The IASTED International Conference on Parallel and Distributed Computing and Systems, Marina del Rey, CA, USA, November 3–5, 2003, pp.43–50

Disjoint Paths in Metacube

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
A new interconnection network with low-degree for very large parallel computers called metacube (MC) has been introduced recently. The MC network has short diameter similar to that of the hypercube. However, the degree of an MC network is much lower than that of a hypercube of the same size. More than one hundred of millions of nodes can be connected by an MC network with up to 6 links per node. The MC network has 2-level cube structure. An MC(k,m) network that connects $p = 2^{m2^k+k}$ nodes with $m+k$ links per node has two parameters, k and m, where k is the dimension of the high-level cubes (class-cubes) and m is the dimension of the low-level cubes (clusters). In this paper, we describe an efficient algorithm for finding disjoint paths in MC networks. We show that, for any two distinct nodes u and v in an MC(k,m), $k+m$ disjoint paths from u to v can be found in $O(\log^2 p)$ time. The length of the paths is at most $H(s,t)+2^k+m+5$, where $H(s,t)$ is the Hamming distance between s and t. The result implies that a fault-free path between any two nonfaulty nodes can be found in an MC(k,m) with up to $m+k-1$ faulty nodes.

KEY WORDS
Interconnection networks, hypercube, disjoint paths, fault-tolerance

1 Introduction

The hypercube has been widely used as the interconnection network in a wide variety of parallel systems such as Intel iPSC [11], the nCUBE [3], the Connection Machine CM-2 [10], and SGI Origin 2000 [4] [9]. An n-dimensional hypercube (n-cube) contains 2^n nodes and has n edges per node. If unique n-bit binary addresses are assigned to the nodes of an n-cube, 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 systems.

However, the number of edges per node increases logarithmically as the total number of nodes in the hypercube increases. Currently, the practical number of links is limited to about eight per node [9]. If one node has one processor, the total number of processors in a parallel system with an n-cube connection is restricted to several hundreds. Therefore, it is interesting to develop an interconnection network which will link a large number of nodes with a small number of links per node while retaining most of the hypercube's topological properties.

Several variations of the hypercube have been proposed in the literature. Those focused on reduction of the number of edges of the hypercube include cube-connected cycles [8], reduced hypercube [12] and the hierarchical cubic network [1]. However, none of them can provide the flexibility that the metacube supports. On practical side, a parallel computer Origin2000 [4] [9] is constructed with the hypercube-like structure. Origin2000 reduces the number of links required by introducing *Cray Router* to connect hypercubes (clusters). A Cray Router is the high level router that does not connect processors directly. The processors are attached to regular routers within the clusters. Each regular router has six links. Two links connect two nodes; each node contains two processors. Three links are 3-cube edges and a CrayLink connects to a Cray Router.

Recently, Y. Li et al. introduced a new interconnection network, called *metacube*, or MC network [7]. The MC network shares many desirable properties of the hypercube (e.g., the key property of the hypercube, low diameter etc.) and can be used as an interconnection network for a parallel computer system of almost unlimited size with just a small number of links per node. For example, an MC$(2,3)$ with 5 links per node has 16384 nodes and an MC$(3,3)$ with 6 links per node has $2^{27} = 134,217,728$ nodes. The number of nodes connected by the MC is much larger than that of the HCN or the RH with the same amount of links per node. The CCC uses only 3 links per node. However, because of its ring structure, the diameter or the length of the routing path in CCC is about twice of that of the hypercube. Compared with the CCC, the MC has shorter diameter, length of the routing path, and the broadcasting time. With metacube architecture, the Origin2000 can connect much more processors directly without using any Cray Router. Note that there is no need to modify any hardware circuit of the Origin2000; what we need to do is to connect router board ports with link cables in metacube topology.

In this paper, we give efficient algorithms for finding disjoint paths in metacube. The remainder of this paper is organized as follows. Section 2 introduces the MC network, its topological properties, and a point-to-point rout-

ing algorithm in the MC network. Section 3 gives the algorithm for finding $m+k$ disjoint paths between two distinct nodes in an MC(k,m). Section 4 concludes the paper and presents some future research directions.

2 Preliminaries

The MC network is motivated by the dual-cube network proposed by Li and Peng [5] [6] that mitigates the port limitation problem in the hypercube network so that the number of nodes in the network is much larger than that of the hypercube with a fixed amount of link per node. The MC network includes the dual-cube as a special case. An MC network has a 2-level cube structure: high-level cubes represented by the leftmost k bits of the binary address of the node which contains $m2^k+k$ bits (these k bits serve as a class indicator), and low-level cubes, called clusters that form the basic components in the network, represented by the m bits of the remain $m2^k$ bits, which occupy the different portions in the $m2^k$ bits for different classes.

More specifically, there are two parameters in an MC network, k and m. An MC(k,m) contains $h=2^k$ *classes*. Each class contains $2^{m(h-1)}$ *clusters*, and each cluster contains 2^m *nodes*. Therefore, an MC(k,m) uses $mh+k$ binary bits to identify a node and the total number of nodes is 2^n where $n=mh+k$. The value of k affects strongly the growth rate of the size of the network. An MC$(1,m)$ containing 2^{2m+1}nodes is called a *dual-cube*. Similarly, an MC$(2,m)$, an MC$(3,m)$ and an MC$(4,m)$ containing 2^{4m+2} nodes, 2^{8m+3} nodes and 2^{16m+4} nodes are called *quad-cube*, *oct-cube* and *hex-cube*, respectively. Since an MC$(3,3)$ contains 2^{27} nodes, the oct-cube is sufficient to construct practically parallel computers of very large size. The hex-cube is of theoretical interest only. Note that an MC$(0,m)$ is a hypercube.

A node in an MC(k,m) can be uniquely identified by a $(mh+k)$-bit binary number. The leftmost k-bit binary number defines a class of the node (*classID*). There are h classes. In each class, there are 2^{mh} nodes and each node is represented by a mh-bit binary number. 2^m nodes of the same class form a cluster. Therefore, there are $2^{m(h-1)}$ clusters in each class. An m-bit binary number, located in a special portion of the mh-bit (will be explained in the next paragraph) identifies a node within the cluster (*nodeID*). Therefore, the $(mh+k)$-bit node address in an MC(k,m) is divided into three parts: a k-bit *classID*, an $m(h-1)$-bit *clusterID* and an m-bit *nodeID*.

In the following discussion, we use $u=(c_u,M_u[h-1],\ldots,M_u[1],M_u[0])$ to denote the *ID* of node u, where c_u is a k-bit binary number and $M_u[i],\ 0\le i\le h-1$ are m-bit binary numbers. Let $classID(u)=c_u$, $nodeID(u)=M_u[c_u]\times 2^{c_u}$ and $clusterID(u)=\sum_{0\le i\le h-1,i\ne c_u}M_u[i]\times 2^i$. The mh-bit number $nodeID(u)+clusterID(u)$ is a unique identifier of node u in class c_u. For example, $u=0100111000$ in an MC$(2,2)$ is denoted as node 56 of class 1 and node set $(48,52,56,60)$ in class 1 forms a cluster with $clusterID=48$. Fig. 1 shows the format of a node address for an MC(k,m).

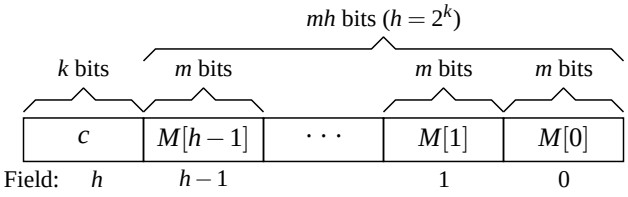

Figure 1. Format of a node address for an MC(k,m)

The links of an MC(k,m) is constructed in the following manner. The m-bit field $M[c]$ in the address of a node of class c forms a low-level m-cube with m links, namely *cube-edge*. These *low-level* m-cubes are called *clusters*. A cluster containing node u is denoted as C_u. The links that connect nodes among clusters are called *cross-edges* and are defined as following. For any two nodes whose addresses differ only in a bit position in the class field, there is a cross-edge connecting these two nodes. That is, the k-bit field c forms a *high-level* k-cube which connects those nodes whose addresses except class field are the same. The high-level k-cube is called *class-cube*.

The addresses of two nodes connected by a cross-edge differ only on one bit position within the k-bit class field and there is no direct connection among the clusters of the same class. Therefore, a node in an MC(k,m) has $m+k$ links: m links construct an m-cube cluster and k links construct a k-cube. For example, the neighbors in the cluster of the node with address (01,111,101,110,000) in an MC$(2,3)$ have addresses (01,111,101,11$\underline{1}$,000), (01,111,101,1$\underline{0}$0,000) and (01,111,101,$\underline{0}$10,000). The underlined bits are those that differ from the corresponding bits in the address of the referenced node. The two neighbors in the high-level cube are (0$\underline{0}$,111,101,110,000) and ($\underline{1}$1,111,101,110,000).

Fig. 2 shows the structure of an MC$(2,2)$, where the clusters in the same square are of the same class. The decimal numbers are $nodeID+clusterID$. In Fig. 2, there are $2^{2(2^2-1)}=64$ clusters in each square and each cluster is a 2-cube. The figure shows only 4 high-level cubes, each of which contains a distinct node in the cluster 0 of the class 0.

The problem of finding a path from a source node s to a destination node t, and forwarding messages along the path is known as the point-to-point routing problem. It is the basic problem for any interconnection network. We describe briefly below the point-to-point routing algorithm in metacube [7]. This algorithm is the building block for find disjoint paths in metacube.

We adopt the following notation. In the metacube MC(k,m), each node has $m+k$ neighbors. Let $s^{(i)},\ 0\le i\le k-1$, be the ith dimensional neighbor of node s within the k-cube, that is, the addresses of s and $s^{(i)}$ differ in the ith bit position (the rightmost bit is the 0th bit) in the class field c. Let $s^{(i+k)},\ 0\le i\le m-1$, be the ith dimensional

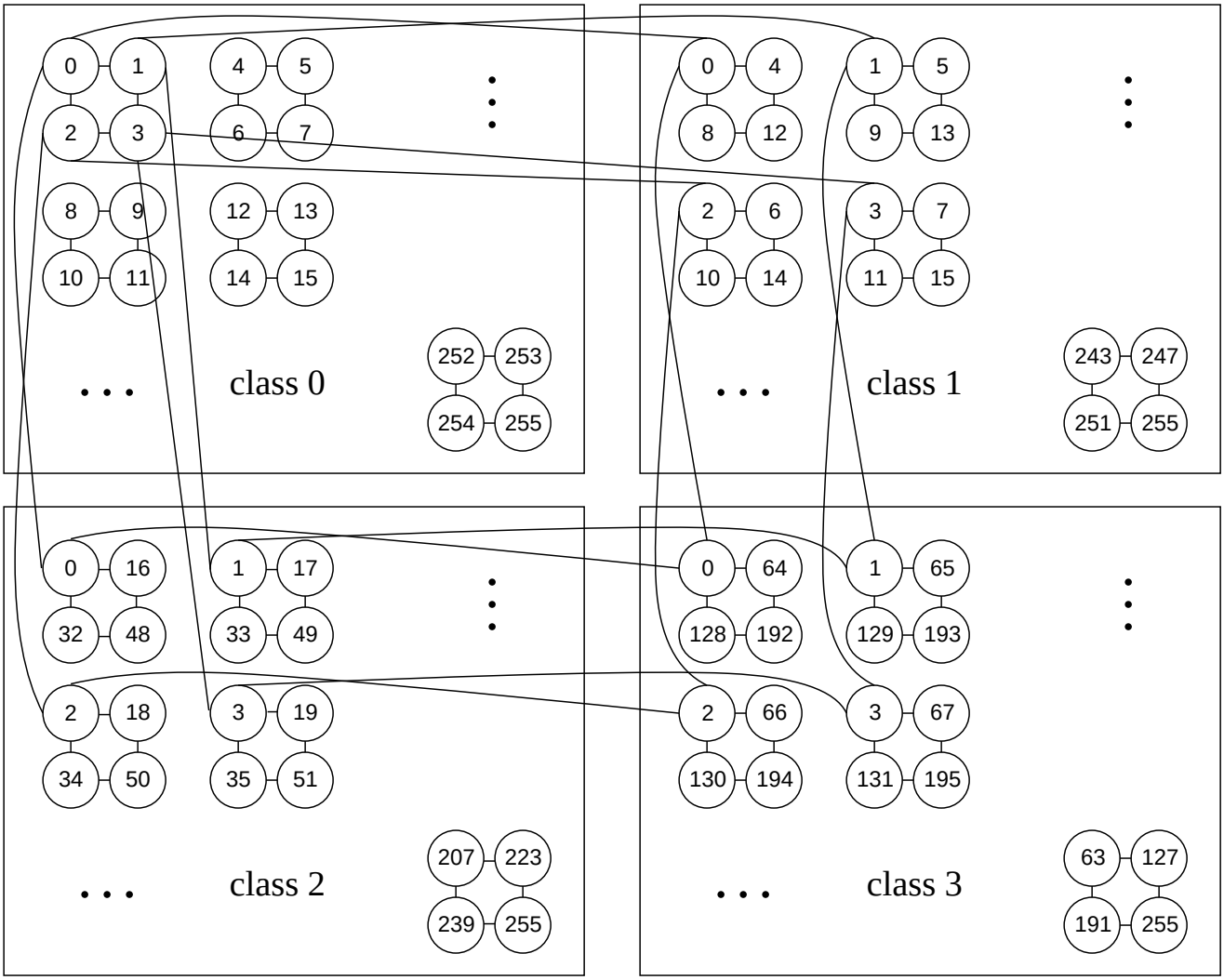

Figure 2. A metacube MC(2,2)

neighbor of node s in the m-cube, that is, the addresses of s and $s^{(i+k)}$ differ in the ith bit position in the field $M[c]$. Let $s^{(i,j)} = (s^{(i)})^{(j)}$ for $0 \le i, j \le m+k-1$. We use $(u \to v)$ to denote a path from node u to node v. If the length of a path $(u \to v)$ is 1 (through a single edge), the path is denoted as $(u : v)$, and the edge is denoted as (u, v).

In a graph $G = (V, E)$ where V is the set of all vertices (nodes) and E is the set of all edges in G, let $P' = (v_0 \to v_{h-1}) = (v_0 : v_1 : \ldots : v_{h-1})$ be a path from node v_0 to node v_{h-1}, where $v_i \in V$ for $0 \le i \le h-1$ and edge $(v_{j-1}, v_j) \in E$ for $1 \le j \le h-1$. We say P' is a *Hamiltonian path* if (1) P' contains every node in V and (2) nodes v_i $(0 \le i \le h-1)$ are all distinct. Let $P = (v_0 \to v_h) = (P' : v_h)$, where $v_h = v_i$, for $i = 0, 1, \ldots,$ or $h-2$. If $v_h = v_0$, then P becomes a *Hamiltonian cycle*[1]; otherwise, we call P a *extended-Hamiltonian path*. The length of a Hamiltonian path in a k-cube is $2^k - 1$; the length of a Hamiltonian cycle or an extended Hamiltonian path is 2^k. Let a *weak-Hamiltonian path* be a Hamiltonian path, a Hamiltonian cycle, or an extended-Hamiltonian path. The following lemma [7] is needed for the routing algorithm.

Lemma 1. *Given any two nodes s and t in an n-cube, there exists a weak-Hamiltonian path from s to t.*

Since we can modify only a small portion (m bits) by cube-edges, we need to move to clusters of distinct classes (along cross-edges) to modify the other portions of the *ID*. We also need to arrange the order of nodes on the path so that the last portion modified is the *nodeID* of t or its neighbor. Since there are h classes, the efficient way to do this is by following a weak Hamiltonian path from c_s (class number of node s) to c_t (class number of node t) in the k-cube.

For each node u in the k-cube, let $next(u)$ be the node next to u in the weak-Hamiltonian path from c_s to c_t. Let the node addresses of s and t be $(c_s, M_s[h-1], \ldots, M_s[1], M_s[0])$ and $(c_t, M_t[h-1], \ldots, M_t[1], M_t[0])$, respectively. For the routing within an m-cube of class c, we can follow the *ascending routing* strategy, by which the

[1] A Hamiltonian cycle is defined as a path through a graph which starts and ends at the same vertex and includes every other vertex exactly once.

least significant non-zero bit of $(M_s[c] \oplus M_t[c])$ is chosen as the first dimension for routing, and so on. The routing algorithm in an MC(k,m) is given below. The **loop** will terminate when the **break** is executed. Notice that the details of routing in the m-cube is omitted in the algorithm.

Algorithm 1 (p1(m,k,s,t))
begin /* build a path $p1$ from s to t in MC(k,m) */
$u = c_s$; $v = s$; $p1 = v$;
loop always
$w = (u, M_v[h-1], \ldots, M_v[u+1], M_t[u],$
$M_v[u-1], \ldots, M_v[0])$; /* $M_s[u] \rightarrow M_t[u]$ */
if $w \neq v$, **then** $p1 = (p1 \rightarrow w)$;
if $w = t$, **then break**;
$v = w$;
$w = (next(u), M_v[h-1], \ldots, M_v[u+1], M_v[u],$
$M_v[u-1], \ldots, M_v[0])$; /* $u \rightarrow next(u)$ */
$p1 = (p1 : w)$;
$u = next(u)$;
endloop
end

Example 1. In an MC$(2,3)$, let $s = 00000000000000$ and $t = 00001110101011$. The weak-Hamiltonian path for $(c_s \rightarrow c_t)$ in the high-level 2-cube is a Hamiltonian cycle, $(00:01:11:10:00)$ for instance. The routing inside clusters may start from any dimension. The path can be
00000000000000 : 00000000000001 : 00000000000011 : 01000000000011 : 01000000001011 : 01000000101011 : 11000000101011 : 11001000101011 : 10001000101011 : 10001010101011 : 10001110101011 : 00001110101011.

In the case of a Hamiltonian cycle, the parameter $next(s)$ give the direction of the Hamiltonian cycle. For example, in Example 1, if we let $next(s) = 10$, then the Hamiltonian cycle for $(c_s \rightarrow c_t)$ in the high-level 2-cube will be $(00:10:11:01:00)$.

Let $H_i(s,t),\ 0 \leq i \leq h-1$, be the Hamming distance between s and t in $M[i]$, i.e. the number of bits with distinct values in $M_s[i]$ and $M_t[i]$. From the algorithm, the longest length of the routing path is $2^k + H_h(s,t)$, where $H_h(s,t) = \sum_{i=0}^{h-1} H_i(s,t)$. This formula gives an upper bound to $d(s,t)$, the distance between s and t in an MC(k,m). Let $H(s,t)$ be the Hamming distance between s and t. Clearly, we have $H(s,t) \leq d(s,t) \leq H_h(s,t) + 2^k$. Because $H(s,t) = H_h(s,t) + H_k(s,t)$, where $H_k(s,t)$ is the Hamming distance between s and t in c field, we have $H(s,t) \leq d(s,t) \leq H(s,t) - H_k(s,t) + 2^k$. The longest path in an MC(k,m) is from $s = 0\cdots0$ to t, where $c_t = 0\cdots0$ and $M_t[i] = 1\cdots1$ for all $i,\ 0 \leq i \leq h-1$. The length of this path is $2^k(m+1)$. It is easy to see that this path is the shortest path for connecting s and t. Therefore, it is the diameter of an MC(k,m). Since the average distance in each cluster is $m/2$, the average distance between any two nodes in an MC(k,m) is at most $(m/2)2^k + 2^k = (n-k)/2 + 2^k$, where $n = m2^k + k$ (in the case of Hamiltonian path, it is $(n-k)/2 + 2^k - 1$). Notice that it is possible to have a routing algorithm in an MC(k,m) which bypasses the class c if $M_s[c] = M_t[c]$. In such a case, the length of the routing path for some s and t might be shorter than that produced by the algorithm above. We put these results into the following theorem.

Theorem 1. *In an MC(k,m), let $d(s,t)$ and $d_{avg}(s,t)$ be the distance and the average distance between any two nodes s and t, respectively. Let $H(s,t)$ be the Hamming distance between s and t. Then $d(s,t) \leq H(s,t) - H_k(s,t) + 2^k$ and $d_{avg} \leq (n-k)/2 + 2^k$, where $H_k(s,t)$ is the Hamming distance between s and t in the class field. The diameter of an MC(k,m) is $2^k(m+1)$.*

3 Disjoint Paths in Metacube

In this section, we will describe an algorithm for finding $k+m$ disjoint paths from a source node s to a destination node t in MC(k,m). For a node u in an MC(k,m), we denote $u^{(i)},\ 0 \leq i \leq k-1$, as global neighbors; and $u^{(k+j)},\ 0 \leq j \leq m-1$, as local neighbors. The ideas for constructing the disjoint paths from s to t are

1. The local neighbor of s will be connected to the local neighbor of t and the global neighbor of s to the global neighbor of t.
2. The locality between s and t is to be considered while constructing the disjoint paths. First, consider the case that s and t are in the same cluster. Second, s and t are in the clusters of the same class. Third, s and t are in the clusters of different classes.

The first case is the simplest one. The k paths are constructed such that $s^{(i)}$ is connected to $t^{(i)},\ 0 \leq i \leq k-1$, through distinct clusters of the same class. The m paths are constructed inside the cluster using the hypercube algorithm.

In the second case that s and t are in the distinct clusters of the same class, a problem for constructing the disjoint path is as follows. When $M_s[u] = M_t[u]$ for some $u,\ 0 \leq u \leq 2^k - 1$, the path $(s \rightarrow t)$ will just advance through the cross-edges of a hamiltonian path and do nothing. This will cause two paths along the distinct dimensions in the k-cube to intersect at some vertex. For example, in an MC$(2,2)$, two paths from $s = 0000000000$ to $t = 1111000000$ along different class-paths $(00:01:11)$ and $(00:10:11)$ will meet at a common vertex 1100000000 before reach t. To guarantee the k paths are disjoint while using hamiltonian cycles to modify the $M[*]$ fields, we adopt the idea of *signature*. We assign each path a unique signature defined through a *key-bit*. A key-bit is a bit in a node address. It will be assigned to each of the k neighbors of node $s,\ s^{(i)},\ 0 \leq i \leq k-1$, a signature that is unique to the path through that neighbor before applying the point-to-point routing algorithm using a Hamiltonian cycle. If we say "the key-bit is at the dimension x", it means that we will negate the value of the key-bit of a node to get the address of that nodes' xth dimensional neighbor. The $k+m$ dimensions of an MC(k,m)

are $0,1,\ldots,k-1,k,k+1,\ldots,k+m-1$, where the first k dimensions are in the class field c_u and the next m dimensions are in the field $M[c_u]$, counted from right to left. The key-bit is a bit in $M[c_u]$ defined in a way that the path holds a unique bit-pattern of $M[c_u]$ after negating that bit.

We use $c_s^{(i)}$ and $c_t^{(i)}$ to denote $c_{s^{(i)}}$ and $c_{t^{(i)}}$, respectively. The key-bit can be determined as below. For the ith disjoint path ($0 \le i \le k-1$), if we can find a bit so that $M_s[c_s^{(i)}]$ and $M_t[c_t^{(i)}]$ in that bit have the same value, then let that bit be the key-bit (*type* 1); otherwise, take any bit as the key-bit (*type* 2). The idea behind this is to enforce a signature (negating the key-bit value) before applying the point-to-point routing algorithm. For the ith disjoint path ($k \le i \le m+k-1$), the ith bit can serve as a key-bit since it is unique to the path P_i through $s^{(i)}$ except the case $H(M_s[c_s], M_t[c_t]) = 1$. If $M_s^{(j)}[c_s] = M_t[c_t]$ then the bit j cannot be used as a key-bit since $M_t[c_t]$ is no longer unique to the path through $s^{(j)}$. Therefore, in our algorithm, we let P_j goes through $s^{(j,j')}$ for some $j', k \le j' \le m+k-1$, and then connected to $t^{(j')}$. The Path $P_{j'}$ will go through $s^{(j')}$ and then connected to $t^{(j)}$. After completing the hamiltonian cycle, the m paths will be back to the cluster C_t. Let the node for P_i after completing the hamiltonian cycle be w_i (w_i differs with t in field $M[c_t]$ only). Then, the subpaths $w_i \to t$ should be disjoint paths in C_t. This can be done through algorithm 3 with $n = m$.

In the last case that s and t are in the clusters of different classes, for constructing the k disjoint path, we connect $s^{(i)}$ to $t^{(i)}$, $0 \le i \le k-1$. This is done through two two subpaths, $s^{(i)} \to w_i$ and $w_i \to t$, where w_i differs with t in class field only. Constructing $s^{(i)} \to w_i$ is similar to that of the second case since the *classID* of nodes $s^{(i)}$ and w_i are the same. The path $w_i \to t$ contains cross-edges only. Since no any signature can apply beyond w_i, we need to find k disjoint paths from w_i to t. This can be done by algorithm 3 with $n = k$. Notice that if $c_s^{(i)} = c_t$ then the path $s^{(i)} \to t$ requires special handling as shown in the algorithm. Constructing m disjoint paths follows the similar strategy as in the previous case. However, as shown in the algorithm, if $s^{(i)} = w_j$, $i \ne j$, we should construct paths $s^{(i)} \to t^{(j)}$ and $s^{(j)} \to t^{(i)}$ instead of $s^{(i)} \to t^{(i)}$ and $s^{(j)} \to t^{(j)}$.

Let x and y be two nodes in an n-cube and $d = |x \oplus y|$ be the Hamming distance between x and y. Let $Z_i = (x^{(i)} \to y)$, $0 \le i \le n-1$, are n disjoint paths in an n-cube, then there are d paths of length $(d-1)$ and $(n-d)$ paths of length $(d+1)$. The n disjoint paths $Z_i = (x^{(i)} \to y)$, $0 \le i \le n-1$, can be constructed by Algorithm 2.

Algorithm 2 (CubeDisjointPaths(n,x,y))
begin
  for $i = 0$ to $n-1$ **do** /* for each path */
    $v = x^{(i)}$; $P_i = v$; $u = v \oplus y$;
    p3(n,i,v,y);
  endfor
end

Algorithm 3 (p3(n,i,x,y))
begin
  $v = x$;
  for $j = 1$ to n **do** /* for each dimension */
    $b = (i+j)$ % n;
    if $(u \,\&\, 2^b) \ne 0$, **then** $v = v^{(b)}$; $p3 = (p3 : v)$;
  endfor
end

In a 3-cube, let $x = 000$, 3 disjoint paths for every y are shown in Table 1. Notice that we also listed the case of $y = x = 000$, it may appear in the case of $c_s = c_t$ when we build $k+m$ disjoint paths in an MC(k,m).

Table 1. Hypercube disjoint path examples

(000, 000)			(000, 001)			(000, 010)			(100, 000)		
P_0	P_1	P_2	P_0	P_1	P_2	P_0	P_1	P_2	P_0	P_1	P_2
000	000	000	000	000	000	000	000	000	000	000	000
001	010	100	001	010	100	001	010	100	001	010	100
000	000	000		011	101	011		110	101	101	
				001	001	010		010	100	100	

(000, 011)			(000, 101)			(000, 110)			(000, 111)		
P_0	P_1	P_2	P_0	P_1	P_2	P_0	P_1	P_2	P_0	P_1	P_2
000	000	000	000	000	000	000	000	000	000	000	000
001	010	100	001	010	100	001	010	100	001	010	100
011	011	101	101	110	101	011	110	110	011	110	101
		111		111		111			111	111	111
		011		101		110					

The algorithm 2 can be used for constructing k or m disjoint paths $(u \to v)$ in MC(k,m) as follows. When u and v differ only in *classID* we can call algorithm 3, p3(k,i,c_u,c_v) to find a class-path in the class-cube. In our algorithm for disjoint paths in MC(k,m), we make no distinction between the path $u \to v$ in MC(k,m) and the the class-path $c_u \to c_v$ while there is no confusion arises. Similarly, when u and v differ only in *nodeID* (in the same cluster) we can call algorithm 3, p3($m,i,M_u[c_u],M_v[c_v]$) to generate a shortest path in a k-cube, and we identify this path as the path $(u \to v)$ in the cluster C_s $(= C_t)$.

Example 2: Assume $m = k = 2$, $s =$ 0000000000, and $t =$ 0000000001. Since $C_s = C_t$, we construct the paths by **case 0**. The four paths are shown in Table 2. The longest path is P_0 or P_1. $|P_0| = |P_1| = H(u,v)+6 = 7$.

Let the two clusters be $C_s = (c_s, M_s[h-1],\ldots,M_s[c_s+1],*,M_s[c_s-1],\ldots,M_s[0])$ and $C_t = (c_t, M_t[h-1],\ldots,M_t[c_t+1],*,M_t[c_t-1],\ldots,M_t[0])$. Let HC_i be a Hamiltonian cycle in H_k containing the directed edge $(s : s^{(i)})$. In what follows, we give an algorithm (Algorithm 4) for constructing $k+m$ disjoint paths from s to t in an MC(k,m). In the algorithm, we first find path P_i for $0 \le i \le k-1$ and then find path P_i for $k \le i \le k+m-1$. A key-bit position y is determines by the following rule. Try to find a 0 in $M_s[c_s^{(i)}] \oplus M_t[c_t^{(i)}]$, $0 \le i \le k-1$, from rightmost bit. If success, let $x =$ the bit position; otherwise, $x = 0$. Then $y = k+x$. We use l to denote the bit position so that $(u : u^{(l)})$ is an one-step path

Algorithm 4 (MetacubeDisjointPaths(k,m,s,t))

begin /* find $k+m$ disjoint paths P_i, $0 \le i \le k+m-1$, from node s to node t in an MC(k,m). */

case 0 ($C_s = C_t$):

$0 \le i < k$:

$P_i = (s : s^{(i)} : s^{(i,k)} : (s^{(i,k,i)} \to t^{(i,k,i)}) : t^{(i,k)} : t^{(i)} : t)$,

where $(s^{(i,k,i)} \to t^{(i,k,i)}) \leftarrow p3(k,i,s^{(i,k,i)},t^{(i,k,i)})$;

$k \le i < k+m$:

$P_i = p3(m,i-k,s^{(i)},t)$;

case 1 ($C_s \neq C_t$ and $c_s = c_t$):

$0 \le i < k$:

$P_i = (s : s^{(i)} : s^{(i,y)} : (s^{(i,y,l)} \to t^{(i)}) : t)$, /* y is a key-bit. */

where $(s^{(i,y,l)} \to t^{(i)}) = p1(k,m,s^{(i,y,l)},t^{(i)})$;

$k \le i < k+m$:

$w_i = (c_t, M_t[h-1],\ldots,M_t[c_t+1],M_s^{(i)}[c_t],M_t[c_t-1],\ldots,M_t[0])$;

$P_i = (s : s^{(i)} : (s^{(i,l)} \to w_i \to t))$,

where $(s^{(i,l)} \to w_i) = p1(k,m,s^{(i,l)},w_i)$ and $(w_i \to t) = p3(m,i-k,w_i,t))$;

if $(M_s^{(j)}[c_s] = M_t[c_t])$

$P_j = (s : s^{(j)} : s^{(j,j')} : (s^{(j,j',l)} \to t^{(j')}) : t)$,

where $(s^{(j,j',l)} \to t^{(j')}) = p1(k,m,s^{(j,j',l)},t^{(j')})$;

$P_{j'} = (s : s^{(j')} : (s^{(j',l)} \to t^{(j)}) : t)$,

where $(s^{(j',l)} \to t^{(j)}) = p1(k,m,s^{(j',l)},t^{(j)})$;

endif

case 2 ($c_s \neq c_t$):

$0 \le i < k$:

$w_i = (c_s^{(i)}, M_t[h-1],\ldots,M_t[0])$.

if $(M_s[j] = M_t[j]$ **for all** $j \neq c_s^{(i)})$

$(s^{(i)} \to w_i) = p1(k,m,s^{(i)},w_i)$

else $(s^{(i)} \to w_i) = (s^{(i)} : s^{(i,y)} : p1(k,m,s^{(i,y,l)},w_i))$;

endif

$(w_i \to t) = p3(m,i,w_i,t)$;

$P_i = (s : (s^{(i)} \to w_i \to t))$;

if $(c_s^{(j)} = c_t)$

$(s^{(j)} \to t^{(j)}) = p1(k,m,s^{(j)},t^{(j)})$;

$P_j = (s : (s^{(j)} \to t^{(j)}) : t)$;

endif

$k \le i < k+m$:

$w_i = (c_s, M_t[h-1],\ldots,M_t[c_t+1],M_t^{(i)}[c_t],M_t[c_t-1],\ldots,M_t[0])$;

if $(M_s^{(i)}[c_s] = M_t[c_s])$

$(s^{(i)} \to w_i) = (s^{(i)} : s^{(i,i')} : p1(k,m,s^{(i,i',l)},w_i))$

else $(s^{(i)} \to w_i) = (s^{(i)} : p1(k,m,s^{(i,l)},w_i))$;

endif

$(w_i \to t^{(i)}) = p3(k,0,w_i,t^{(i)})$;

$P_i = (s : (s^{(i)} \to w_i \to t^{(i)}) : t)$

if $(w_i = s^{(j)})$

if $(M_s^{(i)}[c_s] = M_t[c_s])$

$(s^{(i)} \to w_j) = (s^{(i)} : s^{(i,i')} : p1(k,m,s^{(i,i',l)},w_j))$

else $(s^{(i)} \to w_j) = (s^{(i)} : p1(k,m,s^{(i,l)},w_j))$;

endif

$(w_j \to t^{(j)}) = p3(k,0,w_j,t^{(j)})$;

$P_i = (s : (s^{(i)} \to w_j \to t^{(j)}) : t)$;

$P_j = (s : (s^{(j)} \to t^{(i)}) : t)$;

endif

end

in HC_i. $p1(k,m,u,v)$ gets a path from node u to node v in an MC(k,m) by calling Algorithm 1 and $p3(n,i,u,v)$ gets a path containing cross-edges only, from node u to node v in an MC(k,m) by calling Algorithm 3.

Table 2. Example 2 (0000000000, 0000000001)

P_0 $(i=0)$	P_1 $(i=1)$	P_2 $(i=2)$	P_3 $(i=3)$
000000**0**00	0000**0**00000	0000000000	0000000000
010000**0**00	1000**0**00000	0000000001	0000000010
010000**1**00	1000**1**00000		0000000011
000000**1**00	0000**1**00000		0000000001
000000**1**01	0000**1**00001		
010000**1**01	1000**1**00001		
010000**0**01	1000**0**00001		
000000**0**01	0000**0**00001		

Table 3. Example 3 (0000000000, 0001011101)

P_0 $(i=0)$	P_1 $(i=1)$	P_2 $(i=2)$	P_3 $(i=3)$
000000**0**00	0000**0**00000	0000000000	0000000000
010000**0**00	1000**0**00000	0000000001	0000000010
010000**1**00	1000**1**00000	0000000011	0100000010
110000**1**00	1100**1**00000	0100000011	0100000110
110100**1**00	1101**1**00000	0100000111	0100001110
100100**1**00	0101**1**00000	0100001111	1100001110
100101**0**00	0101**1**00100	1100001111	1101001110
000101**0**00	0101**1**01100	1101001111	1001001110
000101**0**01	0001**1**01100	1001001111	1001011110
010101**0**01	0001**1**01101	1001011111	0001011110
010101**1**01	1001**1**01101	0001011111	0001011100
000101**1**01	1001**1**11101	0001011101	0001011101
	1001**0**11101		
	0001**0**11101		

Table 4. Example 4 (0000000000, 0100001111)

P_0 $(i=0)$	P_1 $(i=1)$	P_2 $(i=2)$	P_3 $(i=3)$
0000000000	0000**0**0000	0000000000	0000000000
0100000000	1000**0**0000	0000000001	0000000010
0100000100	1000**1**0000	0100000001	0100000010
0100001100	1100**1**0000	0100001001	0100000110
1100001100	0100**1**0000	1100001001	1100000110
1000001100	0100**1**0100	1000001001	1000000110
0000001100	0100**1**1100	0000001001	0000000110
0000001101	0000**1**1100	0000001011	0000000111
0000001111	0000**1**1101	0100001011	0100000111
0100001111	0000**1**1111	0100001111	0100001111
	1000**1**1111		
	1000**0**1111		
	1100**0**1111		
	0100**0**1111		

Table 5. Example 5 (0000000001, 1101000000)

P_0 $(i=0)$	P_1 $(i=1)$	P_2 $(i=2)$	P_3 $(i=3)$
000000**0**01	0000**0**0001	0000000001	0000000001
010000**0**01	1000**0**0001	0000000000	0000000011
010000**1**01	1000**1**0001	0100000000	0100000011
110000**1**01	1100**1**0001	1100000000	1100000011
110100**1**01	1101**1**0001	1101000000	1101000011
100100**1**01	0101**1**0001		1111000011
000100**1**01	0001**1**0001		1011000011
000100**0**00	0001**0**0000		0011000011
010100**0**00	1001**0**0000		0011000010
010100**0**00	1001**0**0000		0011000000
110100**0**00	1101**0**0000		0111000000
			1111000000
			1101000000

Example 3: Assume $m=k=2$, $s=0000000000$, and $t=0001011101$. Since $c_s=c_t$ and $C_s\neq C_t$, we construct the paths by **case 1**. First, consider the construction of P_0 and P_1. Since $M_s[1]=00$ and $M_t[1]=11$, we choose $y=2$ for $i=0$ as a key-bit of type 2. Since $M_s[2]=00$ and $M_t[2]=01$, we choose $y=3$ for $i=1$ as a key-bit of type 1. Next, consider paths P_2 and P_3. Since $H(M_s[c_s],M_t[c_t])=1$, from the algorithm we have $j=2$ and $j'=3$. The four paths from node s to node t are shown in Table 3. The longest path is P_1. $|P_1|=H(0,93)+2^2+4=13$.

Example 4: Assume $m=k=2$, $s=0000000000$, and $t=0100001111$. Since $c_s\neq c_t$ we construct path P_0 and P_1 by **case 2**. Since $H(c_s,c_t)=1$, no signature is assigned for P_0. For P_1, since $M_s[2]=M_t[2]=00$ we assign $y=2$. The four paths from node s to node t are shown in Table 4. The longest path is P_1. $|P_1|=H(0,15)+2^2+4+|Z_1|=13$.

Example 5: Assume $m=k=2$, $s=0000000001$, and $t=1101000000$. We construct the paths P_0 and P_1 by **case 2** and choose $y=2$ for both $i=0$ and 1. Since $c_s^{(0)}=c_t^{(1)}$ and $c_s^{(1)}=c_t^{(0)}$, we have $|Z_0|=|Z_1|=1$. The four paths from node s to node t are shown in Table 5. The longest path is P_3. $|P_3|=H(1,64)+2^2+4+2=12$, where $d(s^{(i)},t^{(i)})=d(s,t)+4$, and the value 2 is from the length of the class-path (00:01:11).

Now we are ready to give the main result of this paper in the following theorem.

Theorem 2. *Given any two distinct nodes s and t in MC(k,m), we can find $k+m$ disjoint paths from s to t in $O(\log^2 p)$ time such that the length of the paths is at most $H(s,t)+2^k+m+5$, where p is the number of nodes in MC(k,m), and $H(s,t)$ is the Hamming distance between s and t.*

Proof: Following the algorithm, the proof is divided into three cases. In the first case, since the k paths constructed outside the cluster C_s go through distinct clusters and therefore, cannot intersect each other or the m paths inside the cluster constructed by the hypercube algorithm. The m paths in the cluster are disjoint since the cluster is an m-cube.

In the second case, if P_i has a key-bit of type 1 then it is clear that they cannot intersect with any other paths since no other path will change the value of that key-bit following the shortest path principle. If the key-bits of the

two paths P_i and P_j are of type 2, from the definition of type 2 key-bit, we know that the two paths cannot intersect each other when passing the nodes in the hamiltonian cycle that are neither $c_s^{(i)}$ nor $c_s^{(j)}$. At node $c_s^{(i)} \in H_k$, path P_j updates the values in field $M[c^{(i)}]$ and the values in the field $M[c^{j)}]$ was partially updated only (the key-bit changed and other bits unchanged). However, the values in the field $M[c^{(j)}]$ for P_i is either unchanged or fully updated. Therefore, two paths cannot meet at the nodes of class $c^{(i)}$. Similarly, they cannot meet at the nodes of class $c_s^{(j)}$. Similar argument can be applied to the cases $k \le i, j \le k+m-1$, or P_i (P_j) has key-bit of type 2 and $k \le j \le k+m-1$ ($k \le i_1 \le k+m-1$, assuming $H(s,t) > 1$. If $H(s,t) = 1$ and $M_s^{(j)}[c_s] = M_t[c_t]$ then the algorithm uses $j \neq j$ as a key-bit. The two paths $s\ (j, j') \to t^{(j')}$ and $s^{(j')} \to t^{(j)}$ constructed in the algorithm are disjoint though both use j' as a key-bit. We conclude that the $m+k$ paths are disjoint in this case.

In the third case, consider first $P_i,\ 0 \le i \le k-1$. We divide P_i into to parts, $(s^{(i)} \to w_i)$ and $(w_i \to t)$. Since the first part of the path includes a signature (except P_q in case 2.1), they should be disjoint following the same argument as in the second case (the argument is true even one of the paths does not carry signature). Moreover, it is also disjoint with the second part of other paths because of its unique signature. The second parts of the paths $P_i,\ 0 \le i \le k-1$, contain only the cross-edges that are identical to the disjoint class paths Z_i, they are also disjoint. Therefore, the k paths constructed by the algorithm are disjoint. Next, consider $P_i,\ k \le i \le m+k-1$. Since both $(s^{(i)} \to w_i)$ and $(w_i \to t^{(i)})$ contain the naturally embedded signature on dimension $i-k$ at field $M[c_s]$ or $M[c_t]$ assuming $M_s^{(i)}[c_s] \neq M_t[c_t]$ and $w_i \neq s^{(j)}$, by applying the similar argument as in the second case, P_i cannot intersect with any other $P_{i'}$ with $0 \le i' \le m+k-1, i' \neq i$. If $w_i = s^{(j)}$ then we have $P_i = s^{(i)} \to t^{(j)}$ and $P_j = s^{(j)} \to t^{(i)}$. If $M_s^{(i)}[c_s] = M_t[c_t]$ then we use $i' \neq i$ as a key-bit. It is easy to see that the above arrangement does not affect the correctness of the previous argument for P_i and P_j to be disjoint with other paths. Therefore, we conclude that all $k+m$ paths constructed in this case are disjoint.

The length of the longest path $(s \to t)$ in the first case is $H(s,t)+6$, where $H(s,t)$ is Hamming distance between s and t. The length of the longest path $(s \to t)$ in the second case is $H(s,t)+2^k+4$. Similarly, the longest length of the paths $(s \to t)$ in the third case is $H(s,t)+2^k+4+\max_{0 \le i \le k-1}\{|Z_i|\} \le H(s,t)+2^k+m+5$. □

4 Concluding Remarks

In this paper, we gave an algorithm for finding the $k+m$ disjoint paths in the metacube. The metacube can be used as an interconnection network for very large scale parallel computers connecting hundreds of millions nodes with up to 6 links per node. For this reason, the issues of disjoint paths and fault-tolerance in metacube is very important. From the disjoint paths algorithm, any two nonfaulty nodes in a metacube with up to $k+m-1$ faulty nodes can be connected by a faulty-free path in the metacube.

Recently, much of the community has moved on to lower-dimensional topologies such as meshes and tori. However, the SGI Origin 2000, a fairly recent multiprocessor, does use a hypercube topology so a metacube could be of use to industry. In order to connect 128 processors with 6-link routers, an Origin2000 uses Cray Router. Furthermore, multiple Origin2000 systems are connected into an array to connect more than 128 processors. By using the metacube architecture, the current 6-link routers can connect much more processors directly without using Cray Router. Generally, the metacube system can be built with the same technology adopted in the Origin2000 system: the key point is to design a 6-link router and connect router board ports with link cables. Any application runnable on Origin2000 is possibly runnable on metacube.

References

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

[2] Q.-P. Gu and S. Peng. Optimal algorithms for node-to-node fault tolerant routing in hypercubes. *The Computer Journal*, 39(7):626–629, 1996.

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

[4] J. Laudon and D. Lenoski, The SGI Origin 2000: A cc-NUMA Highly Scalable Server. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 241–251, June 1997.

[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, December 2000.

[6] Y. Li and S. Peng. Fault-tolerant routing and disjoint paths in dual-cube: a new interconnection network. In *Proceedings of the 2001 International Conference on Parallel and Distributed Systems*, pages 315–322. IEEE Computer Society Press, June 2001.

[7] Y. Li, S. Peng, and W. Chu. Metacube – a new interconnection network for large scale parallel systems. *Australian Computer Science Communications*, 24(3):29–36, Jan. 2002.

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

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

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

[11] B. Vanvoorst, S. Seidel, and E. Barscz. Workload of an ipsc/860. In *Proceedings of the Scalable High-Performance Computing Conference*, pages 221–228, 1994.

[12] 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.