Joint Conference of 1996 International Computer Symposium
December 19~21, Kaohsiung, Taiwan, R.O.C.

CONFIGURING GROUP-MULTICAST RINGS WITH RANDOMLY SELECTED NODES ON BINARY N-CUBES[1]

W. Lin and E. W. Chen

Institute of Computer Science
National Chung-hsing University
250 Kuo-kuan Road
Taichung, Taiwan ROC
email: echen@cs.nchu.edu.tw

ABSTRACT

In this paper we present a systematic method of configuring group-multicast rings with randomly selected nodes on binary n-cube multiprocessors. The method configures rings only with disjoint links so that the selected nodes can perform multicast communication without messages collision. Binary n-cube multiprocessors are typically one-port architectures, which allow a processor node to send or receive only one message at a time. Being capable of pipelining messages throughout nodes, rings offer a practical solution to perform group multicast operations on one-port architectures. The basic idea behind our proposed method is to repeatedly merge small rings to form larger rings. Once a multicast ring is configured, messages can circulate around the ring without incurring link contentions. We begin with introducing a basic network model which specifies path routing on n-cubes. Then we present some preliminary results regarding properties of disjoint paths. The concept of the proposed method is described. Finally, the method is formally presented and its correctness is verified. The proposed method presents two unique features. First, it guarantees that the physical distance between adjacent nodes is limited to n links on an n-cube. Secondly, it uses only a single dimension order to route messages across the links towards their respective destinations.

Keywords: Computer networks, Data communication, Interconnection networks

1. INTRODUCTION

In many parallel applications, group multicast is an important communication operation for processor nodes to exchange data messages [1-7]. Processor nodes pertaining essential data are selected to form a group. Multicast is carried out simultaneously by the nodes in the group. Each of them sends the same data message to every other node, while different nodes may disseminate different messages. Most of the binary n-cube multiprocessors are in fact one-port architectures in which a processor node transmits or receives only one message at a time. Ring provides a practical solution to implementing such multicast operations on the one-port architecture. A ring for group multicast works like this. Selected nodes are first ordered and interconnected to form a ring. Messages on the ring will move uniformly in the same direction. As a group multicast operation starts, each node sends its data message to the downstream neighbor. From then on, a node successively receives messages from its upstream neighbor, makes copies of messages, and passes them on to its downstream neighbor. With a ring, nodes can transmit and relay messages in a pipelined fashion. Transmission of messages are so highly overlapped that when messages complete the trip, each node receives all the messages disseminated from the other nodes of the group.

In this paper, we are concerned with configuring multicast rings with randomly selected nodes on binary n-cube multiprocessors. It is well known that the binary reflected Gray code, RGC for short, specifies a ring across an n-cube [6]. Neighboring nodes in the ring are physically connected by direct links of an n-cube. It is a perfect structure for interconnecting all the nodes of an n-cube. But for group multicast over randomly selected nodes, the RGC ring has two apparent drawbacks. First, it may cause excessive transmission latency for messages that circulate around the nodes. For example, consider the 8-node ring specified by the RGC for 3-cube. The nodes are arranged in order of 0 1 3 2 6 7 5 4. Suppose that nodes 1, 4 and 6 are selected for group multicast. In this case, messages from node 1 to node 6 have to pass through nodes 2 and 3. This results in unwanted latency. The problem with the RGC ring is: when nodes are randomly selected for group multicast, there is no control over the physical distance between two adjacent nodes selected in the ring. In terms of link count, the worst-case distance is $O(2^n)$ for an n-cube. The latency continues to grow, as the dimension of n-cube increases. The second drawback comes from the difficulty in routing messages. We need to employ different dimension orders to move messages among selected nodes. To explain this, we give a follow-up example of the previous one. For convenience, we use binary addresses to in routing messages. We need to employ different dimension orders to move messages among selected nodes. To explain this, we give a follow-up example of the previous one. For convenience, we use binary addresses to label these nodes and assume the least significant bit to stand for the lowest dimension. A message from node 1 to node 6 needs to cross over three links: 001→011, 011→010, and 010→110, which links correspond to dimensions 1, 0 and 2, respectively. A message from node 6 to node 4 needs to cross over three links: 110→111, 111→101, and 101→100. They correspond to dimensions 0, 1 and 0, respectively. It is obvious that the two dimension orders are different. This particular example reflects the complication of routing messages on the RGC ring: there is no single dimension order in which messages can move among the selected nodes.

Our goal is to find a systematic method of configuring a ring for randomly selected nodes, without the two aforementioned problems. The proposed method allows us to form a ring with disjoint links. It guarantees that the physical distance between adjacent nodes is limited to n links, in contrast to $O(2^n)$ for the RGC ring. Moreover, the proposed method uses a single dimension order to route messages across the links towards their destinations.

The subsequent sections are organized in the following manner. In the next section, we present a basic network model which specifies path routing on n-cubes. We also define some operations for formal descriptions and provide verification of the proposed method. In Section 3, we introduce a number of preliminary results regarding properties of disjoint paths. In Section 4, we formally present the proposed method, with its concept described at the beginning and its correctness verified at the end. Finally, we conclude our discussion in Section 5.

2. NETWORK MODEL AND SOME BASICS

An n-cube consists of 2^n nodes. Labeled with n-bit addresses, two nodes have a direct link if and only if their addresses differ exactly in one bit position. We assume that the links of an n-cube are full-duplex. Therefore, two messages in the opposite directions can be transmitted simultaneously over a link. To deliver a message, we need to create a path from the source node to its destination. We assume that wormhole routing is used to move messages across the network [8-10]. A message is divided into small units called flits. Upon receiving a flit, an intermediate node immediately forwards it to the next node without waiting for the entire message to arrive. ***In this paper, we employ a deterministic routing scheme to determine the path; it resolves a routing address in the descending order of dimensions.*** For example, consider routing a message from node 011 to node 100 on a 3-cube. The three nodes traversed by the path are in the order of 111, 101, and finally 100.

Wormhole routing is attractive in its pipelining nature; when a path is clear, following the header flit, the succeeding flits fall through intermediate nodes without buffering. This greatly reduces transmission latency of messages. But this attractive feature diminishes immediately if two paths compete to cross over a link in the same direction. This results in link contention. In wormhole routing, if a header flit comes across a link already in use, the header flit as well as the succeeding flits are buffered at the starting node of the link. When the link becomes available, the flits are removed from the buffer and proceed to the next stop. Link contention will significantly increase transmission latency of messages. This problem becomes more severe in the case of group multicast, because each path is responsible for transferring many messages disseminated from various sources. In case that two paths traverse a common link, they will continuously contend for the link throughout the multicast operation, causing excessive transmission delay. To accomplish efficient group multicast, we must avoid creating paths that compete for links. The proposed method is based on this principle. It creates paths with disjoint links to transfer messages, completely eliminating the possibility of link contention. In the following discussion, we present a number of definitions and operations for manipulating addresses of n-cube nodes. They are useful in providing insight into creating link-disjoint paths.

DEFINITION 1 For two n-bit numbers, $X=x_{n-1}x_{n-2}...x_0$ and $Y=y_{n-1}y_{n-2}...y_0$, $X{\downarrow}Y$ yields a sequence $z_{n-1}z_{n-2}...z_0$,

$$z_i = \begin{cases} 0 & \text{if } x_i=y_i=0 \text{ and } x_k=y_k \text{ for all k, i-1}\geq k\geq 0; \\ 1 & \text{if } x_i=y_i=1 \text{ and } x_k=y_k \text{ for all k, i-1}\geq k\geq 0; \\ * & \text{if } x_k\neq y_k \text{ for some k, i}\geq k\geq 0. \end{cases}$$

The operation $\downarrow$ identifies the longest suffix of two addresses and patches the remaining positions with *'s. For example, suppose that X=101110 and Y=000110. Then X↓Y= ***110. Similarly, we define a binary operator for identifying the longest prefix of two addresses.

DEFINITION 2 For two n-bit numbers, $X=x_{n-1}x_{n-2}...x_0$ and $Y=y_{n-1}y_{n-2}...y_0$, $X{\uparrow}Y$ yields a sequence $z_{n-1}z_{n-2}...z_0$,

$$z_i = \begin{cases} 0 & \text{if } x_i=y_i=0 \text{ and } x_k=y_k \text{ for all k, n-1}\geq k\geq \text{i+1}; \\ 1 & \text{if } x_i=y_i=1 \text{ and } x_k=y_k \text{ for all k, n-1}\geq k\geq \text{i+1}; \\ * & \text{if } x_k\neq y_k\text{, for some k, n-1}\geq k\geq \text{i}. \end{cases}$$

For example, suppose that X=110101 and Y=110110. Then X↑Y=1101**. We introduce an operator for indicating differences of two sequences of symbols {0, 1, *}.

DEFINITION 3 ⊗ is a bitwise operator over the domain {0, 1, *} with the arithmetic

⊗	0	1	*
0	0	1	*
1	1	0	*
*	*	*	*

For example, let U=*010**01 and V=*10011*1. U⊗V=*110***0. Note that ⊗ operates on sequences of symbols {0, 1, *}, while ↓ and ↑ operate on n-bit addresses. In fact, ⊗ is a composite operator that works as exclusive-OR when two symbols in the corresponding positions are bits, and it works as a masking operator when one of the two symbols is an asterisk. We further define two functions that count specific symbols in the sequences of {0, 1, *}.

DEFINITION 4 Given a sequence $z_{n-1}z_{n-2}...z_0$ of {0, 1, *}, *One*($z_{n-1}z_{n-2}...z_0$) counts 1's in $z_{n-1}z_{n-2}...z_0$; *Asterisk*($z_{n-1}z_{n-2}...z_0$) counts *'s in $z_{n-1}z_{n-2}...z_0$.

Consider, for instance, a sequence Z=01**010*. *One*(Z)=2 and *Asterisk*(Z)=3. Finally, we define a partial order for comparing two sequences of symbols {0, 1, *}.

DEFINITION 5 Let U= $u_{n-1}u_{n-2}...u_0$ and V= $v_{n-1}v_{n-2}...v_0$ be two equal-length sequences of symbols {0, 1, *}. We say U<•V if for every i, n-1≥i≥0,

(1) $u_i = v_i$

or (2) u_i= * and $v_i \neq$ *.

For example, given U=*010**1 and V=1010*01, we say U<•V.

3. PRELIMINARIES

With all the needed definitions made in the previous section, we now come to exploring conditions under which paths on an n-cube become link-disjoint, and thereby contention-free. For convenience, from now on, link-disjoint paths are simply referred to as disjoint paths. We will first identify a number of properties exhibited by disjoint paths. Later these properties will be used to create disjoint paths with certain patterns. Of particular interest and importance is the following theorem. The importance of the theorem is twofold. First, it serves as a simple test to see if a set of paths is disjoint. Secondly, it characterizes a fundamental property possessed by all the paths which are disjoint. And we can use it as a guide to avoid link contentions, when creating paths for the multicast ring.

THEOREM 1 In an n-cube, *Path*(A,B) and *Path*(C,D) are disjoint, if and only if

$$One((A{\downarrow}C)\otimes(B{\uparrow}D))=0.$$

Proof: Let A=$a_{n-1}a_{n-2}...a_0$, B=$b_{n-1}b_{n-2}...b_0$, C=$c_{n-1}c_{n-2}...c_0$, and D=$d_{n-1}d_{n-2}...d_0$. Under the descending dimension order, *Path*(A,B) traverses intermediate nodes: $b_{n-1}a_{n-2}a_{n-2}...a_0$, $b_{n-1}b_{n-2}a_{n-3}...a_0$, $b_{n-1}b_{n-2}b_{n-3}...a_0$, etc. Similarly, *Path*(C,D) traverses intermediate nodes: $d_{n-1}c_{n-2}\ c_{n-2}...c_0$, $d_{n-1}d_{n-2}c_{n-3}...c_0$, $d_{n-1}d_{n-2}d_{n-3}...c_0$, etc. The necessary and sufficient condition in which the two paths have a link conflict is that they both arrive at some intermediate node and leave for the same node in the next step. Suppose that a link conflict occurs in the i^{th} dimension. The condition stated above can be described by the three simultaneous requirements:

(1) $b_{n-1}...b_{i+1}a_i...a_0= d_{n-1}...d_{i+1}c_i...c_0$,
(2) $b_i= d_i$, and
(3) $a_i \neq b_i$.

Combining the three requirements, we should have

$$One((A{\downarrow}C)\otimes(B{\uparrow}D))\neq 0.$$

On the other hand, if not all the three requirements can be met simultaneously for every dimension i, 0≤i≤n-1, then *One*((A↓C)⊗(B↑D))=0, and vice versa. This assures that *Path*(A,B) and *Path*(C,D) do not have link conflicts if and only if *One*((A↓C)⊗(B↑D))=0. ◆

Let us look an example of contention-free paths. Consider routing the following two paths on a 6-cube: *Path*(101011, 001010) and *Path*(011011, 001101). We first take

(101011↓011011)=**1011
and (001010↑001101)=001***.

Combining them, we obtain

$$One(**1011\otimes 001***) = One(**0***) = 0.$$

We can say that the two paths are disjoint. To verify our claim, we show the two paths along with the intermediate nodes they traverse:

Path(101011, 001010):
$1010 11\rightarrow\underline{0}01011\rightarrow 00101\underline{0}$
Path(011011, 001101):
$011011\rightarrow 0\underline{0}1011\rightarrow 001\underline{1}11\rightarrow 0011\underline{0}1$.

Each underscored bit corresponds to a link traversed by a path in the particular dimension. Note that although the two paths cross at node 001011, they do not compete with each other to pass the same link. Therefore, no link contention occurs.

Two conflicting paths, on the other hand, have a different story. If we carry out the test of Theorem 1 on them, link contentions will be reflected in the resulting sequence of the test. To be more specific, we present the following corollary.

COROLLARY 1 If *Path*(A,B) and *Path*(C,D) are two conflicting paths in an n-cube, then the number of links over which they conflict is *One*((A↓C)⊗(B↑D)).

Proof: Let $z_{n-1}z_{n-2}...z_0$ of {0, 1, *} be the resulting sequence of (A↓C)⊗(B↑D). If there exists some z_i=1, it refers to

(1) $b_{n-1}...b_{i+1}a_i...a_0= d_{n-1}...d_{i+1}c_i...c_0$,
(2) $b_i= d_i$, and
(3) $a_i \neq b_i$.

That is, *Path*(A,B) and *Path*(C,D) have a link conflict in the i^{th} dimension. Therefore, the number of 1's in (A↓C)⊗(B↑D) yields the count of link conflicts between *Path*(A,B) and *Path*(C,D). ◆

For example, consider two paths on a 6-cube: *Path*(001011, 100110) and *Path*(111011, 100101). Let us first figure out intermediate nodes traversed by the two paths:

Path(001011, 100110):
$001011\to\underline{1}01011\to10\underline{0}011\to100\underline{1}11\to10011\underline{0}$
Path(111011, 100101):
$111011\to1\underline{0}1011\to10\underline{0}011\to100\underline{1}11\to1001\underline{0}1$.

Note that in the course of routing, the two paths contend to pass two common links; they are $101011\to100011$ and $100011\to100111$. Now let us take the test

$(001011{\downarrow}111011)\otimes(100110{\uparrow}100101)$
$= {**}1011\otimes1001{**}$
$= {**}11{**}$.

The two 1's in the resulting sequence indicate that *Path*(001011, 100110) and *Path*(111011, 100101) are not disjoint and have two link contentions in dimensions 2 and 3, respectively.

In the following corollary, we present another condition under which two paths become disjoint. It is, in fact, a special case of Theorem 1 but of independent interest and importance. It refers to two paths which do not cross each other at any node. Since the two paths each traverse separate nodes, they would never compete for using the same link.

COROLLARY 2 In an n-cube, if *Path*(A,B) and *Path*(C,D) are two paths and if

$$Asterisk(A{\downarrow}C) + Asterisk(B{\uparrow}D) \geq n,$$

then the two paths are disjoint.

Proof: Let $z_{n-1}z_{n-2}\ldots z_0$ of $\{0, 1, *\}$be the resulting sequence of $(A{\downarrow}C)\otimes(B{\uparrow}D)$; namely, $z_{n-1}\ z_{n-2}\ldots z_0=(A{\downarrow}C)\otimes(B{\uparrow}D)$. Since $Asterisk(A{\downarrow}C) + Asterisk(B{\uparrow}D) \geq n$, $z_{n-1}z_{n-2}\ldots z_0$ must be a sequence of n *'s. Hence, $One(z_{n-1}z_{n-2}\ldots z_0)=One((A{\downarrow}C)\otimes(B{\uparrow}D))=0$. Thus, *Path*(A,B) and *Path*(C,D) are disjoint. ◆

So far, our focus is mainly on devising tests for telling whether paths are disjoint. Next, we present a few interesting ways of creating disjoint paths under some given conditions.

LEMMA 1 In an n-cube, if *path*(A,B) and *path*(C,D) are disjoint, then *path*(A,D) and *path*(C,B) are disjoint.

Proof: If *path*(A,B) and *path*(C,D) are disjoint, we have $One((A{\downarrow}C)\otimes(B{\uparrow}D))=0$. Since ↑ is a commutative operator and $One((A{\downarrow}C)\otimes(D{\uparrow}B)) = One((A{\downarrow}C)\otimes(B{\uparrow}D)) = 0$, this indicates that *path*(A,D) and *path*(C,B) are disjoint.
◆

In plain words, this lemma says that if two disjoint paths swap their destinations, then the resulting paths are also disjoint. It can be better understooa 'y giving an example. Consider two paths on a 6-cube: *Path*(100101, 010010) and *Path*(101101,010110). The intermediate nodes traversed by them are shown below:

Path(100101, 010010):
$100101\to000101\to010101\to010001\to010011\to010010$
Path(101101, 010110):
$101101\to001101\to011101\to010101\to010111\to010110$.

Now, if we swap the destination nodes, the two new paths become:

Path(100101,010110):
$100101\to000101\to010101\to010111\to010110$
Path(101101,010010):
$101101\to001101\to011101\to010101\to010001\to010011\to$
010010.

From this example, we can make an interesting observation: The two paths meet at node 010101; after that, *Path*(100001,010110) takes the route which used to be taken by *Path*(101101,010110), and *Path*(101101,010010) takes the route used to be taken by: *Path*(100101, 010010). The effect of swapping destinations on the route interchange can be easily seen in this example. In essence, the two paths simply interchange their routes at node 010101. The new paths traverse exactly the same links traversed by the original paths. Of course, they are also disjoint. But we should mention here that the simple interchange is due to the fact that $Asterisk(100101{\downarrow}101101) + Asterisk(010010{\uparrow}010110) = 6$. Note that the above example is a special case in which the asterisk sum equals the dimension count of the 6-cube. Path interchange may not happen in other cases, especially for two paths which do not cross each other at intermediate nodes. However, the above lemma always assures that the two paths, after swapping their destinations, are also disjoint. Next, let us look at an interesting result useful in creating disjoint paths.

LEMMA 2 If X, Y and Z are n-bit addresses and $X{\downarrow}Y <\!\bullet\, X{\downarrow}Z$, then $X{\downarrow}Y <\!\bullet\, Y{\downarrow}Z$.

Proof: Let $X=x_{n-1}x_{n-2}\ldots x_0$, $Y=y_{n-1}y_{n-2}\ldots y_0$, and $Z=z_{n-1}z_{n-2}\ldots z_0$. Suppose that $X{\downarrow}Y={*}\ldots{*}x_i\ldots x_0$, and $X{\downarrow}Z={*}\ldots{*}x_j\ldots x_0$. Since $X{\downarrow}Y <\!\bullet\, X{\downarrow}Z$, *j* is greater than or equal to *i*. In other words, Y and Z have at least *i* low-order bits in common. That is, if $Y{\downarrow}Z={*}\ldots{*}y_k\ldots y_0$, then *k* is greater than or equal to *i*. Since $X{\downarrow}Y={*}\ldots{*}x_i\ldots x_0$ and $y_i\ldots y_0= x_i\ldots x_0$, we conclude that $X{\downarrow}Y <\!\bullet\, Y{\downarrow}Z$. ◆

For instance, let A=101001, B=010101 and C=100101. We have $A{\downarrow}B= {****}01$, $A{\downarrow}C= {****}01$, and $B{\downarrow}C= {**}0101$. Apparently, $A{\downarrow}B <\!\bullet\, A{\downarrow}C$. And it becomes true that $A{\downarrow}B <\!\bullet\, B{\downarrow}C$. Now let us turn our attention to the following theorem. Making use of the previous lemma, it shows an essential result to the proposed method.

THEOREM 2 In an n-cube, if *path*(A,B) and *path*(E,F) are disjoint, and if there exists a node C such that $C{\downarrow}E <\!\bullet\, C{\downarrow}A$, then *path*(C,B) and *path*(E,F) are disjoint.

Proof: If *path*(A,B) and *path*(E,F) are disjoint, by Theorem 1, we have $One((A{\downarrow}E)\otimes(B{\uparrow}F))=0$. To prove that *path*(C,B) and *path*(E,F) are disjoint, we must verify $One((C{\downarrow}E)\otimes(B{\uparrow}F))=0$. Since $C{\downarrow}E \lessdot C{\downarrow}A$, by Lemma 2, we obtain $C{\downarrow}E \lessdot A{\downarrow}E$. This implies that

$$(C{\downarrow}E)\otimes(B{\uparrow}F) \lessdot (A{\downarrow}E)\otimes(B{\uparrow}F)$$

and

$$One((C{\downarrow}E)\otimes(B{\uparrow}F)) \le One(\,(A{\downarrow}E)\otimes(B{\uparrow}F)).$$

Now that $One((A{\downarrow}E)\otimes(B{\uparrow}F))=0$, we must have $One((C{\downarrow}E)\otimes(B{\uparrow}F))=0$; thus *path*(C,B) and *path*(E,F) are disjoint. ♦

We would like to take a point of view, for this moment at least: When two paths are known to be disjoint, by the simple comparison of Theorem 2, we can easily find a set of paths which are destined towards the same node as the first path, and which are disjoint from the second path. Consider two disjoint paths on a 4-cube: *Path*(0100, 1101) and *Path*(1010, 1100). Let us just run through the first four nodes of a 4-cube, {0000, 0001, 0010, 0011}, we obtain

$0000{\downarrow}1010 \lessdot 0000{\downarrow}0100$ (TRUE)
$0001{\downarrow}1010 \lessdot 0001{\downarrow}0100$ (TRUE)
$0010{\downarrow}1010 \lessdot 0010{\downarrow}0100$ (FALSE)
$0011{\downarrow}1010 \lessdot 0011{\downarrow}0100$ (TRUE).

We can quickly conclude that among these four paths three of them are disjoint from *Path*(1010, 1100): *Path*(0000, 1101), *Path*(0001, 1101) and *Path*(0011, 1101). We should point out, however, that the comparison described in Theorem 2 provides only a sufficient condition for creating disjoint paths. In other words, if such a comparison fails, it does not necessarily mean that the two paths have link contentions; they could still be disjoint. The importance of the comparison lies in its simplicity. The comparison provides a simple means of changing the source node of a path while keeping it disjoint from other existing paths. This theorem will be appreciated soon.

4. THE PROPOSED METHOD

This section presents a method of configuring random nodes to form a multicast ring. The approach of the method is to merge rings repeatedly to form larger rings. It is important that the disjoint-path property must be held in the course of merging. In the following discussion, we first present some essential results for holding this property. Later these results will be turned into key steps of the proposed method. Up to now, we restrict our discussion to cases with two disjoint paths; we have considered conditions under which a pair of paths become disjoint and their intrinsic properties. As our discussion continues, we will find that these results can be easily employed in creating a set of disjoint paths. In particular, we are interested in a form of disjoint paths: they are a set of paths contained in a subcube.

As mentioned earlier, we use the descending dimension order to route paths. When source and destination nodes of a set of paths are located in a subcube, links traversed by the paths fall exclusively on links of the subcube. Based on this, we can make a simple observation: two sets of paths on two separate subcubes are mutually disjoint. Now the question is how to link nodes on two separate subcubes. Obviously, we need to create paths across subcubes. However, paths created for this purpose will have the source nodes on one subcube and the destination nodes on the other. When paths cross the boundary of subcubes, they may cause link contentions with other paths in existence. Next, we will introduce a systematic method of linking nodes across the boundary of subcubes. The method relies on two deliberate selections for a pair of subcubes and a pair of paths as well. Subcubes under consideration for selection have the following relation:

DEFINITION 6 Two k-subcubes, $\mathbb{C}_1$ and $\mathbb{C}_2$, are said to be buddy subcubes if for every node $X \in \mathbb{C}_1$, there exists a node $Y \in \mathbb{C}_2$ such that X and Y differ only at the k^{th} address bits.

Note that the above definition restricts buddy k-subcubes of an n-cube to be two k-subcubes whose nodes share exactly the highest n-k-1 address bits; this is different from the general definition of subcubes, which only requires the sharing of n-k-1 address bits in arbitrary positions. Apparently, our restriction is due to the descending dimension order used for path routing. Notation for such buddy subcubes is described with the following example: We use {00yy} and {01yy} to denote two 2-subcubes on a 4-cube; {00yy} comprises nodes 0000, 0001, 0010, and 0011; and {01yy} comprises nodes 0100, 0101, 0110, and 0111. In Figure 1, we illustratethe the hierarchy of all buddy subcubes of a 4-cube. Note that buddy subcubes of variable sizes are boxed.

Now we consider the selection of a pair of paths from two buddy subcubes and reroute the paths for linking these subcubes. This will be formally described by the following theorem. Before proceeding to the theorem, we give a conceptual description to show how paths are rerouted for linking buddy subcubes.

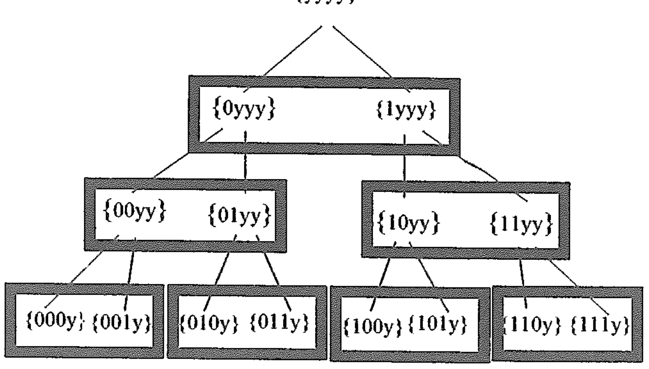


Figure 1. The hierarchy of buddy subcubes of 4-cube.

As illustrated in Figure 2, suppose that we are given two buddy subcubes and each of them contains a set of disjoint paths. And suppose that *Path*(A, B) and *Path*(C, D) are the two paths whose source nodes have the longest low-order address bits in common than other pairs of paths; namely, their addresses share the longest postfix. Note that we intentionally use proximity to symbolize that addresses of nodes A and C share the longest postfix. The two paths are selected. Then we interchange their destination nodes to link the two buddy subcubes. To be more specific, we create two paths, *Path*(A, D) and *Path*(C, B), to replace *Path*(A, B) and *Path*(C, D).

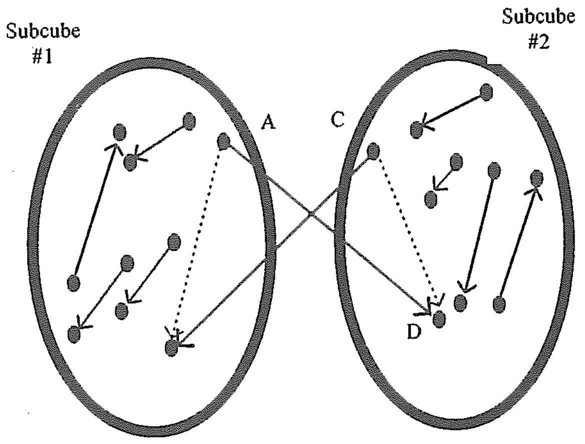


Figure 2. Concept of linking two buddy subcubes.

Next, we need to prove that such an interchange do not cause contentions with other existing paths of the buddy subcubes.

THEOREM 3 Let $\mathbb{C}_1$ and $\mathbb{C}_2$ be two buddy k-subcubes. Let $\{Path_1\}$ and $\{Path_2\}$ denote a set of disjoint paths of $\mathbb{C}_1$ and $\mathbb{C}_2$, respectively. Suppose that *path*(A,B)$\in\{Path_1\}$ and *path*(C,D)$\in\{Path_2\}$ are a pair of paths such that *Asterisk*(A$\downarrow$C)$\leq$*Asterisk*(X$\downarrow$W) for all path pairs, *path*(X,Y)$\in\{Path_1\}$ and *path*(W,Z)$\in\{Path_2\}$. Then *path*(A,D), *path*(C,B) and the other paths of $\{Path_1\}$ and $\{Path_2\}$ are disjoint.

Proof: We first prove that *path*(C,B) and the other paths of $\{Path_1\}$ and $\{Path_2\}$ are disjoint. Aassume that *path*(E,F)$\in\{Path_1\}$, A$\neq$E, and *path*(G,H)$\in\{Path_2\}$, C$\neq$G.

(1) *path*(C,B) and *path*(G,H) are disjoint.

Since C and G are two different nodes of $\mathbb{C}_2$, C and G must differ in some bit position i, $0\leq i\leq k-1$. Hence, *Asterisk*(C$\downarrow$G)$\geq$n-k. Since B$\in\mathbb{C}_1$ and H$\in\mathbb{C}_2$, B and H must differ in the kth bit position. Hence, *Asterisk*(B$\uparrow$H)=k+1. Apparently, we have *One*((C$\downarrow$G)$\otimes$(B$\uparrow$H))=0. As a result, by Theorem 1, *path*(C,B) and *path*(G,H) are disjoint.

(2) *path*(C,B) and *path*(E,F) are disjoint.

Because *Asterisk*(A$\downarrow$C)$\leq$*Asterisk*(X$\downarrow$W) for all path pairs, *path*(X,Y)$\in\{Path_1\}$ and *path*(W,Z) $\in\{Path_2\}$, we should have *Asterisk*(A$\downarrow$C) $\leq$*Asterisk*(C$\downarrow$E). This leads to C$\downarrow$E<$\bullet$C$\downarrow$A. By theorem 2, we are certain that *path*(C,B) and *path*(E,F) are disjoint.

In a similar manner, we can prove that *path*(A, D) and *path*(E, F) as well as *path*(G, H) are mutually disjoint. By Lemma 1, we see that *path*(A, D) and *path*(C, B) are disjoint. We conclude that *path*(A, D), *path*(C, B) and the other paths of $\{Path_1\}$ and $\{Path_2\}$ are disjoint. ♦

THE PROPOSED METHOD

```
for k:=1 to n do
  for all subcubes {$x_{n-1}...x_k y...y$}, $x_{n-1},...,x_k \in \{0,1\}$ in
      parallel do
    if a subcube comprises two selected nodes, A
        and B, which are not linked by a ring
      then Create Path(A, B) and Path(B, A);
    else
    if a subcube comprises a single ring and a
        separate selected node C
      then Find Path(A, B) of the ring such that
            Asterisk(A↓C Subcube ⟨↓C), for all the
              nodes X of      #2
            Create Path(A[illegible] [illegible]th(C, B) to replace
              Path(A, B);
    else
    if a subcube comprises two rings
      then Find a pair of paths Path(A, B) and
              Path(C, D) from the two rings, respectively
              such that Asterisk(A↓C)≤Asterisk(X↓W),
              for all other pairs of paths, Path(X, Y) and
              Path(W, Z), of the two rings;
            Create Path(A, D) and Path(C,B) to replace
              Path(A, B) and Path(C, D);
    fi fi fi
  od
od
```

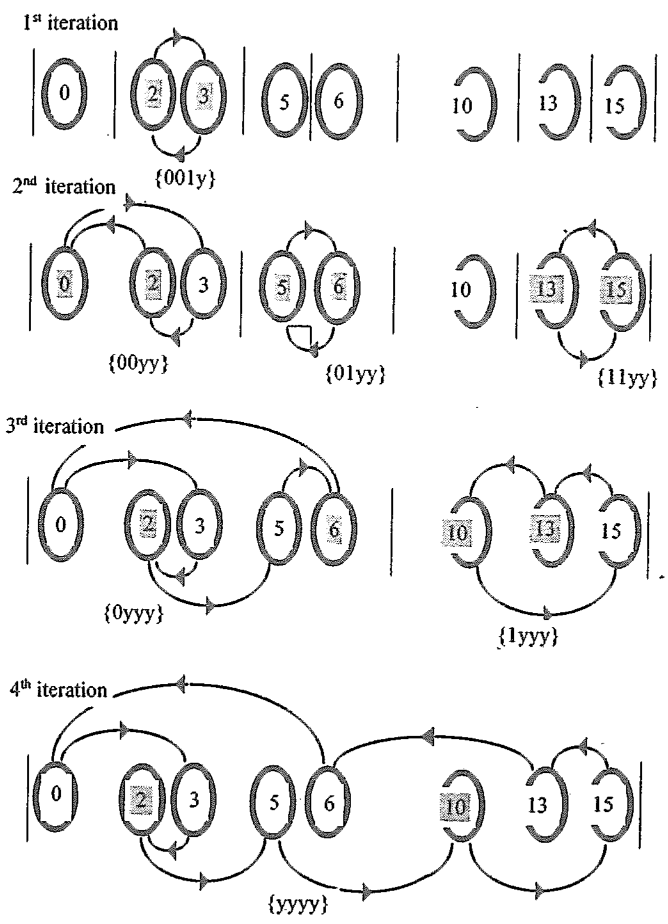
At this point, an explanation of a fine point concerning the intended semantics of our algorithmic notation is in order. The proposed method consists of two nested for loops. For an n-cube, the outer for loop requires *n* iterations for configuring the selected nodes. It proceeds from *1* to *n*. The inner for loop is actually a parallel loop with a variable number of steps, depending on the iteration count of the outer for loop; the kth iteration, $1\leq k\leq n$, contains 2^{n-k} parallel steps, each of which corresponds to a k-subcube. In each step, the corresponding subcube is examined. A step is skipped if its corresponding subcube contains a single ring, a single selected node or no selected nodes at all. Various actions will be taken to change the configuration of the selected nodes in a subcube. The three if statements in the inner for loop define actions to be taken in the three cases where reconfiguration is needed:

(1) If a subcube comprises exactly two selected nodes, then two paths are created to connect the two nodes into a length-2 ring;
(2) If a subcube contains a ring plus a separate selected node, then the ring merges the node;
(3) If a subcube contains two rings, then the two rings are combined into a larger ring.

In essence, the proposed method is an iterative procedure; it repeatedly merges rings on buddy subcubes into larger rings until all the selected nodes are connected to form a ring. Let us pause to look at an example for illustrating the mechanics of the proposed method. Then we will proceed to examine its theoretical aspects. Suppose that the following nodes are selected for configuring a multicast ring on a 4-cube: nodes 0, 2, 3, 5, 6, 10 13, 15. Figure 3(a) shows four snapshots taken at the end of the four iterations. In each iteration, we mark nodes chosen for changing paths and label subcubes involved in reconfiguration of rings. To verify our result, Figure 3(b) shows the links traversed by the individual paths of the final ring.

Now let us consider the theoretical aspects of the proposed method. Basically we need to show two things. First, our proposed method is logically sound; it does connect the selected nodes to form a logical ring, without considering path routing. Secondly, each step of the method does not incur link contentions. We show our first claim by induction. The proposed method starts with 1-subcubes. At this point, each 1-subcube contains at most two selected nodes. If it does, a length-2 ring is formed. At the end of the first iteration, there are three possibilities for a 1-subcube: (1) it contains a length-2 ring; (2) it contains a single selected node; and (3) it contains no selected nodes. Now suppose that, in the beginning of the k^{th} iteration, $2 \leq k \leq n$, a (k-1)-subcube contains (1) a ring, (2) a single selected node or (3) no selected nodes. Note that a k-subcube comprises two buddy (k-1)-subcubes. Two buddy (k-1)-subcubes in the three forms yield six combinations: two rings, two separate selected nodes, a ring plus a separate node, a ring, a single selected node, and no selected nodes. In other words, these six combinations become the six possible initial conditions of a k-subcube. The inner for loop takes care of the first three combinations and produces a ring. The remaining three combinations fall through the inner for loop without changes. Thus, at the end of the k^{th} iteration, a k-subcube contains (1) a ring, (2) a single node or (3) no selected nodes. Therefore, we assert our first claim.

Now let us consider the second claim that no link contentions occur at every step in the course of merging rings. There are three cases in which new paths are created for merging rings. In fact, the main body -- the third case -- has been proved by Theorem 3. The other two cases are shown by the following theorems.



(a)

0000 → 0010 → 0011
0010 → 0110 → 0100 → 0101
0011 → 0010
0101 → 1101 → 1001 → 1011 → 1010
0110 → 0010 → 0000
1010 → 1110 → 1111
1101 → 0101 → 0111 → 0110
1111 → 1101

(b)

Figure 3. Example of configuring a broadcast ring and its path routing.

THEOREM 4 Suppose that A and B are two different nodes of an n-cube. Then *Path*(A, B) and *Path*(B, A) are disjoint.

Proof. Let k= *Asterisk*(A↓B). Since $A \neq B$, $k > 0$ and *Asterisk*(B↑A)$\geq$n-k. This leads to

$$Asterisk(A\downarrow B) + Asterisk(B\uparrow A) \geq n.$$

Therefore, by Corollary 2, we conclude that *Path*(A, B) and *Path*(B, A) are disjoint. ♦

THEOREM 5 Let $\mathbb{C}_1$ and $\mathbb{C}_2$ be two buddy k-subcubes. Suppose that *Path*(A, B) and *Path*(E, F) are two disjoint paths of $\mathbb{C}_1$. And suppose that C is a node of $\mathbb{C}_2$ and $C{\downarrow}E <\bullet C{\downarrow}A$. Then *Path*(A, C), *Path*(C, B) and *Path*(E, F) are disjoint.

Proof: We need to prove the three paths, namely *Path*(A, C), *Path*(C, B) and *Path*(E, F), are mutually disjoint.

Path(A, B) and *Path*(E, F) are two disjoint paths of $\mathbb{C}_1$. Thus we have $One((A{\downarrow}E)\otimes(B{\uparrow}F)) = 0$. Since F is a node of $\mathbb{C}_1$ and C is a node of $\mathbb{C}_2$, we have $C{\uparrow}F <\bullet B{\uparrow}F$ and

$$One((A{\downarrow}E)\otimes(C{\uparrow}F)) \le One((A{\downarrow}E)\otimes(B{\uparrow}F)).$$

Consequently, $One((A{\downarrow}E)\otimes(C{\uparrow}F)) = 0$; that is, *path*(A, C) and *path*(E, F) are disjoint.

$\mathbb{C}_1$ and $\mathbb{C}_2$ together form a (K+1)-subcube. Since $C{\downarrow}E <\bullet C{\downarrow}A$, by Theorem 2, we assert that *path*(C,B) and *path*(E,F) must be disjoint.

Lastly, we need to show that *Path*(A, C) and *Path*(C, B) are disjoint. We know that A and B are nodes of $\mathbb{C}_1$, and C is a node of $\mathbb{C}_2$. Moreover, $\mathbb{C}_1$ and $\mathbb{C}_2$ are two buddy k-subcubes. Thus A and C must have different address bits at the k^{th} position. So do B and C. These yields

$$Asterisk(A{\downarrow}C) + Asterisk(C{\uparrow}B) \ge n.$$

By Corollary 2, we are sure that *Path*(A, C) and *Path*(C, B) are disjoint. ♦

5. CONCLUSIONS

We have presented a method of configuring randomly selected nodes to form rings on the binary n-cube. Most of the n-cube multiprocessors are one-port architectures; each processor node can transmit and receive only one message at a time. The rings are particularly suitable for them to perform group multicast operations. Central to the proposed method is a process of merging. It initially forms some basic rings of length 2. Then it repeatedly merge rings to form larger rings until the final ring is accomplished. Rings created by the proposed method constitute paths which share no common links. Therefore, messages can travel freely on these paths, without incurring link contentions. This is of great significance to the multicast operation, known to be communication-intensive.

The proposed method exhibits two exclusive features the RGC ring lacks. First, it guarantees that the physical distance between adjacent nodes is limited to n links, in contrast to $O(2^n)$ for the RGC ring. This allows us to better synchronize and overlap messages transmission around the selected nodes. Secondly, the proposed method eases message routing on the ring. It uses only the descending order of dimensions to resolve address bits, while the RGC ring requires variable dimension orders. In this paper, we assume wormhole routing for n-cube multiprocessors. The proposed method can be applied to other n-cube multiprocessors that use circuit switching, because the link-disjoint property is also preserved in the case of circuit switching.

REFERENCES

[1] L. Lan, A. H. Esfahanian and L. M. Ni, "Multicast in hypercube multiprocessors," *Journal of Parallel and Distributed Computing*, Jan. 1990, pp. 30-41.

[2] D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods,* Prentice-Hall, Engglewood Cliffs, NJ, 1989.

[3] S. L. Johnsson and C. T. Ho, "Optimal all-to-all personalized communication with minimum span on Boolean cubes," *Proc. 6th Distributed Memory Computing Conference*, 1991, pp. 299-304.

[4] Y. Saad and M. H. Schultz, "data communication in hypercubes," *Journal of Parallel and Distributed Computing*, 1989, pp. 115-135.

[5] Q. F. Stout and B. A. Wagar, " Passing messages in link-bound hypercubes," *Hypercube Multiprocessors 1987*, edited by M. T. Heath, SIAM, Philadelphia, 1987.

[6] V. Kumar, A. Grama, A. Gupta and G. Karypis, *Introduction to Parallel Computing,* Benjamin/Cummings Publishin Company, 1994.

[7] P. K. McKinley, H. Xu, A. H. Esfahanian and L. M. Ni, "Unicast-based multicast communication in wormhole routed networks," *IEEE Trans. on Parallel and Distributed Systems*, Vol. 5, No. 12, Dec. 1994, pp.1252-1265.

[8] L. M. Ni, P. K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, Feb. 1993, pp.62-76.

[9] X. Lin and L. M. Ni, "Deadlock free multicast wormhole routing in multicomputer networks, " *Proc. 18th Int'l Symposium on Computer Architecture*, 1991, pp.116-125.

[10] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," *IEEE Trans. on Computers*, Vol. 40, No.1, Jan. 1991, pp. 2-12.