Third International Conference on Parallel and Distributed Computing, Applications and Technologies, Kanazawa, Japan, September 2002, pp.343–350

Fault-tolerant Routing in Metacube

Yamin Li, Shietung Peng
Department of Computer Science
Hosei University
Tokyo 184-8584 Japan
{yamin,speng}@k.hosei.ac.jp

Wanming Chu
Department of Computer Hardware
University of Aizu
Aizu-Wakamatsu 965-8580 Japan
w-chu@u-aizu.ac.jp

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 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 (classes) and m is the dimension of the low-level cubes (clusters). In this paper, we give an efficient algorithm for fault-tolerant routing in MC networks. The fault-tolerant routing problem in MC(k,m) is solved through a special structure in an MC network, called multi-channel cube. In order to construct k disjoint paths for each node pair in a multi-channel cube, an innovative technique, called signature, is introduced.

1. Introduction

The hypercube has been widely used as the interconnection network in a wide variety of parallel systems such as Intel iPSC [12], the nCUBE [5], the Connection Machine CM-2 [11], and SGI Origin 2000 [10]. 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 [10]. 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 the hypercube's topological properties.

Several variations of the hypercube have been proposed in the literature. Some variations focused on reduction of the hypercube diameter, for example the folded hypercube [1] and crossed cube [2]; some focused on reduction of the number of edges of the hypercube, for example cube-connected cycles [9] and reduced hypercube [13]; and some focused on both, as in the hierarchical cubic network [3]. One major property of the hypercube is: there is an edge between two nodes only if their binary addresses differ in a single bit. This property is at the core of many algorithmic designs for efficient routing and communication in hypercubes. In this paper, we refer to it as the key property. Generally, variations of the hypercube that reduce the diameter, e.g. crossed cube and hierarchical cubic network, will not satisfy this key property.

Recently, Y. Li et al. introduced a new interconnection network, called *metacube*, or MC network [8]. 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.

In this paper, we give efficient an algorithm for fault-

tolerant routing in metacube. The remainder of this paper is organized as follows. Section 2 introduces the MC network and its topological properties. Section 3 gives the routing algorithm in metacube. Section 4 introduces a multi-channel cube structure which is used for fault-tolerant routing in the metacube. Sections 5 gives the fault-tolerant routing algorithm. Section 6 concludes the paper and presents some future research directions.

2. Preliminaries

This section formally introduces the MC network, its topological properties and some related notation. The MC network is motivated by the dual-cube network proposed by Li and Peng [6] [7] 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 (*class_id*). 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 (*node_id*). Therefore, the $(mh+k)$-bit node address in an MC(k,m) is divided into three parts: a k-bit *class_id*, an $m(h-1)$-bit *cluster_id* and an m-bit *node_id*.

In the following discussion, we use $u = (c_u, M_u[h-1], \ldots, M_u[1], M_u[0])$ to denote the address 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 $class\_id(u) = c_u$, $node\_id(u) = M_u[c_u] \times 2^{c_u}$ and $cluster\_id(u) = \sum_{i=0}^{h-1} M_u[i] \times 2^i - node\_id(u)$. The mh-bit number $node\_id(u) + cluster\_id(u)$ 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 $cluster\_id = 48$.

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-edges*. 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.

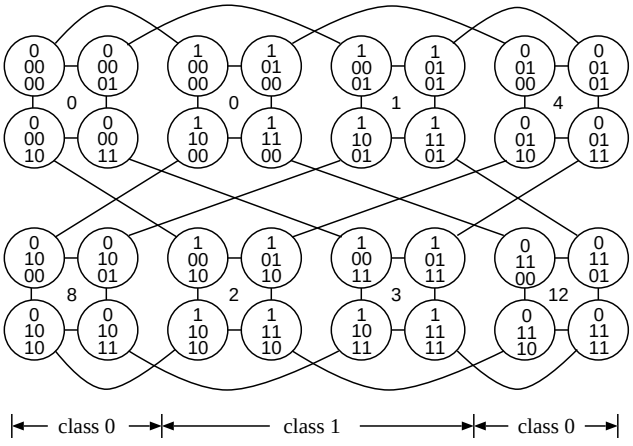

Figure 1. A metacube MC(1,2)

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 (01,111,101,111,000), (01,111,101,100,000) and (01,111,101,010,000). 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 (00,111,101,110,000) and (11,111,101,110,000).

Fig. 1 shows the structure of an MC$(1,2)$, where the cluster is a 2-cube and there are two classes. Each node has a

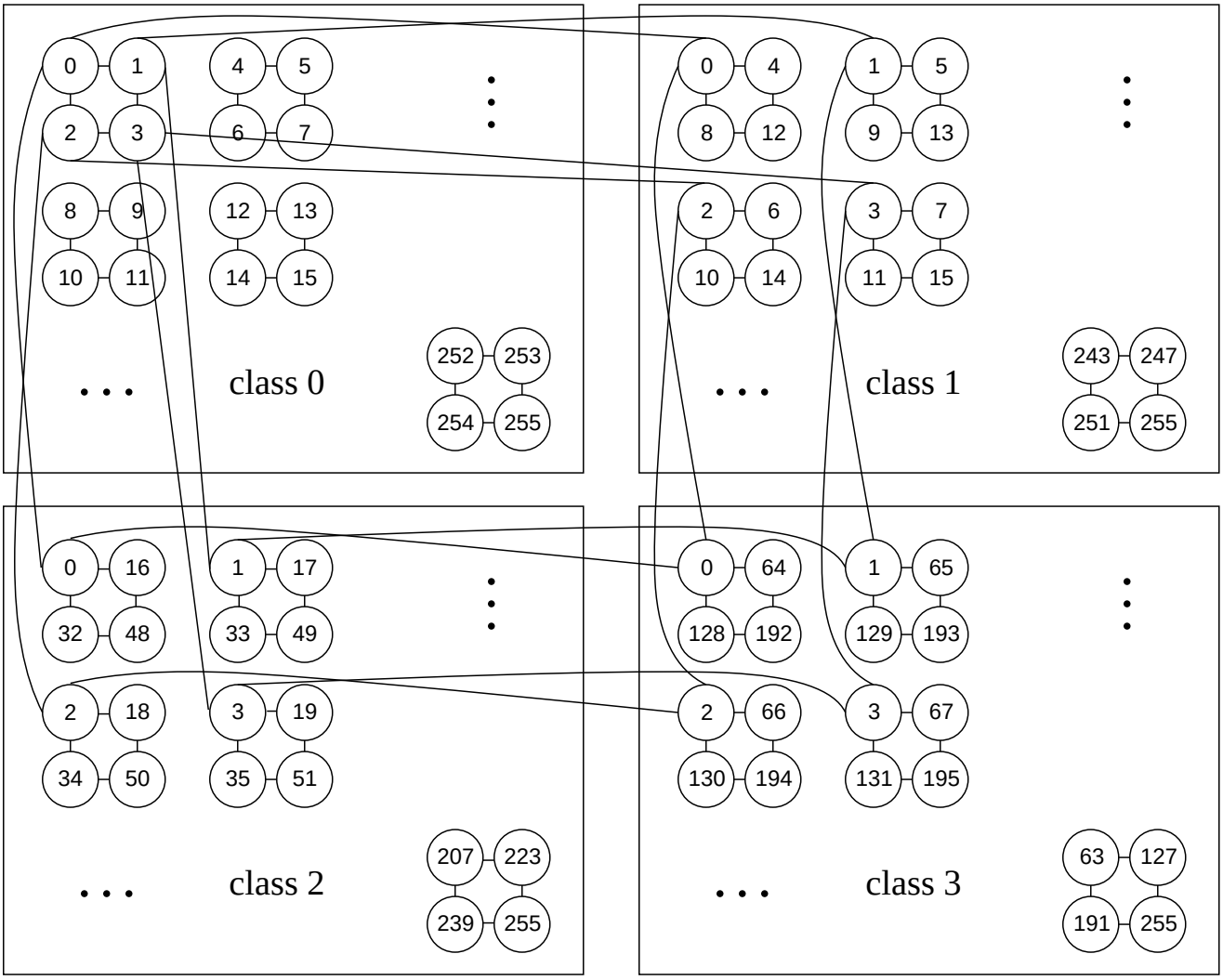

Figure 2. A metacube MC(2,2)

cross-edge attached to a node of the different class. The binary number shown in the center of a cluster is *cluster_id*. 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 $node\_id + cluster\_id$. 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 ratio of the total number of links in the hypercube to the total number of links in the MC network is equal to $n/(m+k)$, where $n = m2^k + k$. For example, for $k = 2$ and $m = 3$ ($n = 14$), each of the two networks contains 16384 nodes; the hypercube contains $16384 \times 14/2 = 114688$ links and the MC network contains $16384 \times (3+2)/2 = 40960$ links. The reduction in the total number of links for this example is 73728 links or about 64%.

3. Point-to-point routing in metacube

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. In this section, we describe briefly the point-to-point routing algorithm in metacube [8]. This algorithm is the building block for the proposed fault-tolerant routing algorithm.

We adopt the following notation. In the metacube MC(k,m), each node has $m+k$ neighbors. Let $s^{(i)}$, $0 \leq i \leq k-1$, 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 $s^{(i+k)}$, $0 \leq i \leq m-1$, be the ith dimensional 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 an *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. It was shown in [8] that given any two nodes s and t in an n-cube, there exists a weak-hamiltonian path from s to t.

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

In the fault-tolerant routing we discuss late, we need directed hamiltonian cycles. Therefore, we introduce a parameter, $next(s)$, to assign the direction of a hamiltonian cycle. For example, in Example 1, if we let $next(s) = 10$, then the hamiltonian cycle for $(C_s \to C_t)$ in the high-level 2-cube will be $(00 : 10 : 11 : 01 : 00)$.

Let $H_i(s,t)$, $0 \le i \le h-1$, 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 $H(s,t) \le d(s,t) \le H_h(s,t) + 2^k$. 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 $H(s,t) \le d(s,t) \le H(s,t) - H_k(s,t) + 2^k$. The longest path in an MC(k,m) is from $s = 0 \cdots 0$ to t, where $c_t = 0 \cdots 0$ and $M_t[i] = 1 \cdots 1$ for all i, $0 \le i \le h-1$. 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.

4. Multi-channel cube

In this section, we will describe a structure in the metacube, called *multi-channel cube*. This structure is useful for designing algorithms in MC(k,m) based on hypercube algorithms. We will use this structure for solving fault-tolerant routing problem in the metacube.

Let C_u and C_v be two distinct clusters in an MC(k,m), $h = 2^k$. A multi-channel cube is defined as $C_u \cup C_v \cup E$, where E is a set of $k2^m$ disjoint paths connecting C_u and C_v, k paths for each node pair (u_i, v_i), $u_i \in C_u$, $v_i \in C_v$, for $i = 0, 1, \ldots, 2^m - 1$. In the other word, $E = \bigcup_{i,j} (u_i \to v_i)_j$, $0 \le i \le 2^m - 1$ and $0 \le j \le k-1$, and $(u_{i_1} \to v_{i_1})_{j_1} \cap (u_{i_2} \to v_{i_2})_{j_2} = \emptyset$ if $i_1 \neq i_2$ or $j_1 \neq j_2$.

Figure 3 shows an example of a multi-channel cube. The dotted line denotes a path, not an edge. In the case of a traditional n-cube, there is an edge (u, v) connecting two nodes u and v for $u \in$ 0-subcube and $v \in$ 1-subcube. Instead of an edge, the two nodes of a node pair in a multi-channel cube are connected by k disjoint paths.

The major problem in constructing the k disjoint paths is that the path $(u_i \to v_i)_j$, $0 \le i \le 2^m - 1$ and $0 \le j \le k-1$, will just advance through cross-edges when $M_{u_i}[l] = M_{v_i}[l]$ for some l, $0 \le l \le 2^k - 1$. This will cause two paths

[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.

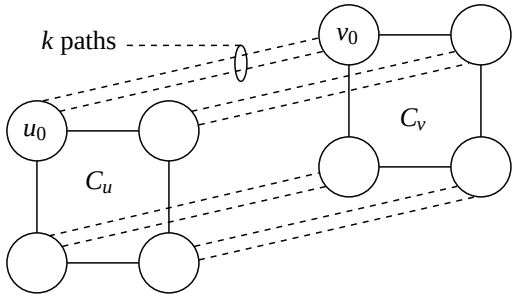

Figure 3. A multi-channel cube

along the distinct dimensions in the k-cube to intersect at some vertex. For example, in an MC$(2,2)$, the paths from 0000000000 to 1111000000 along dimensions 0 and 1 will meet at a common vertex of 1100000000. To guarantee the k paths are disjoint, each path needs 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 u_i to carry the 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, negating the value of the key-bit of a node will get the address of that nodes' xth dimensional neighbor. The $k+m$ dimensions of an MC(k,m) are $0,1,\ldots,k-1,k,k+1,\ldots,k+m-1$.

The key-bit can be determined as below. Notice that all the c_{u_i}, the *class_id* of node $u_i \in C_u$, for $i = 0,1,\ldots,2^m-1$, are the same, so we use c_u to denote c_{u_i}. In such a case, node u may be any of node u_i, $0 \le 2^m-1$. c_v does also. We use $c_u^{(j)}$ and $c_v^{(j)}$ to denote $c_{u^{(j)}}$ and $c_{v^{(j)}}$, respectively. In the case of $c_u = c_v$, if we can find a bit where $M_u[c_u^{(j)}]$ and $M_v[c_v^{(j)}]$ 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 (changing the key-bit value) before applying the point-to-point routing algorithm. In the case of type 1, the key-bit should be removed finally.

In the case of $c_u \neq c_v$, the construction of k disjoint paths has two parts. The first part is the same as the case of $c_u = c_v$ and the second part is to construct the subpaths that contain cross-edges only, since after the first part finished, the updating of fields $M[i]$, $0 \le i \le 2^k-1$, has been done. To guarantee the paths are disjoint, for the second part, we should find k disjoint class-paths in the k-cube. This can be done as the same manner as a traditional hypercube does. We summarize it below.

Let x and y be two nodes in an n-cube and the Hamming distance between x and y, $|x \oplus y| > 1$. Let $Z_j = (x^{(j)} \rightarrow y)$, $0 \le j \le n-1$, are n disjoint paths in an n-cube, d paths are of length $(d-1)$ and $(n-d)$ paths are of length $(d+1)$, where $d = |y \oplus x|$ is the Hamming distance between x and y. Without loss of generality, we assume that the d paths of length $(d-1)$ are $(x^{(j)} \rightarrow y^{((j+1) \bmod d)} : y)$, $0 \le j \le d-1$, and the $(n-d)$ paths of length $d+1$ are $(x^{(j)} \rightarrow y^{(j)} : y)$, $d \le j \le n-1$.

Let the two clusters be $C_u = (c_u, M_u[h-1],\ldots,M_u[c_u+1],*,M_u[c_u-1],\ldots,M_u[0])$ and $C_v = (c_v, M_v[h-1],\ldots,M_v[c_v+1],*,M_v[c_v-1],\ldots,M_v[0])$. Let HC be a hamiltonian cycle in H_k. In what follows, we give an algorithm for pairing up the nodes in C_u and C_v and constructing k disjoint paths for each pair. In the algorithm, the hamiltonian path that follows the direction of $(u_i \rightarrow u_i^{(j)})$ will be used.

Algorithm 2 (Multi_Channel_Cube(C_u, C_v))

begin

Case 1: $c_u = c_v$. We pair up nodes $u_i \in C_u$ and $v_i \in C_v$ so that $M_{u_i}[c_u] = M_{v_i}[c_v]$, for $i = 0,1,\ldots,2^m-1$. Find a 0 in $M_{u_i}[c_u^{(j)}] \oplus M_{v_i}[c_v^{(j)}]$ from rightmost bit. If the bit position be x, let the key-bit is at the dimension $y = k + (x \mod k)$. Then $(u_i \rightarrow v_i)_j = (u_i : u_i^{(j)} : u_i^{(j,y)} : u_i^{(j,y,l)} \rightarrow v_i^{(j)} : v_i)$, where $(u_i^{(j,y)} : u_i^{(j,y,l)})$ is an one-step path in HC and $(u_i^{(j,y,l)} \rightarrow v_i^{(j)})$ is a path constructed by Algorithm 1, for $i = 0,1,\ldots,2^m-1$, and $j = 0,1,\ldots,k-1$.

Case 2: $c_u \neq c_v$. We pair up nodes u_i and v_i so that $M_{u_i}[c_u] \oplus M_{v_i}[c_v] = M_{u_i}[c_v] \oplus M_{v_i}[c_u]$. Let $w_{i,j}$ be of class $c_u^{(j)}$ and its *cluster_id* + *node_id* equal to that of v_i, i.e. $w_{i,j}$ differs with v_i at field c only. In what follows, if $u_i^{(j)} = w_{i,j}$, the path $u_i^{(j)} \rightarrow w_{i,j}$ will be replaced with $u_i^{(j)}$.

Case 2.1: $|c_u \oplus c_v| = 1$. Let the bit position where c_u and c_v have different value be q. We have $w_{i,q} = v_i$ in this case. The path in dimension q is $(u_i : u_i^{(q)} \rightarrow v_i^{(q)} : w_{i,q} = v_i)$, where $(u_i^{(q)} \rightarrow v_i^{(q)})$ is constructed by Algorithm 1. The rest $k-1$ paths are $(u_i : u_i^{(j)} : u_i^{(j,y)} : u_i^{(j,y,l)} \rightarrow w_{i,j} : v_i^{(j)} : v_i)$, $1 \le j \le k-1$, where $u_i^{(j)} : u_i^{(j,y)}$ is a signature, $(u_i^{(j,y)} : u_i^{(j,y,l)})$ is an one-step path in HC, and path $(u_i^{(j,y,l)} \rightarrow w_{i,j})$ is constructed by Algorithm 1, for $0 \le j \le k-1$, $j \neq q$.

Case 2.2: $|c_u \oplus c_v| > 1$. The first d paths are $(u_i : u_i^{(j)} : u_i^{(j,y)} \rightarrow w_{i,j} \rightarrow v_i^{((j+1) \bmod d)} : v_i)$, $0 \le j \le d-1$, where $(u_i^{(j)} : u_i^{(j,y)})$ is a signature, path $(u_i^{(j,y)} \rightarrow w_{i,j})$ is the path constructed by Algorithm 1, and path $(w_{i,j} \rightarrow v_i^{((j+1) \bmod d)} : v_i)$ is constructed by the cross-edges specified by Z_j. The rest $k-d$ paths are $(u_i : u_i^{(j)} : u_i^{(j,y)} \rightarrow w_{i,j} \rightarrow v_i^{(j)} : v_i)$, $d \le j \le k-1$, where $(u_i^{(j)} : u_i^{(j,y)})$ is a signature, path $(u_i^{(j,y)} \rightarrow w_{i,j})$ is the path constructed by Algorithm 1, and path $(w_{i,j} \rightarrow v_i^{(j)} :$

Table 1. Multi-channel examples

Example 1		Example 2		Example 3		Example 4	
(0000000000,0001000000)		(0000000000,0001011100)		(0000000000,0100001111)		(0000000001,1101000000)	
P_0 $(j=0)$	P_1 $(j=1)$	P_0 $(j=0)$	P_1 $(j=1)$	P_0 $(j=0)$	P_1 $(j=1)$	P_0 $(j=0)$	P_1 $(j=1)$
0000000**0**00	00000**0**0000	0000000**0**00	0000**0**00000	0000000000	00000**0**0000	0000000**0**01	00000**0**0001
0100000**0**00	10000**0**0000	0100000**0**00	1000**0**00000	0100000000	10000**0**0000	0100000**0**01	10000**0**0001
0100000**1**00	10000**1**0000	0100000**1**00	1000**1**00000	0100000100	10000**1**0000	0100000**1**01	10000**1**0001
1100000**1**00	11000**1**0000	1100000**1**00	1100**1**00000	0100001100	11000**1**0000	1100000**1**01	11000**1**0001
1101000**1**00	11010**1**0000	1101000**1**00	1101**1**00000	1100001100	01000**1**0000	1101000**1**01	11010**1**0001
1001000**1**00	01010**1**0000	1001000**1**00	0101**1**00000	1000001100	01000**1**0100	1001000**1**01	01010**1**0001
0001000**1**00	00010**1**0000	1001010**1**00	0101**1**00100	0000001100	01000**1**1100	0001000**1**01	00010**1**0001
0101000**1**00	10010**1**0000	0001010**1**00	0101**1**01100	0000001101	00000**1**1100	0001000**1**00	00010**1**0000
0101000**0**00	10010**0**0000	0101010**1**00	0001**1**01100	0000001111	00000**1**1101	0101000**1**00	10010**1**0000
0001000**0**00	00010**0**0000	0101011**1**00	1001**1**01100	0100001111	00000**1**1111	0101000**0**00	10010**0**0000
		0001011**1**00	1001**1**11100		10000**1**1111	1101000**0**00	11010**0**0000
			1001**0**11100		10000**0**1111		
			0001**0**11100		11000**0**1111		
					01000**0**1111		

v_i) is constructed by the cross-edges specified by Z_j.

end.

Example 1: Assume $m = k = 2$. $c_u = c_v = 0$, $C_u = \{0,1,2,3\}$ and $C_v = \{64,65,66,67\}$. According to the algorithm, the four node pairs are $(0,64)$, $(1,65)$, $(2,66)$, $(3,67)$. We only show the two paths for $(0,64)$, so the simplified notations $u = 0$ and $v = 64$ are used (ignored i for $i = 0,1,2,3$). Since $M_u[1] = M_v[1] = 00$ and $M_u[2] = M_v[2] = 00$, we choose $y = 2$ for both $j = 0$ and 1 as a key-bit of type 1. The two paths from node 0 to node 64 of class 0 are shown in Table 1. The key-bit is shown with boldface. $|P_0| = |P_1| = d(u,v) + 4 = 5 + 4 = 9$, where $d(0,64) = H(0,64) + 2^2 = 1 + 4 = 5$.

Example 2: Assume $m = k = 2$. $c_u = c_v = 0$, $C_u = \{0,1,2,3\}$ and $C_v = \{92,93,94,95\}$. $(0,92)$ a pair. Let $u = 0$ and $v = 92$. Since $M_u[1] = 00$ and $M_v[1] = 11$, we choose $y = 2$ for $j = 0$ as a key-bit of type 2. Since $M_u[2] = 00$ and $M_v[2] = 01$, we choose $y = 3$ for $j = 1$ as a key-bit of type 1. The two paths from node 0 to node 92 are shown in Table 1. $|P_0| = d(u,v) + 2 = 10$ and $|P_1| = d(u,v) + 4 = 12$, where $d(0,92) = H(0,92) + 2^2 = 4 + 4 = 8$.

Example 3: Assume $m = k = 2$. $c_u = 0$, $c_v = 1$, $C_u = \{0,1,2,3\}$ and $C_v = \{3,7,11,15\}$. The four pairs of the two clusters are $(0,15)$, $(1,11)$, $(2,7)$, $(3,3)$. To construct the two paths for pair $(0,15)$, we notice that $|c_u \oplus c_v| = 1$ and $c_u^{(0)} = c_v$. Therefore, we construct path P_0 by Case 2.1. We choose $y = 2$ for $j = 1$. The two paths from node $u =$ 0000000000 to node $v =$ 0100001111 are shown in Table 1. $|P_0| = d(u,v) + 1 = 9$, $|P_1| = d(u,v) + 4 + |Z_1| = 8 + 4 + 1 = 13$, where $d(0,15) = H(0,15) + 2^2 = 4 + 4 = 8$.

Example 4: Assume $m = k = 2$. $c_u = 0$, $c_v = 3$, $C_u = \{0,1,2,3\}$ and $C_v = \{0,64,128,192\}$. The four pairs of the two clusters are $(0,0)$, $(1,64)$, $(2,128)$, $(3,192)$. Since $|c_u \oplus c_v| = 2$, we construct path P_0 and P_1 for pair $(1,64)$ by Case 2.2. We choose $y = 2$ for both $j = 0$ and 1. Since $c_u^{(0)} = c_v^{(1)}$ and $c_u^{(1)} = c_v^{(0)}$, we have $|Z_0| = |Z_1| = 1$. The two paths from node $u =$ 0000000001 to node $v =$ 1101000000 are shown in Table 1. $|P_0| = |P_1| = d(u,v) + 4 + |Z_1| = 5 + 4 + 1 = 10$, where $d(1,64) = H(1,64) + 2^2 = 1 + 4 = 5$.

We first prove that the k paths $(u : u^{(j)} \rightarrow v^{(j)} : v)$, $0 \leq j \leq k-1$, constructed in Case 1, are disjoint. For any two paths P_{j_1} and P_{j_2}, if one of them has a key-bit of type 1 then it is clear that they cannot intersect each other since no other path will change the value of that key-bit. If the key-bits of the two paths are of type 2, from the definition of type 2 key-bit, we know that the two paths cannot intersect each other when we use the nodes in the hamiltonian cycle that are not $c_u^{(j_1)}$ nor $c_u^{(j_2)}$. At node $c_u^{(j_1)} \in H_k$, path P_{j_2} updates the values in field $M[c^{(j_1)}]$ and the values in the field $M[c^{j_2)}]$ was partially updated only (the key-bit changed and other bits unchanged), however, the values in the field $M[c^{(j_2)}]$ for P_{j_1} is either unchanged or fully updated. Therefore, two paths cannot meet at the nodes of class $c^{(j_1)}$. Similarly, they cannot meet at the nodes of class $c_u^{(j_2)}$. We conclude that the k paths are disjoint. Next, since each pair (u_i, v_i) has a unique *node_id* and this unique *node_id* will become part of the *cluster_id* of the nodes in path $(u_i^{(p)} \rightarrow v_i^{(q)})$, different paths cannot pass through the same cluster. Therefore, all $k2^m$ paths for connecting C_u and C_v are disjoint for Case 1.

To show the k paths for each node pair constructed in Case 2 are disjoint, we divide each path into to parts, $(u_i^{(j)} \rightarrow w_{i,j})$ and $(w_{i,j} \rightarrow v_i)$. 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 Case 1 (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 $P_j,\ 0 \leq j \leq k-1$, contain only the cross-edges that are identical to the disjoint class paths Z_j, they are also disjoint. Therefore, the k paths constructed by the algorithm are disjoint. To prove that the h sets of paths $(u_i \rightarrow v_i),\ 0 \leq i \leq 2^m - 1$ are disjoint also, we argue as follows: First for each u_i, consider the paths $(u_i \rightarrow u_i^{(j)} \rightarrow w_i \rightarrow w_i')$, where $c_w \neq c_v$ and $c_{next(w_i)} = c_v$, $c_{w_i'} \neq c_v$ and $c_{prev(w_i')} = c_v$. From the algorithm, we know that the h sets of paths $(u_i \rightarrow w_i)$ are disjoint since the value $M_{u_i}[c_u]$ is different for every $u_i \in C_u$ and $M_{u_i}[c_u]$ is part of the *cluster_id* of every node in path $(u_i^{(j)} \rightarrow w_i')$. In path $(w_i \rightarrow w_i')$, the value $M_{u_i}[c_v]$ is changed to $M_{v_i}[c_v]$. Since $M_{u_i}[c_u] \oplus M_{v_i}[c_v] = M_{u_i}[c_v] \oplus M_{v_i}[c_u]$, $M_{v_i}[c_v]$ takes different values for every u_i. That is, node w_i' is in distinct clusters for every u_i. Since $M_{v_i}[c_v]$ is part of the *cluster_id* of every node in path $(w_i' \rightarrow v_i)$, the h sets of paths $(w_i' \rightarrow v_i)$ are also disjoint. Therefore, we conclude that all paths for connecting the clusters C_u and C_v are disjoint for Case 2.

From the path construction in Case 1, the length of the longest path $(u_i \rightarrow v_i)$ is $|(u_i \rightarrow v_i)| = d(u_i, v_i) + 4$, where $d(u_i, v_i) = H(u_i, v_i) + 2^k$, $H(u_i, v_i)$ is Hamming distance between u_i and v_i. From the path construction in Case 2, the longest length of the paths $(u_i \rightarrow v_i)$ is $|(u_i \rightarrow v_i)| = d(u_i, v_i) + 4 + \max_{0 \leq j \leq k-1}\{|Z_j|\} \leq d(u_i, v_i) + m + 5$. We summarize the results in the following theorem.

Theorem 1. *Given any two clusters C_u and C_v in MC(k,m), we can find a multi-channel cube $C_u \cup C_v \cup E$, where E is a set of $k2^m$ disjoint paths connecting C_u and C_v, k paths for each node pair $(u_i, v_i),\ u_i \in C_u,\ v_i \in C_v$, and the length of the paths in E is at most* $\max(d(u_i, v_i)) + m + 5$, *for* $i = 0, 1, \ldots, 2^m - 1$.

5. Fault-tolerant routing

With the multi-channel cube structure, the fault-tolerant routing problem can be solved efficiently in an MC(k,m) assuming that there are up to $k+m-1$ faulty nodes. The fault-tolerant routing from source s to destination t in hypercube has been solved in [4]. The hypercube algorithm is summarized below. First, we partition the n-cube into two $(n-1)$-dimensional subcubes along a dimension k such that s and t are in the distinct subcubes. If the subcube containing t has less faulty nodes then we route s to the subcube containing t by a fault-free path of length at most 2: $(s \rightarrow s^{(k)})$ or $(s \rightarrow s^{(j,k)})$ for some $j,\ 0 \leq j \leq n-1,\ j \neq k$. Then the problem is reduced to the sub-problem in the subcube containing t. Repeat this process at most $\log n$ times until the subcube contains no faulty node. The path $(s \rightarrow s^{(j,k)})$ is a bad candidate if the jth bit of s is the same as that of t (a bad candidate will increase the length of the routing path). The algorithm guarantees that the bad candidate is used by at most one partition. It was shown in [4] that given two nonfaulty nodes s and t, and up to $n-1$ faulty nodes in an n-cube, we can find a fault-free path of length at most $d(s,t)+2$ in $O(n)$ time.

Under the conditions $k \leq m$ and the number of faulty nodes in $C_s \cup C_t$ is at most m, we show that an efficient algorithm for fault-tolerant routing from s to t in $MC(k,m)$ can be found. For most of the interesting MC(k,m), we have $k \leq m$ as indicated in the previous sections. Statistically, the condition that there are at most m faculty nodes in the two clusters is held with very high probability. Assuming uniform distribution, the probability of at most m fault nodes in $C_s \cup C_t$ is $\sum_{i=0}^{m} \binom{m+k-1}{i}\binom{2^{m2^k+k}-(m+k-1)}{2^{m+1}-i} / \binom{2^{m2^k+k}}{2^{m+1}} \approx 1$. For example, the probability is $3186303/3186304 = 0.99999969$ in the case of $k = m = 2$. An algorithm for fault-tolerant routing under the conditions above in an MC(k,m) is given below.

Algorithm 3 (Fault_Tolerant_Routing(k,m,s,t))
begin

Case 1: $C_s = C_t$. If C_s contains at most $m-1$ faulty nodes then we apply the hypercube algorithm. Otherwise, at least m faulty nodes are in C_s. We find k disjoint paths $(s \rightarrow t)$ outside the cluster C_s as follows: $P_j = (s : s^{(j)} : s^{(j,k)} : s^{(j,k,j)} \rightarrow t^{(j,k,j)} : t^{(j,k)} : t^{(j)} : t),\ 0 \leq j \leq k-1$, where $s^{(j,k,j)}$ and $t^{(j,k,j)}$ are in the same cluster $\neq C_s$. Since there at most $k-1$ faulty nodes outside the cluster C_s, there exists a fault-free path P_j.

Case 2: $C_s \neq C_t$. We create a multi-channel cube with C_s and C_t as described in the previous subsection. Since there are at most m fault nodes in $C_s \cup C_t$, we can find a fault-free path of length at most $d(s,t)+2$ in $O(m)$ time in the $(m+1)$-cube formed by $C_s \cup C_t$. If the k paths $(u \rightarrow v)$ of the multi-channel cube, which correspond to a single $(m+1)$th dimensional cube-edge, are all faulty, then we mark v as a faulty node. If the number of faulty nodes in $C_s \cup C_t$ is m, then no marked faulty node will be created. Otherwise, since a marked faulty node consumes k faulty nodes outside clusters C_s and C_t, the total number of faulty and marked faulty nodes will be at most $m+1-1=m$. Therefore, we can apply the hypercube algorithm, and find a fault-free path in the $(m+1)$-cube formed by $C_s \cup C_t$. Replacing the $(m+1)$th dimensional cube-edge (u,v) in the fault-free path $(s \rightarrow t)$ by a fault-free

path $(u \rightarrow v)$ in multi-channel cube, we find a fault-free path $(s \rightarrow t)$ in MC(k,m).

end.

Next, we analyze the length of the fault-free path and the running time of the algorithm in MC(k,m). For Case 1, If there are at most $m-1$ faulty nodes in C_s then a fault-free path of length at most $d(s,t)+4$ can be found in $O(m)$ time. Otherwise, a fault-free path of length at most $d(s,t)+6$ that passes through nodes outside the cluster can be found in $O(km2^k)$ time. For Case 2, the algorithm takes $O(m)$ time to find a fault-free path in the $(m+1)$-cube. To check whether the k paths in the multi-channel cube which correspond to a single $(m+1)$th cube-edge, are all faulty or not takes $O(km2^k)$ time. Since this process is repeated at most twice, the fault-free path in the multi-channel cube can be found in $O(km2^k) = O(kn)$ time, where 2^n is the number of nodes in MC(k,m). Since the path $(u \rightarrow v)$ in the multi-channel cube which corresponds to a single cube-edge is of length at most $d(u,v)+m+5$, and $|(s \rightarrow u)| \leq 1$, $|(v \rightarrow t)| \leq m$ (from the hypercube algorithm) the length of the fault-free path in MC(k,m) is at most $d(s,t)+2+2m+m+5 = d(s,t)+3m+7$.

Example 5: Consider $s = 0100000111$ and $t = 0000000011$ in $MC(2,2)$. Assume that nodes $s^{(i)}$, $i = 0,1$, and node 0100000011 are faulty. The 3-cube formed by $C_s \cup C_t$, where $C_s = \{3,7,11,15\}$, $C_t = \{0,1,2,3\}$, has the edges $(3,3),(7,2),(11,1)$ and $(15,0)$, for connecting C_s and C_t. Since $s^{(i)}$, $i = 0,1$, are faulty, we mark node 2 of class 0 faulty. Then there are two faulty nodes in the 3-cube. Therefore, we can find a fault-free path as follows: $(7 : 15 \rightarrow 0 : 1 : 3)$. Finally, replacing the edge $(15,0)$ by a path $(0100001111 \rightarrow 0000000000)$ given in Example 3, we get a fault-free path $(s \rightarrow t)$ of length $13+3 = 16$. Since $d(s,t) = 1+4 = 5$, we have $|(s \rightarrow t)| < d(s,t)+3m+7 = 18$. We summarize the result of the above argument into the following theorem.

Theorem 2. *Given any two nonfaulty nodes s and t, and at most $m+k-1$ faulty nodes in MC(k,m), if $k \leq m$ and the number of faulty nodes in $C_s \cup C_t$ is at most m, then a fault-free path $(s \rightarrow t)$ of length at most $d(s,t)+3m+7$ can be found in $O(kn)$ time, where the number of nodes in MC(k,m) is 2^n.*

6. Conclusion and future work

In this paper, we introduced an algorithm for fault-tolerant routing 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 issue of fault-tolerance in metacube is very important. We conclude that the fault-tolerant routing in metacube can be done quite efficiently. Some other issues concerning the metacube listed below are worth further research.

1. Evaluate the architecture complexity vs. performance of benchmarks vs. real cost.
2. Investigate the embedding of other frequently used topologies into the metacube.
3. Develop the techniques for mapping application algorithms onto the metacube.

References

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

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

[6] 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.

[7] 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.

[8] 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.

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

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

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

[12] 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.

[13] 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.