The Second International Conference on Software Engineering, Artificial Intelligence, Networking & Parallel/Distributed Computing, Nagoya, Japan, Aug., 2001, pp.422–429

Algorithms of Routing and Matrix Multiplication on Dualcube

Yamin Li and Shietung Peng
Faculty of Computer and Information Sciences
Hosei University
Tokyo 184-8584 Japan
E-mail: {speng,yamin}@k.hosei.ac.jp

Abstract

Dualcube is an interconnection networks that has hypercube-like structure with the capacity to hold much more nodes than the conventional hypercube with the same number of links per node. The motivation of using dualcube as an interconnection network is to mitigate the problem of increasing the number of links in the large-scale hypercube network while keeps most of the topological properties of the hypercube network. In this paper, we focus on the design of efficient algorithms for routing and numerical operations on dualcube such as prefix computation, vector-matrix and matrix-matrix multiplications. Our results show that the routing and the basic numerical computations can be done on dualcube almost as fast as those on hypercube.

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 [4], the Connection Machine CM-2 [9], and SGI Origin 2000 [8]. 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 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.

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 [2]; some focused on the reduction of the number of edges of the hypercube, such as cube-connected cycles [7] and reduced hypercube [10]; and some focused on the both, like hierarchical cubic network [3]. 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 addresses differ in one bit. This key property is at the core of many algorithmic designs for efficient communications and computations.

The 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. The motivation of designing the dualcube as an interconnection network is the following: it keeps most of topological properties of the hypercube including the key property mentioned above, and can contain much more nodes than the hypercube with the same node degree.

The dualcube is a new interconnection network that uses hypercubes as basic components [5]. Each hypercube component is referred to as a *cluster*. Assume that the number of nodes in a cluster is 2^q. In a dualcube, there are two *classes* with each class consisting of 2^q clusters. The total number of nodes is $2^q \times 2^q \times 2$, or 2^{2q+1}. Therefore, the node address has $2q+1$ bits. The leftmost bit is used to indicate the type of the class (class 0 and class 1). For the class 0, the rightmost q bits are used as the node ID within the cluster and the next (to the left) q bits are used as the cluster ID. For the class 1, the rightmost q bits are used as the cluster ID and the next q bits are used as the node ID within the cluster. Each node in a cluster of class 0 (1) has one and only one extra connection to a node in a cluster of class 1 (0). These two node addresses differ only in the leftmost bit position.

Design of efficient algorithms for basic communications and computations is the key issue for any interconnection network. In this paper, we show that one-to-all and all-to-all communications, prefix computation and matrix multi-

plication can be done efficiently on dualcube. The rest of this paper is organized as follows. Section 2 describes the dualcube structure and its topological properties. Section 3 describes the model of communication used in this paper and the communication algorithms for one-to-all personalized communication and all-to-all broadcasting. Section 4 shows the algorithms for prefix computation and vector-matrix multiplication. Section 5 shows the algorithm for matrix-matrix multiplication. Section 6 concludes the paper and presents some future research directions.

2. Dualcube Interconnection Network

An r-connected dualcube F_r is an undirected graph on the node set $\{0,1\}^n$, $n = 2r-1$, such that there is an edge between two nodes $u = (u_{n-1} \dots u_0)$ and $v = (v_{n-1} \dots v_0)$ in F_r if and only if the following conditions are satisfied:

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

Intuitively, the set of the nodes u of form $(0u_{n-2} \dots u_{r-1} * \dots *)$, 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 $(1 * \dots * u_{r-2} \dots u_0)$ 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 the other word, $\langle u, v \rangle$ is a cross-edge if and only if u and v differ at the leftmost bit position only.

Each node in an F_r is identified by its unique $(2r-1)$-bit binary address, ID. Each ID contains three parts: a, b, and c, representing cluster ID ($(r-1)$-bit), node ID ($(r-1)$-bit), and class ID (1-bit), respectively. The bit-position of a and b depends on c. if $c = 0$ (1) then b (a) is the rightmost $r-1$ bits, and a (b) is the next (to the left) $r-1$ bits. The cluster containing 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.

Fig. 1 depicts an F_3 network. c is shown at the top position in the node address. For the nodes of class 0 (class 1), b (a) is shown at the bottom, and a (b) is shown at the middle. Fig. 2 (F_4) shows only those cross-edges connecting to cluster 0 of class 1.

A topology is evaluated in terms of a number of parameters such as degree, diameter, bisection width, cost (defined as the product of the degree and diameter), average distance for any two nodes, regularity, symmetry etc. Existence of efficient algorithms for communications and computations is also an important measure for evaluating networks. The dualcube network has a binary presentation of nodes in which two nodes are connected by an edge only if

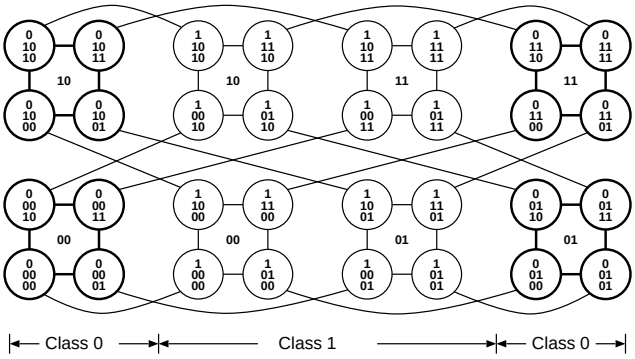

Figure 1. The dualcube F_3

they differ in one bit position, just as in hypercube. This feature is the key for designing efficient routing and communication/computation algorithms on dualcube. Another important feature of the dualcube is that, within a given bound on the number of links per node, say r, the network can have up to 2^{2r-1} nodes, more than the hypercube or the hierarchical cubes can have with the same bound on the node degree. Table 1 summarizes the degree, diameter, cost, average node distance, and bisection width of the hypercube and the dualcube, assuming that the two networks have the same number of nodes which is 2^n, where $n = 2r-1$ is an odd integer. The dualcube shows a significant gain in the cost of the network.

We adopt the following notation throughout this paper. The r neighbor nodes of s, $s^{(i)}$, $0 \le i \le r-1$, are denoted as follows. Assume s is of class 0 and $s = (0 s_{n-2} \dots s_0)$, then $s^{(i)} = (0 s_{n-2} \dots s_{i+1} \overline{a}_i s_{i-1} \dots s_0)$, where $0 \le i \le r-2$, and $s^{(r-1)} = (1 s_{n-2} \dots s_0)$. Assume s is of class 1 and $s = (1 s_{n-2} \dots s_0)$, then $s^{(i)} = (1 s_{n-2} \dots s_{j+1} \overline{a}_j s_{j-1} \dots s_0)$, where $r-1 \le j \le n-2$ and $i = j-(r-1)$, and $s^{(r-1)} = (0 s_{n-2} \dots s_0)$.

3. Model and Algorithms for Communications on Dualcube

An important metric used to evaluate efficiency of communications is *communication latency* or *transmission time*. The transmission time depends on many factors such as contentions, switching techniques, network topologies etc. In this paper, we assume that the communication links are bidirectional, that is, two directly-connected notes can send messages each other simultaneously. Sending a message containing m words takes $t_s + m d t_w$, where t_s is the message startup time, d is the number of link traversed by the message, and t_w is the per-word transfer time. We also assume that a node can send a message on only one of its links at a time.

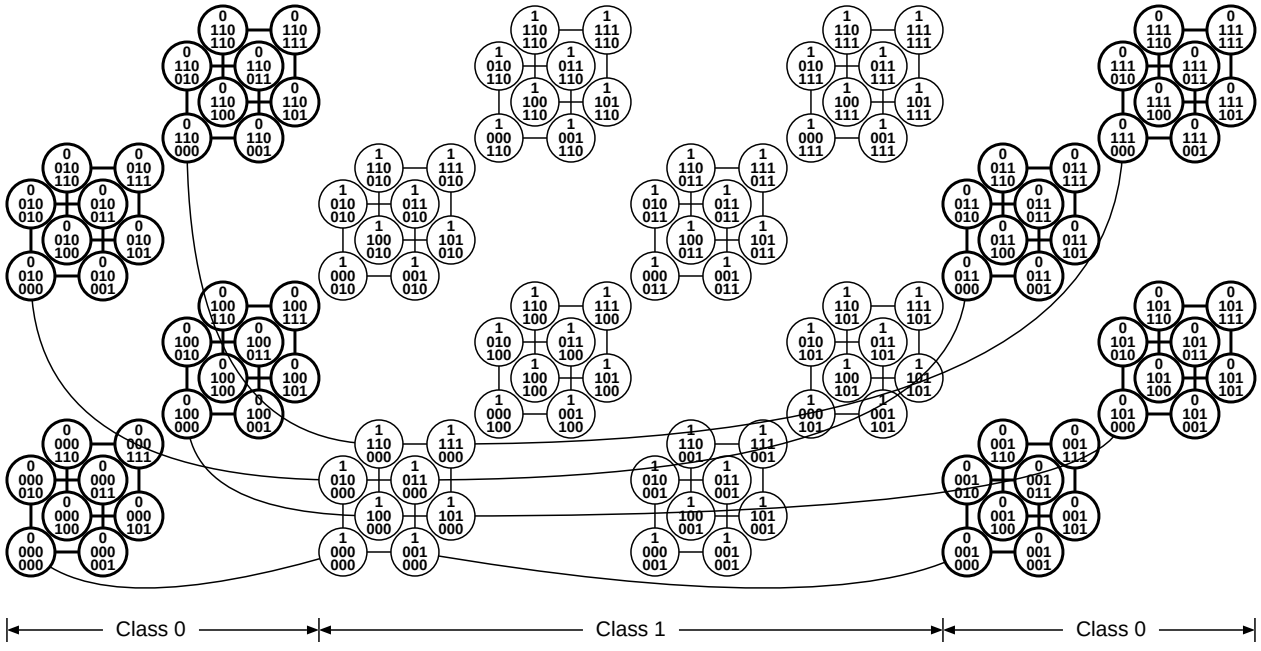

Figure 2. The dualcube F_4

Table 1. Hypercube v.s. dualcube

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

The routing algorithm for the one-to-all personalized communication in dualcube can be described as follows. Initially, the source node s contains $p = 2^{2r-1}$ messages (including a message for itself). First, node s transfers half of the messages to $s' = s^{(r-1)}$ through the cross-edge, and 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, the nodes $u \in C_s \setminus \{s\}$ and $u' \in C_{s'} \setminus \{s'\}$ have the messages for all nodes in cluster C_v and $C_{v'}$, where $v = u^{(r-1)}$ and $v' = (u')^{(r-1)}$. Second, nodes u and u' send the messages of size 2^{r-1} to v and v' through their cross-edges, respectively, and then, u, u', v, and v' send the messages to all nodes in C_u, $C_{u'}$, C_v, and $C_{v'}$ through the binomial trees, respectively. This algorithm is referred to as **One2allP(**F_r, s, M**)** that will be used in the later sections, where s is the source node and M is the set of $p-1$ messages of length m.

The time for the one-to-all personalized communication:

$$T_{One2allP} = T_1 + T_2 = rt_s + \sum_{i=1}^{r} 2^{r+i-2} mt_w + rt_s + \sum_{i=1}^{r} 2^{i-1} mt_w$$
$$= 2rt_s + (2^{2r-1} + 2^{r-1} - 1)mt_w$$
$$= (1+\log p)t_s + (p + \sqrt{p/2} - 1)mt_w.$$

The algorithm for all-to-all broadcast in F_r is shown below. It can be described in three steps. First, the broadcast is done inside each cluster (hypercube). Second, each node in a cluster of class 0 (1) sends the identical message to a node in a cluster of class 1 (0), 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. Third, each node gets the messages of the nodes in other clusters of the same class from the neighbor through the cross-edge. The algorithm is formally described below.

Algorithm 1 (All2allB($F_r, M_x, result$))

1. **begin**
   /* Initially, each node x holds message M_x. At the end, *result* contains $(p-1)$ messages from all other nodes at the end. */
2. **for** processor x, $0 \le x < p$, **pardo**
   /* Stage 1: Broadcast inside the cluster. */
3. **for** $i = 0$ **to** $r-2$ **do**

4. $M'_x =$ **get** $M_{x^{(i)}}$;
5. $M_x = M_x \cup M'_x$;
6. **endfor**

/* Stage 2: Broadcast to the clusters of the other class. */

7. $M'_x =$ **get** $M_{x^{(r-1)}}$;
8. $result = M_x$; $M_x = M'_x$; $temp = M'_x$
9. **for** $i = 0$ **to** $r-2$ **do**
10. $M'_x =$**get** $M_{x^{(i)}}$;
11. $M_x = M_x \cup M'_x$;
12. **endfor**
13. $result = result \cup M_x$; $M_x = M_x \setminus temp$;

/* Stage 3: Include the messages from other clusters of the same class. */

14. $M'_x =$ **get** $M_{x^{(r-1)}}$;
15. $result = result \cup M'_x$;
16. **endfor**
17. **end.**

Let T_1, T_2, and T_3 are the times it takes to complete the first, the second, and the third stages, respectively. Then we have

$$T_1 = \sum_{i=1}^{r-1}(t_s + 2^{i-1}mt_w) = (r-1)t_s + (2^{r-1}-1)mt_w.$$

$$T_2 = t_s + 2^{r-1}mt_w + \sum_{i=1}^{r-1}(t_s + 2^{r+i-2}mt_w) = rt_s + 2^{2r-2}mt_w.$$

$$T_3 = t_s + (2^{2r-2} - 2^{r-1})mt_w.$$

Therefore, the total time to complete the all-to-all broadcast is

$$T_{All2allB} = 2rt_s + (2^{2r-1}-1)mt_w = (1+\log p)t_s + (p-1)mt_w.$$

The following many-to-many multicast in F_r is used frequently for designing computational schemes in F_r. Initially, each node x in the cluster C contains a message msg_x. After the multicast, each node y in F_r with nodeID(x) = nodeID(y) contains the message msg_x. The routing scheme is as follows. For convenience, assume cluster C is of class 0. First, we gather the messages in cluster C so that every node in C contains $\cup_{x \in C} msg_x$. Second, each node x in C sends message to $x' = x^{(r-1)}$ through the cross-edge, and then node x' calls one-to-all personalized communication subroutine to distribute the received messages to every node in $C_{x'}$. Third, node z in cluster of class 1 with nodeID(z) = clusterID(z) broadcasts its message to all other nodes in C_z, and then, all nodes in clusters of class 0 except those in C receive the broadcasted messages through the cross-edges. The algorithm is formally described below.

Algorithm 2 (Many2manyM(F_r, C, msg_x))

1. **begin**

/* Initially, each node $x \in C$ holds message msg_x. At the end, each node $y \in F_r$ with nodeID(x) = nodeID(y) holds message msg_x. */

/* Stage 1: Gather messages inside the cluster. */

2. **for** node $x \in C$ **pardo**
3. **for** $i = 0$ **to** $r-2$ **do**
4. $temp =$ **get** $msg_{x^{(i)}}$;
5. $msg_x = msg_x \cup temp$;
6. **endfor**
7. **endfor**

/* Stage 2: Send messages to the clusters of the other class. */

8. **for** node y, $y^{(r-1)} \in C$ **pardo**
9. $msg_y =$ **get** $msg_{y^{(r-1)}}$;
10. **call One2allP(**C_y, y, msg_y**)**;
11. **endfor**

/* Stage 3: Send messages to the other clusters of the same class. */

12. **for** node z with nodeID(z) = clusterID(z) and classID(z) $\neq$ classID(C) **pardo**
13. $msg'_z = msg_z$;
14. **call One2allB(**C_z, z, msg'_z**)**;
15. **endfor**
16. **for** node $x \notin C$ and classID(x) = classID(C) **pardo**
17. $msg_x =$ **get** $msg'_{x^{(r-1)}}$;
18. **endfor**
19. **end.**

The time for many-to-many multicast is

$$T_{Many2manyM} = 2(r-1)t_s + 2(2^{r-1}-1)mt_w +$$
$$t_s + 2^{r-1}mt_w + (r-1)(t_s + mt_w) + t_s + mt$$
$$= (3r-3)t_s + (3 \times 2^{r-1} + r - 2)mt_w.$$

The following subroutines [6] that find the sum and the prefix of the datum $data_x$ in each node x of a cluster C, respectively, will be used in the later sections.

Algorithm 3 (ClusterSum($C, r-1, data_x, sum$))

1. **begin**
2. **for** node x, $x \in C$ **pardo**
3. $sum = data_x$;
4. **for** $i = 0$ **to** $r-2$ **do**
5. $temp =$ **get** $sum_{x^{(i)}}$;
6. $sum_x = sum_x + temp$;
7. **endfor**
8. **endfor**
9. **end.**

Algorithm 4 (ClusterPrefix($C, r-1, data_x, pf_x$))

1. **begin**
2. **for** each node x, $x \in C$ **pardo**
3. $pf = data_x$;
4. **for** $i = 0$ **to** $r-2$ **do**
5. $temp =$ **get** $pf_{x^{(i)}}$;
6. **if** $x > x^{(i)}$ **then** $pf_x = pf_x + temp$;

7. **endfor**
8. **endfor**
9. **end.**

4. Parallel Prefix Computation and Vector-Matrix Multiplication

In this section, we first give an algorithm for parallel prefix computation. Assume that each processor (node) x holds a datum $data_x$. After prefix computation, each processor x will hold $pf_x = \sum_{y \le x} data_y$. The algorithm for the parallel prefix computation is similar to that of the all-to-all broadcast described in the previous section. The algorithm is described below.

Algorithm 5 (DualPrefix($F_r, data_x, pf_x$))

1. **begin**
   /* Stage 1: Local prefix computation inside the cluster. */
2. **for** each cluster C **pardo**
3. **call** ClusterSum($C, r-1, data_x, sum$);
4. **call** ClusterPrefix($C, r-1, data_x, pf_x$);
5. **endfor**
   /* Stage 2: Include *sum* from clusters of the other class. */
6. **for** each cluster C **pardo**
7. **call** ClusterSum($C, r-1, data_x, sum$);
8. **call** ClusterPrefix($C, r-1, data_x, pf_x$);
9. **endfor**
   /* Stage 3: Include *sum* from other clusters of the same class. */
10. **for** processor $x,\ 0 \le x < p$ **pardo**
11. $temp =$ **get** $sum_{x^{(r-1)}}$;
12. **if** $x > x^{(i)}$ **then** $pf_x = pf_x + temp$;
13. **endfor**
14. **end.**

For vector-matrix multiplication $w = vM$, we assume that the length of vector $|v| = 2^{r-1}$, and the size of matrix M is $2^r \times 2^{r-1}$. The case that $|v| = 2^r$ and M is $2^r \times 2^r$ can be handled sequentially, by calling the algorithm twice and the computational time is doubled.

Assume that processor 0 holds matrix M and vector v initially. First, the algorithm uses the one-to-all personalized communication algorithm to distribute the elements of M to every node in F_r such that each cluster holds one column of matrix M. Vector v is first distributed to every node in the cluster with cluster ID $= 0$ such that node x contains $v[x]$. Then $v[x],\ 0 \le x < 2^{r-1}$ is broadcasted to all other nodes with the same node ID in F_r. Second, the algorithm performs one scalar multiplication and then processors with node ID $= 0$ gathers the sum of the products on each cluster. The algorithm is formally presented as follows.

Algorithm 6 (VectorMatrixMultiplication(v, A, w))

1. **begin**
   /* Stage 1: Broadcast vector x and matrix A. */
2. **call** One2allP(F_r, s, A);
   /* Processor x holds $A_x = A[i, j]$, $i =$ nodeID[x] and $j =$ classID[x] **cat** clusterID[x]. */
3. **call** One2allP(H_{r-1}, s, v);
   /* Processor $x \in C_s$ holds $v_x = v[i]$, $i =$ nodeID[x]*/
4. **call** Many2manyM(F_r, C_s, v);
   /* Stage 2: Compute w */
5. **for** processor $x,\ 0 \le x < p$ **pardo**
6. $w_x = A_x * v_x$;
7. **for** $i = 0$ **to** $r-2$ **do**
8. $temp =$ **get** $w_{x^{(i)}}$;
9. $w_x = w_x + temp$;
10. **endfor**
11. **endfor**
12. **end.**

The above algorithm for the vector-matrix multiplication can be used in pipelined fashion to do the matrix multiplication $C = AB$, where A, B, and C are $m \times m$ matrices. If $m = 2^r$ then the algorithm is called $2m$ times to get product matrix C. In the next section, we show a faster algorithm for the matrix multiplication in F_r.

5 Parallel Matrix Multiplication

Matrix multiplication can be done efficiently in hypercube [6]. In this section, we show that the matrix multiplication can be done efficiently in dualcube also.

5.1 Address Assignment

Consider the problem of multiplying $m \times m$ matrices A and B on an r-dualcube to obtain the product matrix C, where $m = 2^{(2r-1)/3}$ and $p = m^3 = 2^{2r-1}$ processors are used. Note that $(2r-1)$ is assumed to be a multiple of 3. Each processor has three registers R_A, R_B, and R_C. For the sake of algorithm description, it is convenient to label the dualcube processors by three indices i, j, k, where each index is a $((2r-1)/3)$-bit binary number.

Recall that the address of the r-dualcube consists of three fields c, a, and b, where c is 1-bit class indicator, a is $(r-1)$-bit cluster number, and b is $(r-1)$-bit processor number within a cluster. If a processor is of class 0, its address will be cab, otherwise cba. There are 2^r clusters and each cluster has 2^{r-1} processors. Because a cluster is an $(r-1)$-cube, it is convenient to let the m terms which are summed to form an element of C be located within a cluster. We equally distribute $m \times m$ elements of a matrix into 2^r clusters, each cluster holds $m^2/2^r = 2^{(r-2)/3}$ elements. The assignment of

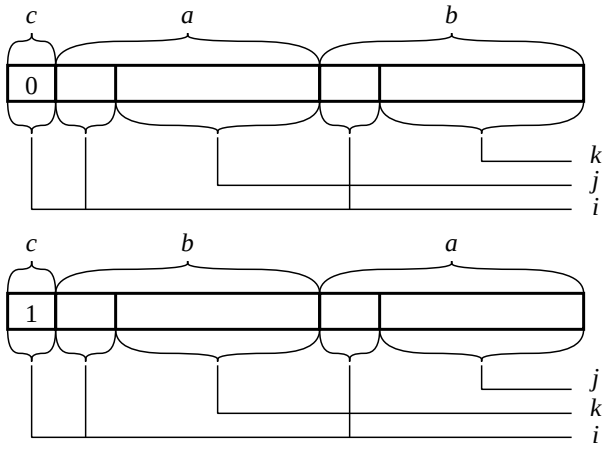

Figure 3. Indices and address fields

indices i, j, k is shown in Fig. 3. The index k is the lower $((2r-1)/3)$ bits of the processor number b in a cluster, j is the lower $((2r-1)/3)$ bits of the cluster number a, and i is the rest bits which consists of 1-bit class indicator c, the higher $((r-2)/3)$ bits of a, and the higher $((r-2)/3)$ bits of b.

5.2 Multiplication Algorithm

Initially, Processor$(i,j,0)$ holds A_{ij} and B_{ij} in its R_A and R_B registers, respectively. All other processor registers are initialized to 0. At the end of computation, Register R_C of Processor$(i,j,0)$ will hold element C_{ij} of the product matrix C. The algorithm performs all m^3 multiplications needed for multiplying two $m \times m$ matrices concurrently, one in each of the m^3 processors. The remainder of the process is to ensure that each processor holds the two elements of A and B that it multiplies and to add the requisite terms that form each element C_{ij} of the result matrix. The algorithm of multiplying $m \times m$ matrices on an r-dualcube, with $2r-1 = 3\log_2 m$ is listed below, where $N_h^l(x)$ denotes a processor address formed by inverting bit l of h of x for $h = i$, j, or k.

Algorithm 7 (MatrixMatrixMultiplication(A,B,m))

1. **begin**
2. **for** $l = 0$ **to** $(2r-1)/3-1$ Processor $x = ijk$, $0 \le i,j,k < m$, **do** /* *first "for" loop* */
3. **if** bit l of k is 1
4. get $y = R_A[N_k^l(x)]$; $z = R_B[N_k^l(x)]$
5. set $R_A[x] = y$; $R_B[x] = z$
6. **endif**
7. **endfor**
8. **for** $l = 0$ **to** $(2r-1)/3-1$ Processor $x = ijk$, $0 \le i,j,k < m$, **do** /* *second "for" loop* */
9. **if** bit l of k and j are different
10. get $y = R_A[N_j^l(x)]$
11. set $R_A[x] = y$
12. **endif**
13. **endfor**
14. **for** $l = 0$ **to** $(2r-1)/3-1$ Processor $x = ijk$, $0 \le i,j,k < m$, **do** /* *third "for" loop* */
15. **if** bit l of k and i are different
16. get $y = R_B[N_i^l(x)]$
17. set $R_B[x] = y$
18. **endif**
19. **endfor**
20. Processor x, $0 \le x < p$, **pardo** $R_C = R_A \times R_B$
21. **for** $l = 0$ **to** $(2r-1)/3-1$ Processor $x = ijk$, $0 \le i,j,k < m$, **do** /* *fourth "for" loop* */
22. **if** bit l of k is 0
23. get $y = R_C[N_k^l(x)]$
24. set $R_C[x] = R_C[x] + y$;
25. **endif**
26. **endfor**
27. **end**.

The first "for" loop copies the elements of A and B into all other processors within clusters. Because Processor$(i,j,0)$ holds A_{ij} and B_{ij}, we start from the least significant bit to do the recursive doubling copies, it requires O(r) communication steps. The second and the third "for" loops copy the elements of A and B, respectively, into all other processors in the different clusters. These two loops require many-to-many personalized communications among clusters, we will describe it in the next subsection. The final "for" loop, after the multiplications, computes the sum of the m terms that form each of the elements of C, again using recursive doubling scheme within clusters.

An example for multiplying two 2×2 matrices on a 2-dualcube is shown in Fig. 4. For this example, $(2r-1)/3 = 1$, so each of "for" loops degenerates into a single communication step. In the first "for" loop, processors with processor number $b = 1$ (i.e., Processor 1, 3, 6, and 7) receive and store R_A and R_B values from their neighbors $N_k^0(x)$ (i.e., from Processor 0, 2, 4, and 5). In the second "for" loop, Processors 1, 2, 5, and 6, i.e., those with different j and k components in their node labels, receive and store R_A values from $N_j^0(x)$ (i.e., from Processor 3, 0, 4, and 7). The third "for" loop updates the R_B values in those processors whose node labels have different i and k components (i.e., Processor 1, 3, 4, and 5 receive values from $N_i^0(x)$: Processor 6, 7, 0, and 2). At this point, data distribution is complete. The eight processors then independently multiply their R_A and R_B values, storing the results in R_C. The final "for" loop adds its value and the value from neighbor $N_k^0(x)$ to obtain elements of the result matrix C in Processors 0, 2, 4, and 5.

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix}$$

Processors		Init	1st	2nd	3rd	Mul	4th
c a b	Lab	R_A, R_B	R_A, R_B	R_A, R_B	R_A, R_B	R_C	R_C
0 0 0	000	1, 5	1, 5	1, 5	1, 5	5	19
0 0 1	001	0, 0	1, 5	2, 5	2, 7	14	
0 1 0	010	2, 6	2, 6	1, 6	1, 6	6	22
0 1 1	011	0, 0	2, 6	2, 6	2, 8	16	
1 0 0	100	3, 7	3, 7	3, 7	3, 5	15	43
1 0 1	110	0, 0	3, 7	4, 7	4, 7	28	
1 1 0	101	4, 8	4, 8	3, 8	3, 6	18	50
1 1 1	111	0, 0	4, 8	4, 8	4, 8	32	

Figure 4. Two 2×2 matrices multiplication

5.3 Data Transfer

In the algorithm described above, the second and third "for" loops copy the elements of *A* and *B*, respectively, into all other processors in different clusters that need them to perform the m^3 parallel multiplications. Processor(i, j, k) needs copying R_A (or R_B) value from a source processor if and only if there is a bit position l where the j (or i) and k components are different. The source address of R_A (or R_B) is determined by inverting the value of l bit position of j (or i). As mentioned early, If a processor is of class 0, its address will be *cab*, otherwise *cba*.

For example, in the 5-dualcube, processor 001010011 ($c = 0, a = 0101, b = 0011, i = 000$, $j = 101$, and $k = 011$) copies R_B from processor 011010011 ($i = 010$, $j = 101$, and $k = 011$) because the components of i and k in the bit position 1 are different, and copies R_A from processor 000010011. The second example is that processor 001010100 ($c = 0, a = 0101, b = 0100, i = 000$, $j = 101$, and $k = 100$) copies R_B from processor 101000101 ($i = 100$, $j = 101$, and $k = 100$) because the components of i and k in the bit position 2 are different, and copies R_B from processor 001000100. Note that the source processor addresses for copying R_B in the first and second examples have formats *cab* and *cba*, respectively.

It is clear that when a processor copies R_A value from a source processor, there is one and only one bit position l in cluster number a where the components of the source and destination addresses are different. Consequently, it always takes three communication steps to finish the data transmission. In the first step, the source processor sends data through its cross-edge to a processor of the different class. Then that processor sends the data to the neighbor $N_j^l(x)$. In the last step, the data will arrive destination processor through the cross-edge. Such communication uses an extra cluster of the different class because there is no direct link between the source and destination clusters.

Copying R_B value is more complex because the binary bit of index i may appear in all the address numbers c, a, and b. In the first example mentioned above, the source and destination processors are located in different clusters of the same class, but the bit position l is in a, so it is the same as the case of copying R_A. The route of the first example will be 011010011 → 111010011 → 101010011 → 001010011.

In the second example, the source and destination processors are of different classes, there is a direct link between the source and destination clusters. But there are 2^{r-2} words in the different processors of the source cluster which need to be transferred to the different processors of the destination clusters. We use the following strategy to route these words. First, we let each word go through the cross-edge and route the word within the cluster to a processor so that the processors' b value will match the destination processors' b value. Then we let the word go through the cross-edge again and route within the cluster to a processor so that the processors' b value will match the destination processors' a value. Finally we let the word go through the cross-edge to reach the destination processor. The route of the second example will be 101000101 → 001000101 → 001000100 → 101000100 → 101010100 → 001010100. Fig. 5 shows the data transmission for preparing the data for the cluster 0 of class 0 in a 5-dualcube. In the figure, only the source addresses of second and third "for" loops are listed.

Generally, If l appears in b, the data can be obtained from a neighbor processor within the cluster. If l appears in a, the data comes from a processor in a different cluster of the same class. The routing from source processor to destination processor can be done by going through the source processors' cross-edge, then routing in the cluster one time, and finally going through the cross-edge to reach the destination processor. It takes three communication steps. If l appears in c, the data comes from a processor of the different class. The routing from source processor to destination processor can be done by going through the cross-edge and routing in the cluster twice, and going through the cross-edge to reach the destination processor. The longest distance will be $2r+1$.

Now, we analyze the transmission time for the matrix multiplication on both the dualcube and hypercube. Note that the computation times on both the dualcube and hypercube are the same. The total time for the communications on hypercube is

$$T_{hypercube} = [(t_s + 2t_w) + 3(t_s + t_w)](2r-1)/3 = (4t_s + 5t_w)(2r-1)/3$$

For the communications on the dualcube, the time for the first and the fourth "for" loops is

$$T_{1,4} = (2t_s + 3t_w)(2r-1)/3$$

Processor	Init	1st	2nd	3rd	Source for R_A (2nd "for" loop)			Source for R_B (3rd "for" loop)		
Labels	R_A, R_B	R_A, R_B	R_A, R_B	R_A, R_B	$l=0$	$l=1$	$l=2$	$l=0$	$l=1$	$l=2$
000000000	1, 65	1, 65	1, 65	1, 65						
000000001	0, 0	1, 65	2, 65	2, 73	000010001			000001001		
000000010	0, 0	1, 65	3, 65	3, 81		000100010			010000010	
000000011	0, 0	1, 65	4, 65	4, 89	000010011	000100011		000001011	010000011	
000000100	0, 0	1, 65	5, 65	5, 97			001000100			101000000
000000101	0, 0	1, 65	6, 65	6, 105	000010101		001000101	000001101		101010000
000000110	0, 0	1, 65	7, 65	7, 113		000100110	001000110		010000110	101100000
000000111	0, 0	1, 65	8, 65	8, 121	000010111	000100111	001000111	000001111	010000111	101110000
000001000	9, 73	9, 73	9, 73	9, 65				000000000		
000001001	0, 0	9, 73	10, 73	10, 73	000011001					
000001010	0, 0	9, 73	11, 73	11, 81		000101010		000000010	010001010	
000001011	0, 0	9, 73	12, 73	12, 89	000011011	000101011			010001011	
000001100	0, 0	9, 73	13, 73	13, 97			001001100	000000100		111000000
000001101	0, 0	9, 73	14, 73	14, 105	000011101		001001101			111010000
000001110	0, 0	9, 73	15, 73	15, 113		000101110	001001110	000000110	010001110	111100000
000001111	0, 0	9, 73	16, 73	16, 121	000011111	000101111	001001111		010001111	111110000

Figure 5. Data transfer for two 8×8 matrices multiplication

For the second and third "for" loops, the times can be evaluated as following. The times T_b, T_a, and T_c for the cases that l appears in b, a, and c, respectively, are

$$T_b = (t_s + t_w)(r-2)/3$$

$$T_a = (t_s + 3t_w)(r-1)$$

$$T_c = t_s + (2r+1)t_w$$

The total time for the communications is

$$T_{dualcube} = T_{1,4} + T_b + T_a + T_c = (4t_s + 11t_w)(2r-1)/3$$

6. Conclusion and Future Work

In this paper, we proposed efficient algorithms for communication and some numerical computations on dualcube. The results of this paper show that the dualcube is very promising: the efficiencies of the algorithms for collective communication and matrix multiplication are almost the same as hypercube with a small extra overhead for routing the data around. Since the dualcube with eight links per node can contain more than thirty-two thousands nodes, it is a good candidate as an interconnection network of the high-performance computer clusters of the next generation. Some directions concerning dualcube which are worth of further research are: embedding other frequently used topologies into dualcube, mapping application algorithms onto dualcube, and investigating the fault-tolerant properties of dualcube with faulty nodes and/or faulty links.

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

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

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

[5] 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.

[6] B. Parhami. *Introduction to parallel processing, algorithm and architecture*. Plenum Press, 1999.

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

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

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

[10] 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.