ALTERNATELY-TWISTED CUBE
AS
AN INTERCONNECTION NETWORK

by

WONG Yiu Chung

A thesis submitted to the
Department of Computer Science
The Chinese University of Hong Kong
in partial fulfillment of the requirements
for the degree of
Master of Philosophy

May 1991

Table of Contents

Acknowledgement

I am greatly indebted to Prof T.C. Chen, my supervisor, for his patient and enlightening guidance throughout the course of my graduate study.

Alternately-Twisted Cube as an Interconnection Network

Abstract

A new network topology called the alternately-twisted cube is proposed. It is based on a modification to the topology of the binary n-cube, or hypercube, by "twisting" its edges along the odd-numbered dimensions.

An alternately-twisted n-cube, denoted as AQ_n, has a diameter of only $\lfloor \frac{n}{2} \rfloor + 1$, which is nearly half of that of the binary n-cube. At the same time, it preserves many salient features of the binary n-cube. It is shown that an AQ_n is node-symmetric, possesses n distinct paths between any 2 nodes, and is able to be partitioned into smaller, disjoint alternately-twisted subcubes. Furthermore, we have specified schemes to embed the following structures into an AQ_n: any HxW grids of size $\leq 2^n$(with dilation 1 if H and W are powers of 2, and dilation 2 otherwise), a complete binary tree of size $2^n - 1$(with dilation 2) and any ring of size k, for k$\leq 2^n$ and $k \neq 3$ (with dilation 1).

In addition, the alternately-twisted cube appears to be more attractive than the binary n-cube as a general purpose interconnection network. We have devised a distributed, shortest-path routing algorithm for the network. Analytic results show that in general it can route messages faster than the hypercube: about 22% smaller in the mean internode distance, nearly 50% smaller in the diameter measure, and nearly 30% shorter in the average message delay under heavy load, when the network size is large. The improvement is better when the dimension of the AQ_n is an odd number than when it is even. Broadcasting on the AQ_n, under the multiple-message accepting mode, takes only $\lfloor \frac{n}{2} \rfloor + 1$ routing cycles, again about 50% of that on the binary n-cube.

The ability of the AQ_n for supporting parallel processing is demonstrated by mapping the parallel versions of the following algorithms onto it: the Ascend/Descend class of algorithms, the combining class of algorithms, and the

algorithms for solving Poisson-type partial differential equations, matrix multiplication, and Gaussian elimination. All but the last two of them can be run on the AQ_n as efficiently as on the hypercube, and for the last two algorithms, the former behaves even better.

Chapter 1
Introduction

The success of Seitz's experimental work in building the Cosmic Cube [Seitz85], showing that current technology is ready for building general purpose multiprocessor systems, has stimulated the construction of a number of commercial parallel machines in the second half of the last decade. The sizes of these machines range from below a hundred to tens of thousands, and is projected to reach a million and beyond within this decade. Therefore the performance of the interconnection network is significant to the efficiency of the parallel machine. It is controlled by two factors: the network topology and the communication method employed. In this thesis we shall concentrate on the first issue only. Interested readers are referred to the literature for the second (e.g. the description given in [Kung89]).

Many of the parallel machines are based on the hypercube network for interconnecting their processing elements. Examples include Ncube's hypercube machines, Intel's iPSCs, the Connection Machine, as well as Seitz's Cosmic Cube. The hypercube, also known as the binary n-cube, draws its popularity from many of its salient features: node- and edge-symmetry, small diameter(see below), existence of simple and distributed routing algorithm, low node degree, efficient simulation of other networks, and fault tolerance capability, to name a few.

Formally, a binary n-cube is defined as follows, using graph notation:

(i) a binary 1-cube is a complete graph of two nodes, named as 0 and 1;

(ii) a binary n-cube, for n>1, is a graph consisting of 2 binary (n-1)-cubes, the names of whose nodes are prefixed by 0 and 1 respectively, and they are joined in the way below:

node 0u (in one of the binary (n-1)-cube) is connected to node 1u (in the other binary (n-1)-cube) by an edge, where u is any binary string of length (n-1).

The binary 3-cube, for example, is shown in Figure 1(a). It can be easily shown that there are 2^n nodes in a binary n-cube, but the worst-case distance among all the node pairs, or the diameter in graph-theoretic terms, is only n. Also, the average internode distance is about $\frac{n}{2}$ [SaSc88].

In spite of the already excellent properties of the hypercube, it seems that it is always possible to improve on some of them by modifying the topology, incurring little or no extra cost to the corresponding network. For example, Tzeng [Tzen90] proposed the Variant Hypercube as a hypercube with additional links connecting pairs of nodes which are farthest away from each other in the original hypercube. By this way he succeeded in reducing the diameter of the resultant network by nearly 50% of that of the original hypercube. As an example, Figure 1(b) depicts the variant hypercube of dimension 3.

Esfahanian *et al.* [Esfa88] [Esfa91] proposed another way of modifying the binary n-cube: by "twisting" exactly one pair of edges in the cube. The graph so obtained is called the twisted n-cube. As an example, the twisted 3-cube is shown in Figure 1(c). The effect of the twist helps to shorten the distance between some pairs of nodes in the graph. As a result, the diameter is brought down to n-1, if n is the dimension of the cube, which is one fewer than that of the corresponding hypercube.

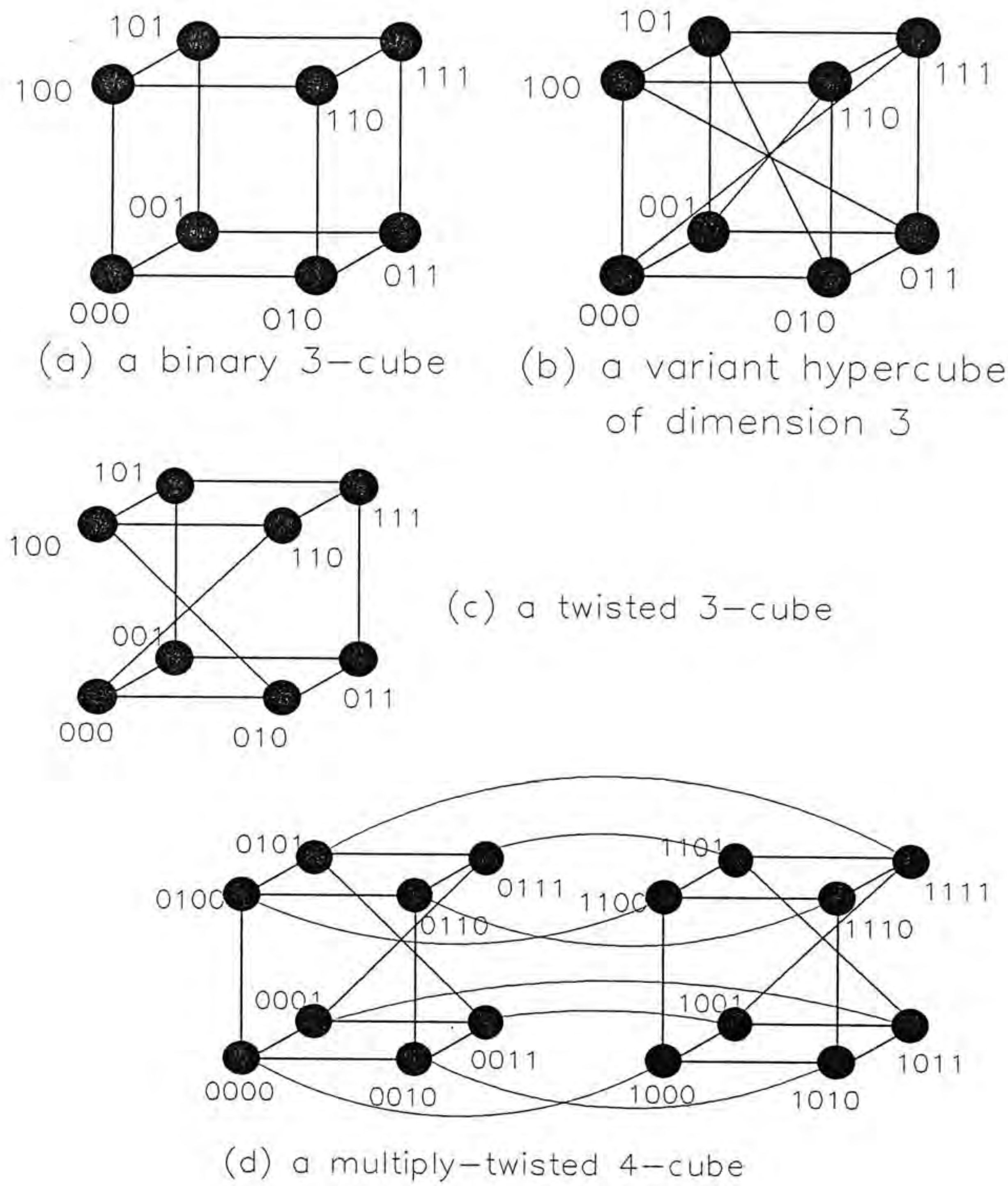


Figure 1: Examples of hypercube & hypercube-like networks

Efe [Efe89] further extended the idea of edge-twisting in the binary n-cube, and arrived at the multiply-twisted n-cube topology. His idea was to apply the twisting operation to edges along all dimensions of the hypercube. A formal definition is given below [Efe89]:

(MQ_n is the shorthand notation for multiply-twisted n-cube)

(i) MQ_1 is the complete graph of the set of 2 nodes {0, 1};

(ii) (let MQ^0_{n-1} and MQ^1_{n-1} be 2 graphs of MQ_{n-1} with the names of all their nodes prefixed by 0 and 1 respectively)

For n>1, MQ_n is the graph containing MQ^0_{n-1} and MQ^1_{n-1} joined as follows: nodes $0u_{n-2}u_{n-3}\ldots u_0$ and $1v_{n-2}v_{n-3}\ldots v_0$ are adjacent iff

1) $u_{n-2} = v_{n-2}$ if n is even, and,

2)

$$(u_{2i+1}u_{2i}, v_{2i+1}v_{2i}) \in \{(00,00),(10,10),(11,01),(01,11)\}$$

$$\text{for all } 0 \le i < \left\lfloor \frac{n-1}{2} \right\rfloor$$

It is easy to verify that MQ_3 is the same graph as the twisted 3-cube. In Figure 1(d) we show the graph of MQ_4 as an example. Again the multiply-twisted n-cube possesses a shorter diameter than the binary n-cube ($\lfloor \frac{n}{2} \rfloor + 1$, to be exact), and a shorter average internode distance as well.

In this thesis, we propose and investigate yet another edge-twisting modification to the hypercube topology. We call it the alternately-twisted n-cube, as the twisting operation is applied to edges along alternate dimensions only. (We came across Efe's paper [Efe89] after having started the work reported in this thesis, and have already formalized the topology of the alternately-twisted n-cube. Our twisted cube happens to be quite close, but not isomorphic, to his.) Some of the striking features of the alternately-twisted n-cube network include:

1) a diameter of $\lfloor \frac{n}{2} \rfloor + 1$, which is nearly half of that of the binary n-cube, (obtained with the same hardware cost as the binary n-cube network, assuming each link has the same cost);
2) an average internode distance about 22% less than that of the binary n-cube, when n is large;
3) an average message delay about 30% less than that of the binary n-cube, under heavy load, when n is large;
4) the ability to simulate efficiently other common network structures including the ring, the grid, the complete binary tree, and the hypercube;
5) nearly 50% reduction in the amount of time needed to broadcast a message to all the nodes of the network, as compared to that of the binary n-cube;
6) for executing parallel algorithms, time complexity to within a factor of 2 as that for running the algorithms on the binary n-cube.

The present thesis is structured as follows. Chapter 2 gives a formal definition of the alternately-twisted n-cube, and an analysis of its graph-theoretic properties. Chapter 3 examines the network performance of the topology. In Chapter 4, we shall show that various classes of parallel algorithms can be efficiently supported by the alternately-twisted n-cube. Finally, the alternately-twisted n-cube is compared with the variant hypercube, the twisted n-cube, and the multiply-twisted n-cube in Chapter 5, where a conclusion is also given.

Chapter 2
Alternately-Twisted Cube:
Definition and Graph-Theoretic Properties

2.1. Construction

A binary hypercube of dimension n can be regarded as constructed by connecting pairs of corresponding nodes in two identical binary hypercubes of dimension (n-1). Figure 2.1 shows such a construction of a 4-cube from two 3-cubes. Such pairing operations account for the symmetry property of the hypercube network and the routing simplicity of the network. One may modify this connection pattern, however, by twisting the pairings so as to result in a new network of relatively better performance. An example is given in Figure 2.2, which shows a 3-cube with one pair of its edges twisted: edges (000, 010) and (100, 110) are replaced by the edges (000, 110) and (100, 010) respectively. It can be seen that the diameter of this cube is reduced from 3 to 2, obtained with no additional requirement to the node degree nor the total number of edges of the underlying graph. This is an instance of the alternately-twisted cube network being examined here.

In this chapter we shall specify a scheme to twist the edges of the hypercube systematically. Note, however, that there are at least 2 other networks in the literature resulting from twisting the hypercube edges, in ways different from ours. They are the Twisted Cube proposed by Esfahanian, *et al.* [Esfa88] and the Multiply-Twisted Cube proposed by Efe [Efe89]. A comparison of our network with these, together with another hypercube variant, will be deferred to Chapter 5. In

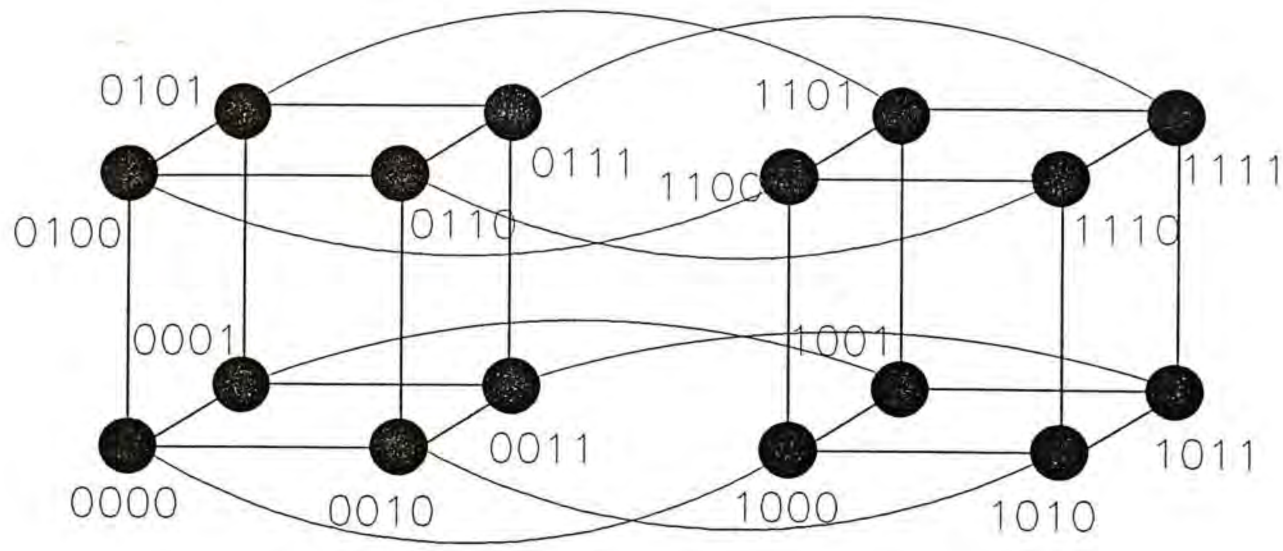


Figure 2.1: Pairing two binary 3-cubes to form a binary 4-cube

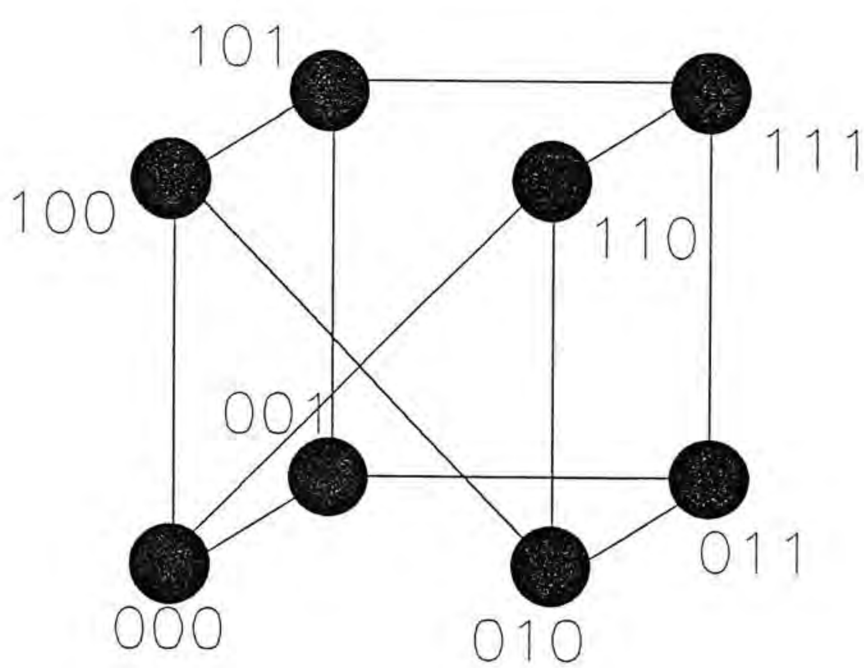


Figure 2.2: A 3-cube being "twisted"

the present chapter we shall concentrate on the topological properties of the alternately-twisted cube, along with a comparison to the hypercube only. The general construction of our network is defined from the graph-theoretic point of view. Before going into a formal definition of the twisted cube, however, we need some notations.

We define a T-code sequence (T for Twisted) which will be useful in defining the linkages of an alternately-twisted cube. Let *a*.S refer to the sequence obtained by prefixing all the elements of the sequence S with the string *a*, and S1,S2 be the sequence obtained by appending sequence S2 to sequence S1. The sequence of the T-code for 2 elements is denoted by T_1, and is defined below

$$T_1 \equiv < 0, 1 >$$

and the (sequence) reverse of T_1 is

$$T_1^R \equiv < 1, 0 >$$

Based on these the sequence of a T-code for $N=2^{2n}$ elements is defined as

$$T_{2n} \equiv (0.T_{2n-1}),(1.T_{2n-1}^R)$$

and that for $N=2^{2n+1}$ elements as

$$T_{2n+1} \equiv (0.T_{2n}),(1.T_{2n})$$

For example, T_2 is the sequence <00, 01, 11, 10> and T_3 denotes the sequence <000, 001, 011, 010, 100, 101, 111, 110>.

We can see that the T-code is quite similar to the reflected Gray code. In fact the latter can be defined as follows:

$$G_1 \equiv T_1$$

$$G_{n+1} \equiv (0.G_n), (1.G_n^R) \qquad \text{for} \quad n \geq 1$$

That is, the reflected Gray code of size 2^m is formed by concatenating the sequence of a Gray code of size 2^{m-1} with its reverse, with appropriate prefixing to the two respective Gray codes. T-code is a permutation of the reflected Gray code sequence, in the sense that T_m is constructed from two T_{m-1}'s with "reversing" only if m is even. As an example, Figure 2.3 lists the T-code and the reflected Gray code sequences of size 32.

Note that each element in a T-code sequence is a binary string. Denote the (j+1)th element of the sequence T_i by T_i (j), where 1≤i and $0 \leq j \leq 2^i - 1$. Then the conversion between T_i (j) and the binary representation of j can be effected by the following formulae, assuming that the binary forms of T_i (j) and j are represented by $t_{i-1} t_{i-2} .. t_1 t_0$ and $j_{i-1} j_{i-2} ... j_1 j_0$ respectively, where the t_k's and j_k's are bits: (⊕ denotes the binary exclusive-or below)

(i) Ordinal number to T-code address transformation

$$T_i(j) = t_{i-1} t_{i-2} ... t_1 t_0$$

where

$$for \quad 0 \le k \le \left\lfloor \frac{i-1}{2} \right\rfloor :$$

$$t_{2k} = j_{2k+1} \oplus j_{2k}$$

$$t_{2k-1} = j_{2k+1} \oplus j_{2k-1}$$

$$(assuming \quad j_i = 0 \quad \text{in both cases})$$

$$if \quad i \quad is \quad even, \quad then \quad t_{i-1} = j_{i-1}$$

(ii) T-code address to ordinal number transformation

$$T_i^{-1}(t_{i-1}t_{i-2}..t_1t_0) = j_{i-1}j_{i-2}..j_1j_0$$

where

$$\text{for} \quad 0 \le k \le \frac{m-1}{2} \quad \text{where} \quad m = \{ \begin{matrix} i-1 & if & i-1 & is & odd \\ i-2 & if & i-1 & is & even \end{matrix}$$

$$j_{2k+1} = t_m \oplus t_{m-2} \oplus \ldots \oplus t_{2k+3} \oplus t_{2k+1}$$

$$j_{2k} = t_m \oplus t_{m-2} \oplus \ldots \oplus t_{2k+3} \oplus t_{2k+1} \oplus t_{2k}$$

$$\text{and} \quad j_{i-1} = t_{i-1} \quad \text{if} \quad i-1 \quad \text{is even}$$

Figure 2.3 shows the conversion for the T_5 code sequence. Imagine that for a T-code sequence, we mark every other element with an '*', starting with the first one. Then it can be seen that a binary string $t = t_{n-1}t_{n-2}..t_1t_0$ is marked in the T_n sequence if and only if $(t_m \oplus t_{m-2} \oplus \ldots \oplus t_3 \oplus t_1) \oplus t_0 = 0$, where m=n-1 if n is even, and m=n-2 otherwise (i.e. m is the largest odd number less than or equal to n-1). Then we say that the parity of the string in the T_n sequence is 0 if it is so marked,

j	binary form of j	$T_5(j)$	$G_5(j)$	j	binary form of j	$T_5(j)$	$G_5(j)$
0	00000	00000	00000	16	10000	10000	11000
1	00001	00001	00001	17	10001	10001	11001
2	00010	00011	00011	18	10010	10011	11011
3	00011	00010	00010	19	10011	10010	11010
4	00100	00100	00110	20	10100	10100	11110
5	00101	00101	00111	21	10101	10101	11111
6	00110	00111	00101	22	10110	10111	11101
7	00111	00110	00100	23	10111	10110	11100
8	01000	01110	01100	24	11000	11110	10100
9	01001	01111	01101	25	11001	11111	10101
10	01010	01101	01111	26	11010	11101	10111
11	01011	01100	01110	27	11011	11100	10110
12	01100	01010	01010	28	11100	11010	10010
13	01101	01011	01011	29	11101	11011	10011
14	01110	01001	01001	30	11110	11001	10001
15	01111	01000	01000	31	11111	11000	10000

Figure 2.3: The sequences of T_5 & G_5 (for the T_5 elements, each underlined one has a parity of 0, otherwise it has a parity of 1)

otherwise its parity is 1. For example, in Figure 2.3, the even elements (i.e. those with parity 0) of the T_5 sequence are underlined. Therefore, we define a parity function π for the position of t in the T_n sequence as

$$\pi(t_{n-1}t_{n-2}\ldots t_1t_0) = p \oplus t_0$$

$$\text{where} \quad p = t_{2k-1} \oplus t_{2k-3} \oplus \;..\; \oplus t_3 \oplus t_1$$

$$\text{and} \quad k = \left\lceil \frac{n-1}{2} \right\rceil$$

For example, $\pi(00101) = 1$, $\pi(01101) = 0$, and $\pi(101101) = 1$, and note that $T_5^{-1}(00101) = (00101)_2 = (5)_{10}$, $T_5^{-1}(01101) = (01010)_2 = (10)_{10}$, and $T_6^{-1}(101101) = (110101)_2 = (53)_{10}$.

Now we can come to the construction of the alternately-twisted n-cube from the graph-theoretic view. Throughout this thesis, we take the convention that nodes are denoted by small letters, and that for any node u, the binary address is denoted by the string $u_{n-1}u_{n-2}\ldots u_1u_0$ where n is the length of the binary node address. For a node set V whose elements are binary strings, we use the notation V^w to refer to the set {wx| x is in V} (note: juxtaposition of two strings means concatenation), where w is a binary string, i.e. each element of V is prefixed by the string w. This notation is extended to an edge set E for the meaning of E^w.

An alternately-twisted 1-cube, denoted as AQ_1, is the graph (V_1, E_1) with node set $V_1 = \{0,1\}$ and edge set $E_1 = \{ (0,1) \}$. That is, it is a graph consisting of just two nodes joined by an edge, and is isomorphic to the binary 1-cube. Actually AQ_2 is also isomorphic to the binary 2-cube. But the coincidence ends here. The topologies of the AQ_n and the binary n-cube will be different for n greater than 2. Basically the node addresses of an alternately-twisted n-cube are the same as that of a binary n-cube. Only the connection pattern is different. The idea of its structure follows:

An AQ_{2n+1} is constructed from 4 AQ_{2n-1}'s . Suppose for each such AQ_{2n-1}, the nodes are visited according to the positions of their binary addresses in the T_{2n-1}-code sequence, while at the same time we mark every other node with an * along the traversal, starting with the first node (i.e. node 00...0 in each AQ_{2n-1}). Then corresponding marked nodes in these 4 AQ_{2n-1}'s are joined together in a "twisted-edge" fashion (say, 00u -> 10u -> 01u -> 11u -> 00u, where u is the address of a marked node), and those not marked are respectively joined exactly as in the normal hypercube (i.e. 00u -> 01u -> 11u -> 10u -> 00u). By cutting an AQ_{2n+1} into 2 equal halves along the plane orthogonal to the next-to-highest dimension (i.e. dimension 2n-1), we get 2 AQ_{2n}'s. As a result, only edges along the odd-numbered dimensions may be twisted (we take the convention that the dimension numbers are counted from 0), and for the other dimensions the hypercube linkages are preserved. (Refer to the definition below and the examples in Figure 2.4)

For example, 4 AQ_3's are joined to form an AQ_5 as follows. On traversing each AQ_3 according to the T_3 sequence order, nodes 000, 011, 100, and 111 are marked. In order to identify each AQ_3, their node addresses are respectively prefixed with the strings 00, 01, 11, and 10. Therefore, the 4 nodes of the form xy000 are joined together in the "twisted edges" manner, the 4 nodes of the form xy001 are joined together in the "hypercube edges" manner, those of the form xy011 are joined together in the "twisted edges" manner, those of the form xy010 are joined together in the "hypercube edges" manner, and so on. By splitting the resultant AQ_5 along the plane orthogonal to the 3rd dimension (i.e. by removing the edges along the 3rd dimension), we get 2 AQ_4's.

Formally, the structure of the alternately-twisted cube is defined recursively as follows:

(i) An alternately-twisted (2n)-cube, for n>0, denoted as AQ_{2n}, is the graph (V_{2n}, E_{2n}) where the node set is given by (the + operator between 2 sets means union)

$$V_{2n} = V^0_{2n-1} + V^1_{2n-1}$$

and the edge set is given by

$$E_{2n} = E^0_{2n-1} + E^1_{2n-1} + \{(0u, 1u) \mid u \in V_{2n-1}\}$$

(Note: The edges in the set $\{(0u, 1u) \mid u \in V_{2n-1}\}$ are the normal hypercube edges as needed in the pairing of two binary (2n-1)-cubes to form a binary (2n)-cube.)

(ii) An alternately-twisted (2n+1)-cube, for n≥0, denoted as AQ_{2n+1}, is the graph (V_{2n+1}, E_{2n+1}) where the node set is given by

$$V_{2n+1} = V^{00}_{2n-1} + V^{01}_{2n-1} + V^{10}_{2n-1} + V^{11}_{2n-1}$$

and the edge set is given by

$$E_{2n+1} = E^{00}_{2n-1} + E^{01}_{2n-1} + E^{10}_{2n-1} + E^{11}_{2n-1} +$$

$$\{(00u, 10u), (10u, 11u), (11u, 01u), (01u, 00u) \mid u \in V_{2n-1} \text{ and } \pi(u) = 1\} +$$

$$\{(00u, 10u), (10u, 01u), (01u, 11u), (11u, 00u) \mid u \in V_{2n-1} \text{ and } \pi(u) = 0\}$$

(Note: This is again very similar to the definition of a binary (2n+1)-cube, except that edges of the forms (10u,01u) or (11u,00u) are used, instead of the normal hypercube edges of the forms (10u,11u) or (01u, 00u), in the last component subset of E_{2n+1} given above. It is these "twisted edges" that account for the differences between the alternately-twisted cube and the hypercube.)

As an example, Figure 2.4 illustrates the graphs of AQ_i for i=1, 2, 3, and 4.

It can be seen that the alternately-twisted n-cube has a growth rate of 2, i.e. to expand from an AQ_n to an AQ_{n+1}, one has to double the number of nodes. This is exactly the same growth rate as that of a binary n-cube. In fact the alternately-twisted cube is derived from the hypercube with roughly a quarter of its pairs of edges being "twisted": the edges along alternate dimensions (the odd-numbered ones, with the convention that the dimension numbers are counted from 0 to n-1) are twisted, and the twisting operation is applied to half of the edges along each such dimension.

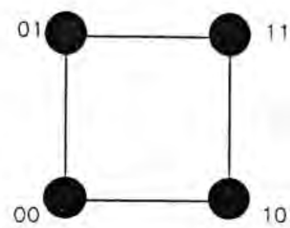
The connectivity rule for the alternately-twisted n-cube can be specified as follows: suppose u and v are two nodes in AQ_n, they are adjacent if and only if, for almost all $0 \leq i \leq n-1$, $u_i = v_i$, with the lone exception being index k≥0 such that either

(i) k is even and only $u_k \neq v_k$;

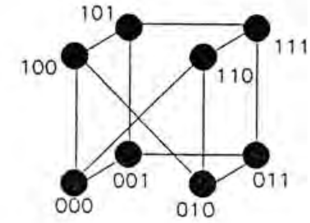
or (ii) k is odd and k=n-1 and $u_k \neq v_k$.

or (iii) k is odd and 0<k<n-1 and

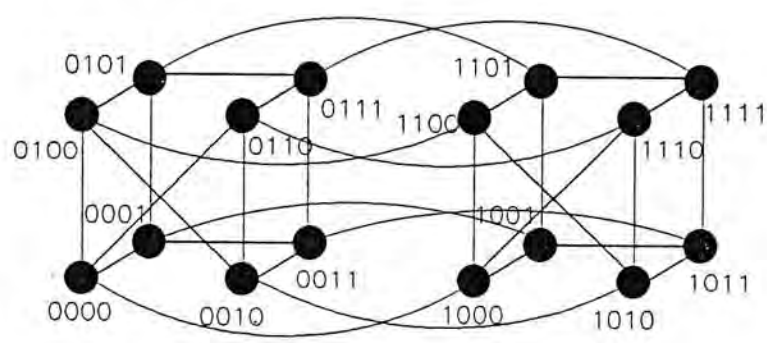
(AQ_1)



(AQ_2)



(AQ_3)



(AQ_4)

Figure 2.4: Examples of alternately-twisted cubes

$$\begin{cases} \text{only } u_k \neq v_k & \text{if } \pi\ (u_{k-1}u_{k-2}..u_1u_0) = 1 \\ u_k \neq v_k \text{ and } u_{k+1} \neq v_{k+1} & \text{if } \pi\ (u_{k-1}u_{k-2}..u_1u_0) = 0 \end{cases}$$

We call the linkage arising from one of these conditions the edge along the k-th dimension, with the value of k defined in the condition. Besides, the concept of dimensions of the alternately-twisted cube is borrowed exactly from that of the corresponding hypercube. Therefore, only in the (2i+1)-th dimensional plane may we find "twisted" edges. For example, the 5 edges incident from the node 00011 in a AQ_5 are respectively (the differing bits are underlined)

	along the 0th dimension:	$0001\underline{1}$ -> $0001\underline{0}$	
	along the 1st dimension:	$000\underline{1}1$ -> $000\underline{0}1$	
	along the 2nd dimension:	$00\underline{0}11$ -> $00\underline{1}11$	
	along the 3rd dimension:	$\underline{00}011$ -> $\underline{11}011$	*(a twisted edge)*
and	along the 4th dimension:	$\underline{0}0011$ -> $\underline{1}0011$.	

2.2. Topological Properties

2.2.1. Node Degree, Link Count, and Diameter

From the definition of the alternately-twisted cube, it is easy to see that each node of an AQ_n has direct connections to n distinct nodes, hence the degree of each node is n, or the AQ_n is an n-regular graph. The total number of nodes in an AQ_n is 2^n, so the total count of the edges in the graph is $n2^{n-1}$. Again these two measures are exactly the same as those of a binary hypercube of the same size. However, the alternately-twisted cube is superior to the hypercube in the worst-case distance measure, or the diameter of the graph: it is found to be about 50% of that of the corresponding hypercube.

Theorem 2.1: The diameter of an alternately-twisted n-cube, AQ_n, is equal to $\lfloor \frac{n}{2} \rfloor + 1$.

(Partial) Proof: The binary address $u_{n-1}u_{n-2}..u_1u_0$ of a node u in AQ_n is partitioned into groups of 1 or 2 bits as follows:

$(u_{n-1}u_{n-2})$, $(u_{n-3}u_{n-4})$, ..., (u_2u_1), (u_0) if n is odd

or (u_{n-1}), $(u_{n-2}u_{n-3})$, $(u_{n-4}u_{n-5})$, ..., (u_2u_1), (u_0) if n is even.

The number of groups in each case is $\lfloor \frac{n}{2} \rfloor + 1$.

From the connectivity rule of the alternately-twisted cube, it is easy to see that a transition along any edge adjacent to u will affect the address bits in at most one group of such partition. Hence the lower bound of the diameter will be $\lfloor \frac{n}{2} \rfloor + 1$, the number of groups in the partition. We defer the rest of the proof to Section 3.1, where a routing algorithm for the alternately-twisted cube network is proposed, and the worst-case number of routing cycles of the algorithm is shown to meet this lower bound.

For example, one of the most distant pair of nodes in an AQ_5 is 00000 and 11111, and a shortest path between them is: 00000 -> 11000 -> 11110 -> 11111, the length of which being 3.

2.2.2. Node Symmetry

The conditional "twisting" of the pairs of edges (as specified by the π function) makes the alternately-twisted cube unlikely to be edge-symmetric (at least, it can be proved by enumeration that the alternately-twisted 3-cube is edge-asymmetric). However, it still possesses the node-symmetric property, that is, each node in the alternately-twisted cube has the same view of the whole topology. There is no differentiation among the nodes, and there exists an address transformation that enables any node u to be mapped to another node v while the topology of the alternately-twisted cube is preserved after the transformation (i.e. the transformation is an automorphism). Formally, we have the following theorem:

Theorem 2.2: Let a and b be respectively any two nodes in an alternately-twisted (2n+1)-cube. There exists an automorphism $\alpha_{a\to b}$ in which node a is mapped to node b and, as a result, node x is mapped to node y, i.e. $\alpha_{a\to b}(x)=y$, according to the following rules:

(i) $y_0 = x_0 \oplus (a_0 \oplus b_0)$

(ii) for $1 \le k \le n$,

$$y_{2k-1} = x_{2k-1} \oplus (a_{2k-1} \oplus b_{2k-1})$$

$$y_{2k} = \hat{x}_{2k} \oplus (\hat{a}_{2k} \oplus b_{2k})$$

where $\hat{x}_{2k} = x_{2k} \oplus (x_{2k-1} \wedge (\pi(a_{2k-2}a_{2k-3}\ldots a_1 a_0) \oplus \pi(b_{2k-2}b_{2k-3}\ldots b_1 b_0)))$

and $\hat{a}_{2k} = a_{2k} \oplus (a_{2k-1} \wedge (\pi(a_{2k-2}a_{2k-3}\ldots a_1 a_0) \oplus \pi(b_{2k-2}b_{2k-3}\ldots b_1 b_0)))$

($\wedge$ *is the binary* AND *operation*)

The same applies to the automorphism for an alternately-twisted (2n)-cube, except that the above transformation for bit 2n is ignored.

Proof: First we show that the mapping is one-to-one, i.e. let $p'=\alpha_{a\to b}(p)$ and $q'=\alpha_{a\to b}(q)$, then $p'=q'$ iff p=q. Consider 3 cases for the position of the rightmost differing bit between p and q:

(1) at bit 0, i.e. $p_0 \ne q_0$, then by rule (i) $p'_0 \ne q'_0$

(2) at bit 2k-1, for some 1≤k≤n, then by rule (ii) $p'_{2k-1} \ne q'_{2k-1}$ since $p_{2k-1} \ne q_{2k-1}$

(3) at bit 2k, for some 1≤k≤n, since $p_{2k-1} = q_{2k-1}$ and $p_{2k} \neq q_{2k}$ then by rule (ii) $p'_{2k} \neq q'_{2k}$

Hence the mapping is indeed a permutation of the whole node set of the graph.

Next we turn to show that the mapping preserves the connectivity of the alternately-twisted cube, i.e. nodes x and y are adjacent in the alternately-twisted cube iff nodes $x' = \alpha_{a \to b}(x)$ and $y' = \alpha_{a \to b}(y)$ are adjacent. Again we consider 3 cases:

Case 1: x and y are joined by an edge along the 0th-dimension, ie. $x_i \neq y_i$ for i=0 only. By rule (i) of the mapping, $x'_0 \neq y'_0$ and $x'_i = y'_i$ for 1≤i≤2n. Therefore x' and y' are still adjacent via an edge along the 0th-dimension.

Case 2: x and y are joined by an edge along the (2k)th-dimension, ie. $x_i = y_i$ except for i=2k. Then for all i in the ranges 0≤i≤2k-1 and 2k+1≤i≤2n, $x'_i = y'_i$. By rule (ii), x'_{2k} depends on x_{2k}, so $x'_{2k} \neq y'_{2k}$. Thus x' and y' are adjacent via an edge along the (2k)th-dimension.

Case 3: x and y are connected through an edge along the (2k-1)th-dimension. Then $x'_i = y'_i$ for all i in the ranges 0≤i≤2k-2 and 2k+1≤i≤2n. Let

$$p = a_{2k-1} \oplus b_{2k-1}$$

$$q = \hat{a}_{2k} \oplus b_{2k}$$

$$\delta = \pi(a_{2k-2}a_{2k-3}\ldots a_1 a_0) \oplus \pi(b_{2k-2}b_{2k-3}\ldots b_1 b_0)$$

Then for the bit pairs at positions 2k and 2k-1, we have

$x'_{2k-1} \neq y'_{2k-1}$ (since $x_{2k-1} \neq y_{2k-1}$)

$x'_{2k} = x_{2k} \oplus (x_{2k-1} \wedge \delta) \oplus q$

$y'_{2k} = y_{2k} \oplus (y_{2k-1} \wedge \delta) \oplus q$

In other words,

if $x_{2k} = y_{2k}$ then

$$\begin{cases} \delta = 0 \Rightarrow x'_{2k} = y'_{2k} \\ \delta = 1 \Rightarrow x'_{2k} \neq y'_{2k} \end{cases}$$

if $x_{2k} \neq y_{2k}$ then

$$\begin{cases} \delta = 0 \Rightarrow x'_{2k} \neq y'_{2k} \\ \delta = 1 \Rightarrow x'_{2k} = y'_{2k} \end{cases}$$

It remains to show that x' and y' are adjacent along the (2k-1)th dimension. From rule (ii) of the mapping, we get

$\pi(x'_{2k-2}x'_{2k-3}\dots x'_1 x'_0) = \pi(x_{2k-2}x_{2k-3}\dots x_1 x_0) \oplus \delta$, and

$\pi(y'_{2k-2}y'_{2k-3}\dots y'_1 y'_0) = \pi(y_{2k-2}y_{2k-3}\dots y_1 y_0) \oplus \delta$

and by the connectivity rule of the alternately-twisted cube,

$(x_{2k} = y_{2k}) \wedge (x_{2k-1} \neq y_{2k-1})$

$\Rightarrow \pi(x_{2k-2}x_{2k-3}\dots x_0) = \pi(y_{2k-2}y_{2k-3}\dots y_0) = 1$

$\Rightarrow \pi(x'_{2k-2}x'_{2k-3}\dots x'_0) = \pi(y'_{2k-2}y'_{2k-3}\dots y'_0) = \bar{\delta}$

and, $(x_{2k} \neq y_{2k}) \wedge (x_{2k-1} \neq y_{2k-1})$

$\Rightarrow \pi(x_{2k-2}x_{2k-3}\dots x_0) = \pi(y_{2k-2}y_{2k-3}\dots y_0) = 0$

$\Rightarrow \pi(x'_{2k-2}x'_{2k-3}\dots x'_0) = \pi(y'_{2k-2}y'_{2k-3}\dots y'_0) = \delta$

Hence the values of $\pi(y'_{2k-2}y'_{2k-3}\dots y'_0)$ and of $y'_{2k}y'_{2k-1}$ are given by the table below

$x_{2k} \overset{?}{=} y_{2k}$	δ	$\pi(y'_{2k-2}y'_{2k-3}\cdot\cdot y'_0)$	$y'_{2k}y'_{2k-1}$
Y	0	1	$x'_{2k}\overline{x}'_{2k-1}$
Y	1	0	$\overline{x}'_{2k}\overline{x}'_{2k-1}$
N	0	0	$\overline{x}'_{2k}\overline{x}'_{2k-1}$
N	1	1	$x'_{2k}\overline{x}'_{2k-1}$

Thus nodes x' and y' are adjacent through an edge along the (2k-1)th-dimension. (Q.E.D.)

For example, the mapping $\alpha_{00000 \rightarrow 11111}$ applied to the nodes 00000 and 01010 and to their respective direct neighbours in an AQ_5 is illustrated below, where the bracketed number following the neighbour's address refers to the corresponding edge which is responsible for the adjacency:

original address	mapped to
00000	11111
neighbours:	neighbours:
00001 (0)	11110 (0)
00110 (1)	11101 (1)
00100 (2)	11011 (2)
11000 (3)	00111 (3)
10000 (4)	01111 (4)

01010	10001
neighbours:	neighbours:
01011 (0)	10000 (0)
01100 (1)	10011 (1)
01110 (2)	10101 (2)
00010 (3)	11001 (3)
11010 (4)	00001 (4)

2.2.3. Subcube Partitioning

Since the alternately-twisted cube is defined recursively, it is natural to ask how many distinct smaller alternately-twisted cubes can be embedded in an alternately-twisted n-cube.

Let $\rho_{n-1}\rho_{n-2}\ldots\rho_1\rho_0$ be a ternary string of length n whose alphabet is {0, 1, X}. We say that a binary string x conforms to ρ if, for 0≤i≤n-1, either

(i) $\rho_i = 0$ and $x_i = 0$, or

(ii) $\rho_i = 1$ and $x_i = 1$, or

(iii) $\rho_i = X$ and x_i is either 0 or 1.

Let S be a set of ternary strings, and $\sigma(AQ_n, S)$ denote the subgraph of an AQ_n induced by the nodes whose binary addresses conform to ρ, $\rho \in S$. For example, the graph $\sigma(AQ_5, \{XX00X\})$ is shown in Figure 2.5. It can be seen that this is also a graph of AQ_3, with each node of the form $u_2u_1u_0$ being mapped to the node $u_2u_100u_0$ of $\sigma(AQ_5, \{XX00X\})$. In a similar way, the graph $\sigma(AQ_5, \{XX11X\})$ is also an AQ_3 (Figure 2.6), but the mapping of nodes has to

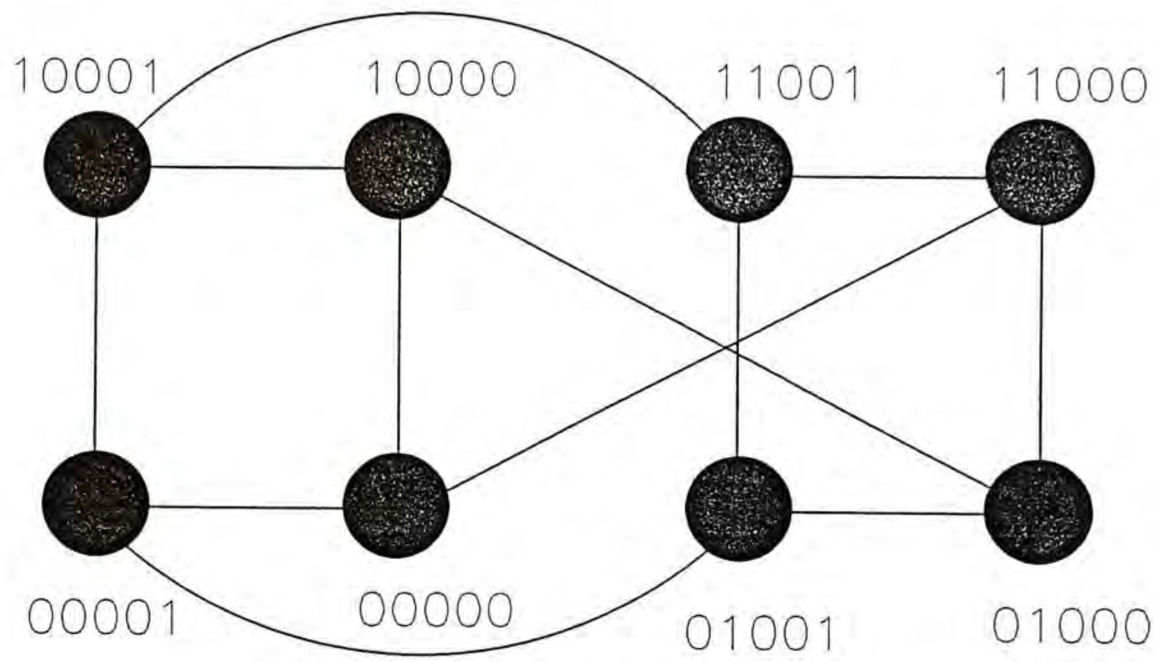


Figure 2.5: The induced graph $\sigma(AQ_5.\{XX00X\})$

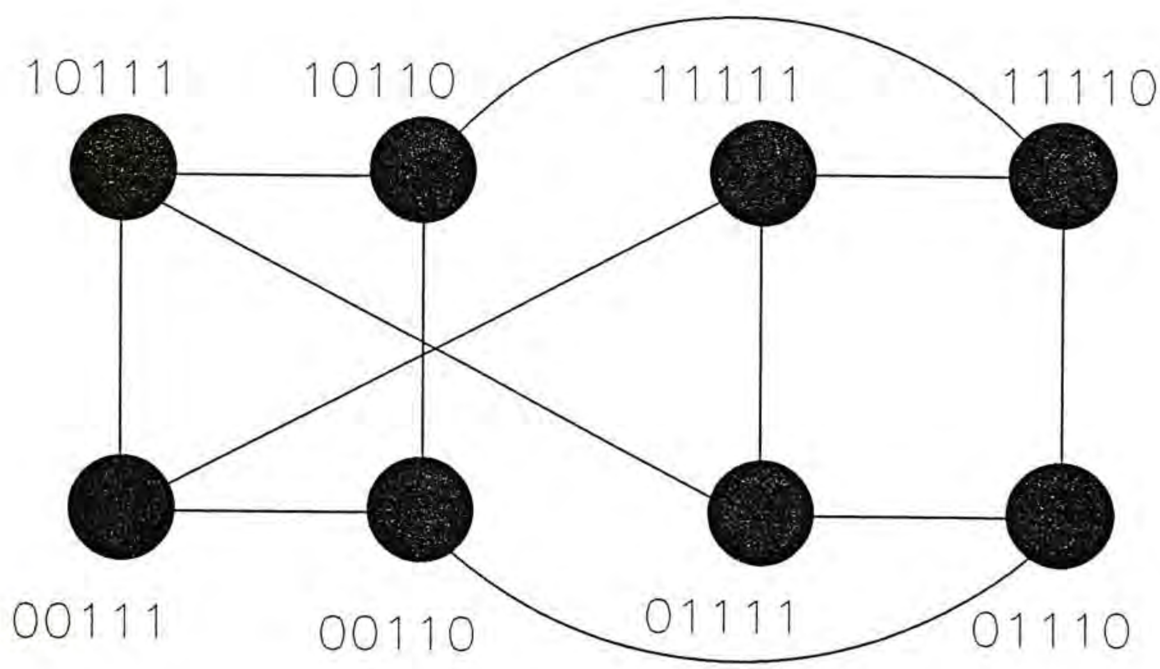


Figure 2.6: The induced graph $\sigma(AQ_5.\{XX11X\})$

be modified because the '11' bits in the string XX11X affects the parity calculation, and hence the connectivity, during the restoration of the AQ_3 graph. In particular a node v of AQ_3 should be mapped to node $v'_2 v_1 1 1 v_0$ of $\sigma(AQ_5, \{XX11X\})$ where $v'_2 = v_2 \oplus v_1$.

In general, the subgraph $\sigma(AQ_n, S)$ of AQ_n is an alternately-twisted k-cube, $1 \le k \le n$, if

(a) there exists a strictly increasing integer function f such that

(i) $f(0) = r$ for some integer r,

either $r = 0$ or (r is odd and $r \le n-1$)

(ii) $\forall 1 \le i \le \left\lfloor \frac{k-1}{2} \right\rfloor$,

$f(2i-2) < f(2i-1) = 2s-1 < f(2i) = 2s \le n-1$, for some integer s

(iii) if k is even, then

$$f(k-1) = \begin{cases} 2t \le n-1 & \text{if } n \text{ is odd} \\ 2t \le n-2 \text{ or } n-1 & \text{if } n \text{ is even} \end{cases}$$

for some integer t, and $f(k-2) < f(k-1)$

and,

(b) if $f(0) = 0$ then

$S = \{\rho_{n-1}\rho_{n-2}\cdots\rho_1\rho_0\}$ where

$\forall 0 \le j \le n-1$,

$$\rho_j = \begin{cases} X & \text{if } j = f(i) \text{ for some integer } i,\ 0 \le i \le k-1 \\ 0 \text{ or } 1 & \text{otherwise} \end{cases}$$

if $f(0) = r \ne 0$ then

$S=\{\rho_{n-1}\rho_{n-2}\cdots\rho_{r+2}\rho_{r+1}\rho_r\rho_{r-1}\cdots\rho_1\rho_0, \rho_{n-1}\rho_{n-2}\cdots\rho_{r+2}\hat{\rho}_{r+1}\hat{\rho}_r\rho_{r-1}\cdots\rho_1\rho_0\}$ where

$\forall 0\leq j\leq n-1, j\not\in\{r+1,r\}$

$$\rho_j=\begin{cases} X & \text{if } j=f(i) \text{ for some integer } i,\ 1\leq i\leq k-1 \\ 0 \text{ or } 1 & \text{otherwise} \end{cases}$$

in addition, $\rho_{r+1}\rho_r$ is either 00 or 10 and

$$\hat{\rho}_{r+1}\hat{\rho}_r=\begin{cases} \bar{\rho}_{r-1}\bar{\rho}_r & \text{if } \pi\ (\rho_{r-1}\cdot\cdot\rho_0)\ =\ 0 \\ \rho_{r+1}\bar{\rho}_r & \text{if } \pi\ (\rho_{r-1}\cdot\cdot\rho_0)\ =\ 1 \end{cases}$$

To see that the subgraph is a legitimate alternately-twisted cube, we just need to specify the mapping of the nodes in AQ_k to the nodes in $\sigma(AQ_n, S)$. Before doing it, we have to define a parity function for a ternary string, based on the ordinary parity function for a binary string:

$$\pi'(\rho_{n-1}\rho_{n-2}\cdots\rho_0)\equiv\pi(p_{n-1}p_{n-2}\cdots p_0)$$

where $p_i=0$ if $\rho_i=$ X or 0, otherwise $p_i=1$. That is, the "don't care bits" in $\rho_{n-1}\cdots\rho_0$ will not affect the parity of the string.

Now we can define the required mapping between nodes in AQ_k and $\sigma(AQ_n, S)$ as follows:

For any node u in AQ_k, the corresponding node in $\sigma(AQ_n, S)$ is node v where

(i) if f(0)=0, then $v_0=u_0$ else

$$v_{f(0)+1}v_{f(0)}=\begin{cases} \rho_{f(0)+1}\rho_{f(0)} & \text{if } u_0\ =\ 0 \\ \hat{\rho}_{f(0)+1}\hat{\rho}_{f(0)} & \text{if } u_0\ =\ 1 \end{cases}$$

(ii) for 1<2i≤k-1, if f(2i)=j then

$v_{j-1} = u_{2i-1}$

$$v_j = \begin{cases} u_{2i} & \text{if } \pi'(\rho_{j-2}\rho_{j-3}\cdots\rho_0) = 0 \\ u_{2i} \oplus u_{2i-1} & \text{if } \pi'(\rho_{j-2}\rho_{j-3}\cdots\rho_0) = 1 \end{cases}$$

(iii) $v_{f(k-1)} = u_{k-1}$ if k is even

(iv) $v_j = \rho_j$ for the remaining v_j's

The exclusive-or operator in rule (ii) is used to cancel the effect caused by the ρ_i's in calculating the parity function of the node addresses of the AQ_k.

As a result, an AQ_n can be divided into 2^{n-k} disjoint AQ_k subgraphs. Considering the restriction of the positions of placing the X's in the string $\rho_{n-1}\rho_{n-2}\cdots\rho_1\rho_0$, we have :

<u>Theorem 2.3</u> For any AQ_n, the number of ways of partitioning it into 2^{n-k} distinct sub-AQ_k is given by the following table:

	n-1 is even	n-1 is odd
k-1 is even	$\binom{\frac{n-1}{2}+1}{\frac{k-1}{2}+1}$	$\binom{\frac{n-2}{2}+1}{\frac{k-1}{2}+1}$
k-1 is odd	$\binom{\frac{n-1}{2}+1}{\frac{k}{2}+1}$	$\binom{\frac{n}{2}+1}{\frac{k}{2}+1}$

where $\binom{a}{b}$ denotes the binomial coefficient.

Proof: Actually this is the number of ways to assign the k X's to the appropriate positions in the string $\rho_{n-1}\rho_{n-2}\cdots\rho_1\rho_0$. According to the definition of function f, the X's are divided into $\lfloor \frac{k}{2} \rfloor + 1$ groups ($\lfloor \frac{k-1}{2} \rfloor$ of which consist of 2 X's, and the remaining $\lfloor \frac{k}{2} \rfloor + 1 - \lfloor \frac{k-1}{2} \rfloor$ group(s) consist of 1 X only), and the assignment of positions is done in terms of these groups. (Note that the rightmost group always consists of a single X only). And the ρ_i's are also divided into groups of two or one, in the same way as we did in the proof of Theorem 2.1. Thus there are $\lfloor \frac{n}{2} \rfloor + 1$ groups of ρ_i's, among which there is the group consisting of ρ_0 only. The results follow immediately from the combinatorics of matching the groups of X's into such groups of ρ_i's. (Q.E.D.)

It should be noted that the number specified in the theorem is generally smaller than the corresponding one in the binary hypercube case, which can be shown to be $\binom{n}{k}$. The reason is that in the latter, we do not have to group the X's in pairs during the matching, and there is no restriction on assigning a single X to a ρ_i , resulting in more freedom of choice.

2.2.4. Distinct Paths

Two paths between two nodes u and v of a graph are said to be node-disjoint if they do not share any intermediate node (ie. no node is common in both paths except the end-points). Likewise, they are edge-disjoint if no common edge exists in both paths. By distinct paths we mean that they are both node-disjoint and edge-disjoint. In the context of an interconnection network, the amounts of distinct paths between any two nodes in the underlying graph is important in two issues: under the realistic threat of hardware failure (be it due to the physical links or to the computation/communication elements at the nodes) the quantity of distinct paths directly affects the robustness and the fault tolerance of the network; even assuming no failure at the edges and the nodes, the availability of distinct paths provides alternative paths for message routing so as to avoid congestion points. This helps to balance the traffic flow under heavy load.

Since the alternately-twisted n-cube is a n-regular graph (each node has uniform degree of n), by Menger's theorem [Hara71, ch.5] there should be n distinct paths between any two nodes. This is the same for the hypercube case. The two networks weigh equally well in this aspect. For example, the 5 distinct paths between nodes 00000 and 11110 in the AQ_5 are:

00000 -> 10000 -> 10110 -> 11110

00000 -> 11000 -> 11110

00000 -> 00100 -> 11100 -> 11010 -> 11110

00000 -> 00110 -> 01110 -> 11110

00000 -> 00001 -> 00101 -> 00111 -> 11111 -> 11110

2.2.5. Embedding other networks

Parallel algorithms for different problems usually require different communication patterns among the computational elements. Common regular patterns include the rings, linear arrays, grids, binary trees, and hypercubes. The ability of an interconnection network to support these communication patterns is crucial to its suitability for a general purpose parallel processing environment. In this section we will show that the alternately-twisted cube can efficiently simulate all the above mentioned networks. The idea is to specify an one-to-one mapping (an embedding) of the nodes and edges in these networks to those in the alternately-twisted cube. If this can be done, we say that it is an embedding of dilation 1. If not, we try to simulate each edge in the graph to be embedded by using as small as possible a number of edges in the alternately-twisted cube. The dilation of the embedding will be the length of the longest of such simulated edges. We will see that the alternately-twisted cube is able to simulate the previously mentioned networks with a dilation of at most 2.

2.2.5.1. Embedding a ring into the alternately-twisted cube

It is easy to verify that each AQ_n contains a Hamiltonian cycle (ie. a closed path visiting every node of the graph exactly once) as its subgraph. A Hamiltonian cycle is exactly given by the T-code sequence, T_n, defined in Section 2.1. Recall that the elements of a T_n sequence is denoted by $T_n(i)$. Clearly T_1 specifies a Hamiltonian cycle of AQ_1. (To simplify the discussion, we consider a path joining two nodes as a Hamiltonian cycle of size 2.) Now suppose it is true that each T_i traces out a Hamiltonian cycle of AQ_i, for i=1,2,..,2k-1 for some k. In other words, nodes of address $T_i(j)$ and $T_i(j+1 \bmod 2^i)$ respectively are adjacent in AQ_i.

By definition, $T_{2k} = (0.T_{2k-1}), (1.T^R_{2k-1})$. Obviously in AQ_{2k}, node $0.T_{2k-1}(2^{2k-1}-1)$ is adjacent to node $1.T^R_{2k-1}(0)$. (N.B. $T^R_{2k-1}(0)$ refers to the 1st element of the sequence T^R_{2k-1}). Also there is an edge between node $0.T_{2k-1}(0)$ and node $1.T^R_{2k-1}(2^{2k-1}-1)$. Thus by induction T_{2k} also specifies a Hamiltonian cycle in AQ_{2k}.

Similarly, T_{2k+1} is defined as

$$T_{2k+1} = (0.T_{2k}),(1.T_{2k})$$
$$= (00.T_{2k-1}), (01.T^R_{2k-1}), (10.T_{2k-1}), (11.T^R_{2k-1})$$

Thus in an AQ_{2k+1}, node $T_{2k+1}(0)$ $(=00.T_{2k-1}(0))$ is adjacent to node $T_{2k+1}(2^{2k+1}-1)$ $(=11.T^R_{2k-1}(2^{2k-1}-1))$, and node $T_{2k+1}(2^{2k}-1)$ $(=01.T^R_{2k-1}(2^{2k-1}-1))$ is adjacent to node $T_{2k+1}(2^{2k})$ $(=10.T_{2k-1}(0))$, because $\pi(T_{2k-1}(0))=\pi(T^R_{2k-1}(2^{2k-1}-1))=0$. Hence AQ_{2k+1} has a Hamiltonian cycle whose nodes are specified by the sequence of T_{2k+1}.

Note that it is impossible to embed a ring of 3 nodes in an AQ_n with dilation 1, since if nodes u and v are adjacent, then the binary patterns of u and v will differ in either a single bit or in 2 consecutive bits at positions 2i and 2i-1, but in neither case can we find a third node that is adjacent to both nodes u and v.

For other rings of smaller sizes than 2^n, however, there does exist at least one embedding for each of them in an AQ_n. Here is the constructive proof. We will denote a ring of size i by R_i. First note that we can embed R_2, R_4, R_5, R_6, R_7, and R_8 respectively in an AQ_3, as shown in Figure 2.7. Note that we take R_2 as consisting of a single *path* between 2 nodes. Since an AQ_{k-1} is normally a subgraph of AQ_k, we only need to consider the embedding of rings R_i, for $2^{k-1} < i \leq 2^k$, in an AQ_k, provided that we already know how to embed rings of smaller sizes in an AQ_{k-1}.

Assume it is true that R_i, for i=2 or $4 \leq i \leq 2^k$, can be embedded in AQ_k with dilation one, for k=2, 3, 4, .., 2n-1.

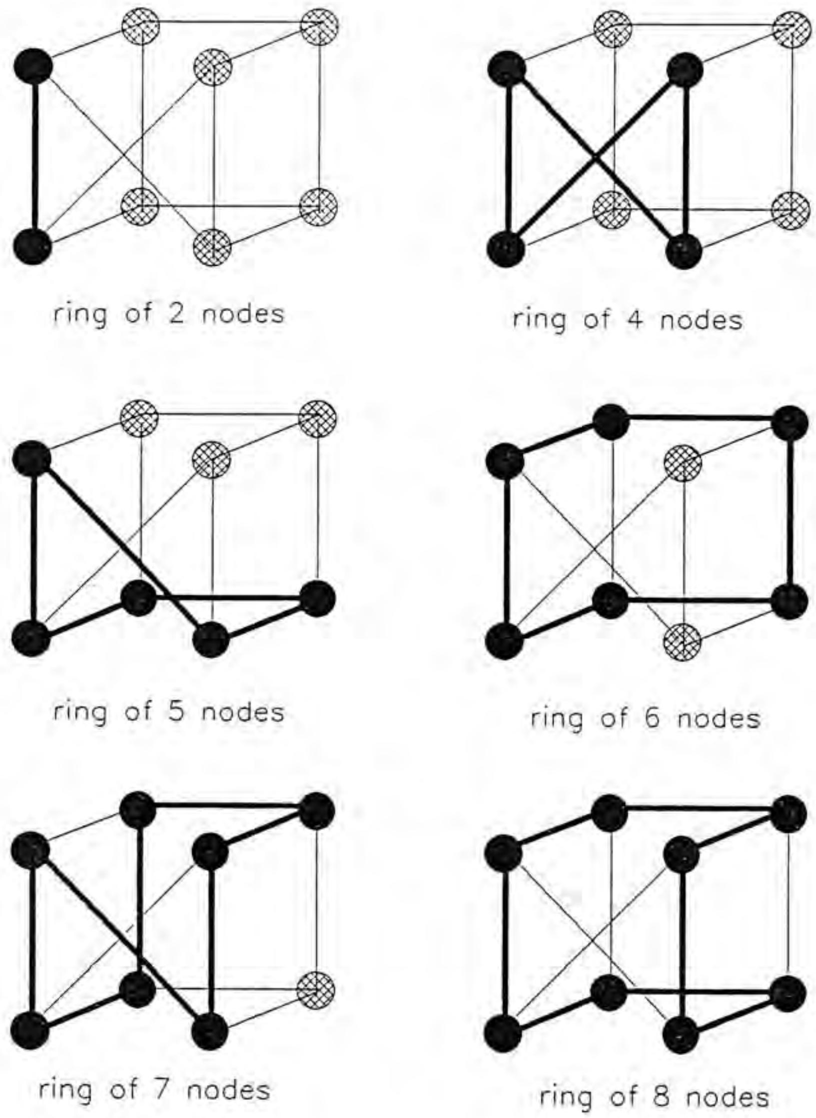


Figure 2.7: Embeddings of rings into an AQ_3

(i) To embed a ring R_i, $2^{2n-1} < i \leq 2^{2n}$, in an AQ_{2n}, we choose two integers x≠3 and y≠3 such that i=x+y. Let X^i denote the string which is X repeated i times. Then by the assumption we can always find 2 rings R' and R" where R' is the R_x embedded in $\sigma(AQ_{2n}, \{0X^{2n-1}\})$ and R" is the R_y embedded in $\sigma(AQ_{2n}, \{1X^{2n-1}\})$, such that there exists 2 nodes u and v, and the edges

(0u,0v) $\in$ R' and (1u,1v) $\in$ R". Then the required R_i can be obtained by the operation

$$R_i = R' + R'' - \{ (0u,0v), (1u,1v) \} + \{ (0u,1u), (1v,0v) \}$$

(ii) Similarly, to embed a ring R_i, $2^{2n} < i \leq 2^{2n+1}$, in an AQ_{2n+1}, we find two integers x and y such that i=x+y and x≠3 and y≠3. We can always choose 2 rings R' and R" such that R' is the R_x embedded in $\sigma(AQ_{2n+1}, \{00X^{2n-1}, 10X^{2n-1}\}$ and R" is the R_y embedded in $\sigma(AQ_{2n+1}, \{01X^{2n-1}, 11X^{2n-1}\}$, and there exists a node u such that the edges (00u,10u) $\in$ R' and (01u,11u) $\in$ R". (Note that graphs $\sigma(AQ_{2n+1}, \{00X^{2n-1}, 10X^{2n-1}\}$ and $\sigma(AQ_{2n+1}, \{01X^{2n-1}, 11X^{2n-1}\}$ are respectively the alternately-twisted 2n-cube formed from 2 AQ_{2n-1}'s) Then the required R_i can be obtained by

$$R_i = R' + R'' - \{ (00u,10u), (01u,11u) \} + E$$

where E = { (00u,11u), (01u, 10u) } if $\pi(u)=0$,

or { (00u,01u), (11u, 10u) } if $\pi(u)=1$

Clearly, the above embedding is of dilation 1. In other words, we have

<u>Theorem 2.4</u> It is always possible to find a dilation-1 embedding of a ring of i nodes, R_i, i=2 or $4 \leq i \leq 2^n$, into an alternately-twisted n-cube AQ_n.

As an example, an embedding of R_{13} into AQ_4 is shown in Figure 2.8. Similar results hold for mapping a linear array of size i, $1 \leq i \leq 2^n$, into an AQ_n, since it is just a ring with one of its edges removed.

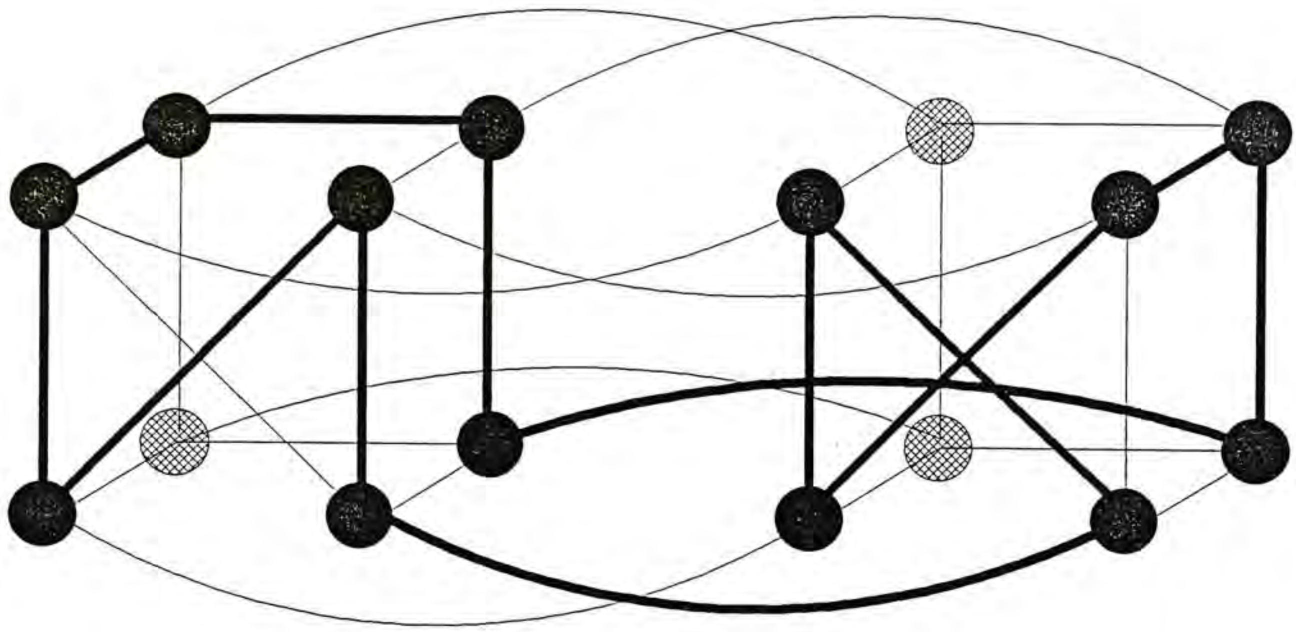


Figure 2.8: R_{13} embedded in an AQ_4

It is known that a binary n-cube can only embed rings of even length i, $2 \le i \le 2^n$, [SaSc88] if we insist on dilation-1 embedding. Therefore the alternately-twisted cube provides more flexibility in this aspect.

2.2.5.2. Grid Embeddings on the Alternately-Twisted Cube

It is possible for an alternately-twisted n-cube to embed, with dilation 1, a $2^p \times 2^q$ grid where $p = \lfloor \frac{n}{2} \rfloor$ and $q = n - \lfloor \frac{n}{2} \rfloor = \lceil \frac{n}{2} \rceil$. The idea is to partition a node address in AQ_n into two parts, in the form of:

$$\begin{cases} x_m y_m x_{m-1} y_{m-1} \ldots x_1 y_1 y_0 & \text{if } n \text{ is odd} \\ x_{m+1} x_m y_m x_{m-1} y_{m-1} \ldots x_1 y_1 y_0 & \text{if } n \text{ is even} \end{cases}$$

where $m = \left\lfloor \frac{n-1}{2} \right\rfloor$ and the x_i's and y_i's are bits. Since the x_i's are at even-numbered positions (except x_{m+1}) they (including x_{m+1}, however) can be complemented independently through traversal over the edge along the corresponding dimension of the AQ_n. Hence when all the y_i's are fixed, the resulting 2^m or 2^{m+1} (depending on the value of n) nodes form a hypercube in its own. Clearly there is a Hamiltonian path in this hypercube traced out by the reflected Gray code. On the other hand, when all the x_i's are fixed, the resulting 2^{m+1} nodes form a subgraph in which is embedded a linear array of size 2^{m+1}nodes. To see this, let us start with all y_i's being set to zeroes, i.e. with the node $(x_{m+1})x_m0x_{m-1}0...x_20x_100$ (N.B. we bracket x_{m+1} in the string to remind that its presence depends on the value of n). Clearly it is adjacent to node $(x_{m+1})x_m0x_{m-1}0...x_20x_101$, no matter what the x_i's are. Again this node has direct linkage to node $(x_{m+1})x_m0x_{m-1}0...x_20x_111$, which is in turn neighbour of node $(x_{m+1})x_m0x_{m-1}0...x_20x_110$. And, the succeeding nodes along the desired linear array, or chain, are

$(x_{m+1})x_m0x_{m-1}0...x_21x_110$, then

$(x_{m+1})x_m0x_{m-1}0...x_21x_111$, then

$(x_{m+1})x_m0x_{m-1}0...x_21x_101$, and then

$(x_{m+1})x_m0x_{m-1}0...x_21x_100$, and then

$(x_{m+1})x_m0x_{m-1}0...x_31x_21x_100$,

... , and finally reaches

$(x_{m+1})x_m1x_{m-1}0x_{m-2}0...x_20x_100$.

It is now clear that the y_i's of nodes along this chain actually form the elements of the reflected Gray code sequence of size 2^{m+1}. Thus formally we have:

<u>Lemma 2.5</u> For any fixed set of x_i's, $0 \le i \le m$ (or m+1, depending on n as discussed above), the induced subgraph of AQ_n, whose nodes have addresses of the form $(x_{m+1})x_m y_m x_{m-1} y_{m-1} \ldots x_2 y_2 x_1 y_1 y_0$, contains a Hamiltonian path which starts at node $(x_{m+1})x_m 0 x_{m-1} 0 \ldots x_2 0 x_1 00$ and ends at node $(x_{m+1})x_m 1 x_{m-1} 0 \ldots x_2 0 x_1 00$. The i-th node along the path, $0 \le i \le 2^{m+1} - 1$, is the node $(x_{m+1})x_m y_m x_{m-1} y_{m-1} \ldots x_2 y_2 x_1 y_1 y_0$ where $y_m y_{m-1} \ldots y_1 y_0$ corresponds to the i-th element of the reflected Gray code of size 2^{m+1}, G_{m+1}.

Now let us come back to the question of embedding the $2^p \times 2^q$ grid in an AQ_n, where $p = \lfloor \frac{n}{2} \rfloor$ and $q = \lceil \frac{n}{2} \rceil$. This can be done by forming the columns with the Hamiltonian paths contained in the hypercubes induced from the AQ_n by fixing the y_i's in the node address $(x_{m+1})x_m y_m x_{m-1} y_{m-1} \ldots x_2 y_2 x_1 y_1 y_0$. Each fixed set of y_i's will give a hypercube whose Hamiltonian path corresponds to an individual column. Likewise, each row of the grid is formed by the Hamiltonian path as specified in Lemma 2.5, where each fixed set of x_i's corresponds to the position of the row. Clearly this is a dilation-1 embedding.

If p and q are not the specific pair $\lfloor \frac{n}{2} \rfloor$ and $\lceil \frac{n}{2} \rceil$, we can still embed with dilation 1 the rectangular $2^p \times 2^q$ grid in an AQ_n, provided that $n = p + q$. Without loss of generality, we assume $p < q$. The method is similar to the previous one, except that some of the rightmost x_i's will be used in defining the column address rather than the row address of the grid. Specifically, the address of the AQ_n will be regarded as having the form

$$x_m y_m x_{m-1} y_{m-1} \cdots x_k y_k z_{2k-2} z_{2k-3} z_{2k-4} \cdots z_0 \quad \text{if } n \text{ is odd}$$

or $$x_{m+1} x_m y_m x_{m-1} y_{m-1} \cdots x_{k+1} y_{k+1} z_{2k} z_{2k-1} z_{2k-2} \cdots z_0 \quad \text{if } n \text{ is even}$$

where $m = \left\lfloor \frac{n-1}{2} \right\rfloor$ and k=m-p+1. Again, fixing the y_i's and z_i's will induce a binary p-cube whose Hamiltonian path is used to form a column of the grid. Now suppose all x_i's are fixed. For the subgraph of AQ_n induced by these fixed x_i's, there is still a linear array, or chain, visiting every node of it. Its construction is given in the following. Since the idea is the same for n is even or odd, except that the subscripts of z differ, the succeeding discussion assumes that n is odd. First observe that for every combination of $y_m y_{m-1} \cdots y_k$, the rightmost 2k-1 bits $z_{2k-2} z_{2k-1} \cdots z_0$ specifies an alternately-twisted (2k-1)-cube, within which there is a Hamiltonian path traced out by the T_{2k-1} sequence, starting with $z_{2k-2} z_{2k-1} \cdots z_0 = 0^{2k-1}$ and terminating with $z_{2k-2} z_{2k-1} \cdots z_0 = 110^{2k-3}$, by the result of Section 2.2.5.1. There will be totally 2^{m-k+1} such chains (for each fixed set of x_i's), each of size 2^{2k-1} and corresponding to a distinct set of y_i's. They are then linked together to form the desired chain of size 2^q. The chain is given by the sequence

$$x_m 0 x_{m-1} 0 \ldots x_{k+2} 0 x_{k+1} 0 x_k 0 . T_{2k-1} ; x_m 0 x_{m-1} 0 \ldots x_{k+2} 0 x_{k+1} 0 x_k 1 . T^R_{2k-1} ;$$

$$x_m 0 x_{m-1} 0 \ldots x_{k+2} 0 x_{k+1} 1 x_k 1 . T_{2k-1} ; x_m 0 x_{m-1} 0 \ldots x_{k+2} 0 x_{k+1} 1 x_k 0 . T^R_{2k-1} ;$$

$$x_m 0 x_{m-1} 0 \ldots x_{k+2} 1 x_{k+1} 1 x_k 0 . T_{2k-1} ; x_m 0 x_{m-1} 0 \ldots x_{k+2} 1 x_{k+1} 1 x_k 1 . T^R_{2k-1} ;$$

$$x_m 0 x_{m-1} 0 \ldots x_{k+2} 1 x_{k+1} 0 x_k 1 . T_{2k-1} ; x_m 0 x_{m-1} 0 \ldots x_{k+2} 1 x_{k+1} 0 x_k 0 . T^R_{2k-1} ;$$

....

$$x_m 1 x_{m-1} 0 \ldots x_{k+2} 0 x_{k+1} 0 x_k 1 . T_{2k-1} ; x_m 1 x_{m-1} 0 \ldots x_{k+2} 0 x_{k+1} 0 x_k 0 . T^R_{2k-1}$$

That is, the chain is a sequence of subchains, starting with the subchain $x_m 0 x_{m-1} 0 \ldots x_{k+2} 0 x_{k+1} 0 x_k 0 . T_{2k-1}$, such that the (2i)-th and (2i+1)-st subchains are given by $x_m y_m x_{m-1} y_{m-1} \ldots x_k y_k . T_{2k-1}$ and $x_m y_m x_{m-1} y_{m-1} \ldots x_k y_k . T^R_{2k-1}$ respectively where $y_m y_{m-1} \cdots y_{k+1} y_k$ is the i-th element of a reflected Gray code of size m-k+1. The linkages between the respective subchains are valid edges in the AQ_n because the parities of the last elements of T_{2k-1} and of T^R_{2k-1} are, respectively, $\pi(T_{2k-1}(2^{k-1}-1)) = 1$ and $\pi(T^R_{2k-1}(2^{k-1}-1)) = 0$.

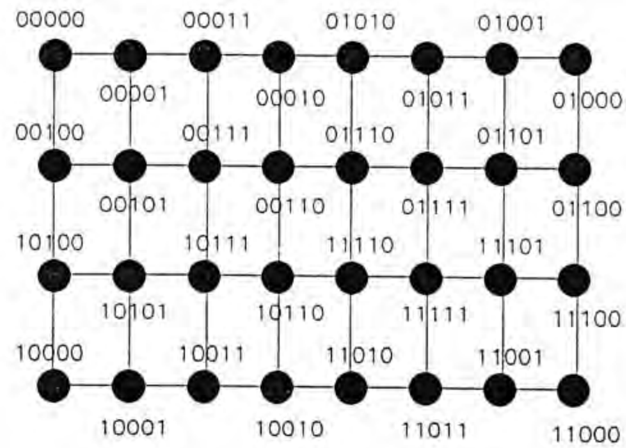
Therefore we can embed any $2^p \times 2^q$ grid into the AQ_n by mapping the rows and columns to the corresponding chains, the formation of which is just discussed. (That is, individual column is formed by fixing the y_i's and z_i's, and individual row is formed by fixing the x_i's). Thus we arrive at the general result:

Theorem 2.6 For any integers p and q, the $2^p \times 2^q$ grid can be embedded into the alternately-twisted (p+q)-cube with dilation one.

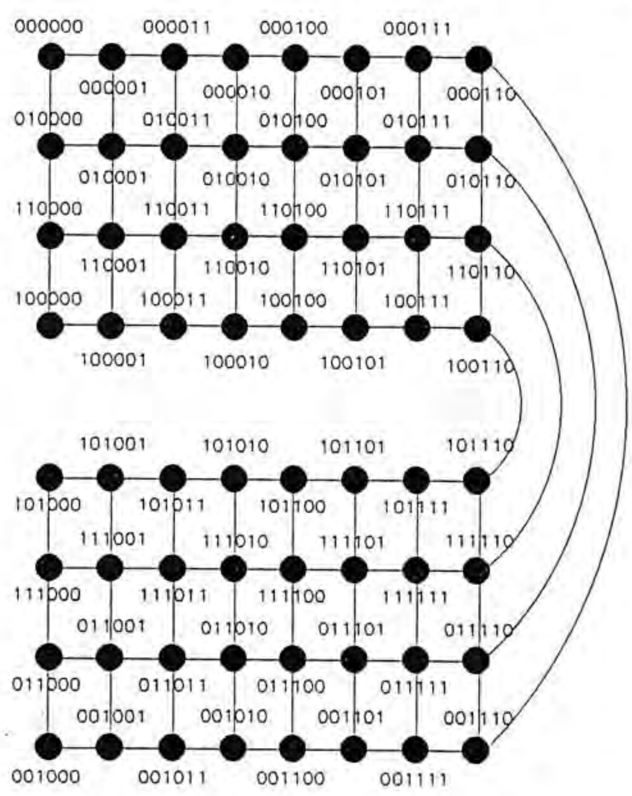
For example, the embeddings of the 4x8 and 4x16 grids into the AQ_5 and AQ_6 respectively are illustrated in Figure 2.9. Only relevant edges are shown in the figure.

Chan [Chan88] has proven that any HxW grids can be embedded into a binary n-cube, where HxW≤ 2^n, with dilation 2. The major basis of her proof is that the grid is first embedded into a $2^p \times 2^q$ grid with dilation 2, p+q=n, which is then embedded with dilation 1 into a hypercube of size 2^n. Since the alternately-twisted cube can also do the latter step, her mapping algorithm can be applied here as well:

Theorem 2.7 An alternately-twisted n-cube is able to embed any HxW grids, where HxW≤ 2^n, with dilation of at most 2.



(a) Embedding of 4x8 grid into an AQ_5



(b) Embedding of 4x16 grid into an AQ_6

Figure 2.9: Two grid embeddings in alternately-twisted cube

(only relevant edges of the cubes are shown)

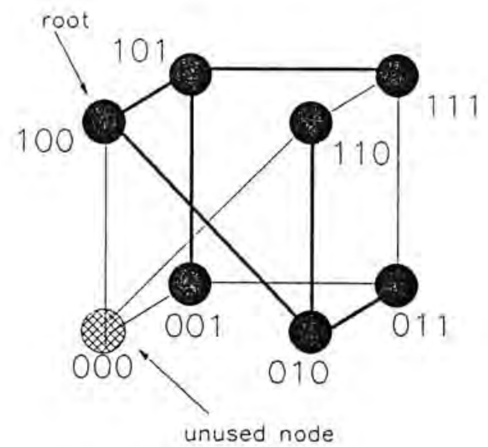
We refer the interested reader to the paper by Chan [Chan88] for a detailed account of the mapping.

2.2.5.3. Simulation of binary trees

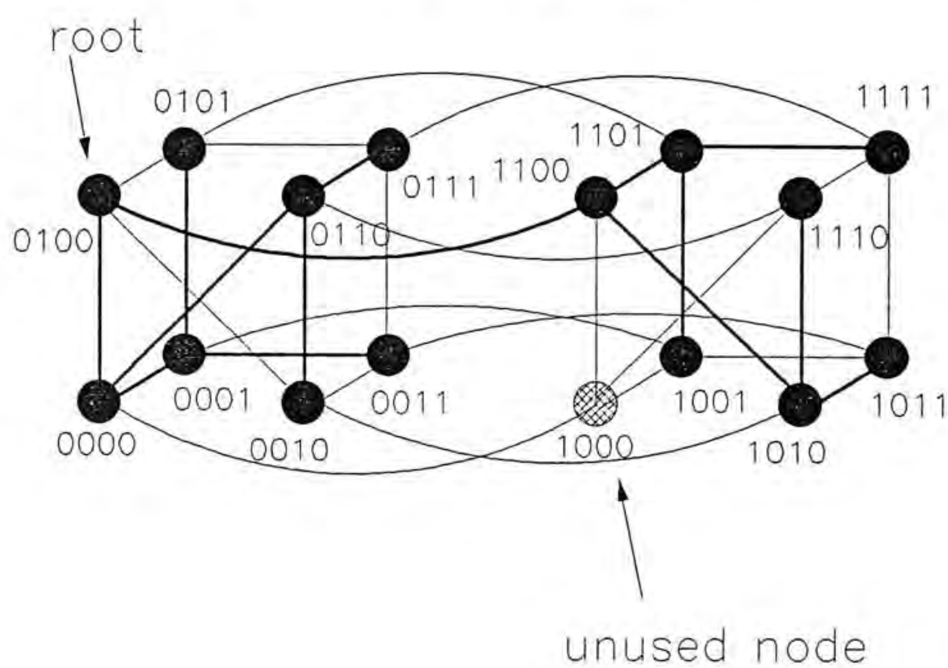
Let BT_n denote the complete binary tree of size 2^n-1. It is possible to specify a dilation-1 embedding of BT_n into the graph of AQ_n, for n=1, 2, 3, and 4. The embeddings of BT_1 and BT_2 are trivial, and those for BT_3 and BT_4 are given in Figure 2.10. However, for larger binary trees, we are only able to embed a BT_{n-1} with dilation one into an AQ_n, ie. into an alternately-twisted cube of double size of the tree.

Theorem 2.8 For any node u in AQ_n, n≥1, there is an embedded BT_{n-1} rooted at u.

Proof: Since AQ_n is node symmetric, we only need to show that there is an embedding of a BT_{n-1} into an AQ_n. The desired binary tree rooted at any specific node u can then be obtained by an appropriate automorphism. It is obvious that the theorem is true for n≤4, because a BT_{n-1} is a subtree of BT_n. Now suppose it is true for all n≤2k, and we are going to show that it must then be true for n=2k+1 and n=2k+2. Observe that since the BT_{2k-1} occupies only about half the nodes of AQ_{2k}, the root of the tree, say r = $r_{2k-1}..r_0$, must be adjacent to a node s = $s_{2k-1}..s_0$ which is not included in the tree. For an AQ_{2k+1}, there is 2 distinct AQ_{2k}, which are, using the notation in Section 2.2.3, $\sigma(AQ_{2k+1}, \{00X^{2k-1}, 10X^{2k-1}\})$ and $\sigma(AQ_{2k+1}, \{01X^{2k-1}, 11X^{2k-1}\})$ respectively. By assumption, there is an embedding of BT_{2k-1} rooted at node $r' = r_{2k-1}0r_{2k-2}r_{2k-3}\cdots r_1r_0$ in the sub-



(a) embedding of BT_3 into an AQ_3



(b) embedding of BT_4 into an AQ_4

Figure 2.10: Dilation-1 embedding of BT_3 and BT_4 into AQ_3 and AQ_4 respectively

graph $\sigma(AQ_{2k+1}, \{00X^{2k-1}, 10X^{2k-1}\})$, and there is another embedding of the BT_{2k-1} rooted at node

$$s' = \begin{cases} s_{2k-1} 1 s_{2k-2} s_{2k-3} \dots s_1 s_0 & \text{if } \pi(s_{2k-2} \dots s_0) = 1 \\ \bar{s}_{2k-1} 1 s_{2k-2} s_{2k-3} \dots s_1 s_0 & \text{if } \pi(s_{2k-2} \dots s_0) = 0 \end{cases}$$

in the subgraph $\sigma(AQ_{2k+1}, \{01X^{2k-1}, 11X^{2k-1}\})$. The desired embedding of the binary tree BT_{2k} into the AQ_{2k+1} is obtained by combining these two (sub)trees through the new root at node $t = s_{2k-1} 0 s_{2k-2} s_{2k-3} .. s_0$ respectively via the edges (r',t) and (s', t). Similarly, for a binary tree BT_{2k} embedded within the AQ_{2k+1}, rooted at node t, there must be a node v adjacent to t and not being a node of the tree. Then for an AQ_{2k+2}, the desired new tree BT_{2k+1} can be formed by including the tree BT_{2k} within the (alternately-twisted) subcube $\sigma(AQ_{2k+2}, \{0X^{2k+1}\})$, rooted at node 0t, and the tree BT_{2k} embedded in the subcube $\sigma(AQ_{2k+2}, \{1X^{2k+1}\})$, rooted at node 1v, and combining them through the new root at node 0v using the edges (0t, 0v) and (1v, 0v) respectively. (Figure 2.11 illustrates the idea for the case of n=2k+2.) Hence by induction the theorem holds for all n. (Q.E.D.)

The embedding of BT_{n-1} into an AQ_n is quite "inefficient", since about 50% of the nodes of AQ_n do not take part in the embedding. It is therefore natural to search for the feasibility of embedding the binary tree BT_n into an AQ_n with dilation one. We know of no such solution, however, and will leave it as an open question. Instead, we will describe a dilation-2 embedding. To this end we define the over-loading factor of an edge e in a graph H for the embedding of a graph G into H as the number of edges of G which are mapped to the paths/edges (in H) that include

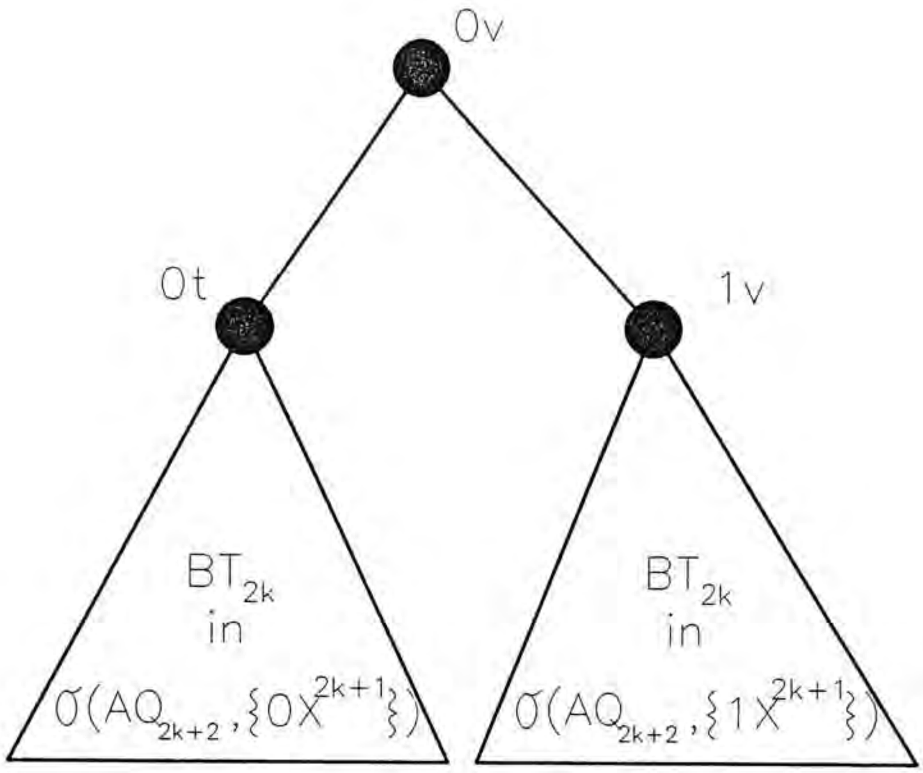


Figure 2.11: Illustrating the (recursive) embedding of a BT_{2k+1} in an AQ_{2k+2}

e. The largest overloading factor among all the edges of H is the overloading factor of the embedding. It is used to measure the edge congestion in the graph H for the embedding. Clearly the smaller such a number, the better the quality of the embedding because large overloading factor tends to cause more unbalanced traffic over the edges. We have the following result for the AQ_n:

Theorem 2.9 For any AQ_n, there exists an embedding of BT_n, with dilation of at most 2 and overloading factor of at most 2.

Proof: Again, the theorem holds for n=1,2,3 and 4 trivially. Suppose that it is also true for n≤2k, and for each such n, if n is even and ≥4, the embedding is such that the root of BT_n is at node $r = r_{n-1}r_{n-2}\ldots r_0$, the unused node (ie. the node of AQ_n that is not mapped to any node of the BT_n) is node $u = \bar{r}_{n-1}\bar{r}_{n-2}r_{n-3}r_{n-4}r_{n-5}\ldots r_0$, that $\pi(r_{n-2}r_{n-3}\ldots r_0)=0$, and that the two edges r -> t and r -> s, where $t = \bar{r}_{n-1}r_{n-2}r_{n-3}\ldots r_0$ and $s = r_{n-1}\bar{r}_{n-2}r_{n-3}r_{n-4}\ldots r_0$, are used in the embedding of the tree and each has an overloading factor of at most 1.

Now for the graph AQ_{2k+1}, there is a dilation-2 embedding of BT_{2k} into each of the 2 (alternately-twisted) subcubes $\sigma(AQ_{2k+1}, \{00X^{2k-1}, 10X^{2k-1}\})$ and $\sigma(AQ_{2k+1}, \{01X^{2k-1}, 11X^{2k-1}\})$, with the nodes r, s, t, and u described in the above assumption mapped to

r' $= r_{2k-1}0r_{2k-2}\ldots r_0$,

s' $= r_{2k-1}0\bar{r}_{2k-2}\ldots r_0$,

t' $= \bar{r}_{2k-1}0r_{2k-2}\ldots r_0$, and

u' $= \bar{r}_{2k-1}0\bar{r}_{2k-2}\ldots r_0$

respectively in the first subcube, and to

r" $= r_{2k-1}1r_{2k-2}\ldots r_0$,

s" $= r_{2k-1}1\bar{r}_{2k-2}\ldots r_0$,

t" $= \bar{r}_{2k-1}1r_{2k-2}\ldots r_0$, and

u" $= \bar{r}_{2k-1}1\bar{r}_{2k-2}\ldots r_0$

respectively in the second subcube. The desired embedding of BT_{2k+1} into the AQ_{2k+1} is formed by joining the two (sub)trees BT_{2k} via the new root u", which is connected to the two (old) roots of the BT_{2k} through the paths r' -> t" -> u" and r" -> t"-> u" respectively. Note that the edges t" -> u" and r' -> t" are newly added edges, so their overloading factors are respectively 2 and 1. Also the edge r" -> t" will have an overloading factor of at most 2. Besides, the two paths between the new root u" and the unused node u' are respectively

u" -> s' -> u' and u" -> s" -> u',

and the overloading factors of these edges are 0, because they are not used in the embedding. Also note that the new root u" has the form $\bar{r}_{2k-1} 1 \bar{r}_{2k-2} \ldots r_0$ and π(u") = 1. Hence we prove the theorem for n = 2k + 1.

For the graph of AQ_{2k+2}, there will be two BT_{2k+1} embedded, one in each of the subcubes $\sigma(AQ_{2k+2}, \{0X^{2k+1}\})$ and $\sigma(AQ_{2k+2}, \{1X^{2k+1}\})$. According to the mapping method for n = 2k + 1, the root of the embedded tree in the first subcube will be 0u", the unused node will be 0u', and the paths

0u" -> 0s' -> 0u' and 0u" -> 0s" -> 0u'

contain edges of overloading factor of zero in the embedding. For the second subcube, since it is node-symmetric, we can specify the embedded BT_{2k+1} as rooted at node 1s", the unused node being 1s' and for the 4 edges of the two paths

1s" -> 1u' -> 1s' and 1s" -> 1u" -> 1s',

each has an overloading factor of 0 only. Then the desired embedding of BT_{2k+2} into the AQ_{2k+2} is obtained by taking the 2 embedded BT_{2k+1}'s as two subtrees which are joined to the new root at node 0u' via the respective two paths:

0u" -> 0s" -> 0u' and 1s" -> 0s" -> 0u'.

These edges are newly added, so the overloading factors are 1 for the edges 0u" -> 0s" and 1s" -> 0s", and 2 for the edge 0s" -> 0u'. The unused node is 1s', to which there exist two paths from the new root 0u', namely, 0u' -> 0s' -> 1s' and 0u' -> 1u' -> 1s'. Each of these 4 edges has an overloading factor of zero in the embedding, because they are not used at all. Note that the involved 4 nodes have the address forms that satisfy those specified in the induction assumption stated in the beginning:

$$0u' = 0\bar{r}_{2k-1}0\bar{r}_{2k-2}r_{2k-3}r_{2k-4}\cdots r_0$$

$$0s' = 0r_{2k-1}0\bar{r}_{2k-2}r_{2k-3}r_{2k-4}\cdots r_0$$

$$1u' = 1\bar{r}_{2k-1}0\bar{r}_{2k-2}r_{2k-3}r_{2k-4}\cdots r_0$$

$$1s' = 1r_{2k-1}0\bar{r}_{2k-2}r_{2k-3}r_{2k-4}\cdots r_0$$

$$\text{and} \quad \pi(\bar{r}_{2k-1}0\bar{r}_{2k-2}r_{2k-3}r_{2k-4}\cdots r_0) = 0$$

for $n=2k+2$. Therefore by induction the theorem holds for all n. (Q.E.D.)

As an example, we show in Figure 2.12 the dilation-2, overloading-factor-2 embedding of BT_5 into an AQ_5. The bold solid lines represent the embedded tree edges that are dilated by 1, and the bold dotted lines represent those edges being dilated by a factor of 2.

It is known that for a binary n-cube, $n>2$, there is no dilation-1 embedding of a BT_n into it, and the largest complete binary tree it can embed with such goal is the BT_{n-1} [LaDh90, p.82-87]. Hence in terms of dilation-1 embedding of binary trees, the alternately-twisted cube is slightly better than the hypercube, as BT_3 and BT_4 can be embedded into AQ_3 and AQ_4 respectively with dilation one. However, if the requirement is relieved to dilation-2 embedding, the binary n-cube is better. It can house a BT_n with only one of the tree edges being actually "dilated", namely, the edge connecting the root and one of its subtrees, and

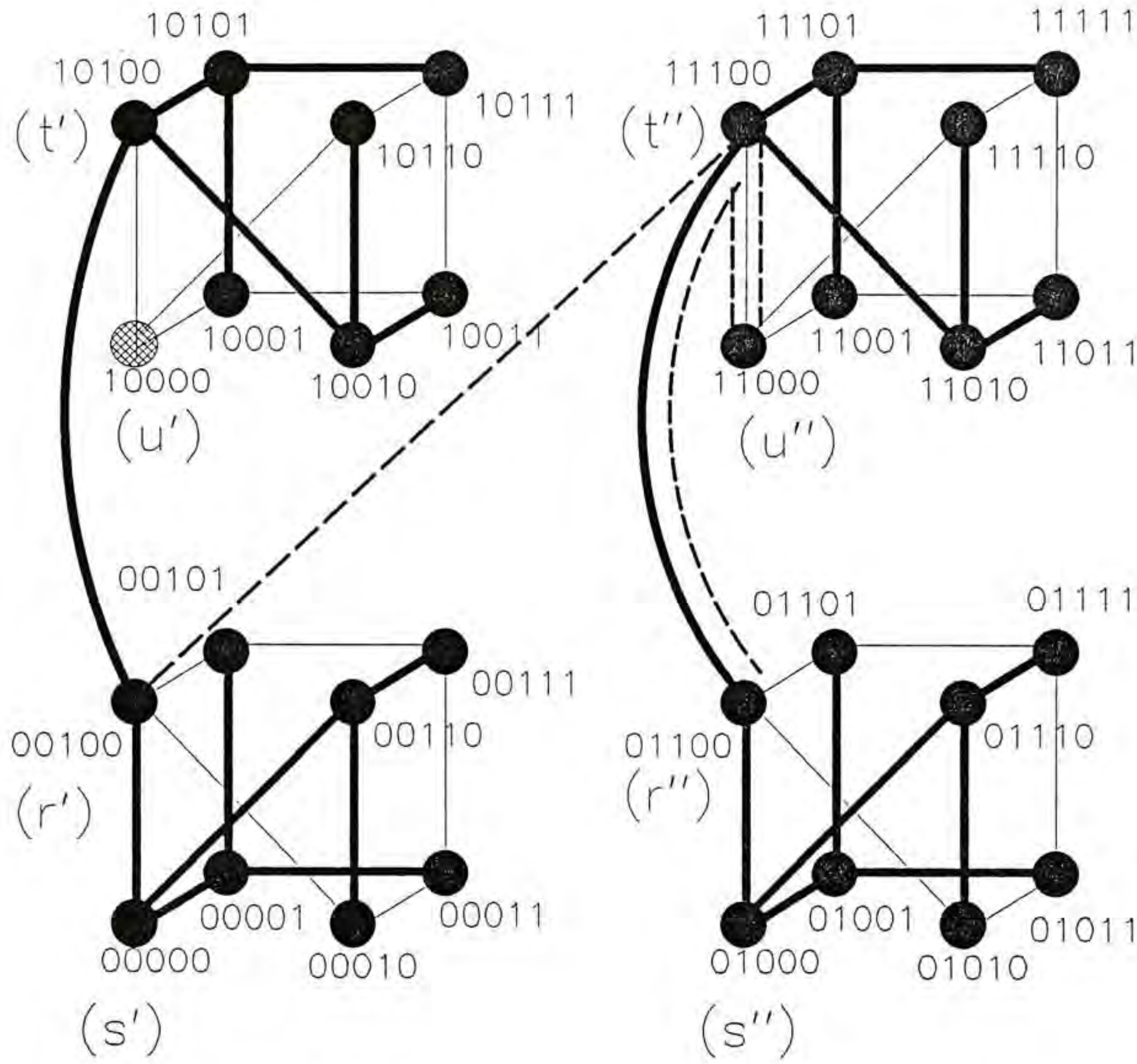


Figure 2.12: Dilation-2 embedding of BT_5 into AQ_5

(N.B.: irrelevant edges along the 3rd- and 4th-dimensions are omitted for clarity)

the embedding has an overloading factor of only 1 [LaDh90, p.91]. Thus it is more suitable, in general, to simulate a BT_n on the binary n-cube than on the alternately-twisted n-cube.

2.2.5.4. Simulating the hypercube

Since binary 1- and 2-cubes are the same graphs as AQ_1 and AQ_2 respectively, we will assume in this subsection that the hypercube being discussed has at least 3 dimensions. From the discussion in Section 2.2.5.2, it can be seen that an alternately-twisted n-cube contains 2^{n-k} binary k-cubes as its subgraphs, where k=(n+1)/2 if n is odd, and k=(n/2)+1 if n is even. For larger-sized hypercubes, however, the alternately-twisted n-cube can only simulate it. In general we have the following result

<u>Theorem 2.10</u> An AQ_n, n>2, can embed a binary n-cube with dilation 2, overloading factor of 2.

Proof: The nodes of the binary n-cube is mapped in an 1-to-1 fashion to the corresponding nodes with the same addresses in the AQ_n. Since the edges along the even-numbered dimensions are not twisted at all, they are mapped directly with the corresponding hypercube edges. For each hypercube edge of the form

$x_{n-1}x_{n-2}..x_{2i}x_{2i-1}x_{2i-2}..x_0 \dashrightarrow x_{n-1}x_{n-2}..x_{2i}\overline{x}_{2i-1}x_{2i-2}..x_0$

it is mapped to the same edge in AQ_n if $\pi(x_{2i-2}...x_0)=1$, and to the paths

$x_{n-1}x_{n-2}..x_{2i+1}x_{2i}x_{2i-1}x_{2i-2}..x_0 \dashrightarrow$

$x_{n-1}x_{n-2}..x_{2i+1}\overline{x}_{2i}\overline{x}_{2i-1}x_{2i-2}..x_0 \dashrightarrow$

$x_{n-1}x_{n-2}..x_{2i+1}x_{2i}\overline{x}_{2i-1}x_{2i-2}..x_0$

if $\pi(x_{2i-2}..x_0)=0$. In this way each edge of the form

$x_{n-1}x_{n-2}..x_{2i+1}x_{2i}x_{2i-1}..x_0 \dashrightarrow x_{n-1}x_{n-2}..x_{2i+1}\overline{x}_{2i}x_{2i-1}..x_0$

will be used twice in the embedding, while all other edges are used once. (Q.E.D.)

It should be noted that the theorem can also be stated in the reverse direction, i.e. a binary n-cube is able to embed an AQ_n with dilation 2, overloading factor of 2.

2.2.6. Summary of Comparison with the hypercube

The foregoing discussion of the topological properties of the alternately-twisted cube is summarized below, along with the corresponding properties of the binary n-cube. In general the former preserves much of the salient features of the hypercube, while at the same time it is superior to the hypercube in the measure of worst-case distance among the nodes of the whole graph. This will be further investigated in the next chapter, when the alternately-twisted cube is analysed in the context of a message-passing multiprocessor network.

	alternately-twisted cube	hypercube
size	2^n	2^n
node degree	n	n
link count	$n2^{n-1}$	$n2^{n-1}$
diameter	$\lfloor \frac{n}{2} \rfloor + 1$	n
symmetry	node-symmetric	both edge- and node-symmetric

	alternately-twisted cube	hypercube
subcube	into 2^{n-k} AQ_k:	into 2^{n-k} k-cubes:
partitioning	n is odd and k is odd: $\binom{\frac{n+1}{2}}{\frac{k+1}{2}}$ ways n is odd and k is even: $\binom{\frac{n+1}{2}}{\frac{k+2}{2}}$ ways n is even and k is odd: $\binom{\frac{n}{2}}{\frac{k+1}{2}}$ ways n is even and k is even: $\binom{\frac{n+2}{2}}{\frac{k+2}{2}}$ ways	$\binom{n}{k}$ ways

	alternately-twisted cube	hypercube
# of distinct paths between 2 nodes	n	n
embedding of rings, R_i	dilation 1 for all $i \leq 2^n$ and $i \neq 3$	dilation 1 for all even $i \leq 2^n$
embedding of HxW grids ($HxW \leq 2^n$)	dilation 1 if H and W are powers of 2, dilation 2 otherwise	dilation 1 if H and W are powers of 2, dilation 2 otherwise
embedding of complete binary trees, BT_k	dilation 1 for $k < n$, dilation 2 with overloading factor of 2 for $k = n$	dilation 1 for $k < n$, dilation 2 with overloading factor of 1 for $k = n$
(simulation of each other)	dilation 2 with overloading factor 2	dilation 2 with overloading factor 2

Chapter 3

Network Properties

In this chapter the properties of the alternately-twisted cube as an interconnection network is analysed. We assume that it is used in a multiprocessor environment, so that each node corresponds to a processing element as well as a communication element, and each edge is a bi-directional channel connecting 2 nodes. Basically the (alternately-twisted) cube will be used as a message-passing network.

3.1. Routing Algorithms

We will present an algorithm to find the shortest path between any two nodes in an alternately-twisted n-cube. The idea is to utilize the 'twisted edges' as much as possible so as to shorten the path length, that is, the number of hop counts a message has to make on travelling along the path.

For any node u, define $\gamma_i(u)$ as follows:

$$\gamma_i(u) = \begin{cases} u_0, & i = 0 \\ u_{2i}u_{2i-1}, & 1 \le i \le \left\lfloor \frac{n-1}{2} \right\rfloor \\ u_{n-1}, & \text{if } i = \frac{n}{2} \text{ and } n \text{ is even} \end{cases}$$

For example, if n=5, then

$$\gamma_0(u) = u_0, \gamma_1(u) = u_2u_1, \gamma_2(u) = u_4u_3$$

and if n=6, we have

$\gamma_0(u) = u_0, \gamma_1(u) = u_2u_1, \gamma_2(u) = u_4u_3, \gamma_3(u) = u_5$

In this way a binary address is divided into substrings of either 2 bits or 1 bit long. To simplify the notation, the concatenation of $\gamma_i(u)$ and $\gamma_j(u)$ will be written as $\gamma_i\gamma_j(u)$. Let $m = \lfloor \frac{n}{2} \rfloor$, then $u = \gamma_m\gamma_{m-1}\cdots\gamma_1\gamma_0(u)$.

The routing algorithm for determining the path from node s to node t in an AQ_n works as follows. It repeatedly finds the next node to go, starting at node s, by determining the leftmost differing γ_k between the current node address and t, such that γ_k(current node) can be changed to $\gamma_k(t)$ via only one routing step, effected by travelling along the edge connecting the current node to the node $\gamma_m\gamma_{m-1}\cdots\gamma_{k+1}(\text{current node})\gamma_k(t)\gamma_{k-1}\gamma_{k-2}\cdots\gamma_0(\text{current node})$. If no such k exists, then the path will be chosen in such a way as to change the rightmost γ_i of the current node which is different from the corresponding γ_i of t. In this case there is no direct connection in the network to effect the change. Rather, we have to go through 2 nodes, instead of 1, to achieve this. The steps are then repeated until node t is reached. Formally the algorithm is specified below:

<u>Algorithm</u> <u>3.1</u> Finding a path from node s to node t in AQ_n:

(0) $m \leftarrow \left\lfloor \frac{n}{2} \right\rfloor$

(1) $c \leftarrow s$ (*c* holds the current node address)

(2) Do while $c \neq t$

(2.1) let k be the largest integer, $0 \le k \le m$, such that $\gamma_k(c) \neq \gamma_k(t)$ and either (i) $\gamma_k(c) \oplus \gamma_k(t) = 01$ and

$\pi(\gamma_{k-1}\gamma_{k-2} \cdots \gamma_0(c)) = 1$

or (ii) $\gamma_k(c) \oplus \gamma_k(t) = 11$ and

$\pi(\gamma_{k-1}\gamma_{k-2} \cdots \gamma_0(c)) = 0$

or (iii) $\gamma_k(c) \oplus \gamma_k(t) = 10$ or 1 $(if\ k = 0)$

(we use $\oplus$ here to denote pairwise exclusive-or)

(2.2) if such k exists then

$next \leftarrow \gamma_m\gamma_{m-1} \cdots \gamma_{k+1}(c)\gamma_k(t)\gamma_{k-1} \cdots \gamma_0(c)$

(2.2.1) include the edge (c -> next) in the path

c $\leftarrow$ next

else

(2.2.2) let k be the smallest intger, $0 \le \dot{k} \le m$, such that

$\gamma_k(c) \neq \gamma_k(t)$

let $\beta = \bar{c}_{2k}c_{2k-1}$

(2.2.3) include the following subpath to the path being built:

$c \text{ -> } \gamma_m\gamma_{m-1} \cdots \gamma_{k+1}(c)\beta\gamma_{k-1} \cdots \gamma_0(c)$

$\text{ -> } \gamma_m\gamma_{m-1} \cdots \gamma_{k+1}(c)\gamma_k(t)\gamma_{k-1} \cdots \gamma_0(c)$

endif

enddo

EndAlgorithm

As an example, the path between nodes 00000 and 01111 in an AQ_5, found by the algorithm, will be .

00000 -> 00110 -> 01110 -> 01111

It is easy to see that the edge added to the partial path by statement (2.2.1) corresponds to a valid edge in AQ_n. It remains to show that the 2 edges specified in statement (2.2.3) is also valid. First observe that the string $\gamma_k(c)$ chosen in statement (2.2.2) must consist of 2 bits, and it must be that either

$$\gamma_k(c) \oplus \gamma_k(t) = 01 \quad \text{and} \quad \pi(\gamma_{k-1} .. \gamma_0(c)) = 0$$

$$\text{or} \quad \gamma_k(c) \oplus \gamma_k(t) = 11 \quad \text{and} \quad \pi(\gamma_{k-1} .. \gamma_0(c)) = 1$$

In both cases we see that the partial path built by statement (2.2.3) employs 2 valid edges in the graph of AQ_n.

Furthermore, if there are more than one string $\gamma_i(c)$ that differ from the corresponding $\gamma_i(t)$ but do not satisfy the condition in statement (2.1), then after the smallest-subscripted one of them is changed to the desired $\gamma_i(t)$ as effected by the establishment of the path in statement (2.2.3), all the remaining of these strings must satisfy the condition in statement (2.1).

Denote the number of differing $\gamma_i()$'s between two node addresses u and v by $\xi(u, v)$. Then we have:

<u>Theorem 3.1</u> Algorithm 3.1 specifies a routing algorithm for finding a shortest-path from node s to node t in AQ_n. Specifically, if the rightmost differing bit between the two node addresses appear in bit 2k or in bit 2k-1, $k>0$, and

$$\text{either} \quad \gamma_k(s) \oplus \gamma_k(t) = 01 \quad \text{and} \quad \pi(\gamma_{k-1} .. \gamma_0(s)) = 0$$

$$\text{or} \quad \gamma_k(s) \oplus \gamma_k(t) = 11 \quad \text{and} \quad \pi(\gamma_{k-1} .. \gamma_0(s)) = 1$$

then the path length is $\xi(u, v)+1$, otherwise it is $\xi(u, v)$.

Proof: From the connectivity rule of the alternately-twisted cube, it is easy to see that a transition along any edge adjacent to a node u will affect exactly one of the $\gamma_i(u)$'s in the binary address. Thus $\xi(s, t)$ is the lower bound of routing steps required for message transmission between nodes s and t. But if the rightmost differing bits between s and t satisfy the condition stated in the theorem, then it must take 2 routing steps to effect the corresponding change in the node address. Hence the algorithm is actually a shortest path routing algorithm. (Q.E.D.)

Since the algorithm does not use any global, dynamic information about the network, it is immediately a distributed routing algorithm if the destination node address is tagged to the message being sent, and each node on receiving a message will perform statements (2.1) and (2.2) of the algorithm and pass the message to the next node along the partial path specified in statement (2.2.1) or in statement (2.2.3).

Before closing this section, we would like to add that it is more complicated in the alternately-twisted cube than in the hypercube for finding the optimal set of distinct paths between any 2 nodes. (By optimal set we mean that, over all the sets of distinct paths between the two nodes, the length of the longest path in the set will be minimum.) The reason is that by changing an γ_i in the address node, the parity of the address may be altered as well. Since this parity affects directly the choice of the shortest path, the order of changing the γ_i's are important. Thus it makes the guarantee of distinction among the chosen paths to be difficult. We will leave it as another open problem to be solved.

3.2. Message Transmission: Static Analysis

Let h(n,d) denote the number of nodes whose shortest distance from a node u in an AQ_n is d. Clearly this number is the same for any node u because AQ_n is node-symmetric. The mean internode distance of AQ_n, i.e. the average hop counts required for a message transmission, is then given by

$$\bar{d}_n = \frac{\sum_{d=1}^{\lfloor \frac{n}{2} \rfloor + 1} d \cdot h(n,d)}{2^n - 1}$$

(assume that a node will never send a message to itself)

By looking into the working principle of the shortest path routing algorithm, we can see that the function h(n,d) satisfies the recurrence relation

h(2k,d) = h(2k-1,d) + h(2k-1, d-1)

for n=2k. The reason is that for any 2 nodes u and v in AQ_{2k} to be apart by a distance of d, either $\gamma_k(u) = \gamma_k(v)$ or $\gamma_k(u) \neq \gamma_k(v)$ (note that $\gamma_k(u)$ and $\gamma_k(v)$ each consists of 1 bit only), which gives rise to the first and second terms of the equation respectively.

If n=2k+1, k>0, h(2k+1,d) is calculated as follows. For any 2 nodes u and v to be seperated by d hops, we have the following 4 cases: (note that $\gamma_k(u)$ and $\gamma_k(v)$ each must consist of 2 bits)

(i) $\gamma_k(u) \oplus \gamma_k(v)$=00:

The contribution of this case to the value of h(2k+1,d) will be h(2k-1, d), since u and v are actually within the same $\sigma(AQ_{2k+1}, \{\gamma_k(u).X^{2k-1}\})$ (alternately-twisted) subcube;

(ii) $\gamma_k(u) \oplus \gamma_k(v) = 10$

The change of $\gamma_k(u)$ to $\gamma_k(v)$ in the node address can be effected by just one transition along the 2k-th dimension. The number of nodes v, given any u, falling into this case is h(2k-1, d-1);

(iii) $$\gamma_k(u) \oplus \gamma_k(v) = \begin{cases} 01 & \text{and } \pi(\gamma_{k-1}\gamma_{k-2}\cdots\gamma_0(u)) = 1 \\ 11 & \text{and } \pi(\gamma_{k-1}\gamma_{k-2}\cdots\gamma_0(u)) = 0 \end{cases}$$

In either subcases, the change from $\gamma_k(u)$ to $\gamma_k(v)$ in the node address is achieved by 1 transition only, along the (2k-1)-th dimension. Therefore the number of nodes v, given any u, satisfying this case is h(2k-1, d-1);

(iv) $$\gamma_k(u) \oplus \gamma_k(v) = \begin{cases} 01 & \text{and } \pi(\gamma_{k-1}\gamma_{k-2}\cdots\gamma_0(u)) = 0 \\ 11 & \text{and } \pi(\gamma_{k-1}\gamma_{k-2}\cdots\gamma_0(u)) = 1 \end{cases}$$

In this case, the change from $\gamma_k(u)$ to $\gamma_k(v)$ in the node address may take 1 or 2 steps. If it takes 2 steps, it must be that throughout the course of changing $\gamma_{k-1}\gamma_{k-2}\cdots\gamma_0(u)$ to $\gamma_{k-1}\gamma_{k-2}\cdots\gamma_0(v)$ the parities of the involved node addresses are not altered at all. This happens only when the γ_i's differing between $\gamma_{k-1}\gamma_{k-2}\cdots\gamma_0(u)$ and $\gamma_{k-1}\gamma_{k-2}\cdots\gamma_0(v)$ are of the form $\gamma_i(u) \oplus \gamma_i(v) = 10$, and there must be d-2 such γ_i pairs. The total number of nodes satisfying this subcase, for fixed u, is thus $\binom{k-1}{d-2}$.

If the change from $\gamma_k(u)$ to $\gamma_k(v)$ in the node address takes 1 step only, there must exist at least one parity change in the node address on travsering along the path from u to v. There will be h(2k-1,d-1) - $\binom{k-1}{d-1}$ nodes satisfying this subcase.

In total, the number of nodes falling into case iv, for any fixed u, is

$$\binom{k-1}{d-2} + h(2k-1,d-1) - \binom{k-1}{d-1}.$$

Summing the number of nodes in each of these 4 cases, we get

$$h(2k+1, d) = h(2k-1, d) + 3\,h(2k-1, d-1) + \binom{k-1}{d-2} - \binom{k-1}{d-1}$$

Starting with the basis h(1,0)=h(1,1)=1, the value of h(n,d) and hence the value of $\overline{d}_n$ for any n>1 can be calculated. Figure 3.1 shows the plot of $\overline{d}_n$, for 1≤n≤21. The mean internode distance of the binary n-cube, which can be shown to be $\frac{\sum_{d=1}^{n} d\cdot\binom{n}{d}}{2^n-1}$, is also plotted in the same figure. In general the mean internode distance of the alternately-twisted n-cube is smaller than that of the n-cube. Figure 3.2 shows the comparison in terms of the percentage saved in the $\overline{d}_n$ of alternately-twisted cube over the hypercube. It is noted that the asymptotic improvement is at about 22% and the improvement is more striking for odd values of n (which is the same for the improvement in the diameter measure). It is because twisted edges of AQ_n are found in alternate dimensions, so AQ_n of odd dimension has relatively larger proportion of twisted edges which are accountable for the shortening of the paths.

The shorter length of an average path in the alternately-twisted cube will result in smaller traffic flow over the edges as compared with that on the hypercube, because on average fewer number of edges will be used for transmitting a message in the former. Quantitatively this is reflected in the traffic density measure. The average traffic density over an edge along the k-th dimension in an AQ_n, denoted as $\tau(n,k)$, is defined to be the ratio of the average number of messages crossing the set of edges along the k-th dimension, to the total number of edges in this set, assuming each node will contribute to the message poppulation by sending a message to a random destination.

For each source node u in AQ_n, let the number of possible destination nodes, such that each corresponding shortest path requires a traversal over an edge along the k-th dimension, be denoted by $\delta(n,k)$. There are altogether 2^n source nodes, each of which will behave (statistically) identically. For each source there are 2^n-1 potential destination nodes (because no node will send a message back to itself). And, there are 2^{n-1} edges along each dimension. Therefore,

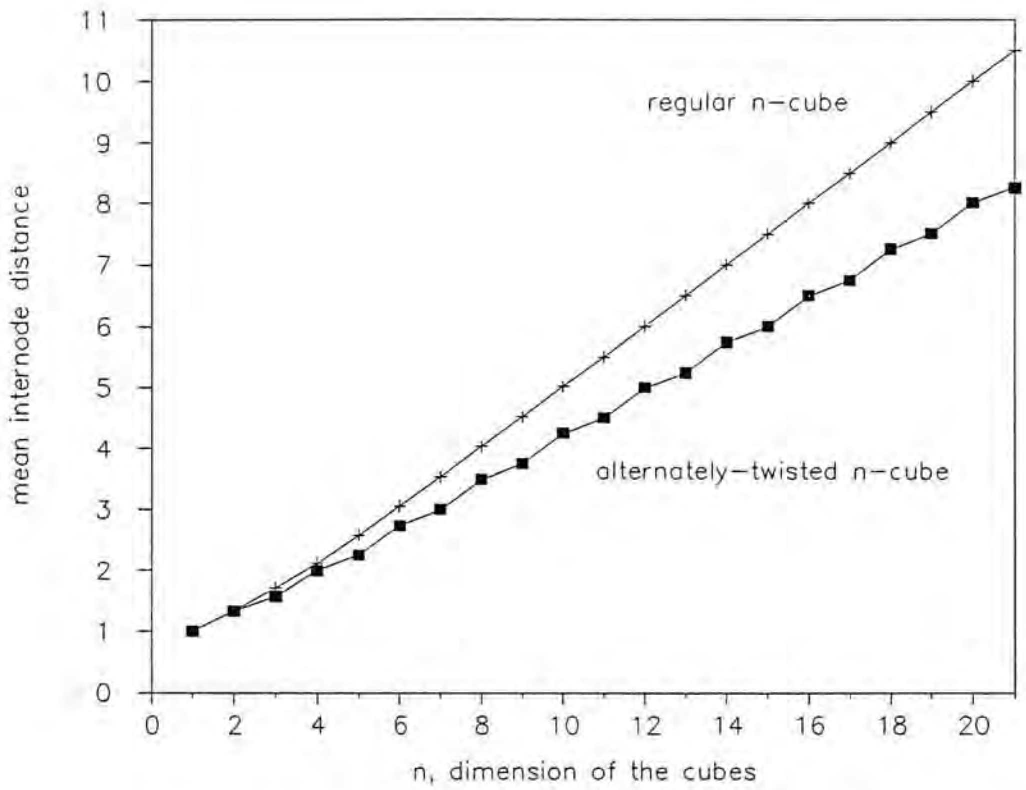


Figure 3.1: Mean Internode Distance comparison between the alternately-twisted cube and the binary n-cube

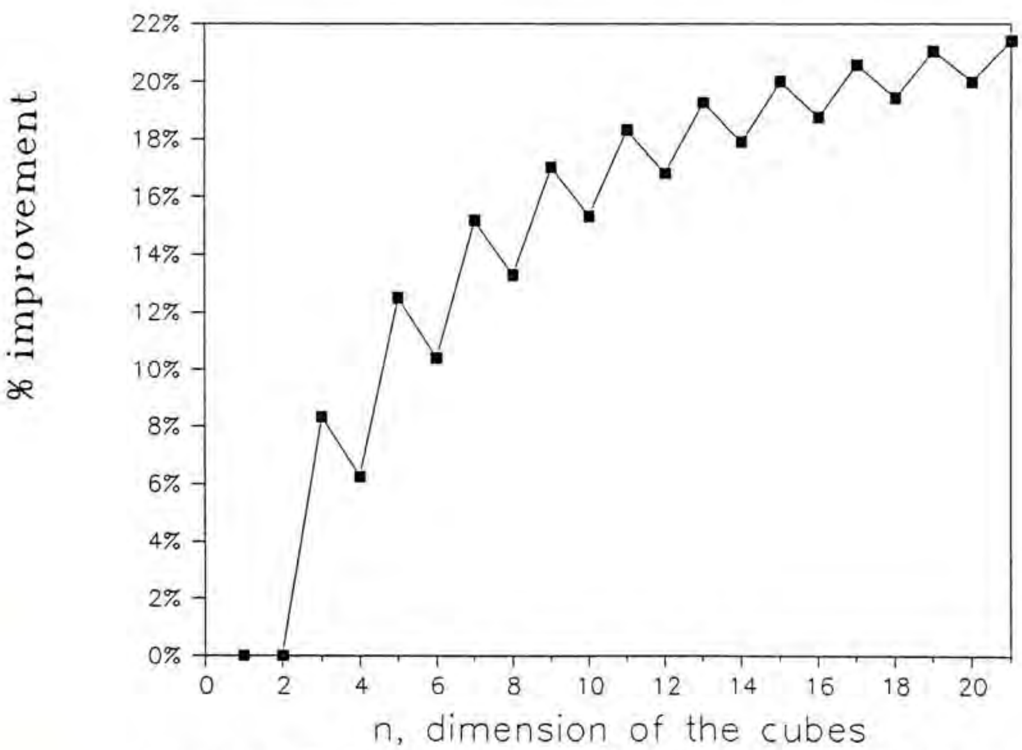


Figure 3.2: Improvement of mean internode distance of the alternately-twisted n-cube over that of the n-cube

$$\tau(n,k)=\frac{2^n\cdot\delta(n,k)}{(2^n-1)\cdot 2^{n-1}}$$

Clearly, $\delta(n,0) = 2^{n-1}$, so $\tau(n,0) = 2^n/(2^n-1)$.

Also, $\delta(n, 2k-1) = 2^{n-1}$ because for any (source, destination) pair (u,v), if $u_{2k-1} \neq v_{2k-1}$, the path between them must use exactly once an edge along the (2k-1)-th dimension. Therefore

$$\forall\quad 0\le k\le\left\lfloor\frac{n}{2}\right\rfloor,\quad \tau(n,2k-1)=\frac{2^n}{2^n-1}$$

For a path between node u and v to take an edge along the (2k)-th dimension, it must be that either (i) $\gamma_k(u)\oplus\gamma_k(v) = 10$, or (ii) during the course for converting $\gamma_k(u)$ to $\gamma_k(v)$ in the node address, the path has to take on two edges, one along the (2k)-th dimension and one along the (2k-1)-th dimension. The condition for case (ii) is that

$$(a)\qquad \gamma_k(u)\oplus\gamma_k(v)=01 \quad\text{and}\quad \pi(\gamma_{k-1}\ldots\gamma_0(u))=0$$

$$\text{or}\quad \gamma_k(u)\oplus\gamma_k(v)=11 \quad\text{and}\quad \pi(\gamma_{k-1}\ldots\gamma_0(u))=1$$

and

$$(b)\quad \forall\quad 0\le i\le k-1\quad \gamma_i(u)\oplus\gamma_i(v)\in\{0,00,10\}$$

Therefore, for $1\le k\le\left\lfloor\frac{n-1}{2}\right\rfloor$,

$$\begin{aligned}\delta(n,2k)&=2^{n-2}+2^{n-1-2k}\cdot 2^{k-1}\\&=2^{n-2}+2^{n-2-k}\end{aligned}$$

The first term of the right hand side is due to those satisfying case (i) and the second term is attributed by those satisfying case (ii). Hence,

$$\tau(n,2k) = \frac{2^n \cdot (2^{n-2} + 2^{n-2-k})}{2^n - 1} \cdot \frac{1}{2^{n-1}}$$

$$= \frac{2^{n-1} + 2^{n-1-k}}{2^n - 1}$$

The plot of the values of $\tau(n,i)$ against n is shown in Figure 3.3 It can been seen that the average traffic densities over the edges along the 0-th and the odd-numbered dimensions are the highest. Each of these edges will serve for about one message of the previously mentioned message population. Note that this value is the same as the average traffic density of a binary n-cube, in which the traffic density is the same over all edges because of the edge-symmetric property. Hence the smaller average internode distance of an AQ_n does not cause higher traffic congestion in any localized points in the $AQ_n$when compared to the binary n-cube. Rather it results in reduced message flow over the edges along the even-numbered dimension. The higher dimension such edges belong to, relatively the less frequent they are used. It is noted that edges along the 10-th or larger, even-numbered dimensions have about only 50% utilization of those along the 0-th or odd-numbered dimensions. The implication is that rather by distributing the communication resources (e.g. channel bandwidth, buffers, amount of time shared, etc) evenly over all the edges of an alternately-twisted n-cube network, it may be more efficient by biasing their allocation according to the relative traffic flow over them as suggested by Figure 3.3.

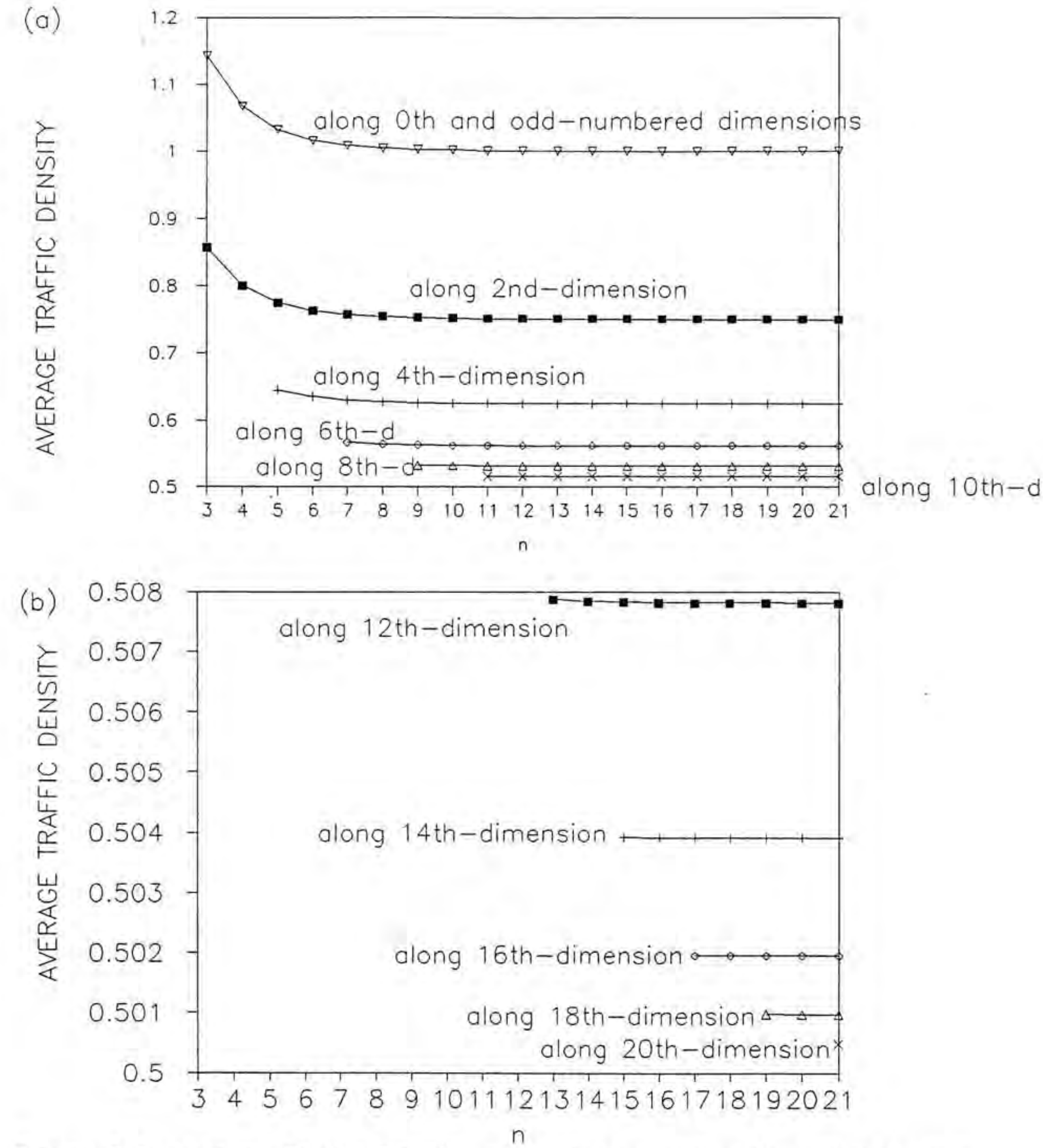


Figure 3.3: Average Traffic Density over edges along various dimensions in the alternately-twisted n-cube

3.3. Message Transmission : Dynamic Analysis

In this section the performance of the alternately-twisted cube as a message-passing network will be investigated under the dynamic environment, i.e. blocking of messages due to resource contention is taken into consideration. We will adopt the analytical technique proposed in [AbPa89]. The model assumes that each node of the AQ_n is composed of a switch and a processing element respectively for communication and computation purposes. The switch has (n + 1) output queues, n of which is assoicated with the n output ports of the node, and the remaining one is used for sending/receiving messages between the processing element and the switch. Messages contending for the same output port will be buffered in these queues. The model also assumes infinte queue length, so that no message will be lost and retransmission is not required. Further, the node is working in the multiple-accepting mode, in which up to n messages can be accepted by the processing element in one cycle, and at the same time the switch can simultaneously send out at most n messages. The performance of the network is measured in terms of the average message delay under the simultaneous, multiple message transmissions environment, where the non-adaptive shortest path routing algorithm proposed in Section 3.1 is used.

At steady state, at any node u the probability of an entering message (that is, not gererated by the processing element of that node) that has i more hops(or edges) to go is given by

$$\frac{\sum_{j=i+1}^{d} h(n,j)}{\sum_{k=1}^{d}\sum_{j=k}^{d} h(n,j)} \quad \text{where} \quad d = \left\lfloor \frac{n}{2} \right\rfloor + 1$$

The numerator is the count of nodes that is more than i hops from u, and the denominator is the sum of all these counts for 1≤i≤d, d being the diameter of the AQ_n. When i=0, it is the probability for an incoming message reaching its destination, i.e. the probability of termination, and is given by

$$P_t(n) = \frac{\sum_{j=1}^{d} h(n,j)}{\sum_{k=1}^{d}\sum_{j=k}^{d} h(n,j)} \quad \text{where} \quad d = \left\lfloor \frac{n}{2} \right\rfloor + 1$$

Now denote the message generation rate at each processing element by g. Clearly it is also the load of the whole network. Then the arrival rate of messages from a particular input port of a node is given by m, where

$$m = \frac{g}{n \cdot P_t(n)}$$

(Because there is no message loss, the birth rate of a message at a node, g, should be equal to the death rate, $n \cdot m \cdot P_t(n)$.) From the result of the last section, we can infer that the rate of message arrival/departure at different ports of a node is not the same if all the ports are identical. However, we can make all the rates equal by allocating resources to the ports in a non-even distribution as discussed at the end of the last section. Here we assume that this is the case. Then by the result of [AbPa89, section IVB], the average number of mesages existing in an output queue of a node in the steady state is given by

$$b = m + \frac{m^2 \cdot (n \cdot (1 - P_t^2(n)) - 2 \cdot (1 - P_t(n)))}{2 \cdot (n-1) \cdot (1-m)}$$

and the average time a message has to wait at an intermediate node is therefore b/m. Since an average message has to go through $\overline{d}_n$ nodes, the delay for the routing is thus $\overline{d}_n \cdot \frac{b}{m}$. Note that there is an additional cycle needed for a message generated by a processing element to be transferred to the switch in the same node. Hence the average message delay in an AQ_n is given by

$$\overline{d}_n \cdot \frac{b}{m} + 1$$

The values of this function is plotted in Figure 3.4 against the dimensions of the alternately-twisted cubes. In the figure, the load is actually the g in the analysis. In the same figure are also shown the corresponding values of the hypercubes, using the same analysis. It can be seen that the performance of the AQ_n is much better than the binary n-cube: the average delay in AQ_n with a load of 1.0 is just slightly greater than that in the hypercube with a load of 0.1, and is smaller than those in the hypercube with a load of 0.5, 0.75, and 1.0 respectively. Figure 3.5 is another view of the same result, showing the percentage improvement. It is not surprised to note that the improvement is more striking when the dimension of the AQ_n is an odd number, as is in the case of the mean internode distance measure. Also, more relative saving of transmission time is achieved at higher load. And, when n exceeds 6, the savings is already more then 10%, even at a load of 0.1, and the asymptotic value of the percentage improvement is about 30% when the load is equal to 1. Figure 3.6 shows the variation of the average delay when the load is varied. It is evident that the gap between the curves for the AQ_n and for the n-cube of corresponding size is wider when the load approaches 1.0. Thus the superiority of the alternately-twisted cube over the hypercube as an interconnection network is more significant when it is heavily loaded.

3.4. Broadcasting

We will consider the case for message broadcasting from a particular node to all the nodes, i.e. one-to-all broadcasting, in the alternately-twisted n-cube. Based on the shortest-path routing algorithm, here is the distributed algorithm for the broadcasting:

<u>Algorithm</u> <u>3.2</u> One-to-All Broadcasting algorithm executed at each node of an AQ_n

, assuming the origin of the message is at node s

(let the address of the node be u)

if u=s then

(1) for $0 \le i \le n-1$,

send the message over the edge along the i-th dimension

else

(2) wait for a message from one of its neighbours

(3) suppose it comes from the edge along the k-th dimension

(4) CASE of k --

(a)

(b)

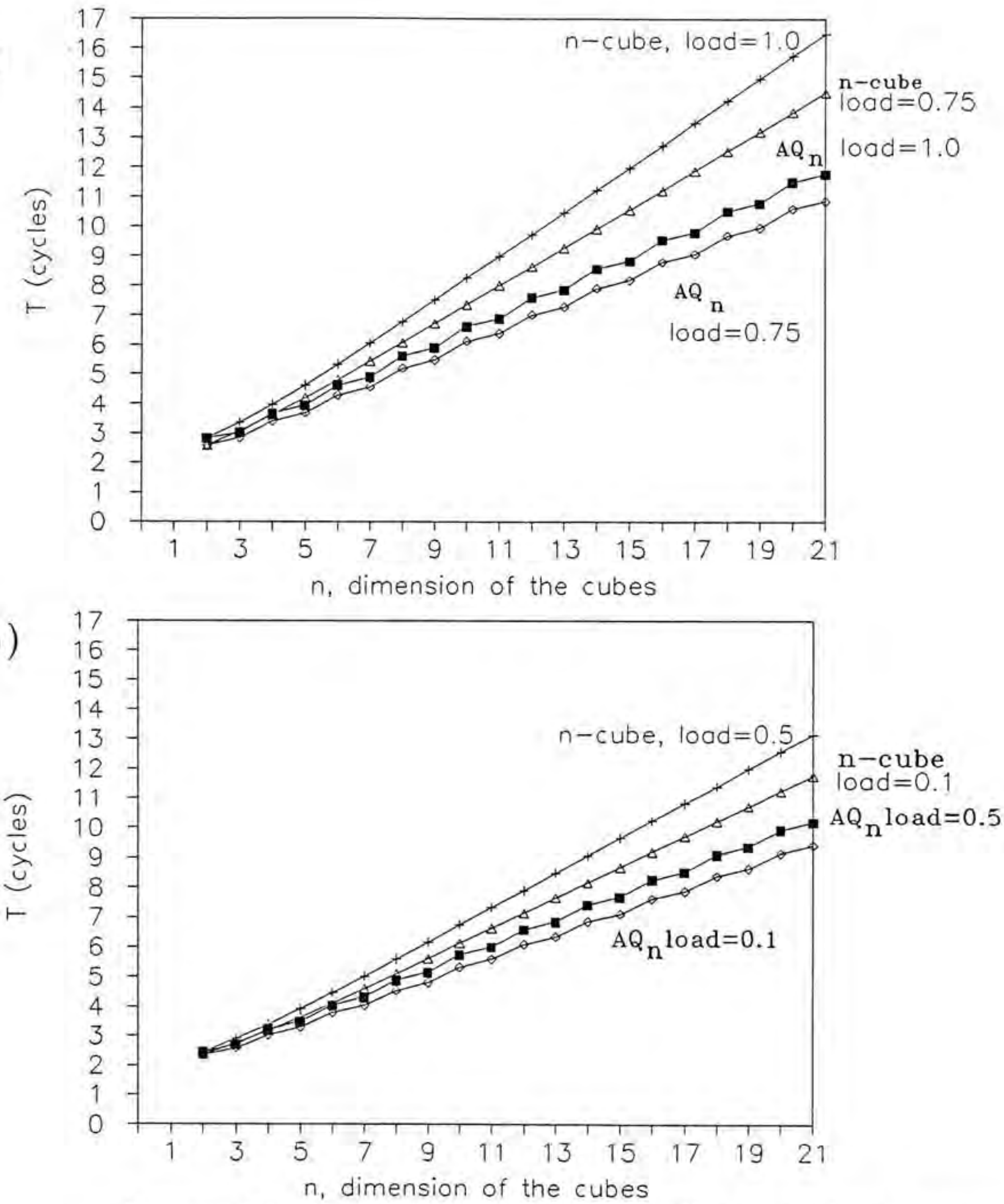


Figure 3.4: Average message delay (T) comparison, assuming infinite buffers for all networks

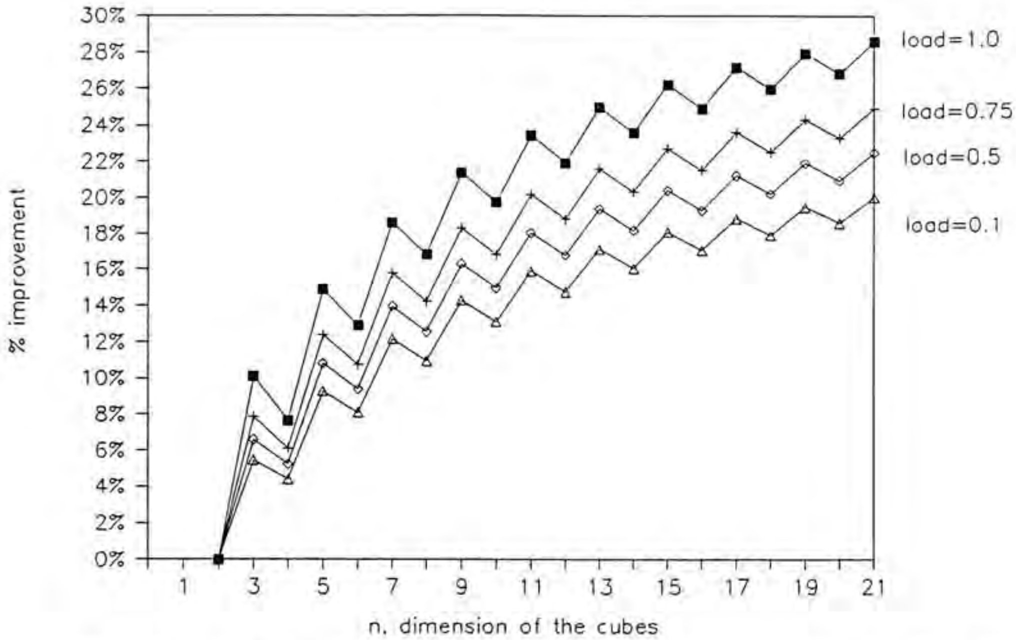


Figure 3.5: Improvement of average message delay in the alternately-twisted n-cube over that in the binary n-cube

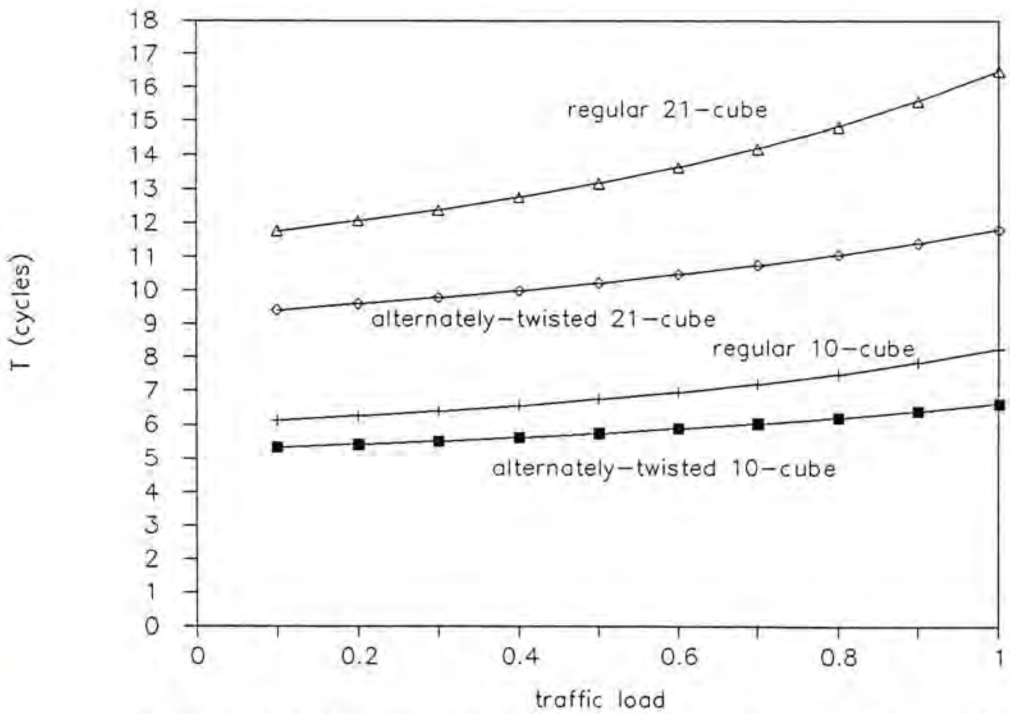


Figure 3.6: Variation of the average message delay (T) under different loads

(i) k=0:

$S = \{2i-1 \mid 1 \le i < j, \text{ and } \gamma_i(u) \oplus \gamma_i(s) = 00\}$

where j is the smallest integer >0 such that $\gamma_j(u) \oplus \gamma_j(s) = 01$ or 1 1, (if no such j exists, then j = $\left\lfloor \frac{n-1}{2} \right\rfloor + 1$)

(ii) k is even and not equal to 0:

$S = \{i \mid 0 \le i \le k-1\} \cup \left\{2i-1 \mid \frac{k}{2} < i < j \text{ and } \gamma_i(u) \oplus \gamma_i(s) = 10\right\}$

where j is the smallest integer > k/2 such that $\gamma_j(u) \oplus \gamma_j(s) = 01$ or 1 1, (if no such j exists, then j=$\left\lfloor \frac{n-1}{2} \right\rfloor + 1$)

(iii) k is odd and not equal to n-1:

set j to be the smallest integer > (k+1)/2 such that $\gamma_j(u) \oplus \gamma_j(s) = 01$ or 1 1, (if no such j exists, then let j=$\left\lfloor \frac{n-1}{2} \right\rfloor + 1$
)

set m to be the largest integer ≤ (k-1)/2 such that $\gamma_m(u) \oplus \gamma_m(s)$ = 01 or 11, or 1, (if m=0, if no such m exists, then m = -1)

if m = -1 then

$$S = \left\{2i-1 \mid \frac{k+1}{2} < i < j \text{ and } \gamma_i(u) \oplus \gamma_i(s) = 00\right\} \cup S'$$

else

$$S = \left\{2i-1 \mid m < i \le \frac{k-1}{2}\right\} \cup S''$$

where

if $u_{k-1}..u_0 = s_{k-1}..s_0$ and

$((u_{k+1}u_k \oplus s_{k+1}s_k = 01$ and $\pi(u_{k-1}...u_0) = 1)$ or

$(u_{k+1}u_k \oplus s_{k+1}s_k = 11$ and $\pi(u_{k-1}...u_0) = 0))$ then

$S' = \{i \mid 0 \le i \le k-1\}$

otherwise $S' = \varnothing$

and

if $u_{2m-2}..u_0 = s_{2m-2}..s_0$ and

$((u_{2m}u_{2m-1} \oplus s_{2m}s_{2m-1} = 01$ and $\pi(u_{2m-2}...u_0) = 1)$ or

$(u_{2m}u_{2m-1} \oplus s_{2m}s_{2m-1} = 11$ and $\pi(u_{2m-2}...u_0) = 0))$ then

$S'' = \{i \mid 0 \le i \le 2m\}$

otherwise $S'' = \varnothing$

(iv) k is odd and k=n-1:

$S = \{i \mid 0 \le i < n-1\}$

endcase

(5) for all $i \in S$,

send a replication of the received message over the edge along the i-th dimension

EndAlgorithm

The spanning tree (i.e. the tree-structured subgraph consisting of all the nodes of the network) arising from this broadcasting algorithm will be one such that the path from the root to any tree node is a shortest path. Figure 3.7 shows an example for the spanning of the one-to-all broadcasting in an AQ_5, the message being originated from node 00000. It is compared to the corresponding spanning tree in a binary 5-cube, shown in Figure 3.8. In general it can be seen that the broadcsting in an AQ_n takes $\lfloor \frac{n}{2} \rfloor + 1$ cycles, assuming a node can handle multiple messages per cycle. This is about 50% of that of the hypercube.

However, if each node can only accept and send a single message per cycle, then the communication time for one-to-all broadcasting may be the same in both the alternately-twisted n-cube and the binary n-cube, assuming each intermediate node of the corresponding spanning trees will send the message to the largest subtree first. It is evident by comparing the two spanning trees in Figures 3.7 and 3.8. Each requires 5 message cycles for accomplishing the broadcast. In general, it is known that one-to-all broadcasting in the binary n-cube takes n cycles with each node operating in the single-accepting mode [JoHo89]. And it is conjectured that the alternately-twisted n-cube happens to take the same time.

In summary, we have

Theorem 3.2 Algorithm 3.2 describes an one-to-all broadcasting in an AQ_n, which takes $\lfloor \frac{n}{2} \rfloor + 1$ routing cycles assuming the nodes are operated in the multiple-message-accepting mode.

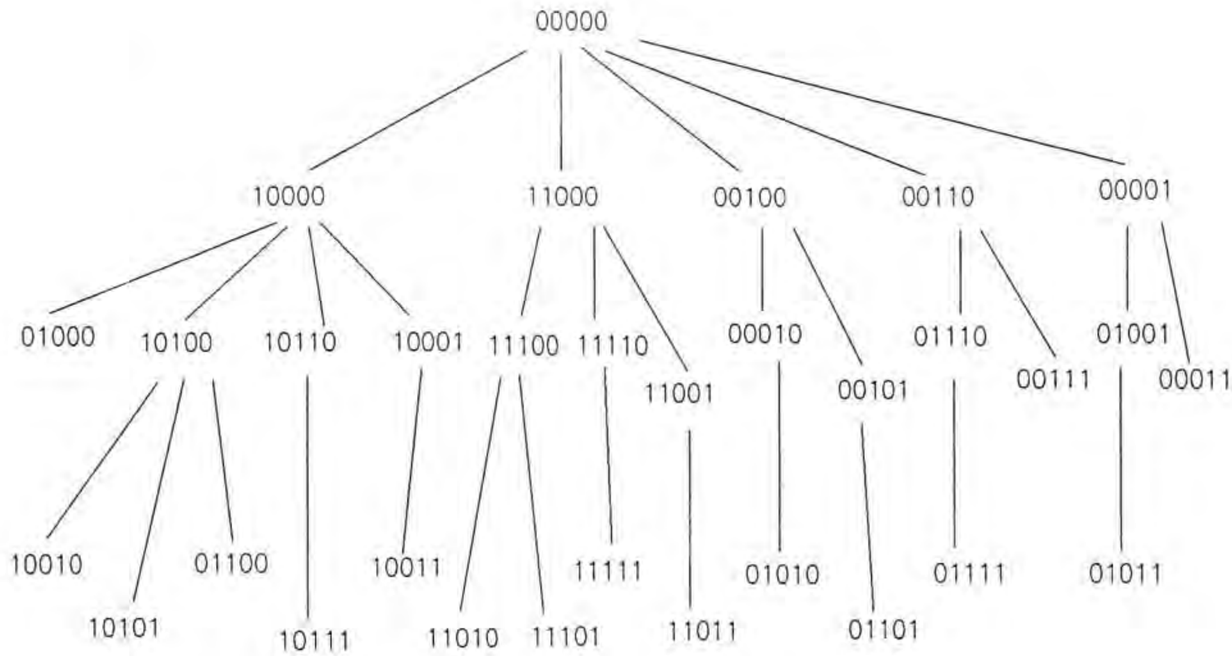


Figure 3.7: Spanning tree of AQ_5, rooted at node 00000

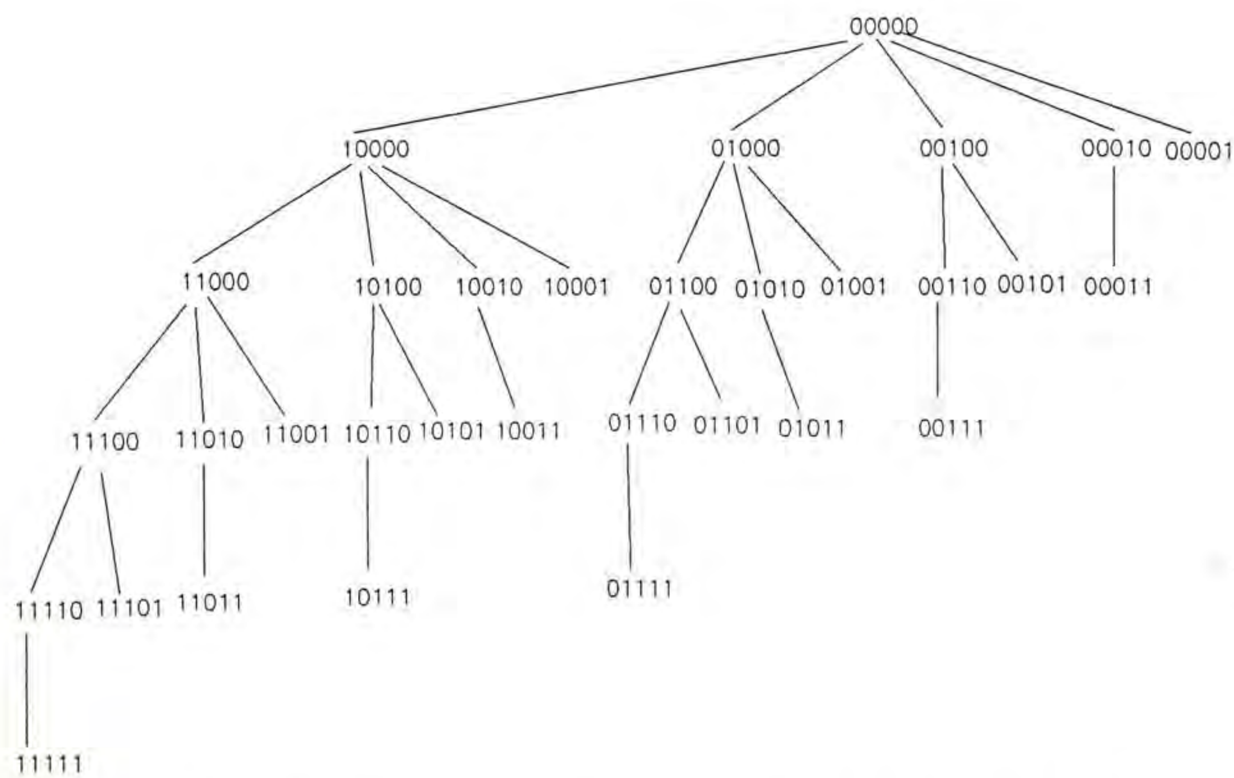


Figure 3.8: Spanning tree of a binary 5-cube, rooted at node 00000

Chapter 4

Parallel Processing on the Alternately-Twisted Cube

The flexibility of the alternately-twisted cube for supporting parallel processing is demonstrated in this chapter. Three kinds of parallel algorithms, namely, the Ascend/Descend class, the combining class, and the numerical algorithms are shown to be easily applied on the AQ_n structure. In general, from the discussion in Section 2.2.5.4 the efficiency of executing these algorithms (as well as others) will be about the same as, to within an overhead of a factor of no more than 2, that of the hypercube. In other words, we have

Corollary 4.1 Any problem solvable in a binary n-cube with time complexity O(f(n)) will also be solvable in an AQ_n with the same time complexity. The factor of difference between the actual time required on the two structures will fall within the range (0.5, 2.0).

Therefore we will focus on the relative communication time required to solve the problems on the alternately-twisted cube as compared to that on the hypercube. We refer an exchange cycle to be the time required for a message (the length of which is dependent on the problem) to be sent from a node to its direct neighbour along an edge in the graph.

4.1. Ascend/Descend Class Algorithms

Suppose that there are initially $N=2^n$ data, $a_0, a_1, \ldots, a_{N-1}$, respectively stored at locations L(0) .. L(N-1). The Ascend/Descend class of algorithms is defined in [PrVu81] to be those that iterate for an index i from 0 to n-1 (for Ascend class) or from n-1 down to 0 (for Descend class), and during each iteration, for each location L(x), its data is modified by a computation using data at L(x) and L(y), where the binary forms of x and y differs in only the i-th bit. Examples of this class include the N-point Fast Fourier Transform (FFT) and convolution algorithms. Obviously execution of algorithms in this class needs only O(log n) parallel time in a binary n-cube.

For instance, the Ascend class algorithms on the binary n-cube can be specified in the general form below:

(Assume each node u is already preloaded with its initial data, a_u and stores it at local storage data(u) within each node)

For i=0 to n-1

  do in parallel for each u : $0 \le u \le 2^n - 1$

    let $u' = u \oplus 2^i$

    data(u) <- OP(i, u, data(u), data(u'))

  enddo

endfor

where OP(i, u, data(u), data(u')) is the specific operation (depending on particular application) that performs computation on the two data at data(u) and data(u') and may depend on the parameters i and u. The result is stored back in data(u). Clearly such operations on nodes u and u' will require a data exchange over the link along the i-th dimension between the two nodes. Therefore the communication time of the Ascend class algorithms on the hypercube is exactly n exchange cycles.

It turns out that the Ascend class algorithms can also be run on the AQ_n with the same communication time. The idea is to change the location of carrying out the OP() computation and storage of its result during some iterations, in such a way as to cancel the effect of the "twisted" edges of the cube on the addressing so that appropriate data can still be aligned properly for data exchange in the next iteration. Beside, the data a_u originally assigned to node u in the beginning of the hypercube has to be preloaded to another node v, where v is defined by the following permutation:

$$v = ascend\_permute(u) \equiv \begin{cases} u_{n-2}u_{n-1}u_{n-4}u_{n-3}\ldots u_1u_2u_0 & \text{if n is odd} \\ u_{n-1}u_{n-3}u_{n-2}u_{n-5}u_{n-4}\ldots u_1u_2u_0 & \text{if n is even} \end{cases}$$

i.e. bit pairs at positions (2k,2k-1) are reversed individually, for $1 \le k \le \left\lfloor \frac{n-1}{2} \right\rfloor$. The Ascend class algorithm for the AQ_n is given below:

<u>Algorithm</u> <u>4.1</u> Ascend class algorithms on an AQ_n

(Assume that each node u is preloaded with the initial data a_v and stores it at location data(u) within the node, where u=ascend_permute(v).)

Do in parallel for each u : $0 \le u \le 2^n - 1$

v = inverse_ascend_permute(u)

/* it happens to be the same as ascend_permute(u) */

index(u) = v

data(u) = OP(0, v, data(u), data($u \oplus 2^0$))

enddo

For i=1 to $\left\lfloor \frac{n-1}{2} \right\rfloor$

do in parallel for each u : $0 \le u \le 2^n - 1$

```
        let u' be the neighbour of u along the (2i)-th dimension
            (ie. u' = u ⊕ 2^{2i})
        if π(u_{2i-2}..u_0) = 1   or   u_{2i-1} = 0
            data(u) = OP(2i-1, index(u), data(u), data(u'))
        else /* exchange OP() computation between nodes u and u' */
            data(u) = OP(2i-1, index(u'), data(u'), data(u))
            index(u) = index(u')
        endif
        let u" be the neighbour of u along the (2i-1)-th dimension
        data(u) = OP(2i, index(u), data(u), data(u"))
    enddo
endfor
If n-1 is odd
    let u' be the neighbour of u along the (n-1)-th dimension
    data(u) = OP(n-1, index(u), data(u), data(u'))
endif
EndAlgorithm
```

The variable index(u) in the algorithm is used to store the address of the node w such that node u of the alternately-twisted n-cube will perform the computation as if it is node w of the binary n-cube running the corresponding algorithm. In other words, algorithm 4.1 can be regarded as the (dynamic) emulation of a binary n-cube by an AQ_n for the execution of the Ascend class algorithms, in which node u of AQ_n is emulating node index(u) of the n-cube at the corresponding instance in the course of execution of the algorithm. For example, Figure 4.1 depicts the execution of the algorithm running on an AQ_3. The bracketed string associated with each node refers to the value of the variable index at that node after the iteration. The arrowed lines indicate the edges along which data exchange takes place in that iteration.

Note that data(u) and index(u) are local storage of node u. Thus the OP() computations will excite simultaneous data exchanges between adjacent nodes along the appropriate dimension. Therefore the algorithm runs in O(n) time and the total communication time occupies exactly n exchange cycles.

The correctness of the algorithm is justified by the observation that after iteration i, $0 \leq i \leq \left\lfloor \frac{n-1}{2} \right\rfloor$, node u of AQ_n is emulating node v of the n-cube where

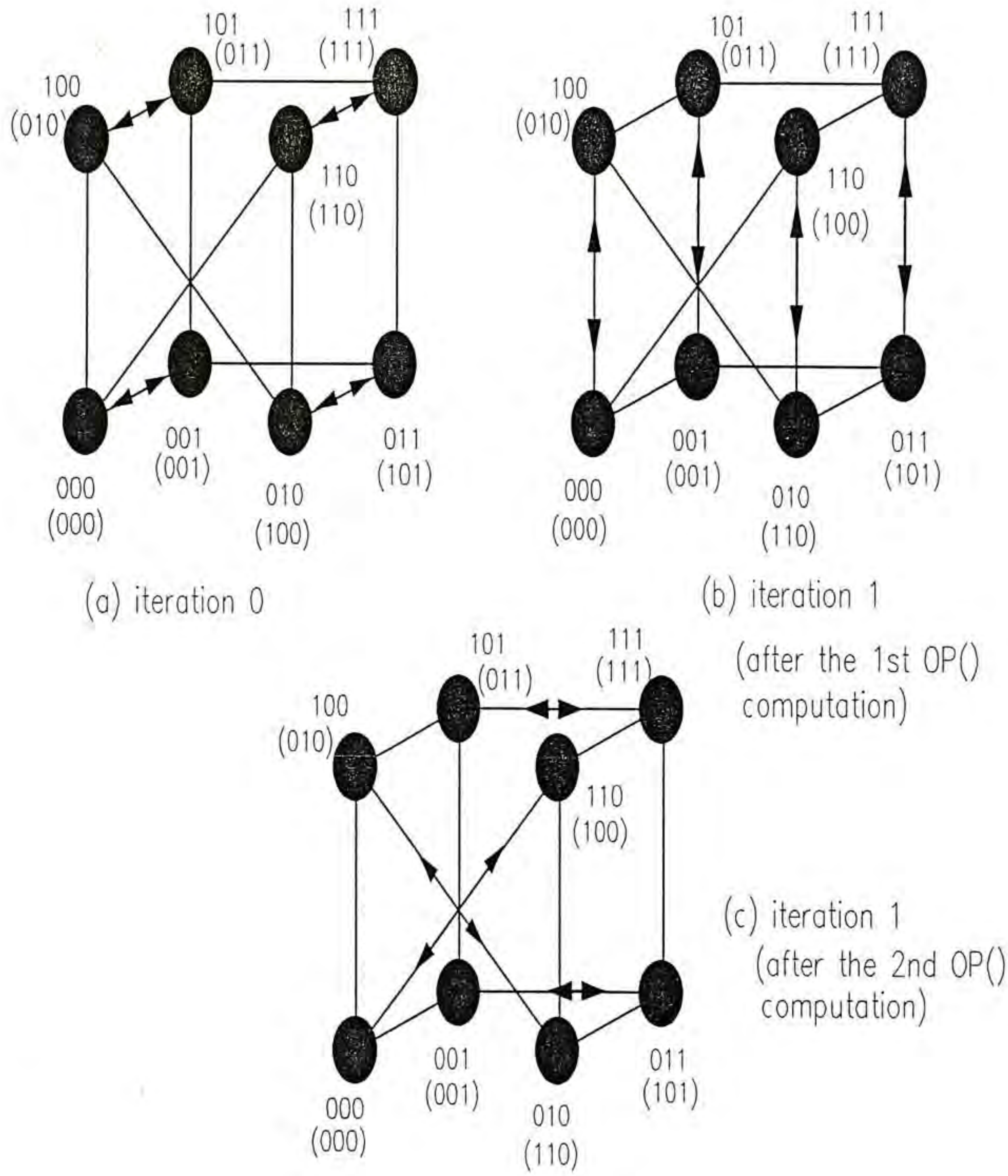


Figure 4.1: Execution of Ascend class algorithm on an AQ_3

$v_{n-1} = u_{n-1}$ if n is even

$\forall \quad i < k \leq \left\lfloor \frac{n-1}{2} \right\rfloor,$

$v_{2k} v_{2k-1} = u_{2k-1} u_{2k}$

$\forall \quad 1 \leq k \leq i,$

$$v_{2k} v_{2k-1} = \begin{cases} u_{2k-1} u_{2k} & \text{if } \pi(u_{2k-2} .. u_0) = 1 \text{ or } u_{2k-1} = 0 \\ u_{2k-1} \bar{u}_{2k} & \text{otherwise} \end{cases}$$

$v_0 = u_0$

It is also worthwhile to note that locations of the set of results of the algorithm will be permuted from that of the algorithm for the hypercube. The mapping can be obtained from above by setting $i = \left\lfloor \frac{n-1}{2} \right\rfloor$.

Since the Descend class is just a dual class of the Ascend class (the duality is established by applying a bit reversal permutation on the index of the data set a_i's [PrVu81]), the time complexity of running a Descend class algorithm on the AQ_n is also O(n), and the communication time is exactly n exchange cycles.

4.2. Combining Class Algorithms

The feature of this class is that, given a set of N data, a_i's for i = 0, 1, 2, ..., N-1, the algorithm computes the value of $a_0 \otimes a_1 \otimes .. \otimes a_{N-1}$ where $\otimes$ is an associative binary operator. Typical examples include the MAX, MIN, SUM and PRODUCT operations over a set of data, which are usually employed in database application. Because of the associativity, algorithms within this class is highly parallelizable. In particular, they can be run in O(n) time on an AQ_n, if the size of the data set is no more than 2^n. The general algorithm is given below:

Algorithm 4.2 Combining class algorithms on an AQ_n

(Assume each node u is preloaded with a data item a_u stored locally at data(u) within the node.)

For i=n-1 down to 0

    do in parallel for each u : $0 \le u \le 2^n - 1$

        if u_i=0

            let u' be the neighbour of u along the i-th dimension

            data(u) = data(u) ⊗ data(u')

        endif

    enddo

endfor

EndAlgorithm

The result will be stored at node 0. At iteration i there will be data transmissions over the edges along the i-th dimension. Hence the communication time of the algorithm amounts to exactly n exchange cycles. Clearly the same algorithm is also applicable on a binary n-cube, and there is no difference in the time complexity for both cubes.

4.3. Numerical Algorithms

Parallel algorithms for many numerical problems usually exhibit regular communication patterns between the locations where the data is stored and processed. It is interesting to see that the alternately-twisted cube is able to support most of these patterns as efficiently as the hypercube. In some cases, it is even better than the latter. We substantiate this claim by giving 3 typical examples below. The multiple-message accepting node model is assumed.

Matrix Multiplication

Suppose we want to multiply two NxN matrices A and B, where $N=2^n$. It is to be carried out in an AQ_{3n}. The algorithm is modified from the one for a binary 3n-cube, given in [Akl89, p.183]. The nodes of the AQ_{3n} are viewed as forming an NxNxN array structure. A node u is given a unique co-ordinate (i,j,k) where u=ixNxN+jxN+k, 0≤i,j,k≤N-1. The processor in node u has 3 local storages a(u), b(u) and c(u), which are also denoted by a(i,j,k), b(i,j,k) and c(i,j,k) respectively. Initially the matrix elements of A and B, A_{xy} and B_{xy}, are loaded into these storages in such a way that $a(0,j,k)=A_{jk}$ and $b(0,j,k)=B_{jk}$ for 0≤ j,k ≤N-1. The algorithm proceeds in 3 stages:

(i) Distribute the matrix elements so that $a(i,j,k) = A_{ji}$ and $b(i,j,k) = B_{ik}$, 0≤i,j,k≤N-1

(ii) Compute c(i,j,k) = a(i,j,k) x b(i,j,k) simultaneously for all 0≤i,j,k≤N-1

(iii) Combine the results as

$$c(0,j,k)=\sum_{i=0}^{N-1} c(i,j,k)$$

simultaneously for all 0≤ j,k ≤N-1

Stage (i) requires 3 steps. First, values in a(0,j,k) and b(0,j,k) are broadcast along the i-axis, so that as a result $a(i,j,k)=A_{jk}$ and $b(i,j,k) = B_{jk}$ for 0≤i≤N-1. Second, values in a(i,j,i) are broadcast along the k-axis, 0≤i≤N-1, after which it should be that a(i,j,k) = A_{ji}. Third, values in b(i,i,k) are broadcast along the j-axis, 0≤i≤N-1, so that b(i,j,k)=B_{ik} afterwards. We are interested in the communication time required for these broadcasts. Consider two cases for n:

(1) n is odd:

By the way of mapping the tuple (i,j,k) to a node address of AQ_{3n} described before, for any fixed $i=i_{n-1}...i_0$ and $j=j_{n-1}...j_0$, the subgraph $\sigma(AQ_{3n},\{i_{n-1}..i_0 j_{n-1}..j_0 X^n\})$ is an AQ_n. Hence broadcasting along the k-axis can be achieved in $\lfloor \frac{n}{2} \rfloor + 1$ exchange cycles.

For any fixed i and k, however, the subgraph $\sigma(AQ_{3n},\{i_{n-1}..i_0 X^n k_{n-1}..k_0\})$ is generally not a perfect alternately-twisted n-cube. But we can still break it into 4 AQ_{n-2} as follows:

$$\sigma(AQ_{3n},\{i_{n-1}..i_0 0X^{n-3}00k_{n-1}..k_0,\ \ i_{n-1}..i_0 0X^{n-3}y_2y_1k_{n-1}..k_0\})$$

$$\sigma(AQ_{3n},\{i_{n-1}..i_0 0X^{n-3}01k_{n-1}..k_0,\ \ i_{n-1}..i_0 0X^{n-3}y'_2y'_1k_{n-1}..k_0\})$$

$$\sigma(AQ_{3n},\{i_{n-1}..i_1 1X^{n-3}00k_{n-1}..k_0,\ \ i_{n-1}..i_1 1X^{n-3}y_2y_1k_{n-1}..k_0\})$$

$$\sigma(AQ_{3n},\{i_{n-1}..i_1 1X^{n-3}01k_{n-1}..k_0,\ \ i_{n-1}..i_1 1X^{n-3}y'_2y'_1k_{n-1}..k_0\})$$

where

if $\pi(k_{n-1}..k_0)=0$ then

$y_2y_1=11,\quad y'_2y'_1=10$

else

$y_2y_1=10,\quad y'_2y'_1=11$

Then broadcasting along the j-axis is done by first sending simultaneously the data to a corresponding node (depending on the origin of the broadcast) in each of these 4 subcubes, which takes at most 3 exchange cycles, and then each subcube performs individually the remained broadcasting, which takes $\lfloor \frac{n-2}{2} \rfloor + 1 = \lfloor \frac{n}{2} \rfloor$ exchange cycles. Hence the total time for broadcasting along the j-axis is $\lfloor \frac{n}{2} \rfloor + 3$ exchange cycles.

In a similar way, if we fix j and k, then the induced graph $\sigma(AQ_{3n}, \{X^n j_{n-1}..j_0 k_{n-1}..k_0\})$ will contain 4 subgraphs of AQ_{n-2}:

$\sigma(AQ_{3n}, \{X^{n-3}000 j_{n-1}..j_0 k_{n-1}..k_0, \quad X^{n-3} y_2 y_1 0 j_{n-1}..j_0 k_{n-1}..k_0\})$

$\sigma(AQ_{3n}, \{X^{n-3}010 j_{n-1}..j_0 k_{n-1}..k_0, \quad X^{n-3} y'_2 y'_1 0 j_{n-1}..j_0 k_{n-1}..k_0\})$

$\sigma(AQ_{3n}, \{X^{n-3}001 j_{n-1}..j_0 k_{n-1}..k_0, \quad X^{n-3} y_2 y_1 1 j_{n-1}..j_0 k_{n-1}..k_0\})$

$\sigma(AQ_{3n}, \{X^{n-3}011 j_{n-1}..j_0 k_{n-1}..k_0, \quad X^{n-3} y'_2 y'_1 1 j_{n-1}..j_0 k_{n-1}..k_0\})$

where

if $\pi(y_{n-1}..y_0 k_{n-1}..k_0)=0$ then

$y_2 y_1 = 11, \quad y'_2 y'_1 = 10$

else

$y_2 y_1 = 10, \quad y'_2 y'_1 = 11$

The broadcast along the i-axis will take place by first sending the data from the originating node to the corresponding nodes, in each of these 4 subcubes, which takes at most 2 exchange cycles, and then the remaining broadcasting will proceed within each subcube, requiring an additional $\lfloor \frac{n-2}{2} \rfloor + 1 = \lfloor \frac{n}{2} \rfloor$ exchange cycles. The total time for broadcasting along the i-axis is therefore

$\lfloor \frac{n}{2} \rfloor + 2$ exchange cycles.

(2) n is even:

The subgraph $\sigma(AQ_{3n}, \{i_{n-1}..i_0 j_{n-1}..j_0 X^n\})$ will be splitted into 2 (alternately-twisted) subcubes first: $\sigma(AQ_{3n}, \{i_{n-1}..i_0 j_{n-1}..j_0 0 X^{n-1}\})$ and $\sigma(AQ_{3n}, \{i_{n-1}..i_0 j_{n-1}..j_0 1 X^{n-1}\})$, for fixed i and j. The number of exchange cycles required for broadcasting along the k-axis is therefore $2 + \lfloor \frac{n-1}{2} \rfloor + 1 = \frac{n}{2} + 2$.

Similarly, with fixed i and k, the subgraph $\sigma(AQ_{3n}, \{i_{n-1}..i_0 X^n k_{n-1}..k_0\})$ is divided into 8 (alternately-twisted) subcubes as:

$\sigma(AQ_{3n}, \{i_{n-1}..i_0 w X^{n-4} 00 z k_{n-1}..k_0,\ i_{n-1}..i_0 w X^{n-4} y_2 y_1 z k_{n-1}..k_0\})$

or

$\sigma(AQ_{3n}, \{i_{n-1}..i_0 w X^{n-4} 01 z k_{n-1}..k_0,\ i_{n-1}..i_0 w X^{n-4} y'_2 y'_1 z k_{n-1}..k_0\})$

for the 8 combinations of bits w and z, and

if $\pi(k_{n-1}..k_0)=0$ then

$y_2 y_1 = 11, \quad y'_2 y'_1 = 10$

else

$y_2 y_1 = 10, \quad y'_2 y'_1 = 11$

Broadcasting along the j-axis therefore requires $4 + \lfloor \frac{n-4}{2} \rfloor + 1 = \frac{n}{2} + 3$ exchange cycles.

The scheme for broadcasting along the i-axis is the same as that in the case for n being odd, and takes $\frac{n}{2} + 2$ exchange cycles.

To summarize, the total time for the broadcasts in stage (i) of the algorithm takes $3\lfloor \frac{n}{2} \rfloor + 6$ exchange cycles if n is odd, and $3(\frac{n}{2}) + 7$ exchange cycles if n is even.

Stage (ii) of the algorithm is an independent computation within each node, and requires no communication among the nodes.

Stage (iii) needs to do a combining operation (the SUM) over all the elements with the same i-coordinates. The way of the combining is similar to that we do for broadcasting along the i-axis. This time, the combining takes place first in the 4 (alternately-twisted) subcubes of the induced subgraph $\sigma(AQ_{3n}, \{X^n j_{n-1} \cdots j_0 k_{n-1} \cdots k_0\})$ simultaneously for each fixed j and k. The results are then combined and stored at the node (0,j,k). The total communication time for stage (iii) is therefore (n-2)+2 = n exchange cycles.

As a result, we get

Theorem 4.2 The multiplication of two $2^n \times 2^n$ matrices can be accomplished on an AQ_{3n} in O(n) time. In particular, the total time spent on data communication is

$3\lfloor \frac{n}{2} \rfloor + n + 6$ exchange cycles if n is odd

and is $\frac{5n}{2} + 7$ exchange cycles if n is even.

It should be noted that the same algorithm applied on a binary 3n-cube requires totally 4n exchange cycles for data communication. The alternately-twisted cube is therefore superior to the hypercube in solving this type of problem.

Gaussian Elimination

The major component of the Gaussian Elimination in solving a system of equation Ay=b is the decomposition of the matrix A into an upper and lower matrix U and L respectively. We will concentrate on how it is to be done on an alternately-twisted cube and the required communication time.

Suppose A is an N by N matrix, and N = 2^n. Each element A_{ij} is initially stored at the local storage a(u) of node u in an AQ_{2n}, where u = $i \cdot 2^n + j$, for all 0≤ i, j ≤ N-1 and a(u) is also denoted as a(i,j). The LU decomposition is carried out in parallel on the AQ_{2n} as follows:

```
    For k=0 to N-1
        do in parallel for i : k+1≤i≤N-1
(1)         a(i,k) = a(i,k) / a(k,k)
            do in parallel for j : k+1≤j≤N-1
(2)             a(i,j) = a(i,j) - a(k,j) * a(i,k)
            enddo
        enddo
    endfor
```

The results, elements of L and of U, are stored in a(i,j) for all i>j, and in a(i,j) for all i≤j respectively. In each iteration of the for-loop, elements of row k, a(k,j), k≤j≤N-1, are broadcast to the respective elements of each row below it. We call this the vertical broadcast. Then statement (1) is executed at each row i>=k+1 in parallel and the result is broadcast to all the elements on the same row to the right of node (i,k). We call this the horizontal broadcast. Then statement (2) is executed at each relevant node simultaneously.

By the way of mapping the coordinate (i,j) to the nodes in AQ_{2n}, each node address is divided into 2 parts, as $i_{n-1}i_{n-2}\cdots i_0 j_{n-1}j_{n-2}\cdots j_0$. By similar analysis as in the case for matrix multiplication, we get:

if n is odd, then each vertical broadcast requires $\frac{n-1}{2}+2$ exchange cycles, and each

horizontal broadcast takes $\lfloor \frac{n}{2} \rfloor + 1$ exchange cycles, and

if n is even, then the vertical broadcast and horizontal broadcast needs $2+\frac{n-2}{2}+1$

$=\frac{n}{2}+2$ and $2+\lfloor \frac{n-1}{2} \rfloor + 1 = \frac{n}{2}+2$ exchange cycles respectively.

Therefore the time complexity of the algorithm can be stated as follows:

<u>Theorem</u> <u>4.3</u> The LU decomposition of a $2^n \times 2^n$ matrix on an AQ_{2n} can be done in $O(n \cdot 2^n)$ time and specifically, the total communication time occupies

$2^n \cdot \left(\frac{n-1}{2} + \lfloor \frac{n}{2} \rfloor + 3\right)$ exchange cycles if n is odd

and $2^n \cdot (n+4)$ exchange cycles if n is even.

Note that the corresponding algorithm applied on a binary 2n-cube needs a total of $2^n \cdot (2n)$ exchange cycles for data communication. Hence once again the alternately-twisted cube surpasses it.

Solving Partial Differential Equations (PDEs)

An important class of PDEs is Poisson's equation [Akl89, p.212]

$$\frac{\partial^2 u(x,y)}{\partial x^2}+\frac{\partial^2 u(x,y)}{\partial y^2}=G(x,y)$$

where u(x,y) is the unknown functions and G the given function, both in two independent variables. Values of u(x,y) are calculated by the difference equation

$$u(x,y)=\frac{1}{4}(u(x+d,y)+u(x-d,y)+u(x,y+d)+u(x,y-d)-d^2G(x,y))$$

where d is the interval between the nodal points corresponding to u(x,y) in a 2-D space. Suppose the space is divided into (n+1) by (n+1) grid points. Parallel technique known as successive overrelaxation (SOR) [Akl89,p.212] is then used to approximate the value of u(x,y) at each of the $(n-1)^2$ interior points. The method executes iterations during each of which the value of an interior nodal point is updated from those values at its north, east, south, and west neighbours in the space. Clearly this exhibits a grid-like communication pattern. And by the result in Section 2.2.5.2, we know that the alternately-twisted 2n-cube is able to support the SOR method over $2^n \times 2^n$ nodal points in solving the Poisson's equation, such that each iteration of the method takes only one exchange cycle for data communication. It is exactly the same time complexity for the method applied on the binary 2n-cube. The 2 cubes weigh equally in solving this type of problem.

Certainly there are many more algorithms we have not touched. But the little investigation in this section already shows that the alternately-twisted cube is flexible enough to support efficiently the simplest and probably also the most common ones. At least its performance is as good as (or even slightly better than) the hypercube.

Chapter 5

Summary, Comparison & Conclusion

5.1. Summary

A new network topology called the alternately-twisted cube is proposed. It is based on a modification to the topology of the binary n-cube, or hypercube, by "twisting" its edges along the odd-numbered dimensions. The twisting is selectively applied on pairs of edges along these dimensions, according to the T-code sequence we defined.

An alternately-twisted n-cube, denoted as AQ_n, has the same number of nodes as that of the binary n-cube, i.e. 2^n, as well as the same node degree and link count, which are respectively n and $n \cdot 2^{n-1}$. However, because of the effect of the edge-twisting, an alternately-twisted n-cube has a diameter of only $\lfloor \frac{n}{2} \rfloor + 1$, which is nearly half of that of the binary n-cube.

Many salient features of the binary n-cube are preserved by the AQ_n. These include the node-symmetry property, the existence of n distinct paths between any 2 nodes, and the ability to embed any HxW grids (with dilation 1 if H and W are powers of 2, and dilation 2 otherwise).

An AQ_n can be partitioned into smaller, disjoint alternately-twisted subcubes. The number of different ways to achieve this, however, is generally smaller than that in the hypercube. The reason is that the former is not edge-symmetric, restricting the freedom of choice for the partitioning.

Both the AQ_n and the binary n-cube can embed a complete binary tree of size $2^n - 1$ with dilation 2, while the overloading factor, or edge congestion, of the embedding is 1 in the latter and is 2 in the former. On the other hand, in AQ_n, we can embed any ring structure of size k, for $k \leq 2^n$ and $k \neq 3$, with dilation 1, but a binary n-cube can do this for rings of even length only.

The AQ_n appears to be more attractive than the binary n-cube as a general purpose interconnection network. We have devised a distributed, shortest-path routing algorithm for the AQ_n network. Analytic results show that in general it can route messages faster than the hypercube: about 22% smaller in the mean internode distance, nearly 50% smaller in the diameter measure, and nearly 30% shorter in the average message delay under heavy load, when the network size is large. The improvement is better when the dimension of the AQ_n is an odd number, than when it is even. Broadcasting on the AQ_n, under the multiple-message accepting mode, takes only $\lfloor \frac{n}{2} \rfloor + 1$ routing cycles, again about 50% of that on the binary n-cube.

The AQ_n is also able to support the following parallel algorithms at least as efficiently as the hypercube: the Ascend/Descend class of algorithms, the combining class of algorithms, and the algorithms for solving Poisson-type partial differential equations, matrix multiplication, and Gaussian elimination. In the last two algorithms, since broadcasting is used extensively, the AQ_n behaves even better than the hypercube, as it takes less communication time for their executions.

5.2. Comparison with other hypercube-like networks

Tzeng has proposed a Variant Hypercube topology [Tzen90] as an extension to the binary hypercube. The idea is to add extra edges in a binary n-cube, such that pair of nodes are connected if the most significant k bits in their addresses are equal, and the last (n-k) bits in one address are complements of those in the other, where k is a parameter of the network topology. Tzeng shows that, for the performance of the network to be optimal or near-optimal, k is chosen to be 0 if n is even, and 1 if n is odd. With this in mind, the diameter of the network is $\lceil \frac{n}{2} \rceil$. Hence we see that the variant hypercube weighs nearly the same as the AQ_n in this measure (as well as the measure of the broadcasting time). The cost of the variant hypercube, however, is larger, because each node has (n+1) linkages instead of n as in the AQ_n. Moreover, the mean internode distance improvement in the variant hypercube of dimension n, over that in the binary n-cube, is found to be 17%, 17%, 14% and 13% respectively for n=5, 10, 15, and 20. The respective improvements of the AQ_n over the hypercube in this measure are 12.5%, 15%, 20% and 19.5%. Hence, while the variant hypercube can route messages slightly faster than the AQ_n for small network size, the latter behaves better in the case of large network size (say, for $n \geq 15$). If the cost of the extra links in the variant hypercube is taken into account, the breakeven point for the performance/cost ratio of the two networks will be even smaller: the alternately-twisted cube surpasses the variant hypercube when the dimension of the cube, n, exceeds 10.

A twisted n-cube network, denoted as TQ_n, has been investigated by Esfahanian *et al.* [Esfa88] [Esfa91]. It is a modification to the binary n-cube, with exactly one pair of its edges twisted (say, the pair (00u --> 10u, 10u --> 11u) is replaced by the pair (00u --> 11u, 10u --> 01u), where $u = 0^{n-2}$). Therefore the node degree and link count in TQ_n is the same as that of the AQ_n. However, the number of twisted edges in the AQ_n is generally larger than that in the TQ_n. For instance, in an AQ_5 there are 8 pairs of twisted edges, while in a TQ_5 there is only 1 pair. This makes it more likely for the former to have shorter paths between most pairs of nodes than the corresponding paths in the latter. Also, the single twisted-edge pair can only reduce the diameter of the TQ_n by 1, to (n-1), for a reduction ratio of $\frac{1}{n}$ relative to that of the hypercube. This is generally poorer than the 50% reduction achieved by the AQ_n topology. Hence it is not likely that the network performance of the TQ_n would be better than the AQ_n. On the other hand, the TQ_n has at least one advantage over the AQ_n, i.e. it can embed a complete binary tree of $2^n - 1$ nodes with dilation 1 only, while we know of no way to embed the same tree with dilation 1 into the AQ_n.

Independent research by Efe [Efe89] has resulted in another way of twisting the binary n-cube. It is known as the multiply-twisted n-cube, denoted as MQ_n, and is quite similar to the alternately-twisted n-cube. Its definition has already been described in Chapter 1. The MQ_n has the same node degree and link count as the AQ_n. The difference between them is that the definition of the MQ_n requires the edges along all dimensions of a binary n-cube to be twisted, while that of the AQ_n just twists the edges along odd-numbered dimensions. Because of this, the former is edge-symmetric but the latter is not. However, both can attain exactly the same amount of reduction in the diameter measure as compared to the binary n-cube, giving a diameter of $\lfloor \frac{n}{2} \rfloor + 1$ for both twisted cubes.

In addition AQ_n and MQ_n have the same fault-tolerance capability because in each there are exactly n distinct paths between any pair of nodes. Subcube partitioning can be more flexible in the MQ_n, however, because of its edge-symmetry.

Figure 5.1 shows the results of the analysis of the mean internode distance of the AQ_n and MQ_n. It can be seen that they are actually quite close for the wide range of network sizes shown, with those in the MQ_n being slightly smaller than those in the AQ_n : no more than 4% for the difference (to be more specific, about 1.5% when n is odd, and below 4% when n is even). However, broadcasting on both twisted cubes takes the same amount of time because their diameters are always equal.

Efe did not include any graph-embedding analysis of the MQ_n in his paper [Efe89], except that a Hamiltonian cycle can always be found in an MQ_n. Therefore no comparison can be made between the capability of the MQ_n and the AQ_n in this aspect. However, as noted in [Efe89], the MQ_n can, in general, execute any parallel algorithm with the same order of time complexity as that needed on a binary n-cube; this is exactly the same for the AQ_n, as we have shown in Chapter 4. Therefore we can conclude that both the multiply-twisted cube and the alternately-twisted cube are equally suited for supporting parallel processing and are comparable to the hypercube.

Moreover, at the time of this writing, analysis results on the network performance of the multiply-twisted n-cube is not available to the author. Therefore we cannot make a thorough comparison here.

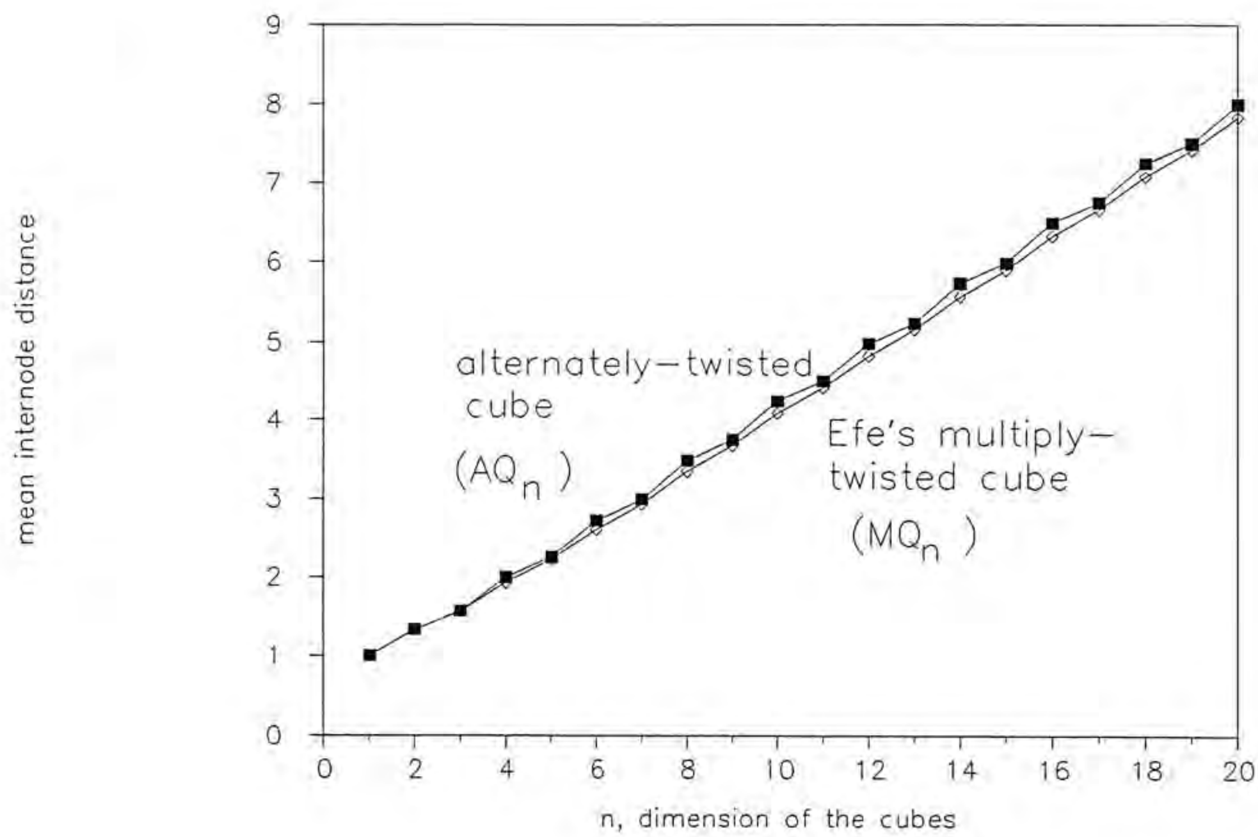


Figure 5.1: Mean internode distance comparison - AQ_n vs MQ_n

On the other hand, programming the AQ_n, or mapping processes in a parallel application onto it, seems easier than programming the MQ_n. It is because, in the AQ_n, adjacent nodes can have at most 2 (consecutive) differing bits in their addresses, while the connectivity rule of the MQ_n may allow as many as $\frac{n}{2}$ differing bits in the addresses of two neighbouring nodes. (For example, node 010101 is adjacent to node 111111 in a MQ_6.) In other words, we can easily identify an alternately-twisted subcube structure of an AQ_n by confining the change of address bits (arising from the adjacency within the subcube structure) to be within certain pairs of bits in the node address. This is certainly not the case for the MQ_n. Such localized change of bit patterns enable the programmer to more easily determine the adjacency, as well as the distance, between any 2 nodes in an AQ_n.

5.3. Conclusion

We have demonstrated the significance of systematic edge-twisting in the hypercube network: the alternately-twisted cube is shown to be an attractive alternative topology to the hypercube for interconnecting multiprocessors or multicomputers in a general purpose parallel processing environment. Moreover, since the AQ_n and the MQ_n have very similar topologies, one can also consider the analysis results obtained here as supplementing Efe's work reported in [Efe89].

5.4. Possible future research

At last, we would like to pose the following problem as a possible future direction of the research:

It is known that the k-dimensional mesh is a generalization of the hypercube. Therefore it is interesting to ask: Is there any way to generalize the idea of twisting a hypercube to obtain a twisted mesh structure? And, if so, how does it compare to the regular mesh structure? Positive solutions to these questions may be valuable because recent research shows that a low-dimensional mesh network can be more efficient than a high-dimensional hypercube [Dall87, section 5.3.1].

Bibliography

[AbPa89] S. Abraham and K. Padmanabhan, "Performance of the Direct Binary n-Cube Network for Multiprocessors," *IEEE Transactions on Computers*, July 1989, p.1000-1011.

[Akl89] S.G. Akl, *The Design and Analysis of Parallel Algorithms*, Prentice-Hall, 1989.

[Chan88] M.Y. Chan, "Dilation-2 Embeddings of Grids into Hypercubes," *Proceedings of 1988 International Conference on Parallel Processing*, The Pennsylvania State University Press, Aug 1988, p.295-298.

[Dall87] W.J. Dally, *A VLSI Architecture for Concurrent Data Structures*, Kluwer Academic Publishers, 1987.

[Efe89] K. Efe, "Programming the Twisted-Cube Architectures," *1989 Proceedings of International Conference on Distributed Computing Systems*, IEEE, 1989, p.254-262.

[Esfa88] A. Esfahanian, L. Ni and B. Sagan, "On enhancing hypercube multiprocessors," *Proceedings of 1988 International Conference on Parallel Processing*, IEEE, 1988, p.86-89.

[Esfa91] A. Esfahanian, L. Ni and B. Sagan, "The Twisted N-Cube with Application to Multiprocessing," *IEEE Transactions on Computers*, Jan. 1991, p.88-93.

[Hara71] F. Harary, *Graph Theory*, Addison-Wesley, 1971.

[JoHo89] S.L. Johnsson and C.T. Ho, "Optimal Broadcasting and Personalized Communication in Hypercubes," *IEEE Transactions on Computers*, Sept. 1989, p.1249-1268.

[Kung89] H.T. Kung, "Network-Based Multicomputers: Redefining High Performance Computing in the 1990s," *Advanced Research in VLSI: Proceedings of the 1989 Decennial Caltech Conference*, MIT Press, 1989, p.49-66.

[LaDh90] S. Lakshmivarahan and S.K. Dhall, *Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problems*, McGraw-Hill, 1990.

[PrVu81] F.P. Preparata and J. Vuillemin, "The Cube-Connected Cycles: A Versatile Network for Parallel Computation," *Communications of ACM*, 1981, p.300-309.

[SaSc88] Y. Sadd and M.H. Schultz, "Topological Properties of Hypercubes," *IEEE Transactions on Computers*, July 1988, p.867-872.

[Seit85] C. Seitz, "The Cosmic Cube," *Communications of ACM*, 1985, p.22-33.

[Tzen90] N.F. Tzeng, "Analysis of a Variant Hypercube Topology," *1990 International Conference on Supercomputing*, 1990, ACM, p.60-70.