Efficient Communication in Metacube: A New Interconnection Network

Yamin Li and 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

This paper introduces a new interconnection network for very large parallel computers called metacube (MC). An MC network has a 2-level cube structure. An MC(k,m) network connects 2^{m2^k+k} nodes with $m+k$ links per node, where k is the dimension of a high-level cube and m is the dimension of low-level cubes (clusters). An MC network is a symmetric network with short diameter, easy and efficient routing similar to that of hypercubes. However, an MC network can connect more than one hundred of millions of nodes with only 6 links per node. Design of efficient routing algorithms for collective communications is the key issue for any interconnection network. In this paper, we also show that total exchange (all-to-all personalized communication) can be done efficiently in metacube.

1. Introduction

An n-dimensional hypercube, or 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, in the hypercube, the number of edges per node increases logarithmically as the total number of nodes increases. 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. We have found an interconnection network which will link millions of nodes with only 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 [3]; some focused on reduction of the number of edges of the hypercube, for example cube-connected cycles [13] and reduced hypercube [15]; and some focused on both, as in the hierarchical cubic network [4]. One major property of the hypercube is: there exists 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 hypercube's *key property*. Generally, variations of the hypercube that reduce the diameter, e.g. the crossed cube and hierarchical cubic network, will not satisfy this key property. An alternative to the Star graph, called Macro-Star [14], also reduces the number of edges, but the routing algorithm on a Macro-Star network is much more complex than that on a hypercube.

Our goal is to accommodate as many nodes as possible with a fixed number of links per node, and at the same time, keep the main structures and desirable properties of the hypercube such as the key property, small diameter, efficient routing, broadcasting, and total exchange etc. In this paper, we propose a new interconnection network, called *metacube* (MC) and describe an optimal routing algorithm for total exchange in an MC. The MC shares many desirable properties of the hypercube 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($3,3$) with 6 links per node has $2^{27} = 134,217,728$ nodes.

The remainder of this paper is organized as follows. Section 2 introduces the MC network. Section 3 gives a total exchange algorithm on an MC and analyzes its time complexity. Finally, Section 4 concludes the paper and presents some further research issues on metacubes.

2. Metacube interconnection network

There are two parameters in an MC: k and m. An MC(k,m) contains 2^k *classes*. Each class contains $2^{m(2^k-1)}$

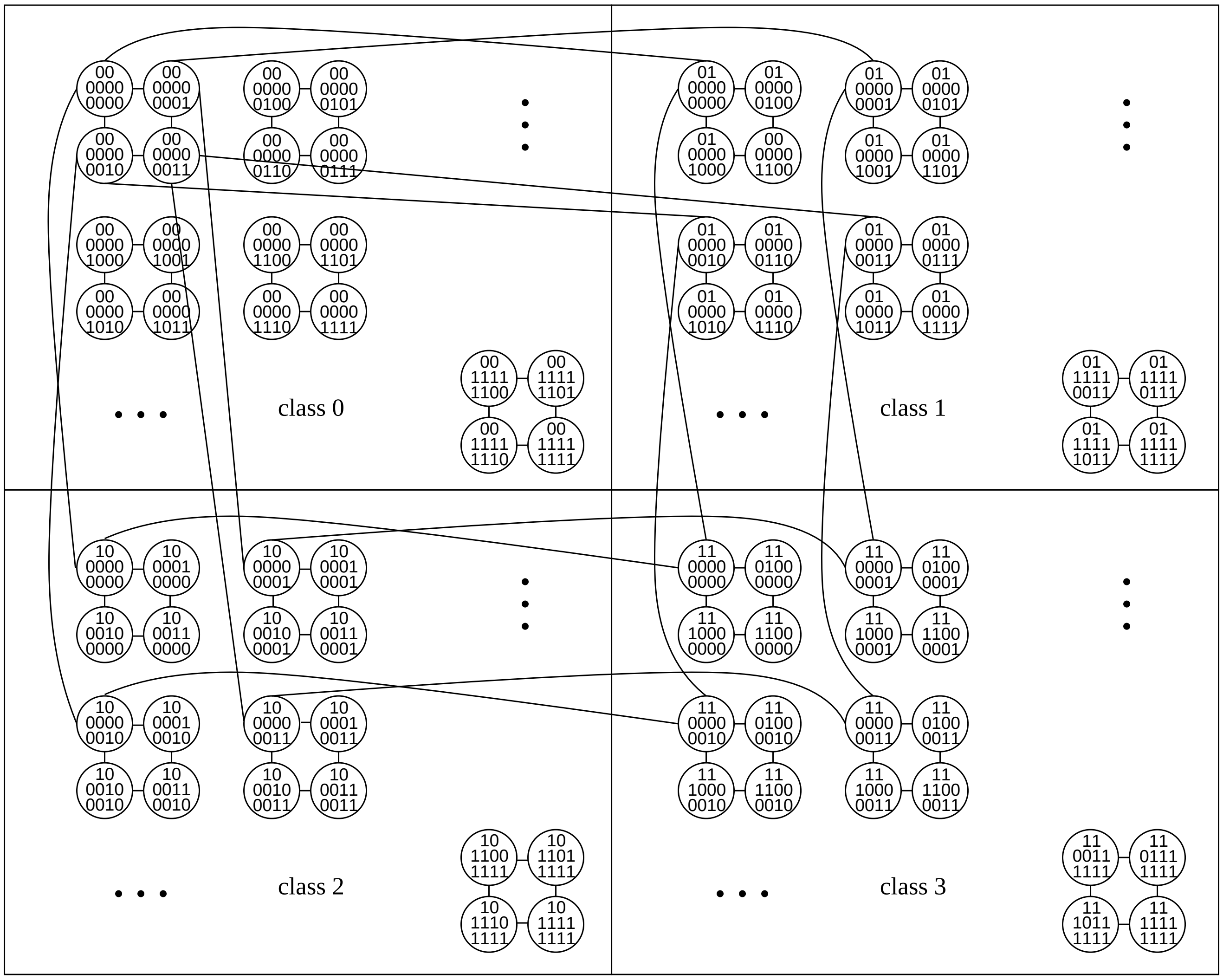

Figure 1. A metacube MC(2,2)

clusters and each cluster contains 2^m *nodes*. Therefore, an MC(k,m) contains $p = 2^{m2^k+k}$ nodes and the number of binary bits of the node address is $n = m2^k + k$. 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* [9] [10]. Similarly, an MC$(2,m)$, an MC$(3,m)$, and an MC$(4,m)$ containing 2^{4m+2}, 2^{8m+3}, 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. Notice that an MC$(0,m)$ is an m-cube.

In the following discussion, we use $(c, m[2^k - 1], \ldots, m[1], m[0])$ to denote a node address. It has (2^k+1) fields. c, located in the 2^kth (leftmost) field position, is the k-bit *class_id*, which defines the *class* of a node; $m[c]$, located in the cth field position, is the m-bit *node_id* and $(m[2^k-1], \ldots, m[c+1], m[c-1], \ldots, m[0])$ is the $m(2^k-1)$-bit *cluster_id*. For the different value of c, the field position of the *node_id* in the address is different. The address of a specific node, node s for instance, is denoted with $s = (c_s, m_s[2^k-1], \ldots, m_s[1], m_s[0])$.

An MC(k,m) is constructed in the following manner. Within a cluster, m links per node form an m-cube. We call these m links *cube-edges*. The links that connect nodes in different clusters are defined as following. Two nodes whose addresses differ only in one bit within the k-bit class field are connected by a link. There are k such links per node, we call these k links *cross-edges*. Therefore, a node in an MC(k,m) has $m+k$ links: m links construct a *low-level* m-cube and k links construct a *high-level* k-cube.

The addresses of two nodes connected by a cross-edge differ only in one bit within the 2^kth field (k-bit *class_id* c) and there is no direct connection among the clusters of the same class. The addresses of two nodes connected by a cube-edge differ only in one bit within the cth field (m-bit *node_id* $m[c]$). For example, the neighbors in the cluster of the node (01,111,101,110,000) (*class_id* $c = 01$ and *node_id* $m[c] = 110$) in an MC(2,3) are (01,111,101,11$\underline{1}$,000), (01,111,101,1$\underline{0}$0,000) and (01,111,101,$\underline{0}$10,000). The two neighbors in the k-cube are (0$\underline{0}$,111,101,110,000) and ($\underline{1}$1,111,101,110,000).

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

The ratio of the total number of links in a hypercube to the total number of links in an MC 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 16,384 nodes; a hypercube contains $2^{14} \times 14/2 = 114{,}688$ links, whereas an MC contains $2^{14} \times (3+2)/2 = 40{,}960$ links. The reduction in the total number of links for this example is 73,728 links, or about 64%.

3. Total exchange algorithm

3.1. Communication model and total exchange

Design of efficient routing algorithms for collective communications is the key issue in message-passing parallel computers or networks [2] [5] [8] [11]. Collective communications are required in load balancing, event synchronization and data exchange. Based on the number of sending and receiving processors, these communications can be classified into one-to-many, one-to-all, many-to-many and all-to-all. The nature of the messages to be sent can be classified into personalized or non-personalized (multicast or broadcast). The all-to-all personalized communication, called total exchange sometimes, is at the heart of numerical applications.

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

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

We use *cut-through* switching technique [6] [7] to perform the total exchange. In cut-through switching, each message is serialized into a sequence of pieces and is sent in a pipeline fashion. The router can start forwarding the header and the following data bytes as soon as routing decisions have been made and the output buffer is free. For long messages, the pipelining effect of cut-through switching reduces the effect of path length on network. The predominant *wormhole* switching [12] is just a special form of cut-through. With cut-through switching, the transmission time for a message of length g to be sent to a node of distance d is $t_s + gt_w + dt_h$, where t_s is startup latency, the time required for the system to handle the message at the source and destination nodes, t_w is the per-word transfer time ($1/t_w$ is the bandwidth of the communication links), and t_h is the per-hop time, the time to switch an intermediate node.

In total exchange, each node sends a distinct message to every other node. The total number of messages is $p(p-1)$ where p is the number of nodes. Because the metacube has much smaller number of links compared to the traditional hypercube, we must find a proper way to perform total exchange in metacube efficiently. The two key points are

1. to determine the order of destination nodes for a source node so that every pair of source and destination nodes has the same path length in every sending step, thus every source node spends the same time to send a message to a destination node; and
2. to perform the conflict-free routing.

In the following, we focus on an MC(k,m) with $k = 2$ (*quad-cube*). The idea should also work for an MC(k,m) with $k \geq 3$. We describe an innovative policy for arranging the order of the $p-1$ destination nodes from a given source node, and an optimal routing strategy for total exchange in an MC($2,m$) in the following subsections.

3.2. An efficient algorithm for total exchange

We adopt the following strategy: each node sends one message to its destination at one *communication step*. There are totally $(p-1)$ communication steps. Our goals are that

1. each node sends a distinct message to every other node through a shortest path,
2. all the p nodes send messages simultaneously and
3. all the paths do not conflict with each other at any given time.

We give an efficient algorithm (Algorithm 1) which determines the order of the $(p-1)$ destination nodes for the source node s, and then calls the routing algorithm to be specified in the next subsection. All the nodes execute the same algorithm simultaneously. The variable s is the address of the node, and $M_{s,*}$ is the set of messages to be sent out from s. We use C/Java style to present our algorithms.

Algorithm 1 (TotalExchange (m, s, $M_{s,*}$))

```
 1. begin    /* $M_{s,d}$ is the message from node s to node d */
 2.   for $c[4] = 0$ to 3 do                    /* each class_id */
 3.     for $c[3] = 0$ to $2^m - 1$ do
 4.       for $c[2] = 0$ to $2^m - 1$ do
 5.         for $c[1] = 0$ to $2^m - 1$ do
 6.           for $c[0] = 0$ to $2^m - 1$ do /* each node_id */
 7.             $x = c[4] \oplus c_s$;  /* $c_s$ is the class_id of s */
 8.             $d = (c[4],\ c[3 \oplus x],\ c[2 \oplus x],$
                     $c[1 \oplus x],\ c[0 \oplus x]) \oplus s$;
 9.             if ($d \neq s$) Routing($m$, $s$, $d$, $M_{s,d}$);
10. end
```

According to the format of a node address for an MC(2, m), we use five "for" loops to generate p destination addresses. Variable $c[4]$ is a *class_id*, $c[0]$ is a *node_id* and $c[1]$, $c[2]$, $c[3]$ together form a *cluster_id*. The operator $\oplus$ is a bitwise exclusive-OR logical operation, and c_s is the *class_id* of s. Consider $s = 0$ ($c_s = 0$), the destination node $d = (0,\ c[3],\ c[2],\ c[1],\ c[0])$ if $c[4] = 0$. That is, $c[i]$ $(0 \leq i \leq 3)$ appears in the field position i of d. For the different value of $c[4]$ (*class_id*), $c[0]$ (*node_id*) should appear in different field position. The rule for the field position of $c[i]$ $(0 \leq i \leq 3)$ is that $c[i]$ should be in the field position $(i \oplus c[4])$. From the algorithm, for $s = 0$, we have

$$\begin{aligned}
c[4] = 0 &\Rightarrow d = (0,\ c[3],\ c[2],\ c[1],\ c[0]);\\
c[4] = 1 &\Rightarrow d = (1,\ c[2],\ c[3],\ c[0],\ c[1]);\\
c[4] = 2 &\Rightarrow d = (2,\ c[1],\ c[0],\ c[3],\ c[2]);\\
c[4] = 3 &\Rightarrow d = (3,\ c[0],\ c[1],\ c[2],\ c[3]).
\end{aligned}$$

For the cases of $s \neq 0$, we get the destination node $d = (c[4],\ c[3 \oplus x],\ c[2 \oplus x],\ c[1 \oplus x],\ c[0 \oplus x]) \oplus s$, where $x = c[4] \oplus c_s$ is the *class_id* of d. The operation $\oplus x$ makes $c[i]$ $(0 \leq i \leq 3)$ located in the proper field position and the operation $\oplus s$ makes every pair of (s, d) symmetric with every s. This algorithm ensures that

1. it generates every other node address exactly once and
2. every pair of (s, d) generated by the algorithm at the same step has the same distance between s and d in an MC$(2, m)$.

Then the message $M_{s,d}$ is routed from s to d if $d \neq s$. The routing algorithm is given in the next subsection.

3.3. A conflict-free routing algorithm

In the total exchange, all nodes send messages simultaneously, each path must not conflict with every other path at any given time. In this subsection, we present an optimal routing algorithm that generates conflict-free shortest path from s to d.

In a hypercube, two nodes s and d whose addresses differ in l bits are connected by a shortest path of length l. l is called *Hamming distance* between s and d that equals the number of non-zero bits in the binary representation of $s \oplus d$. A message traveling from s to d must pass through at least l links. There should be more than one solution for a message traveling from s to d. Through this paper, we follow the *ascending routing* strategy, by which the least significant non-zero bit of $s \oplus d$ is chosen as the first dimension for routing and so on.

In a metacube, only m cube-edges can be used for routing within a cluster. That is, the routing can take place in cth field ($m[c]$) for a cluster of class c. Routing within a cluster can be done as a similar manner as a hypercube does. We list it below (Algorithm 2), where $M_{s,d}$ is the message from s to d, $diff = d \oplus s$, *current* is a node the message arrived and c is the *class_id* of the cluster. Notice that *current* is a global variable.

Algorithm 2 (routingInCluster (m, c, $diff$, $M_{s,d}$))

```
1. begin             /* routing within a cluster of class c */
2.   for $i = 0$ to $m - 1$ do
3.     $go = m_{diff}[c]$ & $2^i$;
4.     if ($go \neq 0$)
5.       $m_{current}[c] = m_{current}[c] \oplus go$;
6.       send $M_{s,d}$ to current;
7. end
```

This algorithm is just for routing within a cluster of class c. Routing from s to d needs to go along cross-edge(s) if s and d are in different clusters. We therefore must determine a class sequence along which a path from s and d can be built. We define *type* of a destination node d to indicate the difference between the addresses of s and d. The definition of type simplifies the construction of a shortest path for class sequence from c_s to c_d, called classPath. The classPath is the key for the conflict-free routing in total exchange. If every $m_s[i] = m_d[i]$, for $0 \leq i \leq 3$, $i \neq c_s$ and $i \neq c_d$, let $type = 0$. In such a case, a path can be obtained by routing within the cluster of s, going along cross-edge(s) in high-level k-cube to the cluster of d, and routing within the cluster of d. That is, no cluster other than the clusters of s and d needs to be routed inside. Otherwise, a cluster of class i must be routed inside. Based on the appearance of such i, *type* may be 1, 2, or 3.

Except for the field positions 0 and *dest*, if there is only one field i in which $m_s[i] \neq m_d[i]$, then *type* is 1 or 2 depend on the field position of i; otherwise, $type = 3$ (the case of $dest = 0$ is special in which $type = 3$ as long as $m_s[3] \neq m_d[3]$). For any s and d, the procedure for getting *type* is shown in Algorithm 3. The input parameters are s and d and the *type* will be returned. We use $dest = c_d \oplus c_s$. The $\oplus c_s$ is also applied to the field position for each field.

Algorithm 3 (getType (s, d))

```
 1. begin
 2.   $type = 0$;
 3.   switch ($dest = c_d \oplus c_s$)
 4.     case 0:                         /* s and d are of the same class */
 5.       if ($m_d[2 \oplus c_s] \neq m_s[2 \oplus c_s]$)   $type = 2$;
 6.       if ($m_d[1 \oplus c_s] \neq m_s[1 \oplus c_s]$)   $type = type + 1$;
 7.       if ($m_d[3 \oplus c_s] \neq m_s[3 \oplus c_s]$)   $type = 3$; break;
 8.     case 1:                         /* $c_s$ and $c_d$ differ in dimension 0 */
 9.       if ($m_d[3 \oplus c_s] \neq m_s[3 \oplus c_s]$)   $type = 2$;
10.       if ($m_d[2 \oplus c_s] \neq m_s[2 \oplus c_s]$)   $type = type + 1$; break;
11.     case 2:                         /* $c_s$ and $c_d$ differ in dimension 1 */
12.       if ($m_d[3 \oplus c_s] \neq m_s[3 \oplus c_s]$)   $type = 2$;
13.       if ($m_d[1 \oplus c_s] \neq m_s[1 \oplus c_s]$)   $type = type + 1$; break;
14.     case 3:                         /* $c_s$ and $c_d$ differ in dimensions 0 & 1 */
15.       if ($m_d[2 \oplus c_s] \neq m_s[2 \oplus c_s]$)   $type = 2$;
16.       if ($m_d[1 \oplus c_s] \neq m_s[1 \oplus c_s]$)   $type = type + 1$; break;
17.   return ($type$);
18. end
```

In order to avoid conflicts, we construct the routing paths based on a set of classPaths which is generated with the *types* and the *dests* of destination node d. The classPath bypasses some classes based on the information carried in the value of *type*. The routing path from s to d in our routing algorithm which is constructed from classPath, is a shortest path. Here, we show the table of classPaths including the lengths of the classPaths (Table 1) for the nodes of class 0 to route from c_s $(= 0)$ to c_d $(= dest)$ in the high-level k-cube along cross-edges. For each pair of $(dest, type)$, the sequence of numbers in the table indicates the classPath and its length: the first number is the length of the classPath, and the rest is the classPath. For example, in the case of $dest = 1$ and $type = 1$, the table shows (3, 2, 3, 1): the path length is 3, the class path is $(0, 2, 3, 1)$ since a cluster of class 2 must be routed inside, where 0 is the *class_id* of s and 1 is the *class_id* of d. Notice that the class of the source node was not stored in the table. In our routing algorithm, this number sequence is accessed with an index *next* $(= 0, 1, \ldots)$: $next = 0$ indicates the first number in the sequence (length), $next = 1$ indicates the second number (next class to s) and so on. Therefore, classPath[*dest*,*type*,0] is the classPath length, and classPath[*dest*,*type*,*next*] for $next = 1, 2, \ldots$ is a class in the class sequence of classPath.

The classPath table listed in Table 1 is for the source nodes of class 0. A source node located in a different class may need a different routing table. Here we adopt a *node renaming* technique so that one such table can be used for any node. The node renaming technique is relied on the fact of $y \oplus x \oplus x = y$ for any x. For a source node s and a destination node d, if c_s and c_d are the same, then the first line ($dest = 0$) in Table 1 will be accessed with the index $dest = c_d \oplus c_s = 0$. The real class for the next cross-edge traveling can be gotten by $c_i \oplus c_s$, where c_i is the class obtained from the table. The conflict-free shortest path routing algorithm for the total exchange is given in Algorithm 4 which is invoked by Algorithm 1.

Table 1. List of classPaths with path lengths

type	0	1	2	3
$dest = 0$	(0)	(2, 1, 0)	(2, 2, 0)	(4, 1, 3, 2, 0)
$dest = 1$	(1, 1)	(3, 2, 3, 1)	(3, 2, 3, 1)	(3, 2, 3, 1)
$dest = 2$	(1, 2)	(3, 1, 3, 2)	(3, 1, 3, 2)	(3, 1, 3, 2)
$dest = 3$	(2, 1, 3)	(2, 1, 3)	(2, 1, 3)	(4, 1, 3, 2, 3)

Algorithm 4 (Routing (m, s, d, $M_{s,d}$))

```
 1. begin    /* $M_{s,d}$ is the message from node s to node d */
 2.   $diff = d \oplus s$;       /* different address bits of d and s */
 3.   $current = s$;                        /* current node = s */
 4.   $dest = c_d \oplus c_s$;          /* $c_d$ ($c_s$) is class_id of d (s) */
 5.   $type$ = getType ($s$, $d$);             /* call Algorithm 3 */
 6.   $next = 0$;
 7.   RoutingInCluster ($m$, $c_{current}$, $diff$, $M_{s,d}$);
 8.   while ($next <$ classPath$[dest, type, 0]$)
 9.     $next = next + 1$;
10.     $c_{current}$ = classPath$[dest, type, next] \oplus c_s$;
11.     send $M_{s,d}$ to $current$;
12.     if ($current \neq d$)
          RoutingInCluster ($m$, $c_{current}$, $diff$, $M_{s,d}$);
13. end
```

The algorithm first routes in the source cluster (line 7, calling Algorithm 2), then goes to a cluster along a cross-edge based on the classPath table (lines 10-11) and routes in the cluster (line 12) if necessary. The operations described in lines 9–12 repeat until reaching the destination node.

All the algorithms (procedures) needed for total exchange were described. We analyze the communication time in the next subsection.

3.4. Time analysis of total exchange algorithm

As mentioned in subsection 3.1, we use $T = t_s + gt_w + dt_h$ to evaluate the total exchange time for metacube. For the n-cube's case, let the number of nodes $p = 2^n$. Because there are $\binom{n}{i}$ nodes at distance i from the source node, the time it takes to send $(p-1)$ messages to all the other nodes is

$$T_C = \sum_{i=1}^{n} (t_s + gt_w + it_h) \binom{n}{i} = (p-1)(t_s + gt_w) + p(\frac{1}{2}\log p)t_h.$$

In a metacube, the path length from s to d is longer than that in a hypercube. That is, the time an MC$(2,m)$ takes for the total exchange, T_M, will has a larger coefficient of t_h. In order to get T_M, we calculate the extra distance a node takes to send messages to all the other nodes. For example, when $type = 1$, the metacube takes 2, 3, 3 and 2 steps in routing on the high-level 2-cube for $dest = 0,1,2,$ and 3, respectively, while the hypercube takes 0, 1, 1 and 2 steps. We subtract these hypercube step values from the corresponding path length classPath[*dest*,*type*,0], then we get the extra distances. Let E_i ($0 \le i \le 3$) be the extra distance for the case of $dest = i$, then

$$\begin{aligned}
E_0 &= [2^m(2^m-1)+2^m(2^m-1)]\times 2+\\
&\quad [2^m(2^m-1)^2+2^m(2^{3m}-2^{2m})]\times 4,\\
E_1 &= [2^{2m}(2^m-1)+2^{2m}(2^m-1)+2^{2m}(2^m-1)^2]\times 2,\\
E_2 &= [2^{2m}(2^m-1)+2^{2m}(2^m-1)+2^{2m}(2^m-1)^2]\times 2,\\
E_3 &= 2^{2m}(2^m-1)^2\times 2.
\end{aligned}$$

The total extra distance is $E = \sum_{i=0}^{3} E_i = 5\times 2^{4m+1} - 2\times 2^{3m+1} - 3\times 2^{2m+1}$. Therefore, the time the metacube takes for total exchange is

$$T_M = (p-1)(t_s+gt_w)+[p(\frac{1}{2}\log p+\frac{5}{2})-\sqrt{2}p^{3/4}-3\sqrt{p}\,]t_h.$$

For an MC$(2,2)$, $T_M = 1023(t_s+gt_w)+7328\ t_h$. The hypercube with the same size is a 10-cube, and $T_C = 1023(t_s+gt_w)+5120\ t_h$. It shows that our result is quite satisfactory since an MC$(2,2)$ uses only 40% of the links in comparison to the hypercube. The ratio of the time difference between metacube and hypercube to the total time will become smaller for larger m.

4. Conclusion and future work

In this paper, we proposed a new interconnection network, the metacube, and showed that the total exchange can be done efficiently on it. The proposed metacube has tremendous potential to be used as an interconnection network for very large scale parallel computers since it can connect more than one hundred of millions of nodes with only 6 links per node and retains hypercube's properties that are useful for efficient communications among the nodes.

Recently, much of the community has moved on to lower-dimensional topologies such as meshes and tori. However, the SGI Origin2000, a fairly recent multiprocessor, does use a hypercube topology, so the metacube could be of use to industry. A lot of issues concerning the metacube require further research. Some of them are:

1. Evaluate the architecture complexity vs. performance of benchmarks vs. real cost.
2. Investigate the embedding of other frequently used topologies into a metacube.
3. Develop techniques for mapping application algorithms onto a metacube.
4. Develop fault-tolerant routing algorithms for a metacube with faulty nodes.

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

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

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

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

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

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

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

[9] 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.

[10] Y. Li, S. Peng, and W. Chu. Efficient collective communications in dual-cube. In *Proceedings of the Thirteen IASTED International Conference on Parallel and Distributed Computing and Systems*, pages 266–271, Aug. 2001.

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

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

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

[14] C.-H. Yeh and E. A. Varvarigos. Macro-star networks: Efficient low-degree alternatives to star graphs. *IEEE Transactions on Parallel and Distributed Systems*, 9(10):987–1003, Oct. 1998.

[15] 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.