Journal of Applied Sciences

ISSN 1812-5654

Journal of Applied Sciences 7 (18): 2666-2670, 2007
ISSN 1812-5654
© 2007 Asian Network for Scientific Information

Efficient Mapping Scheme of Ring Topology Onto Tree-Hypercubes

Wesam Almobaideen, Mohammad Qatawneh, Azzam Sleit, Imad Salah and Saleh Al-Sharaeh
Department of Computer Science, King Abdulla II School for Information Technology,
University of Jordan, Amman, Jordan

Abstract: The Tree-hypercubes network TH(s, d) is an interconnection network that is recursively defined and has excellent properties for scalable message passing computer systems. In this study we describe the ability of a TH(s, d) to implement algorithms that use the communication pattern of linear array and a ring to formulate a Hamiltonian cycle. We propose an algorithm for mapping a ring onto a tree-hypercubes which results in a dilation and congestion of one.

Key words: Tree-hypercube network, linear array, ring, Hamiltonian cycle, mapping

INTRODUCTION

One of the primary design considerations for a parallel system is the topology of the interconnection network used for communication between processors. Some of the important interconnection networks which have been studied so far are hypercube, trees, meshes and tree-hypercubes (Caha and Koubek, 2005; Gupta, 2000; Johnson, 1987; Ouyang and Palem, 1991; Leighton, 1992). Tree-hypercubes is one of the more popular general purpose networks due to its optimal fault tolerance properties, embedding of other networks and a fast sorting algorithm (Omari and Abu-Salem, 1997; Qatawneh, 2005, 2006). Omari and Abu-Salem (1997) presents a new interconnection network called Tree-hypercubes network which combines the advantages of both trees and hypercubes and avoids their shortcoming. The diameter of the tree-hypercubes is less than that of the hypercube.

One of the important properties of any general interconnection network is the ability of an interconnection network to efficiently emulate computation running on other networks. This attribute makes it a good candidate for a topology underlying a general-purpose parallel machine (Goldie and Kause, 1996; Chang and Chen, 1997; Gupta, 2000). A d-dimensional hypercube is one of the most popular interconnection networks for parallel systems. It has regularity and scalability to implement parallel algorithms and enables us to embed many other structures (e.g., linear arrays, rings, meshes and trees) as subgraphs (Tsai, 2004). The tree-hypercubes network can emulate many interconnection topologies such as linear array, ring, tree, hypercubes and meshes. The problem of mapping interconnection topologies into tree-hypercube network has not received much attention from researchers.

A ring is a fundamental topology for parallel and distributed processing. The following is verbatim from (Tsai, 2004). The importance of the ring structure for distributed computing is threefold. Firstly, the number of edges of the ring is low which causes the development cost of the ring structure to be reasonable. Secondly, the ring structure is free of branching. Consequently, much of potential non-determinism is eliminated, which allows for the development of simple algorithms. Thirdly, the ring network topology is often used as a connection structure for local area networks such as token rings (Kurose and Ross, 2005).

In this research, we present an algorithm for mapping ring topology into tree-hypercube network with a dilation and congestion of one.

TREE HYPERCUBE NETWORK

Here, we define tree-hypercubes and describe a recursive construction mechanism. A tree-hypercubes network TH(s, d) is constructed by taking a full tree of degree s, where s is a power of 2 and depth d. Levels of the tree are numbered 0, 1, … and d. Each level k has s^k nodes representing processing elements and labeled from 0 to s^k-1 in binary code and interconnected as a hypercube. Thus, nodes at level k constitute (k log s)-cube. Each node in a tree-hypercubes is identified by a pair (L, X), where L denotes the level number and X is the cube address. The total number of nodes in TH(s, d) is $N = (s^{d+1}-1)/(s-1)$. Figure 1 and 2 show two tree-hypercubes TH (2, 2) and TH (2, 3), respectively.

Corresponding Author: Wesam Almobaideen, Department of Computer Science,
King Abdulla II School for Information Technology, University of Jordan, Amman, Jordan

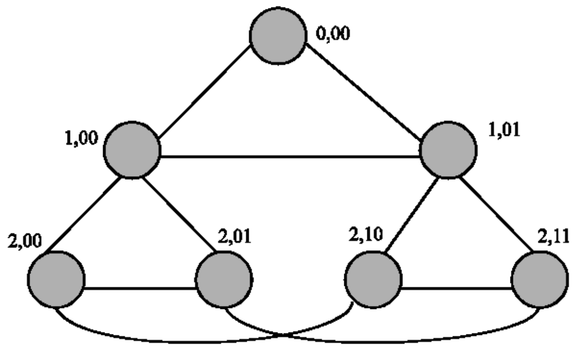

Fig. 1: Tree-hypercubes TH (2, 2)

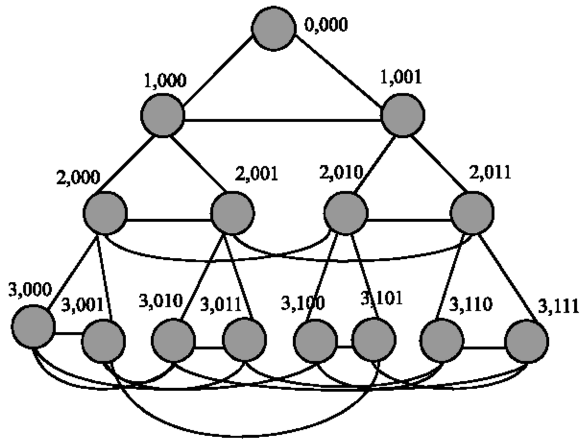

Fig. 2: Tree-hypercubes TH (2, 3)

A Tree-Hypercube TH(s, d) can be formally defined as follows.

Definition 1: Let X.a be the concatenation of the label X and the binary digit(s) a, where the levels and nodes are labeled as above. For every level $0 < L < d$, each node (L, X) in level L, where $X = X_{(d \log s)-1} \ldots . X_0$ adjacent to the following s children nodes at level L+1 (L+1, X.a) for a = 0,…,s-1 if L<d and to the parent node (L-1, $X_{(d \log s)-1} \ldots X_{\log s}$). Node (L, X) is also adjacent to L log s nodes (L, X) in the same level where Y's binary address differs from X in exactly one bit.

Definition 2: Tree-hypercube networks can be viewed as a recursive structure. TH(s, 0) is the smallest tree-hypercube and consists of one node only. TH(s, d) can be constructed from s copies of TH(s, d-1) by adding a new node to be the root of TH(s, d) and label it (0, 0) then connect it to the s TH(s,d-1)'s. Label these TH(s, d-1) networks by 0,1,…,s-1 and transform the label of each of the s nodes (0, X) to (1, X.i), where i = 0,1,… s-1 in binary notation. Then transform the label of each node (L, X) in the i-th network to (L+1, $iX_{(l \log s)-1}X_{(l \log s)-2} \ldots X_0$). Now, any two nodes (L, X.r) and (L,X.j) become adjacent if r and j differ in only one bit. As a result all nodes at level L from an (L log s)-cube.

PRELIMINARIES FOR MAPPING

Embedding a graph G (V, E) into G' (V', E') is a mapping of each vertex in the set V onto a vertex (or a set of vertices) in set V' and each edge in the set E onto an edge (or a set of edges) in E'. When mapping graph G (V, E) into G'(V', E'), the following must be taken into consideration. Firstly, one or more edge in E can be mapped onto a single edge in E'. The maximum number of edges mapped onto any edge in E' is called the congestion of mapping. Secondly, an edge in E may be mapped onto multiple contiguous edges in E'. This is significant because traffic on the corresponding communication link in G must traverse more than one link in G, possibly contributing to the congestion on the network. The maximum number of links in E' that any edge in E is mapped onto is called the dilation of mapping. Thirdly, the sets V and V' may contain different numbers of vertices. A node in V may correspond to more than one node in V' and vice versa. The ratio of the number of nodes in V' to that in V is called the expansion of mapping. If there is a good mapping of a graph G into G', then G' can simulate the behavior of G with less overhead.

Here, we consider embeddings of ring into tree-hypercube networks. Each level k in tree-hypercubes has s^k nodes representing processing elements and labeled from 0 to s^k-1 in binary and interconnected as a hypercube. In order, to obtain the decimal address for each node at level k we will map linear array at each level of tree-hypercubes (Note that the straightforward decimal encoding will not work with the proposed scheme for mapping ring into tree-hypercube). A linear array composed of 2^k nodes (labeled 0 through 2^k-1) can be embedded into a k-hypercube (level k in tree-hypercubes) by mapping node i of the linear array onto node G(i, k) of the hypercube at level k. The function G(i, x) is defined as follows:

$$G(0, 1) = 0$$
$$G(1, 1) = 1$$

$$G(i, x+1) = \begin{cases} G(i,x), & i < 2^x \\ 2^x + G(2^{x+1} - 1 - i, x), & i \geq 2^x \end{cases}$$

The function G is called the binary Reflected Gray Code (RGC) (Fig. 3). A careful look at the Gray Code

1-bit gray code	2-bit gray code	3-bit gray code	3-k hypercube	8-processor linear array
0	00	000	0	0
1	01	001	1	1
	11	011	3	2
	01	001	2	3
		101	6	4
		111	7	5
		101	5	6
		100	4	7

Fig. 3: A three-bit reflected Gray code ring

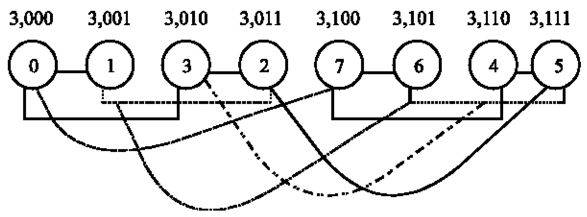

Fig. 4: Embedding 8-node linear array into third level of TH (2, 3)

table reveals that two adjoining entries (G (i, d) and G (i + 1, d)) differ from each other at only one bit position. Since node i in the linear array is mapped to node G (i, d) and node i + 1 is mapped to G (i + 1, d), there is a direct link in the hypercube that corresponds to each direct link in the linear array. (Recall that two nodes whose labels differ at only one bit position have a direct link in a hypercube.) Therefore, the mapping specified by the function G has a dilation of one and a congestion of one. Figure 4 shows an example that shows how we obtain the decimal address for each node at level k, where k = 3, t.e. third level in TH (2, 3), by embedding an eight-node linear array into a three-dimensional hypercube.

ALGORITHM FOR MAPPING RING TOPOLOGY INTO TREE-HYPERCUBE NETWORK

Algorithm for mapping ring described into tree-hpercubes. First we will map a linear array at each level of tree-hypercube networks TH(s, d), in order to obtain the decimal address for each node at each level k in TH(s, d). Then we will find the start node in last level (Note that the algorithm starts working from last level of TH (s, d)). The algorithm is shown in Fig. 5.

The following example explains the procedure of mapping a ring with 2^{d+1}-1 nodes into a TH (2, 3) (Fig. 6).

Step 1 : I = d-1 = 4-1 = 3; Start = Node = $2^1/2$ = 4 with binary address (3,110); Counter = 1.
Step 2 : Minus = false; Plus = false; Careful = false.
Step 3 : I ≠ 0; go to step 4. Step 4: Node ≠ (2^1/2)-1; go to step 5.
Step 5 : Node = (2^1/2); Plus = true; Minus = false; go to step 6
Step 6 : Counter ≠ (2^1/2); go to step 7.
Step 7 : Minus = false; go to step 8.
Step 8 : Node = Node + 1 = 4+1 = 5 with binary address (3,111); counter = 1+1 = 2; go to step 6.
Step 6 : Counter ≠ (2^1/2); go to step 7.
Step 7 : Minus = false; go to step 8.
Step 8 : Node = 5 +1 = 6 with binary address (3,101); Counter = 2+1 = 3; go to step 6.
Step 6 : Counter ≠ (2^1/2); go to step 7. Step 7: Minus = false; go to step 8.
Step 8 : Node = 6+1 = 7 with binary address (3,100); Counter = 3+1 = 4; go to step 6.
Step 6 : Counter = (2^1/2) = 4; Counter = 1; Node = Parent (7) = 3 with address (2,010); go to step 3.
Step 3 : I ≠ 0; then go to step 4.
Step 4 : Node = (2^1/2)-1 = 3; Minus = true; Plus = false; go to step 6.
Step 6 : Counter ≠ (2^1/2); go to step 7.
Step 7 : Minus = true; Node = Node-1 = 3-1 = 2 with binary address (2,011); Counter = 1+ 1 = 2; go to step 6.
Step 6 : Counter = (2^1/2); Counter = 1; Node = parent (2) = 1 with binary address (1,001).
Step 3 : I ≠ 0; go to step 4; Step 4: Node = (2^2/2)-1 = 1; Minus = true;
Step 6 : Counter = (2^1/2) = 1; Node = parent (1) = 0 with binary address (0,000); go to step3.
Step 3 : I = 0 then go to step 9.
Step 9 : Node = left Child (0) = 0 with binary address (1,000); Counter = 1;
Step 10 : Node = 2^1/2-1 = 0;
Step 11 : Plus = true; Minus = false;
Step 12 : Counter = 2^1/2 = 1; Node = left Child (0) = 0 with binary address (2,000);
Step 10 : Node ≠ 2^1/2-1; Step11: Plus = true; Minus = False;
Step 12 : Counter ≠ 2; Step 13: Careful and Node ≠0;
Step 14 : Minus = False; Step 15: Node = 0+1 = 1; Counter = 1+1 = 2

```
Step 1:   Here we define the start node. I = d; Start Node = Node = $2^I/2$; Counter = 1;
Step 2:   Minus = false; Plus = false; Careful = false.
Step 3:   If (I = 0) go to step 9
Step 4:   If (Node = ($2^I$/2) -1 Then Minus = true; Plus = false; go to step 6.
Step 5:   If (Node = ($2^I$/2) Then Plus = true; Minus = false;
Step 6:   If (Counter = ($2^I$/2) Then Counter = 1; Node = Parent (Node); go to step 3.
          (We find the parent node by decrement the level I by 1 I = I-1 and shift right one bit the binary address of that node then get its equivalent address
          in decimal)
Step 7:   If (minus) Then Node = Node-1; counter++; go to step 6.
Step 8:   Node = Node+1; Counter++; go to step 6.
          (Here we will go to the left sub tree of the root)
Step 9:   Node = Left Child (Node); Counter = 1;
          (We get left child by increment the level I by 1, I++ and shift left the binary
          address of that node and then get its equivalent address in decimal).
Step 10:  If (Node = ($2^I$/2-1) OR (Node = $2^I$/2-2 AND I > = 3)) Then
            Minus = true; Plus = false; go to Step 12.
Step 11:  Plus = true; Minus = false;
Step 12:  If (Counter = $2^I$/2) Then Counter = 1;
            If ((I = d-1) and (Node = $2^I$/2-1)) go to step 16;
                Else If (I = d-2 AND d-1 is Odd Number) Then
           Node = right Child (Node); Careful = true; go to step 10
          (Get Right child by increment the level I by 1, I++ and shift left the binary address of the
              node and increment it by 1 (right child) and then get its equivalent address in decimal).
                       Else
                     Node = left Child (Node) go to step 10.
Step 13:  If (Careful AND Node = 0) Then Node = $2^I$/2-1 go to step 16.
Step 14:  If (Minus) Then Node = Node -1; Counter++; go to step12.
Step 15:  Node = Node + 1; Counter++; go to step12.
Step 16:  Node = Start Node.
(Here we embedded a linear array and the last node is $2^I$/2-1 so connect it with start node, which is $2^I$/2 to get Ring)
```

Fig. 5: Mapping algorithm

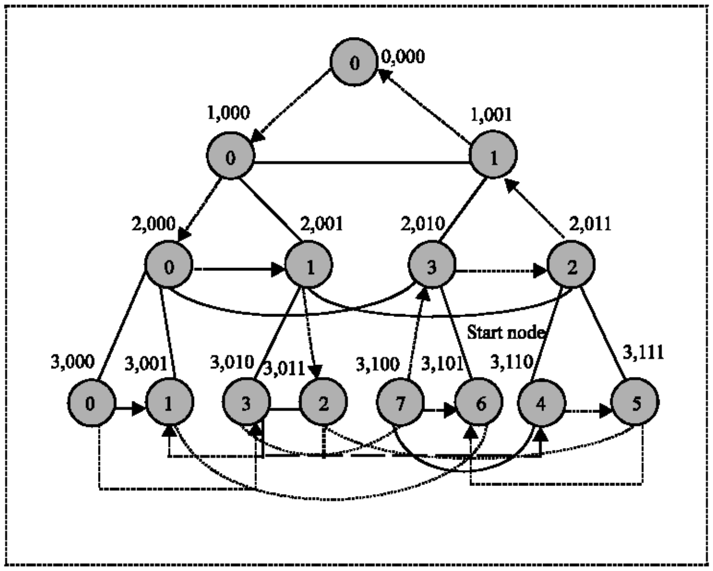

Fig. 6: Mapping a ring with 2^{d+1}-1 nodes into a TH (2, 3)

Step 12 : If Counter = (2^I/2); Then Counter = 1; I = = d -2 = 4-2 = 2 and d-1 3 is odd then Node = Right Child(1) with binary address (2,001); I = 2+1 = 3; Shift left(001) = 010 +1 = 011 then child node = Node = 2 with binary address (3,011); Careful = true;

Step 10 : Node = (2^3)2-1OR Node = (2^I)/2-2 and I> = 3; Minus = true; Plus = false;

Step 12 : Counter ≠ 4; Step 13: Careful and Node ≠ 0;

Step 14 : Minus = true; Node = 2-1 = 1 with binary address (3,001); Counter = 1+1 = 2;

Step 12 : Counter ≠ 4; Step 13: Careful and Node ≠ 0;

```
Step 14 : Minus = true and Node = 1-1 = 0 with binary
          address (3,000); Counter = 2+1 = 3;
Step 12 : Counter ≠ 4;
Step 13 : Careful and Node = 0; Node = 2^3/2-1 = 4-1 = 3;
          Counter = 3+1 = 4;
Step 12 : If Counter = 2^1/2) = 4 Then Counter = 1; I = =
          3 AND Node = = 3;
Step 16 : Node = Startnode = 4;
```

CONCLUSIONS

The problem of mapping interconnection topologies in tree-hypercube network did not receive sufficient attention from researchers. The tree-hypercube network, which combines the advantages of both trees and hypercube, can emulate many interconnection topologies such as linear array, ring, tree, hypercube and meshes topologies. We have proposed a new efficient approach for mapping ring onto tree-hypercube. We test our proposed algorithm onto a tree-hypercube of size TH(2,3) (i.e., 2^(3+1) -1 = 15 nodes) and a ring to be mapped of size 15, which result in a Hamiltonian cycle that utilized every node in targeted system (tree-hypercube) for that system is fully utilized. We calculated as shown previously in section four that the dilation factor of one, which indicates that we have one to one mapping as a result intercommunication between nodes, was reduced to one. Since our proposed algorithm result of a dilation of one it did not just only optimized the communication cost (one hop communication) it also result in an optimal solution to the congestion factor.

REFERENCES

Caha, R. and V. Koubek, 2005. Hamiltonian cycles and paths with a prescribed set of edges in hypercubes and dense sets. J. Graph Theory, 51: 137-169.

Chang, H.Y. and R.J. Chen, 1997. Embedding cycles in IEH graphs. Inform. Proc. Lett., 64: 23-27.

Goldie, A.W. and G. Krause, 1996. Embedding rings with Krull dimension in artinian rings. J. Algebra, 182: 534-545.

Gupta, A., 2000. Embedding trees into low dimensional Euclidean spaces. Discrete Comput. Geom., 24: 105-116.

Johnson, S.L., 1987. Communication efficient basic linear algebra computations on hypercube architectures. J. Parallel Distributed Computing, 4: 133-172.

Kurose, J.F. and K.W. Ross, 2005. Computer Networking, A Top-Down Approach Featuring the Internet. Pearson Education, Inc.

Leighton, F.T., 1992. Introduction to Parallel Algorithms and Architecture: Arrays, Trees, Hypercubes. Morgan Kaufmann, SanMateo, CA.

Omari, M. and H. Abu-Salem, 1997. Tree-Hypercubes: A Multiprocessor Interconnection Topology. Abhath Al-Yarmouk, 6: 9-24.

Ouyang, P. and K.V. Palem, 1991. Very efficient cyclic shifts on hypercubes. Proceedings of Symposium on Parallel and Distributed Processing, pp: 556-563.

Qatawneh, M., 2005. Embedding linear array onto tree-hypercube network. Eur. J. Sci. Res., 10: 2.

Qatawneh, M., 2006. Adaptive Fault Tolerant Routing algorithm in Tree-Hypercube Multicomputer. J. Comput. Sci., 2: 124-126, USA.

Tsai, C.H., 2004. Linear arrays and rings embedding in conditional faulty hypercubes. Theor. Comput. Sci., pp: 431-443.