AN ABSTRACT OF THE THESIS OF

Myung Mun Bae for the degree of Doctor of Philosophy in Computer Science presented on November 14, 1996. Title: Resource Placement, Data Rearrangement, and Hamiltonian Cycles in Torus Networks.

Redacted for Privacy

Abstract approved: _____

Bella Bose

Many parallel machines, both commercial and experimental, have been / are being designed with toroidal interconnection networks. For a given number of nodes, the torus has a relatively larger diameter, but better cost / performance tradeoffs, such as higher channel bandwidth, and lower node degree, when compared to the hypercube. Thus, the torus is becoming a popular topology for the interconnection network of a high performance parallel computers.

In a multicomputer, the resources, such as I/O devices or software packages, are distributed over the networks. The first part of the thesis investigates efficient methods of distributing resources in a torus network. Three classes of placement methods are studied. They are (1) distant-t placement problem: in this case, any non-resource node is at a distance of at most t from some resource nodes, (2) j-adjacency problem: here, a non-resource node is adjacent to at least j resource nodes, and (3) generalized placement problem: a non-resource node must be a distance of at most t from at least j resource nodes.

This resource placement technique can be applied to allocating spare processors to provide fault-tolerance in the case of the processor failures. Some efficient

spare processor placement methods and reconfiguration schemes in the case of processor failures are also described.

In a torus based parallel system, some algorithms give best performance if the data are distributed to processors numbered in Cartesian order; in some other cases, it is better to distribute the data to processors numbered in Gray code order. Since the placement patterns may be changed dynamically, it is essential to find efficient methods of rearranging the data from Gray code order to Cartesian order and vice versa. In the second part of the thesis, some efficient methods for data transfer from Cartesian order to radix order and vice versa are developed.

The last part of the thesis gives results on generating edge disjoint Hamiltonian cycles in k-ary n-cubes, hypercubes, and 2D tori. These edge disjoint cycles are quite useful for many communication algorithms.

Resource Placement, Data Rearrangement, and Hamiltonian Cycles in Torus Networks

by

Myung Mun Bae

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Doctor of Philosophy

Completed November 14, 1996
Commencement June 1997

Approved by Committee:

Redacted for Privacy

Major Professor (Bella Bose)

Redacted for Privacy

Committee Member (Curtis R. Cook)

Redacted for Privacy

Committee Member (Timothy A. Budd)

Redacted for Privacy

Committee Member (Prasad Tadepalli)

Redacted for Privacy

Graduate School Representative (John P. Bolte)

Date thesis presented November 14, 1996

ACKNOWLEDGEMENT

Glory of God!

First of all, I would like to express my gratitude to my advisor, Dr. Bella Bose, who has provided me continuous support, constant encouragement, and broad and deep advice throughout this study. I would like to thank Dr. Curtis Cook, Dr. Timothy Budd, Dr. Prasad Tadepalli, Dr. Vikram Saletore, and Dr. John Bolte for serving on my thesis committee. I would also like to thank Dr. Paul Cull for his constructive comment on this work.

Thanks also to my friends and colleagues, Dokyoung Ok, Seokjun Lee, Bob Broeg, Luca Tallini, Bader Almohammad, and Gowri Ramanathan for their help.

I am very grateful to my wife, Hee Ae, and my daughters, Min Ju and Min Hee, for their constant support, patience, joy, hope, and prayers during this long period of study. I also wish to thank my parents and my parents-in-law for their love and guidance.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

RESOURCE PLACEMENT, DATA REARRANGEMENT, AND HAMILTONIAN CYCLES IN TORUS NETWORKS

1. INTRODUCTION

A multicomputer is a system consisting of multiple nodes that communicate by exchanging messages through an interconnection network. At a minimum, each node normally consists of one or more processing elements, a local memory, and a communication module. A popular topology for the interconnection network is the *torus*. Also called a *wrap-around mesh* or a *toroidal mesh*, this topology includes the k-ary n-cube which is an n-dimensional torus with the restriction that each dimension is of the same size, k, and the hypercube, which is a k-ary n-cube with $k = 2$; a mesh is a subgraph of a torus.

Many linear algebra computations and partial differential equations can be performed efficiently on machines having a topology based on a torus. For a given number of nodes, the torus has a relatively larger diameter, but better cost/performance tradeoffs, such as higher channel bandwidth and a more scalable structure, when compared to the hypercube [40, 41, 2]. Thus, the torus is becoming a popular topology for the interconnection network of a high performance parallel computer. Several parallel machines, both commercial and experimental, have been designed with a toroidal (or mesh) interconnection network. Included among these machines are the following: Cray T3D and T3E (3D torus) [76], the Mosaic (k-ary n-cube) [88], the iWrap (torus) [20], the Tera Parallel Computer (torus) [90], the J-Machine (mesh) [75], the Goodyear Aerospace MPP (mesh) [14], the MasPar MP-1

(mesh) [25], the K2 Parallel Processor (mesh) [6], Ametek 2010 (mesh) [87], the Stanford DASH (mesh) [68], the Touchstone Delta System (mesh) [98], the Cosmic Cube (hypercube) [86], the Ametek S/14 (hypercube) [7], the nCUBE (hypercube) [45, 46, 25], the iPSC (hypercube) [45], and the CM-200 (hypercube) [28].

This thesis deals with three issues related to torus networks. These three problems and the significance of these problems are briefly described below:

1. Resource Placement Problem

   In a multicomputer, there may be resources, such as I/O processors or software packages, that each processor needs to access. However, because of expense or frequency of use, it may not be desirable to place a copy of the resource at each node in the system. In general, then, the problem of resource placement is how should a limited number of copies of a resource be disseminated throughout a system giving comparable access to all processors. Further, as the complexity of the system increases, the possibility of the system failure increases. Thus, in the case of some processor failures, it is desirable to have a reconfiguration scheme to preserve the functionality as well as the logical and physical topology of the system. The resource placement technique can be applied to one class of fault-tolerance in the parallel computer architectures - the resource nodes in the resource placement problems can be treated as the extra spare processors in the spare processor allocation problems.

2. Data Rearrange Problem

   Data placement patterns are generally different for different algorithms. For example, many scientific problems, such as finite element analysis, signal and image processing, etc., use the regular grid data distribution patterns, while several other applications, such as the Cooley-Tukey Fast Fourier Transfor-

mation, reference data that are ordered in Gray codes. Therefore, conversion between the two placements is an important issue.

3. Gray Codes and Edge Disjoint Hamiltonian Cycles

   The Lee distance Gray codes are useful to construct a ring, and the edge disjoint cycles. These ring and edge disjoint cycles are useful in many embedding problems and communication algorithms.

1.1. Thesis Organization

The remainder of this thesis is organized as follows. Most of the placement results described in this thesis are based on the Lee distance error correcting codes, and these concepts are described in the following next few sections of this chapter. Chapter 2 presents the results on distance-t placement, j-adjacency resource placement, and the generalized placement methods. In Chapter 3, an application of the resource placement technique, namely, spare processor allocation method, will be presented. The data rearrangement technique between radix-k patterns and the Lee distance Gray code patterns is provided in Chapter 4. Here several efficient routing algorithms are given for two types of Gray codes. In Chapter 5, the edge disjoint Hamiltonian cycles are generated for some k-ary n-cubes, the hypercubes, and the $2D$ tori. Finally, some future research topics are described in Chapter 6.

The following few sections contain definitions and mathematical background that will be useful for the rest of this thesis.

1.2. Lee Distance and Torus

Let $A = a_{n-1}a_{n-2}\cdots a_0$ be a n-dimensional mixed radix vector over Z_K, where $K = k_{n-1} \times k_{n-2} \times \cdots \times k_0$, i.e., all $x_i \in Z_{k_i}$, for $i = 0, 1, \cdots, n-1$. The Lee weight of A in mixed radix notation is defined as

$$W_L(A) = \sum_{i=0}^{n-1} |a_i|,$$

$$\text{where} \qquad |a_i| = min(a_i, k_i - a_i), \text{for} \quad i = 0, 1, \cdots, n-1.$$

The Lee distance between two vectors A and B is denoted by $D_L(A, B)$ and is defined to be $W_L(A - B)$. That is, the Lee distance between two vectors is the Lee weight of their digit-wise difference. In other words, $D_L(A, B) = \sum_{i=0}^{n-1} min(a_i - b_i, b_i - a_i)$, where $a_i - b_i$ and $b_i - a_i$ are $mod\ \ k_i$ operations. For example, when $K = 4 \times 6 \times 3$, $W_L(321) = min(3, 4-3) + min(2, 6-2) + min(1, 3-1) = 1 + 2 + 1 = 4$, and $D_L(123 - 321) = W_L(202) = 3$.

Let $D_H(A, B)$ be the Hamming distance between two vectors A and B, i.e. the number of position in which A and B differ. Then $D_L(A, B) = D_H(A, B)$ when $k_i = 2$ or 3, for all i, and $D_L(A, B) \geq D_H(A, B)$ when $k_i > 3$ for some i. In the rest of the thesis, it is assumed that $k_i \geq 3$.

Just as Hamming distance may be used to define the binary hypercube graph, Q_n, the generalized hypercube graph [18], and twisted hypercube graph [38], Lee distance may be used to define the k-ary n-cube graph, C_k^n, and the n-dimensional torus, $T_{k_1,k_2,\cdots,k_n}$.

A k-ary n-cube graph (C_k^n) and an n-dimensional torus ($T_{k_1,k_2,\cdots,k_n}$) are $2n$-regular graphs containing k^n and $k_1k_2\cdots k_n$ nodes respectively. Each node in a C_k^n is labeled with a distinct n-digit radix-k vector while each node in a $T_{k_1,k_2,\cdots,k_n}$ is labeled with a distinct n-digit mixed radix vector. In this thesis, node labels will be written as $(a_{n-1}a_{n-2}\cdots a_0)$ or as $a_{n-1}a_{n-2}\cdots a_0$ rather than n-tuples $(a_{n-1}, a_{n-2}, \cdots, a_0)$

when there is no confusion. If u and v are two nodes in the graph, then there is an edge between them *iff* $D_L(u,v) = 1$. From the definition of Lee distance, it can be seen that every node in a C_k^n or a $T_{k_1,k_2,\cdots,k_n}$ shares an edge with two nodes in every dimension, resulting in a regular graph of degree $2n$. In addition, the shortest path between any two vectors, u and v has length $D_L(u,v)$. Note that a C_k^n is an n-dimensional hypercube, Q_n, when $k = 2$.

1.3. Cross Product of Graphs

Given graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$, define the *cross product* of G_1 and G_2, denoted by $G_1 \times G_2$ [67, 22], as the graph $G = (V, E)$, where

$V = \{(u,v) | u \in V_1, v \in V_2\}$, and

$E = \{((u_1,v_1),(u_2,v_2)) | ((u_1,u_2) \in E_1 \text{ and } v_1{=}v_2), \text{ or } (u_1{=}u_2 \text{ and } (v_1,v_2) \in E_2)\}$.

A cycle of length k is denoted by C_k, and each node in C_k is labeled with a radix k number, $0 \cdots k-1$. There is an edge between vertices u and v *iff* $D_L(u,v) = 1$.

A k-ary n-cube (C_k^n) and an n-dimensional torus ($T_{k_1,k_2,\cdots,k_n}$) can alternately be defined as the product of cycles as follows.

$$C_k^n = \underbrace{C_k \times C_k \times \cdots \times C_k}_{n \text{ times}} = \times_{i=1}^{n} C_k$$

$$T_{k_1,k_2,\cdots,k_n} = C_{k_1} \times C_{k_2} \times \cdots \times C_{k_n}$$

The above definition demonstrates a useful topological property of a C_k^n: a C_k^n can be recursively defined in terms of smaller k-ary cubes.

$$C_k^n = \begin{cases} C_k, & \text{if } n = 1 \\ C_k \times C_k^{n-1}, & \text{if } n > 1 \end{cases}$$

1.4. Linear Codes for Lee distance

Many of the resource placement methods described in this thesis are derived from the error correcting linear codes. In this section, the design of error correcting codes for Lee distance is briefly described.

A linear code can be defined in terms of either a parity check matrix, $H_{r\times n}$, or a generator matrix $G_{(n-r)\times n}$. The H matrix contains r vectors of length n, and the elements are over Z_k^n. A vector c of length n over Z_k^n is a code word *iff* $cH^T = 0$. Furthermore, H is a parity check matrix of a t-error correcting code *iff* $vH^T \neq 0$, where v is a vector of length n over Z_k^n and $0 < W_L(v) \leq 2t$. The length n and the number of check digits n of the code are equal to the number of columns and rows of H respectively. The number of information digits m is given by $m = n - r$.

Example 1.1 *The H matrix given below is the check matrix for a single error correcting code of length 12 over Z_5^{12}.*

$$H = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 \end{bmatrix}.$$

For this code, the number of check digits $r = 2$ and the number of information digits, $m = 12 - 2 = 10$. Further $cH^T \neq 0$, for any vector c of length 12 and weight 1 or 2. More on these codes will be described in Chapter 2.

By rearranging the columns of the parity check matrix, a parity check matrix of the form $H = [P\ I_r]$ can be obtained, where P is a $r \times (n - r)$ matrix and I_r is the $r \times r$ identity matrix. Then the generator matrix G of the code is given by $G = \left[I_{n-r}\ (-P)^T\right]$; it is easy to verify that $GH^T = 0$. For example, the above check matrix H can be rearranged as follows.

$$H' = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & 1 & 0 \\ 2 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 & 0 & 1 \end{bmatrix}.$$

Then its generator matrix G is given by

$$G = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 3 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 4 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 3 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 4 & 2 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 4 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 3 & 4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 3 & 1 \end{bmatrix}.$$

Suppose $X = (x_m x_{m-1} \cdots x_1)$ is a given information word, where $m = n - r$. A code word C is obtained by multiplying X by G, i.e., $C = XG$. In the above example, if $X = (1002000001)$, then $C = XG = (100200000113)$.

Note that if the generator matrix of a code is known, then it is easy to obtain the parity check matrix.

2. RESOURCE PLACEMENT

This chapter presents three classes of resource placement problems. The first class of problem is called the distance-t problem. This problem considers placing resources such that each processor in the system either has a copy of the resource or is at a distance of at most t from at least one processor having a copy of the resource. A processor having the resource is known as a *resource processor* or a *resource node.*

For an example, consider a torus with size 13×13. If resources are placed on the solid circle nodes as in Figure 2.1 (wraparound edges not shown), any non-resource node can access one resource at a distance of at most 2. In fact, from a resource node, there are 4 and 8 non-resource nodes at a distance one and two respectively. Thus a resource node covers 13 nodes (including itself). Since there are 13 resources in the system, this placement results in a perfect distance-2 placement.

The second class of problem is called the j-adjacency problem [81, 82]. This problem considers placing resources such that each non-resource node is adjacent to j resource nodes.

The third class of problem is the generalized placement problem, which is a combination of the distance-t and the j-adjacency problems. Here a non-resource node must be a distance of at most t from j resource nodes. In Figure 2.2, any non-resource node can reach at least 2 resources within a distance of 2, and at least 4 resources within a distance of 3.

The remainder of this chapter is organized as follows. In Section 2.2 and Section 2.3, the results on distance-t and j-adjacency resource placement are presented. The generalized placement methods are discussed in Section 2.4.

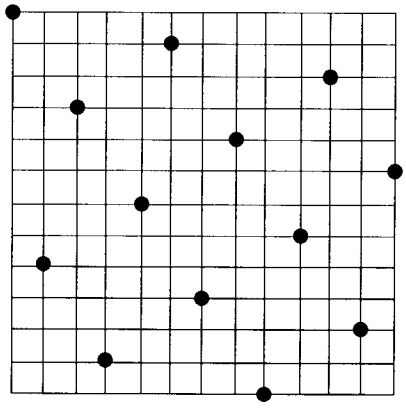


FIGURE 2.1. A perfect distance-2 placement in a 13×13 torus.

In the proposed methods, the resources are placed at the nodes whose addresses correspond to code words of an appropriate linear code. For example, the set of words $C = \{00, 13, 21, 34, 42\}$ over Z_5^2 form a Lee distance 3 code; the H matrix for this code is $H = [\ 2\ 1\ ]$. Thus, in a C_5^2, the resources are placed at these nodes to obtain distance-1 placement. By adding a constant vector to a distance-t code, a *coset* code is obtained. The coset code also satisfies the original distance property. For example, if we add (11) to the code words in C, we obtain $C' = \{11, 24, 32, 40, 03\}$. The minimum Lee distance of C' is also 3. Thus, instead of using the original code, one can as well use a coset code for the resource placement. However, the methods given in this thesis are described using the linear code.

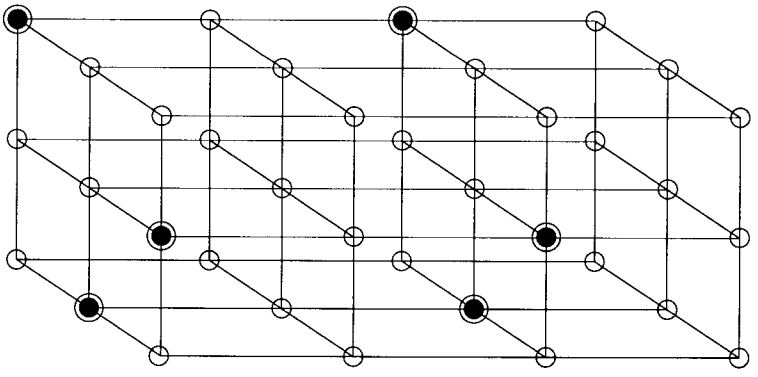


FIGURE 2.2. A placement in a $4 \times 3 \times 3$ torus with t=2, j=2 and t=3, j=4

2.1. Related Work

In the literature, most of the work done in this area so far are for the binary hypercubes. Reddy, Banerjee, and Abraham [84] have given a perfect distance-1 placement in a Q_n, when $(n+1)|2^n$, using Hamming single error correcting codes. Chiu and Raghavendra [33] have developed a hierarchical resource placement strategy as the recursive application of the basic strategy for the hypercube. Chen and Tzeng [30, 31] have studied the problems of perfect and quasi-perfect j-adjacency placement in a hypercube using binary linear codes, and they have also tried to find a generalized placement with the idea of a *bulky hypercube*. Tzeng and Feng [91] are the first to use covering radius codes for resource placement in a hypercube. Ramanathan and Chalasani [81, 82] are the first to study the j-adjacency placement in k-ary n-cubes. Bose *et al* [22] have discussed the topological properties of k-ary n-cubes including some preliminary results on resource placement problems using Lee distance codes. Recently, Cull and Nelson [39] proved that a perfect distance-1

placement is *NP-Complete* for a general graph. A distance-1 placement is perfect if a non-resource node is adjacent to exactly one resource node and no two resources nodes are adjacent.

2.2. Distance-t Placement

As we mentioned earlier, in the case of distance-t placement, each non-resource node should be a distance of at most t from a resource node. Our aim is to achieve this goal using as small a number of resource nodes as possible.

The following theorem derives a lower bound on the number of resource nodes, M, using the sphere packing bound for a k-ary n-cube [22, 23, 10, 12, 52, 15].

Theorem 2.1 *Assuming $t < \frac{k}{2}$, the minimum number of resource nodes, M, for distance-t placement in a k-ary n-cube, is*

$$M \geq k^n / \left(1 + \sum_{i=1}^{min(t,n)} 2^i \binom{n}{i}\binom{t}{i}\right). \tag{2.1}$$

Proof: Let $A = (a_{n-1}a_{n-2}\cdots a_0)$ and $B = (b_{n-1}b_{n-2}\cdots b_0)$ be any two resource nodes. Let S_A and S_B be the set of all nodes at a distance d, $(d = 1, 2, \cdots, t)$ from A and B respectively. If $S_A \cap S_B = \phi$, then $M = \frac{k^n}{|S_A|}$. Let x_i be the number of nodes which differ from node A in i positions, and are at a distance at most t from A, i.e., $x_i = |\{X | D_H(A, X) = i \text{ and } D_L(A, X) \leq t\}|$. Then the number of nodes in S_A is

$$|S_A| = 1 + \sum_{i=1}^{t} x_i.$$

To compute x_i, we consider the following choices;

(1) The number of ways to choose i positions among n possible dimensions $= \binom{n}{i}$.

(2) Let $f(t,i)$ be the number of ways to partition the integer t or less to exactly i parts, with each part greater than 0. Note that if the first part of the partition is the integer j, then we need to partition $t-j$ or less into exactly $i-1$ parts. Thus

$$f(t,i) = f(t-1,i-1) + f(t-2,i-1) + \cdots$$
$$+ f(i,i-1) + f(i-1,i-1).$$

The solution $f(t,i) = \binom{t}{i}$ satisfies the above equation. This is because,

$$\begin{aligned}
\binom{t}{i} &= \binom{t-1}{i-1} + \binom{t-1}{i} \\
&= \binom{t-1}{i-1} + \binom{t-2}{i-1} + \binom{t-2}{i} \\
&= \cdots \\
&= \binom{t-1}{i-1} + \binom{t-2}{i-1} + \cdots + \binom{i}{i-1} + \binom{i}{i} \\
&= \binom{t-1}{i-1} + \binom{t-2}{i-1} + \cdots + \binom{i}{i-1} + \binom{i-1}{i-1}.
\end{aligned}$$

Thus the number of cases to fill i positions with at most t hops is $\binom{t}{i}$.

(Another way of proving this part is given below. Consider the problem of distributing m or fewer balls into b boxes. There is a one to one correspondence between a configuration of this type and a binary vector of length $b+m$ with weight b; in a given configuration, if the j-th box contains s balls, then the equivalent binary vector contains s 0's between the $(j-1)$-st and j-th 1's. For example, when $m = 7$ and $b = 3$, a binary vector 0010110000 is equivalent to the configuration with the first box containing 2 balls, the second box with one ball, and the third box with zero ball (the other four balls are not used). Thus the number of ways of distributing m or fewer identical balls into b boxes is $\binom{m+b}{b}$.

Now consider our problem of partitioning integer t into i parts with no part equal to zero. This problem is equivalent to distributing t balls into i boxes with no box containing empty ball. First distribute one ball to each of i boxes. There are $(t-i)$ balls available. Thus the number of ways of distributing $(t-i)$ or less balls into i boxes is $\binom{(t-i)+i}{i} = \binom{t}{i}$.)

(3) Since each position among the selected i positions can be plus or minus, the total possible cases are 2^i.

Therefore

$$|S_A| = 1 + \sum_{i=1}^{t} x_i = 1 + \sum_{i=1}^{min(t,n)} 2^i \binom{n}{i}\binom{t}{i}.$$

□

In equation (2.1), if the equality holds, then the placement is called *perfect.*

First some results related to perfect distance-1 placement are given and then perfect distance-t placements for $t \geq 2$ are considered. Some results from error correcting codes are used to obtain these results. The extension of these results for perfect placement in a torus is also discussed.

2.2.1. Perfect Distance-1 Placement in a k-ary n-cube

For $t = 1$, equation (2.1) becomes

$$M \geq \frac{N}{1+2n}$$

where N is the total number of nodes. Thus, for the perfect distance-1 placement, we must have

$$M = \frac{N}{1+2n} = \frac{k^n}{1+2n}. \tag{2.2}$$

The perfect distance-1 placement can be achieved using the Lee distance single error correcting code described below.

Let k be an odd integer and let $n = \frac{k^r-1}{2}$ for some integer r. The parity check matrix of a perfect Lee distance single error-correcting code is given by the H-matrix [22],

$$H = [\ V_1\ V_2\ \cdots\ V_n\ ]$$

where V_i is a $r \times 1$ vector with the radix-k representation of the number $(i + \lceil\frac{k-1}{2}\rceil(\frac{k^s-1}{k-1}))$ if $\lfloor\frac{k-1}{2}\rfloor(\frac{k^s-1}{k-1}) < i \leq \lfloor\frac{k-1}{2}\rfloor(\frac{k^{s+1}-1}{k-1})$. In other words, the first non-zero leading term in H can be only $\{1, 2, \cdots, \lfloor\frac{k-1}{2}\rfloor\}$, while the other elements can be any integers from the set $\{0, 1, \cdots, k-1\}$; i.e.,

$$H = \begin{bmatrix} 0\ 0\ \cdots & 0 & 0\ \cdots & 0 & \cdots & 0 & \cdots & \cdots\ \cdots & \cdots & 1\ \cdots & \lfloor\frac{k-1}{2}\rfloor \\ \vdots\ \vdots & & & & & & & & & & \\ 0\ 0\ \cdots & 0 & 0\ \cdots & 0 & \cdots\ \cdots & \cdots & 1\ \cdots & \lfloor\frac{k-1}{2}\rfloor & \cdots & & \\ 0\ 0\ \cdots & 0 & 1\ \cdots & 1 & 2\ \cdots & \lfloor\frac{k-1}{2}\rfloor & 0\ \cdots & k-1 & 0\ \cdots & & \\ 1\ 2\ \cdots & \lfloor\frac{k-1}{2}\rfloor & 0\ \cdots & k-1 & 0\ \cdots & k-1 & 0\ \cdots & k-1 & 0\ \cdots & & \end{bmatrix}.$$

Then an n digit radix-k vector, $A = (a_{n-1}a_{n-2}\cdots a_0)$, is a code word *iff* $AH^T = 0$.

Example 2.1 *Let* $k = 5$ *and* $r = 2$, *then* $n = \frac{5^2-1}{2} = 12$ *and*

$$H = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 \\ 1 & 2 & 0 & 1 & 2 & 3 & 4 & 0 & 1 & 2 & 3 & 4 \end{bmatrix}.$$

Since $n = 12$ *and* $r = 2$, *there are* 5^{10} *vectors orthogonal to* H. *These vectors form a perfect single-error correcting Lee distance code, and placing resources at nodes in a* C_5^{12} *corresponding to code words results in a solution to a perfect distance-*1 *problem.*

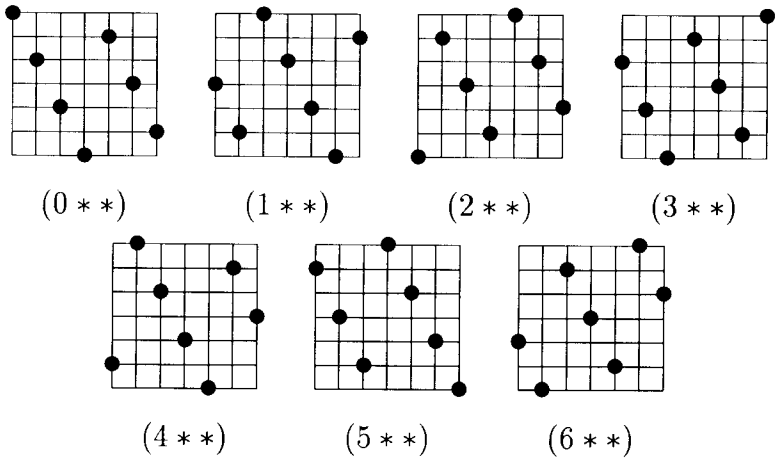


FIGURE 2.3. Perfect 1-hop placement in a 3D torus (C_7^3).

Example 2.2 *In a 3-dimensional torus of size* $7 \times 7 \times 7$, *we can have a perfect distance-1 placement using* 49 *spare nodes. The* H *matrix is given by* $H = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$. *Figure 2.3 shows this placement.*

A placement is called *quasi-perfect* distance-t placement if the maximum possible number of non-resource nodes are at a distance t or less from the resource nodes and each of the remaining non-resource node is at a distance $t+1$ from some resource node. When $n = \lfloor \frac{k-1}{2} \rfloor \frac{k^r-1}{k-1}$ for some even integer k, the code having the H matrix above results in a quasi-perfect distance-1 placement. In this case, $2nM$ non-resource nodes are at a distance one from some resource node and each of the remaining $N-(2n+1)M$ non-resource nodes are at a distance 2 from some resource nodes. Thus this quasi-perfect distance-1 placement is also optimal.

Example 2.3 *Let* $k = 4$ *and* $r = 2$, *then* $n = \lfloor \frac{4-1}{2} \rfloor \frac{4^2-1}{4-1} = 5$. *Thus we can have a quasi-perfect distance-1 placement in a* C_4^5, *and its parity check matrix is given by*

$$H = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 1 & 2 & 3 \end{bmatrix}.$$

Theorem 2.2 shown below gives further results on a perfect or quasi-perfect distance-1 placement.

Theorem 2.2 *A k-ary n-cube (C_k^n) has a perfect distance-1 placement for odd p, and a quasi-perfect distance-1 placement for even p, if*

$$k = pq \quad \text{for } p > 1,\ q \geq 1,$$
$$n = \lfloor \tfrac{p-1}{2} \rfloor \tfrac{p^r-1}{p-1} \quad \text{for } r \geq 1.$$

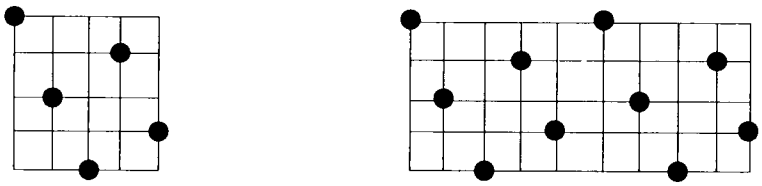
Proof: Suppose p is odd, then $n = \frac{p^r-1}{2}$. In the above equation if $q = 1$, then $k = p$. This will result in the single error correcting Lee distance code described above. For $q > 1$, first a perfect distance-1 placement using a single error correcting code can be obtained in a p-ary n-cube; then, this placement can be replicated q times along each dimension. This results in a perfect distance-1 placement in $k(= pq)$-ary n-cube. Similarly when p is even, it becomes a quasi-perfect distance-1 placement. □

Example 2.4 *This example illustrates the proof given in Theorem 2.2. The two dimensional torus of size* 10×10, C_{10}^2, *has a perfect distance-1 placement. This is because* $k = pq = 5 \times 2$ *and* $(2n+1) = (2 \times 2 + 1) = 5^1 = p^1$.

The H-matrix of the perfect distance-1 placement in C_k^n: Let $c = (c_1 c_2 \cdots c_i \cdots c_n)$ be a code word (resource-node in a C_k^n), and c_i be the i-th digit of the code word over Z_k^n. From Theorem 2.2, C_p^n must satisfy

$$(c \ mod \ p)\ H^T = 0$$

where H is the check matrix of C_p^n constructed in the manner of the previous section, and $c \ mod \ p = ((c_1 \ mod \ p)\ (c_2 \ mod \ p)\ \cdots\ (c_n \ mod \ p))$.



(a) C_5^2 (b) $T_{5,10}$

FIGURE 2.4. Perfect distance-1 placement in 2D tori

2.2.2. Perfect Distance-1 Placement in a Torus

As we can see, there exists a perfect distance-1 placement in 5×5, and 10×10 tori by Theorem 2.2. The extension of this theorem to a multidimensional torus is described here.

Theorem 2.3 *An n-dimensional torus,* $T_{k_1,k_2,\ldots,k_n} = C_{k_1} \times C_{k_2} \times \cdots \times C_{k_n}$, *has a (perfect) distance-1 placement if a k-ary n-cube,* C_k^n, *has a (perfect) distance-1 placement, where* $k = GCD(k_1, k_2, \cdots, k_n)$.

Proof: An n-dimensional torus, $T_{k_1,k_2,\ldots,k_n} = C_{k_1} \times C_{k_2} \times \cdots \times C_{k_n}$, is $\frac{k_i}{k}$ copies of C_k^n along the dimensions i, $i = 1, 2, \cdots, n$. By replicating the (perfect) distance-1 placement of a C_k^n with $\frac{k_i}{k}$ copies along dimension i, $i = 1, 2, \cdots, n$, a (perfect) distance-1 placement for the given torus is obtained. Thus, from Theorem 2.3 and the Lee distance code described in Section 2.2.1, if $(2n+1)|k_i$ for $i = 1, 2, \cdots, n$, then we have a perfect distance-1 placement in $T_{k_1,k_2,\cdots,k_n}$. □

Figure 2.4 shows an example of a perfect distance-1 placement in a torus $T_{5,10}$ by replicating the placement in a C_5^2. Similarly, we can also obtain a perfect distance-1 placement in 3-dimensional tori such as $T_{7,7,7}$, $T_{7,7,14}$, $T_{7,14,14}$, $T_{21,21,28}$, etc,

Can we obtain a perfect distance-1 placement in a torus when all dimensions are not divisible by $2n+1$? Even though we do not have a general result, Theorem 2.4 shows this is not possible, for a two dimensional torus.

Theorem 2.4 *There does not exist a perfect distance-1 placement in a 2D torus,* T_{k_1,k_2}, *if* $5 \not| k_1$ *or* $5 \not| k_2$.

Proof: If a perfect distance-1 placement in a 2D torus (T_{k_1,k_2}) exists, then

$$M = \frac{k_1 k_2}{1+2n} = \frac{k_1 k_2}{5}. \tag{2.3}$$

Note that this is only a necessary condition but not a sufficient condition. To satisfy the above equation, k_1 or k_2 must be divisible by 5. If neither k_1 nor k_2 is a multiple of 5 then M is not an integer. Suppose one of them is not a multiple of 5, say, $k_1 = 5p + r$ (columns), and $k_2 = 5q$ (rows), where $1 \leq r \leq 4$. Consider row i, which forms a cycle of length k_1, and let m_i be the number of resources in a row i. $3m_i$ nodes in this row are covered by the m_i resources; therefore $k_1 - 3m_i$ nodes must be covered by its neighboring rows. The non-covered nodes are divided into m_i subgroups, and at least 3 non-covered nodes among the $k_1 - 3m_i$ nodes must be non-consecutive. The reason is the following.

Suppose 3 non-covered nodes within a box are consecutive as in Figure 2.5. Note that, the black circles represent resource nodes in this figure. Since nodes at $(i, j-3)$ and $(i, j+3)$ have resources and nodes at $(i, j-2)$ and $(i, j+2)$ are covered by the same row, nodes at $(i, j-1)$, (i, j), $(i, j+1)$ must be covered by neighboring rows. Suppose node $(i, j-1)$ is covered by a resource in row $i-1$ and node $(i, j+1)$

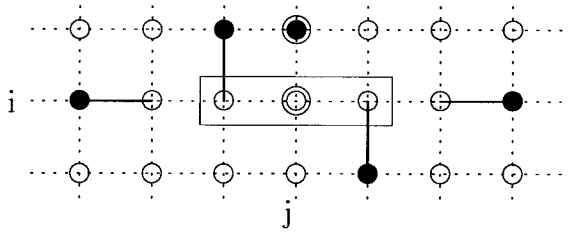


FIGURE 2.5. Non-existence of the perfect distance-1 placement in a 2D torus

is covered by a resource in row $i+1$, then node (i,j) can not be covered by any other resource nodes. This implies

$$\begin{aligned} k_1 - 3m_i &\leq 2m_i \\ m_i &\geq \frac{k_1}{5} = \frac{5p+r}{5} \\ \text{Thus} \quad m_i &\geq p+1. \end{aligned}$$

This implies that at least $p+1$ resources are placed in each row. Therefore the total number of resources is at least $k_2(p+1) = 5q(p+1)$ because the total number of rows is $k_2 = 5q$. However $5q(p+1) = 5pq + 5q > 5pq + qr = M$ which contradicts the equation (2.3). □

The implication of Theorem 2.4 is that we can not obtain a perfect distance-1 placement in a $T_{5,6}$, a $T_{5,7}$, a $T_{5,8}$, etc, even though each torus satisfies the necessary condition given in equation (2.3). Theorem 2.5 follows from the discussion thus far.

Theorem 2.5 *A two dimensional torus T_{k_1,k_2} has a perfect distance-1 placement* iff $5|k_1$ *and* $5|k_2$.

2.2.3. Perfect Distance-t Placement in a k-ary n-cube

When k is a prime, the Lee distance 1-error correcting code described in the previous section is equivalent to a *negacyclic* code [15, ch. 9]. Negacyclic codes correcting t errors, where $2t - 1 < k$, can be constructed. It would be useful if, for example, the 2-error correcting negacyclic code in Z_5^{12} would result in a perfect distance-2 placement in a C_5^{12}. Unfortunately, in general, this it is not the case. However, for some cases we can obtain perfect distance-t placement as described below.

Perfect distance-t placement in a ring.

A ring of size k is a k-ary 1-cube, say C_k. The perfect distance-t placement in a ring is trivial. If $(2t+1)|k$, then there exists a perfect distance-t placement, and its parity check matrix is $H = [\ 1\ ]$ over Z_{2t+1}.

Distance-t placement for $n = 2$.

When $n = 2$, equation (2.1) becomes $M \geq \frac{k^2}{2t^2+2t+1}$. If $(2t^2+2t+1)|k$, then there is, in fact, a perfect distance-t placement. This placement can be obtained using the t-error correcting codes described in [15]. The H and G matrices of this code over $Z_{2t^2+2t+1}^2$ are given by

$$H = [\ 2t^2 \quad 1\ ],$$
$$G = [\ 1 \quad (2t+1)\ ].$$

Example 2.5 *When $t = 2$, the parity check matrix is $H = [8\ 1]$ and $k = 2t^2 + 2t + 1 = 13$. The set of resource nodes are obtained by iG for $i = 0, 1, \cdots, 12$. In fact,*

the perfect distance-2 placement in a C_{13}^2 shown in Figure 2.1 is obtained using this code. Similarly a perfect distance-3 placement in a C_{25}^2 can be obtained.

Distance-t placement for $n > 2$.

- Case $t = 2$ and $k = 3$: The sphere packing bound of radius 2 is exactly $2n^2+1$. In order to have a perfect double error correcting code, this number, $2n^2 + 1$, must equal 3^x for some integer x. The only possible solutions are $n = 1$, 2, or 11. However, for $n = 1$ or 2, there is no perfect double error correcting code. When $n = 11$, this becomes the Golay code [66], which is a perfect double error correcting code. Using the Golay code, we can obtain a perfect distance-2 placement in C_{3i}^{11}, for $i \geq 1$.

- Case $t = 2$ and $k > 3$: The sphere packing bound of radius 2 is exactly $2n^2 + 2n + 1$. For the perfect double error correcting code, the equation $2n^2 + 2n + 1 = k^x$ must hold for some x. Some known solutions of the equation are $(n, k) = (3, 5)$, $(119, 13)$, and $(4059, 5741)$. When $(n, k) = (3, 5)$, it is known that there exists no perfect double error correcting code. For the other two cases, it is not known whether or not a perfect double error correcting code exists [66, 72].

- Case $t > 2$ and $k > 3$: Golomb and Welch [52] conjectured that no perfect t-error correcting code exists for these cases. Thus, there may not be a perfect distance-t placement in these cases.

2.2.4. Perfect Distance-t Placement in a Torus

Similar to the extension of the perfect distance-1 placement, the results of the perfect distance-t placement in a k-ary n-cube can be extended to a multidimensional torus.

Theorem 2.6 *(Generalization of Theorem 2.3) An n-dimensional torus $T_{k_1,k_2,\ldots,k_n} = C_{k_1} \times C_{k_2} \times \cdots \times C_{k_n}$ has a (perfect) distance-t placement if the k-ary n-cube, C_k^n, has a (perfect) distance-t placement, where $k = GCD(k_1, k_2, \cdots, k_n)$.*

Proof: It is same as the proof of Theorem 2.3. □

Lemma 2.1 *A 2D torus T_{k_1,k_2} has a perfect distance-t placement if $k|GCD(k_1, k_2)$, where $k = (2t^2 + 2t + 1)$.*

Proof: Since $k = (2t^2+2t+1)$, a C_k^2 has a perfect distance-t placement as described in Section 2.2.3. We replicate this placement with $\frac{k_1}{k}$ copies in one dimension and $\frac{k_2}{k}$ copies in the other dimension. □

For example, we can have a perfect distance-3 placement in a torus of size 25×25, 25×50, 50×50, 75×50, 100×100, etc.

2.3. j-adjacency Placement

In the previous section, methods to place resources were discussed so that a non-resource node has access to one and only one resource within a distance t. In some systems, like highly fault tolerant systems, it might be desirable to have access to multiple resource copies from any node. The j-adjacency placement, where each non-resource node is adjacent to at least j resource nodes, achieves this goal. Some results obtained here for k-ary n-cube are similar to that of [82]; however, the quasi-perfect placements and the placements in a torus are new.

2.3.1. Perfect j-adjacency Placement in a k-ary n-cube

Definition 2.1 *A j-adjacency placement is said to be perfect* iff *no two resource nodes are adjacent, and every non-resource node is adjacent to exactly j resource nodes.*

Lemma 2.2 *A C_k^n has a perfect 1-adjacency placement if a C_k^n has a perfect distance-1 placement.*

Lemma 2.2 implies that the perfect 1-adjacency placement is equivalent to the perfect distance-1 placement. Therefore, Theorem 2.2 is applied directly to the perfect 1-adjacency placement. For example, perfect 1-adjacency placements in C_5^2 and C_7^3 can be obtained using a Lee distance single error correcting code.

Lemma 2.3 *A ring C_4^1 has a perfect 2-adjacency placement, and its check matrix is $H = [\ 2\ ]$ over Z_4.*

Figure 2.7(a) illustrates this 2-adjacency placement in a ring.

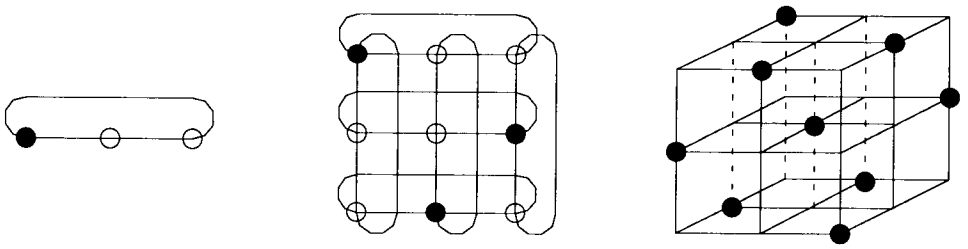
Theorem 2.7 *If a C_k^n has a perfect 1-adjacency placement, then a C_k^{jn} has perfect j-adjacency placement, and its parity check matrix H is*

$$H = [\ \underbrace{H_0\ H_0\ \cdots H_0}_{j\ times}\ ]$$

where H_0 is the check matrix of a perfect 1-adjacency placement in a C_k^n.

Proof: The proof is similar to the one given in [82]. However, H_0 can be taken as the H-matrix of a single error correcting Lee distance code. □

Example 2.6 *Figure 2.6 illustrates Theorem 2.7. Since a C_3^1 has a perfect 1-adjacency placement, C_3^2 and C_3^3 have perfect 2-adjacency and perfect 3-adjacency placements respectively.*



(a) 1-adjacency in C_3^1 (b) 2-adjacency in C_3^2 (b) 3-adjacency in C_3^3

FIGURE 2.6. Perfect j-adjacency placements

Theorem 2.8 *(Generalization of Theorem 2.7). A C_k^{jn} has a perfect $(j \times j_0)$-adjacency placement if a C_k^n has a perfect j_0-adjacency placement. Its parity check matrix H is given by*

$$H = [\underbrace{H_0\ H_0\ \cdots H_0}_{j\ times}]$$

where H_0 is the check matrix of a perfect j_0-adjacency placement in C_k^n.

Proof: To prove this theorem, we must prove that (a) no two resource nodes are adjacent, and (b) each non-resource node is adjacent to exactly $(j \times j_0)$ resource nodes.

Let $U = (U_1 U_2 \cdots U_j)$, where $U \in Z_k^{jn}$ and $U_i \in Z_k^n$. Suppose U and V are two resource nodes in a C_k^{jn}, i.e., $UH^T = VH^T = 0$. If they were adjacent, i.e., $D_L(U,V) = 1$, then $U = V + e_{l_i}$ where e_{l_i} is a unit vector or its inverse in Z_k^{jn}. Thus $UH^T = (V+e_{l_i})H^T = VH^T + e_{l_i}H^T = e_{l_i}H^T \neq 0$ which contradicts the assumption. Therefore no two resource nodes are adjacent in a C_k^{jn}.

Consider a non-resource node U in a C_k^{jn}, i.e., $UH^T = (U_1 U_2 \cdots U_j)[\ H_0 H_0 \cdots H_0\ ]^T = (U_1 + U_2 + \cdots + U_j)H_0^T = U'H_0^T \neq 0$,

where $U' = (U_1 + U_2 + \cdots U_j)$. Moreover, because H_0 is a parity check matrix for a perfect j_0-adjacency placement, we can find exactly j_0 resource nodes $V'_1, V'_2, \cdots, V'_{j_0}$ in a C_k^n such that $U' = V'_1 + e'_{l_1} = V'_2 + e'_{l_2} = \cdots V'_{j_0} + e'_{l_{j_0}}$, and $V'_i H_0^T = 0$ for $i = 1, 2, \cdots, j_0$, where e'_{l_i} is a unit vector or its inverse in Z_k^n. From the construction of H, each column of H_0 is replicated j times, and so $e'_{l_i} H_0^T = e_{l_1} H^T = e_{l_2} H^T = \cdots e_{l_j} H^T$, where e_{l_i} is a unit vector in Z_k^{jn}. Thus we can find $UH^T = e_{l_1} H^T = e_{l_2} H^T = \cdots = e_{l_j} H^T$. Each $e_{l_i} H^T$ has exactly j_0 nodes $V'_{i\,1}, V'_{i\,2}, \cdots, V'_{i\,j_0}$ such that $V'_{i\,1} H_0^T = V'_{i\,2} H_0^T = \cdots = V'_{i\,j_0} H_0^T = 0$. Therefore each non-resource node in a C_k^{jn} is adjacent to exactly $(j \times j_0)$ resource nodes. □

Example 2.7 *A C_4^2 has a perfect 4-adjacency placement since a C_4^1 has a perfect 2-adjacency placement. Their parity check matrices are $H_0 = [\ 2\ ]$ and $H = [\ 2\ 2\ ]$ over Z_4^2 respectively.*

2.3.2. Quasi-perfect j-adjacency Placement in a k-ary n-cube

Definition 2.2 *A placement is said to be a* quasi-perfect j-adjacent *placement if a non-resource node is adjacent to at least j but no more than $j+1$ resources, and no two resource nodes are adjacent.*

Theorem 2.9 *A C_k^{jn+m} has a quasi-perfect j-adjacency placement if a C_k^n has a perfect distance-1 placement, where $1 \leq m < n$.*

Proof: Similar to Theorem 2.8, we can construct a check matrix H_0 for the perfect distance-1 placement in C_k^n, and extend the check matrix H_0 into $H^j = [\ H_0\ H_0\ \cdots\ H_0\ ]$. Consider a check matrix $H = H^j || [V_1\ V_2\ \cdots\ V_m]$, where the V_i's are m column vectors of H_0, and $V_{i_1} \neq V_{i_2}$, for $i_1, i_2 = 1, \cdots, m$. Any codeword orthogonal to H is adjacent to at least j other codewords as in Theorem 2.8, but

are adjacent to no more than $j+1$ codewords, Further, no two resource nodes are adjacent. □

For example, we can obtain a quasi-perfect 1-adjacency placement in a C_5^3 from a perfect distance-1 placement in a C_5^2 because $H_0 = [\ 2\ 1\ ]$ for C_5^2, and $H = H_0 || [\ 1\ ] = [\ 2\ 1\ 1\ ]$.

2.3.3. j-adjacency Placement in a Torus

Lemma 2.4 *An n-dimensional torus* $T_{k_1,k_2,\cdots,k_n} = C_{k_1} \times \cdots \times C_{k_n}$ *has a (perfect/quasi-perfect)* j*-adjacency placement if a* C_k^n *has a (perfect/quasi-perfect)* j*-adjacency placement, where* $k = GCD(k_1, k_2, \cdots, k_n)$.

Proof: Similar to the construction of a (perfect) distance-t placement in a torus, we can extend C_k^n $\frac{k_i}{k}$ times along dimension i, $i = 1, 2, \cdots, n$. The resulting torus $T_{k_1,k_2,\cdots,k_n}$ will have the same (perfect/quasi-perfect) j-adjacency as that of the C_k^n. □

2.4. Generalized Placement

From practical point of view, it is desirable to place the resources so that a non-resource node can reach several resources within a given distance. Thus, the generalized placement discussed in this section is a combination of the distance-t problem and the j-adjacency problems discussed in the previous sections.

The concept of the *covering radius* of a linear code [24, 34, 55, 91, 92], which is defined below, is useful in studying this placement problem.

Definition 2.3 Covering Radius for Lee distance: *The covering radius r of a linear code C is defined as the maximum distance from any non-code word to the*

nearest code word. i.e.,

$$r = \max_{\forall x \notin C} \{\min_{\forall c \in C} D_L(c, x)\}.$$

If a code has a covering radius r, then it is called a *r-covering code.* This r-covering code is different from a r-error correcting code [55].

- In a t-error correcting code, the sets of non-code words at a distance t or less from each code word are mutually disjoint; although, there may be non-code words which are at a distance $t + 1$ or more from one or more code words.

- In a t-covering code, all non-code words are at a distance at most t from one or more code words.

Note that from the resource placement point of view, the parameter, *covering radius*, is more appropriate than the *minimum distance* parameter. This is because if the resource placement is done using a t-error correcting linear code, then it is not guaranteed that all non-resource nodes are within a distance t from some resource node. On the other hand, if we use a linear code with the covering radius t, then all non-resource nodes are at a distance t or less from some resource node. Note that if the covering radius of a code is t, then it can correct at most t-errors; this implies the minimum distance of the code is $2t + 1$ or less. Furthermore, if a code is capable of correcting $t_1 \leq t$ errors, it is not guaranteed that the covering radius of the code is t. Thus, from the resource placement point of view, for a given r and n, we need to choose a code of length n, with covering radius r, and the number of code words as small as possible.

In the literature, even though some results on the covering radius of linear codes have been developed for Hamming distance, very little is known about Lee distance codes. However, in the following, we present some results.

First, we define the direct sum of H-matrices based on Lee distance for a mixed radix vector. This definition is useful in finding a generalized resource placement in a k-ary n-cube or a torus.

Definition 2.4 Direct Sum: *Let $n = n_1 + n_2$. Let H_1 and H_2 be the check matrices of two codes C_1 and C_2 with covering radii r_1 and r_2, respectively. Then the direct sum, H, of H_1 and H_2 is defined as*

$$H = H_1 \uplus H_2 = \begin{bmatrix} H_1 & 0 \\ 0 & H_2 \end{bmatrix}$$

where elements in H_1 and H_2 are over $Z_{k_1}^{n_1}$ and $Z_{k_2}^{n_2}$ respectively.

The H matrix defined above gives a partity check matrix for a new code C with length $n = n_1 + n_2$, the first n_1 digits over $Z_{k_1}^{n_1}$, and the last n_2 digits over $Z_{k_2}^{n_1}$. For any code word $x \in C$, $xH^T = 0$. In calculating xH^T, the first n_1 and the last n_2 digits are computed *modulo* k_1 and *modulo* k_2 respectively.

Theorem 2.10 *The direct sum of H_1 and H_2, $H = H_1 \uplus H_2$, has the covering radius $r_1 + r_2$, where r_1 and r_2 are the covering radii of H_1 and H_2 respectively.*

Proof: Let the dimension of H_1 and H_2 be n_1 and n_2 respectively, and a non-code word $W = (XY) = (x_1 x_2 \cdots x_{n_1} y_1 x_2 \cdots y_{n_2})$, where $x_i \in Z_{k_1}$ for $i = 1, \cdots, n_1$ and $y_j \in Z_{k_2}$ for $j = 1, \cdots, n_2$. Because the covering radii of H_1 and H_2 are r_1 and r_2, we can choose a code word $C = (AB) = (a_1 a_2 \cdots a_{n_1} b_1 b_2 \cdots b_{n_2})$ where $a_i \in Z_{k_1}$ for $i = 1, \cdots, n_1$, and $b_j \in Z_{k_2}$ for $j = 1, \cdots, n_2$, such that $CH^T = C(H_1 \uplus H_2)^T = 0$, (i.e., $AH_1^T = 0$ and $BH_2^T = 0$), and $D_L(X, A) \leq r_1$ and $D_L(Y, B) \leq r_2$. Therefore $D_L(W, C) = D_L(X, A) + D_L(Y, B) \leq r_1 + r_2$. □

Suppose there exists a parity check matrix $H = H_1 \uplus H_2$, where entries in H_1 and H_2 are over $Z_{k_1}^{n_1}$ and $Z_{k_2}^{n_2}$, and their lengths are n_1 and n_2 respectively. A

node $C = (a_1 a_2 \cdots a_{n_1} b_1 b_2 \cdots b_{n_2})$ in $C_{k_1}^{n_1} \times C_{k_2}^{n_2}$ is a resource node if

$$C(H_1 \uplus H_2)^T = [(a_1 a_2 \cdots a_{n_1}) H_1^T \ mod\ k_1 \quad (b_1 b_2 \cdots b_{n_2}) H_2^T \ mod\ k_2] = 0.$$

Define the direct sum of two k-ary n-cubes $C_{k_1}^{n_1}$ and $C_{k_2}^{n_2}$ as $C_k^n = C_{k_1}^{n_1} \uplus C_{k_2}^{n_2}$, where $n = n_1 + n_2$ and $k = LCM(k_1, k_2)$. For example $C_6^2 \uplus C_4^1 = C_{12}^3$. Note that $C_k^{n_1} \uplus C_k^{n_2} = C_k^{n_1} \times C_k^{n_2}$, but $C_{k_1}^{n_1} \uplus C_{k_2}^{n_2} \neq C_{k_1}^{n_1} \times C_{k_2}^{n_2}$ if $k_1 \neq k_2$.

Lemma 2.5 *The direct sum of $C_{k_1}^{n_1}$ and $C_{k_2}^{n_2}$, $C_k^n = C_{k_1}^{n_1} \uplus C_{k_2}^{n_2}$, has a distance-$(t_1+t_2)$ placement if $C_{k_1}^{n_1}$ and $C_{k_2}^{n_2}$ have distance-t_1 and distance-t_2 placements respectively.*

Proof: Let $k = LCM(k_1, k_2)$, and $n = n_1 + n_2$. Let H_1 and H_2 be the parity check matrices used in allocating resources in $C_{k_1}^{n_1}$ and $C_{k_2}^{n_2}$. Since $C_{k_1}^{n_1}$ and $C_{k_2}^{n_2}$ have a distance-t_1 and t_2 placement, the covering radii of H_1 and H_2 are t_1 and t_2. From Theorem 2.10, the direct sum H of H_1 and H_2, $H = H_1 \uplus H_2$, must have a covering radius of $t_1 + t_2$. Thus, in a C_k^n placing resources at nodes which are orthogonal to H results in a distance-$(t_1 + t_2)$ placement. □

Example 2.8 *Using Theorem 2.10 and Lemma 2.5, we can construct higher dimensional k-ary n-cubes and tori having higher covering radius. For example, from the two k-ary n-cubes C_5^1 and C_{13}^2 with covering radii 2 and 2 respectively, we can construct $C_5^1 \uplus C_{13}^2 = C_{LCM(5,13)}^{1+2} = C_{65}^3$ having a covering radius $r = 4$. Its parity check matrix is*

$$H = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 8 & 1 \end{bmatrix}.$$

We now consider a placement that tolerates resource failures. In this situation, several resources must be placed within a given distance from a non-resource node. To solve this problem, we generalize the terminology of *adjacency* of a linear code.

Definition 2.5 Adjacency of a linear code: *The j-adjacency with covering radius r of a linear code C is defined as the minimum number of code words from any non-code word within a distance r.*

The direct sum of the parity check matrix has the following property regarding the adjacency of the code.

Theorem 2.11 *If H_1 and H_2 have j_1 and j_2-adjacencies with covering radii r_1 and r_2 respectively, then the direct sum H of H_1 and H_2, $H = H_1 \uplus H_2$, satisfies the following properties:*

1) *$(j_1 \times j_2)$-adjacency with a covering radius $r_1 + r_2 + 1$, and*

2) *$\min(j_1, j_2)$-adjacency with a covering radius $r_1 + r_2$.*

Proof: Let the dimension of H_1 and H_2 be n_1 and n_2 respectively. Consider a node $W = (XY) = (x_1 x_2 \cdots x_{n_1} y_1 y_2 \cdots y_{n_2})$, where $x_i \in Z_{k_1}$ for $i = 1, \cdots, n_1$ and $y_j \in Z_{k_2}$ for $j = 1, \cdots, n_2$. If $XH_1^T = 0$ and $YH_2^T = 0$, then W is a resource node. On the other hand, suppose W is not a resource node. Because the covering radii of H_1 and H_2 are r_1 and r_2, and H_1 and H_2 have j_1 and j_2-adjacencies, there are at least j_1 n_1-digit vectors,$A_1, A_2, \cdots, A_{j_1}$, which satisfy $D_L(X, A_1) \leq r_1$, $D_L(X, A_2) \leq r_1$, $\cdots$, and $D_L(X, A_{j_1}) \leq r_1$, where $A_1 H_1^T = A_2 H_1^T = \cdots = A_{j_1} H_1^T = 0$ and $A_i \neq A_l$ for $i \neq l$. Similarly, there are j_2 n_2-digit vectors, $B_1, B_2, \cdots, B_{j_2}$, which satisfy $B_1 H_2^T = B_2 H_2^T = \cdots = B_{j_2} H_2^T = 0$, within a distance r_2 from Y. Now consider the following cases.

1. **Case $XH_1^T \neq 0$ and $YH_2^T \neq 0$:**

   $D_L(XY, A_i B_l) = D_L(X, A_i) + D_L(Y, B_l) \leq r_1 + r_2$ for $i = 1, \cdots, j_1$ and $l = 1, \cdots, j_2$. It has $(j_1 \times j_2)$-adjacency with a covering radius $r_1 + r_2$. Obviously, (XY) has $\min(j_1, j_2)$-adjacency with a covering radius $r_1 + r_2 + 1$.

2. **Case $XH_1^T \neq 0$ and $YH_2^T = 0$:**

   $D_L(XY, A_iB_l) = D_L(X, A_i) + D_L(Y, B_l) = D_L(X, A_i) \leq r_1$ for $i = 1, \cdots, j_1$ and some l. It has j_1-adjacency with a covering radius $r_1 \leq r_1 + r_2$. Furthermore, we can find Y' such that $D_L(Y', Y) = 1$ and $Y'H_2^T \neq 0$. From the previous case, we know (XY') has $(j_1 \times j_2)$-adjacency with a covering radius $r_1 + r_2$. Therefore (XY) has $(j_1 \times j_2)$-adjacency with a covering radius $r_1 + r_2 + 1$.

3. **Case $XH_1^T = 0$ and $YH_2^T \neq 0$:**

   This is similar to case 2. It has j_2-adjacency with a covering radius $r_1 + r_2$, and $(j_1 \times j_2)$-adjacency with a covering radius $r_1 + r_2 + 1$.

□

Theorem 2.12 *Suppose $C_{k_1}^{n_1}$ and $C_{k_2}^{n_2}$ have j_1 and j_2-adjacencies with distance-t_1 and t_2 respectively. Then $C_{k_1}^{n_2} \times C_{k_2}^{n_2}$ and $C_k^n = C_{k_1}^{n_2} \uplus C_{k_2}^{n_2}$ have*

(1) $(j_1 \times j_2)$-adjacency with distance-$(t_1 + t_2 + 1)$, and

(2) $\min(j_1, j_2)$-adjacency with distance-$(t_1 + t_2)$.

Proof: Let $n = n_1 + n_2$, and H_1 and H_2 be the corresponding parity check matrices used to place resources in $C_{k_1}^{n_1}$ and $C_{k_2}^{n_2}$ respectively. Since $C_{k_1}^{n_1}$ and $C_{k_2}^{n_2}$ have j_1 and j_2-adjacencies with distance-t_1 and t_2 placement, the covering radii of H_1 and H_2 are t_1 and t_2, and H_1 and H_2 have j_1 and j_2-adjacencies. From Theorem 2.11, the direct sum H of H_1 and H_2, $H = H_1 \uplus H_2$, must have $(j_1 \times j_2)$-adjacency with a covering radius of $t_1 + t_2 + 1$, and $\min(j_1, j_2)$-adjacency with a covering radius of $t_1 + t_2$. Thus, placing resources at nodes in a $C_{k_1}^{n_2} \times C_{k_2}^{n_2}$ corresponding to the code words of H results in a $(j_1 \times j_2)$-adjacency with a distance-$(t_1 + t_2 + 1)$ placement, and a $\min(j_1, j_2)$-adjacency with a distance-$(t_1 + t_2)$ placement. Moreover, let $k =$

(a) C_4^1 (b) C_3^2

FIGURE 2.7. Illustration of generalized placement from two k-ary n-cubes

$LCM(k_1, k_2)$. If we replicate $C_{k_1}^{n_1} \times C_{k_2}^{n_2}$ with the resource nodes $\frac{k}{k_1}$ times along the first n_1 dimensions and $\frac{k}{k_2}$ times along the last n_2 dimensions respectively, we get a $C_k^n = C_{k_1}^{k_2} \uplus C_{k_2}^{n_2}$ which satisfies the results mentioned above. □

Example 2.9 *As shown in Figure 2.7,* C_4^1 *and* C_3^2 *have 2-adjacencies with distance-1 placement. Thus* $C_4^1 \times C_3^2 = T_{4,3,3}$ *has a 2-adjacency with distance-2 placement, and a 4-adjacency with distance-3 placement. This is shown in Figure 2.2. Its parity check matrix is*

$$H = H_1 \uplus H_2 = \begin{bmatrix} H_1 & 0 \\ 0 & H_2 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \end{bmatrix}.$$

Table 2.1 shows some examples of generalized placement in a torus for the dimensions, $n = 1, 2, 3$. The symbol + indicates the perfect placement.

n	j	$(t=1)$	$(t=2)$	$(t=3)$	$(t=4)$		t
1	1	${T_3}^+$	${T_5}^+$	${T_7}^+$	${T_9}^+$	$\cdots$	${T_{2t+1}}^+$
	2	${T_4}^+$	${T_6}^+$	${T_8}^+$	${T_{10}}^+$		${T_{2t+2}}^+$
2	1	${T_{5,5}}^+$	${T_{13,13}}^+$	${T_{25,25}}^+$	${T_{41,41}}^+$	$\cdots$	${C^2_{2t^2+2t+1}}^+$
	2	${T_{3,3}}^+$	$T_{4,4}$	$T_{4,6}$	$T_{6,8}$	$\cdots$	$T_{2t_1+2,2(t-t_1)+2}$
	4	${T_{4,4}}^+$	–	–	$T_{4,6}$		
3	1	${T_{7,7,7}}^+$	$T_{3,5,5}$	$T_{3,13,13}$	$T_{3,25,25}$	$\cdots$	$T_{2(t-t_1)+1} \times C^2_{2t_1^2+2t_1+1}$
	2	–	$T_{4,3,3}$	$T_{6,3,3}$	$T_{6,4,4}$		$T_{2t_1+2,2t_2+2,2t_3+2}$, where $t_1+t_2+t_3=t$
	4	–	–	$T_{4,3,3}$	$T_{6,3,3}$		$T_{2t_1+2,2t_2+2,2t_3+2}$, where $t_1+t_2+t_3=t-1$
	6	${T_{4,4,4}}^+$					

TABLE 2.1. Generalized (j-adjacency with distance-t) placements

3. SPARE PROCESSOR ALLOCATION FOR FAULT-TOLERANCE

As the complexity of the system increases, the possibility of the system failure increases. Thus, in the case of some processor failures, it is desirable to have a reconfiguration scheme to preserve the functionality as well as the logical and physical topology of the system. The techniques to provide the fault-tolerance of the system can be categorized into two ways.

The first approach attempts to bypass the faulty components without adding extra spare nodes or links. The goal of this scheme is to obtain the same functionality with a reasonable slowdown factor [80, 27]. However, this approach may limit the usable number of healthy components in the system.

The second approach uses the redundancy of nodes or links. These schemes try to replace faulty components with alternative healthy components to provide the same functionality with no slowdown or minimal slowdown [26, 64, 94]. Until recent years, hardware solutions to this problem required excessive spare links or nodes. Recently, routing strategies based on circuit switching have been adapted to the multicomputer interconnection [19, 1, 8, 16, 32, 43, 42, 44, 56, 60, 70, 74, 77], and this circuit switching technique reduces the message delays for multiple hop messages as much as or only sightly greater than those for the single hop messages. Banerjee and Peercy [13] showed the feasibility of this approach.

This chapter also follows the second approach with the local spare replacement scheme. The spare processors in the local spare replacement scheme are near-uniformly distributed over the system, and a small group of processors share the spare processors. i.e., a faulty processor can be replaced only by a local spare processor. The local spare replacement [96, 3, 4] gives us more cost-effectiveness compared

to the global replacement [27, 49, 58, 47, 48]. The main goal of this chapter is to find a node configuration using the minimum number of spare processors based on the given design constraints. The proposed method, in a way, generalizes the scheme given in [13, 84, 71].

We use the standard graph model for the interconnection networks of the parallel computer [26]; in this model, a node consists of a processing element and communication subsystem, and an edge represents the connection between two nodes. We assume the spare processors are attached at the specific node (S-node), as in [13]. The S-node contains a normal processor, a spare processor, and its related communication subsystem. The other normal nodes are called P-nodes. The goal is to design a system, where each node has a spare processor or is at a distance of at most t from exactly one node having a spare processor. Thus, the spare processor allocation problem turns out to be one form of the resource placement problems [10–12] because we can treat the S-node as a resource. This type of problem is called the distance-t (or t-hop) problem.

For example, consider a torus with size 25×25. If spare processors (S-nodes) are placed on the solid circle nodes in Figure 3.1, every processor (P-node) can access one unique spare processor at a distance of at most 3. In fact, from a S-node, there are 1, 4, 8, and 12 normal processing nodes (P-nodes) at a distance 0, 1, 2, and 3 respectively. Thus a spare processor covers 25 normal processing nodes. Since there are 25 spare processors in the system, this placement results in a perfect 3-hop placement.

This chapter contains the t-hop placement method and its reconfiguration algorithm for the 2D torus.

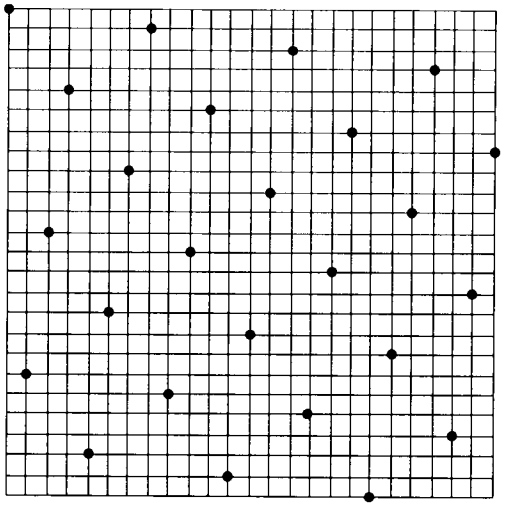


FIGURE 3.1. A perfect spare processor placement in a 25×25 torus.

3.1. t-Hop Spare Processor Placement

3.1.1. 1-Hop Placement in Arbitrary 2D Torus

In Chaper 2, we have shown that there exists a perfect 1-hop placement in a 2D torus, T_{k_1,k_2}, *if* $5|k_1$ and $5|k_2$. However, in general, it is desirable to have a 1-hop placement for 2D torus with arbitrary sizes.

To solve this problem, three basic configurations in $T_{3,3}$, $T_{4,4}$, and $T_{5,5}$ were built with 3, 4, and 5 resources respectively (see Figure 3.2). By replicating these basic configurations, we can obtain 1-hop placement for $T_{3i,3j}$, $T_{4i,4j}$, and $T_{5i,5j}$ with $3ij$, $4ij$, and $5ij$ resources respectively. For example, $T_{8,8}$ can have 1-hop placement using 16 resources by replicating $T_{4,4}$ two times for each dimension.

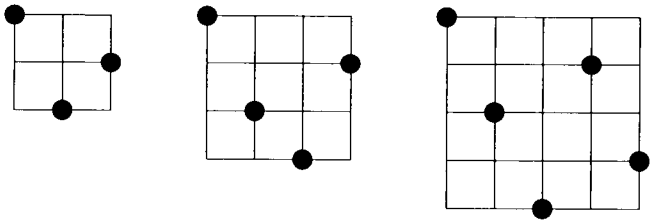


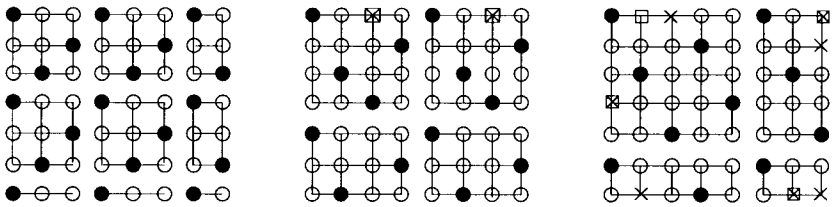
FIGURE 3.2. Basic 1-hop placement in 2D torus

Furthermore, we can find 1-hop placement by deleting rows and columns from an existing 1-hop placement. In this case, we need to add extra spare processors to fill the non-covered nodes. For example, $T_{7,8}$ can be built using $T_{9,9}$, $T_{8,8}$, or $T_{10,10}$ by deleting appropriate rows and columns (see Figure 3.3). In this figure, the mark X indicates the non-covered nodes, and the square ($\square$) indicates the location of the extra spare processors to cover the marked (X) nodes.

In general, a torus T_{k_1,k_2} requires $\frac{k_1k_2}{3} + \frac{k_1}{3} + \frac{k_2}{3} + \alpha_3$, $\frac{k_1k_2}{4} + \frac{k_1}{4} + \frac{k_2}{4} + \alpha_4$, and $\frac{k_1k_2}{5} + \frac{k_1}{5} + \frac{k_2}{5} + \alpha_5$ using the configurations 3×3, 4×4, and 5×5 respectively. Therefore, for large k_1 and k_2, it is better to use 5×5 configuration to minimize the number of spare processors.

3.1.2. t-Hop Placement in Arbitrary 2D Torus

Similar to the solution of 1-hop placement in an arbitrary 2D torus in Section 3.1.1, we can extend the ideas for t-hop placement in a 2D torus. For a given torus T_{k_1,k_2} and a distance t, where $k_1 = p(2t^2 + 2t + 1) + r$, and



(a) Using 3×3 (b) Using 4×4 (b) Using 5×5

FIGURE 3.3. Possible configurations for 1-hop placement in $T_{7,8}$

$k_2 = q(2t^2 + 2t + 1) + s$, we can divide the torus T_{k_1,k_2} into the $(p \times q)$ submeshes of $C^2_{2t^2+2t+1}$ and the remainder submeshes. Using the same ideas as in Section 3.1.1, first, build $(p+1) \times (q+1)$ blocks of $C^2_{2t^2+2t+1}$, with perfect t-hop placement; next remove $(2t^2 + 2t + 1) - r$ rows and $(2t^2 + 2t + 1) - s$ columns to make a torus if size $k_1 \times k_2$ (T_{k_1,k_2}); finally add the necessary spare processors to cover the nodes which are at a distance of more than t.

For example, $T_{24,20}$ can be built using $T_{13,13}$ for the 2-hop placement. $T_{24,20}$ can be divided into 13×13, 13×7, 11×13, and 13×7 submeshes, as in Figure 3.4.

3.2. Reconfiguration

When a processor fault occurs, the system needs to be restructured with the spare processor. The spare processor must be near to the faulty processor so that the communication delay is minimal after the replacement. Since the proposed spare processor allocation method is based on the linear codes, for a given faulty node, the address of the nearest S-node can be calculated using the parity check matrix

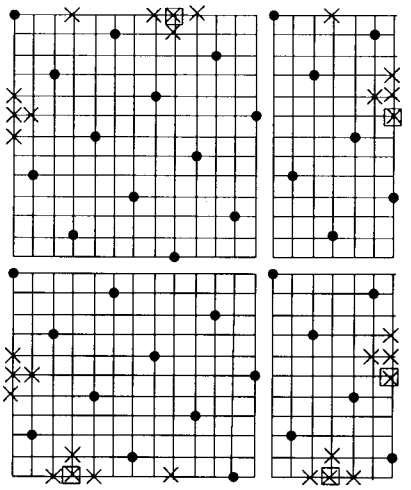


FIGURE 3.4. 2-hop placement in $T_{24,20}$

H; after this, a path from the nearest spare processor to this faulty processor can be constructed. This path plays key role in our proposed reconfiguration scheme.

3.2.1. Address of the Nearest Spare Processor

Let a spare processor location be $u = (u_1 u_2 \cdots u_n)$. From the design of spare processor methods, u must be orthogonal to H^T, i.e., $uH^T = 0$. Consider a node $v = (v_1 v_2 \cdots v_n)$, and $v = u + e$.

$$vH^T = (u + e)H^T = uH^T + eH^T = eH^T.$$

In other words, we can get the syndrome (vH^T) using the processor address and parity check matrix. Once the syndrome is found, the error vector (e) can also be found. Adding this error vector to the address of the faulty processor, the address of the nearest spare processor can be obtained.

Since 2D torus topology is more practical, we devote our attention in developing algorithms for this case. From Lemma 2.1, a $T_{k,k}$ has a perfect t-hop placement when $k = (2t^2 + 2t + 1)$; its parity check matrix is $H = [2t^2\ 1]$. Suppose a faulty processor address and its nearest spare processor address are $v = (v_1 v_0)$ and $u = (u_1 u_0)$ respectively. Let $e = (e_1 e_0) = v - u$, where $|e| = |e_1| + |e_0| \leq t$. Then

$$\begin{aligned} s &= vH^T = eH^T \\ &= (e_1(2t^2) + e_0) \bmod (2t^2{+}2t{+}1) \\ e_1(2t^2) + e_0 &= s \bmod (2t^2{+}2t{+}1). \end{aligned}$$

Let $a = 2t^2$ and $m = 2t^2 + 2t + 1$. Then we get $ae_1 + e_0 = s \bmod m$. Assume $s \geq 0$ because we can use $-e$ if $s < 0$. This equation can be modified as follows.

$$\begin{aligned} ae_1 + e_0 &= s \bmod m \\ ae_1^{(1)} + e_0 &= s^{(1)} \bmod m, e_1^{(1)} = e_1{-}1, s^{(1)} = s - a \\ ae_1^{(2)} + e_0 &= s^{(2)} \bmod m, e_1^{(2)} = e_1{-}2, s^{(2)} = s - 2a \\ &\vdots \\ ae_1^{(i)} + e_0 &= s^{(i)} \bmod m, e_1^{(i)} = e_1{-}i, s^{(i)} = s - ia \end{aligned}$$

Thus, if we recursively apply the formula until $|s^{(i)}| \leq t$, then we can get $e_1^{(i)} = 0$ and $e_0 = s^{(i)}$. Therefore the solution of the error vector is $e = (e_1 e_0) = (i\ s^{(i)})$. The algorithm in Figure 3.5 takes (t, s) as the inputs, and returns the error vector $e = (e_1 e_0)$. From the error vector, we can find the spare processor address as $u = (v - e)$. The time complexity of the algorithm is $O(t)$.

For example, we can obtain a perfect 3-hop placement in a $T_{25,25}$ with the parity check matrix $H = [2t^2\ 1] = [18\ 1]$. The nearest spare processor address of a P-node $v = (4, 15)$ in $T_{25,25}$ will be $u = (2, 14)$. This is because $vH^T = (4, 15)H^T = 12$. The algorithm **FindEV(3,12)** gives $e = (2, 1)$ as follows.

Algorithm FindEV(t,s)

begin

let $m = 2t^2 + 2t + 1$, and $a = 2t^2$.

if ($|s| \leq t$) **then**

$e = (0\ s)$

else if ($s < 0$) **then**

$e = -FindEV(t, -s)$

else

$s' = (s - a) \bmod m$

$e' = FindEV(t, s', e')$

$e = e' + (1\ 0)$

endif

return e

end

FIGURE 3.5. Finding the error vector in 2D torus

$$18e_1 + e_0 = 12 \bmod 25, \text{ since } a = 2t^2 = 18$$

$$18(e_1 - 1) + e_0 = (12 - 18) = -6 = 19 \bmod 25$$

$$18(e_1 - 2) + e_0 = (19 - 18) = 1 \bmod 25$$

Therefore $v = (4, 15) = (2, 14) + (2, 1)$.

3.2.2. Reconfiguring the Nodes

Consider that a fault occurs at node $v = (v_{n-1}v_{n-2}\cdots v_0)$; its nearest spare node $u = (u_{n-1}u_{n-2}\cdots u_0)$, and its error vector $e = (e_{n-1}e_{n-2}\cdots e_0)$, i.e., $vH^T = (u + e)H^T = eH^T$, and $|e| \leq t$. We can construct a path from u to v as follows; let $P(u, v)$ be a path from u and v by decrementing e in a dimensional order. Let $P(u, v) = \langle u = y_0, y_1, y_2, \cdots, y_{m-1}, y_m = v\rangle$. A node y_{i+1} in $P(u, v)$ will be mapped to y_i, for $i > 0$.

In a 2D torus, T_{k_1,k_2}, where $(2t^2 + 2t + 1)|k_1$ and $(2t^2 + 2t + 1)|k_2$, for a given P-node (v_1, v_0), using Algorithm 3.5, we can get the error vector $e = (e_1\ e_0)$ and so the P-node's nearest spare node $u = (u_1\ u_0)$. Thus the path between u and v is

$$\begin{aligned} P(u, v) = \langle &(u_1\ u_0), (u_1{+}1\ u_0), (u_1{+}2\ u_0), \cdots, \\ &(v_1\ u_0), (v_1\ u_0{+}1), \cdots, (v_1\ v_0)\rangle \end{aligned}$$

Thus the mapping of the processors in $P(u, v)$ is as follows.

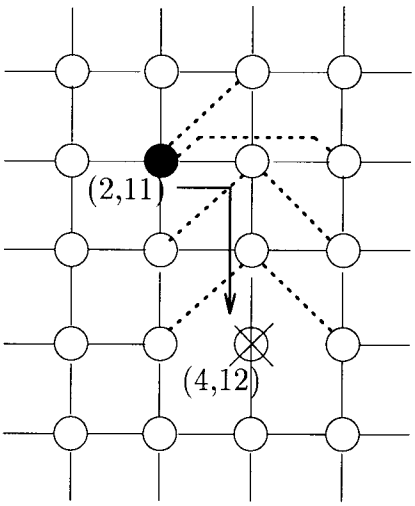


FIGURE 3.6. Reconfiguring nodes

$$
\begin{array}{llll}
(& v_1 & v_0 &) \rightarrow (\; v_1-1 \quad v_0 \quad) \\
(& v_1-1 & v_0 &) \rightarrow (\; v_1-2 \quad v_0 \quad) \\
 & & \vdots & \\
(& u_1+1 & v_0 &) \rightarrow (\; u_1 \quad v_0 \quad) \\
(& u_1 & v_0 &) \rightarrow (\; u_1 \quad v_0-1 \;) \\
(& u_1 & v_0-1 &) \rightarrow (\; u_1 \quad v_0-2 \;) \\
 & & \vdots & \\
(& u_1 & u_0+1 &) \rightarrow (\; u_1 \quad u_0 \quad)
\end{array}
$$

After the reconfiguration, the length of the virtual links will be 2.

For example, suppose a P-node $v = (4\ 15)$ is faulty. The nearest spare node is $u = (2\ 14)$, and its error vector is $(2\ 1)$. Since the path $P(u, v) = \langle (2\ 14), (2\ 15), (3\ 15), (4\ 15) \rangle$, the mapping of the nodes is as follows

$$(4\ 15) \rightarrow (3\ 15)$$

$$(3\ 15) \rightarrow (2\ 15)$$

$$(2\ 15) \rightarrow (2\ 14)$$

In Figure 3.6, the dotted lines are virtual links after reconfiguration.

3.3. Conclusion

In this chapter, we have described some efficient spare processor allocation methods for tolerating node faults in a parallel system whose topology is based on torus. In the presence of faults, these methods also give a simple solution for the system reconfigurations using the spare nodes. After reconfiguration, the length of the virtual link is at most 2. Since many of the present day Massively Parallel Processing Systems(MPPS) use two and three dimensional torus topology for the processor interconnection, the proposed schemes are practical and useful.

4. DATA REARRANGEMENT

Communication between processors and memory hierarchies contribute a high overhead to the total execution time in a multicomputer. Massively parallel systems consisting of thousands of processor nodes are recognized as the only feasible solution to realize performance of the order of trillion operations per second. The interconnection network topology determines the communication bandwidth within the system. Binary n-cubes, also known as hypercubes, provide a high communication bandwidth because of the high ratio between the node degree, n, and the total number of processors, N. For large systems, for example when N is 128 or higher, binary cubes require a very large number of links, thus making the hardware excessively complex and expensive. Topologies based on non-binary systems – such as the k-ary n-cube and the torus – are employed in several modern large multicomputer systems. Our focus here is on the efficient use of communication bandwidth in k-ary n-cube based structures.

Data placement among memory units in a large multiprocessing system offers many alternatives which differ in communication demands. Compilers for many high level languages assign data in a particular order – for example, in the row (column) major order for arrays, although certain new languages, such as High Performance Fortran, provide some flexibility in this direction to the user. Many science and engineering applications, such as finite element analysis, signal and image processing, etc., perform computations on regular grids. Data placement in Cartesian order would be very efficient in terms of the use of communication bandwidth in many of these applications. On the other hand, several other applications, such as the Cooley-Tukey Fast Fourier Transformation, reference data that are located

in processors numbered in the normal order which follows the k-ary space in this architecture. Therefore, conversion between the two placements is an important issue, especially if the distributed memory hierarchy does not allow for data placement enabling data references in both Cartesian and k-ary space. We refer to this problem either as data rearrangement or as code conversion.

Similar to the binary-reflected Gray codes which are employed for embedding arrays in binary cubes, the authors in [22] proposed Lee distance Gray codes for use in k-ary n-cubes. They presented two methods of generating Lee distance Gray codes, leading to block-reflective and non-reflective codes [23]. In this chapter, these two methods are investigated in detail, particularly with reference to the effective use of communication bandwidth during radix-k to Gray code conversion in a k-ary n-cube. A useful metric called Distortion Factor is defined, and this factor is derived for both types of Lee distance Gray codes. Based on this factor, lower bounds for the number of element transfers are derived under two communication constraints, i.e., one-port and all-port communication.

The number of element transfers to convert radix-k to Lee distance Gray code is computed for both types of Gray codes; the block-reflective Gray code requires fewer transfers for all cases. However, the conversion algorithm is simpler for the non-reflective Gray code; also, a Hamiltonian cycle is obtained when the method yielding non-reflective code is used. Next, the problem of transferring (M elements of) data from each processing node numbered in radix k order to the corresponding node in the Gray code order (both types of Gray codes) is studied. Here, the number of ports that can simultaneously communicate data in each processor becomes an important factor. This chapter contains an efficient algorithm, assuming that communication from one node to the other always follows minimum length paths.

More efficient routing algorithms can be developed for k-ary n-cubes (as was shown for binary cubes in [62]) if the minimum path length constraint is relaxed. This chapter also contains such an algorithm for k-ary n-cube structures; the proposed non-minimum path length algorithm is roughly twice as fast as the case where the minimum path length constraint is assumed. Furthermore, the proposed non-minimum path length algorithm for one-port communication is only double the ideal lower bound.

This chapter contains two methods for generating Lee distance Gray codes, the lower bounds for the number of element transfers for k-ary to Gray code conversion, the Distortion Factor metric for both types of Gray codes, and routing algorithms using minimum / non-minimum paths.

4.1. Lee Distance Gray codes

Two methods for constructing Lee distance Gray codes is given here. Let the given n digit radix k vector be $(a_{n-1}a_{n-2}\cdots a_0)$, and let the corresponding Gray code vector be $(g_{n-1}g_{n-2}\cdots g_0)$.

4.1.1. Method 1: (non-reflective code)

$$g_{n-1} = a_{n-1}$$

$$g_i = (a_i - a_{i+1}) \bmod k, \quad \text{for } i = n-2, n-3, \cdots, 0.$$

For example, in a C_5^4 the radix 5 sequence (2304) produces a Gray code sequence (2124). This procedure is proved to be correct in [22]. This method produces a Hamiltonian cycle for any (odd or even) value of k. For example, in a C_5^4, we can construct the following Hamiltonian cycle: (0000, 0001, 0002, 0003, 0004,

0014, 0010, 0011, 0012, 0013, 0023, 0024, 0020, 0021, 0022, 0032, $\cdots$, 4041, 4001, 4002, 4003, 4004, 4000) by applying this method to the normal radix 5 sequence (0000, 0001, 0002, ..., 4443, 4444). Although the resulting sequence forms a Hamiltonian cycle, we can observe that Method 1 is not block-reflective in radix k from the example.

The inverse transformation, namely conversion of a Gray code number to a radix k number, can be obtained as follows:

$$\begin{aligned} a_{n-1} &= g_{n-1} \\ a_i &= (a_{i+1} + g_i) \bmod k \\ &= \sum_{j=i}^{n-1} g_j \bmod k, \quad \text{for } i = n-2, n-3, \cdots, 0. \end{aligned}$$

4.1.2. Method 2: (block-reflective code)

Let $a'_i = a_{i+1}$ if k is even, and $a'_i = \sum_{j=i+1}^{n-1} a_j$ if k is odd. Then,

$$\begin{aligned} g_{n-1} &= a_{n-1} \\ g_i &= a_i, && \text{if } a'_i \text{ is even} \\ &= k - 1 - a_i, && \text{otherwise} \end{aligned}$$

For example, in a C_6^3 , the radix 6 vector (305) produces the Gray code vector (355) according to this method; in a C_7^4 , the radix 7 vector (2106) produces the Gray code vector (2160). This method produces a Hamiltonian cycle only if k is even; it produces a Hamiltonian path if k is odd. For example, in a C_5^4, when we apply this method to the sequence containing numbers in ascending order in radix 5 (0000, 0001, 0002, 0003, 0004, 0010, 0011, 0012, 0013, 0014, 0020, 0021, $\cdots$, 4443, 4444), we obtain the following Gray code sequence: (0000, 0001, 0002, 0003, 0004, 0014,

0013, 0012, 0011, 0010, 0020, 0021, $\cdots$, 4443, 4444). The resulting sequence does not form a Hamiltonian cycle, although it forms a Hamiltonian path. Furthermore, it is easy to observe that the resulting Gray code sequence is block-reflective in radix k.

The inverse transformation, that is, conversion of a Gray code number to radix k number, is given as follows: Let $g'_i = \sum_{j=i+1}^{n-1} g_j$, then.

$$\begin{aligned} a_{n-1} &= g_{n-1} \\ a_i &= g_i, \qquad \text{if } g'_i \text{ is even} \\ &= k-1-g_i, \quad \text{otherwise} \end{aligned}$$

Note that the same procedure works for any value of k, and the above two methods produce identical results for a binary hypercube.

4.1.3. Distortion Factor of Lee Distance Gray codes

Definition 4.1 (Distortion Factor): *For a given sequence S and its mapping f, we define distortion factor as $\lambda_f(S) = \max D_L(a, f(a))$ for $a \in S$.*

This metric gives the extent of distortion the given number suffers as a result of the mapping employed. This factor is used to measure the deviation of the Lee distance Gray code from the corresponding radix k code for both methods of generating Gray codes. In a later section, this factor is used to effect the element transfers when data is transferred on a k-ary n-cube from nodes ordered in radix k to those ordered in Gray code, again following both procedures of generating Gray codes.

Theorem 4.1 *The distortion factor of n digit radix k non-reflective Gray code is:*

$$\lambda_{GN}(k,n) = (n-1)\lfloor \frac{k}{2} \rfloor$$

Proof: From the description of Method 1, $|a_{n-1} - g_{n-1}| = 0$, and $\max|a_i - g_i| = max|a_{i+1}| = \lfloor \frac{k}{2} \rfloor$ for $n-2 \geq i \geq 0$. Thus $\lambda_{GN}(k,n) = \sum_{i=0}^{n-1} \max|a_i - g_i| = \sum_{i=0}^{n-2} \max|a_{i+1}| = (n-1)\lfloor \frac{k}{2} \rfloor$. □

Theorem 4.2 *The distortion factor of n digit radix k block-reflected Gray code is:*

$$\lambda_{GN}(k,2) = \lfloor \frac{k}{2} \rfloor$$

$$\lambda_{GN}(k,n) = (n-1)\lfloor \frac{k}{2} \rfloor, \text{ if } k = 8m+2, 8m+6, 8m+1, 8m+3$$

$$\lambda_{GN}(k,n) = (n-1)(\lfloor \frac{k}{2} \rfloor - 1), \text{ if } k = 8m, 8m+4$$

$$\lambda_{GN}(k,n) = (n-1)(\lfloor \frac{k}{2} \rfloor - 1) + 1, \text{ if } k = 8m+5, 8m+7$$

Proof: Let the node address in radix k be $A = (a_{n-1}a_{n-2}\cdots a_0)$, and the corresponding block-reflective Gray code equivalent be the node address $G = (g_{n-1}g_{n-2}\cdots g_0)$. From Definition 4.1 we know that

$$\begin{aligned}\lambda_{GR}(k,n) &= \max(|a_{n-1} - g_{n-1}| + |a_{n-2} - g_{n-2}| + \cdots + |a_0 - g_0|) \\ &= \max|a_{n-1} - g_{n-1}| + \max|a_{n-2} - g_{n-2}| + \cdots + \max|a_0 - g_0| \\ &= \sum_{i=0}^{n-1} d_i, \text{ where }, d_i = \max|a_i - g_i|.\end{aligned}$$

Since $a_i - g_i = \min((2a_i+1) \bmod k, k - (2a_i+1) \bmod k)$, and k is a multiple of 4, d_i is equal to $\lfloor \frac{k}{2} \rfloor - 1$, i.e., when a_i is equal to one the following four values: $\lfloor \frac{k}{4} \rfloor$, $\lfloor \frac{k}{4} \rfloor - 1$, $\lfloor \frac{3k}{4} \rfloor$, $\lfloor \frac{3k}{4} \rfloor - 1$. For any other value of k, d_i is equal to $\lfloor \frac{k}{2} \rfloor$, and this occurs when a_i is equal to $\lfloor \frac{k}{4} \rfloor$ or $\lfloor \frac{3k}{4} \rfloor$. Also, depending on the value of k, $\lfloor \frac{k}{4} \rfloor$ is either odd or even; similarly, $\lfloor \frac{3k}{4} \rfloor$ is odd or even depending on the value of k.

$k = 8m$: When $a_i = \frac{k}{4}$, $g_i = \frac{3k}{4} - 1$, and $D_L(a_i, g_i) = \frac{k}{2} - 1$; also when $a_i = \frac{k}{4} - 1$, $g_i = \frac{3k}{4}$, and $D_L(a_i, g_i) = \frac{k}{2} - 1$; similarly the Lee distance has the maximum value of $\frac{k}{2} - 1$, also when $a_i = \frac{3k}{4}$, or when $a_i = \frac{3k}{4} - 1$. For all other values

of a_i, the corresponding Gray code number is situated closer. Of these four values of a_i, we know that $\frac{k}{4} - 1$ and $\frac{3k}{4} - 1$ are odd numbers. When the most significant digit of the node address is odd, and every other digit is equal to one of these two odd numbers, the corresponding reflective Gray code number will be situated farthest, thus giving a distortion factor of $(n-1)(\lfloor\frac{k}{2}\rfloor - 1)$.

$k = 8m + 4$: This is similar to the previous case, except that we use either $\frac{k}{4}$ or $\frac{3k}{4}$, as these are odd numbers. The same distortion factor is obtained.

$k = 8m + 2$: The maximum distortion occurs when $a_i = \lfloor\frac{k}{4}\rfloor$ or when $a_i = \lfloor\frac{3k}{4}\rfloor$; the corresponding g_i's are $\lfloor\frac{3k}{4}\rfloor$ and $\lfloor\frac{k}{4}\rfloor$ respectively. In both cases, the deviation is exactly equal to $\lfloor\frac{k}{2}\rfloor$. We know that $\lfloor\frac{k}{4}\rfloor$ is an odd number. Therefore, the distortion factor in this case is $(n-1)\lfloor\frac{k}{2}\rfloor$.

$k = 8m + 6$: This is similar to the previous case, except that we use $\lfloor\frac{3k}{4}\rfloor$, which is an odd number. The same distortion factor is obtained as in the previous case.

$k = 8m + 1$: The maximum deviation of $\lfloor\frac{k}{2}\rfloor$ occurs when $a_i = \lfloor\frac{k}{4}\rfloor$, $g_i = \lfloor\frac{3k}{4}\rfloor$ or when $a_i = \lfloor\frac{3k}{4}\rfloor$, $g_i = \lfloor\frac{k}{4}\rfloor$. We know that both $\lfloor\frac{3k}{4}\rfloor$ and $\lfloor\frac{k}{4}\rfloor$ are even numbers. Therefore, when the most significant digit of the node address is even, and each of the rest of the digits are one of these two numbers, the maximum deviation occurs. The distortion factor in this case is $(n-1)\lfloor\frac{k}{2}\rfloor$.

$k = 8m + 3$: This is similar to the previous case.

$k = 8m + 5$: The maximum deviation of $\lfloor\frac{k}{2}\rfloor$ occurs when $a_i = \lfloor\frac{k}{4}\rfloor$, $g_i = \lfloor\frac{3k}{4}\rfloor$ or when $a_i = \lfloor\frac{3k}{4}\rfloor$, $g_i = \lfloor\frac{k}{4}\rfloor$. We know that both $\lfloor\frac{3k}{4}\rfloor$ and $\lfloor\frac{k}{4}\rfloor$ are odd numbers, and so not useful in generating the maximum deviation. Therefore, we use either $a_i = \lfloor\frac{k}{4}\rfloor - 1$ or $a_i = \lfloor\frac{3k}{4}\rfloor - 1$, which produces a deviation of $\lfloor\frac{k}{2}\rfloor - 1$.

Thus, a processor whose node address has an odd most significant digit and other digits equal to $a_i = \lfloor \frac{k}{4} \rfloor - 1$ or $a_i = \lfloor \frac{3k}{4} \rfloor - 1$ produces a distortion of $(n-1)(\lfloor \frac{k}{2} \rfloor - 1)$. Strictly speaking, we can obtain a distortion factor which is 1 larger than this number, as the least significant digit of the node address could be odd – either $\lfloor \frac{k}{4} \rfloor$ or $\lfloor \frac{3k}{4} \rfloor$.

$k = 8m + 7$: This is similar to the previous case, except that we use either $a_i = \lfloor \frac{k}{4} \rfloor + 1$ or $a_i = \lfloor \frac{3k}{4} \rfloor$. The distortion factor is the same as in the previous case.

These illustrations completes the proof.

□

This metric gives the distortion between numbers in radix k and Gray codes in the worst case, and is useful to evaluate the data routing algorithms for k-ary n-cubes. The maximum difference between a number in the radix k space and the corresponding Gray code number, in any one of the k dimensions, is roughly equal to $k/2$. More precisely, using the non-reflective Gray code, this number is equal to $\lfloor \frac{k}{2} \rfloor$ for any value of k, whereas using the block-reflective Gray code, this number is equal to $\lfloor \frac{k}{2} \rfloor$ for all values of k, except if k is a multiple of 4 when it is equal to $\lfloor \frac{k}{2} \rfloor - 1$. This observation will be useful to determine the effectiveness of a distributed routing algorithm which will be discussed in a subsequent section. The total number of element transfers to convert from radix k code to Gray code on a k-ary n-cube network are dealt in the following section.

4.2. Element Transfers in k-ary n-cubes

Before algorithms for routing data from nodes ordered in radix k order to the corresponding nodes in Gray code space are presented, the volume of communication

in k-ary n-cube structures and lower bounds for effecting this communication are derived.

4.2.1. Total Number of Element Transfers for Gray code Conversion

In the last section, it is shown that both methods to generate Lee distance Gray codes produce approximately the same extent of worst case distortion, either in one of the k dimensions, or in all k dimensions together. However, the following theorems illustrate the difference the two methods in the total distortion over all numbers. This total distortion corresponds to the number of element transfers, or hops, on a k-ary n-cube network, when each node (with a particular node address) communicates with another whose node address is the Gray code equivalent. This gives the volume of traffic during the radix k to Gray code conversion.

Theorem 4.3 *When M units of data is transmitted from each node $A = (a_{n-1}a_{n-2}\cdots a_0)$ in a k-ary n-cube network to another node $G = (g_{n-1}g_{n-2}\cdots g_0)$ where G is the non-reflective Gray code equivalent of A, the total number of element transfers, T_{GN}, is*

$$T_{GN} = M(n-1)k^{n-1}\lceil\frac{k}{2}\rceil\lfloor\frac{k}{2}\rfloor$$

Proof: In the radix-k representation of n digit numbers, there are k^n numbers. Each digit $a_i, a_i = 0, 1, \cdots, k-1$, occurs k^{n-1} times in each of the n-digits. Thus the total number of element transfers can be obtained by multiplying $\sum_{a_i=0}^{k-1}|a_i - g_i| = \sum_{a_i=0}^{k-1}|a_{i+1}| = \sum_{a_{i+1}=0}^{k-1}\min(a_{i+1}, k - a_{i+1})$ by M (the number of elements in each node), k^{n-1} (accounting for the number of times all numbers from 0 to k are repeated in each digit) and $(n-1)$ (accounting for all digits except the most significant digit).

For even k,

$$\sum_{a_{i+1}=0}^{k-1} \min(a_{i+1}, k - a_{i+1}) = 2(\frac{1}{2}(\frac{k}{2} - 1)(\frac{k}{2})) + (\frac{k}{2}) = (\frac{k}{2})^2 = \lfloor\frac{k}{2}\rfloor\lceil\frac{k}{2}\rceil.$$

For odd k,

$$\sum_{a_{i+1}=0}^{k-1} \min(a_{i+1}, k - a_{i+1}) = 2(\frac{1}{2}(\lfloor\frac{k}{2}\rfloor)(\lfloor\frac{k}{2}\rfloor + 1)) = \lfloor\frac{k}{2}\rfloor\lceil\frac{k}{2}\rceil.$$

□

Note that T_{GN} is approximately equal to $M(n-1)k^{n+1}/4$. When $k = 2$, $T_{GN} = M(n-1)2^{n-1}$, which is a well known expression for binary hypercubes.

Theorem 4.4 *When M units of data is transmitted from each node $A = (a_{n-1}a_{n-2}\cdots a_0)$ in a k-ary n-cube network to another node $G = (g_{n-1}g_{n-2}\cdots g_0)$ where G is the block-reflective Gray code equivalent of A, the total number of element transfers, T_{GR}, is*

$$T_{GR} = \begin{cases} M\frac{n-1}{2}k^{n-1}\lfloor\frac{k}{2}\rfloor\lceil\frac{k}{2}\rceil, & \text{if } k = 4m \\ M\frac{n-1}{2}k^{n-1}(\lfloor\frac{k}{2}\rfloor\lceil\frac{k}{2}\rceil + 1), & \text{if } k = 4m+2 \\ M(\frac{n-1}{2}k^{n-1} - \frac{k^n-1}{2(k-1)})\lfloor\frac{k}{2}\rfloor\lceil\frac{k}{2}\rceil, & \text{if } k = 2m+1 \end{cases}$$

Proof: Let the node address in radix k be $A = (a_{n-1}a_{n-2}\cdots a_i\cdots a_0)$ and the corresponding reflective Gray code is equivalent to $G = (g_{n-1}g_{n-2}\cdots g_i\cdots g_0)$. Let us first calculate the total number of element transfers between a_i and g_i for various k and all possible values of a_i, where $k - 1 \geq a_i \geq 0$.

$k = 2m + 1$:

$$\sum_{a_i=0}^{k-1} D_L(a_i, g_i) = 2 \times (1 + 2 + \cdots + \lfloor\frac{k}{2}\rfloor) = \lfloor\frac{k}{2}\rfloor\lceil\frac{k}{2}\rceil.$$

$k = 4m$:

$$\sum_{a_i=0}^{k-1} D_L(a_i, g_i) = k \times 1 + (k-4) \times 2 + (k-8) \times 2 + \cdots + 8 \times 2 + 4 \times 2 = \lfloor \frac{k}{2} \rfloor \lceil \frac{k}{2} \rceil.$$

$k = 4m + 2$:

$$\sum_{a_i=0}^{k-1} D_L(a_i, g_i) = k \times 1 + (k-4) \times 2 + (k-8) \times 2 + \cdots + 6 \times 2 + 2 \times 2 = \lfloor \frac{k}{2} \rfloor \lceil \frac{k}{2} \rceil + 1.$$

Now, let us calculate the number of times $\sum_{a_i=0}^{k-1} D_L(a_i, g_i)$ is repeated over the entire range of the n digit radix k number, for different values of k. First, we note that at least half the a_i's do not require any element transfers in the case of the block-reflective Gray code; in fact when k is odd, more than half a_i's do not need any transfers.

Even k: In each of the $n-1$ digits (i.e., $n-2 \geq i \geq 0$), there is an equal likelihood for $a_i = g_i$ Therefore, totally in each of these $n-1$ digits $\sum_{a_i=0}^{k-1} D_L(a_i, g_i)$ is repeated k^{n-1} times with a probability of 0.5. When each node has M data elements, the total number of element transfers is $M \frac{n-1}{2} k^{n-1} \lfloor \frac{k}{2} \rfloor \lceil \frac{k}{2} \rceil$ when $k = 4m$, and $M \frac{n-1}{2} k^{n-1} (\lfloor \frac{k}{2} \rfloor \lceil \frac{k}{2} \rceil + 1)$ when $k = 4m + 2$.

Odd k: The number of times $\sum_{a_i=0}^{k-1} D_L(a_i, g_i)$ is repeated varies in each digit position as follows: $\sum_{j=0}^{n-2} \lfloor \frac{k^j}{2} \rfloor k^{n-2-j}$; we can expand this series to get the closed form solution as $(\frac{n-1}{2} k^{n-1} - \frac{k^n - 1}{2(k-1)})$. Therefore, when each node has M data elements, the total number of element transfers is $M(\frac{n-1}{2} k^{n-1} - \frac{k^n - 1}{2(k-1)}) \lfloor \frac{k}{2} \rfloor \lceil \frac{k}{2} \rceil$.

□

Note that T_{GR} is approximately equal to $M(n-1)k^{n+1}/8$, which is half of T_{GN} for the same values of k and n. When k is odd, fewer element transfers are required. Again, note that, when $k = 2$, T_{GR} reduces to $M(n-1)2^{n-1}$ as T_{GN} does.

k / n	2	3	4	5	6
2	2 : 2	8 : 8	24 : 24	64 : 64	160 : 160
3	6 : 2	36 : 14	162 : 68	512 : 284	2430 : 1094
4	16 : 8	128 : 64	768 : 384	4096 : 2048	20480 : 10240
5	30 : 12	300 : 132	2250 : 1032	15000 : 7032	93750 : 44532
6	54 : 30	648 : 360	5832 : 3240	46656 : 25920	349920 : 194400

TABLE 4.1. Total element transfers ($T_{GN} : T_{GR}$)

Table 4.1 gives T_{GN} and T_{GR} (as $T_{GN} : T_{GR}$) for various values of k and n. Although the non-reflective method involves nearly twice the number of element transfers, it will be shown later that both methods perform equally well under the proposed routing algorithm.

4.2.2. Some Lower Bounds

The total number of element transfers and the worst case distortion were calculated above for radix k to (both types of) Lee distance Gray code conversion. In this section, it is considered how effectively these element transfers can be carried out using the communication models and channels (that is, links between nodes) provided in a k-ary n-cube network. One of two communication models, namely one-port communication and all-port communication, is commonly employed in a multicomputer system [63]. In both models, a processor can send and receive data simultaneously on the same port. In one-port communication, a processor can send data on only one port at a time. In contrast to one-port communication model,

an all-port communication model permits simultaneous communication on all the channels connected to a processor.

Lower bounds for the both models are presented in this section. First, consider an ideal case, that is, we can use all the links in the network without any being idle at each step. Of course, this assumption is not realistic as it would be difficult to come up with an algorithm which can use all links to effect the data transfer between specified nodes. Nevertheless, this bound is shown as an ultimate lower bound, which it is easily obtained by dividing T_{GN} by the total degrees of links, i.e., $2nk^n$ if $k > 2$, nk^n if $k = 2$.

Theorem 4.5 *For the all-port communication model, a lower bound for the number of element transfers in sequence for radix k to non-reflective Gray code conversion in a k-ary n-cube network is*

$$\max(\frac{T_{GN}}{2nk^n}, (n-1)\lfloor\frac{k}{2}\rfloor) = \max(M\frac{n-1}{2nk}\lfloor\frac{k}{2}\rfloor\lceil\frac{k}{2}\rceil, (n-1)\lfloor\frac{k}{2}\rfloor).$$

For even $k > 2$, this reduces to $\max((1-\frac{1}{n})\frac{M}{4}\frac{k}{2}, (n-1)\frac{k}{2})$. When $n = 2$, this gives a lower bound equal to $\frac{M}{8}\frac{k}{2}$ for $k > 2$, and $\frac{M}{4}$ for $k = 2$. This lower bound $\frac{M}{4}$ for a binary hypercube is the same as the lower bound shown in [62]. The second term in the expression for the lower bound is, of course, the distortion factor as derived in Section 4.1.3.

Under the one-port communication model, a lower bound can be obtained when all processors use exactly one port at each step.

Theorem 4.6 *For the one-port communication model, a lower bound for the number of element transfers in sequence for radix k to non-reflective Gray code conversion in a k-ary n-cube network is*

$$\max(\frac{T_{GN}}{k^n}, (n-1)\lfloor\frac{k}{2}\rfloor) = \max(M\frac{n-1}{k}\lfloor\frac{k}{2}\rfloor\lceil\frac{k}{2}\rceil, (n-1)\lfloor\frac{k}{2}\rfloor).$$

A similar bound can be derived for the block-reflective Gray code too. This will be roughly half of the number for the non-reflective method; that is, when $n = 2$, lower bounds are equal to $\frac{M}{16}\lfloor\frac{k}{2}\rfloor$ for all-port communication, and $\frac{M}{2}\lfloor\frac{k}{2}\rfloor$ for one-port communication.

While developing routing algorithms, we could either stipulate that only minimum length paths be used, or allow usage of non-minimum length paths too. The minimum length path constraint restricts the use of free links in the network and thus does not lead to effective utilization of the existing channel capacity; but on the positive side, routing algorithms based on this restriction are usually simple. On the other hand, algorithms which facilitate routing on non-minimum length paths (in addition to the minimum length paths) are usually quite involved, but may lead to more efficient performance – that is, conversion can be completed in fewer steps than using algorithms with the path length constraint. The bounds given above are also valid for non-minimal length path communication. For minimum length path data transfer, at present, it is not clear whether the above bound is a true lower bound or there exists a better lower bound.

4.3. Routing Algorithms for Code Conversion

Data routing algorithms are presented for transferring information from nodes on a k-ary n-cube network to the corresponding nodes whose node addresses are Gray code equivalents – for both types of Lee distance Gray codes. The algorithms which use only minimum length paths for each type of Gray code is first shown. This is followed by an algorithm which uses non-minimum length paths too.

A routing algorithm to effect binary to Gray code conversion on a binary cube is proposed in [62]. This algorithm is based on exchanging data between two

nodes in each dimension. In Section 4.1.3, it is observed that in each dimension of a k-ary n-cube, the distortion could be up to $\lfloor \frac{k}{2} \rfloor$. In other words, while performing radix k to Lee distance Gray code conversion in a k-ary n-cube, we may have to move data from one node to another which are separated by up to $\lfloor \frac{k}{2} \rfloor$ on the same dimension. Therefore, it is not possible to develop routing algorithms based on simple exchanges as in the case of binary hypercubes. While it is not impossible to come up with routing algorithms without using routing tags, such algorithms would either necessitate a centralized controller (as opposed to distributed routing) or lead to excessively complicated design of the processor.

With the overhead of a header with each data element, the most straightforward approach is to precompute the number of steps (and direction) of data movement required in each dimension (totally $n-1$ dimensions) and use this $n-1$ digit radix k information as the routing tag. It is easy to appreciate that such a routing tag would guarantee correct routing. This routing tag enables fast distributed routing, and it will be shown in this section.

4.3.1. Algorithm 1: Radix k to Gray code Data Routing (single I/O port)

Each of code conversion algorithms has two parts: the first part involves the preparation of the routing tag, and the second part consists of steps involved in the actual routing. The routing tag (or header) depends on the type of Lee distance Gray code employed. Once the routing tag is prepended to the data, the actual routing procedure is identical regardless of the type of Gray code used.

Let the source node be $A = (a_{n-1}a_{n-2}\cdots a_0)$ and the corresponding destination node be $G = (g_{n-1}g_{n-2}\cdots g_0)$, where G is the Gray code (a type of Lee distance Gray code) equivalent of A . The routing tag to be calculated is $r = (r_{n-1}r_{n-2}\cdots r_0)$.

Non-reflective Gray code: $r_i = a_i - g_i = a_{i+1}$, for $n-2 \geq i \geq 0$.

Block-reflective Gray code: For $n-2 \geq i \geq 0$,

$$r_i = a_i - g_i = \begin{cases} 0 & \text{if } a'_i \text{ is even} \\ (2a_i + 1) \bmod k & \text{otherwise} \end{cases}$$

$$\text{where } a'_i = \begin{cases} a_i & \text{if } k \text{ is even} \\ \sum_{j=i+1}^{n-1} a_j & \text{otherwise} \end{cases}$$

Each processor considers the routing tag from left to right.

Case (i): $r_i = 0$. No routing is required in the ith dimension; move to the $(i+1)$-th dimension.

Case (ii): $r_i \leq \lfloor \frac{k}{2} \rfloor$. Decrement the tag digit: $r_i \leftarrow r_i - 1$. Then, transmit the data element on the negative direction edge(-) along the ith dimension, i.e., from the intermediate node $(b_{n-1}b_{n-2}\cdots b_0)$ to node $(b_{n-1}b_{n-2}\cdots b_{i+1}(b_i-1)b_{i-1}\cdots b_0)$.

Case (iii): $r_i > \lfloor \frac{k}{2} \rfloor$. Increment the tag digit: $r_i \leftarrow (r_i+1) \bmod k$. Then, transmit the data element on the positive direction edge(+) along the ith dimension, i.e., from the intermediate node $(b_{n-1}b_{n-2}\cdots b_0)$ to node $(b_{n-1}b_{n-2}\cdots b_{i+1}(b_i+1)b_{i-1}\cdots b_0)$.

If all digits of the tag are zero, the data is in the destination node; no routing required.

Based on the Lee distance Gray code generation methods explained in Section 4.1.3, it is clear that this algorithm calculates the number of hops required along each dimension and uses this information as the routing tag. Therefore, it is rather trivial to show that the algorithm is correct.

In algorithm 1, the number of element transfers in sequence is $(n-1)\lfloor\frac{k}{2}\rfloor$ when each node has one unit of data. With M data elements in each node, this becomes $M(n-1)\lfloor\frac{k}{2}\rfloor$, which is the only double the ultimate lower bound in Theorem 4.6. We note that the data rearrangement time is exactly the same for both types of Lee distance Gray codes, although the number of element transfers for the non-reflective Gray code is nearly twice as much as that for the block- reflective Gray code. This is because the communication time, which is proportional to the number of element transfers in sequence, depends on the distortion factor and not the volume.

If unidirectional channel, meaning a processor can only send or receive data at each step, is assumed, then cases ii. and iii. should be combined as data can be routed only in one direction along each dimension. Then, up to $(k-1)$ element transfers in sequence may be required along each dimension thus nearly doubling the total number to $M(n-1)(k-1)$.

For block-reflective Gray code, when data transfer is proceeding along one dimension, all the links in the other dimensions remain idle. In order to effectively utilize the communication bandwidth provided by the k-ary n-cube network, we propose an improved routing algorithm (see Algorithm 5).

4.3.2. Algorithm 2: Radix k to Gray Code Data Routing (multiple I/O port)

Note that the routing tag consists of $(n-1)$ digits each of which corresponds to the number of element transfers along a particular dimension. The ordering of dimensions is unimportant, and it does not necessarily have to proceed from the most significant to the least significant digit (left to right) as specified in the algorithm, but the header digits can be used in any order. The following lemma summarizes this result.

Lemma 4.1 *The radix k to Lee distance Gray code (either type) conversion can be performed as a sequence of element transfers in dimensions* $\{0, 1, 2, \cdots, n-2\}$ *taken in arbitrary order.*

During radix k to Gray code conversion, each node of a C_k^n transfers M data elements to another node; as discussed earlier, transfer of each data element involves up to $\lfloor \frac{k}{2} \rfloor$ element transfers in sequence along each of the $n-1$ dimensions. Recognizing that different dimension ordering of element transfers gives different paths, consider the set of dimension ordering given below:

$$
\begin{aligned}
DO_0 &= \langle 0, 1, 2, 3, \cdots, n-4, n-3, n-2 \rangle \\
DO_1 &= \langle 1, 2, 3, 4, \cdots, n-3, n-2, 0 \rangle \\
&\vdots \\
DO_i &= \langle i, i+1, i+2, \cdots, n-3, n-2, 0, 1, \cdots, i-2, i-1 \rangle \\
&\vdots \\
DO_{n-2} &= \langle n-2, 0, 1, 2, \cdots, n-4, n-3 \rangle
\end{aligned}
$$

This set of dimension ordering has the property that no same dimension is used at step i. Thus, if the M data elements on each processor are divided into $(n-1)$ subsets, and arranged by the suggested dimension ordering, the total amount of communication steps can be reduced.

Theorem 4.7 *Algorithm 2 for the radix k to Lee distance Gray code (either type) conversion can be carried out in a with element transfers in sequence using only minimum length paths and employing all-port communication.*

Proof: Consider three different sets of values of M, the number of data elements in each node. Case (i) is used for most elements for large values of M and is the

only relevant case when M is a multiple of $n-1$. Case (ii) is used to assure optimal transmission for the remainder of elements when M is large but not a multiple of $n-1$.

Case (i): $M \bmod (n-1) = 0$. Create $n-1$ transfer sequences that are different rotations of the dimensions $n-2, n-3, \cdots, 0$. Partition the data set in each processor into $(n-1)$ sets of the same size and assign one sequence to each data set. Along each dimension there could be anywhere from zero up to $\lfloor \frac{k}{2} \rfloor$ element transfers in sequence, thus adding up to a maximum of $M\lfloor \frac{k}{2} \rfloor$.

Case (ii): $n-1 < M < 2n-2$. Create K transfer sequences that are distinct rotations of the dimensions $n-2, n-3, \cdots, 0, \phi_1, \phi_2, \cdots, \phi_z$, where ϕ_is are dummy dimensions. No transfer is performed in a dummy dimension. Again the maximum number of element transfers in sequence is $M\lfloor \frac{k}{2} \rfloor$.

Case (iii): $M < n-1$. It is easy to see that element transfers in sequence are necessary and sufficient.

For arbitrary $M > n-1$, define the routing by partitioning the data set such that cases (i) and (ii) apply.

□

In implementing the algorithm, a table can be set up in each processor such that for each memory partition the corresponding routing tags are stored.

From Theorem 4.7, the above routing algorithm, which uses only minimum length paths, is approximately 4 times slower than the ideal lower bound. There is also no difference between the speed of conversion from radix k to either type of Lee distance Gray code, as the distortion factors are the same for both. Of course, many links remain idle during conversion to reflective Gray code, as the total number of

element transfers in that case is roughly half of that for the non-reflective Gray code conversion.

If unidirectional channel is used, cases ii) and iii) should be merged as it is effective to move data only in one direction along each dimension. This would nearly double the number of element transfers in sequence to $\max(M(k-1),(n-1)(k-1))$.

4.3.3. Algorithm 3: Data Routing without Tags

As noted earlier, it is possible to develop algorithms that do not use routing tags for the data rearrangement under consideration, although it may not lead to efficient processor design. An algorithm for converting from radix k to reflective Gray code in a 3-ary 3-cube will be presented as an example. This algorithm is similar to that proposed by Johnsson and Ho [62] for binary hypercubes. This algorithm works for conversion only to reflective Gray code for all odd k.

The algorithm does not employ routing tags. For illustration purposes, assume that the data stored in each processor is the Gray code equivalent of the processor address in (which is in radix 3 form). Let us consider a node with address $(a_{n-1}a_{n-2}\cdots a_{i+1}a_ia_{i-1}\cdots a_0)$, where i is the dimension; we should consider $n-2 \geq i \geq 0$. If $\sum_{j=i+1}^{n-1} a_j$ is even, no element transfer is required involving this node in the i^{th} dimension; on the other hand, if $\sum_{j=i+1}^{n-1} a_j$ is odd, the data element in this node is exchanged with that in the node $(a_{n-1}a_{n-2}\cdots a_{i+1}(k-1-a_i)a_{i-1}\cdots a_0)$. The two nodes which exchange data are not adjacent, and up to $\lfloor \frac{k}{2} \rfloor$ element transfers could be required to carry out the exchange. As any given processor always exchanges with another specified processor along any selected dimension, all exchanges along each dimension can be preprogrammed; although this is clearly pos-

data	paddr	1	0	1	0	data	paddr	1	0	1	0
0	000	0	0	0	0	14	121	14	14	14	14
1	001	1	1	1	1	11	120	17	15	9	15
2	002	2	2	2	2	10	121	16	16	10	16
5	010	5	3	3	3	9	122	15	17	11	17
4	011	4	4	4	4	18	200	18	18	18	18
3	012	3	5	5	5	19	201	19	19	19	19
6	020	6	6	6	6	20	202	20	20	20	20
7	021	7	7	7	7	23	210	23	21	21	21
8	022	8	8	8	8	22	211	22	22	22	22
17	100	11	9	15	9	21	212	21	23	23	23
16	101	10	10	16	10	24	220	24	24	24	24
15	102	9	11	17	11	25	221	25	25	25	25
12	110	12	12	12	12	26	222	26	26	26	26
13	111	13	13	13	13						

TABLE 4.2. Conversion of a radix-k to a block-reflective Gray code

sible in theory, it makes the processor design quite complex in practice, making this algorithm an unwise engineering choice.

As in the previous algorithm, here the data set in each processor is partitioned into $n-1$ sets of the same size and to each set one of the sequences obtained by rotations of the dimensions $n-2, n-3, \cdots, 0$ is assigned. This facilitates concurrent

use of links in $n-1$ dimensions. Table 4.2 shows that the order in which the dimensions are considered is immaterial.

When k is even, the different sequences do not produce proper conversion to reflective Gray code if the same procedure is used for exchange. However, we can identify different (but fixed for a given sequence) procedures which produce proper conversion. Again, this makes the processor design very complex, also processors are no longer identical and modular. Similarly, for any k, conversion to non-reflective Gray code require different procedures to be employed for the different sequences, thus making the processors very complex and non-modular. Therefore, this algorithm may not be a good practical choice.

4.3.4. Algorithm 4: Gray to Radix k Conversion

This algorithm is similar to the all-port communication algorithm for radix k to Gray code conversion using routing tags, i.e., Algorithm 2. Partitioning of the data set in each processor into $n-1$ equal sets and assigning a sequence to each set is carried out as in the previous algorithms. As in Algorithm 1, there are two steps, the first step being the formation of the tag and prepending it to each data element, and the second step involving the element transfers. In fact, the second step is identical to that in Algorithm 1, and will not be repeated here.

Let $G = (g_{n-1}g_{n-2}\cdots g_{i+1}g_ig_{i-1}\cdots g_0)$ be the source node and let the corresponding destination node be $A = (a_{n-1}a_{n-2}\cdots a_{i+1}a_ia_{i-1}\cdots a_0)$, where A is the radix k equivalent of G. Let the routing tag to be calculated be $r = (r_{n-1}r_{n-2}\cdots r_{i+1}r_ir_{i-1}\cdots r_0)$.

Non-reflective Gray code: $r_i = g_i - r_i = -\sum_{j=i+1}^{n-1} g_j \bmod k$, for $n-2 \geq i \geq 0$.

Block-reflective Gray code: For $n-2 \geq i \geq 0$,

$$r_i = g_i - a_i = \begin{cases} 0 & \text{if } g'_i \text{ is even} \\ (2g_i + 1) \bmod k & \text{otherwise} \end{cases}$$

$$\text{where } g'_i = \begin{cases} g_i & \text{if } k \text{ is even} \\ \sum_{j=i+1}^{n-1} g_j & \text{otherwise} \end{cases}$$

Note that the routing tag for Gray code to radix k conversion is the negative (radix k) of that for conversion in the opposite direction. Alternatively, we could use the same tag as for the other conversion, but define the direction of transfer to be opposite. The number of element transfers in sequence needed for Gray code to radix k conversion is identical to that for radix k to Gray code conversion.

4.3.5. Algorithm 5: Routing using Non-minimum Length Paths

In the case of reflective Gray code, the total number of element transfers during radix k to Gray code conversion is roughly half of that for the non- reflective code. This means that many links are unused. More than half of the links in the k-ary n-cube are unused during conversion to reflective Gray code.

Lemma 4.2 *During radix k to reflective Gray code conversion, if element transfer is required along dimension i for the node* $(a_{n-1}a_{n-2}\cdots a_{i+1}a_ia_{i-1}\cdots a_0)$, *then no element transfer would be required along dimension i for the nodes* $(a_{n-1}a_{n-2}\cdots(a_{i+1}-1)a_ia_{i-1}\cdots a_0)$ *and* $(a_{n-1}a_{n-2}\cdots(a_{i+1}+1)a_ia_{i-1}\cdots a_0)$.

Proof:

even k: As element transfer is required along dimension i, we know that a_{i+1} is odd, and hence both $a_{i+1}-1$ and $a_{i+1}+1$ are even, thus satisfying the statement of the lemma.

odd k: As element transfer is required along dimension i, we know that $\sum_{j=i+1}^{n-1} a_j$ is odd. This means that $\sum_{j=i+1}^{n-1} a_j$ would be even for the nodes where the $(i+1)^{th}$ digit is $a_{i+1}-1$ or $a_{i+1}+1$, thus satisfying the lemma.

□

The algorithms given in the previous section can be modified to exploit the communication bandwidth available in the k-ary n-cube network. The main idea behind such modifications is to divide the data set in each processor into two parts: one part to be transmitted along the minimum length path and another along an alternate path utilizing the unused links. This algorithm works as follows.

1. Divide the local data set into two equal sets: S_1 and S_2, i..e., $|S_1| = |S_2| = \frac{M}{2}$, by assuming one-port communication. The one data set, S_1, is routed using minimum length paths, whereas the other data set, S_2, is forced to follow longer but unused paths.

2. The next step is to calculate routing tags for each data element. The routing tag here consists of $n-2$ radix k digits as before plus one additional bit to indicate whether the data element belongs to S_1 or S_2. For the data elements in S_1, calculation of the $n-2$ digits of the tag is identical to that in Algorithm 1 or 2. However, for the data elements in S_2, each of the digits of the tag, except the least significant one, should be one less than (mod k) the actual calculated number: $1(r_{n-2}-1)(r_{n-3}-1)\cdots(r_1-1)r_0$.

3. Once the routing tags are computed, the data sets will be routed along the proper order of dimensions, $n-2, n-3, \cdots, 0$ as in Algorithm 1.

Data transfer along a given dimension in step i consists of 3 substeps;

(a) route all data elements in S_2 to $(a_{n-1}a_{n-2}\cdots(a_{i+1}-1)a_ia_{i-1}\cdots a_0)$. This requires $\frac{M}{2}$ steps.

(b) exchange data along the dimension. It takes approximately $\frac{M}{2}\lfloor\frac{k}{2}\rfloor$ steps.

(c) route all data elements in S_2 to $(a_{n-1}a_{n-2}\cdots a_{i+1}a_ia_{i-1}\cdots a_0)$. It takes $\frac{M}{2}$ steps.

Thus the total time to exchange data on each dimension is requires $M+\frac{M}{2}\lfloor\frac{k}{2}\rfloor$.

Therefore the total of $(n-1)M(1+\frac{k}{4}) \approx \frac{k}{4}(n-1)M$ element transfers in sequence is required to complete the conversion under the assumption of one-port communication. Furthermore this algorithm is twice the ideal lower bound for large M and k.

This algorithm can also be applied to the all-port communication. If S_1 both and S_2 are divided into $(n-1)$ subsets, and are routed as described in Algorithm 2, then the total number of steps is approximately $\frac{1}{4}kM$.

This algorithm is clearly better than the speed of the algorithms using only minimum length paths. This algorithm gives better results when M is large, also when k is larger than n (which is usually the case).

4.4. Concluding Remarks

We have investigated the problem of transmitting data elements on k-ary n-cube networks from nodes with addresses in radix k to those whose addresses are the Lee distance Gray code equivalents; we refer to this problem either as data rearrangement or as code conversion. The two types of Lee distance Gray codes presented in an earlier thesis are compared from a standpoint of data rearrangement. We introduced a new measure termed Distortion Factor which basically determines the number of element transfers in sequence for data rearrangement. However, the

total number of element transfers for data rearrangement corresponds to the volume of traffic and indicates the spare communication channel capacity available in the network.

The lower bounds which were derived for the number of element transfers in sequence give us some insight into the complexity of the code conversion problem. These bounds are identical to the corresponding numbers for binary hypercubes but for a multiplying factor of $\lfloor \frac{k}{2} \rfloor$. The routing algorithms presented in this thesis (both for one-port communication and for all-port communication) also feature the same multiplying factor when compared to the corresponding algorithms for binary n-cubes.

We have presented an near optimal algorithm of data rearrangement under the constraint that non-minimum length paths are employed for one-port communication. The algorithm presented here, assuming no minimum path length constraint, produces a significant saving in the number of element transfers in sequence. Table 4.3 shows the summary of an approximate total number of element transfers in sequence required depending on the communication channels and algorithms when M and k are large.

The torus structure is chosen by the designers of many recently developed multicomputers; for example, the Cray T3D, the iWarp, and the Tera parallel computer employ the torus topology. A torus is similar to a k-ary n-cube except that different radices (ks) could be used for different dimensions; therefore, a torus, $T_{k_0,k_1,\cdots,k_{n-2},k_{n-1}}$ is a generalized C_k^n [10, 23, 21]. The bounds and algorithms given in this thesis may be extended to the torus topology.

	all-port		one-port	
	non-reflective	reflective	non-reflective	reflective
Lower bound	$\frac{1}{8}kM$	$\frac{1}{16}kM$	$\frac{1}{4}(n-1)kM$	$\frac{1}{8}(n-1)kM$
Algorithm 1	N/A	N/A	$\frac{1}{2}(n-1)kM$	$\frac{1}{2}(n-1)kM$
Algorithm 2	$\frac{1}{2}kM$	$\frac{1}{2}kM$	N/A	N/A
Algorithm 5	N/A	$\frac{1}{4}kM$	N/A	$\frac{1}{4}(n-1)kM$

TABLE 4.3. Comparison of proposed algorithms

5. EDGE DISJOINT HAMILTONIAN CYCLES AND GRAY CODES

When edge disjoint Hamiltonian cycles are used in a communication algorithm, their effectiveness is improved if more than one cycle exists. Although the existence of disjoint Hamiltonian cycles in the cross product of various graphs has been discussed in the literature [9, 29, 5, 17, 36, 37, 89, 50, 61], a straightforward way of generating such disjoint cycles is not clear. This chapter contains the functions that generate these disjoint cycles for the k-ary n-cubes, the hypercubes, and the $2D$ tori. This chapter also provides some ideas of decomposing the higher dimensional tori into edge disjoint Hamiltonian cycles.

In a Lee distance Gray code C, the set of k^n vectors over Z_k^n are arranged in a sequence such that two adjacent vectors are at a Lee distance one. It is assumed that the first and the last vectors in this sequence are also at a distance 1.

Two Gray codes, C_1 and C_2, over Z_k^n are said to be *independent if* two words, a and b, are adjacent in C_1 (or C_2), then they are not adjacent in C_2 (or C_1). If $k \geq 3$, we can have at most n sets of independent Gray codes; for $k = 2$, this number is $\lfloor \frac{n}{2} \rfloor$.

Theorem 5.1 *The independent set of Gray codes over Z_k^n is equivalent to the set of edge disjoint Hamiltonian cycles in the graph of k-ary n-cube, C_k^n.*

Proof: A Gray code over Z_k^n consists of k^n vectors, and the Lee distance between two contiguous vectors is exactly one. By letting the k^n vectors over Z_k^n be the nodes in a graph G, and joining an edge between two nodes if their Lee distance is exactly one, G becomes a k-ary n-cube graph because of the definition of k-ary n-cube graph described in Chaper 1. Suppose we have two independent Gray codes

C_1 and C_2, and the corresponding cycles H_1 and H_2 in C_k^n. Since C_1 and C_2 are independent, no edge is in both H_1 and H_2. Therefore H_1 and H_2 are edge-disjoint Hamiltonian cycles. □

In the following, $h_i(X) = h_i((x_{n-1}, x_{n-2}, \cdots, x_0); Z_k^n) = (g_{n-1}, g_{n-2}, \cdots, g_0)$ represents the mapping of the radix number form to the corresponding Gray code form, for $i = 0, 1, 2, \cdots, n-1$ (assuming $k \geq 3$), and these mappings give the independent Gray codes. Similarly, $H_0(C_k^n)$, $H_1(C_k^n)$,$\cdots$, $H_{n-1}(C_k^n)$ denote the corresponding Hamiltonian cycles derived by the functions $h_i(X)$, $i = 0, 1, \cdots, n-1$.

For $n = 1$, only one Gray code exists, and so in the rest of the chapter it is assumed that $n \geq 2$ and $k \geq 3$. Further, for $X = (x_{n-1}, x_{n-2}, \cdots, x_0) \in Z_k^n$, let $(X, Z_{k_1}^{n_1})$ represent the n_1 digit long radix-k_1 representation of the radix-k number X. For example, if $X = (2, 4, 4, 3) \in Z_5^4$ then $(X, Z_{25}^2) = (14, 23)$. If there is no confusion, we also assume $X = (X, Z_{k^n}^1)$, i.e., X can be treated an integer in the range 0, 1, $\cdots$, $k^n - 1$.

The following sections show the functions that generate the edge disjoint Hamiltonian cycles for various k-ary n-cubes.

5.1. k-ary 2-cube (i.e., $n = 2$)

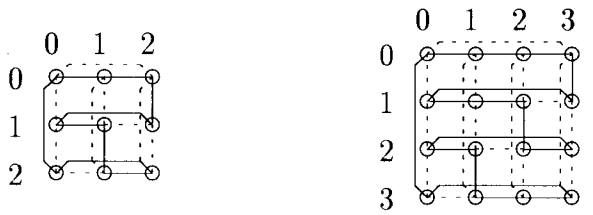
Theorem 5.2 *There are two independent Gray codes in C_k^n and these are generated by the functions h_0 and h_1, where*

$$h_0(X; Z_{k^2}^1) = h_0((x_1, x_0); Z_k^2) = (x_1, (x_0 - x_1) \ mod \ k)$$

$$h_1(X; Z_{k^2}^1) = h_1((x_1, x_0); Z_k^2) = ((x_0 - x_1) \ mod \ k, x_1)$$

Proof: Proof is given in [22, 23]. □

The inverse functions of h_0 and h_1 are as follows;



(a) Z_3^2 (b) Z_4^2

FIGURE 5.1. Independent Gray codes for $n = 2$

$$h_0^{-1}((g_1, g_0); Z_k^2) = (g_1, (g_0 + g_1) \bmod k)$$
$$h_1^{-1}((g_1, g_0); Z_k^2) = ((g_0 + g_1) \bmod k, g_1)$$

Example 5.1 *The two Gray code sequences are shown in Figure 5.1 for ($k = 3$ and $n = 2$) and ($k = 4$ and $n = 2$). The solid line represents the first Gray code and the dotted line the second.*

5.2. $n = 2^r$, and $k \geq 3$

The disjoint Hamiltonian cycles can be recursively constructed as described in the following theorem.

Theorem 5.3 *The n independent sets of Gray codes are defined by the functions $h_i, i = 0, 1, 2, \cdots, n-1$ as described below. Let $i_1 = \lfloor \frac{2i}{n} \rfloor$ and $h_{i_1}(X; Z_k^n) = h_{i_1}((X_1, X_0); Z_{k^{n/2}}^2) = (Y_1, Y_0)$, i.e.,*

$$(Y_1, Y_0) = h_{i_1}((X_1, X_0); Z_{k^{n/2}}^2)$$
$$= \begin{cases} (X_1, (X_0 - X_1) \ mod \ k^{n/2}) & for \ i_1 = 0, \\ ((X_0 - X_1) \ mod \ k^{n/2}, X_1) & for \ i_1 = 1 \end{cases}$$

Then, the i^{th} function h_i is recursively defined as

$$h_i(X; Z_k^n) = (y_{n-1}, y_{n-2}, \cdots, y_{n/2}, y_{n/2-1}, \cdots, y_0)$$

where,

$$(y_{n-1}, y_{n-2}, \cdots, y_{n/2}) = h_{(i \bmod n/2)}(Y_1; Z_k^{n/2}),$$
$$(y_{n/2-1}, y_{n/2-2}, \cdots, y_0) = h_{(i \bmod n/2)}(Y_0; Z_k^{n/2}).$$

Proof: By induction on n.

Base: when $n = 2$, this theorem is reduced to Theorem 5.2.

Induction Hypothesis: Assume that there are n independent sets of Gray codes in C_k^n, H_0', H_1', $\cdots$, H_{n-1}' for $n = 2^r$, i.e., $C_k^n = H_0' + H_1' + \cdots + H_{n-1}'$.

Induction Step: Now consider the case for $n' = 2n = 2^{r+1}$. A C_k^n can be decomposed as

$$\begin{aligned} C_k^{n'} &= C_k^n \times C_k^n \\ &= (H_0' + H_1' + \cdots + H_{n-1}') \times (H_0'' + H_1'' + \cdots + H_{n-1}'') \\ &= (H_0' \times H_0'') + \cdots + (H_{n-1}' \times H_{n-1}'') \end{aligned}$$

Here $G_1 + G_2$ indicates the union of two edge disjoint graphs, G_1 and G_2. Now $H_i' \times H_i''$ is a two dimensional torus of size $k^n \times k^n$ and this is edge-disjoint from $H_j' \times H_j''$ for $i \neq j$. From each $H_i' \times H_i''$, two disjoint Hamiltonian cycles can be constructed using Theorem 5.2. Thus $2n$ disjoint Hamiltonian cycles can be constructed from C_k^n, and the i^{th} Gray code is the i_1^{th} cycle of the component $(H_{i_0}' \times H_{i_0}'')$, where $i_0 = i \bmod \lfloor n/2 \rfloor$ and $i_1 = \lfloor \frac{2i}{n} \rfloor$. □

Example 5.2 *In Z_4^8, there are 8 independent sets of Gray codes, which are generated by the functions $h_i, i = 0, 1, 2, \cdots, 7$. Let $X = (2, 1, 1, 3, 2, 3, 0, 1)$ be a given*

vector over Z_4^8. *We now describe how this vector is mapped under* h_3. *Since* $i = 3$, $i_1 = \lfloor \frac{2\times 3}{8} \rfloor = 0$, *and* $h_0((2,1,1,3,2,3,0,1); Z_{4^4}^2) = h_0((151,177); Z_{256}^2) = (151,26) = (Y_1, Y_0)$.

$$\begin{aligned} h_3(X; Z_k^n) &= (h_{(3 \bmod 4)}(151; Z_4^4), h_{(3 \bmod 4)}(26; Z_4^4)) \\ &= (h_3((2,1,1,3); Z_4^4), h_3((0,1,2,2); Z_4^4)). \end{aligned}$$

To find $h_3((2,1,1,3); Z_4^4)$, *note that* $((2,1,1,3); Z_4^4) = ((9,7), Z_{16}^2)$. *In this recursion,* $i_1 = \lfloor \frac{2\times 3}{4} \rfloor = 1$ *and so* $h_1((9,7); Z_{16}^2) = (14,9)$. *Thus*

$$\begin{aligned} h_3((2,1,1,3); Z_4^2) &= (h_{(3 \bmod 2)}(14; Z_4^2), h_{(3 \bmod 2)}(9; Z_4^2)) \\ &= (h_1((3,2); Z_4^2), h_1((2,1); Z_4^2)) = (3,3,2,2). \end{aligned}$$

Similarly it can be shown that $h_3((0,1,2,2); Z_4^4) = (3,2,1,0)$ *and thus* $h_3((2,1,1,3,2,3,0,1); Z_4^8) = (3,3,3,2,3,2,1,0)$. *Further, it can be verified that* $h_0(X) = (2,3,3,3,0,1,2,3)$, $h_1(X) = (3,2,3,3,1,0,3,2)$, *etc.*

5.3. $n = 3$ and k is odd

Finding the 3 independent Gray codes over Z_k^3 can be achieved by seeing the problem as an edge disjoint Hamiltonian cycle problem in a cross product of three cycles, C_k^3.

$$\begin{aligned} C_k^3 &= C_k \times C_k \times C_k \\ &= (C_k \times C_k) \times C_k \\ &= (H_A + H_B) \times C_k \\ &= H_0' + H_1' + H_2' \end{aligned}$$

In the above, H_A and H_B are the two independent Gray codes in Z_k^2, i.e., $H_A = H_0(C_k^2)$ and $H_B = H_1(C_k^2)$, and H_0' and H_1' are two edge disjoint cycles derived from

$(H_A \times C_k)$. Thus, the independent Gray codes in a mixed radix number, $Z_{k_1 \times k_0}$ are presented first.

Theorem 5.4 *In a mixed radix number system* $Z_{k_1 \times k_0}$, *if* $GCD(k_1, k_0 - 1) = 1$, *and* $k_1 = mk_0$, *for* $m \geq 1$, *then the following two functions generate the independent Gray codes.*

$$f_0(X; k_1, k_0) = (x_1 \ mod \ k_1, (x_0 + (k_0 - 1)x_1) \ mod \ k_0)$$

$$f_1(X; k_1, k_0) = ((x_0 + (k_0 - 1)x_1) \ mod \ k_1, x_1 \ mod \ k_0)$$

$$where \ X = (x_1, x_0), x_1 \in Z_{k_1}, \ and \ x_0 \in Z_{k_0}$$

Proof: The proof has two parts.

(1) If $X' \neq X''$, then it is required to prove that $f_0(X'; k_1, k_0) \neq f_0(X''; k_1, k_0)$, and $f_1(X'; k_1, k_0) \neq f_1(X''; k_1, k_0)$. Let $X' = (x_1', x_0')$, $X'' = (x_1'', x_0'')$, $x_1', x_1'' \in Z_{k_1}$, and $x_0', x_0'' \in Z_{k_0}$.

(a) Suppose $f_0(X'; k_1, k_0) = f_0(X''; k_1, k_0)$. Since $x_1' = x_1'' \bmod k_1$ and $x_1', x_1'' \in Z_{k_1}$, $x_1' = x_1''$. For the second component, $x_0' + (k_0 - 1)x_1' = x_0'' + (k_0 - 1)x_1'' \bmod k_0$, and hence $x_0' = x_0''$. Thus $f_0(X'; k_1, k_0) \neq f_0(X''; k_1, k_0)$ if $X' \neq X''$.

(b) Suppose $f_1(X'; k_1, k_0) = f_1(X''; k_1, k_0)$. $x_1' = x_1'' \bmod k_0$, i.e., $k_0 | (x_1' - x_1'')$, and $x_0' + (k_0 - 1)x_1' = (x_0'' + (k_0 - 1)x_1'') \bmod k_1$, i.e., $x_0' - x_0'' + (k_0 - 1)(x_1' - x_1'') = 0 \bmod k_1$. Since $|x_0' - x_0''| < k_0$ and $k_0 | (x_1' - x_1'')$, $x_0' - x_0'' = 0 \bmod k_0$. Further, $x_1' - x_1'' = 0 \bmod k_1$ because $GCD(k_0 - 1, k_1) = 1$. Thus $f_1(X'; k_1, k_0) \neq f_1(X''; k_1, k_0)$ if $X' \neq X''$.

This implies that f_0 and f_1 are one-to-one mappings over Z_k^2.

(2) f_0 and f_1 generate Hamiltonian cycles, and no edges are in both H_0 and H_1, where H_0 and H_1 are generated by f_0 and f_1 respectively. Consider the following cases.

(a) Case $X = (x_1, x_0)$, and $X + 1 = (x_1, x_0 + 1)$ in H_0: Since $f_0(X) = (x_1, x_0 + (k_0 - 1)x_1)$, and $f_1(X + 1) = (x_1, x_0 + 1 + (k_0 - 1)x_1)$, we have $e_0 = (f_0(X), f_0(X + 1)) = ((x_1, x_0 + (k_0 - 1)x_1), (x_1, x_0 + 1 + (k_0 - 1)x_1))$ and $D_L(f_0(X), f_0(X + 1)) = 1$.

(b) Case $X' = (x'_1, k_0 - 1)$, and $X' + 1 = (x'_1 + 1, 0)$ in H_0: Similar to case (a), we have $e_1 = ((x'_1, (k_0 - 1)(x'_1 + 1)), (x'_1 + 1, (k_0 - 1)(x'_1 + 1)))$ and $D_L(f_0(X'), f_0(X' + 1)) = 1$.

(c) Case $X'' = (x''_1, x''_0)$, and $X'' + 1 = (x''_1, x''_0 + 1)$ in H_1: Similarly, we have $e_2 = (f_1(X''), f_1(X''+1)) = ((x''_0+(k_0-1)x''_1, x''_1), (x''_0+1+(k_0-1)x''_1, x''_1))$ and $D_L(f_1(X''), f_1(X'' + 1)) = 1$.

(d) Case $X''' = (x'''_1, k_0 - 1)$, and $X''' + 1 = (x'''_1 + 1, 0)$ in H_1: we have $e_3 = (f_1(X'''), f_1(X'''+1)) = (((k_0-1)(x'''_1+1), x'''_1), ((k_0-1)(x'''_1+1), (x'''_1+1)))$ and $D_L(f_1(X'''), f_1(X''' + 1)) = 1$.

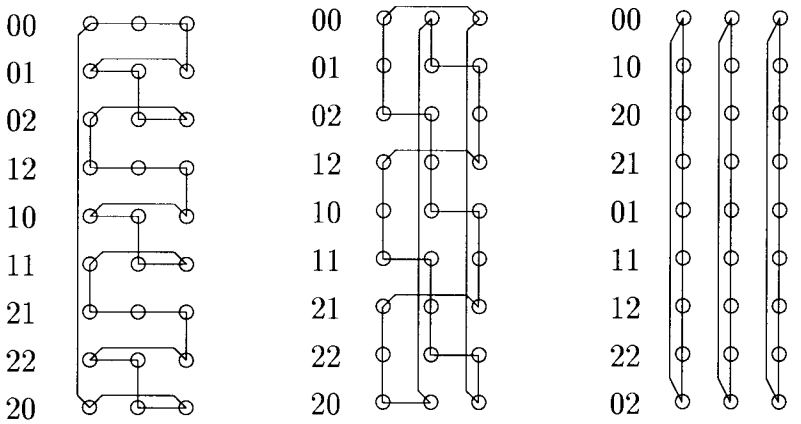
Since f_0 is one-to-one and $D_L(f_0(X), f_0(X+1)) = D_L(f_0(X'), f_0(X'+1)) = 1$, f_0 generates a Hamiltonian cycle. Similarly, f_1 also generates another Hamiltonian cycle. Further, the edges, e_0, e_1, e_2, and e_3, are mutually different. Therefore H_0 and H_1 are edge disjoint.

□

The inverses of f_0 and f_1 are as follows

$$f_0^{-1}(Y; k_1, k_0) = (y_1 \bmod k_1, (y_0 - (k_0 - 1)y_1) \bmod k_0)$$

$$f_1^{-1}(Y; k_1, k_0) = ((y_0 - (k_0 - 1)y_1) \bmod k_1, y_1 \bmod k_0)$$



(a) H'_0 (b) H'_1 (c) H'_2

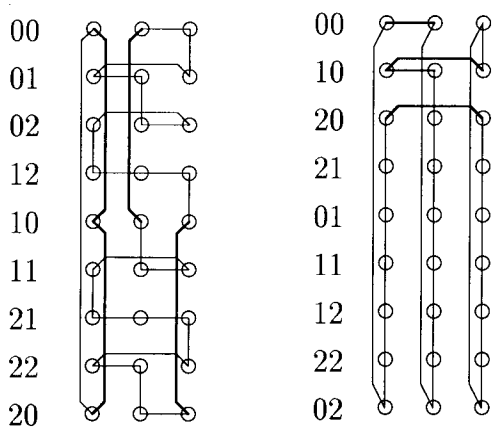
FIGURE 5.2. Incomplete 3 independent Gray codes in C_3^3

where, $Y = (y_1, y_0), y_1 \in Z_{k_1}$, and $y_0 \in Z_{k_0}$

Now go back to the problem of the edge disjoint Hamiltonian decomposition for k-ary 3-cube. Here H'_0 and H'_1 are obtained by Theorem 5.4. This theorem is applicable since $GCD(k^2, k-1) = 1$. Figure 5.2 shows these incomplete Gray codes.

Although H'_0 and H'_1 are edge disjoint Hamiltonian cycles in C_k^3, H'_2 is not a Hamiltonian cycle. Furthermore, H'_0, H'_1, and H'_2 consume all the edges in a k-ary 3-cube, C_k^3. Therefore some edges in H'_0 or H'_1 must be removed and used in constructing a Hamiltonian cycle from the last component, H'_2. The edges that will be removed from H'_0 or H'_1 are called *exchange* edges. However, selecting the proper exchange edges is not straightforward; note that the Hamiltonian cycles in H_1 and H_2 must be maintained even after the exchange process is done.

Figure 5.3 shows an example of incorrect exchange edges. H''_0 in this figure contains two subcycles. In this case, the exchange edges are



(a) H_0'' (b) H_2''

FIGURE 5.3. Incorrect edge exchanges in C_3^3

$\{(000,001),(100,101),(102,100),(202,200)\}$. For the detail steps, we first enumerate the nodes in H_0' of Figure 5.2 in a Gray code sequence.

$$H_0' = \langle 000, 001, 002, \cdots, 122, 102, 100, 101, 111, \cdots, 201, 202, 200 \rangle$$

After the edges $\{(000,001),(100,101)\}$ are exchanged, we get

$$\langle 000, 100, 102, 122, \cdots, 001, 101, \cdots, 201, 202, 200 \rangle. \tag{5.1}$$

However when $\{(102,100),(202,200)\}$ are chosen to be exchanged, we get two subcycles.

$$\langle 000, 100, 200 \rangle, \text{ and } \langle 102, 122, \cdots, 001, 101, \cdots, 201, 202 \rangle \tag{5.2}$$

This is because the exchange is incorrect at the step (5.2) (see Figure 5.4(b)). In other words, as long as the twisting types are the same as in Figure 5.4(a), the

(a) Type-a (b) Incorrect twist

FIGURE 5.4. Twist sequence

Hamiltonian cycle will be maintained. The key idea of choosing the exchange edges is to find the edges that will maintain the twist type-a as in Figure 5.4(a).

The formal description of these twisting edges using the type-a is as follows. Assume an integer sequence, $SEQ = \langle 0, 1, \cdots, a_1, a_2, \cdots, b_1, b_2, \cdots \rangle$, needs to be twisted as $SEQ' = \langle 0, 1, \cdots, a_1, b_1, b_1 - 1, \cdots, a_2, b_2, b_2{+}1 \cdots \rangle$. (see Figure 5.4). Then the twisting function can be defined as

$$\varphi(x, S) = \begin{cases} x, & \text{if } x < a' \\ a' + b' - 1 - x, & \text{if } a' \leq x < b' \\ x, & \text{if } b' \leq x \end{cases}$$

where, $S = \{(\langle a_1, b_1 \rangle, \langle a_2, b_2 \rangle)\}$, $a' = \min(a_2, b_1)$, $b' = \max(a_1, b_2)$, and $(a_2 - a_1)(b_2 - b_1) = 1$.

If the sequence needs to be twisted several times recursively, this function can be generalized as follows. Let $S = \{P_1, P_2, \cdots, P_l\}$, where $P_i = (\langle a_{i,1}, b_{i,1} \rangle, \langle a_{i,2}, b_{i,2} \rangle)$, $a'_i = \min(a_{i,2}, b_{i,1})$, $b'_i = \max(a_{i,1}, b_{i,2})$, and $(a_{i,2} - a_{i,1})(b_{i,2} - b_{i,1}) = 1$, for some integer $a'_i s$ and $b'_i s$. Further, $P_i \odot P_j = 1$ for all i, j, and $i \neq j$, where

$$P_i \odot P_j = \begin{cases} 1, & \text{if } a'_i < a'_j \text{ and } b'_j < b'_i \\ 1, & \text{if } a'_j < a'_i \text{ and } b'_i < b'_j \\ 1, & \text{if } b'_j < a'_i \\ 1, & \text{if } a'_j > b'_i \\ 0, & \text{otherwise} \end{cases}$$

Now, let $\varphi(x, S) = x^{(l)} = \varphi^{(l)}(x, S)$, where $|S| = l$, be the function to twist indices between $a'_i s$ and $b'_i s$.

$$x^{(i)} = \varphi^{(i)}(x, S) = \begin{cases} x^{(i-1)}, & \text{if } x < a' \\ a' + b' - 1 - x, & \text{if } a' \leq x < b' \\ x^{(i-1)}, & \text{if } b' \leq x \end{cases}$$

where, $a' = \min(a_{i,2}^{(i-1)}, b_{i,1}^{(i-1)})$, $b' = \max(a_{i,1}^{(i-1)}, b_{i,2}^{(i-1)})$, and $\varphi^{(0)}(x, S) = x$.

Now consider a k-ary 3-cube, C_k^3. First identify the three points P_p, P_m, and P_s in H_A as follows.

$$P_p = (k-2, 0)$$

$$P_m = (k-1, 0)$$

$$P_s = (k-1, 1)$$

Note that P_p, P_m, and P_s are consecutive nodes in H_B. Further we can find the corresponding values in C_k^2. Let

$$p = f_0^{-1}(P_p; k, k) = ((k-2, k-2), Z_k^2) = ((k-2)k + k - 2, Z_{k^2}^1)$$

$$m = f_0^{-1}(P_m; k, k) = ((k-1, k-1), Z_k^2) = ((k-1)k + k - 1, Z_{k^2}^1)$$

$$s = f_0^{-1}(P_s; k, k) = ((k-1, 0), Z_k^2) = ((k-1)k, Z_{k^2}^1)$$

Now consider a mixed radix number system, $Z_{k^2 \times k}$. Using Theorem 5.4, we can find two independent Gray codes, namely, H'_0 and H'_1. Here we choose the rows p, m, and s in H'_0, and twist the the Gray code sequence as in Figure 5.3.

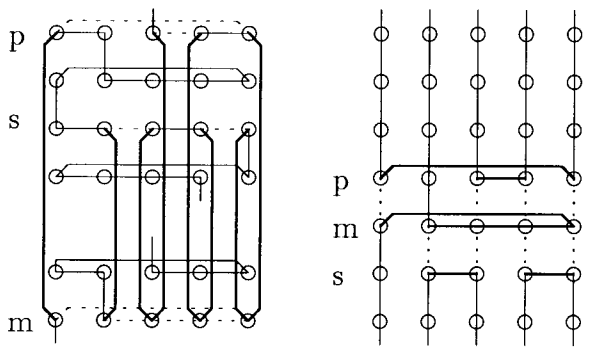


FIGURE 5.5. Exchange edges for odd k

Let $q_{x,i}$ denote the node whose row number is x and column number is i in H_0', where $x \in Z_{k^2}$ and $i \in Z_k$, i.e., $q_{x,i} = f_0^{-1}((x,i); k^2, k)$. If there are two parallel exchange edges, $\{(q_{p,i}, q_{p,i+1}), (q_{m,i}, q_{m,i+1})\}$, between the rows p and m, and columns i and $i+1$, we denote the exchange edges as a pair, $PM_i = (\langle q_{p,i}, q_{p,i+1}\rangle, \langle q_{m,i}, q_{m,i+1}\rangle)$. Similarly, $SM_i = (\langle q_{s,i}, q_{s,i+1}\rangle, \langle q_{m,i}, q_{m,i+1}\rangle)$ denotes the exchange edges between rows s and m, and columns i and $i+1$.

Now select the following set.

$$S = \{SM_1, PM_2, SM_3, PM_4, \cdots, SM_{k-2}, PM_{k-1}\}. \tag{5.3}$$

Let $X = (x_2, x_1, x_0) \in Z_k^3$, and $X' = (x_1', x_0') \in Z_{k^2 \times k}$ is the mixed radix representation of X. The functions that generate the three Hamiltonian cycles in a k-ary 3-cubes, where k is odd, are as follows.

$\underline{h_0(X; Z_k^3) = (y_2, y_1, y_0)}$: Let $(y_1', y_0') \in Z_{k^2 \times k} = f_0(\varphi(X', S); k^2, k)$. Then

$$(y_2, y_1) = h_0((y_1'; Z_k^2), Z_k^2)$$

$$y_0 = y'_0$$

$\underline{h_1(X; Z_k^3) = (y_2, y_1, y_0)}$: Let $(y'_1, y'_0) \in Z_{k^2 \times k} = f_1(X'; k^2, k)$. Then

$$(y_2, y_1) = h_0((y'_1; Z_k^2), Z_k^2)$$

$$y_0 = y'_0$$

$\underline{h_2(X; Z_k^3) = (y_2, y_1, y_0)}$: Let $X'' = (x''_1, x''_0)$, and $(y'_1, y'_0) \in Z_{k^2 \times k} = f_2(X''; k^2, k)$, where $x''_1 = \lfloor \frac{X}{k_1 - 1} \rfloor$, $x''_0 = X \bmod (k_1 - 1)$, and

$$f_2(X; k_1, k_0) = \begin{cases} (k_1 - 2 - x''_0, x''_1), & \text{if } x''_1 \bmod 2 = 0, \text{ and } x''_1 < k_0 \\ (x''_0, x''_1), & \text{if } x''_1 \bmod 2 = 1, \text{ and } x''_1 < k_0 \\ (k_1 - 1, k_0 - 1 - x''_0), & \text{if } x''_1 = k_0 \end{cases} \quad (5.4)$$

Then

$$(y_2, y_1) = h_1((y'_1 + p, Z_k^2); Z_k^2)$$

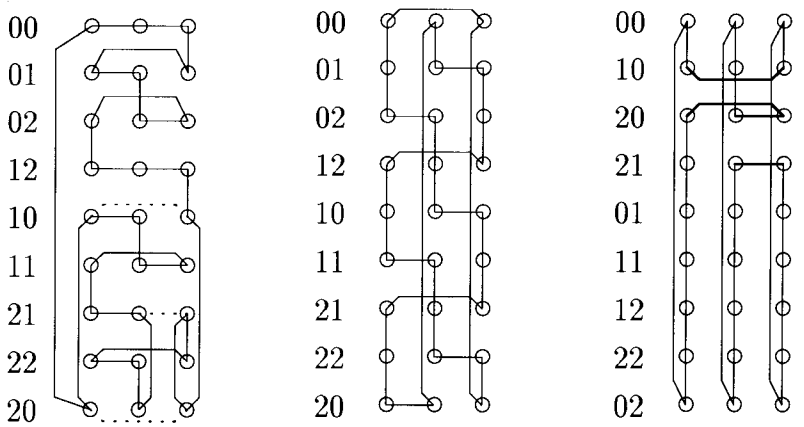
$$y_0 = y'_0 + 1$$

Example 5.3 *Figure 5.6 shows a way of choosing the proper exchange edges in the 3-ary 3-cube, as* $P_p = (1, 0)$, $P_m = (2, 0)$, *and* $P_s = (2, 1)$.

5.4. $n = 3$ and k is even

The solutions in Section 5.3 work fine for odd k; however these solutions can not be directly applied to the case when k is even. This is because there is no edge, $(q_{p,1}, q_{p,2})$ as in Figure 5.7. Thus we need to modify the exchange set S.

Choose another pair of exchange edges that will not overlap the existing exchange set. Let



(a) H_0 (b) H_1 (c) H_2

FIGURE 5.6. Complete 3 independent Gray codes in C_3^3

$$P_{j_a} = (k-3, 2) \in Z_k^2$$

$$P_{j_b} = (k-3, 3) \in Z_k^2$$

Then P_{j_a} and P_{j_b} are consecutive in H_B. Let

$$j_a = f_0^{-1}(P_{j_a}; k, k)$$

$$j_b = f_0^{-1}(P_{j_b}; k, k)$$

Similar to Section 5.3, let $PJ = (\langle q_{j_a,1}, q_{j_b,1} \rangle, \langle q_{j_a,2}, q_{j_b,2} \rangle)$.

The set of exchange edges is denoted by

$$S = \{PJ, SM_2, PM_3, \cdots, SM_{k-2}, PM_{k-1}\}.$$

Since the process of exchange edges are same as the case of odd k, the functions h_0 and h_1 are the same as those in Section 5.3. However h_2 will be different because the set S is different.

$\underline{h_2(X; Z_k^3) = (y_2, y_1, y_0)}$: Let $(y_1', y_0') \in Z_{k^2 \times k} = f_3(X'; k^2, k)$, where

$$f_3(X; k_1, k_0) = \begin{cases} f_2(X; k_1, k_0) + (p, 2), & \text{if } X \leq j_a \\ (2j_a - X + 1, 0) + (p, 1), & \text{if } j_a < X \leq j_a + k_1 \\ f_2(X - k_1; k_1, k_0) + (p, 2), & \text{otherwise} \end{cases} \tag{5.5}$$

Then

$$(y_2, y_1) = h_1((y_1' + p, Z_k^2); Z_k^2)$$

$$y_0 = y_0' + 1$$

Example 5.4 *Figure 5.7 shows a way of choosing the proper exchange edges in the 4-ary 3-cube, as* $P_{j_a} = (1,3)$, $P_{j_b} = (1,2)$, $P_p = (2,0)$, $P_m = (3,0)$, *and* $P_s = (3,1)$.

5.5. $n = 2m$, and $m \geq 2$

The i^{th} Gray code, $h_i(X; Z_k^n) = (y_{2m-1}, y_{2m-2}, \cdots, y_0)$, can be recursively defined as follows, where $(X, Z_k^n) \equiv (X', Z_{k^m}^2)$, i.e., $X' = (X_1, X_0), X_1, X_0 \in Z_{k^m}$. Let $i_1 = \lfloor \frac{2i}{n} \rfloor$, and $(Y_1, Y_0) = h_{i_1}((X_1, X_0); Z_{k^m}^2)$ Then,

$$(y_{2m-1}, y_{2m-2}, \cdots, y_m) = h_{(i \bmod m)}(Y_1; Z_k^m)$$

$$(y_{m-1}, y_{m-2}, \cdots, y_0) = h_{(i \bmod m)}(Y_0; Z_k^m)$$

5.6. $n = 2m + 1$, $m \geq 1$, and k is odd

This is the generalization of the case of k-ary 3-cube; in other words, the construction is based on the exchange of edges. The success of exchanging edges highly depends on the selected rows in H_0'.

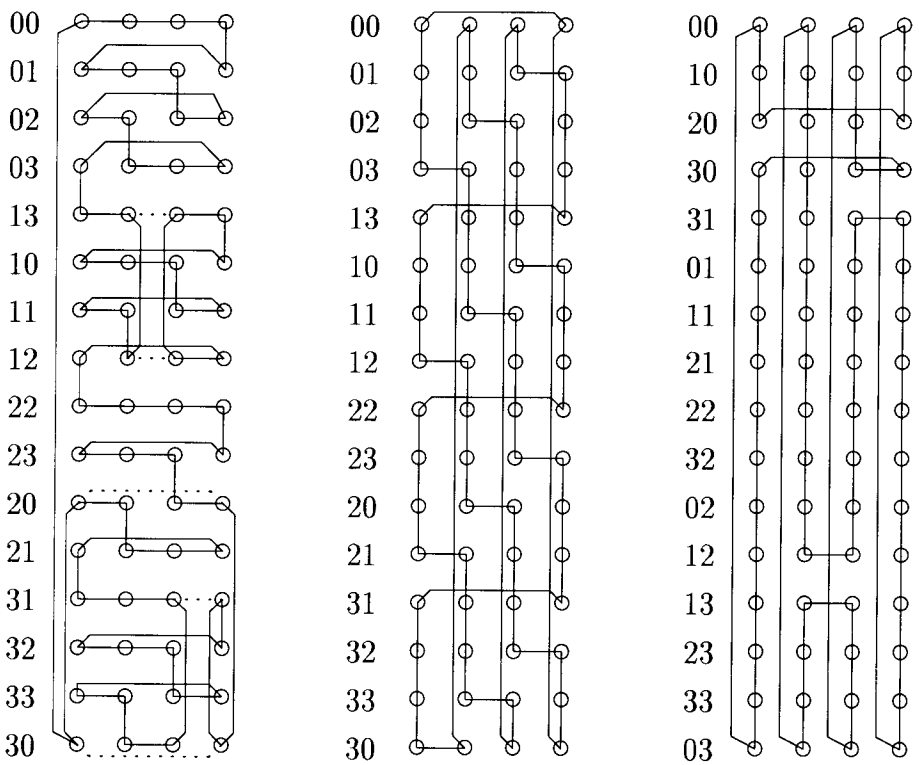


FIGURE 5.7. 3 independent Gray codes in C_4^3

Three points $(k^{\lfloor\frac{m+1}{2}\rfloor}-2,0)$, $(k^{\lfloor\frac{m+1}{2}\rfloor}-1,0)$, $(k^{\lfloor\frac{m+1}{2}\rfloor}-1,1)$ in Z_k^{m+1} are chosen, such that

$$p = f_0^{-1}((k^{\lfloor\frac{m+1}{2}\rfloor}-2,0); k^{\lceil\frac{m+1}{2}\rceil}, k^{\lfloor\frac{m+1}{2}\rfloor})$$
$$m = f_0^{-1}((k^{\lfloor\frac{m+1}{2}\rfloor}-1,0); k^{\lceil\frac{m+1}{2}\rceil}, k^{\lfloor\frac{m+1}{2}\rfloor})$$
$$s = f_0^{-1}((k^{\lfloor\frac{m+1}{2}\rfloor}-1,1); k^{\lceil\frac{m+1}{2}\rceil}, k^{\lfloor\frac{m+1}{2}\rfloor})$$

Let $PM_i = (\langle q_{p,i}, q_{p,i+1}\rangle, \langle q_{m,i}, q_{m,i+1}\rangle)$, and $SM_i = (\langle q_{s,i}, q_{s,i+1}\rangle, \langle q_{m,i}, q_{m,i+1}\rangle)$, where $q_{x,i} = f_0^{-1}((x,i); k^{m+1}, k^m)$. Select the following set.

$$S = \{SM_1, PM_2, SM_3, PM_4, \cdots, SM_{k-2}, PM_{k-1}\}.$$

Let $X = (x_1, x_0)$, and $X' = (x_1', x_0') = \varphi(X, S)$, where $x_1, x_1' \in Z_k^{m+1}$, and $x_0, x_0' \in Z_k^m$.

$\underline{h_0(X; Z_k^n) = (y_{2m}, y_{2m-1}, \cdots, y_0)}$:

$$\text{where}\begin{cases} (y_1', y_0') & = f_0(\varphi(X,S); k^{m+1}, k^m) \\ (y_{2m}, y_{2m-1}, \cdots, y_m) & = h_0(y_1'; Z_k^{m+1}) \\ (y_{m-1}, y_{m-2}, \cdots, y_0) & = h_0(y_0'; Z_k^m) \end{cases}$$

$\underline{h_m(X; Z_k^n) = (y_{2m}, y_{2m-1}, \cdots, y_0)}$:

$$\text{where}\begin{cases} (y_1', y_0') & = f_1(X; k^{m+1}, k^m) \\ (y_{2m}, y_{2m-1}, \cdots, y_m) & = h_0(y_1'; Z_k^{m+1}) \\ (y_{m-1}, y_{m-2}, \cdots, y_0) & = h_0(y_0'; Z_k^m) \end{cases}$$

$\underline{h_{2m}(X; Z_k^n) = (y_{2m}, y_{2m-1}, \cdots, y_0)}$:

$$\text{where}\begin{cases} (y_1', y_0') & = f_2(X; k^{m+1}, k^m) \\ (y_{2m}, y_{2m-1}, \cdots, y_m) & = h_1(y_1' + p; Z_k^{m+1}) \\ (y_{m-1}, y_{m-2}, \cdots, y_0) & = h_0(y_0' + 1; Z_k^m) \end{cases}$$

Let $X'' = (x_1'', x_0'')$, where $x_1'' = \lfloor \frac{X}{k_1 - 1} \rfloor$, and $x_0'' = X \bmod (k_1 - 1)$,

$$f_2(X; k_1, k_0) = \begin{cases} (k_1 - 2 - x_0'', x_1''), & \text{if } x_1'' \bmod 2 = 0, \text{ and } x_1'' < k_0 \\ (x_0'', x_1''), & \text{if } x_1'' \bmod 2 = 1, \text{ and } x_1'' < k_0 \\ (k_1 - 1, k_0 - 1 - x_0''), & \text{if } x_1'' = k_0 \end{cases} \quad (5.6)$$

$\underline{h_i(X; Z_k^n) = (y_{2m}, y_{2m-1}, \cdots, y_0),\ i > 2}$:

$$\text{where,} \begin{cases} (y_1', y_0') & = f_{i_a}(X'; k^{m+1}, k^m) \\ (y_{2m}, y_{2m-1}, \cdots, y_m) & = h_{i_b}(y_1'; Z_k^{m+1}) \\ (y_{m-1}, y_{m-2}, \cdots, y_0) & = h_{i_b}(y_0'; Z_k^m) \\ i_a & = \lfloor \frac{i}{m} \rfloor \\ i_b & = i \bmod 2 \end{cases}$$

5.7. $n = 2m + 1$, $m \geq 1$, **and** k **is even**

This is similar to Section 5.6, but it requires one more entry to the set S

$$j_a = f_0^{-1}((k^{\lfloor \frac{m+1}{2} \rfloor} - 3, 2); k^{\lceil \frac{m+1}{2} \rceil}, k^{\lfloor \frac{m+1}{2} \rfloor})$$

$$j_b = f_0^{-1}((k^{\lfloor \frac{m+1}{2} \rfloor} - 3, 3); k^{\lceil \frac{m+1}{2} \rceil}, k^{\lfloor \frac{m+1}{2} \rfloor})$$

Let $PJ = (\langle q_{j_a,1}, q_{j_a,1} \rangle, \langle q_{j_b,1}, q_{j_b,2} \rangle)$ and $S = \{\ PJ,\ SM_2,\ PM_3,\ SM_4,\ PM_5,\ \cdots,\ SM_{k-2},\ PM_{k-1}\ \}$. Apply all cases as in Section 5.6 except for $h_{2m}(X)$.

$\underline{h_{2m}(X; Z_k^n) = (y_{2m}, y_{2m-1}, \cdots, y_0)}$:

$$\text{where,} \begin{cases} (y_1', y_0') & = f_3(X'; k^{m+1}, k^m) \\ (y_{2m}, y_{2m-1}, \cdots, y_m) & = h_1(y_1' + p; Z_k^{m+1}) \\ (y_{m-1}, y_{m-2}, \cdots, y_0) & = h_0(y_0' + 1; Z_k^m) \end{cases}$$

$$f_3(X;k_1,k_0)=\begin{cases} f_2(X;k_1,k_0)+(p,2), & \text{if } X\le ja' \\ (2ja'-X+1,0)+(p,1), & \text{if } ja'<X\le ja'+k_1 \\ f_2(X-k_1;k_1,k_0)+(p,2), & \text{otherwise} \end{cases} \tag{5.7}$$

5.8. Binary cubes ($k = 2$)

The n-dimensional hypercube, Q_n, can be defined as

$$Q_n = Q_{n-1} \times Q_1$$

$\underline{n = 2m,\ k = 2, \text{ and } m \ge 1:}$

If n is even (i.e., $n = 2m$), we can redefine Q_n as follows

$$\begin{aligned} Q_n &= Q_{2m} \\ &= Q_2 \times Q_2 \times \cdots \times Q_2 \\ &= C_4 \times C_4 \times \cdots \times C_4 \\ &= T_{4,m} \end{aligned}$$

In other words, map $Z_2^2 \to Z_4$ as $00 \to 0$, $01 \to 1$, $11 \to 2$, $10 \to 3$. Thus, we can map $X = (x_{2m-1}, x_{2m-2}, \cdots, x_0)$ to $X' = (x'_{m-1}, x'_{m-2}, \cdots, x'_0)$, where $x_i \in Z_2$, and $x'_i \in Z_4$.

$$h_i(X; Z_2^{2m}) = h_i(X'; Z_4^m) \tag{5.8}$$

$\underline{n = 2m+1,\ k = 2, \text{ and } m \ge 1:}$

If n is odd, $n = 2m + 1$.

$$\begin{aligned} Q_n &= Q_{2m+1} \\ &= Q_{2m} \times Q_1 \end{aligned}$$

$$h_i(X; Z_2^{2m+1}) = \begin{cases} 0||h_i(X; Z_2^{2m}), \text{ if } X < 2^{2m} \\ 1||h_i(2^{2m+1} - X; 2^{2m}), \text{ otherwise} \end{cases} \tag{5.9}$$

5.9. Edge Disjoint Hamiltonian Cycles in a 2D torus

In this section, two functions which generate two disjoint Hamiltonian cycles for a 2D torus are described. Before giving these functions, first some useful operations and definitions are given.

Basic Operations:

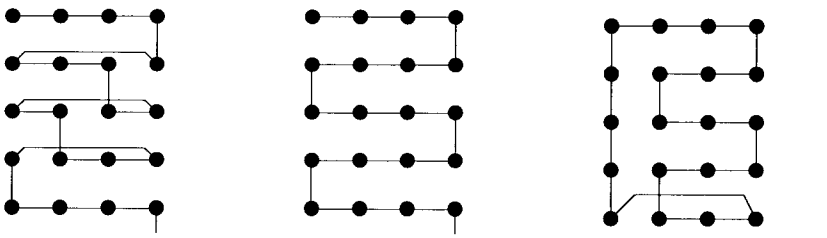
- $\rho(x) = x \bmod 2$.
- $reverse((x_1, x_0)) = (x_1, x_0)^R = (x_0, x_1)$.
- $(x_1, x_0) \bmod (k_1, k_0) = (x_1 \bmod k_1, x_0 \bmod k_0)$.
- $translate(+) : (x_1, x_0) + (d_1, d_0) = (x_1 + d_1, x_0 + d_0)$.
- $translate(-) : (x_1, x_0) - (d_1, d_0) = (x_1 - d_1, x_0 - d_0)$.

Spiral Mapping: Figure 5.8(a) is the graphical view of this mapping.

$$G_s(X; k) = G_s((x_1, x_0); k) = (x_1, (x_0 - x_1) \bmod k)$$
$$G_s^{-1}((y_1, y_0); k) = (y_1, (y_1 + y_0) \bmod k).$$

Maze Mapping: Figure 5.8(b) is the graphical view of this mapping.

$$G_m(X; k) = G_m((x_1, x_0); k) = (x_1, (1 - 2\rho(x_1))x_0 + \rho(x_1)(k - 1))$$
$$\begin{aligned} G_m^{-1}((y_1, y_0); k) &= y_1 k + (1 - 2\rho(y_1))y_0 + \rho(y_1)(k - 1) \\ &= (y_1, \rho(y_1)(k - 1 - y_0) + (1 - \rho(y_1)y_0)) \end{aligned}$$



(a) Spiral (b) Maze (c) Maze + feedback

FIGURE 5.8. Basic mappings

Maze with feedback: Figure 5.8(c) is the graphical view of this mapping.

$$G_{mf}(X; k_1, k_0) = \begin{cases} G_m(X; k_0 - 1), & \text{if } x < k_1 k_0 - k_1 \\ -X, & \text{if } k_1 k_0 - k_1 \leq x \end{cases}$$

In the spiral/maze mappings, X is the integer sequence, and $x_1 = \lfloor \frac{X}{k} \rfloor$ and $x_0 = X \bmod k$.

5.9.1. $k_1 = k + 2r$, and $k_0 = k + 2s$

Without loss of generality, we assume $k_1 = k + 2r$ and $k_0 = k + 2s$ for some $k \geq 3$, $r \geq 0$, and $s \geq 0$.

Definition 5.1 *If $k_1 = k + 2r$, and $k_0 = k + 2s$ for some k, $r \geq 0$, and $s \geq 0$, define a function, $h_0(X; T_{k_1,k_0})$ as follows*

$$h'_0(X; T_{k_1,k_0}) = \begin{cases} G_s(X + 2s; k_0) - (0, 2s), & \text{if } 0 \leq X < p_\alpha \\ G_m^R(X - p_\alpha; k_1 - k + 2) + (k - 1, k), & \text{if } p_\alpha \leq X < p_\beta \\ G_m(X - p_\beta; k) + (k, 0), & \text{if } p_\beta \leq X \end{cases}$$

$$h_0(X; T_{k_1,k_0}) = h'_0(X; T_{k_1,k_0}) \bmod (k_1, k_0).$$

where, $p_\alpha = (k-2)k_0 + 2k - 1$, *and* $p_\beta = k_1 k_0 - k(k_1 - k)$. □

Note that $h_0(p_\alpha; T_{k_1,k_0}) = (k-1, k \bmod k_0)$, and $h_0(p_\beta; T_{k_1,k_0}) = (k \bmod k_1, 0)$.

Theorem 5.5 *The function* $h_0(X; T_{k_1,k_0})$ *generates a Hamiltonian cycle,* $H_0(T_{k_1,k_0})$, *in a 2D torus* (T_{k_1,k_0}).

Proof: The proof has two parts.

(1) If $X \neq X'$ then $h_0(X; T_{k_1,k_0}) \neq h_0(X'; T_{k_1,k_0})$.

Assume $Y = (y_1, y_0) = h_0(X; T_{k_1,k_0})$. By Definition 5.1, the range of Y can be found from the range of X. If these ranges are disjoint, then the claim will be true.

$$\begin{aligned} R_1 = \{h_0(X) | 0 \leq X < p_\alpha\} &= \{(y_1, y_0) | y_1 = 0, 0 \leq y_0 < k\} \cup \\ &\quad \{(y_1, y_0) | 0 < y_1 < k-1, 0 \leq y_0 < k_0\} \cup \\ &\quad \{(y_1, y_0) | y_1 = k-1, 1 \leq y_0 < k\} \\ R_2 = \{h_0(X) | p_\alpha \leq X < p_\beta\} &= \{(y_1, y_0) | y_1 = 0, k \leq y_0 < k_0\} \cup \\ &\quad \{(y_1, y_0) | k-1 \leq y_1 < k_1, k \leq y_0 < k_0\} \\ R_3 = \{h_0(X) | p_\beta \leq X\} &= \{(y_1, y_0) | k \leq y_1 < k_1, 0 \leq y_0 < k\} \end{aligned}$$

Since R_1, R_2, and R_3 are mutually exclusive, the claim is true.

(2) $D_L(h_0(X; T_{k_1,k_0}), h_0(X'; T_{k_1,k_0})) = 1$ if $X = X' + 1$. In each subrange, the proof is trivial. The only cases that need to be considered are the situations where there is a transition from one subrange to another subrange. $h_0(p_\alpha - 1; T_{k_1,k_0}) = (k-1, k-1)$, $h_0(p_\alpha; T_{k_1,k_0}) = (k-1, k)$, $h_0(p_\beta - 1; T_{k_1,k_0}) = (k-1, 0)$, and $h_0(p_\beta; T_{k_1,k_0}) = (k, 0)$.

□

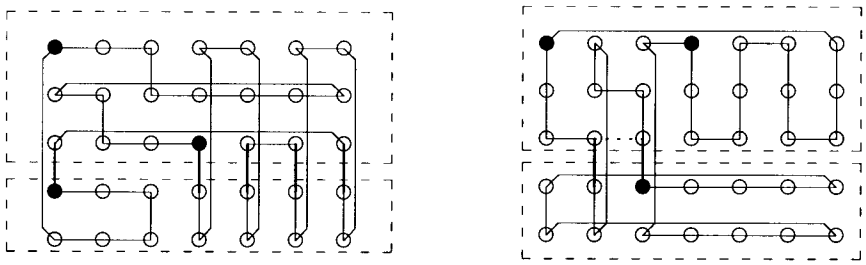


FIGURE 5.9. H_0 and H_1 in $T_{5,7}$ $(k = 3)$.

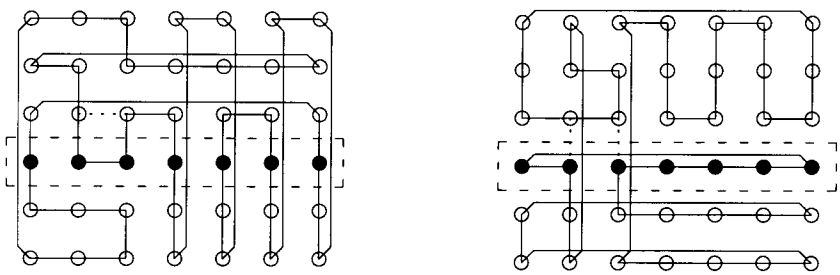


FIGURE 5.10. H_0 and H_1 in $T_{6,7}$ $(k = 3)$.

Corollary 5.1 *(Inverse of $h_0(X; T_{k_1,k_0})$).* $h_0^{-1}((y_1, y_0); T_{k_1,k_0})$ *is described as follows;*

$$h_0^{-1}(Y; T_{k_1,k_0}) = \begin{cases} G_s^{-1}(Y + (0, 2s); k_0) - 2s, & \text{if } Y \in R_1 \\ G_m^{-1}((Y - (k-1, k))^R; k_1 - k + 2) + p_\alpha, & \text{if } Y \in R_2 \\ G_m^{-1}(Y - (k, 0); k) + p_\beta, & \text{if } Y \in R_3 \end{cases}$$

□

Theorem 5.6 *If $H_0(T_{k_1,k_0})$ and $H_1(T_{k_1,k_0})$ are the edge-disjoint Hamiltonian cycles, and $H_0(T_{k_1,k_0})$ is generated by the function $h_0(X;T_{k_1,k_0})$, then the generator function $h_1(X;T_{k_1,k_0})$ for $H_1(T_{k_1,k_0})$ is defined as*

$$h_1(X;T_{k_1,k_0}) = h_0^R(X;T_{k_0,k_1})$$

Proof: (By Induction on k_1 and k_0)

Base step: Consider $T_{k,k}$.

We get $h_0(X;T_{k,k}) = G_s(X;k)$, and $h_1^R(X;T_{k,k}) = G_s^R(X;k)$. Thus $H_0(T_{k,k})$ and $H_1(T_{k,k})$ are edge-disjoint.

Inductive step: Assume that a T_{k_1,k_0}, where $k_1 = k+2r$ and $k_0 = k+2s$, has two edge-disjoint Hamiltonian cycles generated by $h_0(X;T_{k_1,k_0})$ and $h_1(X;T_{k_1,k_0})$.

Case $k_1' = k_1 + 2$: Figure 5.9 illustrates the process of adding two rows to a $T_{3,7}$.

Case $k_0' = k_0 + 2$: Similar to the previous case.

□

5.9.2. $k_1 = 4 + 2r$, and $k_0 = 3 + 2s$

Without loss of generality, we assume $k_1 = 4+2r$, and $k_0 = 3+2s$. We treat the T_{k_1,k_0} as a torus obtained after inserting a row to T_{k_1-1,k_0}. For example, $T_{6,7}$ is the torus obtained from $T_{3,3}$ after inserting rows and columns as in Figure 5.10.

Corollary 5.2 *By inserting a row in between the third and the fourth row of T_{k_1-1,k_0}, the function $h_0(X;T_{k_1,k_0})$ which generates a Hamiltonian cycle, $H_0(T_{k_1,k_0})$, can be described as follows.*

$$h'_0(X;T_{k_1,k_0}) = \begin{cases} G_s(X+2s;k_0)-(0,2s), & \text{if } 0 \le X < k_0+3 \\ G_m^R(X-k_0-3;2)+(2,1), & \text{if } k_0+3 \le X < k_0+7 \\ G_m^R(X-k_0-7;k_1-1)+(2,3), & \text{if } k_0+7 \le X < p_\beta \\ G_m(X-p_\beta;3)+(4,0), & \text{if } p_\beta \le X \end{cases}$$

$$h_0(X;T_{k_1,k_0}) = h'_0(X;T_{k_1,k_0}) \ mod\ (k_1,k_0).$$

where, $p_\beta = k_1k_0 - 3(k_1-4)$. □

Corollary 5.3 *The second function* $h_1(X;T_{k_1,k_0})$*, where* k_1 *is even and* k_0 *is odd, is as follows*

$$h'_1(X;T_{k_1,k_0}) = \begin{cases} G_s^R(X;3), & \text{if } 0 \le X < 4 \\ G_s^R(p_\alpha-1-X;k_1)+(3,1), & \text{if } 4 \le X < p_\alpha \\ G_m(X-p_\alpha+k_0-1;k_0-1)+(2,2), & \text{if } p_\alpha \le X < p_\beta \\ G_m^R(X-p_\beta;3)+(0,3), & \text{if } p_\beta \le X \end{cases}$$

$$h_1(X;T_{k_1,k_0}) = h'_1(X;T_{k_1,k_0}) \ mod\ (k_1,k_0).$$

where, $p_\alpha = k_1+5$, $p_\beta = k_1k_0 - 3(k_0-3)$. □

6. FUTURE RESEARCH

This chapter describes some open research problems related to the topics discussed in this thesis.

6.1. Perfect Placement in a Torus

Although the perfect placement obtained using Lee distance codes are interesting, they give solutions for a limited number of cases. Thus general solutions to those problems, especially for two and three dimensional cases, can be further investigated. Some specific problems are described below.

Can we have a perfect distance-1 placement in a torus of size $k_{n-1} \times k_{n-2} \times \cdots \times k_0$? We have proved that a perfect distance-1 placement is possible in a two dimensional torus of size $k_1 \times k_0$ iff $5|k_1$ and $5|k_0$. For other dimensions, this is still an open problem. For example, we have a perfect distance-1 placement in a 3 dimensional torus of size $k_2 \times k_1 \times k_0$ provided $7|k_2$, $7|k_1$, and $7|k_0$. However, it is not clear, whether we will have a perfect distance-1 placement if 7 divides any one dimension but not the other two. What are the conditions for perfect placement, what is the minimum numbers of resources needed for perfect placement, where to place these resource nodes using minimum number of resources, etc, are some of the problems that can be investigated.

6.2. Quasi Perfect Placement

When it is not possible to have a perfect placement, the best one can expect to have is a quasi-perfect placement. For example, in a 16×16 torus, using 16 I/O nodes, at best one can have is a quasi-perfect distance-2 placement. This is because at most 16×13 nodes will be at a distance of 2 or less from some resource nodes and the remaining nodes must be at a distance of at least 3 from some resource nodes. How to obtain these quasi-perfect placement, especially for two and three dimensional tori, can be investigated.

6.3. I/O node Placement and Message Latency

In the past, the speed and capacity of main memory and processor power have increased many folds; however the throughput and capacity of the I/O subsystems have not kept up with this performance increase. Thus, the I/O subsystem has become the bottleneck for overall system performance, especially when the application is I/O intensive.

Some novel schemes for I/O placement strategy in hypercubes have been proposed by authors in [84, 83] and further studied by other authors in [51]. The schemes use perfect distance-1 and quasi-perfect distance-1 placements based on single error correcting Hamming codes. For many problems, the performance improvement based on these placements are reported in [83, 51]. In many cases, the speed up obtained by perfect / quasi-perfect placement is quite high. Further, the importance of data distribution and the actual data distribution schemes for many problems are also described in [83]. The research studies given in [83, 51] can be extended to torus networks. Note that, in the case of hypercubes, one can obtain perfect distance-t placements for only $t = 1$ using coding theory results. (The

only other perfect code - Golay code, gives perfect distance-3 placement in a Q_{23} (23-dimensional hypercube) and so, is not of much practical use for placement problems). However, in the case of torus, especially for two dimensional case, perfect distance-t and quasi-perfect distance-t placements can be obtained for many values of t. The possible performance improvement of these perfect distance-t placements for various problems can be investigated. Further, in torus networks, extending the results given in [83], the data distribution methods for various problems can also be investigated.

Most of the existing systems use some simple schemes for I/O placement. For example, in INTEL PARAGON (which is a 2D mesh), the I/O nodes are placed in one column. The performance improvement, in terms of message latency, of the proposed I/O placement methods over the existing ones can be investigated.

6.4. Edge Disjoint Hamiltonian Cycles and Gray Codes

In Chapter 5, it is shown that the usefulness of the edge disjoint cycles, especially for the communication algorithms between processors, and the way of generating the edge disjoint cycles in k-ary n-cubes and hypercubes. However a straightforward way of generating such disjoint cycles in a multidimensional torus is not clear. One of the interesting problems is how to extend these results to multi-dimensional tori.

6.5. Embedding Problems

Algorithms/programs developed in one topology network (guest graph) can be easily ported to another topology network (host graph) provided there is an efficient embedding algorithm which maps the guest graph on the host graph [59,

53, 54, 67, 65, 73, 95]. In [67], some efficient algorithms for embedding trees on hypercube are described. These algorithms are based on the binary single error correcting Hamming codes. The application of single error correcting Lee distance code for tree embedding on torus can be investigated.

BIBLIOGRAPHY

[1] Vikram S. Adve and Mary K. Vernon, "Performance analysis of mesh interconnection networks with deterministic routing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 3, pp. 225–246, March 1994.

[2] Anant Agarwal, "Limits on interconnection network performance," *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398–412, October 1991.

[3] M. Sultan Alam and Rami G. Melhem, "Routing in modular fault tolerant multiprocessor systems," *IEEE International Symposium on Fault-Tolerant Computing*, pp. 185–193, 1992.

[4] M. Sultan Alam and Rami G. Melhem, "Channel multiplexing in fault-tolerant modular multiprocessors," *Journal of Parallel and Distributed Computing*, vol. 24, pp. 115–131, 1995.

[5] B. Alspach, J.-C. Bermond, and D. Sotteau, "Decomposition into cycles I: Hamiltonian decompositions," In *Cycles and Rays*, pp. 9–18. Gena Hahn and Geri Sabidussi and Robert E. Woodrow, ed., Kluwer Academic Publishers, 1990.

[6] Marco Annaratone, Marco Fillo, Kiyoshi Nakabayashi, and Marc Viredaz, "The K2 parallel processor: Architecture and hardware implementation," In *Proc. 17th Annual International Symposium on Computer Architecture*, pp. 92–101. Seattle, WA, May 1990.

[7] William C. Athas and Charles L. Seitz, "Multicomputers: Message-passing concurrent computers," *IEEE Computer*, vol. 21, no. 8, pp. 9–24, August 1988.

[8] Didier Badouel, Charles A. Wüthrich, and Eugene L. Fiume, "Routing strategies and message contention on low-dimensional interconnection networks," Technical report, Computer System Research Institute, December 1991.

[9] Myung M. Bae and Bella Bose, "On independent set of Lee distance Gray codes," *Submitted to IEEE International Symposium on Information Theory*, 1996.

[10] Myung M. Bae and Bella Bose, "Resource placement in torus-based networks," In *Proc. IEEE International Parallel Processing Symposium*, pp. 327–331, April 1996.

[11] Myung M. Bae and Bella Bose, "Spare processor allocation for the fault-tolerance in torus-based multicomputers," In *Proc. 26th Annual International Symposium on Fault-Tolerant Computing*, pp. 282–291, June 1996.

[12] Myung M. Bae and Bella Bose, "Resource placement in torus-based networks," *IEEE Transactions on Computers*, (Accepted with minor revisions).

[13] Prithviraj Banerjee and Michael Peercy, "Design and evaluation of hardware strategies for reconfiguring hypercubes and meshes under faults," *IEEE Transactions on Computers*, vol. 43, no. 7, pp. 841–848, July 1994.

[14] Kennet E. Batcher, "Bit-serial parallel processing systems," *IEEE Transactions on Computers*, vol. C-31, pp. 377–384, May 1982.

[15] Elwyn R Berlekamp, *Algebraic Coding Theory*, McGraw-Hill, New York, 1968.

[16] Pablo E. Berman, Luis Gravano, Gustavo D. Pifarreé, and Jorge L. C. Sanz, "Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks," *ACM Symposium on Parallel Algorithms and Architectures*, pp. 3–12, June 1992.

[17] J.-C. Bermond, O. Favaron, and M. Maheo, "Hamiltonian decomposition of Cayley graphs of degree 4," *Journal of Combinational Theory, Series B*, vol. 46, pp. 142–153, 1989.

[18] Laxmi N. Bhuyan and Dharma P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Transactions on Computers*, vol. C-33, no. 4, pp. 323–333, April 1984.

[19] Rajendra V. Boppana and Suresh Chalasani, "A comparison of adaptive wormhole routing algorithms," In *Proc. of the 20th Annual International Symposium on Computer Architecture*, pp. 351–360, May 1993.

[20] Shekhar Borkar, Robert Cohn, George Cox, Sha Gleason, Thomas Gross, H. T. Kung, Monica Lam, Brian Moore, Craig Peterson, John Pieper, Linda Rankin, P. S. Tseng, Jim Sutton, John Urbanski, and Jon Webb, "iWrap: An intergrated solution to high-speed parallel computing," In *IEEE Proceedings of Supercomputing '88*, pp. 330–339, November 1988.

[21] Bella Bose and Bob Broeg, "Lee distance Gray codes," *IEEE International Symposium on Information Theory*, September 1995.

[22] Bella Bose, Bob Broeg, Yoyunggeun Kwon, and Yaagoub Ashir, "Lee distance and topological properties of k-ary n-cubes," *IEEE Transactions on Computers*, vol. 44, no. 8, pp. 1021–1030, August 1995.

[23] Robert Richard Broeg, *Topics in Toroidal Interconnection Networks*, PhD thesis, Computer Science, Oregon State University, 1995.

[24] Richard A. Brualdi, Vera S. Pless, and Richard M. Wilson, "Short codes with a given covering radius," *IEEE Transactions on Information Theory*, vol. 35, no. 1, pp. 99–109, January 1989.

[25] Jehoshua Bruck, Robert Cypher, and Ching-Tien Ho, "Efficient fault-tolerant mesh and hypercube architectures," In *IEEE International Symposium on Fault-Tolerant Computing*, pp. 162–169. Boston, WA, 1992.

[26] Jehoshua Bruck, Robert Cypher, and Ching-Tien Ho, "Fault-tolerant meshes and hypercubes with minimal number of spares," *IEEE Transactions on Computers*, vol. 42, no. 9, pp. 1089–1103, September 1993.

[27] Jehoshua Bruck, Robert Cypher, and Danny Soroker, "Tolerating faults in hypercubes using subcube partitioning," *IEEE Transactions on Computers*, vol. 41, no. 5, pp. 599–605, May 1992.

[28] Jean-Phiippe Brunet and S. Lennart Johnsson, "All-to-all broadcast and applications on the connection machine," *The International Journal of Supercomputer Applications*, vol. 6, no. 3, pp. 241–256, 1992.

[29] M. Y. Chan and S. J. Lee, "On the existence of Hamiltonian circuits in faulty hypercubes," *SIAM J. Disc. Math.*, vol. 4, no. 4, pp. 511–527, November 1991.

[30] Hsing-Lung Chen and Nian-Feng Tzeng, "Fault-tolerant resource placement in hypercube computers," In *Proc. International Conference on Parallel Processing*, vol. 1, pp. 517–524, 1991.

[31] Hsing-Lung Chen and Nian-Feng Tzeng, "Efficient resource placement in hypercubes using multiple-adjacency codes," *IEEE Transactions on Computers*, vol. 43, no. 1, pp. 23–33, January 1994.

[32] Andrew A. Chien, "A cost and speed model for k-ary n-cube wormhole routers," In *Proc. of the 19th International Symposium on Computer Architecture*, pp. 268–277. Association for Computing Machinary, 1992.

[33] Ge-Ming Chiu and C. S. Raghavendra, "Resource allocation in hypercube systems," In *Proc. 5th Distributed Memory Computing*, vol. 1, pp. 894–902, April 1990.

[34] G. D. Cohen, M. R. Karpovsky, and H. F. Mattson, "Covering radius – survey and recent results," *IEEE Transactions on Information Theory*, vol. IT-31, no. 3, pp. 328–343, May 1985.

[35] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest, *Introduction to Algorithms*, The MIT Press, Cambridge, Massachusetts, 6th edition, 1992.

[36] Paul Cull, "Tours of graphs, digraphs, and sequential machines," *IEEE Transactions on Computers*, vol. C-29, no. 1, pp. 50–54, January 1980.

[37] Paul Cull, "Hamiltonian circuits in additive machines," Technical report, Oregon State University, 1982.

[38] Paul Cull and Shawn M. Larson, "Wormhole routing algorithms for twisted cube networks," In *Proc. 6th IEEE Symposium on Parallel and Distributed Processing*, pp. 696–703, 1994.

[39] Paul Cull and Ingrid Nelson, "Error-correcting codes on the towers of hanoi graphs," *To be published.*

[40] William J. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775–785, June 1990.

[41] William J. Dally, "Express cubes: Improving the performance of k-ary n-cube interconnection networks," *IEEE Transactions on Computers*, vol. 40, no. 9, pp. 1016–1023, September 1991.

[42] William J. Dally, "Virtual-channel flow control," *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194–205, March 1992.

[43] William J. Dally and Charles L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Transactions on Computers*, vol. C-36, no. 5, pp. 547–553, May 1987.

[44] José Duato, "A new theory of deadlock-free adaptive routing in wormhole networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, December 1993.

[45] Ralph Duncan, "A survey of parallel computer architectures," *IEEE Computer*, vol. 23, no. 2, pp. 5–16, February 1990.

[46] T. H. Duncan, "Performance of the intel iPSC/860 and NCUBE 6400 hypercubes," *Parallel Computing*, vol. 17, pp. 1285–1302, 1991.

[47] Shantanu Dutt and John P. Hayes, "An automorphic approach to the design of fault-tolerant multiprocessors," *IEEE International Symposium on Fault-Tolerant Computing*, pp. 496–503, 1989.

[48] Shantanu Dutt and John P. Hayes, "On designing and reconfiguring k-fault-tolerant tree architectures," *IEEE Transactions on Computers*, vol. 39, no. 4, pp. 490–503, April 1990.

[49] Shantanu Dutt and John P. Hayes, "Some practical issues in the design of fault-tolerant multiprocessors," *IEEE Transactions on Computers*, vol. 41, no. 5, pp. 588–598, May 1992.

[50] Marsha F. Foregger, "Hamiltonian decompositions of products of cycles," *Discrete Mathematics*, vol. 24, pp. 251–260, 1978.

[51] J. Ghosh, K. Goveas, and J. Drapper, "Performance evaluation of parallel I/O subsystems for hypercube multicomputers," *Journal of Parallel and Distributed Computing*, pp. 90–106, January 1993.

[52] Solomon W. Golomb and Lloyd R. Welch, "Algebraic coding and the Lee metric," In *Error Correcting Codes: Proceedings of a Symposium Conducted by the Mathematics Research Center*, pp. 175–194. Edited by Henry B. Mann, John Wiley & Sons, Inc., May 1968.

[53] Antonio González and Miguel Valero-García, "The xor embedding: Embedding hypercubes onto rings and toruses," In *Proc. Int'l Conference on Application-Specific Array Processors*, pp. 15–28, 1993.

[54] Antonio González, Miguel Valero-García, and Luis Díaz de Cerio, "Executing algorithms with hypercube topology on torus multicomputers," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 8, pp. 803–813, August 1995.

[55] R. L. Graham and N. J. A. Sloane, "On the covering radius of codes," *IEEE Transactions on Information Theory*, vol. IT-31, no. 3, pp. 385–401, May 1985.

[56] Luis Gravano, Gustavo D. Pifarré, Pablo E. Berman, and Jorge L. C. Sanz, "Adaptive deadlock- and livelock-free routing with all minimal paths in torus networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 12, pp. 1233–1251, December 1994.

[57] Nora Hartsfield and Gerhard Ringle, *Pearls in Graph Theory: A Comprehensive Introduction*, Academic Press, San Diego, CA, 1990.

[58] John P. Hayes, "A graph model for fault-tolerant computing systems," *IEEE Transactions on Computers*, vol. C-25, no. 9, pp. 875–884, September 1976.

[59] C. T. Ho and S. L. Johnsson, "On the embedding of meshes in boolean cubes," In *Proc. International Conference on Parallel Processing*, pp. 188–191, 1987.

[60] Ching-Tien Ho and Ming-Yang Kao, "Optimal broadcast in all-port wormhole-routed hypercubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 2, pp. 200–204, February 1995.

[61] Yixiu Huang, "On Hamiltonian decompositions of Cayley graphs on cyclic groups," *Annals of the New York Academy of Sciences*, vol. 576, pp. 250–258, 1989.

[62] S. Lennart Johnsson and Ching-Tien Ho, "On the conversion between binary code and binary-reflected Gray code on binary cubes," *IEEE Transactions on Computers*, vol. 44, no. 1, pp. 47–53, January 1995.

[63] Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, The Benjamin-Cummings Publishing Co., 1994.

[64] Sun-Yuan Kung, Shiann-Ning Jean, and Chih-Wei Chang, "Fault-tolerant array processors using single-track switches," *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 501–514, April 1989.

[65] Ten-Hwang Lai and Alan P. Sprague, "Placement of the processors of a hypercube," *IEEE Transactions on Computers*, vol. 40, no. 6, pp. 714–722, June 1991.

[66] C. Y. Lee, "Some properties of nonbinary error-correcting codes," *IRE Transactions on Information Theory*, pp. 77–82, June 1958.

[67] F. Thomson Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*, Morgan Kaufmann Publishers, Inc., San Mateo, California, 1992.

[68] Daniel Lenoski, James Laudon, Kourosh Gharachorloo, Anoop Gupta, and John Hennessy, "The directory cache coherence protocol for the DASH multicomputer," In *17th Annual International Symposium on Computer Architecture*, pp. 148–159. Seattle, WA, May 1990.

[69] Shu Lin and Jr. Daniel J. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1983.

[70] Daniel H. Linder and Jim C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cubes," *IEEE Transactions on Computers*, vol. 40, no. 1, pp. 2–12, January 1991.

[71] Marilynn Livingston and Quentin F. Stout, "Distributing resources in hypercube computers," In *Proc. 3rd Conference on Hypercube Concurrent Computers and Applications*, vol. 1, pp. 222–231, 1988.

[72] S. P. Lloyd, "Binary block coding," *The Bell System Technical Journal*, pp. 517–535, March 1957.

[73] Y. E. Ma and L. Tao, "Embeddings among toruses and meshes," In *Proc. International Conference on Parallel Processing*, pp. 178–187, 1987.

[74] Lionel M. Ni and Philip K. McKinley, "A survey of wormhole routing techniques in direct networks," *IEEE Computer*, pp. 62–76, February 1993.

[75] Michael D. Noakes, Deborah A. Wallach, and William J. Dally, "The J-machine multicomputer: An architectural evaluation," In *20th International Symposium on Computer Architectures*, pp. 224–235, 1993.

[76] Wilfried Oed, "Massively parallel processor system CRAY T3D," Technical report, Cray Research GmbH, November 1993.

[77] Dhabaleswar K. Panda and Sanjay Singal, "Broadcasting in k-ary n-cube wormhole routed networks using path-based routing," In *Proc. of the 8th International Parallel Processing Symposium*, 1994.

[78] S. Park and Bella Bose, "All to all broadcasting in faulty hypercubes," *IEEE Transactions on Computers*, To be published 1997.

[79] W. Wesley Peterson and Jr. E. J. Weldon, *Error-Correcting Codes*, The MIT Press, Cambridge, Massachusetts, and London, second edition, 1972.

[80] C. S. Raghavendra, Pei-Ji Yang, and Sing-Ban Tien, "Free dimensions – an effective approach to achieving fault tolerance in hypercubes," *IEEE Transactions on Computers*, vol. 44, no. 9, pp. 1152–1157, February 1995.

[81] Parameswaran Ramanathan and Suresh Chalasani, "Resource placement in k-ary n-cubes," In *Proc. International Conference on Parallel Processing*, vol. 2, pp. 133–140, 1992.

[82] Parameswaran Ramanathan and Suresh Chalasani, "Resource placement with multiple adjacency constraints in k-ary n-cubes," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 5, pp. 511–519, May 1995.

[83] A. L. Narasimha Reddy and P. Banerjee, "Design, analysis, and simulation of I/O architectures for hypercube multiprocessors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 1, no. 2, pp. 140–151, April 1990.

[84] A. L. Narasimha Reddy, P. Banerjee, and Santosh G. Abraham, "I/O embedding in hypercubes," In *Proc. International Conference on Parallel Processing*, pp. 331–338, August 1988.

[85] Youcef Saad and Martin H. Schultz, "Topological propertices of hypercubes," *IEEE Transactions on Computers*, vol. 37, no. 7, pp. 867–872, July 1988.

[86] Charles L. Seitz, "The cosmic cube," *Communications of the ACM*, vol. 28, no. 1, pp. 22–33, January 1985.

[87] Charles L. Seitz and et al., "The architecture and programming of the Ametek series 2010," In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pp. 33–37, January 1988.

[88] Charles L. Seitz and et al, "Submicron systems architecture project semiannual technical report," Technical Report Caltec-CS-TR-88-18, California Institute of Technology, November 1988.

[89] Richard Stong, "Hamiltonian decomposition of Cartesian products of graphs," *Discrete Mathematics*, vol. 90, pp. 169–190, 1991.

[90] Tera Computer Systems, *Overview of the Tera Parallel Computer*, 1993.

[91] Nian-Feng Tzeng and Gui-Liang Feng, "On resource allocation in binary n-cube systems," In *Proc. International Conference on Parallel Processing*, vol. II, pp. 209–212, August 1993.

[92] Nian-Feng Tzeng and Gui-Liang Feng, "Resource allocation in cube network systems based on the covering radius," *IEEE Transactions on Parallel and Distributed Systems*, pp. 328–342, April 1996.

[93] Werner Ulrich, "Non-binary error correction codes," *The Bell System Technical Journal*, pp. 1341–1388, November 1957.

[94] Theodora A. Varvarigou, Vwani P. Roychowdhury, and Thomas Kailath, "Reconfiguring processor arrays using multiple-track models: The 3-track-1-spare-approach," *IEEE Transactions on Computers*, vol. 42, no. 11, November 1993.

[95] Yuen wah Eva and Lixin Tao, "Embedding among toruses and meshes," *Journal of Parallel and Distributed Computing*, vol. 18, pp. 44–55, 1993.

[96] Minghsien Wang, Michal Cutler, and Stephen Y. H. Su, "Reconfiguration of VLSI/WSI mesh array processors with two-level redundancy," *IEEE Transactions on Computers*, vol. 38, no. 4, pp. 547–554, April 1989.

[97] David Witte and Joseph A. Gallian, "A survey: Hamiltonian cycles in Cayley graphs," *Discrete Mathematics*, vol. 51, pp. 293–304, 1984.

[98] Glenn Zorpette, "Technology 1991: Minis and mainframes," *IEEE Spectrum*, vol. 28, pp. 40–43, January 1991.