EMBEDDING A COMPLETE BINARY TREE INTO A FAULTY SUPERCUBE

Haun-Chao Keh and Jen-Chih Lin
Department of Computer Science and Information Engineering
Tamkang University, Tamsui, Taipei, Taiwan 251, R.O.C.
*Contact Author e-mail:g3320195@power.cs.tku.edu.tw

Abstract

The supercube is a novel interconnection network that is derived from the hypercube. Unlike the hypercube, the supercube can be constructed for any number of nodes. That is, the supercube is incrementally expandable. In addition, the supercube retains the connectivity and diameter properties of the corresponding hypercube. In this paper, we consider the problem of embedding and reconfiguring binary tree structures in a faulty supercube. Further more, for finding the replaceable node of the faulty node, we allow 2-expansion such that we can show that up to $(n-2)$ faults can be tolerated with congestion 1 and dilation 4 that is $(n-1)$ is the dimension of a supercube.

Key words: supercube, hypercube, embed, complete binary tree, fault tolerance

1 Introduction

Embedding a given guest graph G into a host graph H means mapping the nodes of G to the node of H such that the edges of G also map to paths of H. Four costs associated with graph embedding are *dilation*, *expansion*, *load* and *congestion*. The dilation of an embedding is the maximum distance in the host between the images of the adjacent guest graph. The expansion is the ratio of total number of nodes of G to total number of nodes of H. The load of an embedding is the maximum number nodes of the guest graph that are embedded in any single node of the host graph. The congestion of an embedding is the maximum number of edges of the guest graph G that is embedded using any single edge of the host graph H.

The hypercube has a significant drawback; it is not incrementally extensible. We consider the supercube a novel interconnection network derived from the hypercube that can be constructed for any number of nodes. The supercube has the same connectivity and diameter of the corresponding hypercube. The supercube is shown to have the following desirable characte-ristic[11]: (1) adding a new node to an existing network is easy; it doesn't need reorganization of existing edges. (2) the nodes connectivity of a N-node supercube is at least $\lfloor \log_2 N \rfloor$(3) the node degree of a N-node supercube is between $(k-1)$ and

(0-7803-4229-1/97/$10.00 1997 IEEE)

$(2k-2)$, where $k = \lceil \log_2 N \rceil$, (4) and the diameter is of a N-node supercube at most $\lfloor \log_2 N \rfloor$.

In a multiprocessor system, there are two fault models are considered[7]. One is total fault model that is the computation and the communication fails. The other is partial fault model that is only the communication fails. In this paper, our fault model is partial model. That is, when the computation nodes are fault, the communication links are well. Only the faulty node is remapped. We try to embed the complete binary tree into the faulty supercube with unit load. Further more, for finding the replaceable node of the faulty node, we allow expansion is equal to 2 such that we can show that up to $(n-2)$ faults can be tolerated with congestion 1 and dilation 4 that is $(n-1)$ is the dimension of the supercube.

The remainder of this paper is organized as follows. In the next section, we introduce the necessary notation and definitions. At the same time, we describes how to embedding a complete binary tree into a hypercube. In section 3, we embed a complete binary tree into a faulty supercube. In the last section 4, we give the conclusions of the paper and discuss further research problems.

2 Preliminary

In this section, we briefly describe notations and definitions of the supercube. The following formal properties of the supercube graph are from Sen[9]. The supercube is constructed as follows. The topology of a supercube is represented by an undirected graph $S_N = (V, E)$, where V is the set of processors (in the following called nodes) and E is the set of bidirectional communication links between the processors (called edges). Assume that V contains N nodes, which is numbered from 0 to $(N-1)$. Then, let n is defined $2^{n-1} \leq N \leq 2^n$. Each node can be expressed by an n-bit binary string $i_{n-1}...i_0$ where $i_p \in \{0, 1\}$.

Define2.1[9] Suppose $S_N = (V, E)$ is an $(n-1)$-dimensional supercube, then the node set V can be divided into three subsets V_1, V_2, V_3, where

$V_3 = \{x |\ x \in V,\ x = 1u,$ where u is a $(n-1) - bit$ sequence$\}$,

$V_2 = \{x |\ x \in V,\ x = 0u,\ 1u \notin V,$ where u is a $(n-1) - bit$ sequence$\}$,

$V_1 = \{x |\ x \in V,\ x = 0u,\ 1u \notin V,$ where u is a $(n-1) - bit$ sequence$\}$.

Before we defined the edge set E, let us define a term called Hamming distance.

Define 2.2 The Hamming distance, $H.D.(a, b)$, is given by the number of bit positions where the nodes of a and b differ. In other words, $H.D.(a, b)$, is the number of bits set in the resulting sequence of the bitwise XOR of a and b.

Define 2.3[9] Suppose $S_N = (V, E)$ is an $(n-1)$-dimensional supercube, then the edge set E is the union of E_1, E_2, E_3 and E_4, where

$E_1 = \{(x,y) |\ x, y \in V,\ x = 0u,\ y = 0v,$
where u, v are $(k-1)$-bit sequences and $H.D.(x,y) = 1\}$,

$E_2 = \{(x,y) |\ x, y$ in $V3,\ x = 1u,\ y = 1v,$
where u, v are $(k-1)$-bit sequences and $H.D.(x,y) = 1\}$,

$E_3 = \{(x,y) |\ x$ in $V3,\ y$ in $V2, x = 1u,\ y = 0v,$
where u, v are $(k-1)$-bit sequences and $H.D.(x,y) = 1\}$,

$E_4 = \{(x,y) |\ x$ in $V3,\ y$ in $V1, x = 1u,\ y = 0u,$
where u are $(k-1)$-bit sequences$\}$.

The dimension of the supercube is $(n-1)$ and the Dimension of two nodes x and y is denoted by $Dim(x,y)$. $Dim(x,y)$ is equal to $i (i \leq (j-1))$ if and only if $H.D.(x,y) = 1$ and $x_i \neq y_i$.

In Auletta[1], although a complete binary tree of height h can be embedded into a 2^{h+1}-node supercube with dilation 1 and load 1, the supercube can't find the replaceable node of the faulty node. We can obtain recursive construction of a mapping of a double-rooted complete binary tree into a hypercube with dilation 2 and load 1[3].

Lemma3.1 A double-rooted complete binary tree can be contained in a hypercube with dilation 2 and load 1.

3 Embedding a complete binary tree into a faulty supercube with 2- expansion

In this section, we infer the method of the embedding from the section 2. The method of the embedding is proved by theorem 3.1.

Theorem 3.1 A complete binary tree of height h can be embedded into a $(2^{h+2}-2)$-node supercube with dilation 2, expansion 2 and load 1.

Proof. For finding the replaceable node of the faulty node, we suppose expansion is equal to 2. The total number of nodes of a complete binary tree T_h is $2^{h+1}-1$. The total number nodes of a supercube are $2*(2^{h+1}-1) = 2^{h+2}-2$. By section 2, the node set V of the supercube is partitioned into three subsets V_1, V_2 and V_3. The set $V\prime$ which is the union of V_1 and V_2 has 2^{h+1} nodes. and a complete binary tree can be embedded into a supercube with 2^{h+1} nodes using a double-rooted complete binary tree. Therefore, a complete binary tree of height h can be embedded into a $(2^{h+2}-2)$-node supercube with dilation 2, expansion 2 and load 1.

Define 3.2 A (2^h-2)-node supercube is lack of two nodes, we called them virtual nodes.

We describe our algorithms for embedding a complete binary tree into a faulty supercube as follows.

$searching - path(r)$
1 We can search the node m
/* $m \in V_3$, $H.D.(r,m) = 1$, $Dim(r,m) = (n-1)$*/.
2 *if* the node m is faultily *then*
2.1 $i = 0$
2.2 *while* $i \neq (n-2)$ *do*
2.2.1 we can search the node k
/* $k \in V_3$, $H.D.(m,k) = 1$, $Dim(m,k) = i$*/.
2.2.2 *if* node k is not virtual node and it is free *then*
2.2.2.1 node r is replaced by node k
2.2.2.2 exit the *while-loop*
2.2.5 $i = i + 1$
2.3 *if* $i = (n-2)$ *then*
2.3.1 declare the replaceable node of searching is faultily.
2.3.2 exit the $searching - path()$

$v - searching - path(r)$
1 $i = 0$
2 *while* $i \neq (n-2)$ *do*
2.1 we can search the node k
/*$k \in V_3$, $H.D.(r,k) = 2$, $Dim(r,k) = (n-1,i)$, $E(r,k) \in E_3$*/.
2.2 *if* node k is not virtual node and it is free *then*
2.2.1 node r is replaced by node k
2.2.2 exit the *while-loop*
2.3 $i = i + 1$
3 *if* $i = (n-2)$ *then*
3.1 declare the replaceable node of searching is faultily.
3.2 exit the $v - searching - path()$

Algorithm $searching - rule$:
1 *if* the root r is faultily *then*
1.1 search the spacer node S
1.2 *if* the spacer node S is faultily *then* return the root r
1.3 *else*
1.3.1 node r is replaced by node S.

1.3.2 exit the *algorithm searching − rule*
2 *if* $r \in V_1$ *then searching − path*(r)
3 *if* $r \in V_2$ *then* v *− searching − path*(r)
4 *if* the other node p is faultily *then*
4.1 *if* the node $p \in V_1$ *then searching − path*(p)
4.2 *if* the node $p \in V_2$ *then* v *− searching − path*(p)

Theorem 3.3 The finishing of a searching path is including $(n-2)$ nodes at least and the rule is right.

Proof. Each node can be expressed by an n-bit binary string $i_{n-1}...i_0$ where $i_p \in \{0,1\}$, so we can change a bit in sequence. We can get n different nodes. By define 3.2, we have two virtual nodes. Therefore, the finishing of searching path is including $(n-2)$ nodes at least. By Saad and Schultz[9], we infer the edges of the searching-rule exist. We can get the result of the rule is right

Theorem 3.4 If the root of the tree is faultily and the number of faulty nodes is $< (n-1)$, we can find the replaceable node of the root after $(n+1)$ times of search at most.

Proof. We assume we can't find the replaceable node of the faulty node. That is, all of nodes on the searching path are already used or fault. By theorem 3.3, we show each node can be expressed by an n-bit binary string $i_{n-1}...i_0$ where $i_p \in \{0,1\}$. Now there are two conditions can be considered. First, if the root $r \in V_1$, the searching path is including only one virtual node at most. If the searching path is including the virtual node with $i_0 = 0$, it must be not including $i_0 = 1$ by the searching-rule and vice versa. Second, if the root $r \in V_2$, the searching path is including two virtual nodes at most. We can search $(n-1)$ nodes after $(n+1)$ times of search at least. Because the number of faulty nodes is $< n$, we can find the replaceable node by pigeonhole principle. The originally assume is wrong. We can find the replaceable node of the root r after $(n+1)$ times of search at most.

Theorem 3.5 If a node of a subtree is faultily and the number of faulty nodes is $< (n-2)$, we can find the replaceable node of the faulty node after n times of search at most.

Proof. We assume we can't find the replaceable node of the faulty node. That is, all of nodes all on the searching path is already used or fault. By theorem 3.3, we show each node can be expressed by an n-bit binary string $i_{n-1}...i_0$ where $i_p \in \{0,1\}$. Now there are two conditions can be considered. First, if the faulty node $p \in V_1$, the searching path is including only one virtual node at most. If the searching path is including the virtual node with $i_0 = 0$, it must be not including $i_0 = 1$ by the searching-rule and vice versa. Second,

if the faulty node $p \in V_2$, the searching path is including two virtual nodes at most. We can search $(n-2)$ nodes after n times of search at least. Because the number of faulty nodes is $< (n-2)$, we can find the replaceable node by *pigeonhole principle*[3]. The originally assume is wrong. We can find the replaceable node of the node p after n times of search at most.

Theorem 3.6 There are $O(n)$ faults can be tolerated.

Proof. By theorem 3.4, there are $(n-1)$ faults can be tolerated. By theorem 3.5, there are $(n-2)$ faults can be tolerance. To sum up, we can show that $O(n)$ faults can be tolerated.

Theorem 3.7 The result holds dilation 4, congestion 1, expansion 2 and $O(1)$ load.

Proof. We show that we can embed a complete binary tree of height h into a $(2^{h+2}-2)$-node supercube using nodes of $V_1 \cup V_2$ with dilation 2 by section 2. There are two cases to be considered.

Case 1. First, if the faulty node p, $p \in V_1$, we can search the node m, $H.D.(p,m) = 1$, $m \in V_3$, $E(p,m) \in E4$ by the $searching - path()$. Second, if the node is used or fault, we can search the other nodes k, $k \in V3$, $H.D.(m,k) = 1$, $E(m,k) \in E_2$ by the $searching - path()$. At last, we can get the dilation 2 at most.

Case 2. If the faulty node p, $p \in V_2$, we can search the nodes m, $H.D.(p,m) = 2$, $m \in V_3$, $E(p,m) \in E_3$ by the $v - searching - path()$. We can get the dilation 1 at most.

Because every replaceable path is only one path by the *algorithmsearching–rule* and Saad[5], we can get congestion 1 and load 1. For finding the replaceable node of the faulty node, we allow 2-expansion. Therefore, when root node and spacer node are faultily, it is a worst case the dilation$= 2+2 = 4$ at most, the dilation is 2 in others condition. The other three costs associated with graph embedding are congestion 1, expansion 2 and $O(1)$ load.

We illustrate three case of finding a replaceable node as shown in Figure 1, 2 and 3.

4 Conclusion

In this paper, finding the replaceable node of the faulty node, we allow 2-expansion such that we can show that up to $(n-2)$ faults can be tolerated. Furthermore, we can prove them and present some algorithms to solve them. The result implies that any complete binary tree can be embedded into a supercube with congestion 1 and dilation 4 that is $(n-1)$ is the dimension of the supercube. By the result, we can embed the parallel algorithms developed

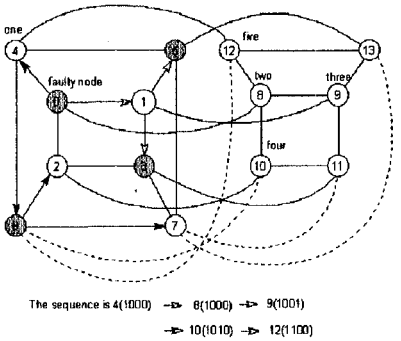

Figure 1:

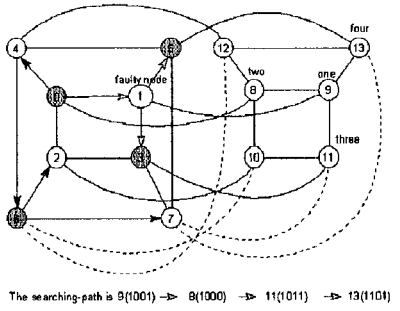

Figure 2:

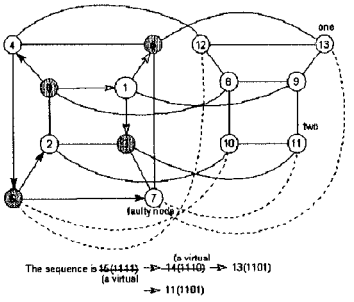

Figure 3:

by the structure of a complete binary tree into a supercube. This method of embedding enables extremely high-speed parallel computation.

After a complete binary tree can be embedded into a supercube with faulty nodes, it would be interesting to generalize our results to unbound expansion, the embedding of an arbitrary binary tree and multi-dimensional meshes into a supercube with faulty nodes.

References

1. V. Auletta, A.A. Rescigno, and V. Scarano, "Embedding Graphs onto the Supercube," *IEEE Trans. on Computers*, Vol. 44, No. 4, pp. 593-597, April 1995.
2. V. Auletta, A.A. Rescigno, and V. Scarano, "Fault Tolerant Routing in The Supercube," *Parallel Processing Letters*, Vol. 3, No. 4, pp. 393-405, 1993.
3. Dimitri P. Bertsekas and John N. Tsitsiklis, "Parallel and Distributed Computation: numerical methods," *Prentice Hall*, Englewood Ciffs, New Jersey, 1989.
4. Bethany M. Y. Chan, Francies Y. L. Chin, Senior member, IEEE, and C.-K. Poon, "Optimal Simulation of Full Binary Trees on Faulty Hypercubes, "*IEEE Trans. on parallel and distributed systems*, Vol. 6, No. 3, pp. 269-286, March 1995.
5. Y. Saad, and M. Schultz, "Topological properties of Hypercube," *IEEE Trans. on Computers*, Vol. 37, No. 7, pp. 867-871, July 1988.
6. C. Seitz, "The Cosmic Cube," *Commun. ACM*, Vol. 28, pp. 22-33, 1985.
7. A. Sen, "Supercube: An Optimally Fault Tolerant Network Architecture," *Acta Informatica*, Vol. 26, pp. 741-748, 1989.
8. A. Sen, A. Sengupta, S. Bandyopadhyay, "On the routing problem in faulty supercubes," *Information Processing Letters*, Vol. 42, pp. 39-46, 1992.
9. A. Sen, A. Sengupta, S. Bandyopadhyay, "Generalized Supercube: An incrementally expandable interconnection network," *Proceedings of the Third Symposium on Frontiers of Massively Parallel Computation-Frontiers'90*, pp. 384-387, 1990.
10. H. Sullivan, T. Bashkow, "A large scale, homogeneous, fully distributed parallel machine, I," *in Proc. 4th Symp. Computer Architecture, ACM*, pp. 105-177, March 1977.
11. S.-M. Yuan, and H.-M. Lien, "The Shortest algorithm for Supercube," *Proceedings of National Computer Symposium*, pp. 556-561, 1991.