given parallel program is generated. In this way, no extra data movement will be incurred. By appropriately rearranging the processors, the new communications can be efficiently realized on the hypercube network. It can be proved that, for any LCC, there exists a reordering mapping such that the new communication has minimum channel contention. An $O(n^3)$ algorithm is proposed to find such a mapping for an n-dimensional hypercube. For a parallel program containing more than one LCC, a dynamic programming algorithm is proposed to find an optimal reordering mapping. With these results, compiler techniques can be used to minimize the channel contention of LCCs on hypercubes. In addition, only the e-cube routing is assumed in the proposed approach and no extra hardware is needed. Experiments based on computer simulation have been conducted and the results clearly show the advantage of the proposed approach.

The discussion in this paper can be directly applied to hypercube computers such as *nCUBE 2* [1]. However, for variations of the hypercube topology, the discussion may require some modifications. For example, in *SCI Origin 2000* [2], two nodes are connected to a single router instead of one. Therefore, the network topology for an Origin system with 2^n nodes is an $(n-1)$-dimensional hypercube. Hence, an n-dimensional LCC on an Origin system can be viewed as performing two LCCs simultaneously on the $(n-1)$-dimensional hypercube. A detailed discussion about performing LCCs on an Origin system can be found in [12].

Related researches for store-and-forward interconnection networks have been reported. Most of them focused on subsets of LCC, such as *linear-complement permutation* (LCP) and *bit-permute-complement permutation* (BPC). For example, Boppana and Raghavendra [4] considered LCPs on hypercubes, Nassimi and Sahni [20], [21] dealt with BPCs on meshes and hypercubes, and Masuyama [17], [18] dealt with BPCs on chordal rings and hypercubes. However, none of these methods can be applied to *linear-complement scatter* (LCS) or *linear-complement gather* (LCG). Lin and Wang [16] considered another set of communications represented by $\binom{s}{d}$-*mask formalism*. Although this representation scheme can encode a broad class of communications, it still cannot represent an LCS or LCG with a single $\binom{s}{d}$-mask. All of the above researches tried to give efficient routing algorithms. Hence, they require newly designed hardware and cannot be applied to existing parallel computers. Moreover, these researches aim at minimizing the number of routing steps and are not suitable for wormhole-routed networks.

The rest of this paper is organized as follows: Section 2 introduces the notations and definitions. In Section 3, we describe the compiler approach. Some properties and an algorithm of processor reordering mapping are presented in Section 4. In Section 5, a dynamic programming algorithm is proposed for a set of LCCs. Section 6 shows the experimental results based on computer simulation. Finally, conclusions are given in Section 7.

2 Background

When a communication is performed, messages are generated by a set of source nodes and transmitted through the interconnection network to their destination nodes. The *communication latency* is the interval from the time the source nodes begin to send messages until the last destination node has received the message. If some of the paths for transmitting these messages contend for the same channel, then the communication latency will be increased. The more paths contend for the same channel, the longer communication latency is required. Therefore, the maximum number of paths contending for the same channel has a severe impact on the communication latency. Let the degree of channel contention for a communication be defined as the maximum number of paths contending for the same channel. In this paper, we consider the problem of performing LCC on a hypercube computer with the e-cube wormhole routing capability. Our goal is to minimize the degree of channel contention so that LCC can be performed efficiently. In this section, the definitions and notations of these terminologies will be clarified.

2.1 The Hypercube Network

An n-dimensional hypercube is a directed graph which contains $N = 2^n$ nodes and $n \times 2^n$ channels. Each node corresponds to an n-bit binary string, $b_{n-1}b_{n-2}\ldots b_1b_0$. We shall use a binary vector $[b_0b_1\ldots b_{n-1}]^t$ to represent it. Two nodes are connected with a pair of channels, one for each direction, if and only if their binary strings differ in exactly one bit. As a consequence, each node is incident to n other nodes through n different channels, one for each bit position. The channel from node x to x' is denoted by (x, x') and said to be at dimension k if x and x' differ in the kth bit position.

2.2 Linear-Complement Communication

The messages generated when performing a communication operation usually can be formulated by some specific pattern. For example, in the bit-reverse communication operation, the source and destination nodes of each message can be represented by $[x_0x_1\ldots x_{n-1}]^t$ and $[x_{n-1}x_{n-2}\ldots x_0]^t$, respectively. In what follows, we will define the class of linear-complement communications on an n-dimensional hypercube. The addition and multiplication in this section are modulo-2, i.e., they are defined in the finite field, GF(2) [19].

Definition 1. *A communication is a linear-complement communication (LCC) if there exits a binary matrix $A_{n\times n}$ and an n-dimensional binary vector b such that, for every message with source node x, its destination node y is given by the equation $y = Ax + b$.*

Definition 2. *An LCC with a binary matrix $A_{n\times n}$ and an n-dimensional binary vector b is a linear-complement permutation (LCP) if $A_{n\times n}$ is nonsingular, i.e., $rank(A) = n$.*

Definition 3. *An LCC with a binary matrix $A_{n\times n}$ and an n-dimensional binary vector b is a linear-complement gather (LCG) if $rank(A) < n$.*

Scatter, the dual operation of gather, can be implemented by simply reversing the direction of message transmission of gather. Thus, we can define linear-complement scatter as follows:

Definition 4. *A communication is a linear-complement scatter (LCS), if there exists a binary matrix $A_{n\times n}$, $rank(A_{n\times n}) < n$, and an n-dimensional binary vector b such that, for every message with destination node y, its source node x is given by the equation $Ay + b = x$.*

Example 1. The bit-reverse communication operation on an 8-dimensional hypercube is a linear-complement communication,

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

For any source-destination pair (x, y) in an LCC defined above, to obtain the address of y is equivalent to performing a linear transformation of the n-dimensional vector space on x and then adding a constant binary vector. Since there is an one-to-one correspondence between the binary matrices of size $n \times n$ and linear transformations on an n-dimensional vector space over GF(2), we shall utilize this property in the following sections.

2.3 Routing Strategies

The interconnection network must allow every node to send messages to every other node. In the absence of a complete network, we need a routing algorithm to determine the path selected by a message to reach its destination. Efficient routing algorithms are critical to the performance of the interconnection network.

The e-cube routing algorithm is the simplest deadlock-free routing strategy on wormhole-routed hypercubes, which reserves the required channels in a strictly increasing order of dimensions. It allows messages to be forwarded only over channels at higher dimensions than that of the last traversed channel. Many current hypercube computers use the e-cube routing because of its simplicity and ease of implementation. However, the e-cube routing establishes only one shortest path between each pair of source-destination nodes and does not take the advantage of the flexibility provided by hypercubes.

Fully adaptive routing strategies can route messages along any of the shortest paths available in the hypercube network. Unfortunately, multiple channels are needed for a pair of neighboring nodes in order to help these strategies prevent deadlock. This means that extra hardware supports, such as buffer space and control logics, have to be added to the routers. Some partially adaptive routing strategies without the need of multiple channels have been proposed [5], [7], [11], [15]. Although they can better utilize the flexibility provided by hypercubes, the complexity of adaptive routers significantly increases the interrouter setup delay and flow control cycle times [6].

Furthermore, even though a partially adaptive routing strategy is used, our simulation results reveal that the average network latency grows much more rapidly than expected as the network throughput increases. This limits the utilization of the communication capacity of an interconnection network. For example, when the adaptive routing strategies in [7] or [15] is used, the throughput of performing bit-reverse on an 8-dimensional hypercube is always less than 1/8 of the communication capacity of the hypercube network. The main reason for the poor performance is that there exists a set of 8 source-destination pairs contending for the channel ($[00010000]^t$, $[00000000]^t$). Similar conditions also happen when performing matrix-transpose or reverse-flip. The adaptive routing strategy in [11] also suffers from similar problems when performing matrix-transpose or bit-reverse.

From the above discussions, it is shown that the adaptive routing algorithms cannot appropriately solve the contention problem for performing LCCs on hypercubes. Hence, in this paper, we propose a compiler approach called *processor reordering mapping* to minimize the channel contention. The hypercube network is assumed to support only e-cube routing. Thus, no extra hardware is needed and the proposed approach is of practical use.

3 The Compiler Approach

In the proposed compiler approach, the communications to be performed in a parallel program can be detected by compilers automatically or specified by programmers. These communications will be transformed into the matrix form for the optimization process. Next, according to the given communications, an optimal processor mapping is determined. The compiler can then generate the SPMD (Single Program Multiple Data) node program for each processor accordingly. By appropriately choosing a processor mapping, the new communications can be efficiently realized on the hypercube network.

As an example, consider executing the parallel loop show in Fig. 1a on hypercubes. The function $reverse(i, n)$ returns the bit-reverse of i, i.e.,

$$i_0 \times 2^{n-1} + i_1 \times 2^{n-2} + \ldots i_{n-2} \times 2 + i_{n-1},$$

for the loop index,

$$i = i_{n-1} \times 2^{n-1} + i_{n-2} \times 2^{n-2} + \ldots i_1 \times 2 + i_0.$$

If array elements $b[i]$ and $r[i]$ are distributed on processor P_i, and the task of executing iteration i is also assigned to processor P_i, then the compiler can determine that the communication to be performed is bit-reverse and transform it into the matrix form as shown in Example 1. The SPMD node program without processor mapping can be generated, as shown in Fig. 1b for comparison.

With processor mapping, the compiler has to determine an optimal one-to-one mapping function f according to the given communication. The function f maps *virtual processor* x onto *physical processor* $x' = f(x)$. From the view point of virtual processors, data distribution and iteration assignment remain unchanged, i.e., array elements $b[i]$ and $r[i]$ are distributed on virtual processor VP_i and the task of executing iteration i is also assigned to virtual processor VP_i. The only difference is that virtual

```
real b[2^n], r[2^n]
doall i = 0 to 2^n-1 do
    b[i] = r[reverse(i,n)]
enddo
```
(a)

```
real b_local, r_local
i = get_pid()
send(reverse(i,n), r_local)
receive(reverse(i,n), b_local)
```
(b)

```
real b_local, r_local
y = get_pid()
send(f(reverse(f^-1(y),n)), r_local)
receive(f(reverse(f^-1(y),n)), b_local)
```
(c)

```
real b_local, r_local
int to_virtual[2^n], to_physical[2^n]
i = v_get_pid()/* returns to_virtual[get_pid()] */
v_send(reverse(i,n), r_local)
    /* v_send(v_pid, data) calls send(to_physical[v_pid], data) */
v_receive(reverse(i,n), b_local)
    /* v_receive(v_pid, data) calls receive(to_physical[v_pid], data) */
```
(d)

Fig. 1. An example of a parallel loop and its corresonding SPMD node programs. (a) An example parallel loop. (b) An example SPMD node program. (c) An example SPMD node program for mapping f. (d) An alternative SPMD node program for mapping f.

processor VP_i is now mapped onto the physical processor $P_{f(i)}$. Accordingly, the SPMD node program for each processor can be generated, as shown in Fig. 1c. Though the communication to be performed between virtual processors is still bit-reverse, the one between physical processors is changed. It will be proved in the next section that the degree of channel contention can be reduced by appropriately choosing the mapping function f. Also, the algorithm for finding an optimal mapping function will be proposed.

Since the sending and receiving of data can be accomplished by hardware, as shown in Fig. 1b, the only overhead of the program shown in Fig. 1c at runtime is the mapping between virtual processors and physical processors. This overhead can be minimized as array indexing operations, as shown in Fig. 1d. Apparently, this overhead is far less than the communication latency and, therefore, can be neglected. Note also that the SPMD node program, shown in Fig. 1d, is very similar to the one shown in Fig. 1b, except that the functions get_pid(), send(), and receive() in Fig. 1b are replaced by the functions v_get_pid(), v_send(), and v_receive(), respectively. This property makes code generation with processor mapping much easier. In addition to generating the node program, the compiler has to determine an appropriate mapping function and set up the two mapping arrays to_virtual[] and to_physical[] for use in the node program.

Another issue arises when more than one communication will be performed in a parallel program. Our approach is to search for an optimal processor mapping for all the communications. We shall propose a dynamic programming algorithm for that purpose in Section 5. Another approach is to perform *processor remapping* at runtime. However, it may result in significant overhead at runtime. Thus, it is not considered in this paper.

4 Processor Reordering Mapping

First, we shall show that the degree of channel contention for an LCC is directly related to the ranks of submatrices of the binary matrix A of the LCC. A submatrix of the binary matrix A is the matrix obtained from A by retaining entries in some row(s) and column(s) and deleting other entries. We shall use $A_{(i)}$ to denote the ith row of A and $A^{(i)}$ to denote the ith column of A. The following defines the special submatrices and their notations to be used in this paper:

Definition 5. *A submatrix of the binary matrix A obtained by retaining rows in the set R and columns in the set C is denoted as $A_{R,C}$.*

We shall use L_i to denote the set of nonnegative integers smaller than i. Therefore, given integer i and j, A_{L_i,L_j} is the upper-left submatrix of A with i rows and j columns. Example 2 shows the submatrix A_{L_3,L_2} of a 4×4 matrix A.

Example 2. Given

$$A = \begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix}, \quad A_{L_3,L_2} = \begin{bmatrix} a_{0,0} & a_{0,1} \\ a_{1,0} & a_{1,1} \\ a_{2,0} & a_{2,1} \end{bmatrix}.$$

Theorem 1. *Given an LCC $y = Ax + b$, the maximum number of paths that contend for the same channel at dimension i, denoted as $T_i(A, b)$, can be determined as follows:*

$$T_i(A,b) = \begin{cases} 0 & \text{, when } x_i = y_i \\ 2^{i-rank\left(A_{L_{i+1},L_i}\right)} & \text{, otherwise.} \end{cases}$$

Proof. For any channel

$$l = ([z_0 z_1 \dots z_i \dots z_{n-1}]^t, [z_0 z_1 \dots \overline{z}_i \dots z_{n-1}]^t)$$

at dimension i, suppose that l is in the path from node $x = [x_0 x_1 \dots x_{n-1}]^t$ to node $y = [y_0 y_1 \dots y_{n-1}]^t$ according to the e-cube routing, then we have

$$[x_i x_{i+1} \dots x_{n-1}]^t = [z_i z_{i+1} \dots z_{n-1}]^t$$

and

$$[y_0 y_1 \dots y_i]^t = [z_0 z_1 \dots z_{i-1}\ \overline{z}_i]^t.$$

Since $y = Ax + b$, we have

$$\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_i \end{bmatrix} = \begin{bmatrix} z_0 \\ z_1 \\ \vdots \\ \bar{z}_i \end{bmatrix} = \begin{bmatrix} a_{0,0} & a_{0,1} & \cdots & a_{0,i-1} \\ a_{1,0} & a_{1,1} & \cdots & a_{1,i-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{i,0} & a_{i,1} & \cdots & a_{i,i-1} \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_{i-1} \end{bmatrix}$$

$$+ \begin{bmatrix} a_{0,i} & a_{0,i+1} & \cdots & a_{0,n-1} \\ a_{1,i} & a_{1,i+1} & \cdots & a_{1,n-1} \\ \cdots & \cdots & \cdots & \cdots \\ a_{i,i} & a_{i,i+1} & \cdots & a_{i,n-1} \end{bmatrix} \begin{bmatrix} z_i \\ z_{i+1} \\ \vdots \\ z_{n-1} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_i \end{bmatrix}.$$

Note that the number of paths contending for channel l is equivalent to the number of solutions of $[x_0 x_1 \ldots x_{i-1}]^t$. According to linear Algebra, either there is no solution or there are exactly $2^{i-rank(A_{L_{i+1},L_i})}$ distinct solutions satisfying the above set of equations. Note also that no solution means no path passing channel l. Hence, for all channels at dimension i, there is no solution if and only if $y_i = x_i$. In all other cases, there exists a channel at dimension i such that exactly $2^{i-rank(A_{L_{i+1},L_i})}$ paths contend for it. □

The above theorem shows that the degree of channel contention is determined only by the ranks of submatrices of the binary matrix A. As an example, we shall compute the degree of channel contention of matrix-transpose on an 8-dimensional hypercube according to Theorem 1.

Example 3. Consider the matrix-transpose communication on an 8-dimensional hypercube,

$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

According to Theorem 1, the degree of channel contention at each dimension can be computed as follows:

$$T_0 = 2^{0-0} = 1,\ T_1 = 2^{1-0} = 2,\ T_2 = 2^{2-0} = 4,$$
$$T_3 = 2^{3-0} = 8,\ T_4 = 2^{4-1} = 8,\ T_5 = 2^{5-3} = 4,$$
$$T_6 = 2^{6-5} = 2,\ T_7 = 2^{7-7} = 1.$$

Definition 6. *The degree of channel contention of an LCC $y = Ax + b$ is defined to be $\underset{0 \le i \le n-1}{MAX}\{T_i(A,b)\}$ and denoted by $T(A,b)$.*

To minimize $T(A,b)$, the binary matrix A must be "changed" and, at the same time, the LCC must be correctly performed. To meet these two requirements, we propose an alternative approach, called *processor reordering mapping*. In this approach, processors are logically reordered by a reordering mapping, which is a permutation of address bits of processors. It is defined formally as follows:

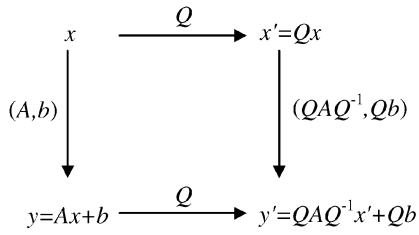


Fig. 2. The new communication after reordering mapping Q for $y = Ax + b$.

Definition 7. *A processor mapping is said to be a linear mapping if there exists a binary matrix $Q_{n\times n}$ such that, for every node x, it is mapped to node $x' = Qx$.*

Definition 8. *A processor mapping is a reordering mapping if it is a linear mapping and the matrix Q is a permutation matrix, i.e. each row and column of Q has exactly one 1.*

Reordering mapping has the property that the neighboring relation of processors is kept unchanged after processors are reordered. Since the communication between neighboring processors is the most frequently used class of communications, this property provides a great advantage for practical use. In order to ensure that the LCC will be performed correctly, the communication after processor reordering mapping must be changed as shown in the following theorem:

Theorem 2. *Given an LCC with a binary matrix A and a binary vector b, the new communication after the reordering mapping with a permutation matrix Q is an LCC with a binary matrix QAQ^{-1} and a binary vector Qb.*

Proof. Fig. 2 provides a good explanation for this theorem. For a source node x in the LCC with A and b, the destination node y can be computed by the equation $y = Ax + b$. After reordering mapping by Q, we have $x' = Qx$ and $y' = Qy$. Since Q is nonsingular, we can derive $x = Q^{-1}x'$. Hence, we have

$$y' = Qy = Q(Ax + b) = QAQ^{-1}x' + Qb.$$

□

In other words, the degree of channel contention after the reordering mapping is now determined by QAQ^{-1}. By choosing an appropriate processor mapping Q, the degree of channel contention could be greatly reduced. In the following discussion, we will show how to find an optimal reordering mapping for an LCC. First, notice the special situation in Theorem 1 where $T_i(A,b) = 0$ at dimension i. It only happens when $y_i = x_i$ for all x, i.e., $A_{(i)} = I_{(i)}$ and $b_i = 0$, where I is the *identity matrix*. Since $T_i(A,b) = 0$ means no communication at dimension i, we wish to keep it unchanged when performing processor mapping. It can be accomplished by applying a reordering mapping Q, which moves the ith row of A to the last dimension before searching for an optimal mapping. Hence, the ith row and column in A will be moved to the last dimension in QAQ^{-1}. Since

$$[QAQ^{-1}]_{L_{n-1},L_{n-1}} = A_{L_n-\{i\},L_n-\{i\}},$$

and

$$T_{n-1}(QAQ^{-1}, Qb) = T_i(A, b) = 0,$$

we can just consider the submatrix $A_{L_n-\{i\},L_n-\{i\}}$ when minimizing the channel contention of A. Without loss of generality, we may assume $T_i(A, b) > 0$ for $0 \le i \le n-1$ in the following discussion:

Theorem 3. *For any nonsingular binary matrix $A_{n\times n}$, there exists a permutation matrix $Q_{n\times n}$ such that*

$$rank([QAQ^{-1}]_{L_{i+1},L_i}) = i$$

for $0 \le i \le n-1$.

Proof. The proof of this theorem is by mathematic induction on the integer i. Obviously, it is true for $i = 0$. Suppose that it is true for $0 \le i \le k$. There must exist a permutation matrix Q such that

$$rank([QAQ^{-1}]_{L_{i+1},L_i}) = i$$

for $0 \le i \le k$.

We shall prove that it is also true for $0 \le i \le k+1$. Note that

$$\begin{aligned}(k+1) &\ge rank([QAQ^{-1}]_{L_{k+2},L_{k+1}}) \\ &\ge rank([QAQ^{-1}]_{L_{k+1},L_{k+1}}) \\ &\ge rank([QAQ^{-1}]_{L_{k+1},L_k}) \ = k.\end{aligned}$$

If $rank([QAQ^{-1}]_{L_{k+2},L_{k+1}}) \ne k+1$, it must be held that

$$rank([QAQ^{-1}]_{L_{k+2},L_{k+1}}) = rank([QAQ^{-1}]_{L_{k+1},L_{k+1}}) = k.$$

Since A is nonsingular and Q is a permutation matrix, we have

$$rank(QAQ^{-1}) = rank(A) = n.$$

This means the columns in QAQ^{-1} are linearly independent. Hence, we have

$$rank([QAQ^{-1}]_{L_n,L_{k+1}}) = k+1.$$

In other words, there must exist a row $j \ge k+1$ that is linearly independent of rows 0 to k in $[QAQ^{-1}]_{L_n,L_{k+1}}$. Let $\tilde{Q}$ be the permutation matrix exchanging rows j and $k+1$. Accordingly, $\tilde{Q}^{-1}$ must be the permutation matrix exchanging column j and $k+1$. We can derive

$$rank([\tilde{Q}(QAQ^{-1})\tilde{Q}^{-1}]_{L_{i+1},L_i}) = i$$

for $0 \le i \le k$, and

$$rank([\tilde{Q}(QAQ^{-1})\tilde{Q}^{-1}]_{L_{k+2},L_{k+1}}) = k+1.$$

Since $(\tilde{Q}Q)^{-1} = Q^{-1}\tilde{Q}^{-1}$, it can be derived that

$$rank(\lfloor(\tilde{Q}Q)A(\tilde{Q}Q)^{-1}\rfloor_{L_{i+1},L_i}) = i$$

for $0 \le i \le k+1$. Since $(\tilde{Q}Q)$ is also a permutation matrix, this theorem is true for $0 \le i \le k+1$. Therefore, by induction, this theorem is true for $0 \le i \le n-1$. This completes the proof of this theorem. □

```
/* Given: an n-dimensional LCP, y=Ax+b.
   Goal:  find an optimal reordering mapping Q and the new
          communication y'=Dx'+d, where D=QAQ^-1 and d=Qb.
   Initial values for Q, D, and d: Q=I, D=A, and d=b. */
1.  LCP_Optimizer(    GF_Matrix A, GF_Vector b,
                      GF_Matrix Q, GF_Matrix D,
                      GF_Vector d){
2.    GF_Matrix V=D;   /* V: for finding independent rows. */
3.    GF_Vector R=0;   /* To mark independent rows. */
4.    int i, j, k;
5.    for(i=0; i< n-1; i++){ /* At the beginning of iteration i, D
                    has already been optimal for dimension≤i
                    and will be optimized for dimension=i+1. */
6.      for(j=0; j<n; j++)/* To find a new independent row in
                             sub-matrix D_Ln,L(i+1). */
7.          if( V[j][i]!=0 && R[j]==0 ) break;
8.      if(j>i+1){  /* The new independent row is not in
                       D_L(i+2),L(i+1), so move it in. */
9.          RowSwap(D, i+1, j); ColSwap(D, i+1, j);
10.         RowSwap(d, i+1, j);
11.         RowSwap(V, i+1, j); ColSwap(V, i+1, j);
12.         RowSwap(R, i+1, j);
13.         RowSwap(Q, i+1, j);
14.         j=i+1;
15.     }
16.     for(k=0; k<n; k++)
17.         if( k!=j && V[k][i]!=0 )   /* V[k]= V[k] - V[j] */
18.             RowSub(V, j, k);
19.     R[j]=1;
20.   }
21. }
```

Fig. 3. The proposed algorithm for finding an optimal reordering mapping for an LCP.

Corollary 1. *For any LCP, there exists a reordering mapping such that the new communication has no channel contention.*

Following the method in the proof for Theorem 3, we can design an algorithm, as shown in Fig. 3, to find an optimal reordering mapping for any LCP. The input of the algorithm, LCP_Optimizer, is an LCP $y = Ax + b$ and the outputs are the optimal reordering mapping Q and the new LCP $y' = Dx' + d$, where $D = QAQ^{-1}$ and $d = Qb$. At the beginning of the LCP_Optimizer, D, d, and Q are set to A, b, and I, respectively. By appropriately exchanging the rows and the columns, the desired new LCP can be obtained. An important job of the LCP_Optimizer is to find a set of linear independent rows in some submatrix of D. In order to do that efficiently, a matrix V is used to find linearly independent rows in a way similar to Gaussian Elimination, and a vector R is used to mark those rows. $R[j]$ set to 1 means the jth row of some submatrix of D is linearly independent and so is the jth row of the corresponding submatrix of V. Initially, V is set to D and R is set to 0. During the processing of the LCP_Optimizer,

$$Rank(D_{L_p,L_q}) = Rank(V_{L_p,L_q})$$

for any (p, q).

The loop from line 5 to line 17 is the main part of the LCP_Optimizer. At the beginning of iteration i, D has already been optimal for those dimensions less than or equal to i, i.e.,

$$Rank(D_{L_{r+1},L_r}) = Rank(V_{L_{r+1},L_r}) = r$$

for $0 \le r \le i$, and will be optimized for dimension $(i+1)$. In the meantime, the selected linearly independent rows of V_{L_n,L_i} are all in V_{L_{i+1},L_i} and marked by R. Each of those rows contains only one 1 and the other elements are 0. All the other rows in V_{L_n,L_i} are zero rows. Hence, to find a new linearly independent row in $D_{L_n,L_{i+1}}$ and $V_{L_n,L_{i+1}}$, we only need to examine column i of $V_{L_n,L_{i+1}}$ and R as shown at lines 6 and 7. If row j is the newly found independent row and $j > i+1$, it will be exchanged with row $(i+1)$ so that

$$Rank(D_{L_{i+2},L_{i+1}}) = Rank(V_{L_{i+2},L_{i+1}}) = i+1.$$

The required row and column exchanging operations are done from line 8 to line 13. At the end of iteration i, except the newly found independent row, all the elements in column i of $V_{L_n,L_{i+1}}$ will be cleared to 0 by subtracting from the new independent row as shown at lines 14 and 15. Therefore, V will be ready to be used in the next iteration. Since each row or column operation takes $O(n)$ execution time, the complexity of the LCP_Optimizer can be proved to be $O(n^3)$.

Example 4. Consider the matrix-transpose communication on an 8-dimensional hypercube as shown in Example 3. From Fig. 3, we can compute the optimal reordering mapping and the new LCP as follows:

$$Q = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&0&0&1&0\\ 0&1&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix}$$

and

$$\begin{bmatrix} y'_0\\ y'_1\\ y'_2\\ y'_3\\ y'_4\\ y'_5\\ y'_6\\ y'_7 \end{bmatrix} = \begin{bmatrix} 0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&0 \end{bmatrix} \begin{bmatrix} x'_0\\ x'_1\\ x'_2\\ x'_3\\ x'_4\\ x'_5\\ x'_6\\ x'_7 \end{bmatrix} + \begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}.$$

The degree of channel contention at each dimension for the new LCP becomes

$$T_0 = T_1 = T_2 = T_3 = T_4 = T_5 = T_6 = T_7 = 2^0 = 1.$$

Compared with Example 3, the degree of channel contention is greatly reduced from 8 to 1, i.e., contention-free by the reordering mapping.

Theorem 4. *For any binary matrix $A_{n\times n}$, $rank(A_{n\times n}) < n$, there exists a permutation matrix $Q_{n\times n}$ such that*

$$i - rank([QAQ^{-1}]_{L_{i+1},L_i}) \le (n-1) - rank(A)$$

for $0 \le i \le n-1$.

Proof. Since $rank(A_{n\times n}) < n$, there exists a column j in A such that column j is a linear combination of other columns. Let P be the permutation matrix exchanging rows j and $n-1$. Accordingly, P^{-1} must be the permutation matrix exchanging column j and $n-1$. Let $B = PAP^{-1}$, we have

$$rank(B_{L_n,L_{n-1}}) = rank(B) = rank(A).$$

In the following, we will prove that there exists a permutation matrix Q such that

$$i - rank([QBQ^{-1}]_{L_{i+1},L_i}) \le (n-1) - rank(B_{L_n,L_{n-1}})$$

for $0 \le i \le n-1$.

The proof is by mathematic induction on the integer i. Obviously, it is true for $i = n-1$. Suppose that it is true for $k \le i \le n-1$. There must exist a permutation matrix Q such that

$$i - rank([QBQ^{-1}]_{L_{i+1},L_i}) \le (n-1) - rank(B_{L_n,L_{n-1}})$$

for $k \le i \le n-1$.

We shall prove that it is also true for $k-1 \le i \le n-1$. Consider $i = k-1$ for the matrix QBQ^{-1}. If

$$(k-1) - rank([QBQ^{-1}]_{L_k,L_{k-1}}) > (n-1) - rank(B_{L_n,L_{n-1}}),$$

it must be held that

$$(k-1) - rank([QBQ^{-1}]_{L_k,L_{k-1}}) > k - rank([QBQ^{-1}]_{L_{k+1},L_k}).$$

That is

$$rank([QBQ^{-1}]_{L_{k+1},L_k}) - 1 > rank([QBQ^{-1}]_{L_k,L_{k-1}}).$$

So, we can derive

$$rank([QBQ^{-1}]_{L_k,L_k}) > rank([QBQ^{-1}]_{L_k,L_{k-1}}),$$

and

$$k - 1 > rank([QBQ^{-1}]_{L_k,L_{k-1}}).$$

This means that, in $[QBQ^{-1}]_{L_k,L_k}$, the column $(k-1)$ is linearly independent of other columns and there must exist a column j, $0 \le j \le k-2$, which is a linear combination of other columns. Let $\tilde{Q}$ be the permutation matrix exchanging rows j and $k+1$. Accordingly, $\tilde{Q}^{-1}$ must be the permutation matrix exchanging column j and $k+1$. We can derive

$$i - rank([\tilde{Q}(QBQ^{-1})\tilde{Q}^{-1}]_{L_{i+1},L_i}) \le (n-1) - rank(B_{L_n,L_{n-1}})$$

for $k \le i \le n-1$, and

$$\begin{aligned}(k-1) - &rank([\tilde{Q}(QBQ^{-1})\tilde{Q}^{-1}]_{L_k,L_{k-1}})\\ &= (k-1) - rank([QBQ^{-1}]_{L_k,L_k})\\ &\le k - rank([QBQ^{-1}]_{L_{k+1},L_k})\\ &\le (n-1) - rank(B_{L_n,L_{n-1}}).\end{aligned}$$

Since $(\tilde{Q}Q)$ is also a permutation matrix, the inequality is true for $k-1 \le i \le n-1$. Hence, by mathematic induction, it can be proved that there exists a permutation matrix Q such that

$$i - rank([QBQ^{-1}]_{L_{i+1},L_i}) \le (n-1) - rank(B_{L_n,L_{n-1}})$$

for $0 \le i \le n-1$. Therefore,

$$i - rank([Q(PAP^{-1})Q^{-1}]_{L_{i+1},L_i} \le (n-1) - rank([PAP^{-1}]_{L_n,L_{n-1}})$$

for $0 \le i \le n-1$. In other words,

$$i - rank([(QP)A(QP)^{-1}]_{L_{i+1},L_i}) \le (n-1) - rank(A)$$

for $0 \le i \le n-1$. Since (QP) is also a permutation matrix, this completes the proof of Theorem 4. □

Corollary 2. *For any LCG or LCS, the new communication with the reordering mapping in Theorem 4 has minimum channel contention.*

Proof. Given an LCG or LCS with binary matrix $A_{n\times n}$, for any reordering mapping Q,

$$\underset{0\le i\le n-1}{MAX}\{i - rank([QAQ^{-1}]_{L_{i+1},L_i})\} \ge (n-1) - rank(A).$$

From Theorem 4, there exists a permutation matrix $\tilde{Q}$, such that

$$i - rank([\tilde{Q}A\tilde{Q}^{-1}]_{L_{i+1},L_i}) \le (n-1) - rank(A)$$

for $0 \le i \le n-1$. Therefore, by Theorem 4, we can find an optimal reordering mapping such that the new communication has minimum channel contention. □

Similar to the LCP_Optimizer, an algorithm for LCS or LCG can be easily derived following the method in the proof for Theorem 4. It is omitted here to save the space.

5 Processor Mapping for a Set of LCCs

From the previous section, we can find an optimal reordering mapping for any LCC. However, there is probably more than one communication in a parallel program. For efficiently executing such a parallel program, we have to deal with the problem of performing a set of LCCs. An example of a set of LCCs is given below, which shows some frequently used subroutines that may be in a library for image processing, including image rotation, reflection, FFT, etc.

Example 5. Suppose a 16×16 image is distributed on an 8-dimensional hypercube such that the pixel at coordinates (px, py) is on the node $(py \times 16 + px)$. Some frequently used subroutines for processing such an image are shown as follows:

1. To reflect on the diagonal, each pixel (px, py) has to be moved to (py, px), i.e., sent from node $(py \times 16 + px)$ to $(px \times 16 + py)$. The communication on the hypercube is an LCC, $y = A_1x + b_1$, which is the matrix transpose shown in Example 3. Similarly, to reflect on the line $py = 15 - px$, the required communication is $y = A_1x + [11111111]^t$. To rotate 90 degrees clockwise, the required communication is $y = A_1x + [11110000]^t$. And, to rotate 90 degrees counterclockwise, the required communication is $y = A_1x + [00001111]^t$.
2. To reflect on a vertical line, each pixel (px, py) has to be moved to $(15 - px, py)$. The required communication is $y = Ix + [11110000]^t$. Similarly, to reflect on a horizontal line, the required communication is $y = Ix + [00001111]^t$. To rotate 180 degrees, the required communication is $y = Ix + [11111111]^t$.
3. To scale the lower-left 8×8 subimage by a factor of 2 along both of the axes, each pixel (px, py) in the subimage has to be sent to four new positions:

$$(2px, 2py),\ (2px+1, 2py),$$
$$(2px, 2py+1),\ (2px+1, 2py+1).$$

The required communication is an LCS,

$$\begin{bmatrix} 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0 \end{bmatrix} \begin{bmatrix} y_0\\ y_1\\ y_2\\ y_3\\ y_4\\ y_5\\ y_6\\ y_7 \end{bmatrix} + \begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix} = \begin{bmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\\ x_7 \end{bmatrix}.$$

4. Consider those pixels of the image to be 256 discrete data and perform 1D FFT on these data. In addition to the neighboring communications, a bit-reverse communication is required [8], [13]. The bit-reverse communication, $y = A_3x + b_3$, is the one shown in Example 1.
5. To perform 2D FFT on the image, the process is to perform 1D FFT for each row of the image and then perform 1D FFT for each column of the image [13]. For performing 1D FFT on each row of the image, in addition to the neighboring communications, the bit-reverse communication for the rows is required. Similarly, for performing 1D FFT on each column of the image, the bit-reverse communication for the columns is required. They are shown as follows:

$$y = \begin{bmatrix} 0&0&0&1&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&1 \end{bmatrix} \begin{bmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\\ x_7 \end{bmatrix} + \begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}$$

$$y = \begin{bmatrix} 1&0&0&0&0&0&0&0\\ 0&1&0&0&0&0&0&0\\ 0&0&1&0&0&0&0&0\\ 0&0&0&1&0&0&0&0\\ 0&0&0&0&0&0&0&1\\ 0&0&0&0&0&0&1&0\\ 0&0&0&0&0&1&0&0\\ 0&0&0&0&1&0&0&0 \end{bmatrix} \begin{bmatrix} x_0\\ x_1\\ x_2\\ x_3\\ x_4\\ x_5\\ x_6\\ x_7 \end{bmatrix} + \begin{bmatrix} 0\\0\\0\\0\\0\\0\\0\\0 \end{bmatrix}.$$

Since a reordering mapping that is good for some LCCs may be harmful for others, our goal is to find a reordering mapping that is good enough for all the LCCs to be performed in a parallel program. For different applications, the objective functions of optimization may also be different. Let $y = A_r x + b_r$, $0 \le r \le m-1$ be the m LCCs to be performed. In some applications, these LCCs are in different subroutines that will be called dynamically. Example 5 shows one of such applications. Since we cannot determine at the compile-time which and how many times the subroutines will be called, a reasonable choice is to minimize the maximum channel contention of these subroutines, i.e., minimize

$$\underset{0\le r\le m-1}{MAX}(T(A_r, b_r)).$$

In some other applications, those LCCs may require to be performed simultaneously. Such a situation may happen when messages are scatter-gathered among processors. Sometimes, the interference between different communications may also lead to this situation. Since those LCCs are performed simultaneously, it is appropriate to minimize

$$\underset{0\le i\le n-1}{MAX}\{\underset{0\le r\le m-1}{SUM}(T_i(A_r, b_r))\}.$$

There are other possible concerns, such as minimizing

$$\underset{0\le i\le n-1}{SUM}\{\underset{0\le r\le m-1}{SUM}(T_i(A_r, b_r))\}.$$

In this section, we propose an algorithm to find an optimal reordering mapping based on the dynamic programming approach. The concept of dynamic programming can be applied to any of the three objective functions in a similar way. Without loss of generality, we focus on the problem of minimizing

$$\underset{0\le r\le m-1}{MAX}(T(A_r, b_r)),$$

i.e., finding an optimal reordering mapping Q that minimizes

$$\underset{0\le r\le m-1}{MAX}\{\underset{0\le i\le n-1}{MAX}\{i - rank([QA_rQ^{-1}]_{L_{i+1},L_i})\}\}.$$

Though this problem can be solved by an exhaustive search on all possible reordering matrices, the largest number of all possible reordering matrices, $n!$, makes such a solution unfeasible for a large hypercube computer. Therefore, we present an algorithm based on the technique of dynamic programming to reduce the search space.

Since the matrix Q is a permutation matrix, there is a one-to-one correspondence between reordering mapping and the order of address bits. We shall use $R(S)$ to denote an optimal ordering of the address bits in the set S. In other words, $R(S)$ defines an optimal reordering mapping for $[A_r]_{S,S}$, $0 \le r \le m-1$. For example, the corresponding order of address bits for the reordering mapping Q in Example 4 is $R(L_8) = (0, 4, 2, 6, 1, 5, 3, 7)$. The following theorem provides the theoretical foundation for applying dynamic programming to reduce the search space.

Theorem 5. *There exists an address bit j in S such that $(R(S-\{j\}), j)$ is an optimal ordering of address bits for $[A_r]_{S,S}$, $0 \le r \le m-1$.*

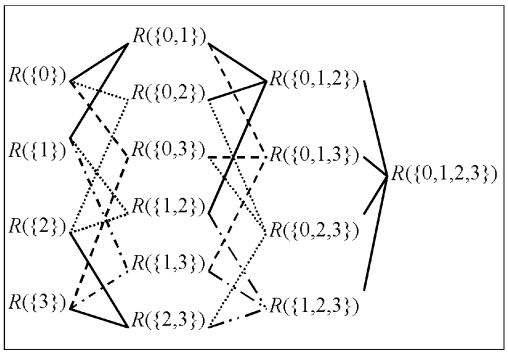


Fig. 4. Search steps for $R(L_4)$.

Proof. Let s be the cardinality of S. Suppose that j is the last address bit of $R(S)$. Thus, the maximum number for channel contention at dimension $s-1$ must be the same for $R(S)$ and $(R(S-\{j\}), j)$. Since $R(S-\{j\})$ is optimal for $[A_r]_{S-\{j\},S-\{j\}}$, $(R(S-\{j\}), j)$ is optimal for $[A_r]_{S,S}$ at dimensions 0 to $s-2$. Therefore, $(R(S-\{j\}), j)$ must be an optimal reordering for $[A_r]_{S,S}$. □

According to Theorem 5, we can designate $R(S)$ as an optimal ordering chosen from $(R(S-\{i\}), i)$ for all i in S and find $R(L_n)$ by computing $R(S)$ for all subsets $S \subset L_n$. The computing of $R(S)$, for $S \subset L_n$, can be performed according to the cardinality of S in increasing order. For any $S \subset L_n$, the number of searches is equivalent to its cardinality. Therefore, we can compute the total number of searches as follows:

$$\sum_{k=1}^{n} k \cdot C_k^n = \sum_{k=1}^{n} n \cdot C_{k-1}^{n-1} = n \cdot 2^{n-1}.$$

It can be observed that the search space is reduced from $n!$ to $n \cdot 2^{n-1}$. Fig. 4 gives the concept for computing $R(L_4)$. Since the running time for computing each case in the search space is $O(m \times n^3)$, the running time for computing $R(L_n)$ is $O(m \times n^4 \times 2^n)$. Although the running time is not polynomial in terms of the dimension n, it is polynomial in terms of the number of processors. Hence, the algorithm is feasible even for a 16-dimensional hypercube.

Example 6. Consider the two LCCs A_1 and A_3 in Example 5. We know that A_1 is for the matrix-transpose operation and A_3 is for the bit-reverse operation on an 8-dimensional hypercube. From the proposed dynamic programming approach, we can find that $R(L_8) = (3, 4, 0, 7, 2, 5, 1, 6)$. Hence, we can compute

$$Q = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix},$$

$$Q^{-1} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

and the two new communications are

$$\begin{bmatrix} y\prime_0 \\ y\prime_1 \\ y\prime_2 \\ y\prime_3 \\ y\prime_4 \\ y\prime_5 \\ y\prime_6 \\ y\prime_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x'_0 \\ x'_1 \\ x'_2 \\ x'_3 \\ x'_4 \\ x'_5 \\ x'_6 \\ x'_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

and

$$\begin{bmatrix} y'_0 \\ y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \\ y'_5 \\ y'_6 \\ y'_7 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x'_0 \\ x'_1 \\ x'_2 \\ x'_3 \\ x'_4 \\ x'_5 \\ x'_6 \\ x'_7 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}.$$

After the processor mapping, the degrees of channel contention now become 2 for matrix transpose and 1 for bit-reverse instead of 8 in the original communications.

6 Performance Study

To investigate the performance improvement of the proposed approach, some experiments were carried out by simulating the network behavior of an 8-dimensional hypercube. In Section 6.1, we compare the e-cube routing and some partially adaptive routing strategies with our approach. These partially adaptive routing strategies include P-cube routing proposed by Glass and Ni [11], the routing strategy proposed by Chiu et al. [7], minimal (Min) routing proposed by Li [15], and MIXab3 and MIXbb3 routing proposed by Chen and Yihng [5]. This simulation is based on those performed in [5] and [11]. In Section 6.2, a practical application, the parallel FFT program, is performed on our simulator to show the benefit of the proposed approach.

The network architecture assumed in the simulation is described as follows: There are 256 nodes connected as an 8-dimensional hypercube. Each node consists of a processor, local memory, a router, and other supporting devices. Between two neighboring routers, there are two unidirectional channels, one for each direction. A router can communicate with its local processor through pairs of ports. A separate buffer with a slot for one flit is associated with each channel. When more than one input channel contains header flits waiting for the same available output channel, the arbitration policy is in favor of the header flit that arrived at the router first. If a header flit in an input channel has more than one available output channel allowed by the routing strategy, the channel with the lowest dimension is selected.

6.1 Simulation for Three Communications

Network performance is significantly affected by the communications, which are application-dependent. In the following discussion, we consider three communications: Matrix-transpose, bit-reverse, and reverse-flip. They are chosen not only because they are frequently used in many scientific and engineering applications but also because they are used as tested cases in many routing algorithms, so we can compare our simulation with their results. For matrix-transpose, every node $x = [x_0x_1x_2x_3x_4x_5x_6x_7]^t$ sends messages to node $y = [x_4x_5x_6x_7x_0x_1x_2x_3]^t$. For bit-reverse, node $x = [x_0x_1x_2x_3x_4x_5x_6x_7]^t$ sends messages to node $y = [x_7x_6x_5x_4x_3x_2x_1x_0]^t$. Reverse-flip behaves like bit-reverse except the address bits of the destination node were complemented, i.e., node $x = [x_0x_1x_2x_3x_4x_5x_6x_7]^t$ sends messages to node $y = [\overline{x}_7, \overline{x}_6, \overline{x}_5, \overline{x}_4, \overline{x}_3, \overline{x}_2, \overline{x}_1, \overline{x}_0]^t$. All three communications are LCPs. We may find reordering mapping for them and see how the performance can be improved.

In the simulation, processors generate messages at time intervals given by a negative exponential distribution random variable. Each message is assumed to have 20 flits, including the header (flits). A flit requires a cycle to be transmitted through a channel. The measures of interest in this section are average message latency and average sustainable network throughput. The message latency is the number of cycles spent by a message in traveling from its source processor to its destination, taking the queuing delay into account. The average network throughput indicates the average number of flits delivered per cycle per processor. It is sustainable if the number of messages queued at their source processors is small and bounded. For a given system, the average message latency, in general, grows as the throughput increases. At low throughput, the network latency is contributed mainly by the message length and the distance to travel because there is little queuing delay involved. As the throughput increases, more channel contention and longer queuing delay happened, giving rise to a higher message latency. One system exhibits better communication performance than another if it has a lower message latency for any given throughput.

Figs. 5, 6, and 7 show the simulation results of matrix-transpose, bit-reverse, and reverse-flip, respectively. In these figures, M-S denotes the simulation result using an optimal reordering mapping for the specific communication, which is proved to be contention-free in Section 4, and M-G uses the reordering mapping chosen for the three communications by using the dynamic programming algorithm proposed in Section 5. After the processor mapping M-G, the degree of channel contention for matrix-transpose is 2 and it is contention-free for bit-reverse and reverse-flip.

In Figs. 5, 6, and 7, it can be observed that the routing strategies MIXab3, MIXbb3, Chiu and Min indeed improve the performance over the e-cube routing since the difference of the interrouter setup delay and flow control cycle time

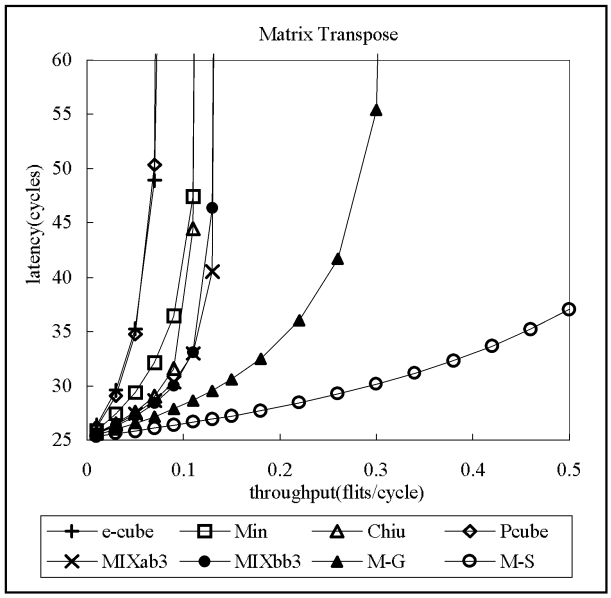


Fig. 5. Simulation result of matrix-transpose on an 8-dimensional hypercube.

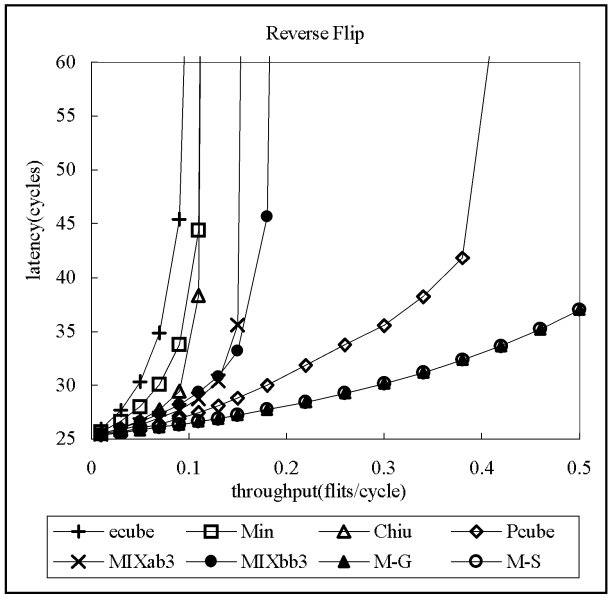


Fig. 7. Simulation result of reverse-flip on an 8-dimensional hypercube.

are not considered in this experiment [6]. The maximum sustainable network throughput of these routing strategies is about 30 percent to 100 percent higher than that of the e-cube routing. Their message latencies are also lower for any given sustainable throughput. The P-cube routing performs quite different for the three communications. It performs well for reverse-flip but worst for bit-reverse. From these results, it can be observed that it is very difficult for a routing algorithm to perform well for all communication patterns.

It can also be observed that, for any of the three communications, the network throughput of the e-cube routing is always less than 0.125, which is 1/8 of the network capacity. As we have pointed out in Section 2, this is due to the degree of channel contention being 8 for the e-cube routing. In other words, the maximum throughput that can be achieved is approximately inversely proportional to the degree of channel contention.

After applying M-S and M-G, there is no channel contention for performing the three communications, except performing matrix-transpose after applying M-G. The degree of channel contention is 2 for matrix-transpose after applying M-G. Theoretically their throughput can approach 1 and 0.5, respectively. The actual value is somewhat smaller because of the queuing delay between messages generated by the same processor. As shown in Fig. 5, its value is about 0.3 instead of 0.5 for M-G. Note also that, for any given sustainable network throughput, the message latencies of M-G and M-S are far lower than that of traditional approaches.

With these results, it is obvious that our approach can greatly reduce the network latency and significantly improve the throughput for LCC. Furthermore, no extra hardware supports it and sophisticated routing strategies are needed. Only the e-cube wormhole routing is assumed in the proposed approach. Therefore, it is of practical use.

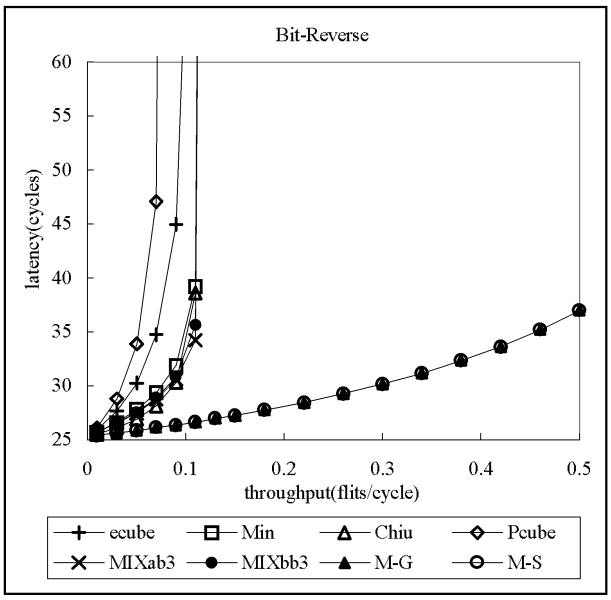


Fig. 6. Simulation result of bit-reverse on an 8-dimensional hypercube.

6.2 Simulation for FFT

The Fast Fourier Transform (FFT) is one of the most commonly used algorithms in digital signal processing and is widely used in applications such as image processing and spectral analysis. The purpose of this section is to investigate the benefit of the proposed approach for such a practical application.

The Discrete Fourier Transform (DFT) of an m-point discrete signal $x(i)$ is defined by

$$X(k) = \sum_{i=0}^{m-1} x(i) W_m^{ik},$$

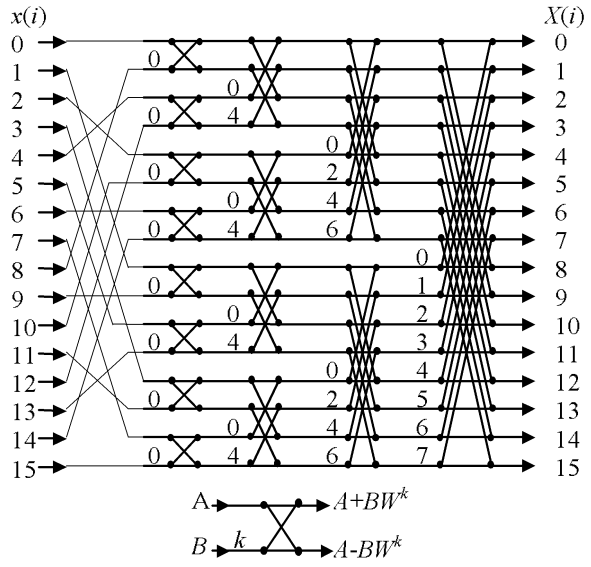


Fig. 8. Flow chart of a 16-point FFT.

$0 \le k \le m$, where $W_m = e^{-j2\pi/m}$, and $j = \sqrt{-1}$. Direct DFT computation requires $O(m^2)$ arithmetic operations. A faster method of computing the DFT is the FFT algorithm, which requires only $O(m \lg m)$ arithmetic operations. A more detailed analysis of FFT can be found in [8]. Fig. 8 shows an example of the flow chart of the FFT algorithm for 16 points. The FFT algorithm begins with a bit-reverse permutation of the inputs, followed by $\lg m$ stages, each stage consisting of $m/2$ butterfly operations. An FFT input x can be identified by a binary vector $[x_0x_1x_2 \ldots x_{\lg(m-1)}]^t$. In the ith computational stage, the two inputs of a butterfly operation are $[x_0x_1 \ldots x_{i-1} \ldots x_{\lg(m-1)}]^t$ and $[x_0x_1 \ldots \overline{x}_{i-1} \ldots x_{\lg(m-1)}]^t$. We can exploit some properties of the FFT algorithm to produce an efficient parallel algorithm. The Parallel_FFT algorithm is described in the following paragraph.

Let the number of data points $m = 2^{n+2d}$, where n is the dimensions of the hypercube network, and d is a positive integer. Each input

$$x = [x_0x_1 \ldots x_dx_{d+1} \ldots x_{n+d-1}x_{n+d} \ldots x_{n+2d-1}]^t$$

is assigned to processor $[x_dx_{d+1} \ldots x_{n+d-1}]^t$. Hence, the m inputs are distributed on the 2^n processors with *block-cyclic distribution*. There are 2^d inputs in each block and 2^d blocks in each processor. The Parallel_FFT algorithm follows the $(1 + \lg m)$ stages in the FFT algorithm. In the first stage, a bit-reverse communication between processors is required for completing a bit-reverse permutation of the inputs. In the following $n + 2d$ computational stages, the first and the last d stages do not require communication operations and a neighboring communication is needed for each of the other n stages. The reasons are explained as follows: In the ith computational stage, $1 \le i \le d$, the two inputs of a butterfly operation,

$$[x_0x_1 \ldots x_{i-1} \ldots x_d \ldots x_{n+2d-1}]^t$$

and

$$[x_0x_1 \ldots \overline{x}_{i-1} \ldots x_d \ldots x_{n+2d-1}]^t,$$

are in the same processor, $[x_dx_{d+1} \ldots x_{n+d-1}]^t$. Hence, the data required for computation are all in local memory and no communication is needed. Similarly, no communication is needed in the ith computational stage,

$$n + d + 1 \le i \le n + 2d.$$

In the computational stage i, $d + 1 \le i \le n + d$, the two inputs of a butterfly operation,

$$[x_0x_1 \ldots x_d \ldots x_{i-1} \ldots x_{n+d-1} \ldots x_{n+2d-1}]^t$$

and

$$[x_0x_1 \ldots x_d \ldots \overline{x}_{i-1} \ldots x_{n+d-1} \ldots x_{n+2d-1}]^t$$

are in processors,

$$[x_d \ldots x_{i-1} \ldots x_{n+d-1}]^t$$

and

$$[x_d \ldots \overline{x}_{i-1} \ldots x_{n+d-1}]^t,$$

respectively. Therefore, a neighboring communication at dimension $i-(d+1)$ is performed. After these $(1 + \lg m)$ stages, the FFT outputs can be obtained and are also distributed on the 2^n processors with block-cyclic distribution as inputs. The Parallel_FFT algorithm requires $O((m/2^n)\lg m)$ arithmetic operations and $(n + 1)$ communication operations for each processor.

By processor reordering mapping, the channel contention of the bit-reverse communication in the first stage of the Parallel_FFT algorithm can be obviated and those neighboring communications will stay unchanged. To see the performance improvement of processor mapping for the Parallel_FFT algorithm, we simulated the algorithm on an 8-dimensional hypercube. Each input of FFT is a complex number which consists of two double precision floating-point numbers, one for the real part and the other for the imaginary part. The characteristics of the hypercube computer are based on $nCUBE\ 2$. The software latency is about 164 μs for a message. The time required for transmitting one byte through a channel is about 0.57μs. A butterfly operation requires about 5.12 μs provided that the value of W^i can be found in a precomputed table. If only half of a butterfly operation is performed in a processor, about 4.47 μs is required.

The simulation results are shown in Table 1. The computation time and neighboring communication time are the same for all the routing strategies and are not changed after processor reordering mapping. Bit-reverse communication time is the most important part in this comparison. We can observe that the performances of those partially adaptive routing strategies are not as good as expected and even worse than e-cube routing. This situation may be caused by the channel contention between the neighboring communications and the bit-reverse communication since there is no barrier synchronization in the program. The benefit of processor reordering mapping can be easily observed here because the bit-reverse communication time is greatly reduced.

TABLE 1
Simulation Result for the Parallel_FFT Algorithm

FFT Size		2^8	2^{10}	2^{12}	2^{14}
Computation Time		35.8	163.5	736.0	3271.7
Neighboring COMM Time		1398.6	1617.5	2493.0	5995.1
BR COMM Time	e-cube	248.9	467.8	1343.3	4845.4
	chiu	276.9	577.8	1781.7	6597.0
	min	248.4	467.2	1342.8	4844.8
	mixab3	258.1	504.3	1489.3	5429.1
	mixbb3	258.6	504.9	1489.8	5429.7
	pcube	299.7	693.5	2225.7	8354.3
	mapping	178.8	206.2	315.6	753.4

TABLE 2
Speedup After Optimal Reordering Mapping

FFT Size		2^8	2^{10}	2^{12}	2^{14}
Computation Time		35.8	163.5	736.0	3271.7
Neighboring COMM Time		1398.6	1617.5	2493.0	5995.1
BR COMM Time	Original	248.9	467.8	1343.3	4845.4
	Mapping	178.8	206.2	315.6	753.4
	Speedup	1.39	2.27	4.26	6.43
EXEC Time	Original	1683.3	2248.9	4572.4	14112.2
	Mapping	1613.2	1987.2	3544.7	10020.2
	Speedup	1.04	1.13	1.29	1.41

Table 2 shows the speedup after applying the processor reordering mapping. The ideal speedup of the bit-reverse communication time is $2^{(n/2-1)}$ for an n-dimensional hypercube because the degree of contention is decreased from $2^{(n/2-1)}$ to 1. However, the speedup in the simulation result is smaller than the ideal speedup due to the effect of the software latency. When the number of data points m is small, the software latency dominates the communication time. Hence, the performance improvement is not evident. As m increases, the message size also increases, and the effect of channel contention becomes more critical for the communication time. Therefore, the performance improvement of processor mapping becomes more significant as m increases. From Table 2, it can be observed that the speedup of the bit-reverse communication time reaches 6.43 for $m = 2^{14}$. The overall execution time is about 41 percent faster after by applying processor mapping.

7 Conclusions

In this paper, we address the problem of minimizing the maximum number of paths contending for the same channel when performing LCC on e-cube wormhole-routed hypercubes. A new approach, called processor reordering mapping, is proposed to solve this problem. We have proved that, for any LCC, there exists a reordering mapping such that the new communication after processor reordering has minimum channel contention. An $O(n^3)$ algorithm is proposed to find such a mapping for an n-dimensional hypercube. As for a set of LCCs, an algorithm based on dynamic programming is proposed to search for an optimal reordering mapping. It can greatly reduce the search space and thus is feasible even for a large hypercube computer. Simulation results clearly show significant performance improvement provided by the proposed approach when compared with partially adaptive routing strategies. With these results, compiler techniques can be used to reduce the message latency without the need of extra hardware costs.

Acknowledgments

The authors would like to thank the anonymous referees for their helpful suggestions. This research is supported in part by the National Science Council, Republic of China, under Grant NSC87-2213-E-001-005.

References

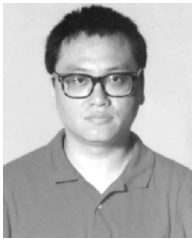
[1] *nCUBE 2 Supercomputers Manual.* NCUBE Company, 1990.
[2] *Origin™ Servers Technical Report.* Silicon Graphics, Inc. 1998.
[3] S. Abraham and K. Padmanabhan, "Performance of the Direct Binary n-Cube Network for Multiprocessors," *IEEE Trans. Computers,* vol. 38, no. 7, pp. 1000-1011, July 1989.
[4] R. Boppana and C.S. Raghavendra, "Optimal Self-Routing of Linear-Complement Permutations in Hypercubes," *Proc Fifth Distributed Memory Computing Conf.,* pp. 800-808, Apr. 1990.
[5] H.L. Chen and H.S. Yihng, "Generalized Wormhole Routing Strategies in Hypercubes," *J. Information Science and Eng.,* vol. 10 pp. 387-341, 1994.
[6] A.A. Chien, "A Cost and Speed Model for k-ary n-cube Wormhole Routers," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 2, pp. 150-162, Feb. 1998.
[7] G.M. Chiu, S. Chalasani, and C.S. Raghavendra, "Flexible, Fault-Tolerant Routing Criteria for Circuit-Switched Hypercubes," *Proc. IEEE 11th Int'l Conf. Distributed Computing Systems,* pp. 582-589, 1991.
[8] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms.* MIT Press, 1990.
[9] W.J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Trans. Computers,* vol. 39, no. 6, pp. 775-785, June 1990.
[10] W.J. Dally and C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Trans. Computers,* vol. 36, no. 5, pp. 547-553, May 1987.
[11] C.J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *J. ACM,* vol. 41, no. 5, pp. 874-902, Sept. 1994.
[12] Y. Hou, C.-M. Wang, and L.-S. Hsu, "Optimal Processor Mapping for Linear-Complement Communication on Hypercubes and Their Variations," Technical Report TR-IIS-00-014, Inst. Information Science, Academia Sinica, Taiwan, R.O.C., Nov. 2000. http://www.iis.sinica.edu.tw/LIB/TechReport/tr00014.ps.gz.
[13] R.C. Gonzalez and R.E. Woods, *Digital Image Processing.* Addison-Wesley Inc. 1992.
[14] F.T. Leighton, *Introduction to Parallel Algorithm and Architectures: Arrays, Trees, Hypercubes.* San Mateo, Calif.: Morgan Kaufmann, 1992.
[15] Q. Li, "Minimum Deadlock-Free Message Routing Restrictions in Binary Hypercubes," *J. Parallel and Distributed Computing,* vol. 15, no. 2, pp. 153-159, 1992.
[16] F.C. Lin and F.H. Wang, "Minimum Deadlock-Free Message Routing Restrictions in Binary Hypercubes," *J. Parallel and Distributed Computing,* vol. 29, no. 2, pp. 27-42, 1995.
[17] H. Masuyama, "Algorithms to Realize an Arbitrary BPC Permutation in Chordal Ring Networks and Mesh Connected Networks," *IEICE Trans. Inf. Syst. (Japan),* vol. E77-D, no. 10, pp. 1118-1129, Oct. 1994.
[18] H. Masuyama, Y. Morita, and E. Masuyama, "A Realization of an Arbitrary BPC Permutation in Hypercube Connected Computer Networks," *IEICE Trans. Inf. Syst. (Japan),* vol. E78-D, no. 4, pp. 428-435, Apr. 1995.
[19] R.J. McEliece, *Finite Fields for Computer Scientists and Engineers.* Kluwer Academic, 1987.

[20] D. Nassimi and S. Sahni, "An Optimal Routing Algorithm for Mesh-Connected Parallel Computers," *J. ACM,* vol. 27, no. 1, pp. 6-29, Jan. 1980.
[21] D. Nassimi and S. Sahni, "Optimal BPC Permutations on a Cube Connected SIMD Computer," *IEEE Trans. Computers,* vol. 31, no. 4, pp. 338-341, Apr. 1982.
[22] L.M. Ni and L.M. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *IEEE Computer,* vol. 26, pp. 62-76, Feb. 1993.
[23] Y. Saad and M.H. Schultz, "Topological Properties of Hypercubes," *IEEE Trans. Computers,* vol. 37, no. 7, pp. 867-872, July 1988.

Yomin Hou received the BS degree in computer science from National Chiao-Tung University, Taiwan in 1989, where he is currently pursuing a doctoral program in the Department of Computer and Information Science. His research interests include interconnection networks and parallel processing.

Chien-Min Wang received the BS and PhD degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, in 1987 and 1991, respectively. He joined the Institute of Information Science, Academia Sinica, Taipei, Taiwan, as an assistant research fellow and is now an associate research fellow. His research interests include parallel compilers, parallel algorithms, parallel computer architectures, and object-oriented technology. He is a member of the IEEE Computer Society.

Chiu-Yu Ku received the BS degree in computer science from National Chiao-Tung University, Taiwan in 1989, the MS degree in computer science from National Taiwan University in 1991, and the MS degree in computer and information science from The Ohio State University, in 1998. Currently, he is working as a software engineer at Avanti Technology Inc., Taiwan. His research interests include computer architecture, parallel computing, and compiling techniques.



Lih-Hsin Hsu received the PhD degree from the University of New York at Stony Brook. He is a professor in the Department of Computer and Information Science at National Chiao Tung University. His research interests include graph algorithms, interconnection networks, and VLSI.