JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 19, 183-203 (2003)

Broadcasting on Uni-directional Hypercubes and Its Applications*

HUANG-MING HUANG, CHANG-BIAU YANG
AND KUO-TSUNG TSENG
Department of Computer Science and Engineering
National Sun Yat-Sen University
Kaohsiung, 804 Taiwan
E-mail: cbyang@cse.nsysu.edu.tw

In this paper, we present three kinds of broadcasting tree for the even dimensional uni-directional hypercube (UHC) and its applications (ASCEND/DESCEND algorithms and bitonic sorting). For the n-dimensional UHC, under the constant evaluation model, one of our all-port broadcasting trees has height $n + 1$, which is optimal. Whereas the best one of our one-port broadcasting trees needs at most $\frac{4}{3}n+1$ steps ($\frac{4}{3}(n-n \bmod 6)+\frac{3}{2}(n \bmod 6)$ steps exactly). We also propose an all-port fault-tolerant broadcasting tree (a family of arc-disjoint spanning trees) whose height is no more than $\frac{3}{2}n+1$. At last, we show that the ASCEND/DESCEND algorithms and bitonic sorting can be implemented in the UHC with the same complexity as the hypercube under the half duplex mode. All of our algorithms can be easily applied to the odd dimensional UHC.

Keywords: interconnection network, uni-directional hypercube, broadcasting, fault-tolerance, ASCEND/DESCEND

1. INTRODUCTION

Among interconnection network topologies, the hypercube [5, 8, 15, 17, 23, 25] has been extensively studied because it has many advantages over other topologies, such as short diameter, short average distance, simple connection method, ease of routing, node symmetry and edge symmetry. In a hypercube, adjacent nodes can communicate with each other through the link connecting them. In fact, most existing hypercube multiprocessor architectures, such as Caltech/JPL Mark II [26] and NCUBE/10 [21], use two uni-directional links to simulate a bi-directional link in the hypercube topology. This approach, however, will double the degree of a node, and thus increase the cost and difficulty of constructing hypercubes of larger size.

With the rapid development of computers, how to connect existing computers efficiently to provide digital services attracts a lot of attention. Therefore, several topologies of Metropolitan Area Networks (MANs), like Manhattan Street Network [19, 20], HR^4-NET [4] and Tree-Net [13], were proposed for use in commercial network services. These topologies seem to have a large diameter, long latency delay, or deficiency of parallelism. Thus, one might think that the hypercube would be a better alternative to MANs. One issue arising from implementing MANs based upon the

Received June 22, 2000; revised November 29, 2000, May 8 & June 26, 2001; accepted July 23, 2001.
Communicated by Gen-Huey Chen.
* This research work was partially supported by the National Science Council of the Republic of China under contract NSC 84-2213-E-110-005.

hypercube with optical fibers is the lack of bi-directional electrical/optical converters. Consequently, messages can be transmitted in only one direction. Although we can use two fibers to achieve the goal of bi-directional transmission, this leads to an increased hardware cost.

Based on the above reasons, Chou and Du [6] proposed the uni-directional hypercube (UHC). Each link in a hypercube is assigned a fixed direction. The UHC preserves the characteristic of small diameter of the hypercube. In addition, they also proposed a simple and efficient routing scheme for the UHC. In this paper, we will further investigate broadcasting and fault-tolerant broadcasting algorithms for the even dimensional UHC.

This paper is organized as follows. Section 2 describes the UHC and the communication model. We prove that the even dimensional UHC is node symmetric, and propose the broadcasting trees in sections 3 and 4. Next, we present our all-port fault-tolerant broadcasting tree in section 5. In section 6 we analyze the performance of our algorithms. In section 7 we show how to implement the ASCEND/DESCEND algorithms and bitonic sorting in the even dimensional UHC. Some concluding remarks and remaining problems are given in section 8.

2. PRELIMINARY

The n-dimensional hypercube can be modeled as a graph $Q_n = (V, E)$ with 2^n nodes and $n2^{n-1}$ edges. Each node represents a processor (or processing unit) and each edge represents a communication link between a pair of processors. Fig. 1 shows the 4-dimensional hypercube if we ignore the hat symbols and link directions. Note that two nodes are linked if and only if their identifiers differ by exactly one bit position. (Nodes are assigned binary numbers from 0 to $2^n - 1$ as identifiers.) Port i of node v represents the link which connects v with the node whose identifier differs from v in the ith bit. Note that the rightmost bit position is 0.

The n-dimensional uni-directional hypercube (n-UHC) is an orientation of Q_n; that is, each edge Q_n is assigned a fixed direction, either incoming or outgoing. The direction of each edge is assigned according to the following polarity function [6]:

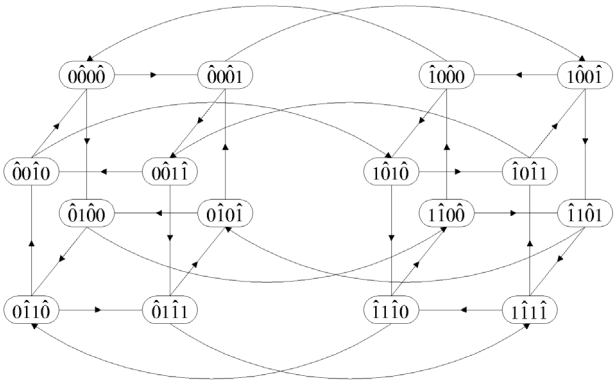

Fig. 1. The 4-diemensional uni-direction hypercube.

$$polarity\ (B,\ i) = \text{sign of } (-1)^{(b_{n-1}+b_{n-2}+\ \ldots+b_1+b_0+i)},$$

where $B = (b_{n-1}b_{n-2}\ \ldots b_1b_0)$ is the binary representation of the identifier of a node in the UHC, and i represents the port number.

If the polarity function is positive, port i is an out-port of node B; otherwise, port i is an in-port. *Out-port* (*in-port*) i of node v may be either port $2i$ or port $2i + 1$, depending on which one is the out-port (in-port). For example, Fig. 1 shows the 4-dimensional uni-direction hypercube (4-UHC) in which the link from 0000 to 0100 is out-port 1 (port 2) of 0000, in-port 1 of 0100.

For convenience we denote a node with hat symbols to represent its out-ports, such as $\hat{0}0\hat{0}\hat{0}$ which means ports 1 and 3 are the out-ports of 0000. The superdimension j of node v, denoted $SD_j(v)$, is comprised of bits $2j + 1$ and $2j$ of v together with their polarities. For instance, $SD_2(0\hat{0}1\hat{0}0\hat{1}1\hat{1}) = 1\hat{0}$. We can denote node S by $\langle s_{\frac{n}{2}-1}, s_{\frac{n}{2}-2},..., s_j,\ \ ..., s_1, s_0\rangle$, where s_j is the superdimension j of node S and contains two bits. $\bar{s}_j$ denotes the result of complementing the out-port bit of s_j and exchanging the polarity of these two bits. For example, $s_j = \hat{0}0$ has out-port $\hat{0}$, and $\bar{s}_j = 1\hat{0}$.

The UHC discussed above is also referred to as the *positive UHC*. By graph duality, there is another class of uni-directional hypercubes (referred as the *negative UHC*) with the opposite polarity function [6]:

$$polarity\ (B,\ i) = \text{sign of } (-1)^{(b_{n-1}+b_{n-2}+\ \ldots+b_1+b_0+i+1)}.$$

In fact, there is no difference between the positive UHC and the negative UHC, except for their link directions. Therefore, in the rest of the paper when we refer to UHC, we mean positive UHC, except where specifically pointed out.

Although the degree of n-UHC is equal to only a half of Q_n's degree, the diameter of n-UHC is only $n + 1$ when n is even, and $n + 2$ when n is odd. An n-UHC can be decomposed into a positive $(n - 1)$-UHC and a negative $(n - 1)$-UHC [6].

When a processor can communicate with all of its ports concurrently, we call it *all-port* communication. If a processor can only send and receive on one of its ports at any time, and the ports on which a processor sends and receives can be different [16], it is called *one-port* communication.

In this paper we adopt the model of communication cost proposed by Fraigniaud [11, 12]. The time T required for sending a message from one processor to one of its neighboring processors is assumed to be the sum of a start-up time β and a propagation time L_τ proportional to the length of messages L ($1/\tau$ is the bandwidth), i.e. $T = \beta + L_\tau$. This evaluation model is said to be linear. If $T = 1$, it is called the constant evaluation model. The constant evaluation model is used in most published papers. The authors of those papers, however, did not consider that the length of messages would play an important role in the communication cost. If the transmitted message is long, it may dominate the communication cost.

3. BROADCASTING TREES

Before we present our broadcasting trees, we shall first prove that an n-UHC is node symmetric if n is even. A graph is *node symmetric* if, given any two nodes v_1 and v_2, there exists an automorphism θ such that $\theta(v_1) = v_2$ [3]. If a graph is node symmetric, then it looks the same from every node in the graph. Thus, the routing and broadcasting algorithms for the network need not be modified for change of starting node. First, we define the operation $lrot(x)$ (or $rrot(x)$) as rotating the binary representation of x with its polarity one bit to the left (right). For example, $lrot(\hat{b}_3 b_2 \hat{b}_1 b_0) = b_2 \hat{b}_1 b_0 \hat{b}_3$.

Theorem 1. An even dimensional UHC is a node symmetric graph.

Proof: Given any two nodes $u = (u_{n-1}u_{n-2}\ldots u_1u_0)$ and $v = (v_{n-1}v_{n-2}\ldots v_1v_0)$ in n-UHC, n is even, we define the mapping $\theta: V \to V$ as

$$\theta(x) = \begin{cases} x \oplus u \oplus v & \text{if } polarity(u,0) = polarity(v,0), \\ lrot(x) \oplus lrot(u) \oplus v & \text{if } polarity(u,0) \neq polarity(v,0), \end{cases}$$

where $\oplus$ is an exclusive-or operation.

We can define an inverse function $\theta': V \to V$ as

$$\theta'(x) = \begin{cases} x \oplus u \oplus v & \text{if } polarity(u,0) = polarity(v,0), \\ rrot(x \oplus lrot(u) \oplus v) & \text{if } polarity(u,0) \neq polarity(v,0), \end{cases}$$

It is clear that $\theta'(\theta(x)) = x$, and therefore θ is bijective. If there exists a link (x, y) in n-UHC, it implies x and y differ in one bit position. Thus $\theta(x)$ and $\theta(y)$ also differ in one bit position; in other words, there exists a link between $\theta(x)$ and $\theta(y)$. Next, we shall examine the direction of the link. If $polarity(u, 0) = polarity(v, 0)$, the number of 1's in u and v will both be either even or odd. As a result, the number of 1's in $u \oplus v$ is even, and the directions of each port of x and $\theta(x)$ are the same. Consider the case that $polarity(u, 0) \neq polarity(v, 0)$. Suppose x and y differ at bit position i, then $\theta(x)$ and $\theta(y)$ will differ at bit position $i + 1 \pmod n$. In addition, the number of 1's in $lrot(u) \oplus v$ is odd. Thus port $i + 1 \pmod n$ in $\theta(x)$ and $\theta(y)$ will have the same direction as port i in x and y. Hence θ is a graph automorphism because any arc (x, y) in n-UHC implies the existence of arc $(\theta(x), \theta(y))$ in n-UHC. Therefore an even dimensional UHC is node symmetric. ❑

In addition to the node symmetry property, there is another property which is important for our subsequent algorithms. When we exchange bits $2i$ and $2i + 1$ of one node along with their polarities, node $(\hat{b}_{n-1}b_{n-2}\ldots\hat{b}_{2i+1}b_{2i}\ldots\hat{b}_1b_0)$ becomes $(b_{n-2}\hat{b}_{n-1}\ldots b_{2i}\hat{b}_{2i+1}\ldots b_0\hat{b}_1)$ and node $(b_{n-1}\hat{b}_{n-2}\ldots b_{2i+1}\hat{b}_{2i}\ldots b_1\hat{b}_0)$ becomes $(\hat{b}_{n-2}b_{n-1}\ldots\hat{b}_{2i}b_{2i+1}\ldots\hat{b}_0b_1)$.

Lemma 2. In an even dimensional positive UHC, if we exchange bits $2i$ and $2i + 1$ along

with their polarities in all node identifiers, for all $0 \le i \le \frac{n}{2}$, the resulting graph is an even dimensional negative UHC.

Proof: If we exchange every pair of bits $2i$ and $2i + 1$ along with their polarities, the polarity function will become that of a negative UHC. Thus the resulting graph is a negative UHC. ❑

A naive way to broadcast a message is to send a message along a broadcasting tree rooted at the source node. However, the performance depends highly upon the choice of the broadcasting tree. We propose three different broadcasting trees based on the one-port and all-port schemes.

In most interconnection networks, the divide-and-conquer strategy is usually used to construct broadcasting trees. In our first broadcasting tree, denoted n-BT$_1$, an n-UHC is recursively divided into four $(n-2)$-UHCs. (Only the even dimensional UHCs are node symmetric.) The message in the source node is first transmitted to the *sub-root* of each $(n-2)$-UHC; this sub-root becomes a source node in the $(n-2)$-UHC. Then this method is applied recursively. In n-BT$_1$, without loss of generality, we assume that the source node is $(00b_{n-3}\ldots b_1b_0)$, and we use the leftmost two bits of identifiers to divide an n-UHC. The sub-roots in the four $(n-2)$-UHCs are identified as 00, 01, 11 and 10, which represent the difference in the leftmost two bits between the sub-roots and the source node. Because the source node is embedded in a cycle of length four via its ports $n-1$ and $n-2$, and each node in the cycle is in a distinct $(n-2)$-UHC, the message can be broadcast to the nodes in the cycle via the leftmost out-ports of these nodes. To be precise, the message is sent via the path $(\hat{00}b_{n-3}\ldots b_1b_0) \rightarrow (\hat{01}b_{n-3}\ldots b_1b_0) \rightarrow (\hat{11}b_{n-3}\ldots b_1b_0) \rightarrow (\hat{10}b_{n-3}\ldots b_1b_0)$ to the sub-roots. And, the tree can be recursively constructed. For example, the graph shown in Fig. 2 is 4-BT$_1$ rooted at $0\hat{0}0\hat{0}$. Note that the distance between the root and the farthest sub-root is 3. The precise construction for n-BT$_1$ will be given in the next section.

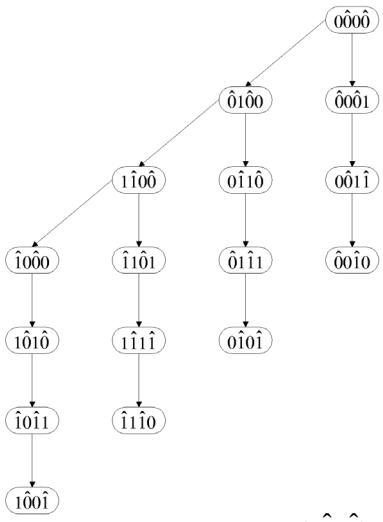

Fig. 2. 4-BT$_1$ rooted at $(0\hat{0}0\hat{0})$.

To improve n-BT$_1$, we have to reduce the distance between the root and the farthest sub-root. In n-BT$_1$, we use a cycle to transmit messages to the sub-roots of the other three $(n-2)$-UHCs. We construct the second broadcasting tree, denoted as n-BT$_2$, which uses two paths to transmit messages to the other sub-roots. In n-BT$_2$, the source $(0\hat{0}\hat{0}\hat{0}b_{n-5}...b_1b_0)$ sends the message via two paths $(0\hat{0}\hat{0}\hat{0}b_{n-5}...b_1b_0) \rightarrow (\hat{0}1\hat{0}\hat{0}b_{n-5}...b_1b_0)$ $\rightarrow (1\hat{1}\hat{0}\hat{0}b_{n-5}...b_1b_0)$ and $(0\hat{0}\hat{0}\hat{0}b_{n-5}....b_1b_0) \rightarrow (\hat{0}\hat{0}\hat{0}1b_{n-5}...b_1b_0) \rightarrow (1\hat{0}\hat{0}1b_{n-5}...b_1b_0)$. The subcube $10x_{n-3}\ldots x_1x_0$ does not receive messages from node $(1100b_{n-5}...b_1b_0)$ any more. Instead it receives messages from $(0001b_{n-5}\ldots b_1b_0)$. Hence the distance between the root and the farthest sub-root is reduced to 2, which is shorter than that of BT$_1$. The smallest BT$_2$ to show construction recursion is 6-BT$_2$, which is shown in Fig. 3.

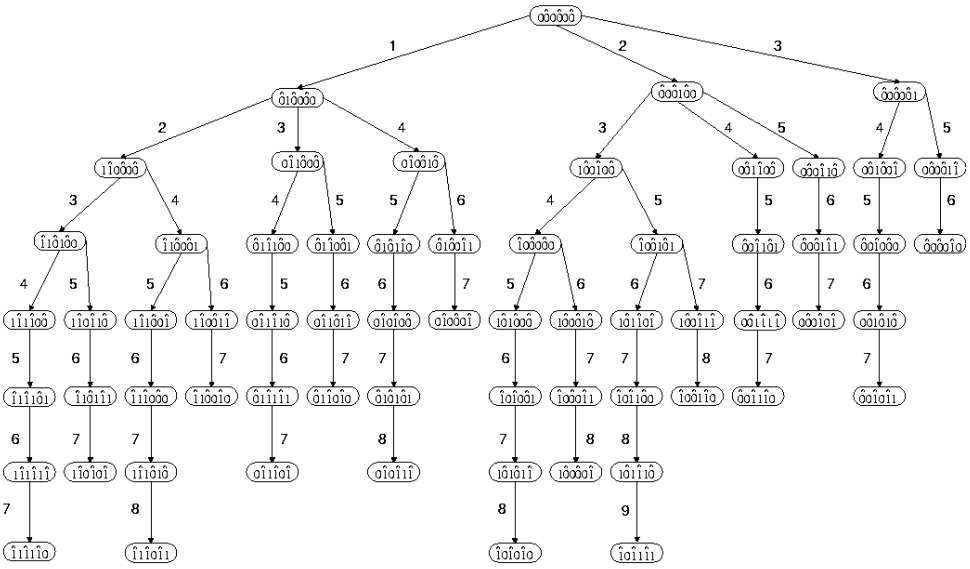

Fig. 3. 6-BT$_2$ rooted at $(0\hat{0}\hat{0}\hat{0}\hat{0}\hat{0})$.

Theorem 3. The height of n-BT$_1$ is $\frac{3}{2}n$, and that of n-BT$_2$ is $n + 1$, where n is even.

Proof: Let $H_1(n)$ and $H_2(n)$ denote the heights of n-BT$_1$ and n-BT$_2$, respectively. Obviously, it takes three steps to get from the root node to the sub-root $\hat{1}0$ in n-BT$_1$. Hence, we can derive a recursive function for BT$_1$, $H_1(n) = H_1(n-2) + 3$, $n \geq 4$, with $H_1(2) = 3$. Thus, $H_1(n) = \frac{3}{2}n$. In n-BT$_2$, the distance between the root and the farthest sub-root is 2. Hence, the recursive function is $H_2(n) = H_2(n-2) + 2$, $n \geq 4$, with $H_2(2) = 3$. The result is that $H_2(n) = n + 1$.

Corollary 4. For all-port communication in n-UHC, the broadcasting time steps required for BT$_1$ and BT$_2$ are $\frac{3}{2}n$ and $n + 1$, respectively, where n is even.

Corollary 5. For all-port communication in n-UHC, BT_2 is optimal.

Proof: Because the diameter of n-UHC is $n + 1$, it is impossible to construct a tree with height less than $n + 1$. Thus, for all-port communication with the constant evaluation model, BT_2 is optimal. ❑

Theorem 6. For one-port communication in n-UHC, the broadcasting time steps required for either BT_1 or BT_2 is $\frac{3}{2}n$ where n is even.

Proof: For one-port communication, we consider the order of the message transmission. In both BT_1 and BT_2, messages are sent first through the highest out-port (i.e., the out-port with the largest port number), since there are more nodes in the subtree connecting the highest out-port. BT_1 takes 3 steps from the source to the sub-root $\hat{1}0$. Thus, sub-root $\hat{1}0$ begins to broadcast at step 4. The other sub-roots $0\hat{0}, \hat{0}1$ and $1\hat{1}$ begin to broadcast in each of their subcubes at steps 2, 3 and 4, respectively. Therefore, we have the recursive function $T(n) = T(n - 2) + 3$ with $T(2) = 3$. The result is $T(n) = \frac{3}{2}n$. In BT_2, no matter how we arrange the order of message transmission, the root needs at least three steps to send messages to all of the other three sub-roots. Thus, the number of broadcasting steps in BT_2 is equal to that of BT_1. ❑

As an example, the number associated with each link in Fig. 3 is the time schedule in 6-BT_2 for one-port communication.

For one-port communication, we can construct a more efficient broadcasting tree, which we denote n-BT_3. 6-BT_3 is shown in Fig. 4. In the figure, we prune node (101111) and glue it as a child of node (111111). If the message is propagated in the order of the subscripts associated with the links, it takes only eight steps to finish one-port broadcasting. As a result, instead of dividing an n-UHC into four $(n - 2)$-UHCs as in BT_1 and BT_2, we now recursively divide an n-UHC into 64 $(n - 6)$-UHCs in BT_3. The root broadcasts the message to the sub-roots using 6-BT_3 as a base. If each subcube cannot be further divided into 64 units (i.e., each subcube contains less than 64 nodes), we use BT_1 to broadcast the message in each of the small subcubes. The number of time steps in one-port broadcasting required for BT_3 can be reduced. However, the method of construction is more complicated.

Theorem 7. For one-port communication in n-UHC, the broadcasting time steps required for BT_3 are $\frac{4}{3}(n - (n \bmod 6)) + \frac{3}{2}(n \bmod 6)$, where n is even.

Proof: Let $T(n)$ denote the time steps required for broadcasting in n-UHC using BT_3. If n is a multiple of 6, we have the recursive function $T(n) = T(n - 6) + 8$, $n > 6$ and $T(6) = 8$. Thus, we obtain the result $T(n) = \frac{4}{3}n$. For the case that n is not a multiple of 6, if each subcube cannot be further divided into 64 smaller cubes, we use BT_1 to accomplish the broadcast. The recursive function is now generalized as follows: $T(n) = T(n - 6) + 8$, $n > 6$, with $T(2) = 3$, $T(4) = 3$, and $T(6) = 8$. Thus it needs $\frac{4}{3}(n - (n \bmod 6)) + \frac{3}{2}(n \bmod 6)$ steps.

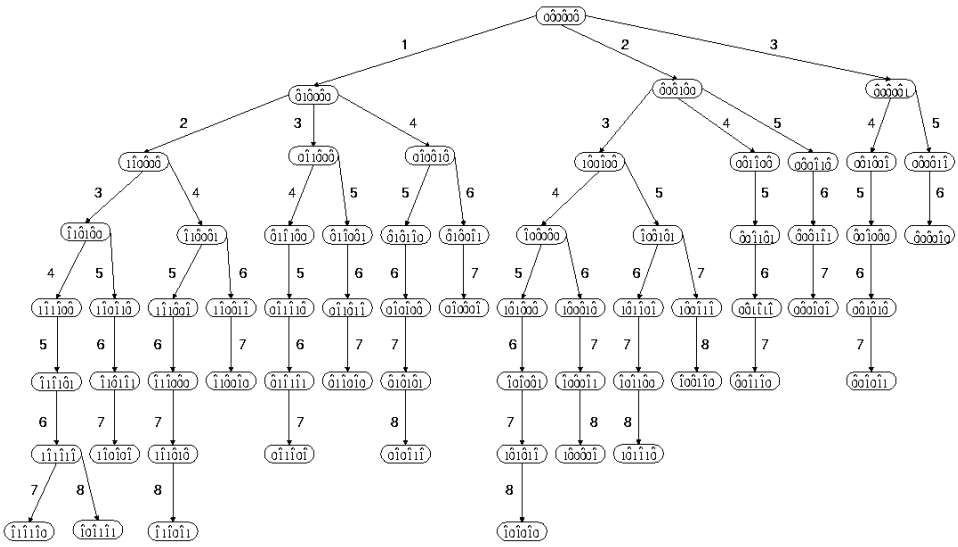

Fig. 4. 6-BT_3 rooted at $(0\hat{0}0\hat{0}0\hat{0})$.

4. CONSTRUCTION ALGORITHMS FOR THE BROADCASTING TREES

In this section, we present the distributed construction algorithms for the broadcasting trees proposed in the previous section.

In BT_1 a cycle of length four is used as the backbone to distribute the message from the source node to the sub-root of each subcube. Thus, an n-UHC is recursively divided into four $(n-2)$-UHCs, and BT_1 can be recursively constructed. In BT_2 we use two disjoint paths as the broadcast backbone. The recursion procedure is the same as that of BT_1. However, the tree height in BT_2 is reduced. BT_3 is constructed in a purely artificial manner. We first construct the broadcasting tree 6-BT_3 for 6-UHC as shown in Fig. 4. Then, we use 6-BT_3 as a backbone in the construction recursion. Hence, in BT_3 an n-UHC is recursively divided into 64 $(n-6)$-UHCs. If a subcube contains less than 64 nodes, BT_1 construction is applied.

BT_1 is a very simple structure. Without loss of generality we assume the root is the node with all bits being zero. We can easily define a children function to generate all children of a node in BT_1 as follows.

$$children(i) = \begin{cases} \langle i_{\frac{n}{2}-1},...,\bar{i}_k,...,i_0\rangle, \forall 0 \le k \le \frac{n}{2}-1 & \text{if } i \text{ is root,} \\ \langle i_{\frac{n}{2}-1},...,\bar{i}_k,...,i_0\rangle, \forall 0 \le k \le l & \text{if } i_l \in \{\hat{0}1, 1\hat{0}, \hat{1}1, 1\hat{1}\}, \\ \langle i_{\frac{n}{2}-1},...,\bar{i}_k,...,i_0\rangle, \forall 0 \le k < l & \text{if } i_l \in \{\hat{1}0, 0\hat{1}\} \text{ and } l \ne 0, \\ \phi & \text{if } i_l \in \{\hat{1}0, 0\hat{1}\} \text{ and } l = 0, \end{cases}$$

where l is the first superdimension, scanning from right to left, that is not $\hat{0}0$ or $0\hat{0}$. Note that the rightmost superdimension is 0.

BT_2 is much more complicated than BT_1, but first, we shall present some characteristics of BT_2.

Proposition 1. In BT_2, there does not exist a path $v_0e_0v_1e_1 \ldots e_{k-1}v_k$ such that e_i, e_{i+1} and e_{i+2}, are out-port j of v_i, v_{i+1} and v_{i+2}, respectively, with $j \neq 0$.

Proposition 2. In BT_2, if node u receives a message from in-port i, and its parent receives one from in-port j, where $j \geq i + 2$, then node u should send the message to each out-port k, $k \leq i + 1$.

For example, consider Fig. 3. Node (010010) is connected with its parent via in-port 0, while its parent is connected with its grandparent via in-port 2. Thus the children of node (010010) are connected with it via out-ports 1 and 0.

Proposition 3. In BT_2, node k is a leaf if and only if the last three arcs of the path from the root to node k use in-port 0 of the last three nodes.

For example, (111101), (111111) and (111110) are connected with their parents via in-port 0.

For BT_2, messages in an n-UHC should be sent to each $(n - 2)$-UHC in two steps except those in 2-UHCs. Hence, a message can be broadcast with a tag which informs the children where the message was from. The algorithm is as follows.

Algorithm 1. Distributed BT_2 construction algorithm

```
if the current node is the source node then
    send the message with tag = n/2 to all out-ports
else
    receive a message from in-port i
    if tag > i + 1 then
        send the message with tag = i to out-port j, ∀ 0 ≤ j ≤ i + 1
    else if tag = – 1
        stop
    else if i = 0 then
        set tag = tag – 1
        send the message with tag to out-port 0
    else if tag ≤ i then
        send the message with tag = i to out-port j, ∀ 0 ≤ j ≤ i – 1
    else if tag = i + 1
        send the message with tag = i to out-port j, ∀ 0 ≤ j ≤ i
        end
    end
end
```

Note that in this algorithm, the identifier of the source node is no longer needed. The tag, in fact, takes the place of the source identifier. Therefore, in comparison with the broadcasting algorithm in hypercubes, this algorithm does not need any extra field encapsulated in the message.

For instance, $(0\hat{0}0\hat{0}0\hat{0})$ sends the message with *tag* = 3 to $(\hat{0}1\hat{0}0\hat{0}0)$, $(\hat{0}0\hat{0}1\hat{0}0)$ and $(\hat{0}0\hat{0}0\hat{0}1)$. $(\hat{0}1\hat{0}0\hat{0}0)$ receives this message on in-port 2, and then relays it with *tag* = 2 to $(1\hat{1}0\hat{0}0\hat{0})$, $(0\hat{1}1\hat{0}0\hat{0})$ and $(0\hat{1}0\hat{0}1\hat{0})$. When $(0\hat{1}0\hat{0}1\hat{0})$ receives the message with *tag* = 2 on in-port 0, it relays the message via out-ports 1 and 0 with *tag* = 0.

Since n-BT$_3$ is based on 6-BT$_3$, we split the binary representation of a node into several sections, each containing 6 bits, except the rightmost section, i.e., section 0. Again we assume that the root is (0 … 0). Each node contains a table describing the basic information of a 6-BT$_3$, which is extracted from Fig. 4. Given a node in 6-BT$_3$, the table suggests to which out-port the node should begin sending the message. If the current node is the root, it sends the message via all of its out-ports. If a node receives a message from other node, the algorithm for this node is as follows:

Algorithm 2. Distributed BT$_3$ construction algorithm
receive a message from in-port i
scan the binary representation from right to left; find the first
section k whose content is not zero
if ($k \neq 0$) or (n mod 6 = 0) then
look up the construction table of 6-BT$_3$ with the content of section k to find out
to which out-port j it should start sending the message
if $k \neq 0$ then
send the message to out-port l,
$\forall\ 0 \leq l \leq j + 3(k - 1) + (n \bmod 6)/2$
else
send the message to out-port l, $\forall 0 \leq l \leq j + 3k$
else
use the content of section k to send the message according
to the construction algorithm for BT$_1$
end
end

For example, $(1\hat{0}0\hat{1}0\hat{0}0\hat{0})$ can be split into two sections, $1\hat{0}0\hat{1}0\hat{0}$ and $0\hat{0}$, see Fig. 4. $1\hat{0}0\hat{1}0\hat{0}$ sends the message via out-ports 1 and 0. Hence $(1\hat{0}0\hat{1}0\hat{0}0\hat{0})$ should send the message via out-ports 2, 1 and 0. By Lemma 2 either the construction table of the positive UHC or that of the negative UHC is sufficient for the needs in both cases. When we consult the table, the polarity should be consistent. Consider another example $(0\hat{1}0\hat{0}1\hat{1}0\hat{0})$. Its first section is $0\hat{1}0\hat{0}1\hat{1}$. Since, the polarity of $0\hat{1}0\hat{0}1\hat{1}$ is negative, $\hat{1}0\hat{0}0\hat{1}1$ rather than $0\hat{1}0\hat{0}1\hat{1}$ should be used to look up the table.

After we discuss broadcasting in even dimensional UHC, we shall describe how to broadcast in odd dimensional UHC. An n-UHC can be decomposed into a positive $(n - 1)$-UHC and a negative $(n - 1)$-UHC. Thus we can divide an odd dimensional n-UHC by bit $n - 1$. If port $n - 1$ of the source node v is an out-port, the source node sends the message to $v \oplus 2^{n-1}$; otherwise, it sends the message via the path $(v \rightarrow v \oplus 2^{n-2} \rightarrow v \oplus (2^{n-1} + 2^{n-2}))$. Then we can apply the broadcasting trees of the even dimensional UHC in

previous sections to broadcast. The cost of these algorithms will increase one step if port $n-1$ of the source node is an out-port, or increase two steps otherwise.

5. FAULT-TOLERANT BROADCASTING TREES

As the number of components in a distributed system increases, the probability that some components work incorrectly cannot be neglected. In this paper we consider both link failures and processor failures. A link or a processor is *faulty* if it cannot transmit any message. Note that in our model, a processor or a link either transmits or does not transmit messages; in other words, it does not corrupt messages. All faults are assumed to be permanent (or at least are considered to be permanent during the whole execution of the communication algorithm).

A graph (digraph) G is said to have *fault tolerance* k if, when as many as k arbitrary nodes are deleted from G, the resulting subgraph is still connected (strongly connected) [2]. An algorithm for communicating in a digraph $D(V, A)$ can tolerate f faults if and only if each data element can be routed through at least $f + 1$ node-disjoint paths from its source to its destination [11].

It has been proven by Edmonds [10] that every digraph possesses as many *arc-disjoint spanning trees* rooted at any node as the arc-connectivity of the digraph. Here, for a digraph $D = (V, A)$, *arc-connectivity* represents the minimum number of elements in an arc set A' such that $D' = (V, A - A')$ is not a strongly connected digraph. It is clear that the arc-connectivity of an n-UHC is at most $\frac{n}{2}$, where n is even. Thus n-UHC has at most $\frac{n}{2}$ arc-disjoint spanning trees (ADST) rooted at any node. In this section we propose an explicit construction of $\frac{n}{2}$ arc-disjoint spanning trees in an n-UHC, where n is even. This family of trees is denoted n-ADST. Also, we identify each spanning tree by the out-port number from where it is derived. For example, the right subtree in 4-ADST in Fig. 5 is called subtree 0.

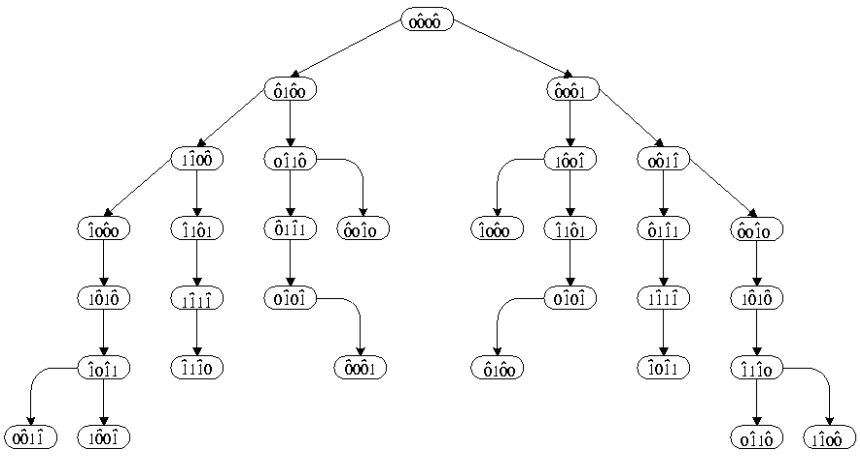

Fig. 5. 4-ADST rooted at $(0\hat{0}0\hat{0})$.

Our ADST is based on BT_1. Without loss of generality, let node (0 … 0) be the root. To begin, we show how to construct subtree $\frac{n}{2}-1$, which is very similar to BT_1, except that some nodes change the connecting positions in the tree. The root first sends the message via the path $(0\hat{0}0\cdots0)\rightarrow(\hat{0}10\cdots0)\rightarrow(1\hat{1}0\cdots0)\rightarrow(\hat{1}00\cdots0)$. Then nodes $(\hat{0}10\cdots0)$, $(1\hat{1}0\cdots0)$ and $(\hat{1}00\cdots0)$ serve as the roots of the subcubes $(01x_{n-3}\ \ldots\ x_1x_0)$, $(11x_{n-3}\ \ldots\ x_1x_0)$ and $(10x_{n-3}\ \ldots\ x_1x_0)$, respectively. These three nodes; in turn, broadcast the message as in BT_1. Finally, each node with identifier $(\hat{1}0x_{n-3}...x_1x_0)$ or $(0\hat{1}x_{n-3}...x_1x_0)$ sends the message via its out-port $\frac{n}{2}-1$. Subtree $j, 0\le j\le\frac{n}{2}-2$, can be obtained by rotating the identifiers of the nodes in subtree $\frac{n}{2}-1$ right by $n-2j-2$ bits. For example, the tree in Fig. 6 is subtree 1 of 6-ADST.

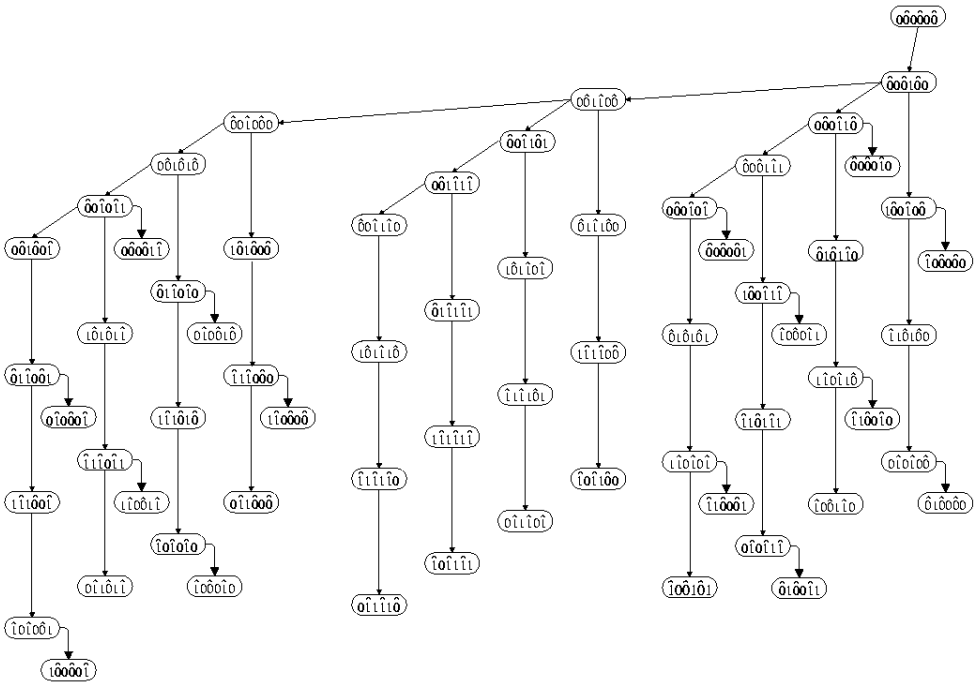

Fig. 6. Subtree 1 of 6-ADST rooted at $(0\hat{0}0\hat{0}0\hat{0})$.

For a given node i in subtree j, let l be the first superdimension position in the sequence $j+1, j+2,..., \frac{n}{2}-1,0,1,...j-1$ such that $SD_l(i)\notin\{\hat{0}0,0\hat{0}\}$. If there does not exist such a superdimension position, we set $l = -1$. For instance, suppose that $n = 10$ and $j = 2$. For $(\hat{1}1\hat{0}0\hat{1}0\hat{0}0\hat{0})$ and $(\hat{0}0\hat{0}0\hat{1}0\hat{0}0\hat{1}1)$, we have $l = 4$ and $l = 0$, respectively. Thus the children of node i in subtree j are given by

$$children(i) = \begin{cases} \langle i_{\frac{n}{2}-1} \cdots \overline{i_j} \cdots i_0 \rangle & \text{if } i \text{ is root,} \\ \langle i_{\frac{n}{2}-1} \cdots \overline{i_k} \cdots i_0 \rangle \quad \forall k \in Z_{\frac{n}{2}}(j, j+1) & \text{if } i_j \in \{\hat{1}1, 1\hat{1}, \hat{0}1, 1\hat{0}\} \text{ and } l = -1, \\ \langle i_{\frac{n}{2}-1} \cdots \overline{i_k} \cdots i_0 \rangle \quad \forall k \in Z_{\frac{n}{2}}(j-1, j+1) & \text{if } i_j \in \{\hat{1}0, 0\hat{1}\} \text{ and } l = -1, \\ \langle i_{\frac{n}{2}-1} \cdots \overline{i_k} \cdots i_0 \rangle \quad \forall k \in Z_{\frac{n}{2}}(l, j+1) & \text{if } i_j \in \{\hat{1}1, 1\hat{1}, \hat{0}1, 1\hat{0}\} \text{ and } i_l \notin \{\hat{1}0, 0\hat{1}\}, \\ \langle i_{\frac{n}{2}-1} \cdots \overline{i_k} \cdots i_0 \rangle \quad \forall k \in Z_{\frac{n}{2}}(l-1, j+1) & \text{if } i_j \in \{\hat{1}1, 1\hat{1}, \hat{0}1, 1\hat{0}\} \text{ and } i_l \in \{\hat{1}0, 0\hat{1}\} \text{ and } l \neq j+1, \\ \langle i_{\frac{n}{2}-1} \cdots \overline{i_k} \cdots i_0 \rangle \quad \forall k \in Z_{\frac{n}{2}}(l, j) & \text{if } i_j \in \{\hat{1}0, 0\hat{1}\} \text{ and } i_l \notin \{\hat{1}0, 0\hat{1}\}, \\ \langle i_{\frac{n}{2}-1} \cdots \overline{i_k} \cdots i_0 \rangle \quad \forall k \in Z_{\frac{n}{2}}(l-1, j) & \text{if } i_j \in \{\hat{1}0, 0\hat{1}\} \text{ and } i_l \in \{\hat{1}0, 0\hat{1}\}, \\ \emptyset & \text{if } (i_j \in \{\hat{1}1, 1\hat{1}, \hat{0}1, 1\hat{0}\} \text{ and } i_l \in \{\hat{1}0, 0\hat{1}\} \text{ and } l = j+1) \text{ or } (i_j \in \{\hat{0}0, 0\hat{0}\} \text{ and } l \neq -1), \end{cases}$$

where

$$Z_{\frac{n}{2}}(x, y) = \begin{cases} \langle x, x-1, \cdots, y+1, y \rangle & \text{if } x \geq y, \\ \langle x, x-1, \cdots, 0, \frac{n}{2}-1, \cdots, y+1, y \rangle & \text{if } x < y. \end{cases}$$

Lemma 8. The $n/2$ subtrees in an n-ADST graph are arc-disjoint, for even n.

Proof: By definition, subtree $\frac{n}{2}-1$ of n-ADST is based on n-BT_1. Subtree j, $0 \leq j \leq \frac{n}{2}-2$, can be obtained by rotating the identifiers of the nodes in subtree $\frac{n}{2}-1$ right by $n-2j-2$ bits. So, it is clear that the $n/2$ subtrees in n-ADST graph are arc-disjoint. ❑

Lemma 9. In subtree j of n-ADST, for n even, the nodes in $V' = \{v \mid SD_j(v) \in \{\hat{0}0, 0\hat{0}\}$, where v is not the root$\}$ are all leaves.

Proof: It is obvious by the construction method. Without the last step, the subtree does not contain any node v such that $SD_j(v) \in \{\hat{0}0, 0\hat{0}\}$. In the last step, every node in V' will attach a node u such that $SD_j(u) \in \{\hat{1}0, 0\hat{1}\}$. Therefore the nodes in V' are all leaves.❑

Theorem 10. Two paths between the source and any other node in two subtrees of the n-ADST, for n even, are node-disjoint.

Proof: We shall prove this theorem by contradiction. Let $P_j(x)$ denote the node set $\{v_1, v_2, \ldots, v_k\}$ such that the path $(s \to v_1 \to v_2 \to \ldots \to v_k \to x)$ is a path from the source s to node x in subtree j. Suppose that $P_i(x) \cap P_j(x) \neq \phi, i \neq j$. Let $y = \langle y_{\frac{n}{2}-1} \ldots y_0 \rangle \in P_i(x) \cap P_j(x)$. Without loss of generality, one can assume $j = \frac{n}{2} - 1$. Since $y \in P_i(x) \cap P_j(x)$, y is a nonleaf node in both subtree i and subtree $\frac{n}{2} - 1$. From Lemma 9, $y_{\frac{n}{2}-1}$ and $y_i \notin \{\hat{0}0, 0\hat{0}\}$. Let r and t be the first superdimension such that y_r and $y_t \notin \{\hat{0}0, 0\hat{0}\}$ counted cyclically left to right from superdimension 0 and superdimension $i + 1$, respectively. From our construction algorithm, $i \geq r \geq 0$ and $\frac{n}{2} > t \geq i + 1$, because $y_{\frac{n}{2}-1}$ and $y_i \notin \{\hat{0}0, 0\hat{0}\}$. Consequently, the descendants of y in subtree $\frac{n}{2} - 1$ differ from y only in the superdimensions which range from r to 0; the descendants of y in subtree i differ from y only in the superdimensions which range from t to i. However, these two intervals $[0, r]$ and $[i, t]$ are disjoint. Node y has no descendant in both subtree $\frac{n}{2} - 1$ and subtree i, and thus, node y is a leaf node. This contradicts that y is a nonleaf node. Therefore, there is no such node y and $P_i(x) \cap P_j(x) = \phi$. ❑

From the above two theorems, we have the following corollary:

Corollary 11. The fault tolerance of n-UHC is $\frac{n}{2} - 1$, for even n.

Theorem 12. The height of n-ADST is $\frac{3}{2}n + (n \bmod 4)/2$, for even n.

Proof: Since every subtree of n-ADST is of the same height, we consider only subtree $\frac{n}{2} - 1$. Without the last step, the height of subtree $\frac{n}{2} - 1$ is the same as BT_1. Using the same reasoning as in Lemma 9, the last step increases the height of the tree by only 1. Let x be the node at level $\frac{3}{2}n$ (the level of root is 0). If n is not a multiple of 4, $SD_0(x) = \hat{1}0$, because the distance from the root to x is odd and x is the descendant of $(0 \ldots 0\hat{0}1)$. Thus the height of n-ADST is $\frac{3}{2}n + 1$ when n is not a multiple of 4. Similarly, if n is a multiple of 4, then $SD_0(x) = 1\hat{0}$, and the height of n-ADST is $\frac{3}{2}n$. ❑

In our n-ADST, no link appears more than once in all spanning trees. However, at every step, each processor may appear more than once in all spanning trees. In other words, some processors are required to have a multiple-port communication capability or at least a buffering capability when multiple-port communication is encountered.

6. PERFORMANCE ANALYSIS

Let h denote the height of a broadcasting tree. The cost of sending a message of length L under all-port communication is $h(\beta + L\tau)$. In the case of long messages, we can use the pipeline technique proposed by many researchers [11, 12, 16, 24] to speed up broadcasting under all-port communication. First, we cut the message into L/B packets of size B. The message can then be sent one packet after another into the pipeline. The broadcasting time is $h(\beta + B\tau)$ for the first packet to reach the farthest node, plus $(L/B - 1)(\beta + B\tau)$ for the remaining packets. This gives a total time of $(h - 1 + \frac{L}{B})(\beta + B\tau)$. Minimizing the total time by selecting the packet size, we get the minimum time

$(\sqrt{(h-1)\beta}+\sqrt{L_\tau})^2$ when $B=\sqrt{\frac{L\beta}{(h-1)_\tau}}$.

Now we consider all-port fault-tolerant broadcasting. Let f denote the number of faults in n-UHC. If $f = n/2 - 1$, the message is duplicated $n/2$ times, and each copy is sent through one subtree in n-ADST. The time complexity is $(\frac{3}{2}n+\frac{n \bmod 4}{2}-1+\frac{L}{B})(\beta+B_\tau)$, where n is even and B is the packet size. If $f < n/2 - 1$, we cut the message into $n/2$ blocks, $M_0, M_1, \ldots, M_{n/2-1}$. Each subtree j transmits the message, which is the composition of blocks $M_k, \forall k \in \{j, j+1, \ldots, (j+f) \bmod (n/2)\}$. If there is no fault, each node can receive $f + 1$ copies of the original message, because each block is transmitted via $f + 1$ subtrees. Thus each node can receive at least one copy of the message if there are f faults. Since each subtree transmits a message of length $2(f + 1)L/n$, the time required for the algorithm becomes $(\frac{3}{2}n+\frac{n \bmod 4}{2}-1+\frac{2(f+1)L}{B_n})(\beta+B_\tau)$.

If we do not consider the fault-tolerant tree, the formula discussed above is still valid in the one-port communication, except that the term h should be replaced by the total start up time needed. Table 1 is a summary of the time complexities of our broadcasting trees.

Table 1. Time complexities of broadcasting in n-UHC.

Communication model	Routing	Constant evaluation model	Linear evaluation model	
			Optimal packet size	Minimum time
one-port	BT_1	$\frac{3}{2}n$	$\sqrt{\frac{L\beta}{(\frac{3}{2}n-1)\tau}}$	$(\sqrt{(\frac{3}{2}n-1)\beta}+\sqrt{L\tau})^2$
	BT_2	$\frac{3}{2}n$	$\sqrt{\frac{L\beta}{(\frac{3}{2}n-1)\tau}}$	$(\sqrt{(\frac{3}{2}n-1)\beta}+\sqrt{L\tau})^2$
	BT_3	$x_1(\approx \frac{4}{3}n)$	$\sqrt{\frac{L\beta}{(x_1-1)\tau}}$	$(\sqrt{(x_1-1)\beta}+\sqrt{L\tau})^2$
all-port	BT_1	$\frac{3}{2}n$	$\sqrt{\frac{L\beta}{(\frac{3}{2}n-1)\tau}}$	$(\sqrt{(\frac{3}{2}n-1)\beta}+\sqrt{L\tau})^2$
	BT_2	$n+1$	$\sqrt{\frac{L\beta}{n\tau}}$	$(\sqrt{n\beta}+\sqrt{L\tau})^2$
	BT_3	$x_2(\approx \frac{7}{6}n)$	$\sqrt{\frac{L\beta}{(x_2-1)\tau}}$	$(\sqrt{(x_2-1)\beta}+\sqrt{L\tau})^2$
	ADST	$\frac{3}{2}n+\frac{n \bmod 4}{2}$	$\sqrt{\frac{2(f+1)L\beta}{(x_3-1)n\tau}}$	$(\sqrt{(x_3-1)\beta}+\sqrt{\frac{2(f+1)L\tau}{n}})^2$

$x_1 = \frac{4}{3}(n - n \bmod 6) + \frac{3}{2}(n \bmod 6)$
$x_2 = \frac{7}{6}(n - n \bmod 6) + \frac{3}{2}(n \bmod 6)$
$x_3 = \frac{3}{2}n + \frac{n \bmod 4}{2}$
L : message length.
f : number of faults.
β : start-up time.
τ : a constant parameter.

7. APPLICATIONS OF UHC

In this section, we will present ASCEND/DESCEND algorithms on the UHC. These algorithms are based on one-port communication, and do not possess fault-tolerant ability.

The ASCEND/DESCEND algorithms [22] are two classes of parallel algorithms which are derived from the divide-and-conquer paradigm. Assume that input data δ_0, $\delta_1, \ldots, \delta_{n-1}$ are stored, respectively, in storage locations $T[0], T[1], \ldots, T[n-1]$, and the

number of inputs is a power of 2 (i.e., $n = 2^k$). An algorithm is in the ASCEND (DESCEND) class if it performs a sequence of basic operations on pairs of data that are successively 2^0, 2^1, …, 2^{k-1} (2^{k-1}, 2^{k-2}, …, 2^0) locations apart. Each basic operation OPER(m, j; U, V) modifies the two data items stored in locations U and V; the computation performed affects only the contents of U, V and may depend upon parameters m and j, where $0 \le m \le n-1$, $0 \le j \le k-1$. Algorithms in the ASCEND class can then be specified as follows:

Algorithm 3. *ASCEND*
for $j = 0$ to $k-1$ do
 for each m, $0 \le m \le n-1$, do in parallel
 if the jth bit of m is 0 then
 OPER(m, j; $T[m]$, $T[m \oplus 2^j]$)
end

For some applications, such as *bitonic merge* and *cyclic shift*, the corresponding algorithms are directly within the ASCEND or DESCEND class. Other applications (such as *permutation*, *shuffle*, *unshuffle*, *bit-reversal*, *odd-even merge*, *Fast Fourier Transform*, *convolution*, *matrix transposition*) have programs consisting of a short sequence of algorithms in these two classes. As for applications such as *bitonic sort*, *odd-even merge sort*, and *calculations of symmetric functions*, they are the combination of the two results of a recursive call which is in the two classes.

It is quite easy to implement ASCEND/DESCEND algorithms in hypercubes, since, by the definition of a link in a hypercube, two nodes u and v are adjacent if and only if $u \oplus v = 2^j$, i.e., $|u - v| = 2^j$, $0 \le j \le k-1$. Let δ_m denote the original data stored in node m, and let δ_m^j denote the data stored in node m after the completion of iteration $j-1$ (note that j is counted from 0). The *for* statement in the ASCEND algorithm can be replaced by:

for each node m, $0 \le m \le n-1$, do in parallel
 send δ_m^j to node $m \oplus 2^j$
 receive $\delta_{m\oplus 2^j}^j$ from node $m \oplus 2^j$
 OPER (m, j; $\delta_m^j, \delta_{m\oplus 2^j}^j$)

In the UHC, each *send* or *receive* in the above algorithm takes three steps to accomplish. It is not efficient for the UHC to perform the algorithms of the hypercube. Therefore we have to modify it to suit the UHC.

Since nodes m and $m \oplus 2^j$ cannot directly communicate with each other in the UHC, we combine two iterations, say $2j$ and $2j+1$, of the original algorithm. Here we assume n is even. Let v_0 be a node whose port $2j$ is an out-port, and $v_1 = v_0 \oplus 2^{2j}$, $v_2 = v_0 \oplus 2^{2j+1}$ and $v_3 = v_0 \oplus (2^{2j} + 2^{2j+1})$. As illustrated in Fig. 7, nodes v_0 and v_3 send $\delta_{v_0}^{2j}$ and $\delta_{v_3}^{2j}$ to nodes v_1 and v_2, respectively. Then v_1 and v_2 perform the operation $\mathrm{OPER}(v_1, 2j; \delta_{v_0}^{2j}, \delta_{v_1}^{2j})$ and $\mathrm{OPER}(v_2, 2j; \delta_{v_2}^{2j}, \delta_{v_3}^{2j})$, respectively. After the operations, nodes v_1 and v_2 send $\delta_{v_0}^{2j+1}$ and $\delta_{v_3}^{2j+1}$ to v_2 and v_1. Thus nodes v_1 and v_2 can perform the operations $\mathrm{OPER}(v_1, 2j+1; \delta_{v_1}^{2j+1}, \delta_{v_3}^{2j+1})$ and $\mathrm{OPER}(v_2, 2j+1; \delta_{v_0}^{2j+1}, \delta_{v_2}^{2j+1})$, respectively. Finally, nodes v_1

and v_2 send $\delta_{v_3}^{2(j+1)}$ and $\delta_{v_0}^{2(j+1)}$ back to v_3 and v_0, respectively. Two iterations in the ASCEND algorithm can therefore be done by using four communication steps. Observing the ASCEND/DESCEND algorithms under the hypercube with the half duplex mode, we see that both use 4 communication steps within two iterations.

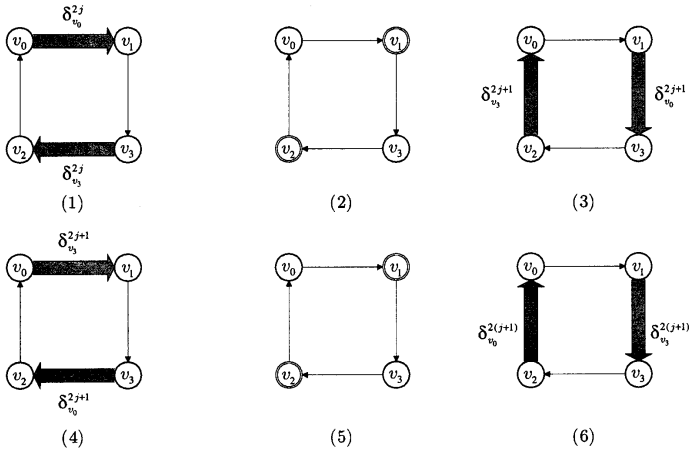

Fig. 7. Iteration $2j$ and $2j + 1$ of the ASCEND algorithm in the UHC.

Algorithm 4. *ASCEND for the even dimensional UHC*

for $j = 0$ to $\frac{k}{2} - 1$ do

for each node m, $0 \le m < n$, do in parallel

if port $2j$ is an out-port then

send δ_m^{2j} to node $m \oplus 2^{2j}$

receive $\delta^{2j+1}_{m\oplus(2^{2j}+2^{2j+1})}$ from node $m \oplus 2^{2j+1}$

send $\delta^{2j+1}_{m\oplus(2^{2j}+2^{2j+1})}$ to node $m \oplus 2^{2j}$

receive $\delta_m^{2(j+1)}$ from node $m \oplus 2^{2j+1}$

else

receive $\delta^{2j}_{m\oplus 2^{2j}}$ from node $m \oplus 2^{2j}$

$\mathrm{OPER}(m, 2j; \delta_m^{2j}, \delta^{2j}_{m\oplus 2^{2j}})$

send $\delta^{2j+1}_{m\oplus 2^{2j}}$ to node $m \oplus 2^{2j+1}$

receive $\delta^{2j+1}_{m\oplus 2^{2j+1}}$ from node $m \oplus 2^{2j}$

$\mathrm{OPER}(m, 2j+1; \delta_m^{2j+1}, \delta^{2j+1}_{m\oplus 2^{2j+1}})$

send $\delta^{2(j+1)}_{m\oplus 2^{2j+1}}$ to node $m \oplus 2^{2j+1}$

end

end

If simultaneous *send* and *receive* operations are allowed, we can transmit the messages within 3 steps. Each node m performs the three operations $OPER(m,2_j;\delta_m^{2j}, \delta_{m\oplus 2^{2j}}^{2j}), OPER(m\oplus 2^{2j+1},2_j;\delta_{m\oplus 2^{2j+1}}^{2j},\delta_{m\oplus(2^{2j}+2^{2j+1})}^{2j})$, and OPER $(m,2j+1;\delta_m^{2j+1},\delta_{m\oplus 2^{2j+1}}^{2j+1})$. This method needs three communication steps and three operations with two iterations.

From the above description, we find that the BT_1 algorithm described in the previous section is one example of the DESCEND algorithm.

In the following we use bitonic sorting to illustrate the ASCEND/DESCEND algorithms. Bitonic sorting is a combination of several DESCEND algorithm calls. Given a sequence of 2^k data elements, which are stored in $T[0]$, $T[1]$, …, $T[2^k - 1]$, this algorithm is

Algorithm 5. *Bitonic Sorting*

```
for i = 0 to k – 1 do
  for j = i down to 0 do
    for each m, 0 ≤ m ≤ n – 1, do in parallel
      if the jth bit of m is 0 then
        if T[m] > T[m ⊕ 2^j] then
          exchange T[m] and T[m ⊕ 2^j]
      else
        if T[m] < T[m ⊕ 2^j] then
          exchange T[m] and T[m ⊕ 2^j]
end
```

In the original ASCEND/DESCEND algorithms, the number of iterations is even, while that of the inner loop of bitonic sorting may be either odd or even. If the number of iterations is odd, the DESCEND algorithm cannot be directly applied. As a result, we have to add an iteration to the inner loop when the number of iterations is odd, i.e., when i is even in Algorithm 5. This added iteration is used only for data transmission, and thus, we do not perform the comparison and exchange operations in the iteration. Our resulting bitonic sorting for the UHC is as follows.

Algorithm 6. *Bitonic Sorting for the UHC*

```
for i = 0 to k – 1 do
  for j = (⌊i/2⌋) to 0 do
    for each m, 0 ≤ m ≤ n – 1, do in parallel
      if port 2j + 1 is an out-port then
        send δ_m to node m ⊕ 2^(2j+1)
          receive δ_(m⊕(2^2j + 2^(2j+1))) from node m ⊕ 2^2j
        send δ_(m⊕(2^2j + 2^(2j+1))) to node m⊕2^(2j+1)
        receive δ_m from node m⊕2^2j
      else
        receive δ_(m⊕2^(2j+1)) from node m⊕2^(2j+1)
          if (i is odd or j ≠ (⌊i/2⌋) then
           if ((the (2j +1)th bit of m is 0) and (δ_m > δ_(m⊕2^(2j+1)))) or
           ((the (2j +1)th bit of m is 1) and (δ_m < δ_(m⊕2^(2j+1)))) then
              exchange δ_m and δ_(m⊕2^(2j+1))
```

send $\delta_{m\oplus 2^{2j+1}}$ to node $m\oplus 2^{2j}$
receive $\delta_{m\oplus 2^{j}}$ from node $m \oplus 2^{2j+1}$
if ((the $(2j)$th bit of m is 0) and ($\delta_m > \delta_{m\oplus 2^{2j}}$)) or
((the $(2j)$th bit of m is 1) and ($\delta_m < \delta_{m\oplus 2^{2j}}$) then
exchange δ_m and $\delta_{m\oplus 2^{2j}}$
send $\delta_{m\oplus 2j}$ to node $m \oplus 2j$
end
end

8. CONCLUSIONS

In this paper we present three kinds of broadcasting tree for the even dimensional uni-directional hypercube (UHC) and some applications. In the constant evaluation model, our all-port broadcasting tree BT_2, which requires $n + 1$ steps, is optimal. The best one of our one-port broadcasting trees, BT_3, needs $\frac{4}{3}(n - n \bmod 6) + \frac{3}{2}(n \bmod 6)$ steps. Our algorithms can be easily applied to an odd dimensional UHC. We also propose an all-port fault-tolerant broadcasting tree whose height is $\frac{3}{2}n + \frac{n \bmod 4}{2}$. Lastly, we show that the ASCEND/DESCEND algorithms can be implemented in the UHC with the same time complexity as the hypercube under the half duplex mode.

Given any node v in n-UHC, where n is even, let $V' = \{u|$ the distance between u and v is $n + 1.\}$. Then, obviously, $|V'| \geq 2$ if $n \geq 4$. Hence, the lower bound on the number of steps needed in one-port broadcasting is greater than or equal to $n + 2$, i.e., $d + 1$, where d is the diameter of the UHC. As a result, we do not know if our best one-port broadcasting algorithm is optimal. Since the height of n-BT_2 is only $n + 1$, can the height $\frac{3}{2}n + \frac{n \bmod 4}{2}$ of n-ADST be improved? Our ADST requires that some processors are cabable of multi-port communication. We are trying to figure out if it is possible to find a scheduling discipline which allows one-port fault-tolerant broadcasting in our n-ADST. These problems are worthy of further investigation.

Many interconnection networks have been proposed over the years, such as star graphs [1, 7, 14] and de Bruijn graphs [9, 18], and they are claimed to have shorter diameter or higher fault tolerance than hypercubes. One drawback with them, however, is the difficulty of designing parallel programs on these networks. That is why they cannot compete with the hypercube. ASCEND/DESCEND algorithms represent the solution of many practical problems, and it is very easy to implement ASCEND/DESCEND algorithms on hypercubes. On the other hand, some computer engineers think the high degree of each node in a hypercube makes it difficult to construct larger networks. With a degree that is only half that of the hypercube's, and the ease of implementing ASCEND/DESCEND algorithms, the UHC is an attractive alternative to the hypercube.

REFERENCES

1. S. Akers and B. Krishnamurthy, "A group-theoretic model for symmetric interconnection networks," *IEEE Transactions on Computers*, Vol. 38, 1989, pp.

555-566.
2. B. Alspach, "Cayley graphs with optimal fault tolerance," *IEEE Transactions on Computers*, Vol. 41, 1992, pp. 1337-1339.
3. J. A. Bondy and U. S. R. Murty, *Graph Theory With Applications*, the Macmillan Press Ltd., 1976.
4. F. Borgonovo and E. Cadorin, "HR^4-NET: A hierarchical random-routing reliable and reconfigurable network for metropolitan area," *IEEE INFOCOM*, 1987, pp. 320-326.
5. T. F. Chan and Y. Saad, "Multigrid algorithms on the hypercube multiprocessor," *IEEE Transactions on Computers*, Vol. C-35, 1986, pp. 969-977.
6. C. H. Chou and D. H. C. Du, "Uni-directional hypercubes," in *Proceedings of Supercomputing* '90, 1990, pp. 254-263.
7. K. Day and A. Triathi, "A comparative study of topological properties of hypercubes and star graphs," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 5, 1994, pp. 31-38.
8. F. Dehne, Q. T. Pham, and I. Stojmenovic, "Optimal visiblility algorithms for binary images on the hypercube," *International Journal of Parallel Programming*, Vol. 19, 1990, pp. 213-224.
9. D. Du and F. K Hwang, "Generalized de Bruijn digraphs," *Networks*, Vol. 18, 1988, pp. 27-38.
10. J. Edmonds, "Edge-disjoint branchings, combinatorial algorithms," *Combinatorial Algorithms*, R. Rustin, ed., New York, Algorithmic Press, 1972.
11. P. Fraigniaud, "Asymptotically optimal broadcasting and gossiping in faulty hypercube multicomputers," *IEEE Transactions on Computer*s, Vol. 41, 1992, pp. 1410-1419.
12. P. Fraigniaud, "Complexity analysis of broadcasting in hypercubes with restricted communication capabilities," *Journal of Parallel and Distributed Computing*, Vol. 16, 1992, pp. 15-26.
13. M. Gerla, "Tree-net, a multi-level fiber optics MAN," *IEEE INFOCOM*, 1988, pp. 363-372.
14. S. C. Hu and C. B. Yang, "Fault tolerance on star graphs," *International Journal of Foundations of Computer Science*, Vol. 8, 1997, pp. 127-142.
15. S. L. Johnsson, "Communication efficient basic linear algebra computers on hypercube architectures," *Journal of Parallel and Distributed Computing*, Vol. 4, 1987, pp. 133-172.
16. S. L. Johnsson and C. T. Ho, "Optimum broadcasting and personalized communication in hypercubes," *IEEE Transactions on Computers*, Vol. 38, 1989, pp. 1249-1268.
17. H. J. Kim and J. G. Lee, "Partial sum problem mapping into a hypercube," *Information Processing Letters*, Vol. 36, 1990, pp. 221-224.
18. J. W. Mao and C. B. Yang, "Shortest path routing and fault-tolerant routing on de Bruijn networks," *Networks*, Vol. 35, 2000, pp. 207-215.
19. N. Maxemchuk, "The Manhattan street network," in *Proceeding of GLOBECOM* '85, 1985, pp. 255-261.
20. N. Maxemchuk, "Regular and mesh topologies in local and metropolitan area networks," *AT&T Tech. Journal*, Vol. 64, 1985, pp. 1659-1686.
21. NCUBE Corp., Beavertos Oregon, *NCUBE Handbook*, 1986.

22. F. P. Preparata and J. Vuillemin, "The cube-connected cycles: A versatile network for parallel computation," *Communications of the ACM*, Vol. 24, 1981, pp. 300-309.
23. S. H. Sheu and C. B. Yang, "Multicast algorithms for hypercube multiprocessors," *Journal of Parallel and Distributed Computing*, Vol. 61, 2001, pp. 137-149.
24. Q. Stout and B. Wagar, "Internsive hypercube communication, prearranged communication in link-bound machines," *Journal of Parallel and Distributed Computing*, Vol. 10, 1990, pp. 161-181.
25. S. T. Tan and D. H. C. Du, "Embedded unidirectional incomplete hypercubes for optical networks," *IEEE Transactions on Communication*, Vol. 41, 1993, pp. 1284-1298.
26. J. Tuazon, "Caltech/JPL Mark II hypercube concurrent processor," *IEEE ICCP*, 1985, pp. 666-672.

Huang-Ming Huang (黃皇銘) received his BS degree in Mechanical Engineering and MS degree in Applied Mathematics from National Sun Yat-Sen University, Kaohsiung, Taiwan, in 1992 and 1994, respectively. He worked in Institute for Information Industry in Taiwan from 1994 to 2000. He is now studying in USA.

Chang-Biau Yang (楊昌彪) received the BS degree in Electronic Engineering from National Chiao Tung University, Hsinchu, Taiwan, in 1982 and the MS degree in Computer Science from National Tsing Hua University, Hsinchu, Taiwan, in 1984. Then, he obtained his PhD degree in Computer Science from National Tsing Hua University in 1988. He is currently a professor and the chairman of the Department of Computer Science and Engineering, National Sun Yat-Sen University. His research interests include computer algorithms, interconnection networks, data compression and bioinformatics.

Kuo-Tsung Tseng (曾國尊) received his BS degree in Applied Mathematics from National Sun Yat-Sen University, Kaohsiung, Taiwan, in 1997. He is now a PhD candidate in Computer Science and Engineering at National Sun Yat-Sen University, Kaohsiung, Taiwan. His current research interests are computer algorithms and wireless networks.