The Thirteen IASTED International Conference on Parallel and Distributed Computing and Systems, Anaheim, USA, August 21-24, 2001, pp.266-271

Efficient Collective Communications in Dual-cube

Yamin Li, Shietung Peng
Department of Computer Science
Hosei University
Tokyo 184-8584 Japan

Wanming Chu
Department of Computer Hardware
University of Aizu
Aizu-Wakamatsu 965-8580 Japan

ABSTRACT
The binary hypercube, or *n*-cube, has been widely used as the interconnection network in parallel computers. However, the major drawback of the hypercube is the increase in the number of communication links for each node with the increase in the total number of nodes in the system. This paper introduces a new interconnection network for large-scale parallel computers called dual-cube. This network mitigates the problem of increasing number of links in the large-scale hypercube network while keeps most of the topological properties of the hypercube network. Design of efficient routing algorithms for collective communications is the key issue for any interconnection networks. In this paper, we show that collective communications can be done efficiently in dual-cube.

KEY WORDS
Interconnection networks, hypercube, collective communication, broadcast and personalized communication

1. Introduction

The binary hypercube has been widely used as the interconnection network in a wide variety of parallel systems such as Intel iPSC, the nCUBE [6], the Connection Machine CM-2 [18], and SGI Origin 2000 [17]. A hypercube network of dimension n contains up to 2^n nodes and has n edges per node. If unique n-bit binary addresses are assigned to the nodes of the hypercube, then an edge connects two nodes if and only if their binary addresses differ in a single bit. Because of its elegant topological properties and the ability to emulate a wide variety of other frequently used networks, the hypercube has been one of the most popular interconnection networks for parallel computer/communication systems.

However, the conventional hypercube has a major shortage, that is, the number of edges per node in a system increases logarithmically as the total number of nodes in the system increases. Since the number of links is limited to eight per node with current IC technology, the total number of nodes in a hypercube parallel computer is restricted to several hundreds. Therefore, it is interesting to develop an interconnection network which keeps most of topological properties of the hypercube, and have more nodes in the system than the hypercube with the same number of edges per node.

Several variations of the hypercube have been proposed in the literature. Some variations focused on the reduction of diameter of the hypercube, such as folded hypercube [1] and crossed cube [3]; some focused on the reduction of the number of edges of the hypercube, such as cube-connected cycles [16] and reduced hypercube [19]; and some focused on the both, like hierarchical cubic network [5]. Generally, the variations of the hypercube that reduce the diameter, e.g. crossed cube and hierarchical cube, will not satisfy the following key property in the hypercube: each node can be represented by a unique binary number such that two nodes are connected by an edge only if the two binary numbers differ in one bit. This key property is at the core of many algorithmic designs for efficient routing and communication.

In this paper, we propose a new interconnection network, called *dual-cube*. The dual-cube shares the desired properties of the hypercube (e.g., the key property of the hypercube mentioned above), and increases tremendously the total number of nodes in the system compared with the hypercube of the same node degree. The size of the dual-cube can be as large as thirty thousands with up to eight links per node. It is practically important to refine the hypercube networks such that the size of the network can be increased while the number of the links per node is limited by the technology. Design of efficient routing algorithms for collective communications is the key issue for any interconnection networks. In this paper, after introducing the architecture of the dual-cube, we show that collective communications can be done efficiently in dual-cube.

2. Dual-cube Interconnection Network

An r-connected dual-cube F_r [12] is an undirected graph on the node set $\{0,1\}^{2r-1}$ such that there is an edge between two nodes $u = (u_{2r-1} \dots u_1)$ and $v = (v_{2r-1} \dots v_1)$ in F_r if and only if the following conditions are satisfied:

(1) u and v differ exactly in one bit position i.
(2) if $1 \le i \le r-1$ then $u_{2r-1} = v_{2r-1} = 0$.
(3) if $r \le i \le 2r-2$ then $u_{2r-1} = v_{2r-1} = 1$.

Intuitively, the set of the nodes u of form $(0u_{2r-2} \dots u_r * \dots *)$, where $*$ means "don't care", constitutes an $(r-1)$-dimensional hypercube. We call these hypercubes *clusters* of class 0. Similarly, the set of the nodes u of form $(1 * \dots * u_{r-1} \dots u_1)$ constitutes an $(r-1)$-dimensional hypercube, and we call them clusters of class 1. The edge connects two nodes in two clusters of distinct class is called *cross-edge*. In other word, $\langle u, v \rangle$ is a cross-

Table 1. Hypercube v.s. Dual-cube

Network	Degree	Diam.	Cost	Avg. distance	Bisec. width	# of edges
Hypercube	n	n	n^2	$n/2$	$2^n/2$	$2^n n$
Dual-cube	$(n+1)/2$	$n+1$	$(n+1)^2/2$	$n/2+1-1/2^{(n-1)/2}$	$2^n/4$	$2^n(n+1)/2$

edge if and only if u and v differ at the leftmost bit position only.

We divide the binary representation of a node into three parts: Part I is the rightmost $r-1$ bits, part II is the next $r-1$ bits, and part III is the leftmost bit. Part III is a *class indicator*. For the nodes in a cluster of class 0 (class 1), part I (part II) is called *node ID* and part II (part I) is called *cluster ID*. The cluster contains node u is denoted as C_u. For any two nodes u and v in F_r, $C_u = C_v$ if and only if u and v are in the same cluster.

Table 1 summarizes the degree, diameter, cost, average node distance, and bisection width of the hypercube and the dual-cube network, assuming that the two networks have the same number of nodes which is 2^n, where n is an odd integer [12].

3. Model of Communication

Collective communication is the key issue in message-passing parallel computers or networks [2] [7] [8] [11] [13] [15]. Collective communication is required in load balancing, event synchronization, and data exchange. Based on the number of sending and receiving processors, these communications can be classified into one-to-many, one-to-all, many-to-many, and all-to-all. The nature of the messages to be sent can be classified as personalized or non-personalized (multicast or broadcast). The all-to-all personalized communication (total exchange) is at the heart of numerical applications. In this paper, we will present efficient algorithms for collective communication in dual-cube.

An important metric used to evaluate efficiency of communication is *communication latency* or *transmission time*. The transmission time depends on many factors such as contentions, switching techniques, network topologies etc. Therefore, we first define the communication model used in this paper.

We assume that the communication links are bidirectional; that is, two directly-connected processors can send messages to each other simultaneously. We also assume the processor-bounded model (one-port model) in which each processor can use only one link at a time. The port model of a network system refers to the number of internal channels at each node. In order to reduce the complexity of communication hardware, many systems support one-port communication architecture, in which each node can access the network through a single input port and a single output port. Many existing networks use one-port architecture. We also assume the linear cost model [4] in which the transfer time for a message is linearly proportional to the length of the message.

We assume cut-through routing strategy [9], in which each message is serialized into a sequence of pieces and is sent in a pipeline fashion. The router can start forwarding the header and the following data bytes as soon as routing decisions have been made and the output buffer is free. Since the message can cut through to the input of the next router before the complete message has been received at the current router, this switching technique is referred as virtual cut-through switching. For long message, the pipelining effect of cut-through routing reduces the effect of path length on network. The predominant wormhole routing [14] is just a special form of the cut-through routing scheme. Under the cut-through routing scheme and in the absence of blocking, the transmission time for a message of length m to be sent to a node of distance d is $t_s + mt_w + dt_h$, where t_s is startup latency, the time required for the system to handle the message at the source and destination nodes, t_w is the per-word transfer time ($1/t_w$ is the bandwidth of the communication links), t_h is the per-hop time, the time to switch an intermediate node. Through this paper, we will use the above formula for estimating the communication time of the proposed routing algorithms.

In order to simplify the description of the routing algorithms, we adopt the following notation. The r neighbor nodes of s, $s^{(i)}, 1 \le i \le r$, are denoted as follows. Assume s is of class 0 and $s = (0a_{2r-2}\dots a_1)$, then $s^{(i)} = (0a_{2r-2}\dots a_{i+1}\overline{a}_i a_{i-1}\dots a_1)$, where $1 \le i \le r-1$, and $s^{(r)} = (1a_{2r-2}\dots a_1)$. Assume s is of class 1 and $s = (1a_{2r-2}\dots a_1)$, then, $s^{(i)} = (1a_{2r-2}\dots a_{j+1}\overline{a}_j a_{j-1}\dots a_1)$, where $r \le j \le 2r-2$ and $i = j-(r-1)$, and $s^{(r)} = (0a_{2r-2}\dots a_1)$. Each node in an F_r is identified by its unique $(2r-1)$-bit address, *id*. Each *id* contains three parts: *class_id* (1-bit), *cluster_id* ($(r-1)$-bit), and *node_id* ($(r-1)$-bit). That is, $id = \{class\_id,\ cluster\_id,\ node\_id\}$. The bit-position of the *cluster_id* and *node_id* depends on the *class_id*. if $class\_id = 0$ (1) then the *node_id* (*cluster_id*) is the rightmost $r-1$ bits, and the *cluster_id* (*node_id*) is the next (to the left) $r-1$ bits.

4. One-to-All Broadcast and Personalized Communication

In this section, we discuss one-to-all broadcast and one-to-all personalized communication.

Parallel algorithms often require a single processor to send identical data to all other processors or to a subset of them. This operation is known as one-to-all broadcast or one-to-many multicast. In our communication model, a message is not routed in parts along separate paths and communication is allowed on only one link of each pro-

cessor at a time. It can be shown that one-to-all broadcast cannot be performed in less than $(t_s + mt_w)\log p$ time on any architecture, where p is the number of processors [10].

We show an algorithm which performs one-to-all broadcast in dual-cube efficiently. The algorithm for broadcasting from a source s works as follows: Assume that C_s is of class 0 (the case that C_s is of class 1 can be done similarly). The source s first sends the message to $s' = s^{(r)}$ through a cross-edge. Then, s and s' broadcast simultaneously the message to all nodes in C_s and $C_{s'}$ using binomial trees of C_s and $C_{s'}$ with roots s and s', respectively. Next, every node $u \in C_s \setminus \{s\}$ and every node $u' \in C_{s'} \setminus \{s'\}$ send the message to v and v' through a cross-edge, respectively. Finally, every v and v' broadcast the message to all other nodes in C_v and $C_{v'}$. The algorithm is listed below.

Algorithm 1 (One_To_All_Bcast (my_id, r, s, msg))

1. **begin** /* The source of the broadcast is node(s) */
   /* Stage 1: Broadcast to the nodes in C_s and $C_{s^{(r)}}$ */
2. node(s) **send** *msg* to node($s^{(r)}$)
3. **for** $i = 1$ **to** $r-1$ **do**
4. **if** ($my\_cluster\_id = cluster\_id$ of s) AND (the right $r-i$ bits of my_node_id = the right $r-i$ bits of $node\_id$ of s) **then**
5. **send** *msg* to node($my\_id^{(i)}$);
6. **endfor**
   /* Stage 2: Broadcast to nodes in all other clusters */
7. **if** ($my\_cluster\_id =$ the $cluster\_id$ of s) AND ($my\_node\_id \neq node\_id$ of s) **then**
8. **send** *msg* to node($my\_id^{(r)}$);
9. **for** $i = 1$ **to** $r-1$ **do**
10. **if** ($my\_cluster\_id \neq cluster\_id$ of s) AND (the right $r-i$ bits of $my\_node\_id =$ the right $r-i$ bits of $node\_id$ of s) **then**
11. **send** *msg* to node($my\_id^{(i)}$);
12. **endfor**
13. **end.**

From the above algorithm, the broadcasting is completed in $1+(r-1)+1+(r-1) = 2r$ steps. Therefore, the total communication time $T_{one-to-all-b} = (t_s + mt_w)(1 + \log p)$, where $\log p = \log_2 p$, and $p = 2^{2r-1}$.

In one-to-all personalized communication, a single node sends a unique message of size m to every other node. Since the source node transmits m words for each of the other $p-1$ nodes, the lower bound on the communication time of one-to-all personalized communication is $m(p-1)t_w$. This lower bound is independent of the architecture or routing scheme. The routing algorithm for the one-to-all personalized communication in dual-cube is similar to the one-to-all broadcast described above. Initially, the source node, s, contains all the messages. In the first communication step, the node s transfers half of the messages to $s' = s^{(r)}$ through a cross-edge. Then, s and s' simultaneously send the messages to all nodes in C_s and $C_{s'}$ using binomial trees of C_s and $C_{s'}$ with roots s and s', respectively. After each communication step, the sizes of the messages to be sent are reduced by half. At the end of this stage, each node $u \in C_s - \{s\}$ and $u' \in C_{s'} - \{s'\}$ have their message and the messages of all nodes in cluster C_v and $C_{v'}$, where $v = u^{(r)}$ and $v' = (u')^{(r)}$. At the second stage, nodes u and u' send the messages of size 2^{r-1} to v and v' through a cross-edge, respectively. Finally, v and v' send the messages to all nodes in C_v and $C_{v'}$ through the binomial trees of C_v and $C_{v'}$, respectively. The time for the one-to-all personalized communication $T_{one-to-all-p} = rt_s + \sum_{i=1}^{r} 2^{r+i-2}mt_w + rt_s + \sum_{i=1}^{r} 2^{i-1}mt_w = 2rt_s + (2^{2r-1}-1)mt_w = (1+\log p)t_s + (p-1)mt_w$.

5. All-to-all Broadcast

All-to-all broadcast is a generalization of one-to-all broadcast in which all nodes simultaneously initiate a broadcast. A node sends the same m-word message to every other node, but different nodes may broadcast different messages. The communication pattern of all-to-all broadcast can be used to perform some other operations, such as reduction and prefix sums.

The lower bound for the communication time of all-to-all broadcast for parallel computers on which a node can communicate on only one of its ports at a time is $(p-1)mt_w$, where p is the number of nodes. This is because each node receives at least $(p-1)m$ words of data, regardless of the architecture or routing scheme.

An efficient way to perform all-to-all broadcast is to perform all p one-to-all broadcasts simultaneously so that all messages traversing the same path at the same time are concatenated into a single message whose size is the sum of the sizes of individual messages.

The algorithm for all-to-all broadcast (see algorithm2) in dual-cube can be described in three stages. In the first stage, the broadcasts are done inside each cluster. In the second stage, each node in a cluster of class 1 (0) sends the identical message to a node in a cluster of class 0 (1), and then, the received message is broadcasted inside the cluster. After this stage, each node has messages from all other nodes except those in the different clusters of the same class. Finally, in the last stage, each node gets the messages of the nodes in other clusters of the same classes from the neighbor through the cross edge.

Algorithm 2 (All_To_All_Bcast ($my\_id, M_{my\_id}, r, result$))

1. **begin** /* *result*, empty initially, contains all $(p-1)$ messages at the end */
   /* Stage 1: Broadcast inside the cluster */
2. **for** $i = 1$ **to** $r-1$ **do**
3. $partner\_id = (my\_class\_id, my\_cluster\_id,$ my_node_id XOR $2^{i-1})$;
4. **send** M_{my_id} to *partner*;
5. **receive** $M_{partner\_id}$ from *partner*;
6. $M_{my\_id} = M_{my\_id} \cup M_{partner\_id}$;
7. **endfor**
   /* Stage 2: Broadcast to clusters of the other class */
8. $partner\_id = (my\_class\_id$ XOR $1, my\_cluster\_id,$ $my\_node\_id)$;

9. **send** M_{my_id} to *partner*;
10. **receive** $M_{partner\_id}$ from *partner*;
11. $result = M_{my\_id}$;
12. $temp = M_{partner\_id}$;
13. $M_{my\_id} = M_{partner\_id}$;
14. **for** $i = 1$ **to** $r-1$ **do**
15. $partner\_id = (my\_class\_id, my\_cluster\_id,$ my_node_id XOR $2^{i-1})$;
16. **send** M_{my_id} to *partner*;
17. **receive** $M_{partner\_id}$ from *partner*;
18. $M_{my\_id} = M_{my\_id} \cup M_{partner\_id}$;
19. **endfor**
20. $result = result \cup M_{my\_id}$;
21. $M_{my\_id} = M_{my\_id} \setminus temp$;
/* Stage 3: Include the messages from other clusters of the same class */
22. $partner\_id = (my\_class\_id$ XOR $1, my\_cluster\_id,$ $my\_node\_id)$;
23. **send** M_{my_id} to *partner*;
24. **receive** $M_{partner\_id}$ from *partner*;
25. $result = result \cup M_{partner\_id}$;
26. **end.**

The time it takes to complete the 1st stage is $T_1 = \sum_{i=1}^{r-1}(t_s + 2^{i-1}mt_w) = (r-1)t_s + (2^{r-1}-1)mt_w$. The time it takes to complete the 2nd stage is $T_2 = t_s + 2^{r-1}mt_w + \sum_{i=1}^{r-1}(t_s + 2^{r+i-2}mt_w) = rt_s + 2^{2r-2}mt_w$. The time it takes to complete the 3rd stage is $T_3 = t_s + (2^{2r-2} - 2^{r-1})mt_w$. Therefore, the total time to complete the all-to-all broadcast is $T_{all-to-all-b} = 2rt_s + (2^{2r-1}-1)mt_w = (1+\log p)t_s + (p-1)mt_w$.

6. All-to-All Personalized Communication

In all-to-all personalized communication, each node sends a distinct message of size m to every other node. The total number of messages is $p(p-1)$. Because the number of links in dual-cube is just about half compared to the corresponding traditional hypercube, we must find a way to perform all-to-all personalized communication with as shorter time as possible. We will describe sending rule, routing strategy, and time analysis for the all-to-all personalized communication in dual-cube in the following sub-sections.

6.1 All-to-All Personalized Sending Rule

We adopt the strategy that, at one communication step, every node sends one message to its destination simultaneously. There are totally $p-1$ communication steps. Similar to all-to-all broadcast, we divide these $p-1$ steps into three stages. In the first stage, each node sends $2^{r-1}-1$ messages to those nodes which are located in the same cluster. This can be done in the same manner as the hypercube does. In the second stage, each node sends messages to those clusters located in different classes. The total number of messages one node sends in this stage is $2^{r-1} \times 2^{r-1}$ because there are 2^{r-1} clusters of size 2^{r-1}. In the last stage, we route messages to the different clusters with the same class as the sender. The number of messages one node sends in this stage is $(2^{r-1}-1) \times 2^{r-1}$. The algorithm is described below. The procedure **Routing (***r*, *my_id*, *partner***)** is given in the next subsection (Algorithm 4).

Algorithm 3 (All_To_All_Pers (*r*, *my_id*, $M_{my\_id,*}$))

1. **begin** /* $M_{s,t}$ *is the message from node(s) to node(t)* */
/* stage 1: within cluster */
2. **for** $i = 1$ **to** $2^{r-1}-1$ **do**
3. $partner = (my\_class\_id, my\_cluster\_id,$ my_node_id XOR $i)$;
4. **Routing (***r*, *my_id*, *partner***)** and send $M_{my\_id,partner}$ to *partner*;
5. **endfor**
/* stage 2: to clusters of different classes */
6. **for** $i = 0$ **to** $2^{r-1}-1$ **do**
7. **for** $j = 0$ **to** $2^{r-1}-1$ **do**
8. $partner = (my\_class\_id$ XOR $1,$ $my\_cluster\_id$ XOR j, my_node_id XOR $i)$;
9. **Routing (***r*, *my_id*, *partner***)** and send $M_{my\_id,partner}$ to *partner*;
10. **endfor**
11. **endfor**
/* stage 3: to different clusters of the same classes */
12. **for** $i = 1$ **to** $2^{r-1}-1$ **do**
13. **for** $j = 0$ **to** $2^{r-1}-1$ **do**
14. $partner = (my\_class\_id, my\_cluster\_id$ XOR $i,$ my_node_id XOR $j)$;
15. **Routing (***r*, *my_id*, *partner***)** and send $M_{my\_id,partner}$ to *partner*;
16. **endfor**
17. **endfor**
18. **end**

In the first stage, node(*my_id*) and node(*partner*) exchange data each other, where $partner = my\_id$ XOR i. Notice that the node IDs are located in different positions for different classes.

In the second stage, it sends the data in the $(i \times 2^{r-1} + j)$th step to the node of different class with the cluster ID = *my_cluster_id* XOR j, and the node ID = *my_node_id* XOR i. It is no longer true that two nodes exchange data each other. For example, in the $(i = 0, j = 1)$th step, node 00000 sends data to node 10100, but receives data from 10001 (node 10100 sends data to node 00101). Actually, once we decided the rule for sending, a node can only receive data arriving it. The reason why exchanging data does not work is that the message is sent from one class to the other class while the field meaning of the addresses is different. We can say that if *my_cluster_id* = *my_node_id* for both receiver and sender, they exchange data each other. We will introduce a congestion-free schedule by following this sending rule.

In the third stage, all pairs exchange data again. Node(*my_id*) exchanges data with the node with the cluster ID = *my_cluster_id* XOR i, and the node ID = *my_node_id* XOR j.

Each node sends $2^{r-1}-1$ messages in the first stage, $2^{r-1} \times 2^{r-1}$ messages in the second stage, and $(2^{r-1}-1) \times$

2^{r-1} messages in the last stage. The total number of messages each node sends in $p-1$ steps is $2^{2r-1}-1=p-1$, one for each step.

6.2 All-to-All Personalized Routing

The basic building block of the dual-cube is the $(r-1)$-cube. All-to-all personalized routing in the $(r-1)$-cube is simple. We first choose the destination node by the rule described in the previous sub-section. For the first stage, source node $s=$ node(*my_id*) exchanges data with destination node $t=$ node(*my_id* XOR *i*) in the *i*th step. The Hamming distance between s and t is s XOR $t=i$, that is, a message travels from s to t must pass through at least l links where l equals the number of non-zero bits in the binary representation of i. For example, in a 4-link dual cube, there are 2^7, or 128 nodes, forming 16 clusters with each has 8 nodes. The basic building block is a 3-cube. For routing within 3-cube, it takes 7 steps. In step 1 (001 in binary representation), step 2 (010), or step 4 (100) , a message traveling through one link will arrive destination; step 3, 5, or 6 travels 2 links; step 7 travels 3 links, node 0 to node 7 for instance. There should be more than one solution for a message routing from source to destination. Through this paper, we follow the *ascending routing* strategy, by which the least significant non-zero bit of i is chosen as the first dimension for routing, and so on.

For the second and the third stages, the routing is more complicate than the first stage because these stages use cross edges and even other clusters for just traveling through. The following algorithm shows the all-to-all personalized routing in dual-cube.

Algorithm 4 (Routing (r, s, t))

1. **begin**
2. *diffBits* $=s$ XOR t;
3. *current_id* $=s$; /* current node = s */
4. *route* $=$ *current_id*;
5. **case 1:** $C_s=C_t$:
6. routingCluster (); /* internal routing */
7. **case 2:** *class_id* of $s\neq$ *class_id* of t:
8. routingCluster (); /* routing in C_s */
9. routingCrossEdge (); /* go through cross edge */
10. routingCluster (); /* routing in C_t */
11. **case 3:** $C_s\neq C_t$ AND *class_id* of $s=$ *class_id* of t:
12. routingCrossEdge (); /* go through cross edge */
13. routingCluster (); /* routing in $C_{s'}$ */
14. routingCrossEdge (); /* go through cross edge */
15. routingCluster (); /* routing in C_t */
16. **end**
17. **Procedure** routingCluster ()
18. **begin**
19. **for** $i=0$ **to** $r-1$ **do**
20. *go* $=$ the *i*th bit of *diffBits*
21. **if** (*go* $\neq 0$) **then**
22. *current_id* $=$ *current_id* XOR *go*;
23. *route* $=$ route **concat** *current_id*;
24. **endfor**
25. **end**
26. **Procedure** routingCrossEdge ()
27. **begin**
28. *current_id* $=$ *current_id*$^{(r)}$;
29. *route* $=$ *route* **concat** *current_id*;
30. **end**

Routing in the first stage is simple (line 6). In the second stage, each node sends a distinct message to every other node in the other classes. It uses cross edges. In the first step, each node directly send a message through a cross edge to the neighborhood node in a cluster of the other class, then to every other node in that cluster. The sending order is determined with the sending rule described in the previous sub-section. At this point, $1/2^{r-1}$ work finished because there are 2^{r-1} such clusters. For other clusters, the source node needs to route messages within its own cluster first, then go through a cross edge, and finally route within the destination cluster (line 8–10). Routing within a cluster is the same as the first stage.

In the third stage (line 12–15), a message travels through a cluster of different class to a destination cluster whose class is the same as that of source node.

6.3 Timing Analysis of All-to-All Personalized Communication

The time it takes to send messages to the nodes in the same clusters (stage 1) is $T_1=\sum_{i=1}^{r-1}(t_s+mt_w+it_h)\binom{r-1}{i}=(2^{r-1}-1)(t_s+mt_w)+gt_h$, where $g=\frac{1}{2}(r-1)2^{r-1}$, the total distance for a node sends $2^{r-1}-1$ messages to other $2^{r-1}-1$ nodes within the same cluster. Next, consider sending messages to the nodes located in different classes. There are 2^{r-1} such clusters with each has 2^{r-1} nodes. Each message travels through a cross edge, the distance is $2^{r-1}\times 2^{r-1}=2^{2r-2}$. The distance of routing within 2^{r-1} clusters is $2^{r-1}g$, and, before going through cross edges, it needs to route in the cluster of the source node. This distance is also $2^{r-1}g$. Therefore, the total time in this stage is $T_2=2^{2r-2}(t_s+mt_w)+(2^{2r-2}+2^rg)t_h$

For sending messages to the different clusters of the same class, the time T_3 is shown below. The sending procedure consists of four parts: going out through a cross edge, routing within a cluster of the different class (intermediate cluster), going out again to the destination clusters, and routing within the destination clusters. Notice that there are only $2^{r-1}-1$ clusters which a node sends messages to and each message will travel through two cross edges. The item $g(2^{r-1}-1)$ in T_3 equation is the distance for routing within $2^{r-1}-1$ destination clusters, $g2^{r-1}$ is the distance for routing within 2^{r-1} intermediate clusters whose class is different from the class of the source node. $T_3=2^{r-1}(2^{r-1}-1)(t_s+mt_w)+((2^r(2^{r-1}-1)+(2^r-1)g)t_h$

The total time to complete the all-to-all personalized communication is the sum of T_1, T_2, and T_3. $T_{all-to-all-p}=(2^{2r-1}-1)(t_s+mt_w)+((r+\frac{1}{2})2^{2r-1}-2^r)\,t_h=(p-1)(t_s+mt_w)+(p(\frac{1}{2}\log p+1)-\sqrt{2p})\,t_h$ where $p=2^{2r-1}$, the total number of nodes. Comparing to the time of traditional

Table 2. Communication Times: Hypercube v.s. Dual-cube

Comm. Pattern	Hypercube	Dual-cube
$T_{one-to-all-b}$	$\log p(t_s + mt_w)$	$(1+\log p)(t_s + mt_w)$
$T_{one-to-all-p}$	$(\log p)t_s + (p-1)mt_w$	$(1+\log p)t_s + (p-1)mt_w$
$T_{all-to-all-b}$	$(\log p)t_s + (p-1)mt_w$	$(1+\log p)t_s + (p-1)mt_w$
$T_{all-to-all-p}$	$(p-1)(t_s + mt_w) + (p/2(\log p))t_h$	$(p-1)(t_s + mt_w) + (p/2(2+\log p) - \sqrt{2p})t_h$

hypercube for performing all-to-all personalized communication, which is $(p-1)(t_s + mt_w) + \frac{1}{2}p \log p\ t_h$, our result is quite satisfactory because the dual-cube uses about half links in comparison to a corresponding hypercube. From the total time equation, we know that the average distance of the dual-cube is $((r+\frac{1}{2})2^{2r-1} - 2^r)/2^{2r-1} = r + \frac{1}{2} - \frac{1}{2^{r-1}}$ while the average distance of $(2r-1)$-cube is $r - \frac{1}{2}$.

The communication times of our routing algorithms for collective communication are summarized and compared to that of the hypercube in Tab. 2.

7. Conclusions

In this paper, we proposed a new interconnection network, dual-cube, and showed efficient routing algorithms for collective communication in dual-cube. The proposed dual-cube increases tremendously the total number of nodes in the system compared to the hypercube: If the number of links per node is n, the conventional hypercube can connect up to 2^n nodes, the dual-cube can connect 2^{2n-1} nodes, i.e., 2^{n-1} times large than hypercube. More importantly, the dual-cube keeps almost the desired properties of the hypercube and reduces the cost to about half compared to hypercube. The efficiency of the routing algorithms for all kinds of communication patterns is almost the same as hypercubes with much less links (about half of the number of links in hypercubes).

Recently, much of the community has moved on to lower-dimensional topologies such as mesh and torus. However, the SGI Origin 2000, a fairly recent multiprocessor, does use a hypercube topology so a dual-cube could be of use to industry.

References

[1] A. E. Amawy and S. Latifi. Properties and performance of folded hypercubes. *IEEE Transactions on Parallel and Distributed Systems*, 2(1):31–42, 1991.

[2] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection networks: an engineering approach.* IEEE Computer Society Press, 1997.

[3] Kemal Efe. The crossed cube architecture for parallel computation. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):513–524, Sep. 1992.

[4] A. M. Farley. Minimum-time broadcast networks. *Networks*, 10:59–70, 1980.

[5] K. Ghose and K. R. Desai. Hierarchical cubic networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(4):427–435, April 1995.

[6] J. P. Hayes and T. N. Mudge. Hypercube supercomputers. *Proc. IEEE*, 17(12):1829–1841, Dec. 1989.

[7] C. C. Huang and P. K. McKinley. Communication issues in parallel computing across atm networks. *IEEE Parallel and Distributed Technology*, 1991.

[8] S. L. Johnson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computers*, 38(9):1249–1268, 1989.

[9] P. Kermani and L. Kleinrock. Virtual cut-through: a new communication switching technique. *Computer Networks*, 13:267–286, 1979.

[10] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to parallel computing: design and analysis of algorithms*. Benjamin/Cummings Press, 1994.

[11] Y. Lan, A. H. Esfahanian, and L. M. Ni. Multicast in hypercube multiprocessors. *Journal of Parallel and Distributed Computing*, 16(1):30–41, 1990.

[12] Y. Li and S. Peng. Dual-cubes: a new interconnection network for high-performance computer clusters. In *Proceedings of the 2000 International Computer Symposium, Workshop on Computer Architecture*, pages 51–57, ChiaYi, Taiwan, December 2000.

[13] P. K. McKinley, Y. J. Tsai, and D. Robinson. Collective communication in wormhole-routed massively parallel computers. *IEEE Transactions on Parallel and Distributed Systems*, 7(2):184–190, 1996.

[14] L. Ni and P. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, 26(2):62–76, 1993.

[15] J. G. Peters and M. Syska. Circuit-switched broadcasting in torus networks. *IEEE Transactions on Parallel and Distributed Systems*, 7(3):246–255, 1996.

[16] F. P. Preparata and J. Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Commun. ACM*, 24:300–309, May 1981.

[17] SGI. *Origin2000 Rackmount Owner's Guide, 007-3456-003*. http://techpubs.sgi.com/, 1997.

[18] L. W. Tucker and G. G. Robertson. Architecture and applications of the connection machine. *IEEE Computer*, 21:26–38, August 1988.

[19] S. G. Ziavras. Rh: a versatile family of reduced hypercube interconnection networks. *IEEE Transactions on Parallel and Distributed Systems*, 5(11):1210–1220, November 1994.