Efficient Realization of Frequently Used Bijections on Cube-Connected Cycles

Liu Zhiyong (刘志勇), Liu Qun (刘 群) and Zhang Xiang (张 祥)

Institute of Computing Technology, The Chinese Academy of Sciences, Beijing 100080

Received December 12, 1993.

Abstract

CCC has lower hardware complexity than hypercube and is suited for current VLSI technology. LC-permutations are a large set of important permutations frequently used in various parallel computations. Existing routing algorithms for CCC cannot realize LC-permutations without network conflict. We present an algorithm to realize LC-permutations on CCC. The algorithm consists of two periods of inter-cycle transmissions and one period of inner-cycle transmissions. In the inter-cycle transmissions the dimensional links of CCC are used in a "pipeline" manner and in the inner-cycle transmissions the data packets are sorted by a part of its destination address. The algorithm is fast ($O(\log_2 N)$) and no conflict will occur.

Keywords: Hypercube, cube-connected cycles, linear complement permutation, routing algorithm, conflict, complexity.

1 Introduction

Hypercube is a promising topology for parallel and distributed processing systems. In a hypercube of size $N = 2^n$, n links are connected to each node. The hardware complexity of hypercube increases quickly as n increases. CCC (Cube-Connected Cycles) is a feasible substitute of hypercube, in which constant(3) links are connected to each node[10]. A set of algorithms (Descend and Ascend) on hypercube can be simulated on CCC without significant degradation of performance[10].

Permutation is the communication pattern in a multiprocessor system by which each processor communicates with one and only one processor[4,5]. The traditional hypercube routing algorithm is the naive routing algorithm, which uses the n dimensions of hypercube in a low to high or high to low order, and the n bits of the destination address are compared with the n bits of the current node address in order to determine if the message should be sent or just remain in the same node. Naive routing algorithm can be easily simulated on CCC. However, permutations which can be passed by the naive algorithm on a hypercube or CCC without conflict are very limited. In such a permutation the destination addresses on each subcube must constitute a complete residue system (CRS)[4,5]. LC-permutations are a large

This research is supported by China National "863" High Technology Project.

set of permutations frequently used in various parallel computation tasks in image processing, pattern recognition, numerical analysis, signal processing, and other scientific and engineering computations, and they include BPC-permutations as a subset[3,5–7,11]. As an instance, the data alignment requirements in the non-linear parallel storage schemes, known as XOR-schemes, are LC-permutations[3,6,11]. Unfortunately, although CCC is less complex in hardware than hypercube, existing routing algorithms (Descend or Ascend) for CCC cannot realize such a large set of LC-permutations without conflict.

If no conflict occurs in the transmission process, the communication will be more efficient in terms of both message delay time and hardware utilization. Various algorithms have been proposed for LC (as well as BPC) permutations on hypercubes and multistage interconnection networks (MINs)[1,2,5,7–9,12–14]. In this paper, we give a conflict-free routing algorithm for LC-permutations on CCC, based on a corresponding routing algorithm on hypercube[5], with the same time complexity. This is the first conflict-free routing algorithm for such a large set of permutations (LC-permutations) on CCC.

2 Some Definitions

Consider a multiprocessor system with $N = 2^n$ nodes. Each node M has an address $m_{n-1}m_{n-2}\cdots m_1m_0$.

In a hypercube, there is a link between node A and node B if and only if the addresses of A and B differ in exactly one bit position k $(0 \le k < n)$. We call this link the k-th dimensional link. And we call the collection of all the k-th dimensional links the sheaf k.

In CCC, the address of node M is divided into two parts:

$$M = (M_u, M_y), \quad M_u = m_{n-1}\cdots m_{n-u}, \quad M_y = m_{y-1}\cdots m_0,$$

where $u + y = n$, and y is the smallest integer for which $y + 2^y \ge n$. We call the u bits M_u the cube-address of node M, while the y bits M_y the cycle-address. Each node M of CCC has three ports: F, B, and L (Forward, Backward, and Lateral). F is connected to node $(M_u, (M_y + 1)\bmod 2^y)$, B is connected to node $(M_u, (M_y - 1)\bmod 2^y)$ and L is connected to node $(M_u \oplus 2^{M_y}, M_y)$. The last link does not exist when $M_y \ge u$. All the nodes with the same M_u are linked as a cycle by F-B links, and all the cycles are linked as a u-cube by the L-L links.

In a Linear-Complement (LC) permutation, the source address $S = s_{n-1}\cdots s_0$ and the destination address $D = d_{n-1}\cdots d_0$ can be expressed as follows:

$$D^\tau = T \times S^\tau \oplus C^\tau$$

where T is a nonsingular $n \times n$ binary matrix:

$$T = \begin{pmatrix} t_{n-1,n-1} & t_{n-1,n-2} & \cdots & t_{n-1,0} \\ t_{n-2,n-1} & t_{n-2,n-2} & \cdots & t_{n-2,0} \\ \cdots & \cdots & \cdots & \cdots \\ t_{0,n-1} & t_{0,n-2} & \cdots & t_{0,0} \end{pmatrix}$$

and C is a binary vector. Thus any bit in the destination address can be expressed as:

$$d_i = \sum_{j=0}^{n-1} t_{i,j} \times s_j \oplus c_i, \quad \text{for } 0 \leq i < n$$

where the operations $\times$ and $\oplus$ are the logic operations AND and XOR (exclusive-or).

3 The Conflict-Free Routing Algorithm for LC-Permutations on CCC

3.1 Simulate Hypercube Algorithm on CCC

Existing algorithms for hypercube which can be simulated on CCC are those in the algorithm class of DESCEND or ASCEND[10].

Algorithm DESCEND

```
/* this algorithm is for each node $M = m_{n-1}, m_{n-2}, \cdots, m_0$ of the system */
BEGIN
  FOR $k := n-1$ DOWNTO 0
  DO
    compute and communicate with node $M \oplus 2^k$;
  ENDDO
END.
```

The ASCEND algorithm is similar to DESCEND, but the loop control is changed to:

FOR $k := 0$ TO $n-1$

Naive routing algorithm is in the class of DESCEND (or ASCEND). In each time unit, the k-th bit of the destination address of the data packet is compared with the k-th bit of the address of the current node, and then the algorithm decides whether the data packet is transmitted along the k-th dimensional link. Unfortunately, the LC-permutations cannot be implemented simply by just using DESCEND or ASCEND without network conflict.

Theorem 1. *The LC-permutation cannot be implemented by an algorithm in the class of DESCEND without conflict.*

Proof. Suppose the LC-permutation can be implemented by an algorithm in the class of DESCEND without conflict.

Consider a data packet P. Assume $A(P,t)$ is the address of packet P at time unit t, then we have:

$A(P,0) = S = s_{n-1}s_{n-2}\cdots s_0, \quad A(P,n) = D = d_{n-1}d_{n-2}\cdots d_0.$

From the algorithm DESCEND, we know that at each time unit t, the data packet can only be transmitted along the k-th dimensional link, while $k = n-t$. So $A(P,t)$ only differs from $A(P,t-1)$ in bit position k. Therefore we can know:

$$A(P,1) = d_{n-1}s_{n-2}\cdots s_0.$$

If there is no conflict in time unit $t = 1$, we have:

$$\forall A(P, 0) \neq A(P', 0), A(P, 1) \neq A(P', 1).$$

That is $\forall (s_{n-1}s_{n-2}\cdots s_0) \neq (s'_{n-1}s'_{n-2}\cdots s'_0), (d_{n-1}s_{n-2}\cdots s_0) \neq (d'_{n-1}s'_{n-2}\cdots s'_0)$.
If: $s_i = s'_i$, for $0 \leq i \leq n-2$, then: $\forall s_{n-1} \neq s'_{n-1}, d_{n-1} \neq d'_{n-1}$.
Because we assume that: $d_{n-1} = \sum_{j=0}^{n-1} t_{n-1,j} \times s_j \oplus c_{n-1}$,
so we have: $(d_{n-1} \oplus d'_{n-1}) = \sum_{j=0}^{n-1} t_{n-1,j} \times (s_j \oplus s'_j)$
Because: $s_i = s'_i$, for $0 \leq i \leq n-2$,
so we have: $(d_{n-1} \oplus d'_{n-1}) = t_{n-1,n-1} \times (s_{n-1} \oplus s'_{n-1})$.
Because: $\forall s_{n-1} \oplus s'_{n-1} \neq 0, d_{n-1} \oplus d'_{n-1} \neq 0$, so: $t_{n-1,n-1} \neq 0$.

Because, for arbitrary LC-permutation, we can say the transformation matrix T is nonsingular, but we can **not** say $t_{n-1,n-1} \neq 0$, so when $t_{n-1,n-1} = 0$, conflicts will occur at time unit $t = 1$. □

For example, let us consider an 8×8 LC-permutation whose transformation matrix is:

$$T = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 1 \\ 1 & 0 & 0 \end{pmatrix}.$$

The destination addresses for nodes 000, 001, 010, 011, 100, 101, 110, 111 are: 000, 110, 010, 100, 001, 111, 011, 101 respectively. At time unit $t = 1$, these data packet will be transmitted to nodes: 000, 101, 010, 111, 000, 101, 010, 111. Thus conflicts occur.

3.2 Conflict-Free Routing Algorithm for LC-Permutations on Hypercubes

An algorithm has been proposed for LC-permutations on hypercubes[5]. The routing includes two stages: First, an algorithm called CRS-rearranging is used, and it takes at most $n-1$ steps; Second, naive routing algorithm is used, and it takes at most n steps.

This algorithm needs precomputations. The precomputation needs to be executed once for a set of permutations which have a common transformation matrix, and generates two vectors called *jump*$[0 : n-1]$ and *buddy*$[0 : n-1]$. The two vectors will be used in the CRS-rearranging and should be loaded into two shift registers in all the nodes before routing. One may refer to [5,9] for the algorithm to produce the vectors *jump* and *buddy*, which is carried out off- line. We also include the algorithm producing the vectors *jump* and *buddy* in Appendix in this paper for self-containing. Note also that for BPC-permutations, which are a subset of LC-permutations, a simpler method can be used to decide the two vectors, since BPC-permutations have simpler (and more regular) transformation matrix. One may refer to [7] for this.

We present the algorithm in the following:

```
Algorithm LC1 /* routing LC-permutation on hypercubes */
  /* This algorithm is for each node $M = m_{n-1}, m_{n-2}, \cdots, m_0$ of the system */
```

```
BEGIN
  /* The following is CRS-rearranging */
  FOR $k := n - 1$ DOWNTO 1
  DO
    IF $jump[k] = 0$ /* Transmitting only along the necessary dimensions */
    THEN
      IF $d_k \oplus d_{buddy[k]} = 1$
      THEN
        Send data packet to and receive data packet from node $M \oplus 2^k$;
      ENDIF;
    ENDIF;
  ENDDO;
  /* The following is naive routing */
  FOR $l := 0$ TO $n - 1$
  DO
    IF $d_l \neq m_l$
    THEN
      Send data packet to and receive data packet from node $M \oplus 2^l$;
    ENDIF
  ENDDO
END.
```

In the above algorithm, $D = d_{n-1} \cdots d_0$ is the destination address of the data packet on the current node M. The algorithm is conflict-free and takes time $O(n)$[5].

3.3 Conflict-Free Routing Algorithm for LC-Permutations on CCC

Although Algorithm LC1 is not in the class of DESCEND or ASCEND, we can see it is similar in terms of the order of the dimensions used in each routing step. They differ essentially in that different bits are used as the routing bit. Here we develop an algorithm for conflict-free routing on CCC, which is an adaptation of LC1. In this algorithm, certain auxiliary transmission steps are carried out in the cycles to arrange the message packets in each cycle so that conflict will not occur either inside any cycle or between cycles.

```
Algorithm LC2 /* routing LC-permutation on CCC */
/* This algorithm is for each node $M = m_{n-1}, m_{n-2}, \cdots, m_0$ of the system */
BEGIN
  /* The following is the first period — inter-cycle transmission */
  FOR $t := 0$ TO $2^{y+1} - 1$
  DO
    IF $(M_y < u) \wedge (2^y \leq t + M_y < 2^{y+1})$
    THEN
      IF $jump[M_y + y] = 0$
      THEN
        IF $d_{M_y+y} \oplus d_{buddy[M_y+y]} = 1$
        THEN
          send data packet to and receive data packet from node $(M_u \oplus 2^{M_y}, M_y)$;
          /* Exchange data packets with the Lateral neighbor */
```

```
            ENDIF;
          ENDIF;
        ENDIF;
        send data packet to node $(M_u, (M_y - 1)\text{mod}2^y)$ and
        receive data packet from node $(M_u, (M_y + 1)\text{mod}2^y)$;
        /* Shift around the cycles */
      ENDDO;
  /* The following is the second period — inner-cycle transmission */
  FOR $p := 0$ TO $2^{y-1} - 1$
  DO
    IF $M_y\text{mod}2 = 1$
      IF $M_y \neq 2^y - 1$
      THEN
        send $D_y$ to and receive $D'_y$ from node $(M_u, M_y + 1)$;
        /* Get destination cycle address from the Forward neighbor */
        IF $D_y > D'_y$
        THEN
          send data packet to and receive data packet from node $(M_u, M_y + 1)$;
          /* Exchange data packets with the Forward neighbor */
        ENDIF
      ENDIF
    ENDIF
    IF $M_y\text{mod}2 = 0$
    THEN
      IF $M_y \neq 0$
      THEN
        send $D_y$ to and receive $D'_y$ from node $(M_u, M_y - 1)$;
        /* Get destination cycle address from the Backward neighbor */
        IF $D_y < D'_y$
        THEN
          send data packet to and receive data packet from node $(M_u, M_y - 1)$;
          /* Exchange data packets with the Backward neighbor */
        ENDIF
      ENDIF
    ENDIF
  ENDDO
  /* The following is the third period — also inter-cycle transmission */
  FOR $q := 0$ TO $2^{y+1} - 1$
  DO
    IF $(M_y < u) \wedge (0 \leq q - M_y < 2^y)$
    THEN
      IF $d_{M_y+y} \neq m_{M_y+y}$
      THEN
        send data packet to and receive data packet from node $(M_u \oplus 2^{M_y}, M_y)$;
        /* Exchange data packets with the Lateral neighbor */
      ENDIF
        send data packet to node $(M_u, (M_y + 1)\text{mod}2^y)$ and
        receive data packet from node $(M_u, (M_y - 1)\text{mod}2^y)$;
      /* Shift around the cycles */
    ENDIF
  ENDDO
END.
```

In the above algorithm, the 2^y groups of data (2^u messages in each group) use the u cube dimensions in a "pipelined" manner. After a certain initiative period (2^y steps), all the u dimensions are fully used concurrently for inter-cycle transmission. It is obvious that the above algorithm takes $2^{y+1} + 2^{y-1} + 2^{y+1} = O(2^y) = O(n)$ steps.

4 Validity

In order to prove that Algorithm LC2 is valid, we need to prove that the packets do arrive at their destinations correctly after the transmission process in LC2, and that no conflict will occur in any transmission step (either inside each cycle, or between cycles). For ease of understanding, we present the proof in a way that the "intermediate" addresses of the transmission of each period in LC2 are compared with those in LC1, and show that this will result in correct transmission of the message packets.

Suppose a data packet P whose source address is S and destination address is D.

Algorithm LC2 consists of three loops, and the control statements are:

```
FOR $t := 0$ TO $2^{y+1} - 1$,
FOR $p := 0$ TO $2^{y-1} - 1$, and
FOR $q := 0$ TO $2^{y+1} - 1$.
```

Algorithm LC1 consists of two loops, and the control statements are:

```
FOR $k := n - 1$ DOWNTO 1, and
FOR $l := 0$ TO $n - 1$.
```

Corresponding to Algorithm LC2, the whole process of Algorithm LC1 can be split into three periods:

```
First:   FOR $k := n - 1$ DOWNTO $y$
Second:  FOR $k := y - 1$ DOWNTO 1
         FOR $l := 0$ TO $y - 1$
Third:   FOR $l := y$ TO $n - 1$
```

Let's use functions $A(k = x), A(l = x), A(t = x), A(p = x), A(q = x)$ to record the addresses of the data packet P in each loop of the algorithms, and use $A(k = 0)$,$A(l = n)$,$A(t = 2^{y+1})$,$A(p = 2^{y-1})$,$A(q = 2^{y+1})$ to record the addresses of the data packet P after each loop of the algorithms.

Lemma 1. *In the first period of Algorithm LC2 the transmissions along the lateral links happen only when: $S_y + 2^y - u + 1 \le t \le S_y + 2^y$, and:*

$$\begin{cases} A_y(t = S_y + 2^y - u + x) = n - x - y = k - y \\ A_u(t = S_y + 2^y - u + x) = A_u(k = n - x) \end{cases}, \forall x : 1 \le x \le u + 1.$$

Theorem 2. *Each data packet will be transmitted to its destination after Algorithm LC2.*

Because the formal proofs of the above lemma and theorem are very long, we just give them in the appendix. Here, for ease of understanding of the proof as well as the algorithm itself, we explain briefly the effect of each period of the routing procedure.

- Firstly, it should be noticed that a permutation of size of $2^n = 2^u \times 2^y$ is not necessarily 2^y permutations of size of 2^u. The purpose of the first and the second periods in the algorithm is to rearrange the 2^n packets into 2^y permutations of size of 2^u. The transmission along the lateral links in the first period **is not to match** the high order u bits step by step as in the traditional DESCEND routing process.
- Secondly, the shift around in each cycle in the first period is to rearrange the 2^y packets in each cycle simultaneously so that the packets will be aligned with the dimensions on which they need to be transmitted to their lateral neighbors.
- Thirdly, the purpose of the second period is to sort the packets in each cycle so that the 2^y packets on the 2^u cycles will constitute 2^y permutations passable without network conflict. With the above two periods being completed, the third period of the algorithm is simply a series of naive routing of the 2^y permutations (with some shift arounds being carried out in each cycle to align the packets with the correct dimensions as those in the first period).

Theorem 3. *No conflict will happen in Algorithm LC2.*

Proof. It is easy to know that no conflict will happen in the second period of Algorithm LC2. And the proof of the third period is similar to that of the first period, so here we just prove that no conflict will happen in the first period.

In the first period, no conflict will happen in the process of shift around the cycles, so we just need to consider the transmission along the lateral links. Consider a lateral link which connects the nodes (M_u, M_y) and (M'_u, M_y). We know $M'_u = M_u \oplus 2^{M_y}$.

Suppose the data packets in the two nodes are P and P', then we have:

$A_y(P,t) = A_y(P',t) = A_y$

$A_u(P,t) = A_u(P',t) \oplus 2^{A_y}$

From Lemma 1, we know the transmissions along the lateral links happen when $S_y + 2^y - u + 1 \le t \le S_y + 2^y$, and we have:

$A_y = n-x-y = k-y$, $A_u(t = S_y+2^y-u+x) = A_u(k = n-x)$, $\forall x : 1 \le x \le u+1$,

so: $A_u(P, k = n - x) = A_u(P', k = n - x) \oplus 2^{A_y}$.

Because: $A_y(P, k = n - x) = A_y(P', k = n - x) = S_y$,

we have: $A(P, k = n - x) = A(P', k = n - x) \oplus 2^{A_y+y}$,

and because: $A_y + y = n - x = k$, we know that at the time $k = n - x$ in Algorithm LC1, these two data packets are at the two sides of a k-th dimensional link of hypercube. The condition of the transmission in Algorithm LC1 is:

$(jump[k] = 0) \wedge ((d_k \oplus d_{buddy[k]}) = 1)$

The condition of the transmission in Algorithm LC2 is:

$(jump[M_y + y] = 0) \wedge ((d_k \oplus d_{buddy[M_y+y]}) = 1)$

We can see the conditions are the same. Because no conflict will happen in Algorithm LC1, the case is the same in Algorithm LC2. □

5 Conclusions

Various algorithms have been proposed for LC-permutations on hypercubes as well as through MINs. But no algorithm has been proposed for LC-permutations on CCC. In this paper we present a routing algorithm for the realization of LC-permutations on CCC. The feature of the algorithm is that, no conflict will occur at each routing step. The routing is completely distributed for inter-cycle transmission, as each node can decide its transmission based only on the destination of its message packet; and, only an exclusive-or operation of two bits is needed in each routing step. Within each cycle, only an arithmetic comparison in each routing step is needed to sort the messages due to the simple interconnection scheme within each cycle. The whole routing process is fast and simple. The algorithm proposed in this paper can realize more permutations frequently used in various computations on CCC without network conflict than existing algorithms for CCC. The routing algorithm can be easily implemented in both hardware and software.

References

[1] Boppana R, Raghavendra C S. On self-routing in Benes and shuffle exchange networks. In *Proc. of 1988 International Conference on Parallel Processing,* Aug. 1988, pp. 196–200.

[2] Boppana R, Raghavendra C S. Optimal self-routing of linear complement permutations in hypercubes. In *Proc. of the 5th Distributed Memory Computing Conference*, April 1990, pp. 800–808.

[3] Kim K, Kumar V K P. Parallel memory systems for image processing. In *Proc. of the 1989 Conference on Computer Vision and Pattern Recognition,* pp. 650–659.

[4] Lee K Y. On the rearrangeability of $2(\log_2 N - 1) - 1$ stage permutation networks. *IEEE Trans. Comput.*, 1985, C-34(5): 412–425.

[5] Liu Zhiyong, Li X. Routing linear complement permutations on hypercubes. Technical Report TR92-01, Dept. of Computing Science, Univ. of Alberta, Edmonton, Alberta, Canada.

[6] Liu Z, Li X, You J. On storage scheme for parallel array access. Supercomputing'92, July 1992, pp. 282–291.

[7] Liu Z, You J. Conflict-free routing for BPC-permutations on synchronous hypercubes. *Parallel Computing,* 1993, 19: 323–342.

[8] Liu Z, You J, Li X. Conflict-free routing on hypercubes. In *Proc. of the International Conference of Computers and Information*, May 1992, pp. 153–158.

[9] Liu Z, Li X. On the implementation of LC-permutations on hypercubes. In *Proc. of International Symposium for Young Investigators*, Feb. 1994.

[10] Franco P Preparata, Jean Vuillemin. The cube-connected cycles: A versatile network for parallel computation. *Communications of the ACM*, 1981, 24(5): 300–309.

[11] Raghavendra C S, Boppana R. On methods for fast and efficient parallel memory access. In *Proc. of the International Conference on Parallel Processing,* Vol. I, pp. 76–83.

[12] Raghavendra C S, Boppana R. On self-routing in Benes and shuffle exchange networks. *IEEE Trans. on Computers,* 1991, 40(9): 1057–1064.

[13] Sengupta A, Zemoudeh K. Self-routing algorithms for strongly-regular multistage interconnection networks. *Journal of Parallel and Distributed Computing,* 1992, 14: pp. 187–192.

[14] Zemoudeh K, Sengupta A. Routing frequently used bijections on hypercube. In *Proc. of the 5th Distributed Memory Computing Conference,* April 1990, pp. 824–832.

Appendices:

A.1 Proof of Lemma 1.

In the first period of Algorithm LC1, we have:

$A_y(t+1) = (A_y(t)-1)\mathrm{mod}2^y$

So we get:

$$A_y(t) = \begin{cases} S_y - t & 0 \le t \le S_y \\ S_y + 2^y - t & S_y + 1 \le t \le S_y + 2^y \\ S_y + 2^{y+1} - t & S_y + 2^y + 1 \le t \le 2^{y+1} \end{cases}$$

That means:

$$A_y(t) + t = \begin{cases} S_y & 0 \le t \le S_y \\ S_y + 2^y & S_y + 1 \le t \le S_y + 2^y \\ S_y + 2^{y+1} & S_y + 2^y + 1 \le t \le 2^{y+1} \end{cases}$$

Because the transmissions through the lateral links happen only when:

$(2^y \le t + M_y < 2^{y+1})$,

and at time unit t, $A_y(t) = M$, that means only when:

$S_y + 1 \le t \le S_y + 2^y$,

inter-cycle transmission can happen, and because the transmissions through the lateral links happen only when: $M_y < u$, that means: $S_y + 2^y - t < u$, so we have:

$S_y + 2^y - u + 1 \le t \le S_y + 2^y$, and

$A_y(t = S_y + 2^y - u + x) = S_y + 2^y - t = u - x = n - x - y,\ \forall x : 1 \le x \le u+1$.

Then let's consider the equation:

$A_u(t = S_y + 2^y - u + x) = A_u(t = 0) = A_u(k = n - x) = S_u,\ \forall x : 1 \le x \le u+1$.

When $x = 1$, $A_u(t = S_y + 2^y - u + x) = A_u(t = 0) = A_u(k = n - x) = S_u$, and we know:

if $(jump[M_y + y] = 0) \wedge ((d_{M_y+y} \oplus d_{buddy[M_y+y]}) = 1)$,

then $A_u(t = S_y + 2^y - u + x + 1) = A_u(t = S_y + 2^y - u + x) \oplus 2^{M_y}$,

else $A_u(t = S_y + 2^y - u + x + 1) = A_u(t = S_y + 2^y - u + x)$.

Because:

$M_y + y = A_y(t) + y = S_y + 2^y - t + y = S_y + 2^y - (S_y + 2^y - u + x) + y = u - x - y = n - x$,

so: if $(jump[n-x] = 0) \wedge ((d_{n-x} \oplus d_{buddy[n-x]}) = 1)$,

then $A_u(t = S_y + 2^y - u + x + 1) = A_u(t = S_y + 2^y - u + x) \oplus 2^{n-x-y}$,

else $A_u(t = S_y + 2^y - u + x + 1) = A_u(t = S_y + 2^y - u + x)$.

From Algorithm LC1, we know:

if $(jump[n-x] = 0) \wedge ((d_{n-x} \oplus d_{buddy[n-x]}) = 1)$,

then $A(k = n - x - 1) = A(k = n - x) \oplus 2^{n-x}$,

else $A(k = n - x - 1) = A(k = n - x)$.

Consider $A_u(k)$ — the left u bits of $A(k)$, that means:

if $(jump[n-x] = 0) \wedge ((d_{n-x} \oplus d_{buddy[n-x]}) = 1)$,

then $A_u(k = n - x - 1) = A_u(k = n - x) \oplus 2^{n-x-y}$,

else $A_u(k = n - x - 1) = A_u(k = n - x)$.

So if the equation is true for x, it is true for $x+1$, thus the equation is true for all $x : 1 \le x \le u$. □

A.2 Proof of Theorem 2.

Because we know:

$A(k = n - 1) = A(t = 0) = S$, and

$A(l = n) = D$,

so we just need to prove: $A(q = 2^{y+1}) = A(l = n) = D$.

We will prove in turn that after each period the address of P is the same in the two algorithms:

$A(t = 2^{y+1}) = A(p = 0) = A(k = y - 1)$,

$A(p = 2^{y-1}) = A(q = 0) = A(l = y)$,

$A(q = 2^{y+1}) = A(l = n - 1) = D$.

It is easy to find out that:

$A_y(k = y - 1) = A_y(k = n - 1) = S_y$,

$A_u(l = y) = A_u(k = y - 1)$,

$A_y(l = y) = A_y(l = n - 1) = D_y$,

$A_u(p = 2^{y-1}) = A_u(p = 0)$.

And in the first and third periods of Algorithm LC2, the shifts around the cycles repeat for 2^{y+1} times, exactly twice of the length of the cycle. So we have:

$A_y(t = 2^{y+1}) = A_y(t = 0) = S_y$,

$A_y(q = 2^{y+1}) = A_y(q = 0)$.

So we just need to prove in turn that:

$A_u(t = 2^{y+1}) = A_u(k = y - 1) = S_y$,

$A_y(p = 2^{y-1}) = A_y(l = y)$,

$A_u(q = 2^{y+1}) = A_u(l = n - 1)$.

Let's consider the first period in Algorithm LC2.

From Lemma 1, we know the lateral links happen only when: $S_y + 2^y - u + 1 \leq t \leq S_y + 2^y$, so:

$A_u(t = 2^{y+1}) = A_u(t = S_y + 2^y + 1)$

When $x = u + 1$, we have:

$A_u(t = S_y + 2^y + 1) = A_u(k = n - u - 1) = A_u(k = y - 1)$,

and the proof of the first period is done.

Then let's consider the second period.

At the end of the second period of Algorithm LC1, the cycle address of each data packet is the same as its destination cycle address:

$$A_y(l = y) = A_y(l = n - 1) = D_y,$$

In the second period of both algorithms, the transmissions of the data packets always happen between the nodes with the same M_u, just within the cycles of the CCC. And at the beginning of the second period, the addresses of the data packets in both algorithms are the same. Thus at the beginning of the second period the cycle addresses (D_y) of the data packets in each cycle of CCC are a permutation of the integers of 0 to $2^y - 1$. One may also cite Theorem 2 in [14] to know that the D_ys in each cycle really constitute a CRS mod 2^y.

The second period of Algorithm LC2 is just a sorting process of the 2^y data packets on each cycle. So we know the cycle address of each data packet at the end of the second period in Algorithm LC2 is exactly its destination cycle address, just the same as that in Algorithm LC1. And the proof of the second period is done.

The proof of the third period is similar to that of the first period. We omit the proof here. □

A.3 The Algorithm for Finding the Two Vectors for a Type of Permutations

```
Algorithm Buddy-Jump
 /* To decide if a dimension can be jumped over, or which bit(s) should be used
    for routing */
 BEGIN
   FOR $k := n - 1$ DOWNTO 1
   DO
     IF $T_{(k)}$ is singular
     THEN
       $jump[k] := 0$;
       IF $t_{k,k} = 1$
       THEN
         Look for an $i$ such that $i < k$ and $t_{i,k} = 1$;
         $buddy[k] := i$;
         $r_{k,k} := \bar{r}_{k,k}$;
         $r_{k,i} := \bar{r}_{k,i}$;
       ELSE
         $buddy[k] := n$;
         $r_{k,k} := \bar{r}_{k,k}$;
       ENDIF
       Modify $T$ according to $R$;
     ELSE
       $jump[k] := 1$;
     ENDIF
   ENDDO
 END.
```

In the algorithm, T is the transmission matrix of the LC-permutation, R is the inverse matrix of T, $T_{(k)}$ is the $k \times k$ submatrix at the bottom-right of T. The precomputation needs to be carried out once for a set of LC-permutations with the common transmission matrix and takes time $O(n^3)$ in the worst case.

Liu Zhiyong received his M.S. degree from Northwest Telecommunication Institute and Ph.D degree from the Institute of Computing Technology, The Chineses Academy of Sciences in 1983 and 1987, respectively. He worked as a visiting scholar and a postdoctor fellow in U.S.A. and Canada from 1988 through 1992. He is currently a Professor in the Institute of Computing Technology, The Chinese Academy of Sciences. His research interests include parallel algorithms and architectures, interconnection networks, and artificial intelligence.

Liu Qun received his M.S. degree from the Institute of Computing Technology, The Chinese Academy of Sciences in 1992. He is currently an Assistant Professor in the Institute of Computing Technology, The Chinese Academy of Sciences. His research interests include parallel processing and artificial intelligence.

Zhang Xiang is a Professor in the Institute of Computing Technology, The Chinese Academy of Sciences. He is the director of the 2nd department of the Institute. His research interests include multimedia technology and systems, and parallel processing.