SCALEABILITY OF A BINARY TREE ON A HYPERCUBE

Sanjay R. Deshpande

Department of Computer Sciences
University of Texas at Austin
Austin, Texas 78712

TR-86-01 January 1986

Abstract

The concept of scaleability is important for the next generation of super-multiprocessors. Two aspects of scaleability of an architecture are hardware scaleability and application scaleability. While the concept of hardware scaleability is a familiar one, the one of application scaleability is new and equally important. The paper introduces the latter concept through an example of hypercube architecture and binary tree application. It is observed that the mapping problem is intimately connected with the problem of application scaleability. The paper includes the proof that a binary tree application scales well on a hypercube.

Introduction

In this paper we investigate the scaleability of a complete, balanced binary tree on a hypercube. Specifically, we look at the question of mapping the largest sized binary tree on a hypercube graph. An important consideration today in designing an architecture for a super-multiprocessor system is the scaleability of the architecture [4,5]. By scaleability we imply the asymptotic incremental cost of building larger and larger systems based on the defining topology of the architecture. Smaller the incremental cost, more scaleable is the architecture. The concept of scaleability of an architecture is not very new and has been used in the past. The multistage interconnection networks such as the omega, the banyan and the baseline are prefered for multiprocessor systems over the crossbar for the very reason that these networks scale better than the crossbar. In gross terms, the hardware cost increases as NlogN for these networks while the same increases as N^2 for the crossbar, where N is the number of interconnected resources.

Related to the concept of hardware scaleability, but quite different, is the concept of application scaleability of a given architecture and is of great concern for the future multiprocessor systems. Application scaleability could make or break the promise of an architecture as a basis for dedicated high performance multiprocessors. The concept of application scaleability is subtle, and we have just begun to understand its importance. In this paper we try to demonstrate the concept by considering an application whose computation graph resembles a complete, balanced binary tree.

The concept of application scaleability involves the mapping problem [6]. In addition, it also involves accurately accounting for the architectural constructs embedded in software. It is conceivable that as the application and the supporting archiecture grow in size, the hardware scales gracefully, but the software architectural constructs do not. Usually these constructs are not accounted for, and may result in bottlenecks and degraded performance. Here we also demonstrate the importance and complexity of the mapping problem by attempting to evaluate the scaleability of a binary tree structure on a supporting hypercube architecture.

We have chosen an application graph which is a binary tree. Binary tree graphs are popular in the area of computer science and appear in several computation graphs. They naturally result from *divide and conquer* methodology of formulating parallel algorithms for sorting, merging and min-max problems. The binary tree based algorithm is used for the recursive doubling method of computing global histogram [8]. The Dictionary Machine of Atallah and Kosaraju uses a binary tree structured machine architecture [9]. Binary tree structures result from search trees in inference systems. Horowitz and Zorat suggest application of binary tree shaped interconnection network for multiprocessor systems [10]. It would therefore be interesting to see how such an important computational topology is supported on the hypercube architecture.

Recently, substantial attention has been given to the hypercube architecture for multiprocessors [1,2]. Many applications have been successfully attempted and impressive speed-ups have been obtained [3]. Most previous architectures have failed to furnish respectable speed-ups for more than just a few applications. On the other hand, hypercube architecture seems to have lived up to its promise of being a good supporting architecture for a wide variety of application areas. This effectiveness of the hypercube architecture is seen to derive from the rich internode connection topology afforded by the underlying graph. But, before hypercube topology can be considered as a defining topology for a future generation super-multiprocessor, its hardware scaleability should be evaluated. Also of consequence to its applicability for a dedicated high performance computing engine, is its application scaleability for the targeted use. Binary tree structured algorithms being so common in the area of parallel processing, we focus here on their scaleability on the hypercube.

Hardware Scaleability

Figure 5 shows an example of hypercube of order 3. In a hypercube based multiprocessor, nodes of the graph are occupied by independent processing elements. The edges between the nodes represent the point-to-point communication links between the processors. Each link or edge is dedicated to the corresponding node-pair. In some architectures like the Intel's iPSC System, there is a separate processor called the Cube Manager to coordinate the parallel execution of a job [11]. The Cube Manager is

connected to the processors in the cube by a broadcast bus for global communication, i/o and control. However, the concepts of Cube Manager and the global bus is not inherent to the hypercube architecture and we will therefore ignore them.

Each processing element in an n-cube has communication links to n other processors. If we focus on the number of processors $N = 2^n$, in the cube, then there are logN communication interfaces on each processor. Thus the total communication interface cost of the network is NlogN. This is same as the cost of the multistage interconnection networks like banyan, omega etc., although there is one minor difference: when the size of the hypercube is doubled, each of the original nodes has to be modified to allow one more communication link. However, this is not a serious drawback. The nodes of the original cube could be provided with ports for extra links, and when the cube is doubled one of these could be utilized.

Like many multistage interconnection networks hypercube is a non-planar graph and is most naturally implemented in three-dimensional space, with a point-to-point interconnection scheme. The volume of the hypercube architecture increases linearly with the number of nodes or processing elements.

The communication reliability of the hypercube scales much better than that of two-sided multistage interconnection networks. For most two-sided multistage networks like the banyan, baseline etc., unless extra stages are specifically provided for, there is a unique path from a source node to a destination node. The communication takes place directly between the two communicating nodes. If an interconnection node or a link in the network fails, it results in disconnection between one or more pairs of nodes. On the other hand, in the hypercube architecture, communication between a pair of nodes passes over a path involving other processing nodes. There are logN! paths available for any given pair of nodes. If one of the paths fails, the communication can still be completed over the other paths. Thus the number of contingent paths and the reliability of interprocessor communication increases with N.

The diameter of a network is defined as the maximum distance between the

source and destination nodes. For the multistage networks, the diameter is equal to the number of stages in the network, which is logN. For these networks, therefore, the diameter grows as logN. For a hypercube, the message passes at most logN links before reaching the destination. For any given node, there is one node which is at distance logN from it. So for the hypercube too, the diameter scales as logN.

Application Scaleability

For the sake of this analysis, we assume that the binary tree application is the only one being executed in the system (scaleability of an application in presence of other concurrent applications being beyond the scope of this paper). Under these conditions, we would like to be able to execute the largest possible size of the application. Ideally, we would like to utilize the hypercube architecture fully, so that, when the architecture and the application are scaled no wastage is observed. If that is not possible, a constant amount of resource-wastage is desirable, since the percentage of wastage would diminish with increase in the size of the application. The wastage that grows logarithmically or linearly with the number of nodes in the hypercube is much less desirable.

It is observed that a balanced binary tree application is not well scaleable on a hypercube. The largest size of the tree that can be mapped on an n-cube is of n-1 levels. Thus the tree occupies only 2^{n-1}-1 nodes out of 2^n available nodes. Resultant wastage of computing resources is $2^{n-1}+1$, which is slightly more than 50%. In terms of the total computing resource available in the system, the wastage is linear and therefore undesirable. Proof of this part is given in Appendix I.

The result obtained in Appendix I is very pessimistic and casts a serious doubt on the merit of hypercube as a topology for super-multiprocessors. However, an interesting and quite unexpected result obtained in Appendix II resurrects hypercube in terms of the scaleability of the binary tree on it. Appendix II shows that it is possible to slightly modify the original tree graph and make it well-scaleable on the cube. By introduction of single node at the first level of the tree, and thereby *stretching* of an edge out of its root, the tree can be made to utilize the cube completely. The extra node so introduced, is used only for communication between the root and one of its sons. It is

interesting to note that only one such node is required for any n. This node therefore represents a constant overhead (or wastage) for all $n \geq 3$.

Summary

The concepts of hardware scaleability and application scaleability of an architecture were introduced. Both types of scaleability are of significance for future super-multiprocessor systems. The concepts were illustrated by considering hypercube as an example architecture and the binary tree as an example application. Mapping of application on the architecture forms an important first step in analyzing application scaleability of the architecture, but no universal method is known for obtaining such a mapping. A clever modification of the algorithm graph may often lead to the architecture being scaleable. This modification might lead to overheads, but may still be desirable in terms of the overall utilization of system resources.

Acknowledgements

This research was supported by AFOSRF49620-84-C-0020, DCR-8116099, DE-AS05-81ER-10987 and Space and Naval Warfare Systems Command Contract N00039-86-C-0167.

Appendix I

Definition: A hypercube of order n (an n-cube) is an undirected graph with 2^n vertices labelled 0 through 2^n-1. There is an edge between a given pair of vertices if and only if the binary representation of their labels differ by one and only one bit. (An example of hypercube of order 3 is shown in Figure 5.)

Definition: Parity of a node is odd or even and is determined by the number of 1's in the label of the node.

Lemma 1: If nodes a and b are connected by an edge, a and b have opposite parities.

Proof: Let a be of even parity. since a and b are connected by an edge, their binary representations differ by one bit. The representation of b may be obtained by changing a "1" to a "0" or a "0" to a "1". In either case the number of "1"s in b's representation is odd. Thus b has odd parity. Similarly, if a's parity is odd, b's parity is odd.

We now try to map a binary tree on a hypercube.

Assume, without loss of generality, that the root of the tree is located at an even parity node. The root is designated level 0. By the above Lemma 1, the two sons of the root have odd parity. Thus, the nodes at level 1 have odd parity. Consider nodes at level i. All nodes at levels i-1 and i+1 are of opposite parity to that of level i. Thus parities alternate over the levels of the mapped binary tree.

We now count the numbers of odd and even nodes necessary to map a binary tree, and the number of odd and even nodes available in a hypercube:

By symmetry, The numbers of odd and even nodes available in the hypercube is same and is half the total number of nodes $= \frac{1}{2}(2^n) = 2^{n-1}$.

For a binary tree, we get the following values:

Case 1: n is even.

Number of even nodes $= 2^0+2^2+....+2^{n-2} = \frac{1}{3}(2^n-1)$

Number of odd nodes $= 2^1+2^3+....+2^{n-1} = \frac{2}{3}(2^n-1)$

Clearly, the numbers of even and odd nodes required by the binary tree do not match those available in the hypercube.

Case 2: n is odd.

Number of even nodes $= 2^0+2^2+....+2^{n-1} = \frac{1}{3}(2^{n+1}-1)$

Number of odd nodes $= 2^1+2^3+....+2^{n-2} = \frac{2}{3}(2^{n-1}-1)$

Again, the numbers of even and odd nodes required by the tree do not match those of the cube. This completes the proof.

Appendix II

In Appendix I we saw that a balanced, complete binary tree can not be mapped on a hypercube such that all but one nodes of the cube are occupied. This would mean mapping an n-level binary tree on an n-cube, where n-cube has 2^n nodes and the binary tree has 2^n-1 nodes.

Consider the following: Figure 1 shows a binary tree of n=3. The figure also shows the even and odd levels of the tree assuming level 0 of the tree is at an even level. Figure 2 shows the construction of a 4-level tree out of two 3-level trees using an extra node at level 1 of the right hand subtree. It is clear that the resultant tree utilizes 16 nodes of which 8 are even and 8 are odd. These can be matched by the even and odd nodes of a 4-cube respectively. In fact if an n-1 level binary tree has A even nodes and B odd nodes, then an n-level *stretched* binary tree created in the manner of figure 2 always has 2^n total number of nodes of which $A+B+1=2^{n-1}$ are even and an equal number are odd. Therefore an n-cube could provide the necessary even and odd nodes to map the n-level stretched binary tree. The conjecture we can make is that, whether such a mapping is at all possible. This is a conjecture because the match of the number of even and odd nodes between the two graphs is only a necessary condition but not a sufficient one.

The following is a constructive proof that such a mapping is indeed possible for $n \geq 3$.

Definition: An n-cube is said to be transformed in the ith dimension when the ith bit in the binary representation $x_{n-1}..x_1x_0$ of the label of each node of the cube is modified according to certain rule. The rule is called the transformation.

Definition: An n-cube is said to be k-dimensionally transformed when k bits in the label of each node of the cube are transformed according to a

transformation. (Note that the bits undergoing transformation need not be contiguous.)

We define two 3-dimensional transformations, FT3 and BT3. The two transformations are shown in figure 6. The x_i, x_j and x_k are the original values of the bits, and y_i, y_j and y_k are the corresponding resultant values.[1]

Definition: Distance between a pair of nodes in an n-cube is the number of bits the binary representations of the labels of the two nodes differ by.

Definition: Adjacency in an n-cube is the distance relationship of each node with the rest of the nodes in the cube. (This relationship is usually captured in the distance matrix for a graph. In general graph theoretic terms, distance refers to the minimum number of edges that have to be traversed from the source node to the destination node. For an n-cube this corresponds to distance as defined above.)

We now prove that both FT3 and BT3 maintain the adjacency of an n-cube. That is, after label-transformations every node has the same distance relationships with other nodes as before.

Theorem 1: FT3 maintains cube adjacency.

Proof: Let $S_1x_iS_2x_jS_3x_kS_4$ be the binary representation, of length n, of a node-label in an n-cube. Bits x_i, x_j and x_k are the bits of interest. S_1, S_2, S_3 and S_4 are strings of bits of length zero or more.

For distance calculations S_1, S_2, S_3 and S_4 can be combined into a single string S. A node label can thus be represented as $Sx_ix_jx_k$. After transformation the label becomes $Sy_iy_jy_k$.

Let $|Sx_ix_jx_k|_P$ and $|Sx_ix_jx_k|_Q$ be the original labels of nodes P and Q. The distance between nodes P and Q is given by the number of bits by which labels of P and Q differ. The contribution to the distance comes from two parts of their labels, namely S and x_i, x_j and x_k. Thus,

---

[1] The two transformations define two of the many *rigid* transformations of a 3-cube. The rigid transformations include rotations and reflections of the original cube. Rigid transformation can be looked upon as the transformations that maintain the adjacency relationships between nodes of the cube. Transformations of rotation and reflection of a cube form a *group* [7].

$$\text{Distance}(P, Q) = \text{Distance}(|S|_P, |S|_Q) + \text{Distance}(|x_i x_j x_k|_P, |x_i x_j x_k|_Q)$$

After FT3 transformation,

$$\text{Distance}(P, Q) = \text{Distance}(|S|_P, |S|_Q) + \text{Distance}(|y_i y_j y_k|_P, |y_i y_j y_k|_Q)$$

To maintain adjacency, Distance(P, Q) should not change from its value before the transformation. The first term on the right hand side in the above two expressions is same. To maintain adjacency then, we must prove

$$\text{Distance}(|x_i x_j x_k|_P, |x_i x_j x_k|_Q) = \text{Distance}(|y_i y_j y_k|_P, |y_i y_j y_k|_Q)$$

The contribution to the distance between nodes P and Q by three bits is 0, 1, 2 or 3. If the contribution to the initial distance between P and Q by bits x_i, x_j and x_k is 0, then it implies that the three bits under consideration were identical for P and Q. The bits will be transformed identically for both P and Q, and therefore will not contribute to changing the distance between the two nodes. Therefore Distance(P, Q) is unchanged after FT3 transformation for those nodes which have identical combinations for the designated three bits. For the cases where the bits contribute 1, 2, or 3 counts to the distance, the post-FT3 adjacency is proved in Table 1 exhaustively. The table shows for each combination of three bits, combinations of bits at distances 1, 2 and 3. It also shows the transformed values of bits for node pairs and demonstrates that adjacency is indeed maintained after the transformation. This concludes the proof.[2]

Theorem 2: BT3 maintains cube adjacency.

Proof: The considerations involved here are the same as above. Table 2 gives the exhaustive proof for BT3.

---

[2]One does not have to prove the above using the exhaustive method of Table 1. As mentioned before, the rotations and reflections of a 3-cube can be considered to be rigid transformations which essentially leave relative positions of the nodes of the cube intact. By focusing on three bits of the labels in an n-cube, we analyze the projection of the n-cube in the three desired dimensions. Each node in this projection corresponds to 2^{n-3} nodes of the original n-cube. FT3 can be shown to be a combination of rotations of this three-dimensional projection about three orthogonal axes, and a reflection about a plane passing through the projection.

We can now state the following corollaries:

Corollary 1: Nodes with distinct labels map to distinct resultant labels.

Corollary 2: If two nodes are adjacent to each other and thus have labels differing in one bit position, they get consistent labels after the transformations.

Definition: A graph G is said to be mapped on an n-cube when the nodes in G map on to nodes of the n-cube in a one-to-one fashion, and the edges in G map on to those of the n-cube.

Theorem 3: A graph G mapped on an n-cube remains unchanged in structure after the transformations.

Proof: The proof follows as a direct consequence of corollaries 1 and 2. By virtue of the fact that the graph was initially mapped on the cube, the nodes of the graph correspond to the nodes of the cube and edges of the graph correspond to the edges of the cube; clearly the graph has no self-loops since they can not be mapped on to the edges of an n-cube. The adjacent nodes in the graph are mapped on to the adjacent nodes of the cube. After the transformations, the resultant nodes of the cube, and therefore of the graph, are still adjacent and the edges of the graph remain intact. The transformations do not result in collapse of nodes or stretching of edges of the graph by introducing extra nodes.

Corollary 3: The transformations FT3 and BT3 preserve a tree structure mapped on an n-cube.

Theorem 4: A stretched binary tree (of form shown in Figure 2) of level n can always be mapped on an n-cube, for $n \geq 3$ (such that all the nodes of the cube are utilized).

Proof: We give an inductive proof for this theorem.

The case of n=3 is shown in Figure 3a. Figure 3 shows the process of going from n=3 to n=4. We shall list the steps involved in this process:

- Duplicate the cube and the mapping.
- Apply FT3 on the least significant 3 bits of the duplicate cube (Figure 3b).

- Create a 4-cube by connecting the nodes in one cube to the nodes with the same label in the other cube. Append a 0 to the left of labels in the original 3-cube, and a 1 to the left of labels in the transformed 3-cube.

- Deallocate links: 0100-0110 and 1100-1000, and allocate links: 0100-1100, 0000-1000 and 0110-1110. Figure 3c shows only the useful links between the two subcubes.

- Apply BT3 to the most significant bit, the third significant bit and the least significant bit of the 4-cube.

Figure 3d shows the 4 level stretched binary tree embedded in the resultant 4-cube.

We will now describe the process for the inductive step. We assume that an n level stretched binary tree, as shown in Figure 4a, is mapped on an n-cube. Its root is assumed to be located at node S000, where S is an n-3 bit-long string of most significant bits. The extra node is located at S100. Two n-1 level binary (unstretched) subtrees hang from nodes S001 and S110.

- Duplicate the cube and the mapping. See Figure 4b.

- Apply FT3 to the duplicate cube using bit-2 as x_i, bit-1 as x_j and bit-0 as x_k. See Figure 4c.

- Form an n+1-cube by connecting the nodes with like labels in the two n-subcubes. Append a 0 to the left of labels in the original subcube and a 1 to the left of duplicate subcube. See Figure 4d.

- Deallocate links: 0S100-0S110 and 1S100-1S000, and allocate links: 0S000-1S000, 0S110-1S110 and 0S100-1S100. We get the structure shown in Figure 4e. Notice that the figure shows only the important edges.

- Apply BT3 to the n+1-cube to obtain an n+1 level stretched binary tree rooted at 0S000. Use the most significant bit as x_i, the third least significant bit as x_j and the least significant bit as x_k. Replace string 0S by S' to obtain structure similar to the base structure. See Figures 4f and 4g.

References

1 "The Mark III Hypercube-Ensemble Concurrent Computer", J. C. Peterson et al., ICPP 1985.

2 "The Cosmic Cube", C. Seitz, CACM, January 1985.

3 "CALTECH/JPL Mark II Hypercube Concurrent Processor", J. Tuazon et al., ICPP 1985.

4 "Inductive Computer Architectures: A Class of Supercomputer Architectures", G. J. Lipovski, M. Malek and J. C. Browne.

5 "Microprocessors: Architecture and Applications", P. C. Patton, IEEE Computer, June 1985.

6 "On the Mapping Problem", S. Bokhari, IEEE Transactions on Computers, March 1981.

7 Finite Reflection Groups, 2nd Ed., L. C. Grove and C. T. Benson, Springer-Verlag.

8 Introduction to Computer Architecture, H. Stone, Science Research Associates Inc., 1975.

9 "A Generalized Dictionary Machine for VLSI", M. J. Atallah and S. R. Kosaraju, IEEE Transaction on Computers, February 1985.

10 "The Binary Tree As An Interconnection Network: Applications to Multiprocessor Systems and VLSI", E. Horowitz and A. Zorat, IEEE Transactions on Computers, April 1981.

11 iPSC System Overview, Intel Corporation, October 1985.

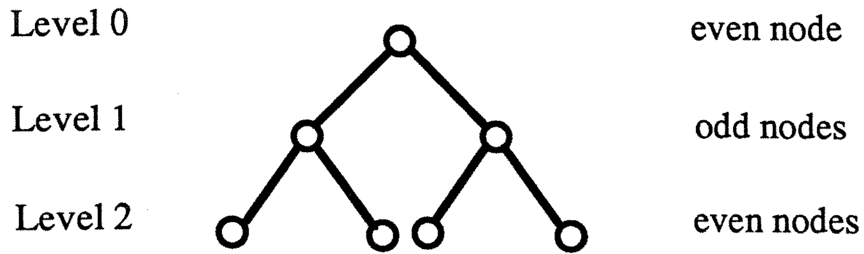

Figure 1

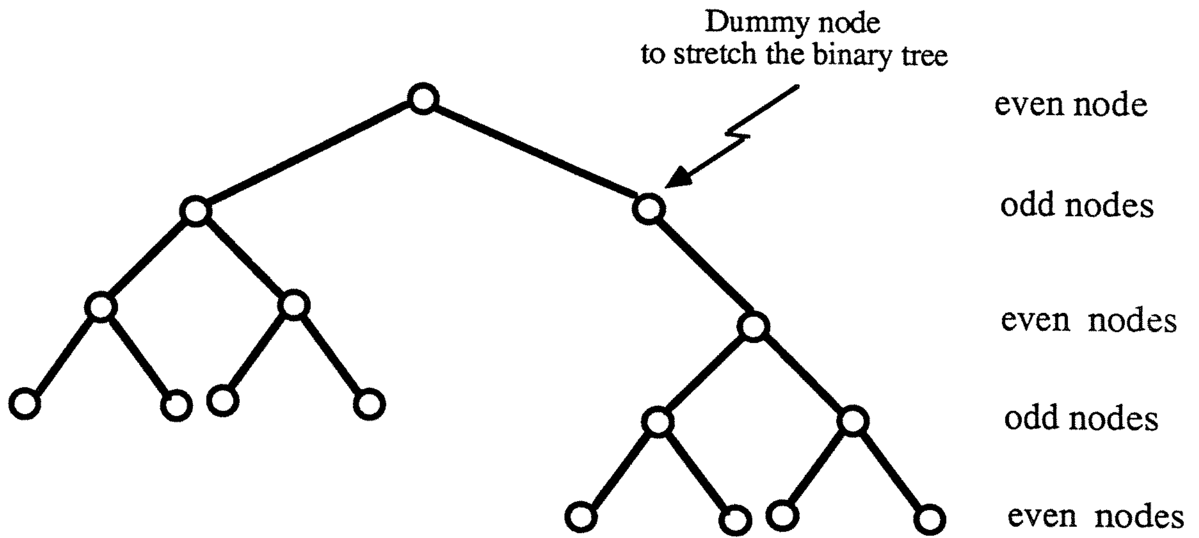

Figure 2

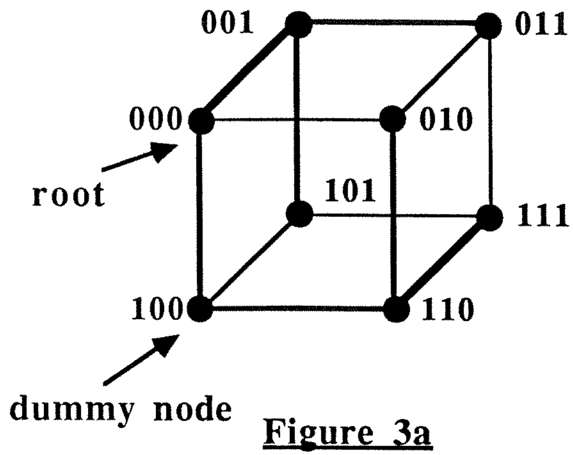

Figure 3a

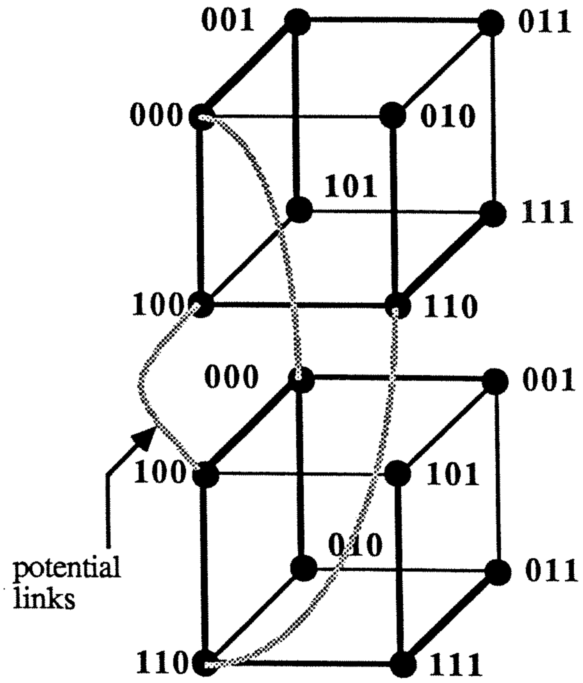

Figure 3b

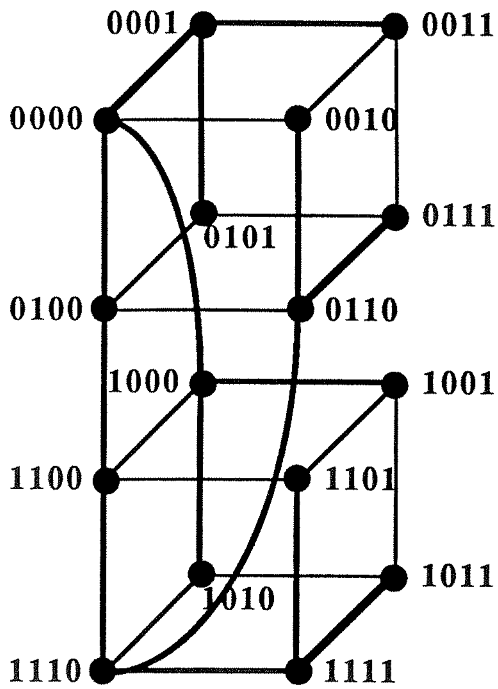

Figure 3c

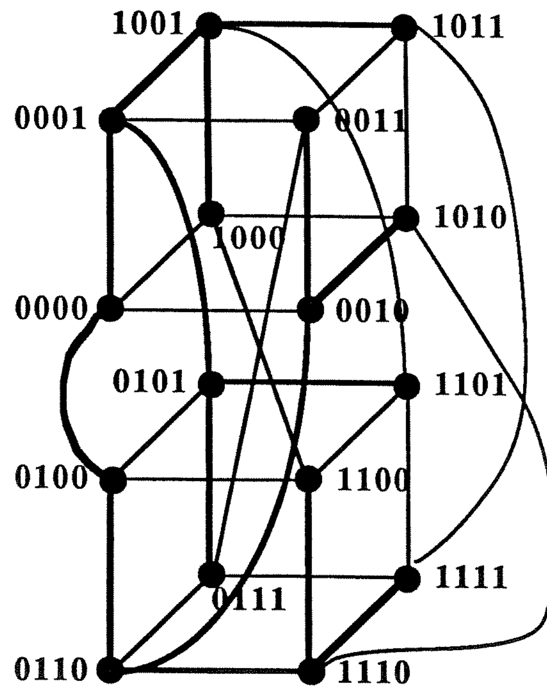

Figure 3d

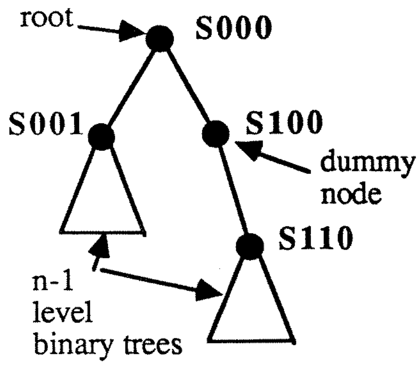

Figure 4a

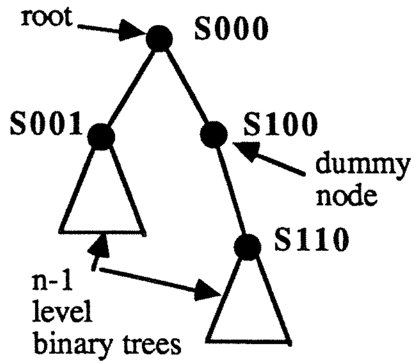

Figure 4b

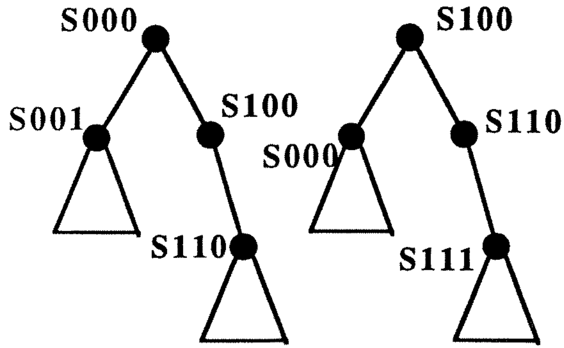

Figure 4c

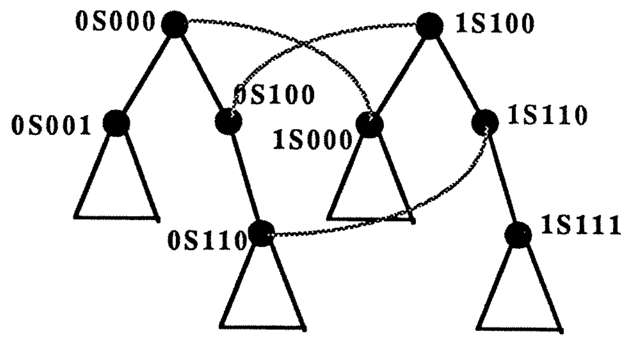

Figure 4d

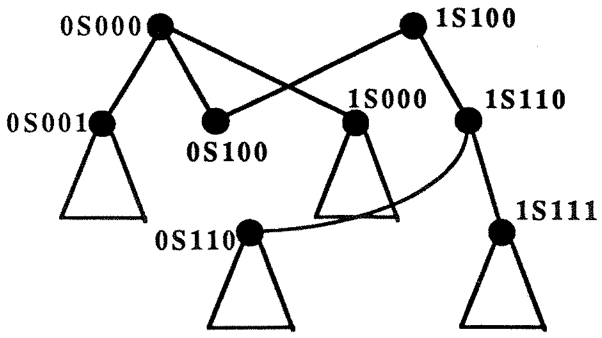

Figure 4e

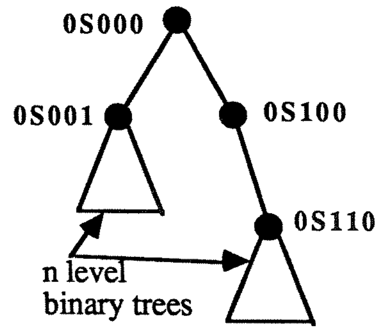

Figure 4f

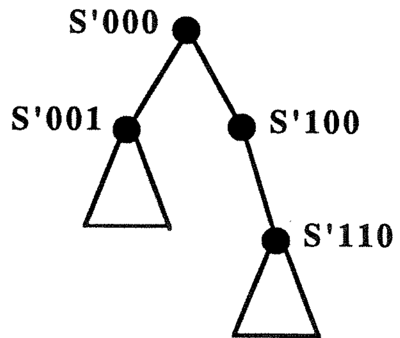

Figure 4g

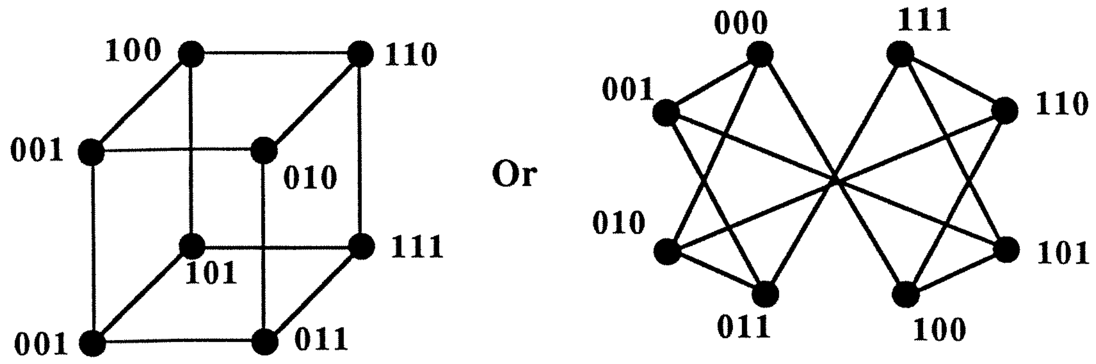

Figure 5

FT3						BT3					
x_i	x_j	x_k	y_i	y_j	y_k	x_i	x_j	x_k	y_i	y_j	y_k
0	0	0	1	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	1	1	0	1
0	1	0	1	0	1	0	1	0	0	0	0
0	1	1	0	0	1	0	1	1	1	0	0
1	0	0	1	1	0	1	0	0	0	1	1
1	0	1	0	1	0	1	0	1	1	1	1
1	1	0	1	1	1	1	1	0	0	1	0
1	1	1	0	1	1	1	1	1	1	1	0

Figure 6

		Distance of 3		Distance of 2		Distance of 1		
P	**P'**	**Q**	**Q'**	**Q**	**Q'**	**Q**	**Q'**	√/X
000	100	111	011	011 101 110	001 010 111	001 010 100	000 101 110	√
001	000	110	111	010 100 111	101 110 011	011 000 101	001 100 010	√
010	101	101	010	001 100 111	000 110 011	000 011 110	100 001 111	√
011	001	100	110	000 101 110	100 010 111	001 010 111	000 101 011	√
100	110	011	001	001 010 111	000 101 011	000 110 101	100 111 010	√
101	010	010	101	011 000 110	001 100 111	001 100 111	000 110 011	√
110	111	001	000	000 011 101	100 001 010	010 100 111	101 110 011	√
111	011	000	100	001 010 100	000 101 110	011 101 110	001 010 111	√

Table 1

		Distance of 3		Distance of 2		Distance of 1		
P	**P'**	**Q**	**Q'**	**Q**	**Q'**	**Q**	**Q'**	**√/X**
000	001	111	110	011 101 110	100 111 010	001 010 100	101 000 011	√
001	101	110	010	010 100 111	000 011 110	011 000 101	111 001 100	√
010	000	101	111	001 100 111	101 011 110	000 011 110	001 100 010	√
011	100	100	011	000 101 110	001 111 010	010 001 111	000 101 110	√
100	011	011	100	001 010 111	101 000 110	000 101 110	001 111 010	√
101	111	010	000	011 000 110	100 001 010	001 100 111	101 011 110	√
110	010	001	101	000 011 101	001 100 111	010 100 111	000 011 110	√
111	110	000	001	001 010 100	101 000 011	011 101 110	100 111 010	√

Table 2