Properties and Communication Algorithms *for* *k*-ary *n*-cube Interconnection Networks

by

Yaagoub A. Ashir

A Research Paper Submitted to Partially Fulfill the Requirement for the Degree of

Master of Science

Committee Members :

Prof. Bella Bose
Prof. Vikram Saletore
Prof. Toshi Minoura

Department of Computer Science
Oregon State University

January 15, 1993

Properties and Communication Algorithms *for* *k*-ary *n*-cube Interconnection Networks

by

Yaagoub A. Ashir

A Research Paper Submitted to Partially Fulfill the Requirement for the Degree of

Master of Science

Committee Members :

Prof. Bella Bose
Prof. Vikram Saletore
Prof. Toshi Minoura

Department of Computer Science
Oregon State University

January 15, 1993

Acknowledgment

My deepest gratitude and sincere appreciation go to Professor Bella Bose, my major professor, for his invaluable advice, encouragement, and his unending patience. I would like to thank Dr. Vikram Saletore and Dr. Toshi Minoura for serving on my committee and for their instructive directions. Thanks are extended to include many of the department faculty members for providing me with excellent education.

I am also grateful to the University of Bahrain for supporting my studies. Many thanks for my friends here in Corvallis who contribute to the nice atmosphere. Finally, special thanks to my family for their unlimited support and encouragement. In particular, to my brother Fawzi for his continuous help.

Abstract

The *k*-ary *n*-cube structure is presented in this paper for interconnecting a network of microcomputers in parallel and distributed environments. Machines based on the *k*-ary *n*-cube topology have been advocated as ideal parallel architectures for their powerful interconnecting features.

In this paper, we examine the *k*-ary *n*-cube from the graph theory point of view and consider those features that make its connectivity so attractive. Among other things, we propose several effective global data communication algorithms on the *k*-ary *n*-cube interconnection network.

Table of Contents

Chapter 1

Introduction

1.1 Parallel Computers

Parallel computers in general are classified into two groups: *multiprocessors with shared memory* organization and *multicomputers with non-shared* or *distributed memory* organization [LaD90, BeT89, SaS86]. There are also a variety of hybrid designs lying in between. The first type uses a global sharing memory that can be accessed by all processors (see Figure 1.1.1). To allow efficient access of the memory by several processors, the memory is divided into several memory banks. A processor can communicate with another by writing into the global memory and having the second processor read the same location in the memory using *switching systems*. The advantage of this architecture is that the algorithm design is simple. Moreover, it enables us to make the data access transparent to the user who may regard data as being held in a large memory which is readily accessible to any processor. However, as the number of nodes increases, the switching network becomes complex to build. Also, the decision of shutting down failing nodes and choosing alternate routes is local. The GF-11 Supercomputer, the Butterfly multiprocessor, and the Ultracomputer are some examples of this type of architecture.

In the second important type of parallel processors, there is no shared memory and no global synchronization, but rather each processor has its own *local memory*. Processors communicate through *interconnection network* consisting of direct communication links joining certain pairs of processors, as shown in Figure 1.1.2. Which processors are connected together is an important design choice. It would be best if all processors were directly linked to each other which leads to increased cost, or the processors communicate through a shared bus, which leads to excessive delays when the number of processors is very large, due to the necessary bus contention. Moreover, interconnection achieved by *message-passing* directly or through some

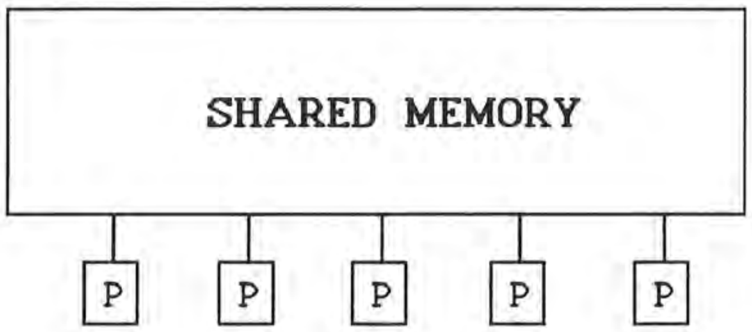


Figure 1.1.1 The Shared Memory Model

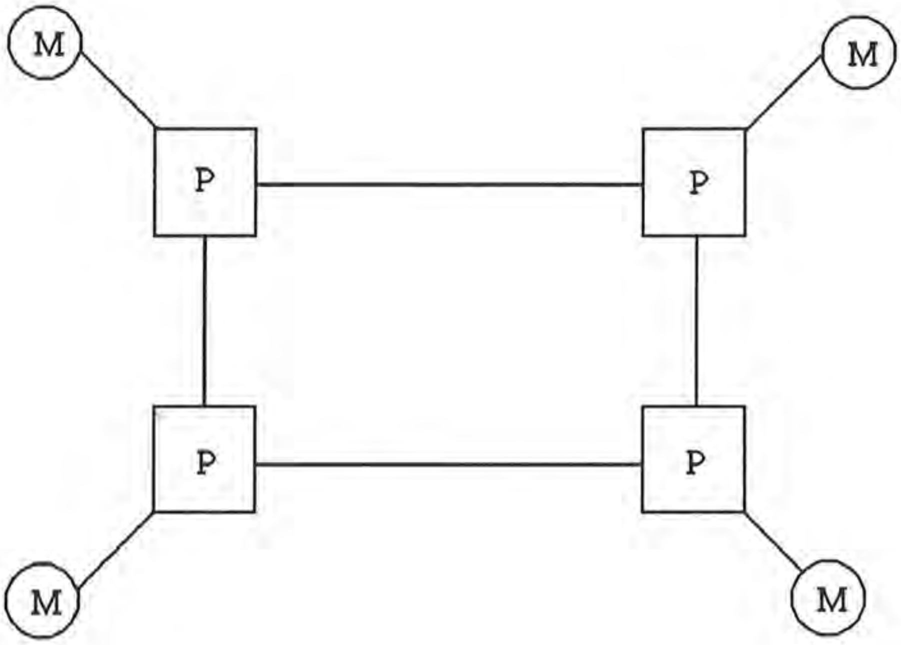


Figure 1.1.2 The Distributed Memory Model

intermediate processors, and computation is data driven. The main advantage of such architectures is the simplicity of their design. The nodes are identical, or are of a few different kinds and can therefore be fabricated at relatively low cost. Moreover, these models can easily be made fault tolerant by shutting down failing nodes. Examples of this type of organization include the Cosmic Cube [Sei85], Intel's hypercube, NCUBE Machine, and the Connection Machine.

1.2 Why *k*-ary *n*-cubes?

Hypercubes are loosely coupled parallel processors based on the binary *n*-cube network and introduced under different names (cosmic cube, *n*-cube, binary *n*-cube, Boolean *n*-cube, etc.). An *n*-cube parallel processor consists of 2^n identical processors, each provided with its own sizable memory and interconnected with *n* neighbors (see Figure 1.2.1). The hypercube gains its popularity due to the fact that it has some attractive features like symmetries, high level of concurrency and efficiency, regularity and high potential for the parallel execution of various algorithms. Moreover, most other networks can be directly mapped into a hypercube.

However, one drawback to the hypercube is that the number of connections to each processor grows logarithmically with the size of the network [Lei92]. While this is not a problem for small hypercubes, it can present some difficulties for very large machines (e.g., machines with tens of thousands of processors). VLSI systems are wire-limited. Although hypercubes can provide small diameter, the property of high dimension is not consistent with the properties of VLSI technology. Networks with many dimensions require more and longer wires than do low-dimensional networks. Thus high-dimensional networks cost more and run more slowly than low-dimensional networks. It is shown that low-dimensional networks achieve lower latency and better hot-spot throughput than do high-dimensional networks [Dal90, LiH91].

The binary *n*-cube is a special case of the family of *k*-ary *n*-cubes, cubes with *n* dimensions and *k* nodes in each dimension. In order to overcome the problem of high dimensionality of hypercubes, we can increase *k* and decrease *n* obtaining low-dimensional *k*-ary *n*-cube. For

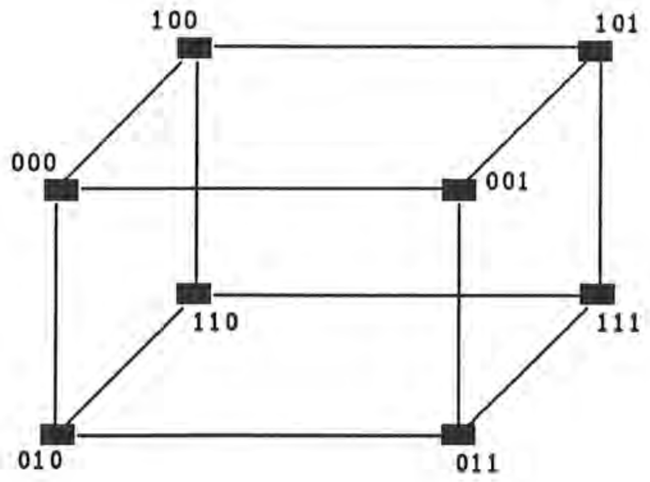


Figure 1.2.1 3-Dimensional Hypercube

example, the 4096 processors in a binary 12-cube with a total of 24576 links can be interconnected using 16-ary 3-cube model with a total of only 12288 links.

In this paper we introduce the k-ary n-cube model ($k>2$). We propose the recursive structure and some of the topological properties of this model in Chapter 2. To extend the strategy of the binary reflected Gray codes, we introduce in Chapter 2 a class of generalized Gray codes called *k-ary reflected Gray codes*. Some communication algorithms like *one-to-one, single node broadcasting, multinode broadcasting, single node scattering, and total exchange* are proposed in Chapter 3. We conclude this paper by summarizing the results and stating the future research in Chapter 4.

Chapter 2

The *k*-ary *n*-cube Network

In this chapter, we define the *k*-ary *n*-cube and show its properties. The binary *n*-cube has been extensively studied (see [Lei92, BeT89, SaS88, BhA84] for references), so we restrict ourselves on the *k*-ary *n*-cube where $k > 2$. We begin this chapter with some definitions and describing the recursive structure in section 2.1. We propose the *k*-ary *n*-cube's topological properties in section 2.2. In section 2.3, we show that the *k*-ary *n*-cube is Hamiltonian by introducing a general type of the reflected Gray codes called *k-ary reflected Gray codes*.

2.1 Definitions and Structures

In order to be able to define the *k*-ary *n*-cube network, we begin this section by introducing some definitions from coding theory [PeW72]. Then, we define the *k*-ary *n*-cube network and show that it can be built recursively from lower dimensional cubes.

Definition 2.1.1

Let $<k> = \{0, 1, 2, ..., k\text{-}1\}$. Let $A = (a_n, a_{n-1}, ..., a_1)$ be an *n*-tuple where $a_i \in <k>$. The *Hamming weight* of a vector *A*, denoted $W_H(A)$, is defined to be the number of nonzero components. The *Hamming distance* between two vectors *A* and *B*, denoted $D_H(A,B)$ is equal to $W_H(A - B)$. In other words, the Hamming distance between *A* and *B* is the number of positions in which they differ.

Definition 2.1.2

Let $<k> = \{0, 1, 2, ..., k\text{-}1\}$. Let $A = (a_n, a_{n-1}, ..., a_1)$ be an *n*-tuple where $a_i \in <k>$. The *Lee weight* of a vector *A* is defined as

$$W_L(A) = \sum_{i=1}^{n} | a_i |$$

where

$$|a_i| = a_i, \qquad 0 \le a_i \le \frac{k}{2},$$

$$= k - a_i \quad , \qquad \frac{k}{2} < a_i \le k - 1.$$

The *Lee distance* between two vectors A and B, denoted $D_L(A, B)$, is $W_L(A - B)$, the Lee weight of their difference.

Clearly for $k=2$ and 3, the Lee and Hamming distances are equal. For $k>3$ the Lee distance between two n-tuple vectors is greater than or equal to the Hamming distance between them.

Example 2.1.1

Let $k=5$ and $n=6$. Let $A = (3\ 0\ 1\ 2\ 3\ 4)$ and $B = (3\ 0\ 4\ 0\ 0\ 0)$. Then their difference term by term modulo 5 is

$A - B = (0\ 0\ 2\ 2\ 3\ 4)$, and

$D_H(A,B) = W_H(A-B) = 4$

$D_L(A,B) = W_L(A-B) = 0 + 0 + 2 + 2 + 2 + 1 = 7.$

Definition 2.1.3

The k-ary n-cube network model is a cube of n dimensions radix k. A node in the k-ary n-cube can be identified by n-digit radix k address, $(a_n, a_{n-1}, \ldots, a_1)$ where $a_i \in \langle k \rangle$. Two nodes A and B in the k-ary n-cube are adjacent if and only if $D_L(A,B) = 1$. The i^{th} digit of the address a_i represents the node's position in the i^{th} dimension. The dimension, n, the radix, k, and the number of nodes, N, have the following relation

$$N = k^n,\ k = \sqrt[n]{N},\ n = \log_k N .$$

Definition 2.1.4

In a k-ary n-cube network, two nodes are said to be *opposite in direction i* if their addresses differ in position i and the Lee distance between them is unity.

The union of all the opposite nodes in direction i from a *ring in direction i.*

Proposition 2.1.1

The k-ary n-cube network can be constructed recursively from lower dimensional cubes.

Proof

Let $N = k^n$, $k > 2$, $n \geq 1$. Let $<k> = \{0, 1, 2, ..., k - 1\}$. An N-node k-ary n-cube is a Graph $G = (V,E)$ where

$$V = \{a \mid a \text{ is an } n\text{-digit radix } k \text{ integer, i.e. } a = a_n a_{n-1} \ldots a_1 \text{ and } a_i \in <k>\},$$

and

$$E = \{ (a,b) \mid a, b \in V \text{ and } D_L(a,b) = 1\}$$

if A is a set of strings over $<k>$, then define

$$xA = \{ x\alpha \mid \alpha \in A \text{ and } x \in <k> \}$$

where $x\,\alpha$ is a string obtained by concatenating x and α.

Let

$$G_k^{(n)} = (V_k^{(n)}, E_k^{(n)})$$

be the graph of the k-ary n-cube, then $G_k^{(n)}$ can be constructed recursively as follows

$$V_k^{(1)} = <k>, \text{ and}$$
$$E_k^{(1)} = \{(x,y) \mid x,y \in <k>, x<y, \text{ and } D_L(x,y)=1\};$$
$$V_k^{(n)} = xV_k^{(n-1)} \quad \text{for all } x \in <k>$$

and

$$E_k^{(n)} = \{ (s\alpha, t\alpha) \mid \alpha \in V_k^{(n-1)}, s, t \in <k>, s<t \text{ and } D_L(s,t) = 1\}. \qquad \square$$

Stated in words, we can construct a k-ary n-cube network recursively by making k copies of the k-ary $(n-1)$-cube, renumber the labels by concatenating digit 0 to each label of the first copy, digit 1 to each label of the second copy, ..., and digit k-1 to each label of the k^{th} copy. Finally, link the nodes that are opposite in direction n. Figure 2.1.1 illustrates an example of constructing a 4-ary 3-cube recursively starting from 4-ary 1-cube.

2.2 Topological Properties

When designing a large multiprocessor, one of the most important design decisions involves the topology of the communication structure among the processors. The degree (number of incident links) of each node, the total number of links, and the diameter of the network should be known before choosing the network. In this section, we introduce the topological properties of the k-ary n-cube $(k > 2)$ hoping to be a powerful network.

Proposition 2.2.1

In a k-ary n-cube network

a) The degree of each node is $2n$.

b) The total number of links is nk^n.

Proof

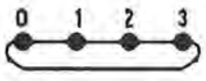
a) Let $A = (a_n, a_{n-1}, ..., a_1)$ be a node in a k-ary n-cube. Then both $(a_n, a_{n-1}, ..., (a_1+1) \bmod k)$ and $(a_n, a_{n-1}, ..., (a_1-1) \bmod k)$ are adjacent to A in direction one. Similarly we can prove that in each direction, there are two adjacent nodes to A, resulting in a total of $2n$ adjacent nodes.

b) Summing the number of links incident to each node, we get $2nk^n$. Since each link will be counted twice, we get the total number of links is

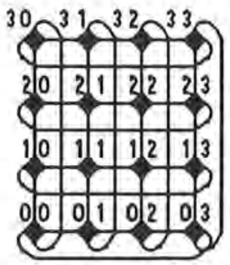
$$2nk^n/2 = nk^n \text{ links.} \qquad \square$$

We notice that the above property does not hold if $k=2$ (in a binary hypercube) because $(a_1+1) \bmod 2$ is equal to $(a_1-1) \bmod 2$ resulting in n links incident to each node, and total links of

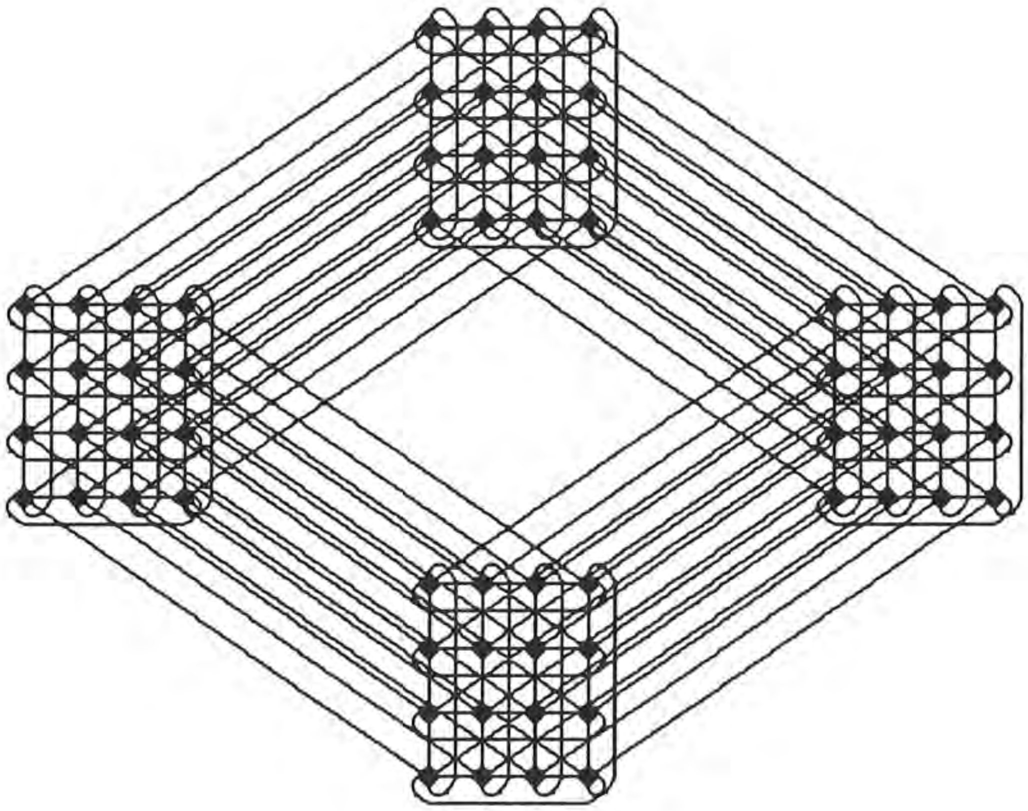
$$n2^n/2 = n2^{n-1} .$$



(a)



(b)



(c)

Figure 2.1.1

Recursive structure of 4-ary 3-cube (a) 4-ary 1-cube, (b) 4-ary 2-cube, (c) 4-ary 3-cube.

Define the *diameter* of the network to be the maximum Lee distance between any two nodes in the network. We can state the following proposition.

Proposition 2.2.2

The diameter of the k-ary n-cube network is

$$\lfloor k/2 \rfloor * n.$$

Proof

Let A and B be two nodes in the k-ary n-cube network. Then from the definition of the Lee distance, we have

$$D_L(A,B) = W_L(A\text{-}B) = \sum_{i=1}^{n} | a_i | .$$

However, $| a_i | \leq \lfloor k/2 \rfloor$

This results in

$$D_L(A,B) \leq \lfloor k/2 \rfloor * n \qquad \square$$

Proposition 2.2.3

In a k-ary n-cube network, there are k^{n-1} node disjoint k-rings in each direction.

Proof

Let the label of a node be $a_n a_{n-1} \dots a_1$ n-tuples radix k. Fix $a_1 = (0, 1, 2, \dots, k\text{-}1)$ to be a ring in direction one. Then from the remaining $n\text{-}1$ radix k tuples, we have a total combination of k^{n-1} concatenated to a_1 resulting in k^{n-1} node disjoint k-rings in direction one. Similarly, we can prove for direction two to n. $\square$

Example 2.2.1

In a 4-ary 2-cube there are 4 rings in direction one and 4 rings in the second direction.

First direction	**Second direction**
$R_1 = \{00, 01, 02, 03\}$	$R_1 = \{00, 10, 20, 30\}$
$R_2 = \{10, 11, 12, 13\}$	$R_2 = \{01, 11, 21, 31\}$
$R_3 = \{20, 21, 22, 23\}$	$R_3 = \{02, 12, 22, 32\}$
$R_4 = \{30, 31, 32, 33\}$	$R_4 = \{03, 13, 23, 33\}$

Obviously, the rings in each direction are node disjoint 4-rings.

We would like to know if there exist different paths between any two nodes A and B. The existence of such paths might be useful for speeding up transfers of large amounts of data between two nodes (see section 3.2). It also provides a way of selecting alternative routes in case a given node in a path is failing [BhA84]. In order for this to be possible, the paths must not have common nodes, except for nodes A and B. We will refer to such paths as *node-disjoint parallel paths*, and we will prove in the following proposition that there are $2n$ node-disjoint parallel paths between any two nodes in the k-ary n-cube.

Proposition 2.2.4

Let $A = (a_n, a_{n-1}, ..., a_1)$, $B = (b_n, b_{n-1}, ..., b_1)$.

Let $l = D_L(A, B)$, $h = D_H(A, B)$ and $w_i = D_L(a_i, b_i)$. Then, in a k-ary n-cube, there are a total of $2n$ node-disjoint parallel paths between A and B of which

(i) h paths are of length l,

(ii) $2(n-h)$ paths are of length $l+2$, and

(iii) for each i, $w_i > 0$, there is a path of length

$$l + k - 2w_i ,$$

(a total of h paths).

Proof

Without loss of generality, we can assume that the first h digits of the labels of A and B are different, and the remaining $n-h$ digits are the same.

(i) Let $1 \leq i \leq h$, the i^{th} path is constructed as follows: start from the label of A; correct sequentially digit i, using the shortest path of the ring in direction i between a_i and b_i, through digit h, then digit 1 through digit i-1. This will result in h paths each of length l.

(ii) We construct next the $2(n-h)$ paths from A to B having l+2 links each as follows: for each j, $h < j \leq n$, add 1 to digit j, correct digits 1 through h, using the shortest path in each ring, then add -1 to digit j. This will result in a total of n-h paths of length $l+2$. We can construct the remaining n-h paths of length $l+2$ by adding -1 to digit j, correct digits 1 through h, using the shortest path in each ring, then add 1 to digit j.

(iii) The remaining h paths are constructed as follows: for each i, $1 \leq i \leq h$, move digit i one position toward the longest path in the ring in direction i (by adding or subtracting 1), correct digits $i+1$ through h and digits 1 through $i-1$ using the shortest path in each ring, then continue in correcting digit i using the longest path in the ring in direction i. Correcting digit i using the longest path in the ring in direction i needs $(k - w_i)$ links, and correcting the remaining digits using the shortest path in each ring needs $(l - w_i)$ links. Summing these links results in a path of length

$$l + k - 2w_i \text{ links.}$$

It can be seen that all these paths do not share any node other than A and B. □

Note that when $k = 2$, the above proposition will not be valid because adding one to a bit is equal to subtracting one from the same bit in the binary hypercube.

Example 2.2.2

In a 5-ary 3-cube, let $A = 013$, $B = 034$. The set of 6 parallel paths between A and B is

Path 1: $013 \longrightarrow 014 \longrightarrow 024 \longrightarrow 034$

Path 2: $013 \longrightarrow 023 \longrightarrow 033 \longrightarrow 034$

Path 3: $013 \longrightarrow 113 \longrightarrow 114 \longrightarrow 124 \longrightarrow 134 \longrightarrow 034$

Path 4: $013 \longrightarrow 413 \longrightarrow 414 \longrightarrow 424 \longrightarrow 434 \longrightarrow 034$

Path 5: $013 \longrightarrow 012 \longrightarrow 022 \longrightarrow 032 \longrightarrow 031 \longrightarrow 030 \longrightarrow 034$

Path 6: $013 \longrightarrow 003 \longrightarrow 004 \longrightarrow 044 \longrightarrow 034$

2.3 Gray Codes

The binary reflected Gray codes have been extensively used in binary hypercubes [BeT89, SaS88, Lei92]. This strategy can be extended to k-ary n-cubes to construct a sequence of k^n distinct k-ary numbers with n digits each, with the property that the Lee distance of successive numbers is one. The Lee distance of the first and the last numbers in the sequence is also one. In order to obtain this property, we need to define a new class of generalized codes called *k-ary Reflected Gray codes* which can be used for both the binary and the k-ary n-cubes.

Definition 2.3.1

Let $G_k(n)$ be the k-ary reflected Gray Code of k-ary n-cube, $Q_k(n)$ be the sequence of $G_k(n)$ with zero in the rightmost digit and $S_k(n)$ be the sequence of $G_k(n)$ excluding $Q_k(n)$. More generally, denoting by $S_k^R(n)$ (resp., $Q_k^R(n)$) the sequence obtained from $S_k(n)$ (resp., $Q_k(n)$) by reversing its order, and by $iS_k(n)$ (resp., $iQ_k(n)$) the sequence obtained from $S_k(n)$ (resp., $Q_k(n)$) by prepending digit i to each element of the sequence.

Define

$$G_k(1) = \{0, 1, 2, 3, ..., k-1\}.$$

We can get

$$Q_k(1) = \{0\}, \text{ and}$$

$$S_k(1) = \{1, 2, 3, ..., k-1\}.$$

Then the k-ary Reflected Gray codes of arbitrary order can be generated by the recursion

$$\begin{aligned} G_k(n+1) = \{&0S_k(n), 1S_k^R(n), 2S_k(n), 3S_k^R(n), ..., \\ &(k-1)S_k^R(n), (k-1)Q_k^R(n), (k-2)Q_k(n), \\ &(k-3)Q_k^R(n), ..., 0Q_k(n)\} \end{aligned}$$

if k is even, and

$$\begin{aligned} G_k(n+1) = \{&0S_k(n), 1S_k^R(n), 2S_k(n), 3S_k^R(n), ..., (k-1)S_k(n), \\ &(k-1)Q_k(n), (k-2)Q_k^R(n), (k-3)Q_k(n), ..., 0Q_k(n)\} \end{aligned}$$

if k is odd. Clearly,

$$| G_k(n) | = k^n.$$

Table 2.3.1 $G_4(2)$

$\mathcal{G}_4(1)$	$\mathcal{Q}_4(1)$	$\mathcal{S}_4(1)$
0	0	1
1		2
2		3
3		
$\mathcal{G}_4(2)$	$\mathcal{Q}_4(2)$	$\mathcal{S}_4(2)$
01	30	01
02	20	02
03	10	03
13	00	13
12		12
11		11
21		21
22		22
23		23
33		33
32		32
31		31
30		
20		
10		
00		

The *k*-ary Reflected Gray code provides a mapping of a linear array or a ring with k^n nodes into the *k*-ary *n*-cube. Refer to Tables 2.3.1 and 2.3.2 for examples of $G_4(2)$ and $G_3(3)$. This strategy proves that the *k*-ary *n*-cube is Hamiltonian.

Table 2.3.2 $G_3(3)$

$G_3(1)$	$Q_3(1)$	$S_3(1)$
0	0	1
1		2
2		
$G_3(2)$	$Q_3(2)$	$S_3(2)$
01	20	01
02	10	02
12	00	12
11		11
21		21
22		22
20		
10		
00		
$G_3(3)$	$Q_3(3)$	$S_3(3)$
001	220	001
002	210	002
012	200	012
011	100	011
021	110	021
022	120	022
122	020	122
121	010	121
111	000	111
112		112
102		102
101		101
201		201
202		202
212		212
211		211
221		221
222		222
220		
210		
200		
100		
110		
120		
020		
010		
000		

Chapter 3
Communication Algorithms

One of the most important components of any large-scale general-purpose parallel computer is its communication algorithm. This is because most large-scale general-purpose machines spend a large portion of their resources making sure that the right data gets to the right place within a reasonable amount of time. Most of the algorithms discussed in the literature under various assumptions are for the binary n-cube (see [BOS91, JoH89, SaS85, BhA84]). So, we restrict ourselves on the k-ary n-cube where $k > 2$. In this chapter, we propose the problems of moving data from one processor to another processor, a single processor sending the same data to every other processor, a single processor sending different data to every other processor, simultaneous broadcast of the same data from every processor to all other processors, and simultaneous exchange of different data between every pair of processors.

Information is transmitted along the k-ary n-cube links in groups of bits called *packets*. In our algorithms we assume that the time required to cross any link is the same for all packets and is taken to be one unit. All packets have roughly equal length. We assume that packets can be transmitted along a link in one direction and that their transmission is error free. Only one packet can travel along a link in one direction at any one time; thus, if more than one packet is available at a node and is scheduled to be transmitted on the same incident link of the node, then only one of these packets can be transmitted at the next time period, while the remaining packets must be stored at the node while waiting in queue. This scheme is known as *store-and-forward routing*.

Each node is assumed to have infinite storage space. Moreover, we assume that at any time, a node can transmit a packet along at most one incident link and can simultaneously receive a packet along at most one incident link; this is called *one-port communication*. Another possibility is the *2n-port communication* where it is assumed that all incident links of a node can be used simultaneously for packet transmission and reception. All the accounts assume that the overhead per packet, propagation and queuing delays on all links are negligible.

3.1 Pipelining

Assume that a message is to be transmitted over a path of $k > 1$ communication links. Dividing the message into m packets and transmitting them sequentially over the k-link path will reduce the delay time from mk, if the message transmitted as a whole packet, to $m+k-1$ [BeT89].

Proposition 3.1.1

If a message to be transmitted over a path of $k > 1$ communication links is divided into m packets that transmitted sequentially over the k-link path, then the delay time will be reduced from mk to $m+k-1$.

Proof

If the message transmitted as a whole packet then in each link it will take m time units. But we have k-link path implying a total of mk time units.

Now assume that each packet is transmitted sequentially over the k-link path.

Then Packet1 will reach the destination in time k.

Packet2 will reach the destination in time $k+1$.

Packet3 will reach the destination in time $k+2$.

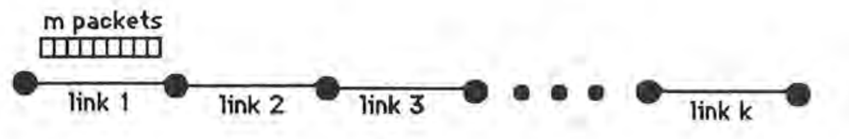
.... ,

Packet m will reach the destination in time $k+m-1$. □

See Figure 3.1.1 for more clarification.

3.2 Moving Data Between Two Nodes

Let A and B be any two nodes of the k-ary n-cube and consider the problem of sending data from node A to node B. Using the result of section 2.1 we can send a data packet from node A to node B in time equal to the Lee distance between A and B by modifying successively the bits of A one by one in order to transform the label A into the label B.



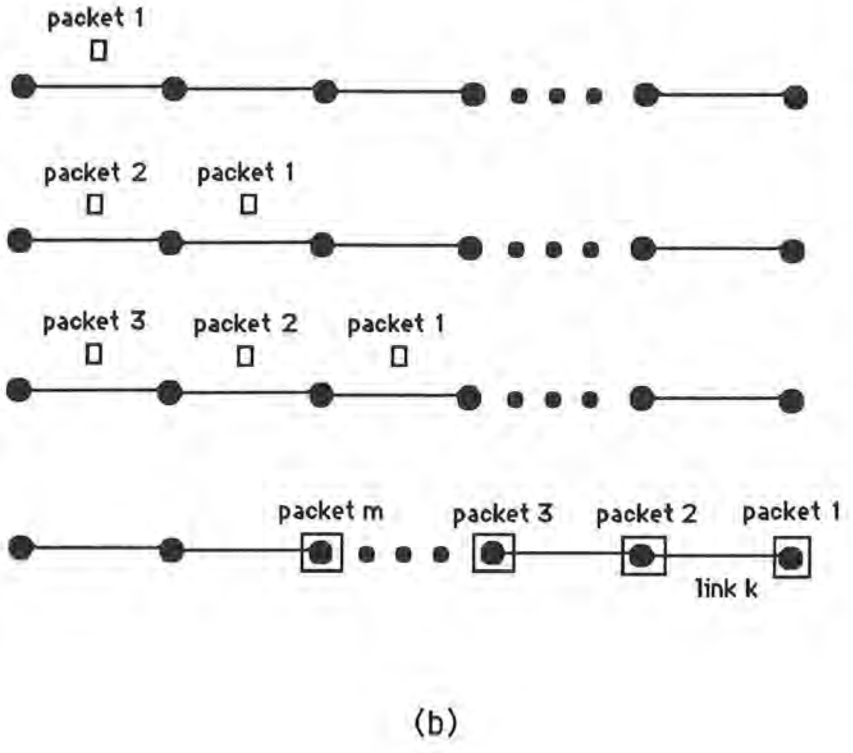


Figure 3.1.1

Segmentation of a message into *m* packets to take advantage of pipelining over a *k*-link communication path. (a) a message of length *m* packets is transmitted as a whole packet on each link requiring *m* time units on each link for a total of *mk* time units. (b) the message is divided into *m* packets each requiring one time unit for transmission over a single link for a total of $m+k-1$.

Consider now sending a message of *m* packets from node *A* to node *B*. Then the time required for this process using one-port communication and pipelining discussed in section 3.1 is

$$m + D_L(A,B) - 1$$

where $D_L(A,B)$ is the Lee distance between A and B. The above result can be improved by splitting the message into $2n$ parts each of $\lceil m/2n \rceil$ packets. Then each part can be sent along one of the $2n$ parallel paths discussed in section 2.2. If $P_{max}(A,B)$ is the length of the maximum parallel path between A and B, we have proved the following result.

Proposition 3.2.1

The time required to send a message of m packets from node A to node B on a k-ary n-cube of 2n-port communication is

$$\lceil m/2n \rceil + P_{max}(A,B) - 1 .$$

Observe that this time is optimal if $\lceil m/2n \rceil > P_{max}(A,B)$.

3.3 Single Node Broadcasting

The single node broadcast algorithm requires sending data from one processor to all other processors.

Starting from node (00...0), the single node broadcast algorithm constructs a spanning tree sequentially starting from the root by using the rule that the addresses of the children of each node are obtained as follows:

1) adding one to the leftmost non-zero digit of the parent address if the leftmost non-zero digit is less than $\lfloor k/2 \rfloor$.

2) subtracting one from the leftmost non-zero digit of the parent address if the leftmost non-zero digit is greater than $\lfloor k/2 \rfloor + 1$.

3) adding and subtracting one from the zero digits of the parent address that follow the leftmost non-zero digit.

The resulting leaf nodes will have $\lfloor k/2 \rfloor$ or $\lfloor k/2 \rfloor + 1$ as the leftmost digit in their address.

Each node (except the leaves) of the spanning tree utilize more than one of its incident links at the same time (2n-port communication). The above algorithm can be easily modified to make

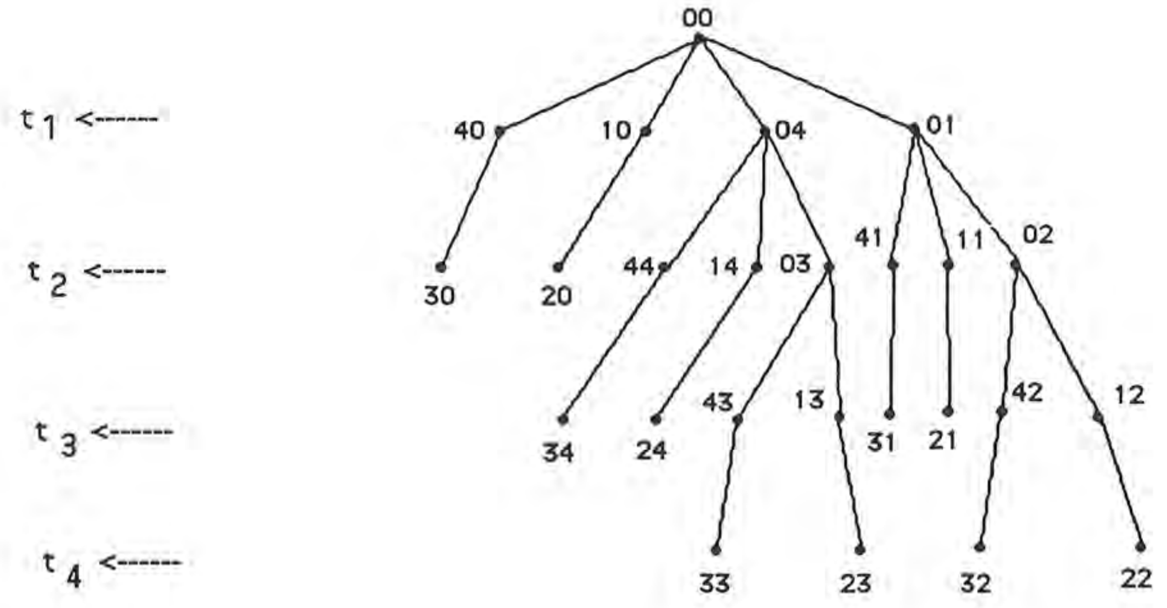


Figure 3.3.1

A spanning tree for a single node broadcast algorithm on a 5-ary 2-cube of 2n-port communication.

each node use only one incident link each time by making the steps of the above algorithm performing in different times. Figure 3.3.1 illustrates an example of steps performed on a 5-ary 2-cube of 2n-port communication and Figure 3.3.2 illustrates the steps performed on a 3-ary 3-cube of one-port communication.

The time complexity may be analyzed for both the 2n-port and one-port communications. In the 2n-port communication, the data packet sent from the root in both counterclockwise and clockwise directions simultaneously to all the adjacent nodes in rings in all directions. Then each node will send the packet to its adjacent nodes in the rings of the same or higher directions (not lower directions). Since the data packet takes $\lfloor k/2 \rfloor$ steps to reach the most remote node in the ring, the total time required is

$$\lfloor k/2 \rfloor * n .$$

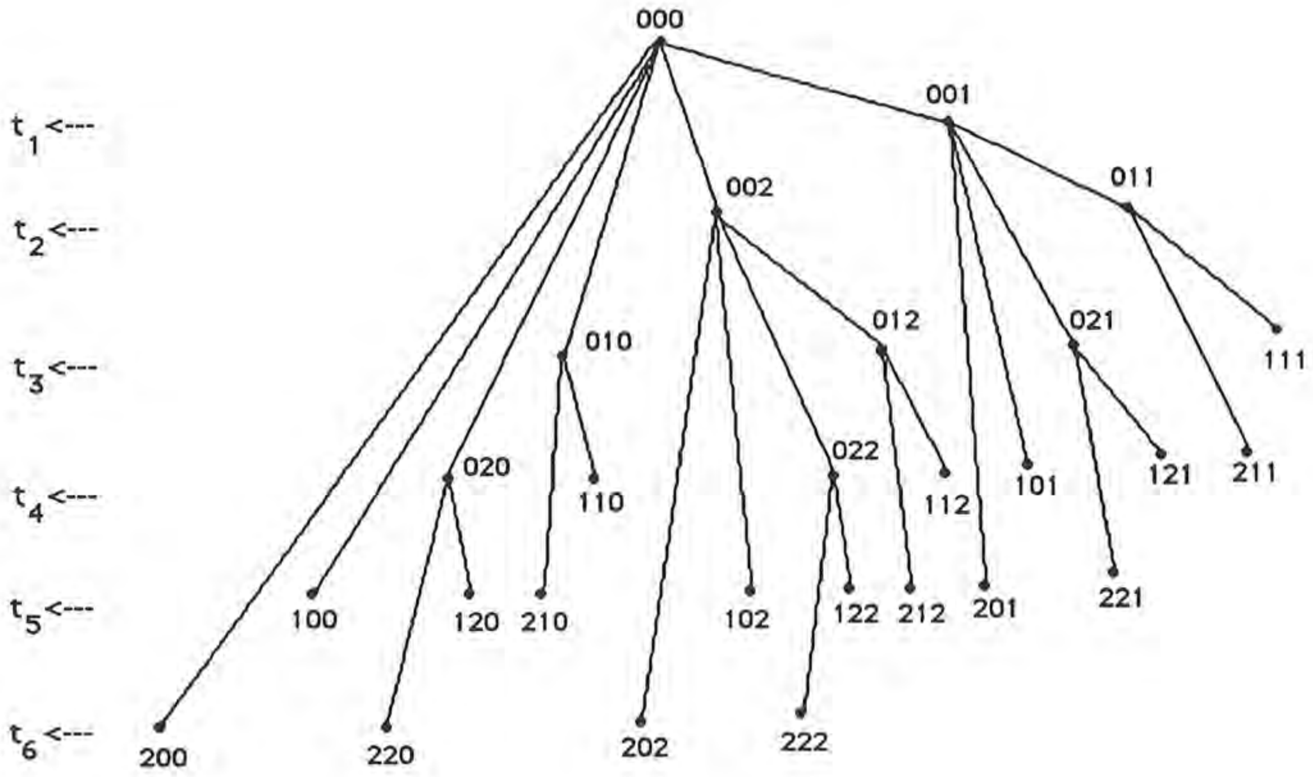


Figure 3.3.2

A spanning tree for a single node broadcast algorithm on a 3-ary 3-cube of one-port communication.

Since the one-port communication requires to send the data packet in the ring in one direction at a time, this results that the most remote node in the ring will receive the data packet in step $\lceil k/2 \rceil$. This requires a total time of

$$\lceil k/2 \rceil * n.$$

The discussion above may be summarized in the following.

Proposition 3.3.1

The single node broadcast algorithm on a *k*-ary *n*-cube requires a total time of

$$\lfloor k/2 \rfloor * n$$

using 2n-port communication, and

$$\lceil k/2 \rceil * n$$

using one-port communication.

Proposition 3.3.2

The time required to perform the single node broadcast algorithm on a k-ary n-cube of 2n-port communication is optimal.

Proof

The time required to perform the single node broadcast algorithm on a k-ary n-cube of $2n$-port communication is the diameter of the network. □

The above algorithm broadcast the data from node (00...0). If we want to broadcast the data from any other node say $A = (a_n, a_{n-1}, ..., a_1)$, we simply renumber the nodes so that node A becomes node (00...0) and the new numbering consistent with the k-ary n-cube numbering, i.e., each neighbors still have their new labels having a Lee distance of one. This can be achieved by adding the vector $(k-a_n, k-a_{n-1}, ..., k-a_1)$ to all the node labels.

3.4 Multinode Broadcasting

The process of moving data from every node to every other node is called multinode broadcast since every processor wants to do single node broadcast simultaneously.

The easiest way to solve this problem is to broadcast the data from each node in turn using the single node broadcast algorithm of section 3.3. Since we perform a single node broadcast from each of the k^n nodes in turn, this will require a total time of

$$k^n(\lceil k/2 \rceil * n)$$

using one-port communication, and

$$k^n(\lfloor k/2 \rfloor * n)$$

using the 2n-port communication.

Another method is based on mapping a ring of k^n nodes into the k-ary n-cube, and exchange the data along the ring in a daisy-chain manner [SaS85, BeT89]. Using the k-ary reflected Gray code strategy of section 2.3, we can map a ring of k^n nodes to the k-ary n-cube and perform a multinode broadcast on the ring as follows: at stage 1 each node sends its own packet to its counterclockwise neighbor. At stage 2, ..., k^n-1, each node sends to its counterclockwise neighbor the packet received from its clockwise neighbor at the previous stage. The total time required to end this process is

$$k^n-1$$

using one-port communication. This is optimal since each node receives a single packet in each unit of time from the k^n-1 other nodes over a single communication link. This algorithm can be improved to utilize all the $2n$ communication links at each stage. Figure 3.4.1 illustrates this process for 2-ary 2-cube.

3.5 Single Node Scattering

The single node scatter problem involves sending a unique data packet from a single node, called the root, to every other node.

The general strategy for single node scatter algorithm for one-port communication is to map a linear array of k^n nodes to the k-ary n-cube with the help of the k-ary reflected Gray codes of section 2.3, and schedule the data packet for the most remote nodes first. Hence, the total time required for performing this algorithm is

$$k^n - 1 .$$

This algorithm is optimal since the root node sends one packet each time over one communication link for all other k^n -1 nodes. This algorithm can be improved to utilize all the $2n$ incident links. Figure 3.5.1 illustrates an example of 5-ary 1-cube.

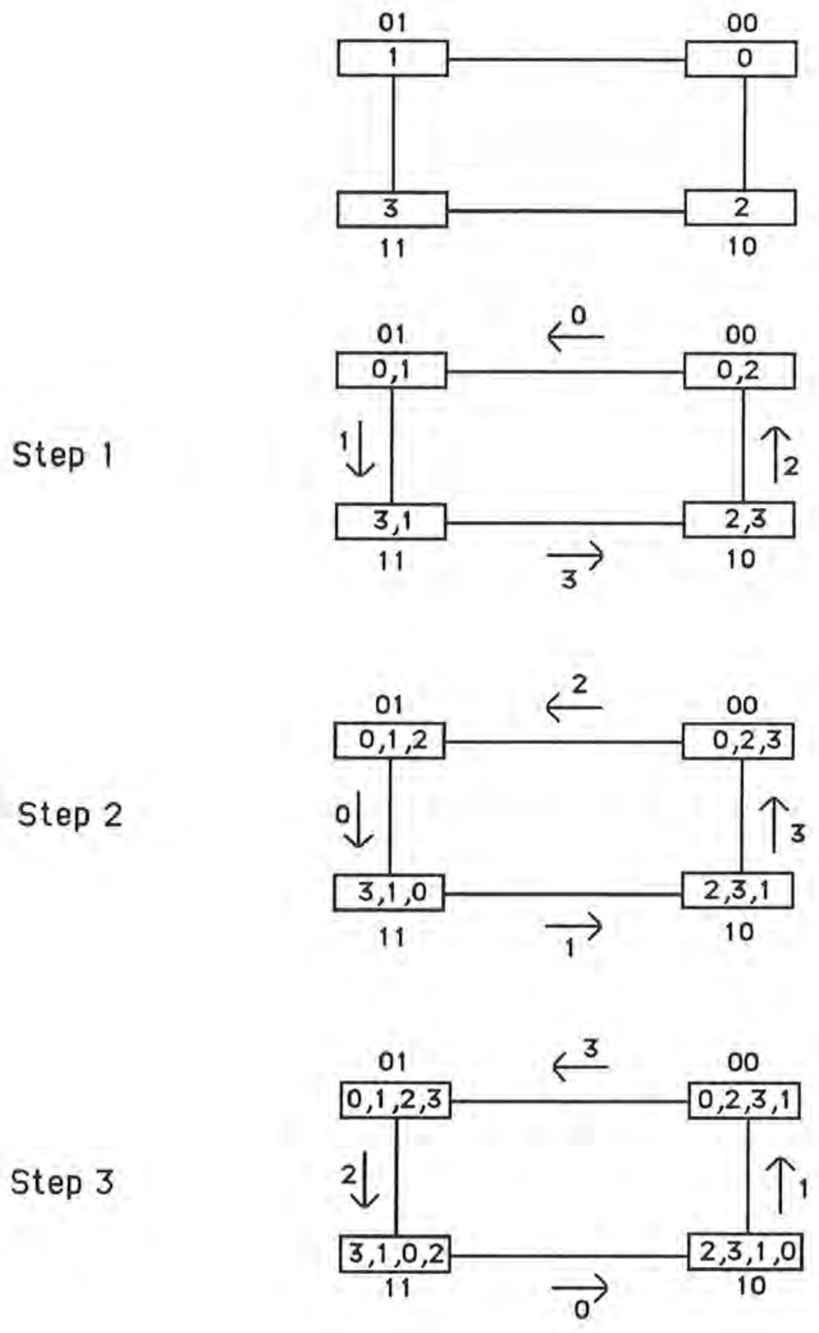


Figure 3.4.1

A ring of 4 nodes mapped to a 2-ary 2-cube and performs a Daisy-Chaining multinode broadcast with one-port communication. The process ends after three time units.

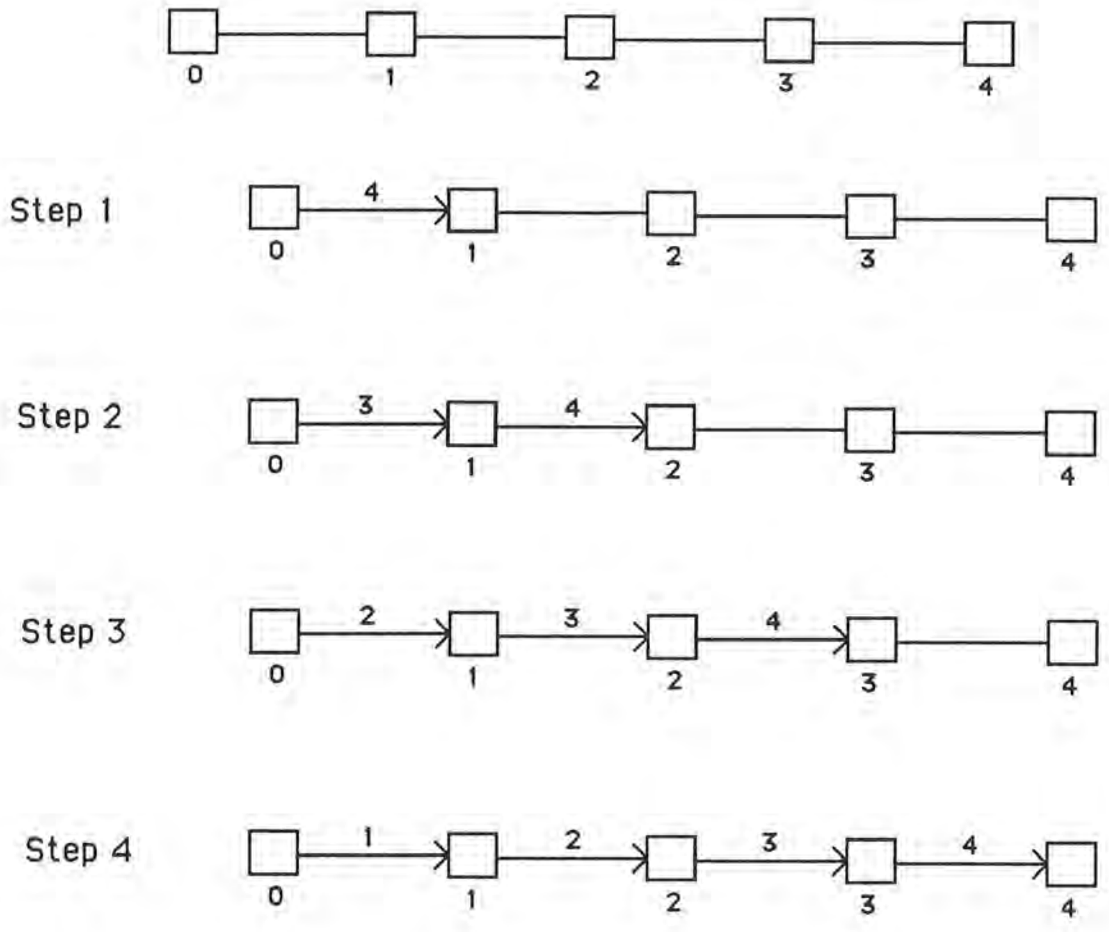


Figure 3.5.1

A linear array of five nodes mapped to a 5-ary 1-cube and performs a single node scatter algorithm in four time units.

3.6 Total Exchange

The total exchange problem involves sending different packet from every node to every other node (in contrast with the multinode broadcast problem, where every node sends the same packet to every other node).

The total exchange algorithm on a k-ary n-cube consists of n stages. In stage i, $1 \leq i \leq n$, each ring im direction i will perform in parallel the total exchange algorithm such that a node $A = (a_n, a_{n-1}, ..., a_i, ..., a_1)$ will send all the accumulated data packet (his packet together with the packets received from the previous step) to node $B = (a_n, a_{n-1}, ..., b_i, ..., a_1)$ (e.g., both A and B in the same ring in direction i) if the i^{th} digit of the destination address of the data packet sent from A to B is equal to b_i. In other words, any data packet sent from node A to node B on a k-ary n-cube will follow the shortest path (Lee distance) from A to B.

To analyze the time complexity of this algorithm, we will compute first the lower bound for the time needed by any total exchange algorithm using *one*-port communication.

Proposition 3.6.1

Let $G=(N,A)$ be the k-ary n-cube network of P processors where transmission along at most one of the incident links of a node is allowed (one-port communication). Let $D_L(i, j)$ be the Lee distance from node i to node j. Then the quantity

$$D(G) = \sum_{j \in N} D_L(i, j)$$

is a lower bound for the time taken by any total exchange algorithm.

Proof [BeT89]

The number of packet transmissions before j receives the packet is no less than the Lee distance from i to j. Therefore, the total number of packet transmissions is at least

$$\sum_{i \in N} \sum_{j \in N} D_L(i, j) = PD(G)$$

where P is the number of processors. Since at most one packet transmission per processor is allowed at a time, the number of simultaneous packet transmissions can be at most P, thereby establishing the lower bound of $D(G)$ time units. □

Proposition 3.6.2

The total exchange algorithm on a ring of k nodes requires at least

$$£ = \begin{cases} \frac{k^2-1}{4} & \text{if } k \text{ is odd} \\ \\ \frac{k^2}{4} & \text{if } k \text{ is even} \end{cases}$$

time units.

Proof

From the previous result, and since a *k*-ring is a *k*-ary *1*-cube, we can fix a node *i* and take the sum of the distances from *i* to all the other nodes. See Figure 3.6.1, if *k* is odd, then from Figure 3.6.1a we have

$$£ = 2 * 1 + 2 * 2 + 2 * 3 + \ldots + 2 * \frac{k-1}{2}$$

$$= 2\left(1 + 2 + 3 + \ldots + \frac{k-1}{2}\right)$$

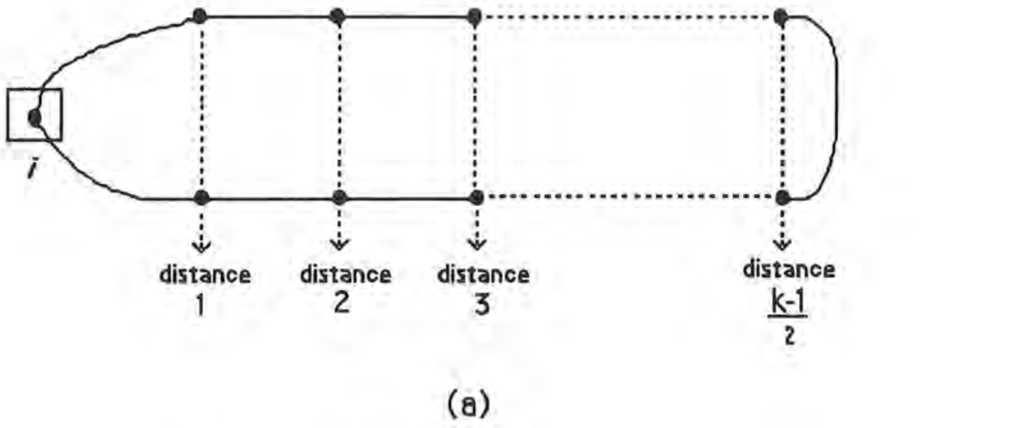
$$= 2\left(\frac{\frac{k-1}{2}\left(\frac{k-1}{2} + 1\right)}{2}\right)$$

$$= \frac{k-1}{2}\left(\frac{k-1}{2} + 1\right) = \frac{k^2-1}{4}.$$

If *k* is even, then from Figure 3.6.1b we have

$$£ = 2 * 1 + 2 * 2 + 2 * 3 + \ldots + 2 * \left(\frac{k}{2} - 1\right) + \frac{k}{2}$$

$$= 2\left(1 + 2 + 3 + \ldots + \left(\frac{k}{2} - 1\right)\right) + \frac{k}{2}$$



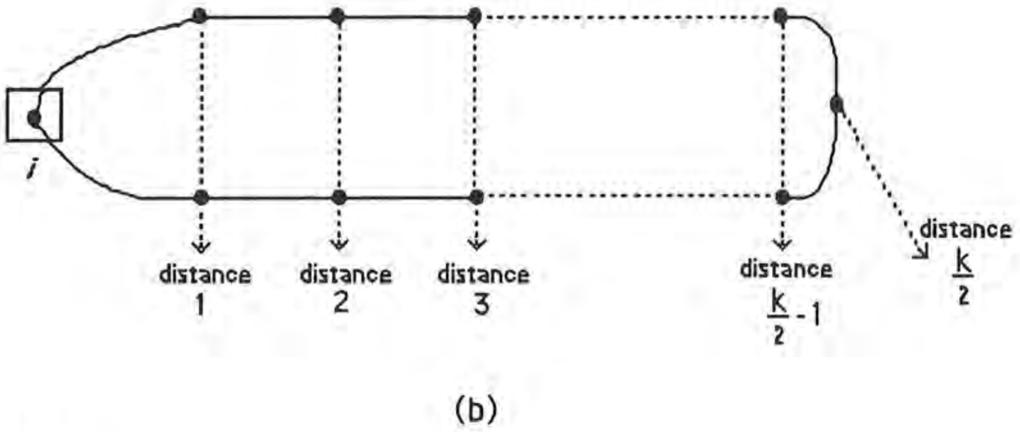


Figure 3.6.1

Illustration of the distance of each node from specific node *i* in a ring of *k* nodes. (a) if *k* is odd, (b) if *k* is even.

$$= 2\left(\frac{\left(\frac{k}{2} - 1\right)\frac{k}{2}}{2}\right) + \frac{k}{2} = \frac{k^2}{4}. \qquad \square$$

Corollary

In a *k* processor ring where $G=(N,A)$,

$$\pounds = D(G) = \sum_{j \in N} D_L(i,j) = \sum_{j=0}^{k-1} W_L(j) \quad .$$

To attain the optimal time unit, each processor should send data packets in both clockwise and counterclockwise directions. It is clear from Figure 3.6.1a that node *i* will send half of its packets in clockwise and the other half in counterclockwise directions. This will result in

$$\frac{k^2 - 1}{8}$$

steps needed in both clockwise and counterclockwise directions, if *k* is odd.

If *k* is even, however, processor *i* from Figure 3.6.1b will send $(k/2 - 1)$ of his data in one direction and$(k/2)$ of his data in the other direction. This will result in

$$\frac{k^2}{8} - \frac{k}{4}$$

steps needed in one direction, and

$$\frac{k^2}{8} + \frac{k}{4}$$

steps needed in the other direction.

To prove this result, Let x be the number of steps in one direction, then from Figure 3.6.1b, the other direction will have $x + \frac{k}{2}$ steps.

The total steps is

$$£ = \frac{k^2}{4} = x + x + \frac{k}{2}$$

solving for x, we get

$$x = \frac{k^2}{8} - \frac{k}{4},$$

this is the steps needed in one direction, and the other direction will have

$$x + \frac{k}{2} = \frac{k^2}{8} - \frac{k}{4} + \frac{k}{2} = \frac{k^2}{8} + \frac{k}{4}$$

steps. The proof is complete. Figure 3.6.2 illustrates an example of total exchange algorithm on a four processor ring.

We can use the results of the total exchange algorithm on a *k* processor ring to analyze the time complexity of the total exchange algorithm on a *k*-ary *n*-cube.

Proposition 3.6.3

Any total exchange algorithm on a *k*-ary *n*-cube of one-port communication will require at least

$$n£k^{n-1}$$

time units.

Proof

Let $G(N,A)$ be the *k*-ary *n*-cube. Let v_0=(00....0). Arrange the k^n labels of the *k*-ary *n*-cube into k^n Xn matrix M. Let $i \in <k>$. From the symmetry of the network, each element i will appear k^{n-1} times in each column of M. This implies that the total Lee weight of M is

$$W_L(M) = nk^{n-1} \sum_{i=0}^{k-1} W_L(i).$$

From the pervious corollary, we get

$$W_L(M) = n£k^{n-1}.$$

From proposition 3.6.1, the lower bound is

$$D(G) = \sum_{j \in N} D_L(i,j) = \sum_{j \in N} D_L(v_0,j)$$

$$= \sum_{j \in N} W_L(j-v_0) = \sum_{j \in N} W_L(j)$$

$$= W_L(M) = n£k^{n-1}.$$ □

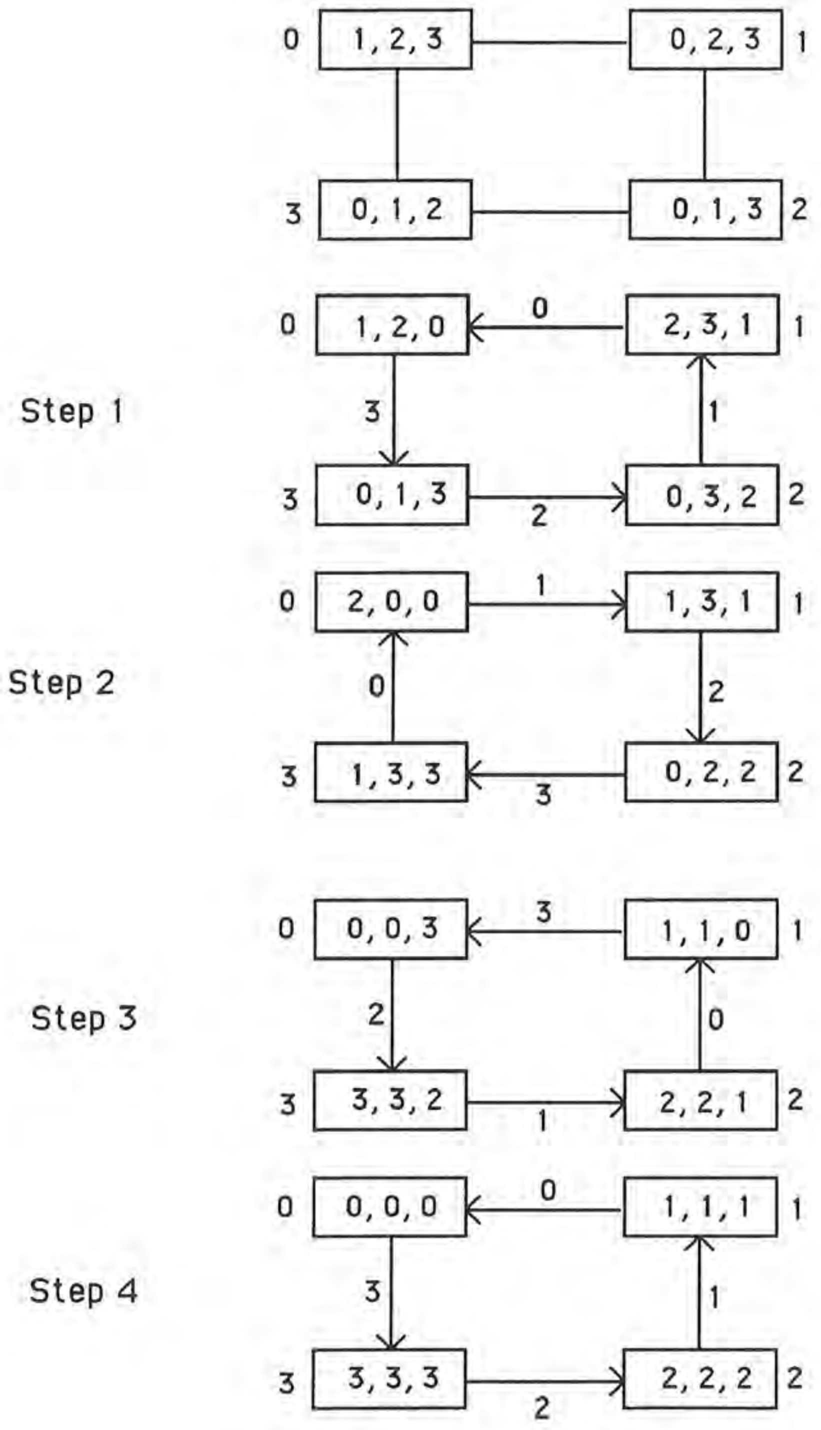


Figure 3.6.2

An example of total exchange algorithm in a ring of four nodes. It needs three counterclockwise steps and one clockwise step to end the process.

Now we reach a point that we can prove that our total exchange algorithm is optimal using one-port communication.

Proposition 3.6.4

At the end of each stage of the total exchange algorithm, each node will receive a total of

$$(k-1)k^{n-1}$$

data packets using one-port communication.

Proof (by induction)

For $n=1$, the result is a ring of k nodes and each node will receive a packet from the other $k-1$ nodes.

Now, assume that it is true for stage n, then at the end of stage n every node of the k-ary n-cube will have a total of k^n different data packets distined to each node of the ring of direction $n+1$. This results in every node will receive from each of the $k-1$ other nodes in the ring of direction $n+1$ a total of k^n data packets resulting in receiving accumulative of

$$(k-1)k^n$$

data packets in stage $n+1$ of the k-ary $(n+1)$-cube. □

Proposition 3.6.5

The time units required by the total exchange algorithm on a k-ary n-cube is

$$n£k^{n-1}$$

Proof

From the previous proposition, each node will receive a total of

$$(k-1)k^{n-1}$$

data packets at the end of each stage. However, our algorithm requires n stages. This will result in each node will receive a total of

$$n(k-1)k^{n-1}$$

data packets all over the n stages. The time required to receive $k-1$ data packets is £, implying that the time units required by the algorithm is

$$n£k^{n-1} \quad .$$ □

This algorithm is optimal since each processor exchanges its packet through the Lee distance with all the other processors using one-port communication. The algorithm can be improved to utilize all the $2n$ incident links.

Chapter 4

Conclusion

4.1 Summary

We have shown some properties of *k*-ary *n*-cubes ($k>2$), hoping to make these networks more attractive. Using the *k*-ary *n*-cube model, we can construct a low-dimensional network by increasing *k* and decreasing *n* to make it consistent with VLSI technology. The properties of the *k*-ary *n*-cube may be summarized as follows:

- The *k*-ary *n*-cube network can be constructed recursively from low dimensional cubes.
- The degree of each node is $2n$.
- The total number of links is nk^n.
- The diameter of the network is $\lfloor k/2 \rfloor * n$.
- There are k^{n-1} node-disjoint *k*-rings in each direction.
- There are $2n$ node-disjoint parallel paths between any two nodes.
- A ring or a linear array with k^n nodes can be mapped into the *k*-ary *n*-cube network using the *k*-ary reflected Gray codes strategy.

All the communication algorithms discussed are optimal using one-port communication. One-to-one and single node broadcasting algorithms are optimal using $2n$-port communication. Multinode broadcasting, single node scattering and total exchange algorithms can be improved to utilize all the $2n$ communication links simultaneously.

4.2 Future Research

We have shown some of the properties of the *k*-ary *n*-cube network ($k>2$) in Chapter 2. We are trying to find *n-edge disjoint Hamiltonian Cycles* in the network. Another fundamental issue of both theoretical and practical importance is related to the *portability* of the algorithms across various parallel architectures. Embedding of many topologies like linear arrays, rings, two dimensional meshes and binary trees into the k-ary n-cube is important to study. Examining the *fault tolerance* of the network is another interesting subject to study. Solving the problems of multinode broadcasting, single node scattering and total exchange algorithms using $2n$-port communication will improve the results of Chapter 3.

References

[BeT89] D. Bertsekas and J. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Prentice-Hall, Englewood Cliffs, NJ. (1989).

[BhA84] L. Bhuyan and D. Agrawal. Generalized Hypercube and Hyperbus Structures for a Computer Network. *IEEE Trans. Comput. c-33 (1984) 323-333.*

[BOS91] D. Bertsekas, C. Ozveren, G. Stamoulis, P. Tseng, and J. Tsitsiklis. Optimal Communication Algorithms for Hypercubes. *J. Parallel and Distr. Comput. 11 (1991) 263-275.*

[DaE92] S. Dandamudi and D. Eager. Hot-Spot Contention in Binary Hypercube Networks. *IEEE Trans. Comput. 41 (1992) 239-244.*

[Dal90] W. Dally. Performance Analysis of *k*-ary *n*-cube Interconnection Networks. *IEEE Trans. Comput. 39 (1990) 775-785.*

[Dal91] W. Dally. Express Cubes: Improving the Performance of *k*-ary *n*-cube Interconnection Networks. *IEEE Trans. Comput. 40 (1991) 1016-1023.*

[DaS87] W. Dally and C. Seits. Deadlock-Free Message Routing in Multiprocessors Interconnection Networks. *IEEE Trans. Comput. C-36 (1987) 547-553.*

[Gal90] J. Gallian. *Contemporary Abstract Algebra*. D. C. Heath and Company, Lexington, MA (1990).

[HaR90] N. Hartsfield and G. Ringel. *Pearls in Graph Theory: A Comprehensive Introduction.* Academic Press, Inc., Boston, MA (1990).

[JoH89] S. Johnsson and C. Ho. Optimum Broadcasting and Personalized Communications in Hypercubes. *IEEE Trans. Comput. 38 (1989) 1249-1268.*

[LaD90] S. Lakshmivarahan and S. Dall. *Analysis and Design of Parallel Algorithms: Arithmetic and Matrix Problems*. McGraw-Hill, New York, NY (1990).

[Lei92] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays. Trees. Hypercubes*. Morgan Kaufmann, San Mateo, CA (1992).

[LiH91] D. Linder and J. Harden. An Adaptive and Fault Tolerant Wormhole Routing Strategy for *k*-ary *n*-cubes. *IEEE Trans. Comput. 40 (1991) 2-12.*

[Ozv87] C. Ozveren. Communication Aspects of Parallel Processing. Rep LIDS-P-1721, Laboratory for Information and Decision Systems, MIT, Cambridge, MA (1987).

[PeW72] W. Peterson and E. Weldon, JR. *Error-correcting Codes*. MIT, Cambridge, MA (1972).

[RaS88] P. Ramanathan and K. Shin. Reliable Broadcast in Hypercube Multicomputers. *IEEE Trans. Comput. 37 (1988) 1654-1657.*

[SaS85] Y. Saad and M. Schultz. Data Communications in Hypercubes. Res. Rep. YALEU/DCS/RR-428. Yale University, Oct, 1985.

[SaS86] Y. Saad and M. Schultz. Data Communication in Parallel Architectures. Res. Rep. YALEU/DCS/RR-461, Yale University, Mar, 1986.

[SaS88] Y. Saad and M. Schultz. Topological Properties of Hypercubes. *IEEE Trans. Comput. 37 (1988) 867-872.*

[Sei85] C. L. Seitz. The Cosmic Cube. *The Communications of ACM, 28 (1985) 22-33.*