01 Dec 1993

Fault-Tolerant Ring Embeddings in Hypercubes -- A Reconfigurable Approach

Jun-Lin Liu

Bruce M. McMillin
Missouri University of Science and Technology, ff@mst.edu

Fault-Tolerant Ring Embeddings in Hypercubes -- A Reconfigurable Approach

J. L. Liu* and B. McMillin

CSC-93-32

Department of Computer Science
University of Missouri-Rolla
Rolla, MO 65401

*This report is substantially the Ph. D. dissertation of the first author, completed December 1993

Abstract

We investigate the problem of designing reconfigurable embedding schemes for a fixed hypercube (without redundant processors and links). The fundamental idea for these schemes is to embed a basic network on the hypercube without fully utilizing the nodes on the hypercube. The remaining nodes can be used as spares to reconfigure the embeddings in case of faults. The result of this research shows that by carefully embedding the application graphs, the topological properties of the embedding can be preserved under fault conditions, and reconfiguration can be carried out efficiently.

In this dissertation, we choose the ring as the basic network of interest, and propose several schemes for the design of reconfigurable embeddings with the aim of minimizing reconfiguration cost and performance degradation. The cost is measured by the number of node-state changes or reconfiguration steps needed for processing of the reconfiguration, and the performance degradation is characterized as the dilation of the new embedding after reconfiguration. Compared to the existing schemes, our schemes surpass the existing ones in terms of applicability of schemes and reconfiguration cost needed for the resulting embeddings.

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Bruce McMillin for his help and guidance at every stage of my graduate study at UMR. I learned from him not only the research methods, but also the attitude to which a researcher should commit himself (herself). I would also like to thank Dr. Thomas Sager for his idea of the composition method in Chapter VI and his interest in our work. I also appreciate Dr. Arlan DeKock, Dr. C. Y. Ho, Dr. Thomas Sager, and Dr. Tim Randolph for serving on my committee.

I am also thankful to all my friends at UMR who kept me company, and my office-mates at MCS 208 who worked with me and provided me with many stimulating ideas and discussion.

I am most grateful to my parents who supported me mentally and financially. Specially my mother who encouraged me all the time and helped me through some tough times. Also, thanks to my two sisters for their encouragement, and a very special friend of mine, Miss Yin, for her sincere friendship, encouragement, and patience.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure

LIST OF TABLES

Table

I. INTRODUCTION

The need for the embedding arises from at least two different directions. First, with the widespread availability of distributed memory architectures based on the hypercube interconnection scheme, there is an ever-growing interest in the portability of algorithms developed for architectures based on other topologies, such as linear arrays, rings, two-dimensional meshes, and complete binary trees, into the hypercube. Clearly, this question of portability reduces to one of embedding the above interconnection schemes into the hypercube. Second, the problem of mapping parallel algorithms onto parallel architectures naturally gives rise to graph embedding problems. Graph embedding problems have applications in a wide variety of computational situations. For example, the flow of information in a parallel algorithm defines a program graph and embedding this into a network tells us how to organize the computation on the network. Other problems that can be formulated as graph embedding problems are laying out circuits on chips, representing data structures in computing memory, and finding efficient program control structures.

A. What is an Embedding?

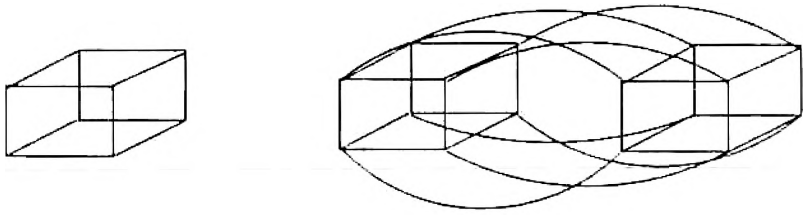
The problem of mapping a graph representing the computation and communication needs of the program onto the underlying physical interconnection of a multiprocessor so as to minimize the communication overhead and maximize the parallelism is called the mapping problem. Bokhari [Bokh81] defines the mapping problem as the assignment of processes to processors so as to maximize the number of pairs of communicating processes that fall on pairs of directly connected processors.

In mapping problems, the *guest graph* G is the network topology that we are interested in simulating using a *host graph* H. Let V_G and V_H denote the vertex sets of the graph G and H, respectively, and E_G and E_H denote the edge sets of the graph G and H, respectively. An *embedding* f of a graph G into a graph H is a mapping of the vertices of G into the vertices of H, together with a mapping of the edges of G into the simple paths of H such that if $e = (u, v) \in E_G$, then $f(e)$ is a simple path of H with endpoints $f(u)$ and $f(v)$. If $f(e)$ has length greater than one, then it has one or more intermediate nodes which are all nodes on the path other than the two endpoints. An embedding f is *isomorphic* if it is injective and for each $(u, v) \in E_G$, $(f(u), f(v)) \in E_H$. Throughout this dissertation, unless indicated otherwise the term "embeddings" will always means isomorphic embeddings, and the terms "embedding" and "mapping" will mean the same and will be used interchangeably.

B. Embeddings in a Hypercube

Various supercomputer architectures interconnecting hundreds or thousands of processors have been proposed for many years. Prominent is the binary hypercube which has emerged as one of the most important network architectures for large-scale concurrent computers. Several different versions of concurrent computers based on the hypercube architecture have been built at Intel, NCUBE, and Thinking Machines. An *d-dimensional binary hypercube* is a graph with 2^d vertices labeled by integers 0, 1, ..., $2^d - 1$ represented as d-bit binary strings, called *address*, and with edges joining two vertices whenever their addresses differ in a single bit. A 3-dimensional and a 4-dimensional hypercubes are shown in Figure 1(a) and 1(b), respectively. The hypercube offers a rich interconnection topology with high communication bandwidth, low diameter, and a recursive structure naturally suited to divide- and-conquer applications. More

importantly, the hypercube supports efficient routing algorithms and can therefore simulate many realistic network topologies, like linear arrays, rings, 2-D meshes, higher-dimensional meshes, trees, and pyramids. The topological properties of the hypercube were examined in [SaSc88].



(a) (b)

Figure 1. (a) A 3-dimensional hypercube. (b) A 4-dimensional hypercube.

It has been known for a long time that the general graph embedding problem (i.e., subgraph isomorphism problem) is NP-complete. It was shown that the embedding of general graphs into the binary hypercube is also NP-complete [AfPP85, KrVC86, CyKV87]. It remains NP-complete even if embedding arbitrary trees in a binary hypercube [WaCo90]. However, with rich interconnection structure the hypercube contains as a subgraph many the regular structures (i.e., rings, two-dimensional meshes, higher-dimensional meshes, and almost complete binary trees). Most of the mapping research in these years have dealt with effectively simulating these regular structures in the hypercubes, [Wuay85, BhIp85, BrSc85, Stou86, Chan88, ChCh88, Chan89a, Chan89b, HoJo89, HoJo90, LaWh90, Vara91].

C. Fault-Tolerant Embeddings in a Hypercube

As the complexity and the size of computers increase, the probability of processors and communication links becoming faulty during the operation of the system increases. For this reliability reason, the consideration of fault tolerance in the system is needed. In particular, it is crucial that such a system be able to withstand an accumulation of faults among a reasonable number of its components. Even for the latest CM-5 massively parallel computer, an important aspect of this design is rapid diagnosis and smooth degradation in the face of component failures. Failed components in CM-5 can be logically and electrically isolated from the rest of the system under control of the Diagnostic Network. Surrounding components are instructed to ignore any and all signals from failed components. The failed section of the system can then independently execute diagnostic tests or be powered down for repair or replacement, while the rest of the system continues normal operation [PaSt92].

For network mappings in a hypercube, embedding techniques which rely on a fault-free host graph break down in the presence of faults (nodes or links) in the hypercube. Even worse, if a fault occurs during processing, the guest graph must be reembedded before processing can continue. In a time-critical application or an application which needs a massive amount of computation time, this would not be feasible. Hence, when constructing network embeddings, fault tolerance should also be taken into account. However, in this dissertation we do not consider the usage of redundant nodes or links to achieve fault tolerance in a hypercube. We assume, instead, that the number of nodes and links in a hypercube are fixed. This is a more realistic assumption, since in many cases, we do not have the luxury to build a specific host machine for an application. Based on this assumption of fixed host, the research of fault-tolerant embeddings in a hypercube

can be divided into two categories: *static* and *dynamic*. Static research explores the existence of network topologies in a hypercube with faulty nodes or links [WaOz90, ChLe91a, ChLe91b, ChCP91, WaCM91, YaTR91, WaCy92, LaZB92, TsLa93]. Dynamic research concerns finding intelligent strategies to map the basic network into the host so that reconfiguration can be easily carried out at runtime in the presence of faults [DeJe86, PrMe88, ChLT88a, ChLT88b, Leet90, LiTs90, BuPr90, KrHI91, LiMc92a, LiMc92b, LiSM92, PrMe92]. Because of the reconfiguration nature, the dynamic fault-tolerant embeddings is also categorized as *reconfigurable* embeddings.

D. Basic Definitions and Preliminaries

1. Terminologies for a Hypercube. An d-dimensional binary hypercube is denoted as *d-cube* or Q_d. We will usually identify the vertices of the hypercube with their addresses. Hence u = 010110 says that the address of the vertex u is 010110. The jth bit of the address of the vertex u is called jth *coordinate* of the vertex u. Each l-dimensional subcube of d-cube can be uniquely specified by its *address* - a d-bit string of symbols of the set $\sum = \{0,1,*\}$, where the symbol $*$ is a *don't care* symbol. Equally as for vertices, we will identify subcubes with their addresses. The set of all coordinates in which the addresses of two vertices u, v differ is denoted $\delta(u,v)$. Two vertices u, v on the hypercube is *adjacent along the jth coordinate* if $\delta(u,v) = \{j\}$. A *walk* on the hypercube is an alternating sequence of vertices and coordinates $v_0, c_1, v_1, c_2, \cdots, v_{n-1}, c_n, v_n$ beginning and ending with vertices, in which $\delta(v_{i-1},v_i) = \{c_i\}$.

A path of the hypercube is specified completely by listing the $L+1$ vertices $V_0, \cdots, V_L$ in order. For example, on the 3-cube, a list might be

000, 001, 011, 010, 110, 100, 101, 111.

Ignoring the starting vertex, a more compact notation is to list in order only the coordinate places in which the change occurs. In the example cited, one would obtain <3,2,3,1,2,3,2>. This L-tuple of coordinate places will called the *transition sequence* [Gilb58] (or *coordinate sequence* [Doug77]) for the path. In a similar way, a cycle or an embedding of a ring in the hypercube may be represented by a transition sequence.

According to [Gilb58], two cycles, C and C', are of the same type if and only if one of the $k!$ permutations of coordinates changes the transition sequence of C into the transition sequence of C'. Hence two cycles with transition sequences <1,2,3,1,2,3> and <3,2,1,3,2,1>, respectively, are of the same type. Although there are tremendous numbers of cycles, it suffices to consider one of each type. The canonical form of the coordinate sequence adopted in this dissertation, is to label the coordinates in the order of their frequency of occurrence. Where coordinates are equally frequent, we will normally label them according to their first appearance in the sequence, with the start of the sequence chosen to make the 2^k-digit number as small as possible. However, there are circumstances where a different ordering is more convenient for emphasizing a particular characteristic, and an alias may also be given.

$|P|$ where P is a path on the hypercube denotes the length (number of links) of P. For paths P_1, P_2, $\cdots$ on the hypercube, $T(P_1)$ and $T^*(P_1, P_2, \ldots)$ denote the transition sequence for P_1 and the concatenation of transition sequences for paths P_1, P_2, $\cdots$, respectively. $\tau(i, j, \ldots)$ denotes a permutation on digits $i, j, \cdots$.

2. Embedding Terminologies. In this dissertation, we will use the notation R_k to denote a length-k ring, and the notation $R_k \rightarrow Q_d \equiv S \mid T'$ to denote an embedding of a length-k ring in a d-cube, where S is the binary label of the starting node (0 for $00\cdots0$) and T' is the transition sequence of the embedding. For instance, the embedding of a length-12 ring in a 4-cube in Figure 2 is referred as $R_{12} \rightarrow Q_4 \equiv 0 \mid$ <1,2,3,1,2,4,1,2,3,1,2,4>. Throughout the figures in this dissertation, we will use directed arcs on the hypercube to denote a ring embedding.

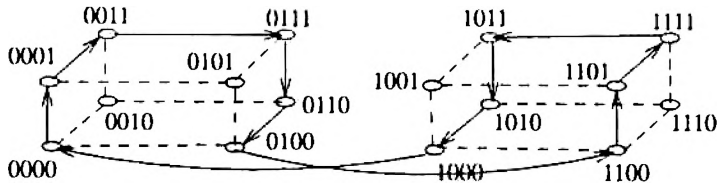


Figure 2. An embedding of a length-12 ring in a 4-cube.

Let f be an embedding function which maps a guest graph G into a host graph H. $|V_G|$ denotes the cardinality of the set V_G. Some terminology related to the mapping problem is formally defined as follows.

- The *expansion* of the mapping is the ratio of the size (in number of nodes) of the host graph to that of the guest graph, that is, $\frac{|V_H|}{|V_G|}$. (If the embedding is injective, then the expansion is a measure of processor utilization).

- The *edge dilation* of edge $(i,j) \in E_G$ is $dist(f(i),f(j))$. The *dilation* of the mapping is $max(dist(f(i),f(j)),\ \forall(i,\ j) \in E_G$. The *average edge dilation* is $\frac{1}{|E_G|}$

$\sum_{(i,j)\in E_G} dist(f(i), f(j))$. (The dilation issues of an mapping represents the communication delay between the communication nodes).

- The *congestion of an edge* $e' \in E_H$ is the cardinality of $\{e \in E(G)$: e' is in path $f(e)\}$ That is, $\sum_{e\in E_G} |e' \cap E_{f(e)}|$. The *congestion* of the mapping is $\max \{ \sum_{e\in E_G} |e' \cap E_{f(e)}|\}$, $\forall e' \in E_H$. The *average congestion* of the mapping is similarly defined.

- The *max-load* is the maximum number of nodes in G that are mapped to a node in H. Max-load = 1 if the mapping is one-to-one.

It should be noted that unit dilation implies unit congestion. Thus the class of dilation-1 embeddable graphs in a hypercube is a proper set of the class of congestion-1 embeddable graphs. If each node of the guest graph is mapping to a distinct node of the host, the slow-down due to nearest neighbor communication in the original graph being extended to communication along paths is a function of the length of the path (i.e., edge dilation) and the congestion of the edges on the path.

E. What Follows

We had considered hypercubes, embeddings in a hypercube, fault-tolerant embeddings in a hypercube, and some terminologies related to embedding. The remainder of this dissertation is organized as follows:

- In Chapter II, we discuss previous research of fault-tolerant ring embeddings in a hypercube. We briefly mention some approaches of existing schemes and their results and then discuss some limitations of these schemes. The chapter is concluded with the introductions of the ideas of our proposed schemes which will be discussed in more detailed in Chapter IV, V, and VI.

- In Chapter III, first, we summarize the failure model that we deal with in our proposed schemes. Second, an existing simple distributed reconfiguration algorithm is described and supplied with an example. We conclude this chapter with a formal definition of a reconfiguration step which we use for evaluating the performance of a reconfigurable embedding scheme.

- In Chapter IV, we present the algorithm of our first proposed scheme, the *DC* scheme, which efficiently embeds even length rings in a hypercube with small expansion and recovery cost. Some characteristics and performance comparison, in terms of issues of expansion and recovery cost, of this scheme with others is addressed in this chapter.

- In Chapter V, we present another scheme, the *Modified_DC* scheme, which is an enhanced version of the *DC* scheme. The algorithm and the characteristics of the *Modified_DC* scheme, and the performance comparison with the *DC* schemes are included in this chapter.

- In Chapter VI, we explore the possibility of finding a general scheme to construct 1-step recoverable embeddings. We propose a composition method to systematically construct such embeddings for rings of length $\leq (3/4)2^d$ in a d-cube. Formal proof is given to show the correctness of our approach. The two special embedding cases of a length-10 ring in a 4-cube and a length-22 ring in a 5-cube are shown to be no 1-step recoverable embeddings existent.

- In Chapter VII, we describe a sufficient condition for the non-existence of 1-step recoverable embeddings of rings $> (3/4)2^d$ in a d-cube.
- In Chapter VIII, we apply our reconfigurable ring embedding schemes to an application, the *protein sequence comparison*, in a parallel computer. We show that with our reconfigurable embedding schemes, the total running time for the fault-tolerant system of parallel protein sequence comparison will take less time than those with other fault-tolerant approaches.
- Finally, in Chapter IX, we discuss some related issues and open problems, and state some further research relating to the dynamic fault-tolerant embeddings in a hypercube.

II. FAULT-TOLERANT RING EMBEDDINGS IN A HYPERCUBE

The ring is widely used for variety of applications [Lium78, Same85, IpSS86, KuRR88, GoVa89, BHGS90, GaPS90, SeYv91, Leig92]. For instance, Gaussian elimination for dense systems on the ring is discussed in [Same85], an implementation of parallel depth-first search on the ring is presented in [KuRR88], and a data redistribution algorithm which aims at dynamically balancing the workload of image processing algorithms on the ring is introduced in [SeYv91]. The ring is essentially a linear array with a wrap-around connection between the first and last processor. Thus, algorithms which are designed for the linear array are also suitable for the ring.

The linear array and the even length ring can be easily mapped in a hypercube with the binary reflected Gray code [SaSc88]. However, when take into account fault tolerance, the mapping problems of rings in a hypercube become non-trivial, and some of the related problems are even shown to be NP-complete [ChLe91b]. For our research, we have chosen the ring as the guest graph of interest, and consider fault-tolerant ring embeddings in a hypercube. Note that, the wrap-around mesh contains the ring as a subgraph, so the schemes of constructing fault-tolerant embeddings in a hypercube can also be applicable to the mesh. In this chapter, we will discuss several existing fault-tolerant embedding schemes for rings in a hypercube, and state the approaches of our proposed schemes.

A. Static Fault Tolerance

Several static fault-tolerant ring embedding schemes in a hypercube have been introduced in [ChLe91a, ChLe91b, LaZB92]. Basically, the research in this area emphasizes exploring efficient strategies to embed rings in a hypercube with faulty nodes or links, and characterizing the scenarios of faults such that the successful construction of the embedding is possible.

Chan and Lee [ChLe91a] proposed a simple distributed fault-tolerant algorithm, RING_1PFT, to embed a ring in a hypercube with faulty nodes. This distributed algorithm implies the assumption of only local knowledge of faults rather than global knowledge with each processor having knowledge of the status of only its immediate neighbors. It is shown that RING_1PFT will successfully construct a ring of length 2^d in a d-cube with no faults occurring, and a ring of length $2^d - 2$ in a d-cube with one fault occurring. Moreover, the RING_1PFT is able to tolerate more than one fault in some circumstances.

Theorem 2.1: [ChLe91a] Let an embedding of a ring of length 2^d in a d-cube be denoted as a sequence of nodes, $(v_1, v_2, \cdots, v_{2^d})$. RING_1PFT will successfully construct a ring of length at least $2^d - 2f$ in a d-cube with f faults if and only if, for any pair of faults v_i and v_j, $i < j$, of the embedding, either $j - i \geq 3$, or $j = i + 1$ and i is even.

Latifi, Zheng, and Bagherzadeh [LaZB92] considered the problem of dilation-1 embedding of rings of length 2^d in the d-cube with faulty links. If consider maximizing the processor utilization, the above problem reduces to finding a fault-free Hamiltonian circuit consisting of all nodes in a faulty hypercube. It is known that a d-cube contains

$\lfloor d/2 \rfloor$ link-disjoint Hamiltonian circuits [AlBS87]. This implies that at least one Hamiltonian circuit can be specified with any set of no more than ($\lfloor d/2 \rfloor$ - 1) faulty links being present in the network. These three authors presented a centralized algorithm, FIND_PERM2, that can always find a fault-free Hamiltonian circuit in a d-cube in $O(d^2)$ time if there are no more than (d-2) faulty links present. FIND_PERM2 outperforms the existing fault-tolerant ring embeddings based on link-disjoint Hamiltonian circuits with twice as many faulty links present.

Theorem 2.2: [LaZB92] FIND_PERM2 identifies a characterization of a Hamiltonian cycle in a d-cube with as many as $(d-2)$ faulty links.

The problem of finding Hamiltonian circuits in hypercubes with faulty links was considered earlier by Chan and Lee in [ChLe91b]. In partial answer the question of the existence of a Hamiltonian circuit in a hypercube with faulty links, Chan and Lee showed the following. First, they proved the following theorem and showed the result of this theorem is optimal.

Theorem 2.3: [ChLe91b] In a d-cube with $\leq 2d-5$ link faults, where each node is incident to at least two non-faulty links and $d \leq 3$, for any node S, there exists a fault-free Hamiltonian circuit from S to itself.

Second, they showed that the problem of determining whether a d-cube with an arbitrary number of link faults has a Hamiltonian circuit is NP-complete.

Theorem 2.4: [ChLe91b] Given a d-cube with faulty links, the problem of finding if this faulty d-cube contains a Hamiltonian circuit comprised of only nonfaulty links is NP-complete.

B. Dynamic Fault Tolerance

Several reconfigurable ring embedding schemes used to recover from faults as well as maintain the quality of the embedding have been introduced [PrMe88, ChLT88a, ChLT88b, Leet90, LiMc92a, LiMc92b, LiSM92]. The basic idea of these schemes is to leave certain nodes in a hypercube as spare nodes (i.e., nodes not the image nodes of the embedding) for later recovery usage. This leads to the inevitable question: Given a ring, what is the smallest hypercube that should be chosen to accommodate the ring, and how many nodes in a hypercube should be saved as spare nodes in order to achieve efficient recovery? The fault recovery efficiency of an embedding should take both recovery cost (i.e., how many steps needed) and the performance degradation (i.e., dilation and congestion) into account. Since this efficiency depends heavily on the initial embedding, the research of dynamic fault-tolerant embedding focuses on deriving intelligent initial embeddings which facilitate the processing of reconfiguration with minimal recovery cost and performance degradation.

1. Previous Approaches. Chen, Liang, and Tsai [ChLT88a] proposed an initial mapping scheme, Mapping I, which can embed rings of length 2 to $2^d - 2$ in a d-cube and tolerate one single faulty node. Their scheme attempts to maximize the length of rings that can be embedded in a hypercube. Hence, the cost of recovering one single fault of their scheme is sometimes very large (i.e., $O(d^2)$), since a long reconfiguration may be required to eventually absorb the fault.

Theorem 2.6: [ChLT88a] The embeddings constructed by Mapping I can tolerate any single faulty node.

Chen, Liang, and Tsai [ChLT88b] then proposed other initial mapping schemes, Mapping II and Mapping III, for certain length of rings such that the embeddings can be recovered from one single faulty node within one step. The rings which are applicable to Mapping II and Mapping III are lengths of 2^{d-1} and $(3/4)2^d$, respectively.

Theorem 2.7: [ChLT88b] The embeddings constructed by Mapping II and Mapping III can tolerate any single faulty node and recover the fault within one step.

In the same year, Provost and Melhem [PrMe88] also proposed a simple scheme to embed a ring of length $(3/4)2^d$ in a d-cube such that any simple fault can be tolerated.

Theorem 2.8: The proposed ring embedding scheme in [PrMe88] can tolerate any single faulty node and recover the fault within one step.

Later work by Lee [Leet90] constructed a ring embedding in a hypercube to tolerate any single fault within only one reconfiguration step. Basically, Lee's idea is to embed rings in a hypercube such that spare nodes are always close to failed nodes whenever reconfiguration occurred. Lee's scheme increases the size of the embedding hypercube necessary to accommodate the embedded rings. For instance, using his scheme a ring of length 16 will be embedded in a 6-dimensional hypercube to achieve efficient recovery ability. Although the recovery cost of this scheme is always minimum (i.e., 1), the number of wasted nodes in a hypercube is quite large. In terms of node utilization (i.e.,

expansion), Lee's scheme is not efficient.

2. Our Approaches. By comparing the schemes in [PrMe88, ChLT88a, ChlT88b, Leet90], that there are trade-offs between the issues of expansion and efficient recovery for a dynamic fault-tolerant embedding scheme. With this observation, we propose our first scheme which is a simple strategy, but reaches a good compromise between the trade-off of expansion and efficient recovery issues. Our scheme is based on the idea of divide-and-conquer and thus, we call it the *DC* scheme. By treating a ring as a path with the same starting and ending point, the *DC* scheme recursively divides the embedded path and the embedding subcube into half until all the divided subcube are 3-cube. For the *DC* scheme, a 3-cube will be treated as a basic embedding unit and used for embedding a partial portion of the ring. With the divide-and-conquer approach, the *DC* scheme guarantees that spare nodes will be almost evenly scattered over the 3-cubes of the embedding hypercube. This important feature gives the DC scheme with efficient recovery ability since a faulty node will have more chances to be replaced by a proper spare node that resides in the same subcube as the faulty node. That is, without often propagating the fault from one subcube to another for recovery, the *DC* scheme tends to use less steps to recover a single fault. It is shown that the average expansion for the *DC* scheme is 1.58, and the average number of reconfiguration steps needed is 1.3. Moreover, within 3 reconfiguration steps, any single fault in the embedding can be recovered (i.e., 3-step recoverable). The algorithm of the *DC* scheme and the performance comparison with other schemes will be discussed in details in Chapter IV.

The second scheme that we proposed is an enhanced version of the *DC* scheme. We designate the second scheme as the *Modified_DC* scheme. The main difference between the *Modified_DC* scheme and the *DC* scheme lies in the addition of the embedding

sequence permutations in the *Modified_DC*. The embedding sequence of the *Modified_DC* scheme is generated by permuting some partial portions of the embedding sequence generated by the *DC* scheme. We show that with such permutation the *Modified_DC* achieve better recovery ability: the average number of reconfiguration steps needed is 1.1 steps, and any single fault can be recovered within 2 steps if the dimension of the hypercube ≥ 5. The algorithm of the *Modified_DC* scheme and the performance comparison with the *DC* schemes are discussed in Chapter V.

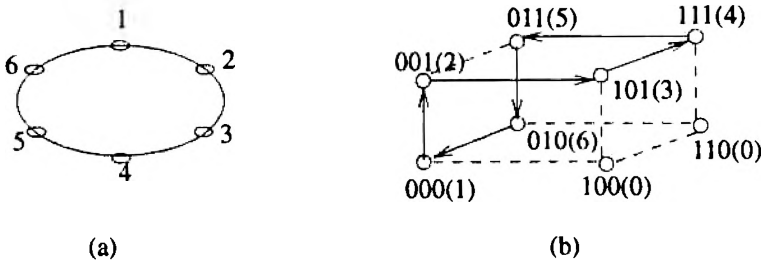
Since we have derived the *DC* scheme and the *Modified_DC* scheme which are 3-step recoverable and 2-step recoverable, respectively, it is natural that we explore the problem of finding dynamic fault-tolerant embedding schemes which are 1-step recoverable. For simplicity, throughout this dissertation, we use 1SRE to denote the phrase "1-step recoverable embedding". First, we define what a 1-step recoverable embedding is, and then, propose an efficient scheme to systematically construct 1SRE's of length-k rings in d-cubes where k is even, $6 \leq k \leq (3/4)2^d$, and $d \geq 3$. Our scheme is based on the following conjecture: If there exists a 1SRE of a length-k_1 ring in a d-cube and a 1SRE of a length-k_2 ring in a d-cube, then there exists a 1SRE of a length-$(k_1 + k_2)$ ring in a $(d+1)$-cube. We then show that the conjecture above is provable. In fact, a 1SRE of a length-$(k_1 + k_2)$ ring in a $(d+1)$-cube can be obtained by directly combining a 1SRE of a length-k_1 ring in a d-cube and a 1SRE of a length-k_2 ring in a d-cube with some modification. A formal proof is given to show the correctness of our approach. The algorithm and the proofs are given in Chapter VI.

III. Failure Model and Reconfiguration Algorithm

It is easy to see that R_k is embeddable to Q_d if and only if there exists a subgraph in Q_d which is isomorphic to R_k. So, the idea of reconfiguration is to maintain a fault-free subgraph in Q_d that is isomorphic to R_k. The failure model used in this dissertation follows the one in [YaHa86]. In this model, each node in a hypercube has a unique state, and the system is in an operational state if and only if all the distinct states exist. A fault will cause the missing of a state, and the system should be able to perform a distributed reconfiguration via the local operations of faulty free processors until the missing state is recovered. The reconfiguration algorithm that we used is first described in [ChLT88a]. For the completeness of this dissertation, we will summarize the fault model in [YaHa86] and the reconfiguration in [ChLT88a] in this chapter.

A. Failure Model

Every node of Q_d is given a state: *active state*, *inactive or spare state*, or *faulty state*. Active states are denoted by the labels $1, 2, \cdots, k$, where k is the length of the ring embedded. Inactive or spare states are denoted by 0, and faulty states are denoted by -1. The state of Q_d is the 2^d-tuple $S(x_0, x_1, \cdots, x_{2^d-1}) = (S(x_0), S(x_1), \cdots, S(x_{2^d-1}))$ where $S(x_i)$ is the state of node x_i in Q_d. Figure 3 shows an example of the above state assignments of embedding an R_6 in a Q_3. Suppose the initial state of Q_3 is $S(0, 1, 2, 3, 4, 5, 6, 7) = (1, 2, 6, 5, 0, 3, 0, 4)$. The guest graph R_6 and the embedding in the host graph Q_3 with every node labeled with its state is shown in Figure 3(a) and 3(b), respectively. As indicated by the arrowed lines, Q_3 contains a subgraph isomorphic to R_6.



(a) (b)

Figure 3. (a) The guest graph R_6. (b) The embedding in the host graph Q_3 with every node labeled with a state.

A fault $F(S(x_j))$ happens if Q_d has no active node in state $S(x_j)$ corresponding to a node of R_k. For example, if the node 011 of the Q_3 in Figure 3(b) becomes faulty, the state of the Q_3 will become to $S(C_3)$ = (1, 2, 6, -1, 0, 3, 0, 4) and state 5 is replaced by state -1. Thus, fault detection will be based on the identification of a missing state of the k possible active states. It is assumed that reliable fault diagnosis mechanisms are available if a node becomes faulty [MeMa78, ArGr81, Bhat83]. To ensure correctness, each active node periodically tests itself or is tested by a watchdog processor. If it is faulty, its state changes to -1. We also assume that the links of a hypercube are faulty-free and only one single faulty node can be present in the system, and the recovery actions are assumed faulty-free.

B. Reconfiguration Algorithm

Let node a_t be an active node with state t in an embedding. For every active node a_t, $t = 1, 2, \cdots, k$ in an embedding of length-k rings in a hypercube, node a_t saves and updates the work environment of node $a_{(t+1) \bmod k}$ periodically, and is responsible for detecting and recovering the fault $F((t+1) \bmod k)$. When active node a_{t+1} becomes

faulty (i.e., fault $F(t+1)$), active node a_t will detect this fault and act as a local supervisor to invoke following reconfiguration actions:

1) Compute b = XOR(a_t, a_{t+1}, a_{t+2}) (i.e., nodes b, a_t, a_{t+1}, and a_{t+2} constitute the four nodes of a 2-D plane of hypercubes).

2) Change the state of node b to $t+1$, and assign the work environment of node a_{t+1} to node b (i.e., we will denote this step as node a_{t+1} » node b).

3) Let the previous state of node b to be $S(b)_{old}$. If $S(b)_{old} = 0$ (i.e., spare state) then the recovery is done; otherwise, a propagated fault $F(S(b)_{old})$ is issued and the reconfiguration continues.

The XOR operation above denotes the bitwise exclusive on binary numbers. We shall call the above algorithm the *xor-reconfiguration* algorithm. Now, let's consider an example for the *xor-reconfiguration* algorithm. Figure 4(a) shows the original fault-free embedding of an R_{10} in a Q_4. A fault $F(5)$ occurring in node 0110 is shown in Figure 4(b). Since node 0111 with state 4 is responsible for detecting and recovering the fault $F(5)$, it will eventually detect the fault and begin to act as a local supervisor. Then, with $XOR(0111, 0110, 1110) = 1111$ and $S(1111) \neq 0$, the action of node 0110 » node 1111 activated and a propagated fault $F(7)$ is issued. The consequence of above actions is shown in Figure 4(c). Similarly, node 1110 with state 6 detects the propagated fault $F(7)$ and computes that $XOR(1110, 1111, 1101) = 1100$. Since $S(1100) = 0$, the action of node 1111 » node 1100 activated and finally the reconfiguration is done. The embedding after recovery is shown in Figure 4(d).

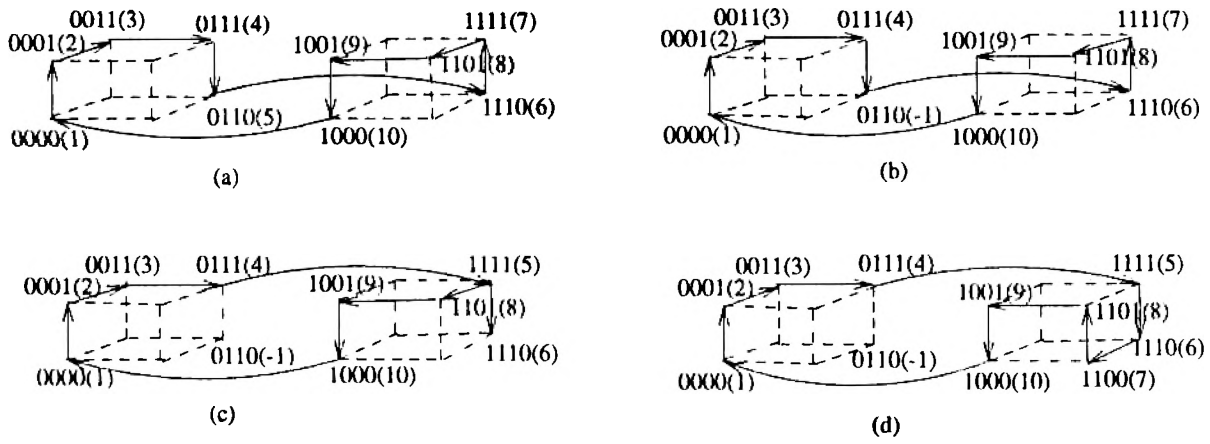


Figure 4. (a) The embedding (R_{10} to Q_4) with no faulty nodes. (b) The fault $F(5)$ occurs. (c) Active node 1111 recovers $F(5)$ and a propagated fault $F(7)$ is issued. (d) Spare node 1100 recovers $F(7)$ and the reconfiguration is done.

The cost of recovering one single fault is measured by the number of states changed of fault-free nodes (i.e., spare to active or active to spare) during the reconfiguration actions. For example, to recover the fault $F(5)$ in the embedding in Figure 4(a), the system experiences following replacing sequence: node 0110 » node 1111 » node 1100. There are two states changed of fault-free nodes (i.e., node 1111 changed from state 7 to state 5 and node 1100 changed from state 0 to state 7), so we will say that it costs 2 steps to recover the fault in this case. Now, we can have the following definition.

Definition: A system with an embedding scheme and a reconfiguration algorithm is t-step recoverable if the system can recover any single fault by changing the states of at most t fault-free nodes.

Since, in this paper, when we mention a reconfiguration algorithm we mean the *xor-reconfiguration* algorithm. For convenience, in the following we shall say an embedding is t-step recoverable instead of saying the system with an embedding scheme and a reconfiguration algorithm is t-step recoverable.

IV. THE DC SCHEME: 3-STEP RECOVERABLE EMBEDDINGS

In this chapter, we present the algorithm of the *DC* scheme which is based on the idea of divide-and-conquer to efficiently embed even length rings in a hypercube to achieve small expansion and recovery cost. The characteristics of the scheme and the performance comparison with others will also be addressed in this chapter.

A. The Algorithm

The algorithm of the *DC* scheme of embedding an even length-k ring in a d-cube is following:

Procedure *DC_SCHEME* (k, d) /* $d \geq 4, (7/8)2^{d-1} < k \leq (7/8)2^d$ */
begin
$p := k/2 - 1$
$R_k \rightarrow Q_d \equiv 0 \,|\, Concate\,(\,PATH\,(p, d-1), < d >\,, PATH\,(p, d-1), < d >\,)$
end

Procedure *PATH* (p', d') /* $p'+1$: path length, d': size of a subcube */
Const $d_1 = 1, d_2 = 2, d_3 = 3, d_4 = 1, d_5 = 2, d_6 = 1$
begin
if $d' = 3$ **then return** $(< d_1, d_2, \cdots, d_{p'} >)$
else begin
$A := PATH\,(\lfloor \frac{p'-1}{2} \rfloor, d'\text{-}1)$
$B := PATH\,(\lceil \frac{p'-1}{2} \rceil, d'\text{-}1)$
$C := Concate\,(A, < d' >, B)$

return (C)
end
end

According to the algorithm above, at most six links (i.e., $d_1, \cdots, d_6$) in a 3-cube of the embedding d-cube will be included in the transition sequence of an embedding. Therefore, at most seven nodes in a 3-cube will be in active state and at least one node will be in spare state. So, with the *DC* scheme, the largest and smallest rings that can be embedded in a d-cube is $(7/8)2^d$ (i.e., leave one spare node in each 3-cube) and $(7/8)2^{d-1} + 2$, respectively. The function *Concate* in the algorithm concatenates sets of sequences together to form one sequence (i.e., *Concate* (< 1, 2 >, < 1, 2, 3 >) = < 1, 2, 1, 2, 3 >.

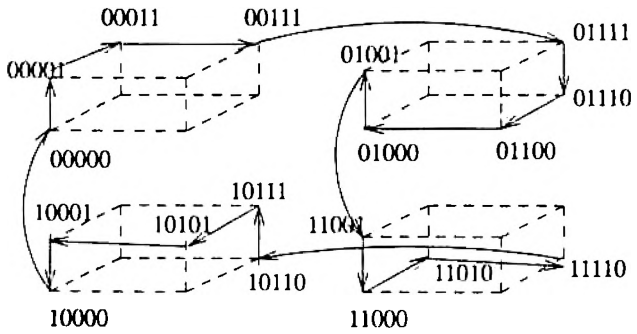


Figure 5. An embedding of a length-18 ring in a 5-cube.

Example: Consider embedding a length-18 ring in a 5-cube with the *DC* scheme. Tracing the algorithm we first have $R_{18} \rightarrow Q_5 = 0 \mid$ (PATH(8, 4), < 5 >, PATH(8, 4), < 5 >). Since the divided subcube is not a 3-cube, PATH(8, 4) is split to two parts, PATH(3, 3) and PATH(4, 3). Then, we got PATH(3, 3) = < 1, 2, 3 > and PATH(4, 3) = < 1, 2, 3, 1 >. So, at last PATH(8, 4) = < 1, 2, 3, 4, 1, 2, 3, 1 >, and we have the final embedding $R_{18} \rightarrow$

$Q_5 \equiv 0 \mid <1, 2, 3, 4, 1, 2, 3, 1, 5, 1, 2, 3, 4, 1, 2, 3, 1, 5>$. This final embedding is depicted in Figure 5 as in arrowed lines).

The transition sequences of embedding rings from lengths 8 to 20 in a hypercube with the *DC* scheme are given in Table I.

Table I. Total transition sequences of embedding various rings with the DC scheme.

Ring	Total transition sequence of the embedding
8	1 2 3 4 1 2 3 4
10	1 2 3 1 4 1 2 3 1 4
12	1 2 3 1 2 4 1 2 3 1 2 4
14	1 2 3 1 2 1 4 1 2 3 1 2 1 4
16	1 2 3 4 1 2 3 5 1 2 3 4 1 2 3 5
18	1 2 3 4 1 2 3 1 5 1 2 3 4 1 2 3 1 5
20	1 2 3 1 4 1 2 3 1 5 1 2 3 1 4 1 2 3 1 5

B. Characteristics of the Resulting Embeddings

Definition: With the DC scheme, each 3-cube of the embedding d-cube will contribute one of the following sequences to the transition sequence of the embedding: $<1, 2>$, $<1, 2, 3>$, $<1, 2, 3, 1>$, $<1, 2, 3, 1, 2>$, and $<1, 2, 3, 1, 2, 1>$, and we will denote these 3-cubes as Q_3^2, Q_3^3, Q_3^4, Q_3^5, and Q_3^6, respectively.

Theorem 4.1: The *DC* scheme can tolerate any single fault.

***Proof*: We consider the fault recovery in a Q_3^4 here. The similar proof can be applied to the cases for Q_3^i, i = 2, 3, 5, or 6.**

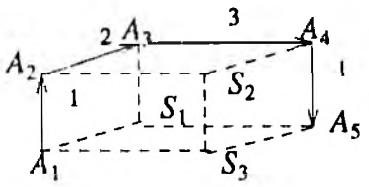


Figure 6. A 3-cube with the transition sequence < 1, 2, 3, 1 >.

A Q_3^4 with node labels is given Figure 6. Nodes with labels beginning with letter 'A' are in active state, and nodes with labels beginning with letter 'S' are in spare state. By applying the xor-reconfiguration algorithm, the following facts can be easily verified:

1) The action A_2 » S_1 will recover the faulty node A_2.

2) The action A_3 » S_2 will recover the faulty node A_3.

3) The action A_4 » S_1 will recover the faulty node A_4.

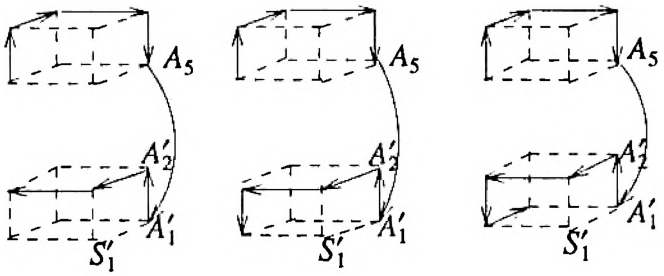
Then, how about fault recovery for nodes A_1 and A_5? For these two nodes, we need to consider the connection of this Q_3^4 with neighboring 3-cubes. Due to the divide-and-conquer approach of the *DC* scheme, a Q_3^4 will connect only to a Q_3^3, Q_3^4, or Q_3^5.

For A_5, we consider the following cases:

1) The Q_3^4 is connected to a Q_3^3 as in Figure 7(a). Then, the action A_5 » A_2' » S_1' will recover the faulty node A_5.

2) The Q_3^4 is connected to another Q_3^4 as in Figure 7(b). Then, the action A_5 » A_2' » S_1' will recover the faulty node A_5.

3) The Q_3^4 is connected to a Q_3^5 as in Figure 7(c). Then, the action A_5 » A_2' » S_1' will recover the faulty node A_5.



(a) (b) (c)

Figure 7. (a) The Q_3^4 connects to a Q_3^3. (b) The Q_3^4 connects to another Q_3^4. (c) The Q_3^4 connects to a Q_3^5.

For A_1, we consider the following cases:

1) A Q_3^3 is connected to the Q_3^4 as in Figure 8(a). Then, the action A_1 » S_1'' will recover the fault in A_1.

2) Another Q_3^4 is connected to the Q_3^4 as in Figure 8(b). Then, the action A_1 » A_4'' » S_1'' will recover the fault in A_1.

3) A Q_3^5 is connected to the Q_3^5 as in Figure 8(c). Then, the action A_1 » S_1'' will recover the fault in A_1.

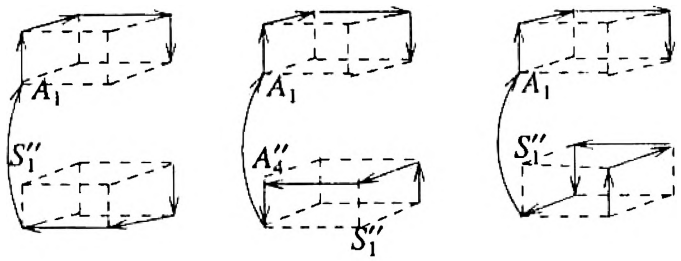


Figure 8. (a) A Q_3^3 connects to the Q_3^4. (b) Another Q_3^4 connects to the Q_3^4. (c) A Q_3^5 connects to the Q_3^4.

Thus, for each possible faulty node A_1, $\cdots$, A_5 in the Q_3^4, the system with the xor-reconfiguration algorithm is able to recover it. □

Lemma 4.1: Applying the DC scheme to construct embeddings. If *all* the 3-cubes in the embedding d-cube result in one of the following 3-cube: Q_3^2's, Q_3^3's, or Q_3^5's, then the embedding can recover any single fault in 1 step.

Proof: This lemma can be verified by a similar proof technique to that of Theorem 4.1. □

Lemma 4.2: To embed a length-2^{d-1} or $(3/4)2^d$ rings in a d-cube ($d \geq 4$) with the *DC* scheme, the 3-cubes in the embedding d-cube will all be Q_3^5 and Q_3^3, respectively.

Proof: To embed an even length-k ring in a d-cube, the DC scheme divides the ring into two $k/2$ paths (containing $k/2$ nodes), and then embeds these two paths in two $(d-1)$-cubes. The ring embedding is done by connecting these two embedded paths

together with two links of dimension d.

Here, we show an induction proof to the case of $(3/4)2^d$.

Induction base: For d = 4 (i.e, embed an R_{12} in a Q_4), the *DC* scheme will come out the embedding, $R_{12} \rightarrow Q_4 \equiv 0 \mid < 1, 2, 3, 1, 2, 4, 1, 2, 3, 1, 2, 4 >$, such that both 3-cubes of the embedding 4-cube are Q_3^5.

Induction hypothesis: Assume for $d = m$ (i.e., embed an $R_{(3/4)2^m}$ in a m-cube) the lemma holds. This implies that the *DC* scheme will embed a path of length $(3/4)2^{m-1}$ in a (m-1)-cube such that the 3-cubes of the embedding (m-1)-cube are all Q_3^5.

Induction step: Consider the case for $d = m + 1$ (i.e., embedding an $R_{(3/4)2^{m+1}}$ in a (m+1)-cube). Following the DC scheme, the original embedding problem reduces to embed two paths of length $(3/4)2^m$ in two m-cubes. With the recursive nature of the *DC* scheme, the original embedding problem further reduces to embedding four paths of length $(3/4)2^{m-1}$ into four $(m-1)$-cubes. By the induction hypothesis, the *DC* scheme will embed each of these four paths in a $(m-1)$-cube and the 3-cubes of the embedding $(m-1)$-cube are all Q_3^5. By connecting these four embedded paths together with two links of dimension m and two links of dimension of $m+1$, the *DC* scheme forms an embedding, $R_{(2^{m+1}-2^{m-1})} \rightarrow Q_{(m+1)}$, such that all 3-cubes in the embedding $(m+1)$-cube are Q_3^5's.

Figure 9 provides an example to clear the idea in the induction step above. The two dotted lines among 3-cubes in Figure 9 represent dividing the ring embedding into several path embeddings.

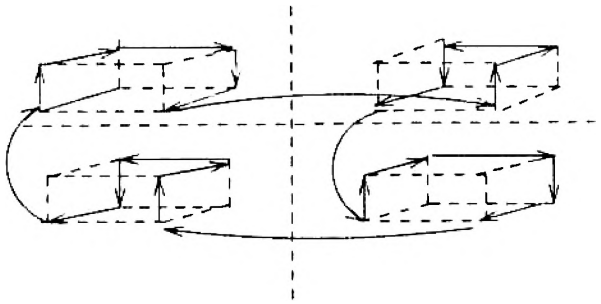


Figure 9. An example of dividing a ring embedding to several path embeddings.

A similar proof can be used to show case 2^{d-1}. □

Theorem 4.2: With the *DC* scheme, the embeddings of length-$(3/4)2^d$ or length-2^{d-1} rings in a d-cube ($d \geq 4$) can recover any single fault in 1 step.

Proof: Directly from Lemma 4.1 and Lemma 4.2. □

Theorem 4.3: With the *DC* scheme, the embeddings of any even length rings in a d-cube can recover any single fault within 3 steps.

Proof: Intuitively the more spare nodes exist in a 3-cube, the greater chance system can recover any single fault in less steps. Therefore, we will consider the fault recovery of the worst situation such that every 3-cube of the embedding hypercube is Q_3^6 (one spare node in every 3-cube). Figure 10 depicts the partial portion of the connection among

Q_3^6's.

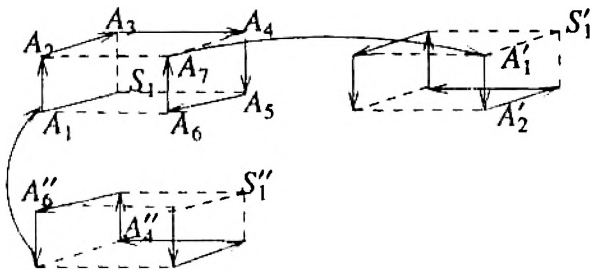


Figure 10. Partial portion of the connection among Q_3^6's.

By applying the xor-reconfiguration algorithm, we obtain the following results:

1) The action A_1 » A_6'' » A_4'' » S_1'' will recover the faulty node A_1.

2) The action A_2 » S_1 will recover the faulty node A_2.

3) The action A_3 » A_7 » A_2' » S_1' will recover the faulty node A_3.

4) The action A_4 » S_1 will recover the faulty node A_4.

5) The action A_5 » A_7 » A_2' » S_1' will recover the faulty node A_5.

6) The action A_6 » A_4 » S_1 will recover the fault in A_6.

We have shown that in the worst case any single fault in the embedding constructed by *DC* scheme can be recovered within 3 steps. Similar proof can be applied to other cases (Q_3^i, $i = 2, \cdots, 5$). Since for other cases there will be more spare nodes in 3-cubes, we claim that overall the system needs at most 3 steps to recover any single fault. □

Lemma 4.3: Embedding a length-k ring in a d-cube with the *DC* scheme. If there exist p Q_3^4's in the embedding d-cube, then there will be $2p$ active nodes which need 2 steps for recovery when they become faulty.

Proof: Consider the three connection cases in Figure 7. In these three cases, A_5 and A_1' are the two nodes which need 2 steps for recovery. The case in Figure 7(b) needs further consider about what the second Q_3^4 connects to. Again, by following the proof technique in Theorem 4.1, it can be seen that there are two more nodes in the second Q_3^4 which need 2 steps for recovery. So, if there are p Q_3^4's in an embedding d-cube, by considering the 3-cube to which each Q_3^4 is connected, there will be $2p$ nodes that need 2 steps for recovery when they become faulty. □

Lemma 4.4: Embedding a length-k ring in a d-cube with the *DC* scheme. If there exist q Q_3^6's in the embedding d-cube, then there will be $3q$ active nodes which need 3 steps for recovery and $2q$ nodes which need 2 steps for recovery when they become faulty.

Proof: This lemma can be verified by a similar proof technique to that of Lemma 4.3. □

Theorem 4: Embedding a length-k ring in a d-cube with the *DC* scheme. If 3-cubes of the embedding d-cube consists of p Q_3^4's and q Q_3^6's, then we can conclude that:

. There are $3q$ active nodes needing 3 steps for recovery when they become faulty.

. There are $2(p+q)$ active nodes needing 2 steps for recovery when they become faulty.

. There are $r-5q-2p$ active nodes needing 1 step for recovery when they become faulty.

Proof: Directly from Lemma 4.1, Theorem 4.3, Lemma 4.3, and Lemma 4.4. □

C. Performance Comparison

For convenience, we shall designate the embedding scheme in [ChLT88a] as Tasi's_1, the embedding scheme in [ChLT88b] as Tasi's_2, and the embedding scheme, L1, in [Leet90] as Lee's. In this section we compare the performance among Tsai's_1, Tsai's_2, Lee's, and the *DC* scheme. Several parameters are used to characterize a reconfigurable embedding scheme:

1) The number of faults tolerated.

2) The dilation and the average dilation of the original embedding and the recovered embedding.

3) The expansion and the average expansion of the original embedding and the recovered embedding.

4) The average recovery steps and the maximum recovery steps needed for recovering one single fault.

Since all four schemes assume one fault occurring, they all tolerate one fault only. All four schemes achieve dilation-1 original embedding and dilation-1 recovered embedding. So the parameters left over to distinguish different schemes are: expansion and recovery step issues. In this dissertation, we use the following measures: expansion, average_expansion, average_recovery_steps, and maximum_recovery_steps to compare the performance of different embedding schemes. Average_expansion is the average of expansions of embedding some certain ranges rings in those hypercubes of the least required size by each embedding scheme. Average_recovery_steps is the average steps

needed to recover one single fault, when consider embedding some certain lengths of rings in the least required hypercubes (the sizes of the hypercube needed will be depended on the embedding scheme chosen). Maximum_recovery_steps is the maximal numbers of steps required to recover one single fault, when consider embedding some certain lengths of rings in the least required hypercubes.

Since Tsai's_2 is an embedding scheme which deals only with rings of lengths $(3/4)2^d$ and 2^{d-1}. We will divide the performance comparison into two groups. In the first group, we compare the performance of Tsai's_2, Lee's, and the DC scheme of embedding rings of length $(3/4)2^d$ and 2^{d-1} in hypercubes of the least required sizes by each scheme. In the second group, we compare the performance of Tsai's_1, Lee's, and the DC scheme of embedding rings of length $\neq (3/4)2^d$ and 2^{d-1} for $d \geq 4$ in hypercubes of the least required sizes by each scheme.

Table II demonstrates the performance measures in the first group. For rings of length $(3/4)2^d$ and 2^{d-1}, all three schemes is 1-step recoverable, and both Tsai's_2 and the *DC* scheme have smaller numbers in average_expansion and expansion than those of Lee's. Table III and Table IV demonstrate the performance comparison in the second group with rings ≤ 100 and ≤ 1000, respectively. It can be seen that Tsai's_1 has the smallest values both in average_expansion and expansion among those of three schemes, however, the average_recovery_steps and maximum_recovery_steps of Tsai's_1 are much larger than those of two other schemes. Moreover, for length constraints from 100 to 1000, the average_recovery_steps and the maximum_recovery_steps of Tsai's_1 almost increase 10 times. So, generally, Tsai's_1 is not an efficient recovery scheme. Lee's has 1-step recoverable ability. However, from Table III and Table IV, we can easily

see that the average_expansion and expansion of Lee's are the largest among those of three schemes. In terms of processor utilization, Lee's embedding scheme is inefficient. As we mentioned, there is trade-off between the issues of expansion and recovery for a fault-tolerant embedding scheme. From Table II, 3, and 4, we can conclude that the *DC* scheme reaches a very good compromise between the trade-off of expansion and efficient recovery.

Table II. Comparison among different schemes for embedding rings of length k in hypercubes of the least required sizes by each scheme, where $k \leq 1000$, $k = 2^{d-1}$ or $(3/4)2^d$, and $d \geq 4$.

	Average Expansion	Average Recovery Steps	Expansion	Maximum Recovery Steps
Tsai's_2	1.67	1	2.0	1
Lee's	2.76	1	4.0	1
DC	1.67	1	2.0	1

Table III. Comparison among different schemes for embedding rings of length k in hypercubes of the least required sizes by each scheme, where $k \leq 100$, $k \neq 2^{d-1}$ or $(3/4)2^d$, and $d \geq 4$.

	Average Expansion	Average Recovery Steps	Expansion	Maximum Recovery Steps
Tsai's_1	1.47	10.8	1.94	49
Lee's	2.77	1	5.12	1
DC	1.58	1.3	2.2	3

Table IV. Comparison among different schemes for embedding rings of length k in hypercubes of the least required sizes by each scheme, where $k \leq 1000$, $k \neq 2^{d-1}$ or $(3/4)2^d$, and $d \geq 4$.

	Average Expansion	Average Recovery Steps	Expansion	Maximum Recovery Steps
Tsai's_1	1.39	96	1.99	499
Lee's	2.93	1	5.3	1
DC	1.64	1.27	2.28	3

Note that with a simple change to the DC scheme which permutes the embedding sequences generated by the DC scheme, the system with this updated version of the DC scheme and the xor-reconfiguration algorithm can recover any single fault in 2 steps. The proof, however, is involved and the interested readers are referred to [LiMc92b].

D. Concluding Remarks

In this chapter we present a general scheme based on the idea of divide-and-conquer that efficiently embeds even length rings on hypercubes with small expansion and recovery cost. It is shown that the average expansion for the proposed scheme is 1.58, and the average number of recovery steps is 1.3. Moreover, within 3 steps the system applying the proposed embedding scheme is able to recover any single fault. Compared to other embedding schemes that result in either large expansion or great recovery cost, our proposed embedding scheme performs better.

V. THE MODIFIED_DC SCHEME: 2-STEP RECOVERABLE EMBEDDINGS

In this chapter, we present an enhanced version of the *DC* scheme, the *Modified_DC* scheme, to efficiently embed even length rings in a hypercube to have small expansion and recovery cost. The characteristics of the scheme and the performance comparison with the *DC* scheme will also be addressed in this chapter.

A. The Algorithm

Before we discuss the algorithm of the *Modified_DC* scheme, we need one more definition.

Definition: Let $< d_1, d_2, \cdots, d_k >$ be the transition sequence of an embedding of a length-k ring in a d-cube. Any partial but connective portion of the transition sequence, $< d_i, d_{i+1}, \cdots, d_{j-1}, d_j >$ where $i > 1$ and $j < k$, will be designated as *partial transition sequence* of the embedding. Subscripts 'L' or 'R' will be appended to a partial transition sequence of the embedding (i.e., $< d_1, d_2, \cdots >_L$ or $< d_1, d_2, \cdots >_R$) to symbolize the result of the left or right portion of the division processing. The subscripts are just for denotation, and $< d_1, d_2, \cdots d_i >_L = < d_1, d_2, \cdots d_i >_R = < d_1, d_2, \cdots d_i >$.

The algorithm of the *Modified_DC* scheme of embedding an even k length ring in a d-cube is following:

Procedure *MODIFIED_DC_SCHEME* (k, d) /* $d \geq 4$ and $(7/8)2^{d-1} < k \leq (7/8)2^d$ */

begin

$p := k/2 - 1$

$S := PATH\ (p, d\text{-}1, 'L')$

if $d \geq 5$ **then begin**

if $\frac{4}{8}2^d < k \leq \frac{6}{8}2^d$ **then**

Change all $< 1, 2, 3, 1 >_R$ in S to $< 2, 1, 3, 2 >$

elseif $\frac{6}{8}2^d < k \leq \frac{7}{8}2^d$ **then**

Change all $< 1, 2, 3, 1, 2, 1 >_R$ in S to $< 2, 1, 3, 2, 1, 2 >$

end

$R_k \rightarrow Q_d \equiv 0 \mid Concate\ (S, < d >, S, < d >)$

end

Procedure *PATH* (p', d', h') /* $p' + 1$: path length, d': subcube size, and $h' = 'L'$ or $'R'$ */

Const $d_1 = 1, d_2 = 2, d_3 = 3, d_4 = 1, d_5 = 2, d_6 = 1$

begin

if $(d' = 3)$ then

if $(h' = 'L')$ **then**

return $(< d_1, d_2, \cdots, d_{p'} >_L)$

else return $(< d_1, d_2, \cdots, d_{p'} >_R)$

else begin

$left := PATH\ (\lfloor \frac{p'-1}{2} \rfloor, d'\text{-}1, 'L')$

$right := PATH\ (\lceil \frac{p'-1}{2} \rceil, d'\text{-}1, 'R')$

return $(left, < d' >, right)$

end

end

The function *Concate* in the algorithm concatenates several partial transition sequences together to form one larger partial transition sequence or a transition sequence (i.e., *Concate* (< 1, 2 >, < 1, 2, 3 >) = < 1, 2, 1, 2, 3 >). Again, the largest ring and the smallest ring that can be embedded in a d-cube with the *Modified_DC* is the same to those for the *DC* scheme.

Basically, the *Modified_DC* scheme differs from the *DC* scheme by the addition of permutation to some partial transition sequences (i.e., some partial transition sequences < 1, 2, 3, 1 > will be permuted to < 2, 1, 3, 2 > and some partial transition sequences < 1, 2, 3, 1, 2, 1 > will be permuted to < 2, 1, 3, 2, 1, 2 >). We will show later that with such simple permutation, the recovery efficiency of the resulted system can be better improved.

Example: Consider using the *Modified_DC* scheme to embed a length-18 ring in a 5-cube. First, we have S = *PATH* (8, 4, 'L'). Since the divided subcube is not a 3-cube, *PATH* (8, 4, 'L') is split to two parts, *PATH* (3, 3, 'L') and *PATH* (4, 3 'R'). Then, *PATH* (3, 3, 'L') = < 1, 2, 3 >$_L$, *PATH* (4, 3, 'R') = < 1, 2, 3, 1 >$_R$, and S = *PATH* (8, 4, 'L') = < 1, 2, 3 >$_L$, < 4 >, < 1, 2, 3, 1 >$_R$. Since $(4/8)2^5 = 16 < 18 < (6/8)2^5 = 24$, the partial transition sequence < 1, 2, 3, 1 >$_R$ in S is permuted to be < 2, 1, 3, 2 >. So, $R_{18} \rightarrow Q_5 \equiv 0$ | *Concate* (< 1, 2, 3 >, < 4 >, < 2, 1, 3, 2 >, < 5 >, < 1, 2, 3 >, < 4 >, < 2, 1, 3, 2 >, < 5 >) $\equiv$ 0 | < 1, 2, 3, 4, 2, 1, 3, 2, 5, 1, 2, 3, 4, 2, 1, 3, 2, 5 >. This embedding is shown in Figure 11 (i.e., denotes as arrowed lines).

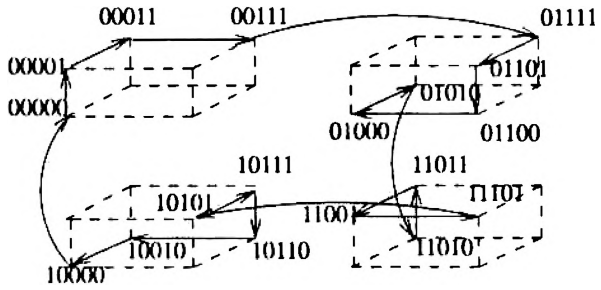


Figure 11. Embedding a ring of length 18 in a 5-cube with the *Modified_DC* scheme.

With the *Modified_DC* scheme, the transition sequences of the embeddings of length-16, 18, $\cdots$, 28 rings in a 5-cubes are shown in Table V.

Table V. Total transition sequences of embedding various rings in a 5-cube with the *Modified_DC* scheme.

Rings	Total transition sequence of an embedding
16	1 2 3 4 1 2 3 5 1 2 3 4 1 2 3 5
18	1 2 3 4 2 1 3 2 5 1 2 3 4 2 1 3 2 5
20	1 2 3 1 4 2 1 3 2 5 1 2 3 1 4 2 1 3 2 5
22	1 2 3 1 4 1 2 3 1 2 5 1 2 3 1 4 1 2 3 1 2 5
24	1 2 3 1 2 4 1 2 3 1 2 5 1 2 3 1 2 4 1 2 3 1 2 5
26	1 2 3 1 2 4 2 1 3 2 1 2 5 1 2 3 1 2 4 2 1 3 2 1 2 5
28	1 2 3 1 2 1 4 2 1 3 2 1 2 5 1 2 3 1 2 1 4 2 1 3 2 1 2 5

B. Characteristics of the Resulting Embeddings

Corollary 5.1: To embed rings in a 4-cube with the *Modified_DC* scheme, the

embeddings generated are 3-step recoverable in worst case.

Proof : According to the algorithm of the *DC* scheme and that of the *Modified_DC* scheme, both schemes will produce the same transition sequences for embedding rings in 4-cubes. These embeddings in 4-cube are,

$R_8 \rightarrow Q_4 \equiv 0 \mid < 1, 2, 3, 4, 1, 2, 3, 4 >$

$R_{10} \rightarrow Q_4 \equiv 0 \mid < 1, 2, 3, 1, 4, 1, 2, 3, 1, 4 >$

$R_{12} \rightarrow Q_4 \equiv 0 \mid < 1, 2, 3, 1, 2, 4, 1, 2, 3, 1, 2, 4 >$

$R_{14} \rightarrow Q_4 \equiv 0 \mid < 1, 2, 3, 1, 2, 1, 4, 1, 2, 3, 1, 2, 1, 4 >$

It is easy to check that both embedding $R_8 \rightarrow Q_4$ and $R_{12} \rightarrow Q_4$ are 1-step recoverable, $R_{10} \rightarrow Q_4$ is 2-step recoverable, and $R_{14} \rightarrow Q_4$ is 3-step recoverable. □

For now on, the rest of this chapter deals with embeddings in a d-cube, where $d \geq 5$.

Definition: Using the *Modified_DC* scheme, each 3-cube of the embedding hypercubes may contain one of the following partial transition sequences: < 1 >, < 1, 2 >, < 1, 2, 3 >, < 1, 2, 3, 1 >, < 2, 1, 3, 2 >, < 1, 2, 3, 1, 2 >, < 1, 2, 3, 1, 2, 1 > and < 2, 1, 3, 2, 1, 2 >. We will use following notation to denote a 3-cube in embedding hypercubes,

Q_3^1 : a 3-cube contains partial transition sequence < 1 >.

Q_3^2 : a 3-cube contains partial transition sequence < 1, 2 >.

Q_3^3 : a 3-cube contains partial transition sequence < 1, 2, 3 >.

Q_3^4 : a 3-cube contains partial transition sequence < 1, 2, 3, 1 > or < 2, 1, 3, 2 >.

Q_3^5 : a 3-cube contains partial transition sequence < 1, 2, 3, 1, 2 >.

Q_3^6 : a 3-cube contains partial transition sequence < 1, 2, 3, 1, 2, 1 > or

< 2, 1, 3, 2, 1, 2 >.

Lemma 5.1: Embed a length-k ring in a d-cube with the *Modified_DC* scheme, (i) if $k = (5/8)2^d$, then all the 3-cubes in the embedding d-cube will be Q_3^4; (ii) if $k = (7/8)2^d$, then all the 3-cubes in the embedding d-cube will be Q_3^6.

Proof : This lemma can be verified by a similar proof technique (i.e., induction proof based on the size of the embedding hypercube) to that of Lemma 4.2 in Chapter IV. Intuitively thinking, with the divide-and-conquer nature of the *Modified_DC* scheme, the embedding of a $(5/8)2^d$ length ring in a d-cube will have 5 consecutive active nodes on each 3-cube of the embedding d-cube. According to the previous definition, these 3-cubes with 5 consecutive active nodes must contains the partial transition sequence < 1, 2, 3, 1 > or < 2, 1, 3, 2 >. A similar argument can also be applied to the case of a length-$(7/8)2^d$ ring. □

Lemma 5.2: Embed a ring of length k in a d-cube with the *Modified_DC* scheme, (i) if $(7/8)2^{d-1} < k \leq (5/8)2^d$, then the possible combination of 3-cubes in the embedding d-cube are: all Q_3^3, all Q_3^4, Q_3^2 with Q_3^3, and Q_3^3 with Q_3^4; (ii) if $(5/8)2^d < k \leq (7/8)2^d$, then the possible combination of 3-cubes in the embedding d-cube are: all Q_3^5, all Q_3^6, Q_3^4 with Q_3^5, and Q_3^5 with Q_3^6.

Proof : This follows directly from Lemma 5.1 and the algorithm of the *Modified_DC* scheme. Note that, it is impossible to have the combinations of 3-cubes like all Q_3^1, all Q_3^2, and Q_1^1 with Q_3^2, since for these combinations the length of rings embedded will be $\leq (7/8)2^{d-1}$. □

Lemma 5.3: Embed a length-k ring in a d-cube with the *Modified_DC* scheme, where $(7/8)2^{d-1} < k \leq (5/8)2^d$ and $d \geq 5$, the embedding generated is 1-step recoverable.

Proof : According to Lemma 5.2, this theorem can be divided to several cases to prove it based on the combination of the 3-cubes in the embedding d-cube (i.e., cases for all Q_3^3, all Q_3^4, Q_3^2 with Q_3^3, and Q_3^3 with Q_3^4). Here, we only consider the case that all 3-cubes are Q_3^4, and show that the system can recover any single fault in 1 step. A similar proof can be applied to other cases to show that in those cases the system is also 1-step recoverable.

We further divide the case for all Q_3^4 to two subcases: one case for fault happens in a Q_3^4 with partial transition sequence < 1, 2 , 3, 1 >, and the other case for fault happens in a Q_3^4 with partial transition sequence < 2, 1, 3, 2 >.

Case I : Fault happens in a Q_3^4 with the partial transition sequence < 1, 2, 3, 1 >. Figure 12 shows the connection pattern for such a Q_3^4 with < 1, 2, 3, 1 > to other Q_3^4's in the embedding d-cube. In Figure 12, nodes with labels beginning with 'A' are in active state, and nodes with labels beginning with 'S' are in spare state. By applying *xor-reconfiguration* algorithm, it is easy to see the following facts:

1) The action: A_1 » S_1'' will recover the faulty node A_1.

2) The action: A_2 » S_1 will recover the faulty node A_2.

3) The action: A_3 » S_2 will recover the faulty node A_3.

4) The action: A_4 » S_1 will recover the faulty node A_4.

5) The action: A_5 » S_1' will recover the faulty node A_5.

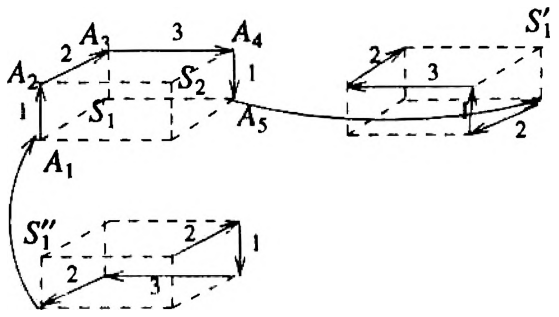


Figure 12. Connection pattern of a Q_3^4 (i.e., with partial transition sequence < 1, 2, 3, 1 >) with other Q_3^4's.

Thus, for each possible faulty node in any Q_3^4 with the partial transition sequence < 1, 2, 3, 1 >, the faulty node can be recovered in 1 step.

Case II : A fault happens in a Q_3^4 with the partial transition sequence < 2, 1, 3, 2 >. Figure 13 shows the connection pattern for such a Q_3^4 with < 2, 1, 3, 2 > to other Q_3^4's in the embedding d-cube. In Figure 13, nodes with labels beginning with 'B' are in active state, and nodes with labels beginning with 'S' are in spare state. By applying *xor-reconfiguration algorithm*, it is easy to see the following facts:

1) The action: B_1 » S_1'' will recover the faulty node B_1.

2) The action: B_2 » S_1 will recover the faulty node B_2.

3) The action: B_3 » S_2 will recover the faulty node B_3.

4) The action: B_4 » S_1 will recover the faulty node B_4.

5) The action: B_5 » S_1' will recover the faulty node B_5.

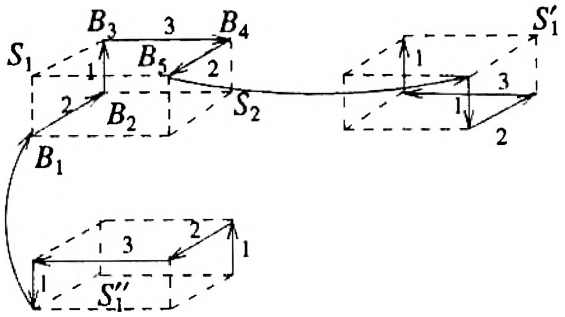


Figure 13. Connection pattern of a Q_3^4 (i.e., with partial transition sequence < 2, 1, 3, 2 >) with other Q_3^4's.

Thus, for each possible faulty node in any Q_3^4 with the partial transition sequence < 2, 1, 3, 2 >, the faulty node can be recovered in 1 step. □

Lemma 5.4: Embed a length-k ring in a d-cube with the *Modified_DC* scheme, where $k = (6/8)2^d$, the embedding generated is 1-step recoverable and all the 3-cubes in the embedding d-cube are Q_3^5.

Proof: In Chapter IV, we have shown that to embed a ring of length $(6/8)2^d$ in a d-cube with the *DC* scheme, the embedding generated are 1-step recoverable, and all the 3-cubes in the embedding d-cube are Q_3^5 (i.e., with partial transition sequence < 1, 2, 3, 1, 2 >). Comparing the algorithms of the *DC* scheme and the *Modified_DC* scheme, it can be checked that both schemes will come out the same transition sequence for embedding of a length-$(6/8)2^d$ ring in a d-cube. Therefore, the lemma follows. □

Lemma 5.5: Embed a length-k ring in a d-cube with the *Modified_DC* scheme, where $(5/8)2^d < k \leq (7/8)2^d$, $k \neq (6/8)2^d$, and $d \geq 5$, the embedding generated is 2-step recoverable.

Proof: Using the similar proof technique to that of Lemma 5.3. □

Theorem 5.1: Embed rings in d-cubes, $d \geq 5$, with the *Modified_DC* scheme, the embeddings generated are 2-step recoverable in worst case.

Proof: Directly from Lemma 5.3, Lemma 5.4, and Lemma 5.5. □

C. Performance Comparison

In this section, we compare the performance between the *DC* scheme and the *Modified_DC* scheme.

Table VI. Comparison of recoverability between the *DC* scheme and the *Modified_DC* scheme for dimensions of embedding hypercubes ≥ 5.

($d \geq 5$)	*DC* scheme	*Modified_DC* scheme
$(7/8)2^{d-1} < k \leq (4/8)2^d$	1-step recoverable	1-step recoverable
$(4/8)2^d < k \leq (5/8)2^d$	2-step recoverable	1-step recoverable
$(5/8)2^d < k < (6/8)2^d$	2-step recoverable	2-step recoverable
$k = (6/8)2^d$	1-step recoverable	1-step recoverable
$(6/8)2^d < k \leq (7/8)2^d$	3-step recoverable	2-step recoverable

In Table VI we compare the recoverability between the system with the *DC* scheme and the system with the *Modified_DC* scheme. It can be seen that the system with the *Modified_DC* scheme has better performance than that with *DC* scheme. Overall, the system with the *DC* scheme is 3-step recoverable and the system with the *Modified_DC* scheme is 2-step recoverable.

Corollary 5.2: Embed a length-k ring in a d-cube with the *Modified_DC* scheme, if $(7/8)2^{d-1} < k \leq (5/8)2^d$ or $k = (6/8)2^d$, then all the k active nodes in the embedding will need 1 step for recovery when they become faulty.

Proof: Directly from Lemma 5.3 and Lemma 5.4. □

Corollary 5.3: To embed a length-k ring in a d-cube with the *Modified_DC* scheme, where $(5/8)2^d < k < (6/8)2^d$, if there exists p Q_3^4's with partial transition sequence < 1, 2, 3, 1 > and q Q_3^4's with < 2, 1, 3, 2 > in the embedding d-cube, then there are $2(p-q)$ active nodes which need 2 steps for recovery and $k-2(p-q)$ active nodes which need 1 step for recovery when they become faulty.

Proof : According to Lemma 5.2, Lemma 5.4, and the algorithm of the *Modified_DC* scheme, embed a ring of above length constrain in a d-cube with the *Modified_DC* scheme, the resulting combination of 3-cubes in the embedding d-cube is Q_3^4 with Q_3^5. That is, the embedding d-cube consists of only Q_3^4's and Q_3^5's. The possible connection patterns among these Q_3^4's and Q_3^5's are follows,

Case (i) A Q_3^4 with < 1, 2, 3, 1 > connecting to a Q_3^4 with < 2, 1, 3, 2 > as in Figure 14(a). In Figure 14(a), the recovery processing for active nodes A_1 and A_{10} are undecidable yet (i.e., need more information) and the rest of the active nodes can be recovered in 1 step. To discuss the recovery processing for nodes A_1 and A_{10}, we need to further consider about what kinds of 3-cubes connecting to the Q_3^4 with < 1, 2, 3, 1 > in Figure 14(a) and what kinds of 3-cubes that the same Q_3^4 connecting to. However, those possible connections will fall into the five cases that we discuss here.

Case (ii) A Q_3^4 with < 1, 2, 3, 1 > connecting to a Q_3^5 as in Figure 14(b). In Figure 14(b), the active nodes A_5 and A_6 can be recovered in 2 steps, the recovery processing for active nodes A_1 and A_{11} are undecidable yet, and the rest of the active nodes can be recovered in 1 step.

Case (iii) A Q_3^4 with < 2, 1, 3, 2 > connecting to a Q_3^4 with < 1, 2, 3, 1 > as in Figure 14(c). In Figure 14(c), the recovery processing for active nodes A_1 and A_{10} are undecidable yet, and the rest of the active nodes can be recovered in 1 step.

Case (iv) A Q_3^5 connecting to a Q_3^4 with < 1, 2, 3, 1 > as in Figure 14(d). In Figure 14(d), the recovery processing for active nodes A_1 and A_{11} are undecidable yet, and the rest of the active nodes can be recovered in 1 step.

Case (v) A Q_3^5 connecting to a Q_3^5 as in Figure 14(e). In Figure 14(e), the recovery processing for active nodes A_1 and A_{12} are undecidable yet, and the rest of the active nodes can be recovered in 1 step.

So, in these five connection cases, only case (ii) results in 2-step recoverable active nodes. Hence, to count the number of active nodes which are 2-step recoverable, we need only to count the number of Q_3^4's with < 1, 2, 3, 1 > which connect to Q_3^5's. That is,

The number of 2-step recoverable nodes

= 2 × (the number of Q_3^4's with < 1, 2, 3, 1 > that connect to Q_3^5's)

= 2 × (subtracting the number of Q_3^4's with < 1, 2, 3, 1 > that connect to Q_3^4's with < 2, 1, 3, 2 > from the total number of Q_3^4 with < 1, 2, 3, 1 >)

= 2 × (subtracting the number of Q_3^4's with < 2, 1, 3, 2 > from the total number of Q_3^4 with < 1, 2, 3, 1 >)

= $2 \times (p - q)$ □

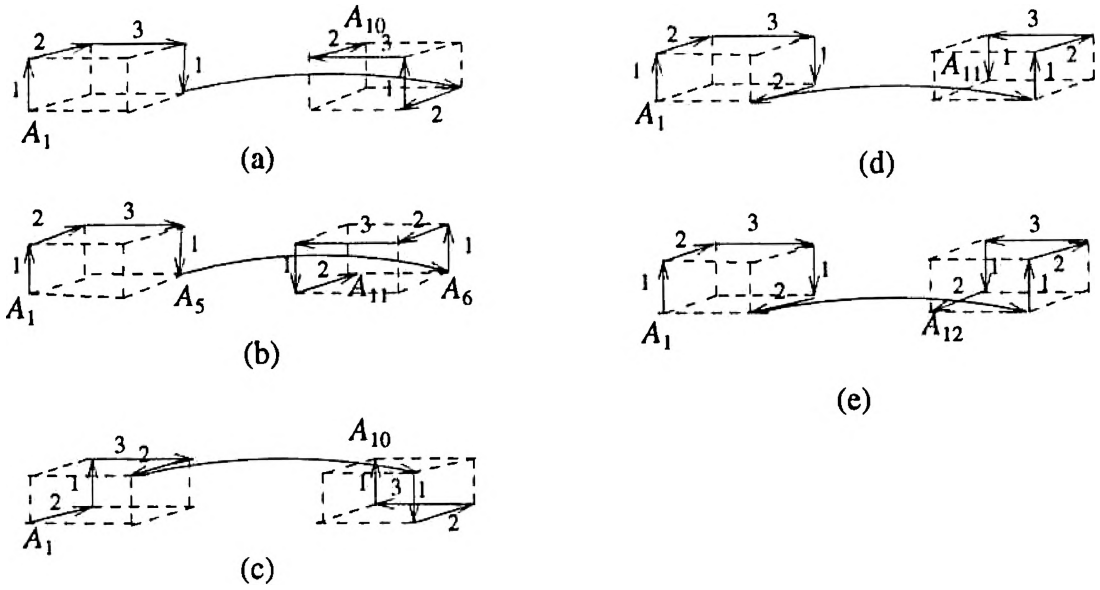


Figure 14. (a) A Q_3^4 with < 1, 2, 3, 1 > connecting to a Q_3^4 with < 2, 1, 3, 2 >. (b) A Q_3^4 with < 1, 2, 3, 1 > connecting to a Q_3^5. (c) A Q_3^4 with < 2, 1, 3, 2 > connecting to a Q_3^4 with < 1, 2, 3, 1 >. (d) A Q_3^5 connecting to a Q_3^4 with < 1, 2, 3, 1 >. (e) A Q_3^5 connecting to a Q_3^5.

Corollary 5.4: Embed a length-k ring in a d-cube with the *Modified_DC* scheme, where $(6/8)2^d < k < (7/8)2^d$, if there exist p Q_3^6's with partial transition sequence < 1, 2, 3, 1, 2, 1 > and q Q_3^6's with < 2, 1, 3, 2, 1, 2 > in the embedding d-cube, then there are $2p + 6q$ active nodes which need 2 steps for recovery and $k - 2p - 6q$ active nodes which need 1

step for recovery when they become faulty.

Proof : This corollary can be verified by a similar proof technique to that of Corollary 5.3. □

Now, we use the results from the Corollary 5.2, Corollary 5.3, and Corollary 5.4 to compare the performance between the *DC* scheme and the *Modified_DC* scheme with the same criteria: *average expansion, maximum expansion, average recovery steps, and maximum recovery steps*, which are described in Chapter IV. The result of this comparison is given in Table VII.

Table VII. Comparison of the performances of the *DC* scheme and the *Modified_DC* scheme for embedding rings of length k in d-cubes such that $d \geq 5$, $(7/8)2^{d-1} < k \leq (7/8)2^d$, and $k \leq 1000$.

Schemes	Average Expansion	Maximum Expansion	Average Recovery Steps	Maximum Recovery Steps
DC	1.64	2.28	1.3	3
Modified_DC	1.64	2.28	1.1	2

From Table VII we can see that the average expansion and the maximum expansion of both embedding schemes are the same, since both schemes impose the same length constrain, $(7/8)2^{d-1} < k \leq (7/8)2^d$, on rings that are embeddable in d-cubes. Note that, for the system with the *Modified_DC* scheme the average number of recovery steps needed to recover one single fault is 1.1, and it is 1.3 to the case of the *DC* scheme. Again, this shows the performance improvement of the *Modified_DC* scheme over the

DC scheme.

D. Concluding Remarks

In this chapter, we have presented the *Modified_DC* scheme which is an enhanced version of the reported *DC* scheme for rings embedded in hypercubes. It is shown that with some simple permutations of partial transition sequences, the system with the *Modified_DC* scheme turns out to have better performance, in term of recoverability and average number of recovery steps needed, than that with the *DC* scheme. Overall the *Modified_DC* scheme is 2-step recoverable and needs average 1.3 steps to recover one single fault when the dimension of the an embedding hypercube is ≥ 5.

VI. 1-STEP RECOVERABLE EMBEDDINGS

An embedding is 1-step recoverable if any single fault occurs the embedding can be reconfigured in one reconfiguration step such that the new embedding is still a ring of the original length. 1SRE's are most interesting because of their efficient reconfigurability due to faults. In this chapter, we present an efficient method to systematically construct such embedding. Our scheme is based on a composition idea by which a 1SRE in a given dimension hypercube is formed by combining two 1SRE's in two lower dimension hypercubes. We show that to embed a length-k (even) ring in a d-cube where $6 \leq k \leq (3/4)2^d$ and $d \geq 3$, our scheme will guarantee finding a 1SRE, provided such an embedding exists. Compared with other existing schemes for constructing 1SRE's, our scheme surpasses them in terms of applicability of the scheme and the expansion of the resulting embeddings.

A basic condition derived from the topological properties of the hypercube for a given embedding to be a 1SRE is described as follows.

Corollary 6.1: Let $a_1, a_2, \cdots, a_{k-1}, a_k$ be the listing in order of the k active nodes for an embedding of a length-k ring in a d-cube, and let $A = \{a_1, a_2, \cdots, a_{k-1}, a_k\}$. This embedding will be a 1SRE if and only if,

$$\{a_{XOR} \mid a_{XOR} = XOR(\, a_i, a_{(i \bmod k)+1}, a_{((i+1) \bmod k)+1}\,),\ 1 \leq i \leq k,\} \cap A = \varnothing$$

Proof: It will be clear that the above condition is a sufficient condition for the existence of a 1SRE. However, we need to discuss about the condition being a necessary one.

Figure 15 shows a partial portion of an embedding (i.e., denotes in arrowed lines) in a 2D plane of a hypercube. a_i, a_{i+1}, a_{i+2}, and s represent the addresses of four nodes in a 2D plane of a hypercube. According to the topological properties of hypercubes, XOR(a_i, a_{i+1}, a_{i+2}) = s and there exist only 2 node-disjoint shortest paths from node a_i to node a_{i+2} ($a_i \rightarrow a_{i+1} \rightarrow a_{i+2}$ and $a_i \rightarrow s \rightarrow a_{i+2}$) since the Hamming distance between nodes a_i and a_{i+2} is 2 [SaSc88]. With the definition of recovery step, we know that node s is the only node that can be used to recover the node a_{i+1} in 1 step. So, node s must be in spare state if the faulty node a_{i+1} need to be recoverable in 1 step. Since a 1SRE means any single faulty node must be recovered in 1 step, every node with address XOR(a_i, $a_{(i \bmod k)+1}$, $a_{((i+1) \bmod k)+1}$), $1 \le i \le k$, needs to be in spare state. We conclude that condition (1) is also a necessary condition for the existence of a 1SRE. □

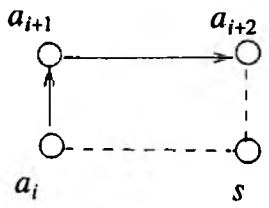


Figure 15. A partial portion of an embedding in a 2D plane of a hypercube.

It is easy to check that the embedding in Figure 2 is, in fact, a 1SRE, since any possible faulty node in the embedding can be recovered by spare nodes 0010, 0101, 1001, or 1110 in 1 step (e.g., faulty node 0001, 0111, or 1000 can be recovered by spare node 0010). It can also be verified that the condition in corollary 6.1 is satisfied for this 1SRE.

A. A Composition Method

In this section, we discuss a composition method to systematically construct 1SRE's for embedding a length-k (even) ring in a d-cubes, where $6 \leq k \leq (3/4)2^d$ and $d \geq 3$. Our scheme is based on the following idea: if there exists a 1SRE of length-k_1 ring in a d-cube and a 1SRE of a length-k_2 ring in a d-cube, then there exists a 1SRE of a length-$(k_1 + k_2)$ ring in a $(d + 1)$-cube.

Lemma 6.1: For odd length rings and rings with length less than 6, there are no 1SRE's in hypercubes.

Proof : It is shown that there are no cycles of odd length in hypercubes in [SaSc88], therefore, no odd length rings can be embedded in hypercubes. Since the shortest cycle in a hypercube has length 4, a length-2 ring cannot be embedded in hypercubes. For a length-4 ring in a hypercube, the embedding must constitute the 4 nodes of a 2-D plane of the hypercube, and it is easy to see that such embedding is not a 1SRE. □

Lemma 6.2: If there exists a 1SRE of a length-k ring in a d-cube, then there exists a 1SRE of a length-k ring in a $(d + 1)$-cube.

Proof: The proof is trivial. Just leave one d-cube unused, and the same 1SRE of a k ring in a d-cube is also a 1SRE of a k length ring in a $(d + 1)$-cube. □

Lemma 6.3: If there exists a 1SRE of a length-k_1 ring in a d-cube and a 1SRE of a length-k_2 ring in a d-cube, then there exists a 1SRE of a length-(k_1+k_2) ring in a $(d + 1)$-cube.

Proof: First, we present a compositional method to form a 1SRE of a length-(k_1+k_2) ring in a $(d+1)$-cube from two existing 1SRE's of length-k_1 and length-k_2 rings in a d-cube. Let the two existing 1SRE's be

$$R_{k_1} \to Q_d \equiv 0 \mid < x_1, y_1, z_1, \cdots >, \text{ and}$$
$$R_{k_2} \to Q_d \equiv 0 \mid < x_2, y_2, z_2, \cdots >.$$

By swapping some coordinates in the transition sequence of $R_{k_1} \to Q_d$, we can have the first three coordinate places of both transition sequences be the same. For example, if $R_{k_1} \to Q_d \equiv 0 \mid < 1, 2, 3, 1, 2, 4, \cdots >$ and $R_{k_2} \to Q_d \equiv 0 \mid < 1, 4, 3, \cdots >$, then we have $R_{k_1} \to Q_d \equiv 0 \mid < 1, 4, 3, 1, 4, 2, \cdots >$ after the swapping (exchange 2 and 4). Then, we change the starting node of the new $R_{k_1} \to Q_d$ to be $0 \oplus 2^{y_2-1}$ ($\oplus$: the bitwise-XOR operation). So, we have

$$R_{k_1} \to Q_d \equiv 0 \oplus 2^{y_2-1} \mid < x_2, y_2, z_2, \cdots >, \text{ and}$$
$$R_{k_2} \to Q_d \equiv 0 \mid < x_2, y_2, z_2, \cdots >$$

such that both embeddings remain 1-step recoverable.

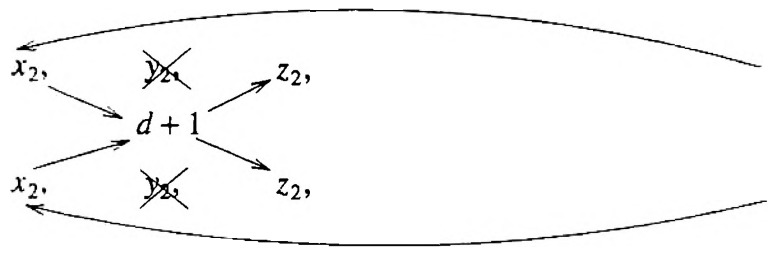


Figure 16. A transition sequence for $R_{(k_1+k_2)} \to Q_{(d+1)}$ by composing two transition sequences of $R_{k_1} \to Q_d$ and $R_{k_2} \to Q_d$.

To form a transition sequence for a 1SRE $R_{(k_1+k_2)} \rightarrow Q_{(d+1)}$, we simply replace the first y_2 coordinate place in both transition sequences for $R_{k_1} \rightarrow Q_d$ and $R_{k_2} \rightarrow Q_d$ with $d+1$ coordinate place, and then combine these two transition sequences together. This processing is depicted in Figure 16. Thus, we have

$$R_{k_1+k_2} \rightarrow Q_{d+1} \equiv 0 \mid < x_2, d+1, z_2, \cdots, x_2, d+1, z_2, \cdots >$$

Second, we need to show that the embedding $R_{k_1+k_2} \rightarrow Q_{d+1}$ cited is a 1SRE. Figure 17 will help us to explain this proof more clearly. Figure 17(a) and 17(b) depict the partial portions of $R_{k_1} \rightarrow Q_d$ and $R_{k_2} \rightarrow Q_d$, respectively. Figure 17(c) depicts the $R_{(k_1+k_2)} \rightarrow Q_{(d+1)}$, constructed by following the composition method. Note that, nodes s_1 and s_2 in Figure 17(a) and nodes s_3 and s_4 in Figure 17(b) must be spare nodes. For $R_{(k_1+k_2)} \rightarrow Q_{(d+1)}$ in Figure 17(c), all the active nodes except nodes a_{i_1}, a_{i_2}, a_{i_3}, and a_{i_4} remain to be 1-step recoverable as they are in $R_{k_1} \rightarrow Q_d$ and $R_{k_2} \rightarrow Q_d$. And, all the spare nodes in $R_{k_1} \rightarrow Q_d$ and $R_{k_2} \rightarrow Q_d$ remain to be spare nodes as they join in $R_{(k_1+k_2)} \rightarrow Q_{(d+1)}$. Thus, we need only to check the recovery processes for nodes a_{i_1}, a_{i_2}, a_{i_3}, and a_{i_4} for verifying the 1-step recoverability for $R_{(k_1+k_2)} \rightarrow Q_{(d+1)}$. Since active nodes a_{i_1}, a_{i_2}, a_{i_3}, and a_{i_4} can be recovered by spare nodes s_3, s_4, s_1, and s_2 in one step, respectively, this $R_{(k_1+k_2)} \rightarrow Q_{(d+1)}$ is a 1SRE. □

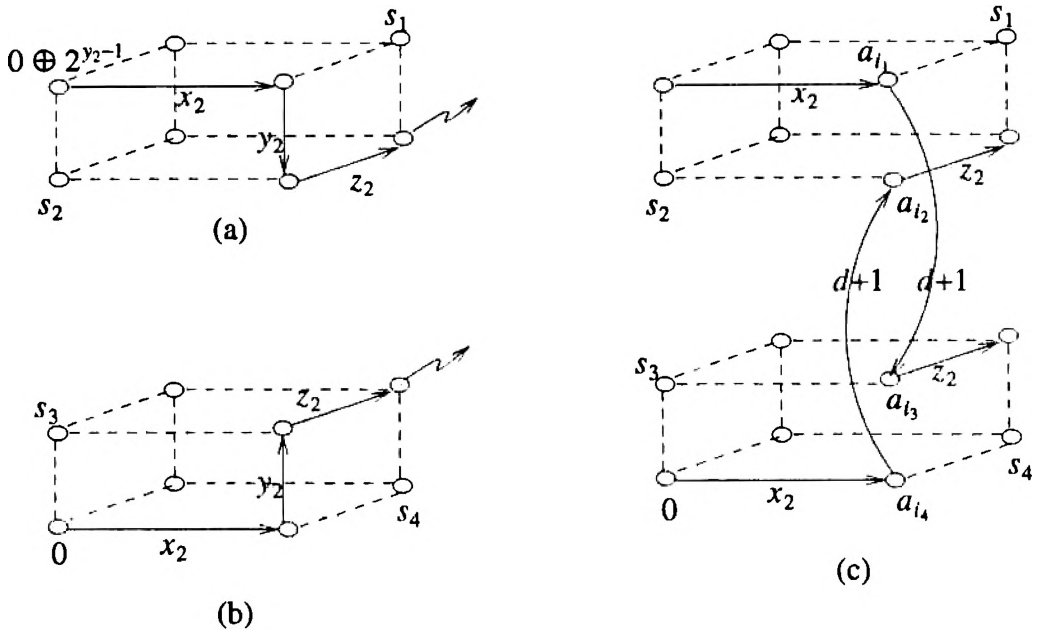


Figure 17. (a) Partial portion of $R_{k_1} \rightarrow Q_a \equiv 0 \oplus 2^{y_2-1} \mid < x_2, y_2, z_2, \cdots >$. (b) Partial portion of $R_{k_2} \rightarrow Q_d \equiv 0 \mid < x_2, y_2, z_2, \cdots >$. (c) $R_{k_1+k_2} \rightarrow Q_{d+1}$ composed from $R_{k_1} \rightarrow Q_d$ and $R_{k_2} \rightarrow Q_d$.

Example: By using the composition method in Lemma 6.3, we demonstrate the construction of a 1SRE of a length-20 ring in a 5-cube by two smaller 1SRE's, $R_8 \rightarrow Q_4 \equiv 0 \mid < 1, 2, 3, 4, 1, 2, 3, 4 >$ and $R_{12} \rightarrow Q_4 \equiv 0 \mid < 1, 4, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3 >$. With the changing of starting node and swapping the elements in the transition sequence, we have $R_8 \rightarrow Q_4 \equiv 0 \oplus 2^3 \mid < 1, 4, 3, 2, 1, 4, 3, 2 >$ and $R_{12} \rightarrow Q_4 \equiv 0 \mid < 1, 4, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3 >$. Deleting two coordinate-4 places and combining the two sequences together by adding two coordinate-5 places, we have $R_{20} \rightarrow Q_5 \equiv 0 \mid < 1, 5, 3, 2, 1, 4, 3, 2, 1, 5, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3 >$, which is a 1SRE. Fig. 18 depicts such construction of a 1SRE.

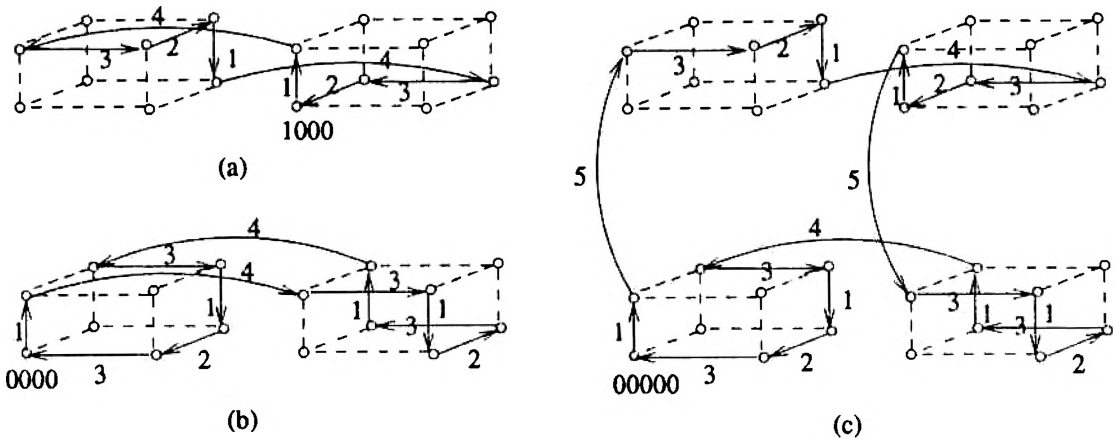


Figure 18. (a) 1SRE $R_8 \rightarrow Q_4 == 0 \oplus 2^3 \mid < 1, 4, 3, 2, 1, 4, 3, 2 >$. (b) 1SRE $R_{12} \rightarrow Q_4 \equiv 0 \mid < 1, 4, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3 >$. (c) 1SRE $R_{20} \rightarrow Q_5 \equiv 0 \mid < 1, 5, 3, 2, 1, 4, 3, 2, 1, 5, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3 >$.

Lemma 6.4: There exists a 1SRE of a length-6 ring in a 3-cube.

Proof: The embedding, $R_6 \rightarrow Q_3 \equiv 0 \mid < 1, 2, 3, 1, 2, 3 >$, is 1-step recoverable. □

Lemma 6.5: There are 1SRE's of length-6, 8, and 12 rings in a 4-cube, but no 1SRE of a length-10 ring in a 4-cube.

Proof: The proof is divided into following cases:

Case I. According to Lemma 6.2 and Lemma 6.4, For a length-6 ring, $R_6 \rightarrow Q_4 \equiv 0 \mid < 1, 2, 3, 1, 2, 3 >$ is a 1SRE (Lemma 6.2 and Lemma 6.4).

Case II. For a length-8 ring, $R_8 \rightarrow Q_4 \equiv 0 \mid < 1, 2, 3, 4, 1, 2, 3, 4 >$ is a 1SRE.

Case III. For a length-12 ring, by using the composition method in Lemma 6.3, we can construct a 1SRE by combining two 1SRE's of $R_6 \rightarrow Q_3 \equiv 0 \mid < 1, 2, 3, 1, 2, 3 >$. We have $R_{12} \rightarrow Q_4 \equiv 0 \mid < 1, 4, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3 >$, which is a 1SRE.

Case IV. For a length-10 ring, an easy computer program was used to enumerate all possibilities, and we found out there is no 1RE for a length-10 ring in a 4-cube. □

Lemma 6.6: For length-6, 8, 10, $\cdots$, 18, 20, and 24 rings, there are 1SRE's in a 5-cube, but no 1SRE of a length-22 ring in a 5-cube.

Proof: The proof is divided into the following cases:

Case I. For length-6, 8, and 12 rings, there are 1SRE's in a 5-cube (Lemma 6.2 and Lemma 6.5).

Case II. For a length-10 ring, the embedding $R_{10} \rightarrow Q_5 \equiv 0 \mid < 1, 2, 3, 4, 5, 1, 2, 3, 4, 5 >$ is a 1SRE.

Case III. For length-14, 16, 18, 20, and 24 rings, there are 1SRE's in a 5-cube. These 1SRE's are shown in Table VIII.

Case IV. For a length-22 ring, the composition method does not work, since 22 = 12 + 10 and there is no 1SRE of a length-10 ring in a 4-cube. A computer program was used to enumerate all possibilities, and it shows there is no 1SRE of a length-22 ring in a 5-cube. □

Table VIII. 1SRE's of length-14, 16, 18, 20, and 24 rings in a 5-cube

Embeddings	Constructed from
$R_{14} \rightarrow Q_5 \equiv 0 \mid < 1, 5, 3, 4, 1, 2, 3, 4, 1, 5, 3, 1, 2, 3 >$	$R_6 \rightarrow Q_4$, $R_8 \rightarrow Q_4$
$R_{16} \rightarrow Q_5 \equiv 0 \mid < 1, 5, 3, 4, 1, 2, 3, 4, 1, 5, 3, 4, 1, 2, 3, 4 >$	$R_8 \rightarrow Q_4$, $R_8 \rightarrow Q_4$
$R_{18} \rightarrow Q_5 \equiv 0 \mid < 1, 5, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3, 1, 5, 3, 1, 4, 3 >$	$R_6 \rightarrow Q_4$, $R_{12} \rightarrow Q_4$
$R_{20} \rightarrow Q_5 \equiv 0 \mid < 1, 5, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3, 1, 5, 3, 2, 1, 4, 3, 2 >$	$R_8 \rightarrow Q_4$, $R_{12} \rightarrow Q_4$
$R_{24} \rightarrow Q_5 \equiv 0 \mid < 1, 5, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3, 1, 5, 3, 1, 2, 3, 1, 4, 3, 1, 2, 3 >$	$R_{12} \rightarrow Q_4$, $R_{12} \rightarrow Q_4$

Lemma 6.7: For length-6, 8, 10, $\cdots$, 44, 46, and 48 rings (i.e., rings of length k, $6 \leq k \leq (3/4)2^6$), there are 1SRE's in a 6-cube.

Proof: The proof is divided into the following cases:

Case I. For length-6, 8, $\cdots$, 20, and 24 rings, there are 1SRE's in a 6-cube (Lemma 6.2 and Lemma 6.6).

Case II. For length-22, 26, $\cdots$, 44, and 48 rings, there are 1SRE's in a 6-cube (Lemma 6.3). Summary of constructing these 1SRE's is given in Table IX.

Case III. For a length-46 ring, the composition method does not work, since 46 = 24 + 22 and there is no 1SRE for a length-22 ring in a 5-cube. However, the following embedding comes out,

$$R_{46} \rightarrow Q_6 \equiv 0 \mid < 1, 2, 3, 1, 2, 4, 1, 2, 3, 1, 2, 5, 1, 2, 3, 1, 2, 4, 1, 2, 3, 1, 6,$$

3, 4, 1, 2, 4, 1, 5, 4, 3, 5, 1, 2, 5, 1, 3, 4, 1, 3, 2, 5, 1, 6, 5 >,

which is a 1SRE.

Table IX. Summary of constructing 1SRE's of length-22, 26, $\cdots$ 44 and 48 rings in a 6-cube

Embeddings	Constructed from	Embeddings	Constructed from
$R_{22} \rightarrow Q_6$	$R_{10} \rightarrow Q_5$, $R_{12} \rightarrow Q_5$	$R_{36} \rightarrow Q_6$	$R_{18} \rightarrow Q_5$, $R_{18} \rightarrow Q_5$
$R_{26} \rightarrow Q_6$	$R_{12} \rightarrow Q_5$, $R_{14} \rightarrow Q_5$	$R_{38} \rightarrow Q_6$	$R_{18} \rightarrow Q_5$, $R_{20} \rightarrow Q_5$
$R_{28} \rightarrow Q_6$	$R_{14} \rightarrow Q_5$, $R_{14} \rightarrow Q_5$	$R_{40} \rightarrow Q_6$	$R_{20} \rightarrow Q_5$, $R_{20} \rightarrow Q_5$
$R_{30} \rightarrow Q_6$	$R_{14} \rightarrow Q_5$, $R_{16} \rightarrow Q_5$	$R_{42} \rightarrow Q_6$	$R_{18} \rightarrow Q_5$, $R_{24} \rightarrow Q_5$
$R_{32} \rightarrow Q_6$	$R_{16} \rightarrow Q_5$, $R_{16} \rightarrow Q_5$	$R_{44} \rightarrow Q_6$	$R_{20} \rightarrow Q_5$, $R_{24} \rightarrow Q_5$
$R_{34} \rightarrow Q_6$	$R_{16} \rightarrow Q_5$, $R_{18} \rightarrow Q_5$	$R_{48} \rightarrow Q_6$	$R_{24} \rightarrow Q_5$, $R_{24} \rightarrow Q_5$

Theorem 6.1: There exists a 1SRE of a length-k (even) ring in a d-cube, where $6 \leq k \leq (3/4)2^d$ and $d \geq 3$, except when k=10 with d=4, and k=22 with d=5.

Proof: We use an induction proof based on the size of d to prove this theorem.

Induction base: For $d = 3$, $d = 4$, $d = 5$, and $d = 6$, the theorem follows based on Lemma 6.4, Lemma 6.5, Lemma 6.6, and Lemma 6.7, respectively.

Induction hypothesis: Assume the theorem follows when $d = m \geq 6$. That is, there exists a 1SRE of a length-k ring in a m-cube, where $6 \leq k \leq (3/4)2^m$.

Induction step: $d = m + 1$

Case I: $6 \le k \le (3/4)2^m$. According to induction hypothesis and Lemma 6.2, there exists a 1SRE of a length-k ring in a $(m+1)$-cube.

Case II: $(3/4)2^m < k \le (3/4)2^{(m+1)}$. Since k is even, we have $k = k_1 + k_2$, where k_1 and k_2 are even, and $6 \le k_1 \le (3/4)2^m$ and $6 \le k_2 \le (3/4)2^m$. By the induction hypothesis there exists a 1SRE of a length-k_1 ring in a m-cube and a 1SRE of a length-k_2 ring in a m-cube. By using the composition method in Lemma 6.3, we can construct a 1SRE of a length-k ring in a $(m+1)$-cube with these two smaller 1SRE's. □

B. Complexity and Comparison

The running time complexity of our schemes for embedding a length-n ring in the hypercube is approximately about

$$T(n) = 2T(n/2) + c,$$

and we have $T(n) = O(n)$. So, the running time complexity of our scheme is polynomial to the length of the ring embedded.

The comparison of our scheme to the existing schemes for 1SRE's (i.e., [ChLT88b,Leet90,PrMe88]) is described as follows. Basically, the lengths of rings that are applicable to other schemes in [ChLT88b,PrMe88] are very restricted, since both schemes are designed for embedding rings of lengths 2^{d-1} and $(3/4)2^d$ only in a d-cube. It is shown that for rings of such lengths both schemes have efficient ways to construct 1SRE's, however, they are not able to construct 1SRE's of rings of other lengths. Compared with the schemes in [ChLT88b,PrMe88], our scheme can be applied to more rings of different lengths.

Unlike the schemes in [ChLT88b,PrMe88], the scheme in [Leet90] can construct 1SRE's of any even length ring in the hypercube. However, the trade-off for such excellent performance is that the resulting 1SRE's are of large expansion (the ratio of the size (in number of nodes) of the embedding hypercube to that of the embedded ring). For many case, the scheme will embed a ring in a hypercube that is one dimension larger than is necessary in order to achieve efficient reconfiguration due to faults. For example, a length-16 or length-20 ring will be embedded in a 6-cube by the scheme , although a 5-cube is enough to accommodate the ring. Compared with the scheme in [Leet90], our scheme has better performance in terms of the processor utilization of the resulting 1SRE's.

We summarize the comparison of our scheme to other existing schemes for constructing 1SRE's in Table X.

Table X. Comparison among schemes for constructing 1SRE's

Schemes	Applicability to the ring	Expansion of resulting 1SRE's
[ChLT88b]	Rings of length 2^{d-1} and $(3/4)2^d$ in a d-cube	-
[PrMe88]	Rings of length $(3/4)2^d$ in a d-cube	-
[Leet90]	All even length rings	Large expansion in many cases
Our scheme	Rings of length from 6 to $(3/4)2^d$ in a d-cube	Smaller expansion than that for [8]

C. Two Special Cases

In this section, we will formally show that there is no existence of 1SRE's for the two embedding cases: a length-10 ring in a 4-cube and a length-22 ring in a 5-cube.

1. A Necessary and Sufficient Condition. Note that, not every list of digits is the coordinate sequence of a cycle. A typical list, such as <1,2,1,3,2,3,1,3,1>, represents a way of wandering along lines of the cube graph, possibly visiting some vertices more than once. The observation leads simply to the following result.

Theorem 6.2: [Gilb58] A k-tuple $T = <t_1, t_2, \cdots, t_k>$ $t_i = 1, \cdots, d$ is the transition sequence of a cycle of a d-cube if and only if every one of the blocks of length $1, \cdots,$ or $k-1$ contains some digit an odd number of times while T itself contains every digit an even number of times.

In the following we will consider a necessary and sufficient condition for a k-tuple $T = <t_1, t_2, \cdots, t_k>$ $t_i = 1, \cdots, d$ not being the transition sequence of a 1SRE.

Corollary 6.2: Let $W = v_0, c_1, v_1, c_2, \cdots, v_{k-1}, c_k, v_k$ be a walk on the hypercube. Two distinct vertices v_i and v_j $(i < j)$ on W are adjacent along the pth coordinate if and only if the block $c_{i+1}, c_{i+2}, \cdots, c_j$ (the coordinates between v_i and v_j along W) contains digit p an odd number of times and other digits all even numbers of times.

Proof : By thinking each coordinate change i as a bit-flipping processing from "0" to "1" or "1" to "0" on the ith coordinate, this lemma clearly follows. □

Theorem 6.3: A k-tuple $T = <t_1, t_2, \cdots, t_k> t_i = 1, \cdots, d$ is *not* a transition sequence of a 1SRE of a length-k ring in a d-cube if and only if one of the following conditions is true.

(1) For some block $t_i, \cdots, t_j$ where $j - i + 1 < k$, every digit in it is repeated an even number of times.

(2) For some block $t_i, t_{i+1}, \cdots, t_{i+m-1}, t_{i+m}$ (m is even) where $t_{i+1} = q$, it contains digit q an odd number of times and each other digit an even number of times.

(3) For some block $t_i, t_{i+1}, \cdots, t_{i+m-1}, t_{i+m}$ (m is even) where $t_{i+m-1} = q$, it contains digit q an odd number of times and each other digit an even number of times.

Proof: (1) is clearly implied by corollary 1. Here we consider only (2) and (3).

For the k-tuple $T = <t_1, t_2, \cdots, t_k> t_i = 1, \cdots, d$, we construct a walk $W = v_0, t_1, v_1, t_2, \cdots, v_{k-1}, t_k, v_k$ on the hypercube by following T as the sequence of coordinates. The choice of v_0 is arbitrary.

Let $t_i = p$. If the condition (2) above is true for T, then the block $t_{i+2}, \cdots, t_{i+m}$ (without $t_i = p$ and $t_{i+1} = q$) contains digit p an odd number of times and each other digit an even number of times. According to Corollary 6.2, v_{i+1} and v_{i+m} then are adjacent along pth coordinate. The situation is depicted in Figure 19. Thus, v_{i-1}, v_i and v_{i+1}, and v_{i+m} constitute the four vertices of a 2-D plane of the hypercube, and T can not be the transition sequence of a 1SRE.

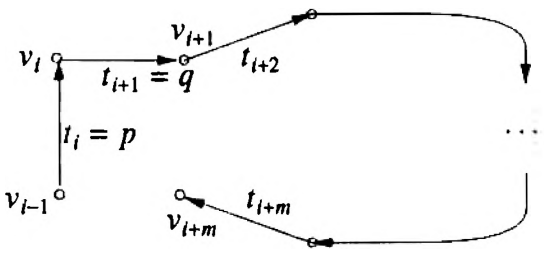


Figure 19. Some block $t_i, t_{i+1}, \cdots, t_{i+m}$ (m is even) where $t_{i+1} = q$ of the transition sequence of an embedding contains digit q an odd number of times and other digits even numbers of times.

Let $t_{i+m} = p$. If the condition (3) above is true for T, then the block $t_i, \cdots, t_{i+m-2}$ (without $t_{i+m-1} = q$ and $t_{i+m} = p$) contains digit p an odd number of times and each other digit an even number of times. According to Corollary 6.2, v_{i+m-2} and v_{i-1} then are adjacent along pth coordinate. The situation is depicted in Figure 20. Thus, v_{i-1}, v_i and v_{i+1}, and v_{i+m} constitute the four vertices of a 2-D plane of the hypercube, and T can not be the transition sequence of a 1SRE.

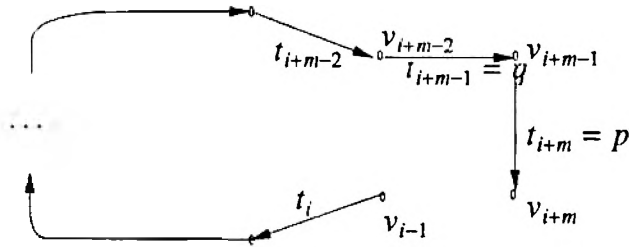


Figure 20. Some block $t_i, \cdots, t_{i+m-1}, t_{i+m}$ (m is even) where $t_{i+m-1} = q$ of the transition sequence of an embedding contains digit q an odd number of times and other digits even numbers of times.

If T is not the transition sequence of a 1SRE, then there must be some v_i on W which is not 1-step recoverable. That is, there exist some v_{x-1}, v_x, v_{x+1}, and v_y on W such that $v_y = \text{XOR}(v_{x-1}, v_x, v_{x+1})$, and v_{x-1}, v_x, v_{x+1}, and v_y constitute the four vertices of a 2-D plane of the hypercube.

Case (I): $x < y$ (i.e., $y = x + m, m > 0$). Let $x = i$ and $y = i + m$, and let $t_i = p$ and $t_{i+1} = q$. Since v_{i-1}, v_i, v_{i+1}, and v_{i+m} form a 2-D plane and v_{i-1} and v_i are adjacent along pth coordinate ($t_i = p$), v_{i+1} and v_{i+m} must be also adjacent along the pth coordinate. According to Corollary 6.2, the block $t_{i+2}, t_{i+3}, \cdots, t_{i+m}$ then contain digit p an odd number of times and each other digit an even numbers of times. Hence, the block $t_i, t_{i+1}, \cdots, t_{i+m}$ contain digit q an odd number of times and each other digit an even number of times. Note that, m is even since the number $(i + m) - i + 1 = m + 1$ is odd.

Case (II): $y < x$ (i.e., $x = y + m, m > 0$). Let $y = i - 1$ and $x = i + m - 1$, and let $t_{i+m} = p$ and $t_{i+m-1} = q$. Since v_{i+m-2}, v_{i+m-1}, v_{i+m}, and v_{i-1} form a 2-D plane and v_{i+m-1} and v_{i+m} are adjacent along pth coordinate ($t_{i+m} = p$), v_{i+m-2} and v_{i-1} must be also adjacent along the pth coordinate. According to Corollary 6.2, the block $t_i, t_{i+1}, \cdots, t_{i+m-2}$ then contain digit p an odd number of times and each other digit an even number of times. Hence, the block $t_i, t_{i+1}, \cdots, t_{i+m}$ contain digit q an odd number of times and each other digit an even number of times. Note that, m is even since the number $(i + m) - i + 1 = m + 1$ is odd. □

2. Change Numbers. Let T be a coordinate sequence and let N_k, $k = 1, \cdots, d$, be the number of appearances of the digit k in T. N_k will be called the kth *change number* of T. For instance, the transition sequence <1,2,3,1,2,4,1,2,3,1,2,4> has the change numbers 4,4,2,2. The change numbers of any other sequence T' of the same type as T are just a

rearrangement of $N_1, \cdots, N_d$. Hence a comparison of change numbers often suffices to prove two transition sequences to be of different type. There are, however, many examples of transition sequences of different type but having the same set of change numbers.

Note that, the transition sequences of embeddings of a specific length ring in a given size hypercube can be classified into groups based on their change numbers. Clearly, many of these sets of change numbers are not feasible for 1SRE's. By using the change numbers to systematically consider the possible transition sequences for 1SRE's, we can more easily show that there is no 1SRE's for some embedding cases. Let $P_e(N, M, n)$ denote the number of partitions of n into M parts, each of which is even and does not exceed N. It turns out the number of possibly feasible sets of change numbers for the transition sequences of all 1SRE's of a length-k ring in a d-cube is bounded by the number, $P_e(\lfloor k/3 \rfloor, d, k)$. However, there is no existing closed form for $P_e(N, M, n)$, and it seems to be very difficult to generate such a closed form [Andr76].

In counting partitions the order is unimportant, so there are three different partitions of 6 with 3 parts: 4+1+1, 3+2+1, and 2+2+2. On the other hand, in counting the number of solutions of an equation, order is important since the integers represent different variables. Therefore the number of positive integer solutions of $x_1 + x_2 + x_3 = 6$ is not 3, but $\binom{5}{2} = 10$. Let $S_e(N, M, n)$ denote the number of positive solutions of $x_1 + x_2 + \cdots + x_M$ = n where x_i is even and $\leq N$. Clearly $P_e(N, M, n) \leq S_e(N, M, n)$, so we can used the closed form for $S_e(N, M, n)$ to derive a asymptotic formula for $P_e(N, M, n)$.

Corollary 6.3: [JaTh90] The number of positive integer solutions of $x_1 + x_2 + \cdots + x_M = n$ is $\binom{n + (M-1)}{n}$.

Corollary 6.4: [JaTh90] The number of positive integer solutions of $x_1 + x_2 + \cdots + x_M = n$ where at least one $x_i \geq$ N is $\binom{n+(M-1)-N}{n-N} \cdot M$.

According to the previous definition, $S_e(N, M, n)$ is the number of positive even integer solutions of the equation

$$x_1 + x_2 + \cdots + x_M = n, \quad x_i \text{ is even}, 2 \leq x_i \leq N$$

$$\Rightarrow y_1 + y_2 + \cdots + y_M = \frac{n}{2}, \quad 1 \leq y_i \leq \frac{N}{2}$$

$$\Rightarrow z_1 + z_2 + \cdots + z_M = \frac{n}{2} - M, \quad 0 \leq z_i \leq \frac{N}{2} - 1.$$

With Corollary 6.3 and Corollary 6.4 we have

$$S_e(N, M, n) = \binom{\frac{n}{2} - M + (M-1)}{\frac{n}{2} - M} - \binom{\frac{n}{2} - M + (M-1) - (\frac{N}{2} - 1)}{\frac{n}{2} - M - (\frac{N}{2} - 1)} \cdot M$$

$$= \binom{\frac{n}{2} - 1}{\frac{n}{2} - M} - \binom{\frac{n}{2} - \frac{N}{2}}{\frac{n}{2} - M - (\frac{N}{2} - 1)} \cdot M$$

$$= \frac{(\frac{n}{2} - 1)!}{(\frac{n}{2} - M)!\,(M-1)!} - \frac{(\frac{n}{2} - \frac{N}{2})!}{(M-1)!\,((\frac{n}{2} - \frac{N}{2}) - (M-1))!} \cdot M$$

$$= \frac{(\frac{n}{2}-1)\cdots(\frac{n}{2}-(M-1))}{(M-1)!} - \frac{(\frac{n}{2}-\frac{N}{2})\cdots((\frac{n}{2}-\frac{N}{2})-(M-2))}{(M-1)!}\cdot M$$

$$= O((\frac{n}{2})^{M-1}) - O((\frac{n}{2}-\frac{N}{2})^{M-1})$$

$$= O(n^M)$$

Therefore we have $P_e(N, M, n) = O(n^M)$. So, given a length-k (even) ring and a d-cube, the number of sets of change numbers we need to consider is bounded by the number $P_e(\lfloor k/3 \rfloor, d, k) = P_e(\lfloor k/3 \rfloor, \lceil \log k \rceil, k) = O(k^{\lceil \log k \rceil})$.

In the following of this paper, when we consider a specific set of change numbers, we will then consider only one transition sequence for each type of embeddings.

3. Proof of the Non-Existence of 1SRE's. In this section, we will use the necessary and sufficient condition for a k-tuple not being a transition sequence of a 1SRE and the idea of change numbers to show that there is no 1SRE of a length-10 ring in a 4-cube and a length-22 ring in a 5-cube.

Theorem 3: There is no 1SRE of a length-10 ring in a 4-cube.

Proof: Consider any embedding of length l which contains no pair of active nodes which differ in all four coordinates. Complementing all four coordinate places changes this embedding into a new disjoint one. Then $2l \leq 16$. It follows that every embedding of a length-10 ring in a 4-cube contains such a pair of "diametrically opposite" nodes. The

embedding can be cut into two paths ,P_1 of length 4 and P_2 of length 6, joining these nodes.

Note that 4,2,2,2 is the only possible set of change numbers of a transition sequence for an embedding of a length-10 ring in a 4-cube. Hence, $T(P_1)$ and $T(P_2)$ must be π(1,2,3,4) and π(1,1,1,2,3,4), respectively. Let $T(P_1)$ be <1,2,3,4>, then the possible pattern for $T(P_2)$ is <1,_,_,1,_,1> where "_" represents digits 2,3, and 4. Clearly, $T(P_2)$ contains the block "1,_,1" which satisfies (2) or (3) of Theorem 6.3. □

Lemma 6.8: There is no 1SRE of a length-22 ring in a 5-cube with change numbers of its transition sequence being 6,6,6,2,2.

Proof: Assume that there is a transition sequence T_1 for a 1SRE of a length-22 ring in a 5-cube and T_1 has change numbers 6,6,6,2,2. According to the "structure" of the change numbers, the configuration of this 1SRE must be like the one depicted in Figure 21.

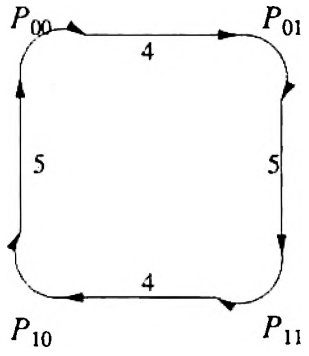


Figure 21. An configuration of an embedding with change numbers 6,6,6,2,2 for its transition sequence. P_{ij} i,j = 0,1, denotes the portion of the embedding resident in the subcube ij***.

Let P_{ij} defined as in Figure 21. Note that, if $|P_{ij}| \geq 6$ then some active nodes of the 1SRE in the subcube $ij***$ can not be recovered in one step. Since $|P_{00}|+|P_{01}|+|P_{11}|+|P_{10}|=18$, we have $3 \leq |P_{ij}| \leq 5$ and Assume that $|P_{00}|+|P_{01}| \geq |P_{11}|+|P_{10}|$.

Case (I): $|P_{00}|+|P_{01}|=10$. Thus, $|P_{00}|=|P_{01}|=5$. Let $T(P_{00})$ be <1,2,3,1,2>. Then, $T(P_{01})$ must also be <1,2,3,1,2> since $T(P_{01})$ cannot begin with digit 2 because of the resulting block "2,4,2" or digit 3 because of the resulting block "1,2,3,1,2,4,3". Thus T_1 can be

$$< 1, 2, 3, 1, 2, 4, 1, 2, 3, 1, 2, 5, 3, \_, \_, 3, \_, \_, 3, \_, 3, 5 >$$

where "_" represents some digit among 1,2,or 4. This T_1 can not be a transition sequence for a 1SRE because of the portion "3,_,3".

Case (II): $|P_{00}|+|P_{01}|=9$. Let $|P_{00}|=5$ and $|P_{01}|=4$, and let $T(P_{00})$ be <1,2,3,1,2>. Then, $T(P_{01})$ must be <1,2,3,1> for the same reason indicated in case (I). Thus, T_1 can be

$$< 1, 2, 3, 1, 2, 4, 1, 2, 3, 1, 5, 3, \_, \_, 3, \_, \_, 3, \_, \_, 3, 5 > .$$

This T_1 can not be a transition sequence for a 1SRE because of the portion "4,1,2,3,1,5,3, ...,3,5" consisting of digit 3 odd number of times and other digits all even numbers of times.

Therefore, we have a contradiction. □

Lemma 6.9: There is no 1SRE of a length-22 ring in a 5-cube with change numbers of its transition sequence being 6,6,4,4,2.

Proof: Assume that there is a transition sequence T_1 for a 1SRE of a length-22 ring in a 5-cube and T_1 has change numbers 6,6,4,4,2. According to the "structure" of the change numbers, the configuration of this 1SRE must be like the one depicted in Figure 22.

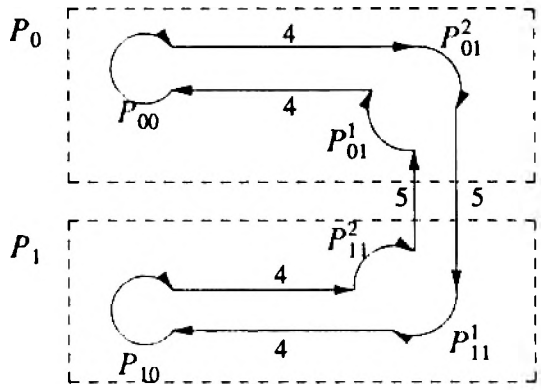


Figure 22. An configuration of an embedding with change numbers 6,6,4,4,2 for its transition sequence. P^l_{ij} i,j = 0,1 and l = 1,2, denotes the portion of the 1SRE resident in the subcube ij***.

Let P^l_{ij} i, j = 0,1 and l = 1,2 defined as in Figure 22. Note that, if $|P^1_{ij}|+|P^2_{ij}| \geq 5$, then some active node of the 1SRE in the subcube ij*** can not be recovered in one step. So, $|P^1_{ij}|+|P^2_{ij}| \leq 4$. The possible patterns for P^1_{ij} and P^2_{ij} where $|P^1_{ij}|+|P^2_{ij}| = 4$ are as follows.

- $P^1_{ij} = P^2_{ij} = \pi(1,2)$, $\pi(1,3)$, or $\pi(2,3)$
- $P^1_{ij} = \pi(1,2,3)$, and $P^2_{ij} = \pi(1)$, $\pi(2)$, or $\pi(3)$
- $P^1_{ij} = \pi(1,1,2,3)$, $\pi(1,2,2,3)$, or $\pi(1,2,3,3)$, and $P^2_{ij} = \varnothing$.

Let P_0 denotes the path starting from P^1_{01} and ending with P^2_{01}, and let P_1 denote the path starting from P^1_{11} and ending with P^2_{11} as shown in Figure 22. Since $|P^1_{01}|+|P^2_{01}| \leq 4$ and $|P_{00}| \leq 5$, we have $|P_0| \leq 11$. Similarly, $|P_1| \leq 11$. Without loss of generality, we assume that $|P_0| \geq |P_1|$.

Case (I): $|P_0| = 11$.

(i) $T(P_0)$ contains digit 1 odd number of times and other digits all even numbers of times. $T(P_0)$ and $T(P_1)$ then must be $\pi(1,1,1,2,2,2,2,3,3,4,4)$ and $\pi(1,1,1,2,2,3,3,4,4)$, respectively. The feasible $T(P_1)$ may be

$$< 1,\_,\_,1,\_,\_,1,\_,\_ >$$

$$< \_,1,\_,\_,1,\_,\_,1,\_ >$$

$$< \_,\_,1,\_,\_,1,\_,\_,1 > .$$

The first and third ones above contain the block "_,_,1,_,_,1,_,_" with digits 1, 2, 3, and 4 all even numbers of times. The second one with the block "_,1,_,_,1,_,_,1,_" satisfies (2) or (3) of Theorem 6.3.

(ii)$T(P_0)$ contains digit 3 odd number of times and other digits all even numbers of times. $T(P_0)$ and $T(P_1)$ then must be $\pi(1,1,1,1,2,2,3,3,3,4,4)$ and $\pi(1,1,2,2,2,2,3,4,4)$, respectively. The feasible $T(P_1)$ may be

$$< 1,\_,\_,1,\_,\_,1,\_,1 > ,$$

which consists the block "1,_,1" satisfying (2) or (3) of Theorem 6.3.

(iii)$T(P_0)$ contains digits 1, 2, and 3 all odd numbers of times. $T(P_0)$ and $T(P_1)$ then must be $\pi(1,1,1,2,2,2,3,3,3,4,4)$ and $\pi(1,1,1,2,2,2,3,4,4)$, respectively. The feasible $T(P_0)$ may be

$$< 3,2,4,1,2,3,1,2,4,1,3 >$$

$$< 4,1,2,3,1,2,4,3,1,2,3 > .$$

However, the first one above contains the block "3,2,4,1,2,3,1,2,4" and the second one contains the block "1,2,3,1,2,4,3", and both satisfy (3) of Theorem 6.3.

Case (II): $|P_0| = 10$.

(i) $T(P_0)$ contains digits 1 and 3 odd numbers of times and other digits even odd numbers of times. $T(P_0)$ and $T(P_1)$ then must be π(1,1,1,2,2,3,3,3,4,4) and π(1,1,1,2,2,2,2,3,4,4), respectively. The feasible $T(P_0)$ may be

$$< 4, 1, 3, 2, 1, 3, 4, 1, 3, 2 >$$

$$< 2, 1, 3, 4, 1, 3, 2, 1, 3, 4 > .$$

Other patterns like <3,4,1,3,2,1,3,4,1,2> (with the block "4,1,3,2,1,3,4,1,2"), <4,1,2,3,1,4,3,1,2,3> (with the block "4,1,2,3,1,4,3,1,2"), and <2,3,4,1,2,3,1,4,3,1> (with the block "2,3,4,1,2,3,1,4") which satisfy (1) or (2) of Theorem 2 need not be considered. The feasible $T(P_1)$ may be <2,_,_,2,_,_,2,_,_,2>, hence, the feasible T_1 may be

$$< 4, 1, 3, 2, 1, 3, 4, 1, 3, 2, 5, 2, \_, \_, 2, \_, \_, 2, \_, \_, 2, 5 >$$

$$< 2, 1, 3, 4, 1, 3, 2, 1, 3, 4, 5, 2, \_, \_, 2, \_, \_, 2, \_, \_, 2, 5 > .$$

The first one above contains the block "2,5,2" and the second one contains the block "4,1,3,2,1,3,4,5,2", thus, both satisfy (3) of Theorem 6.3.

(ii)$T(P_0)$ contains digits 1 and 2 odd numbers of times and other digits even numbers of times. Both $\tau(P_0)$ and $\tau(P_1)$ then must be π(1,1,1,2,2,2,3,3,4,4). The feasible $T(P_0)$ may be

$$< 4, 1, 2, 3, 1, 2, 4, 1, 2, 3 >$$

$$< 3, 1, 2, 4, 1, 2, 3, 1, 2, 4 >$$

Other patterns like <...,4,1,2,3,1,2,4,3,...> (satisfies (1) of Theorem 6.3), <...,4,1,2,3,1,2,4,1,3,...> (satisfies (2) or (3) of Theorem 6.3), <1,2,4,1,2,3,1,4,3,2> (containing the block "4,1,2,3,1,4,3,2"), <2,4,1,2,3,1,4,3,2,1> (containing the block "4,1,2,3,1,4,3,2"),

<4,1,2,3,1,4,2,1,3,2> (containing the block "4,1,2,3,1,4,2,1,3") need not be considered. Thus, the feasible T_1 may be

$$< 4, 1, 2, 3, 1, 2, 4, 1, 2, 3, 5, 4, 1, 2, 3, 1, 2, 4, 1, 2, 3, 5 >$$

$$< 4, 1, 2, 3, 1, 2, 4, 1, 2, 3, 5, 3, 1, 2, 4, 1, 2, 3, 1, 2, 4, 5 >$$

$$< 3, 1, 2, 4, 1, 2, 3, 1, 2, 4, 5, 3, 1, 2, 4, 1, 2, 3, 1, 2, 4, 5 > .$$

However, the first one above containing the block "3,5,4,1,2,3,1,2,4", the second one containing the block "3,5,3", and the third one containing the block "4,1,2,3,1,2,4,5,3" all satisfy (3) of Theorem 6.3.

Therefore, we have a contradiction. □

Lemma 6.10: There is no 1SRE of a length-22 ring in a 5-cube with change numbers of its transition sequence being 6,4,4,4,4.

Proof: Assume that there is a transition sequence T_1 for a 1SRE of a length-22 ring in a 5-cube and T_1 has change numbers 6,4,4,4,4. According to the "structure" of this change numbers, the configuration of this 1SRE must be like the one depicted in Figure 23(a). Since $|P_{01}^1|, |P_{01}^2| \geq 2$ and $|P_{01}^1| + |P_{01}^2| \leq 4$, we have $|P_{01}^1| = |P_{01}^2| = 2$.

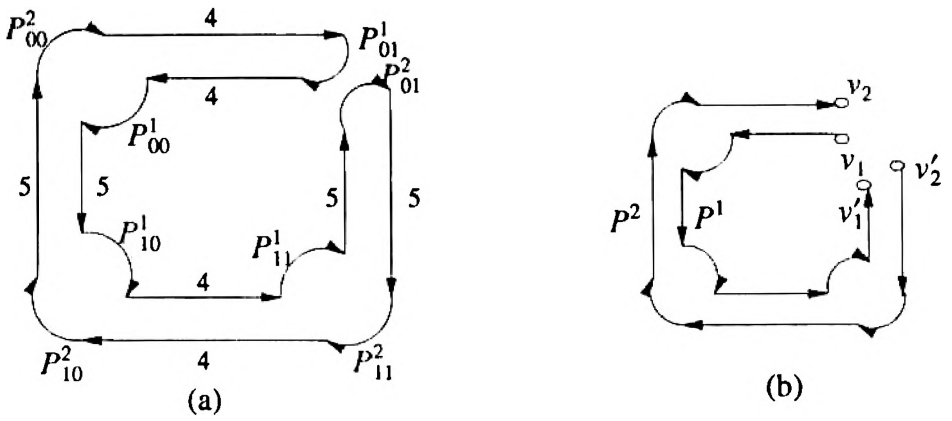


Figure 23. (a) An configuration of an embedding with change numbers 6,4,4,4,4 for its transition sequence. P^l_{ij} i, j = 0,1 and l = 1,2, denotes the portion of the 1SRE resident in the subcube ij***. (b) P^1 and P^2.

Let P^l_{ij} $i, j = 0,1$ and $l = 1,2$ defined as in Figure 23. We will only consider the case where $T(P^1_{01}) = T(P^2_{01})$ = <1,2>, and other cases can be easily proved by following the same technique used here.

Let v_1 and v_2 be two endpoints of P^1_{01} and v'_1 and v'_2 be two endpoints of P^2_{01}. Let P^1 and P^2 be the path from v_1 to v'_1 and the path form v'_2 to v_2, respectively. P^1 and P^2 are shown in Figure 23(b). Assume $|P^1| \geq |P^2|$.

(I) $\delta(v_1,v'_1) = \{3\}$. $T(P^1)$ then must contain digit 3 odd number of times and other digits even numbers of times. $T(P^2_{00})$ cannot end with digits 1 and 3 because of the resulting blocks "1,4,1" and "3,4,1,2, $T(P^1)$,1,2" (consisting all digits even numbers of times). $T(P^1_{00})$ cannot begin with digits 2 and 3 for the same reason. Moreover, the situations where $T(P^2_{00})$ ending with digit 2 and $T(P^1_{00})$ beginning with digit 1 cannot exist together because of the resulting block "2,4,1,2,4,1". Hence, the feasible $T(P^2_{00})$ may be <2> and <3,2>, and other

possibilities like <1,2>, <1,3,2>, or <_,1,3,2> all result some block of T_1 satisfies (2) of Theorem 6.3. Similarly, the feasible $T(P^1_{00})$ may be <1> and <1,3>. Thus, we have $|P^1_{00}|+|P^2_{00}| \le 2$ and $|P^1_{11}|+|P^2_{11}| \le 2$, and $\sum_{i=0}^{1}\sum_{j=0}^{1}|P^1_{ij}|+|P^2_{ij}| \le 12 \le$ 14.

(II) $\delta(v_1, v_1') = \{1,3\}$.

(a) $T^*(P^1_{00},P^1_{01},P^1_{11}) = \pi(1,1,1,2,2,3)$. Then, $T^*(P^2_{11},P^2_{10},P^2_{00}) = \pi(1,3,3,3)$, and some block (i.e., "3,4,3" or "3,5,3") of $T(P^2)$ must satisfy (2) of Theorem 6.3.

(b) $T^*(P^1_{00},P^1_{01},P^1_{11}) = \pi(1,1,1,3,3,3)$. Then, $T^*(P^2_{11},P^2_{10},P^2_{00}) = \pi(1,2,2,3)$. Thus, $T(P^1_{00}) = T(P^1_{01}) = T(P^1_{11}) = \pi(1,3)$, and at least one of $T(P^2_{11})$, $T(P^2_{10})$, and $T(P^2_{00})$ contains digit 2. By drawing in pictures we can tell that the embedding can not be a 1SRE if $T(P^1_{ij})$ is $\pi(1,3)$ and $T(P^2_{ij})$ contains digit 2.

(c) $T^*(P^1_{00}\ P^1_{01},P^1_{11}) = \pi(1,2,2,3,3,3)$. Then, $T^*(P^2_{11},P^2_{10},P^2_{00}) = \pi(1,1,1,3)$. This case has the similar situation as case (a).

(III) $\delta(v_1, v_1') = \{2,3\}$. Note that, $T(P^1_{11})$ cannot end with digits 3 and 1 because of the resulting blocks "2,$T(P^1)$" (consisting of digit 3 odd number of times and other digits ever numbers of times) and "1,5,1", respectively. $T(P^2_{11})$ cannot begin with digit 2. Moreover, if $T(P^2_{11})$ begins with digit 1, then $|P^1_{11}|$ will be forced to be zero. The feasible $T(P^1_{11})$ and $T(P^2_{11})$ then may be

. $T(P^2_{11})$ = <3,2> and $T(P^1_{11})$ = <3,2>, <2>, or <>

. $T(P^2_{11})$ = <3,2,1> and $T(P^1_{11})$ = <2>, or <>

. $T(P^2_{11})$ = <3,1,2> and $T(P^1_{11})$ = <>

. $T(P_{11}^2) = <3>$ and $T(P_{11}^1) = <3,2>, <2>$.

Similarly, the feasible $T(P_{00}^1)$ and $T(P_{00}^2)$ may be

. $T(P_{00}^1) = <3,2>$ and $T(P_{00}^2) = <3,2>, <2>$, or $<>$

. $T(P_{00}^1) = <3,2,1>$ and $T(P_{00}^2) = <2>$, or $<>$

. $T(P_{00}^1) = <3,1,2>$ and $T(P_{00}^2) = <>$

. $T(P_{00}^1) = <3>$ and $T(P_{00}^2) = <3,2>, <2>$.

(a) $\sum_{i=1}^{2}(|P_{00}^i|+|P_{11}^i|) = 8$. Then, digit 2 will have to appear six times in T_1.

(b) $\sum_{i=1}^{2}(|P_{00}^i|+|P_{11}^i|) = 7$. Then, digit 2 will have to appear five or six times in T_1.

(c) $\sum_{i=1}^{2}(|P_{00}^i|+|P_{11}^i|) = 6$. Then, the only possible situation is that $T(P_{11}^2) = T(P_{00}^1)$ $= \pi(1,2,3)$ and $T(P_{00}^2) = T(P_{11}^1) = <>$. Since $T(P^1)$ contains odd numbers of digits 2 and 3, $T(P_{10}^1)$ must be $\pi(1,1)$, $\pi(3,3)$, or $\pi(1,1,3,3)$. Clearly, the transition sequence of P_{10}^1 satisfies (1) of Theorem 6.3.

(d) $\sum_{i=1}^{2}(|P_{00}^i|+|P_{11}^i|) = 5$. Then, $\sum_{i=0}^{1}\sum_{j=0}^{1}|P_{ij}^1|+|P_{ij}^2| \le 13 \le 14$.

(IV) $\delta(v_1, v_1') = \{1,2,3\}$. The situation is similar to (I). That is, $|P_{00}^1|+|P_{00}^2| \le 2$ and $|P_{11}^1|+|P_{11}^2| \le 2$, and $\sum_{i=0}^{1}\sum_{j=0}^{1}|P_{ij}^1|+|P_{ij}^2| \le 12 \le 14$.

Therefore, we have a contradiction. □

Theorem 6.4: There is no 1SRE of a length-22 ring in a 5-cube.

Proof: Clearly, the change numbers with numbers ≥ 8 can not be the change numbers of the transition sequence for a 1SRE of a length-22 ring in a 5-cube. Hence, the only three possibilities of the change numbers are 6,6,6,2,2, 6,6,4,4,2, and 6,4,4,4,4. However, according to Lemma 6.8, Lemma 6.9, and Lemma 6.10, there is no transition sequences of a 1SRE of a length-22 ring in a 5-cube with these three sets of numbers, there is no 1SRE of a length-22 ring in a 5-cube. □

D. Concluding Remarks

We present a composition scheme to systematically construct 1SRE's of length-k (even) rings in d-cubes, where $6 \leq k \leq (3/4)2^d$ and $d \geq 3$. The composition method builds up a larger 1SRE by combining two 1SRE's in two lower dimension hypercubes. The scheme is base on the proved theorem: If there exists a 1SRE of a length-k_1 ring in a d-cube and a 1SRE of a length-k_2 ring in a d-cube, then there exists a 1SRE of length-$(k_1 + k_2)$ ring in a $(d+1)$-cube. So, we can systematically build up large 1SRE's by combining two one-dimension smaller 1SRE's. build up large 1SRE's

Compared to other schemes for constructing 1SRE's which are either applicable to rings of limited lengths [ChLT88b,PrMe88] or resulting in large expansion of their embeddings [Leet90], our scheme has more applicability to rings of different lengths, and the resulting 1SRE's are of small expansion.

VII. A SUFFICIENT CONDITION FOR THE NON-EXISTENCE OF 1-STEP RECOVERABLE EMBEDDINGS

In this section, we discuss a sufficient condition for the non-existence of a 1SRE of a length-k (even) ring in a d-cube where $k > (3/4)2^d$ and $d \geq 3$. The sufficient condition is based on the idea: no spare node can serve more than $d-1$ active nodes when $k > (3/4)2^d$.

Definition : Let a_i, a_{i+1}, and a_{i+2} be three hypercube nodes with consecutive active states on a 1SRE, and s be the label of a spare node where $s = \text{XOR}(a_i, a_{i+1}, a_{i+2})$.

(1) The spare node s will be referred as the *1-step recovery spare node* for active node a_{i+1} in the 1SRE.

(2) Nodes s, a_i, a_{i+1}, and a_{i+2} will constitute the four nodes of a 2-D plane of the hypercube. This 2-D plane will be referred as a *1-step recovery hyperplane* (1SRHP) with respect to s, and denoted specifically as $[s, a_i, a_{i+1}, a_{i+2}]$

Lemma 7.1: For a 1SRE, it is impossible to have more than two 1SRHP's (with respect to the same spare node) that intersect on a hypercube link with the spare node as one end-point.

Proof : Suppose there exists a valid 1SRE where three 1SRHP's with respect to the same spare node intersect on a link with the spare node as one endpoint. Figure 24 shows such a situation where three 1SRHP's, $[s, a_{i_1}, a_{i_2}, a_{i_3},]$, $[s, a_{i_3}, a_{i_6}, a_{i_7}]$, and $[s, a_{i_3}, a_{i_4}, a_{i_5}]$, intersecting on the hypercube link (s, a_{i_3}). By definition of 1SRHP, the three hypercube links, (a_{i_2}, a_{i_3}), (a_{i_3}, a_{i_4}), and (a_{i_3}, a_{i_6}), must all be the image links of the embedding.

This is a contradiction. □

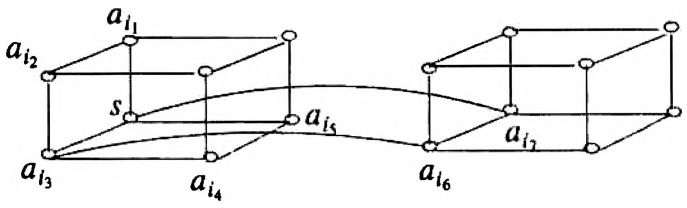


Figure 24. Three 1SRHP's with respect to s intersect on the edge (s, a_{i_3}).

Theorem 7.1: For a 1SRE in a d-cube, each spare node can serve as a 1-step recovery spare node for at most d active nodes.

Proof : According to Lemma 7.1, the best recovery case for a spare node is that each hypercube link incident to it is contained in exactly two 1SRHP's with respect to it. Such a situation is depicted in Figure 25. □

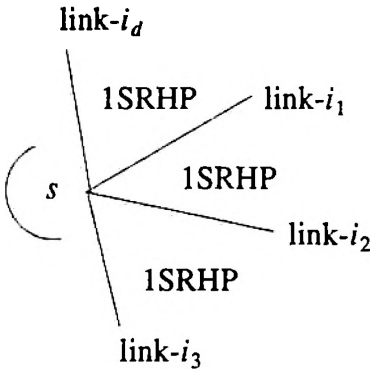


Figure 25. Each hypercube link incident to the spare node s is contained in exactly two 1SRHP's with respect to s.

Lemma 7.2: If $[s, a_{i_1}, a_{i_2}, a_{i_3}]$ and $[s, a_{i_3}, a_{i_4}, a_{i_5}]$ are two 1SRHP's with respect to spare node s in a 1SRE, then $a_{i_1}, a_{i_2}, a_{i_3}, a_{i_4}, a_{i_5}$ or $a_{i_5}, a_{i_4}, a_{i_3}, a_{i_2}, a_{i_1}$ must be contained in the active node listing (in order) of the 1SRE.

Proof: According to the definition of 1SRHP, $[s, a_{i_1}, a_{i_2}, a_{i_3}]$ is a 1SRHP with respect to s if $a_{i_1}, a_{i_2}, a_{i_3}$ or $a_{i_3}, a_{i_2}, a_{i_1}$ is contained in the active node listing of the 1SRE. Similarly, $a_{i_3}, a_{i_4}, a_{i_5}$ or $a_{i_5}, a_{i_4}, a_{i_3}$ must also be contained in the active node listing of the 1SRE. Thus, by combination, $a_{i_1}, a_{i_2}, \cdots, a_{i_5}$ or $a_{i_5}, a_{i_4}, \cdots, a_{i_1}$ must be contained in the active node listing of the 1SRE. □

Lemma 7.3: For a 1SRE of a length-k ring in a d-cube, if a spare node serves for d active nodes as a 1-step recovery spare node, then $k = 2d$.

Proof: From Fig.23, we can see that if a spare node serves for d active node as 1-step recovery spare node, then each hypercube link incident to this spare node must be contained in exactly two 1SRHP's with respect to this spare node. Such a situation is depicted in Figure 26 where s is a spare node, n_i (in active state) are neighbors to s, and a_i are active nodes which use s as 1-step recovery spare node. Then, according to Lemma 7.2,

$$n_1, a_1, n_2, a_2, n_3,$$

$$n_2, a_2, n_3, a_3, n_4,$$

$$\cdots$$

$$n_{d-1}, a_{d-1}, n_d, a_d, n_1$$

must be all contained in the active node listing in order of the 1SRE. Since the combination of these sublists forms a cycle of active nodes (i,e, $n_1, a_1, \cdots, n_d, a_d, n_1$), the length of this cycle must be equal to the length of the ring embedded. That is, $k = 2d$. □

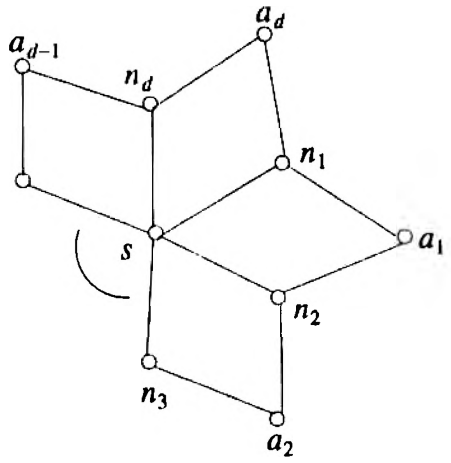


Figure 26. A spare node s is used for d active nodes as 1-step recovery spare node.

Theorem 7.2: For a 1SRE of a length-k ring in a d-cube where $k > (3/4)2^d$, each spare node can serve as a 1-step recovery spare node for at most $d-1$ active nodes.

Proof: Since $(3/4)2^d \geq 2d$ for $d \geq 3$, according to Theorem 7.1 and Lemma 7.3 to embed a length-k ring in a d-cube as a 1SRE where $k > (3/4)2^d$, a spare node can serve for at most $d-1$ active nodes as 1-step recovery spare node. □

Theorem 7.3: There is no 1SRE of a length-$((3/4)2^d + q)$ (even) ring in a d-cube, if $\frac{2^d}{4} - \frac{2^d}{d} < q$.

Proof : As a consequence of Theorem 7.2, if the number of the total number of spare nodes multiplying $d-1$ is less than the total number of active nodes, then some active nodes will not be able to be recovered in one step. That is, if the following inequality is satisfied,

$$(\frac{1}{4}2^d - q) \times (d-1) < \frac{3}{4}2^d + q$$

$$\Rightarrow \frac{d}{4}2^d - \frac{1}{4}2^d - qd + q < \frac{3}{4}2^d + q$$

$$\Rightarrow \frac{2^d}{4} - \frac{2^d}{d} < q,$$

the embedding is not a 1SRE. □

By Theorem 7.3, we have the summary of those lengths for rings having no 1SRE's in a d-cube in Table XI.

Table XI. Rings of length > $(3/4)2^d$ and < 2^d having no 1SRE's in a d-cube

d	$\frac{3}{4}2^d$	$\frac{2^d}{4} - \frac{2^d}{d}$	Lengths of rings having no 1SRE's in a d-cube
4	12	0	14
5	24	1.6	26, 28, 30
6	48	5,3	54, 56, 58, 60, 62
7	96	13.9	110, 112, 114, ..., 126
...			

A sufficient condition for the non-existence of 1SRE's for rings of length > $(3/4)2^d$ in a d-cube is also addressed in this chapter. Clearly there exists a gap between $(3/4)2^d$ and the lower bound of lengths for rings having no 1SRE's in a d-cube, $d \geq 6$ (Table IV), and the size of the gap increases as d increases. It remains an open problem to diminish the size of the gap by either improving the lower bound for non-existence of 1SRE's or

extending the $(3/4)2^d$ bound for the existence of 1SRE's. Experimentally, we have determined there exist 1SRE's of rings of length $> (3/4)2^d$ in large d-cubes, however, there seems to be no algorithmic technique to construct those 1SRE's.

VIII. APPLICATION TO THE PROTEIN SEQUENCE COMPARISON

In this chapter, we discuss applying our reconfigurable embedding schemes to an active research problem in molecular biology, the protein sequence comparison, which is applied by modern biologists to attempt to understand the semantics of DNA and the function and structure of proteins. We discuss several possible ways to extend the existing parallel protein sequence comparison system [LaMe88] to a fault-tolerant one. For the fault-tolerant system, if one processor becomes faulty during the computation, the entire system can still function correctly. We compare the performances, in terms of fault-tolerant system running time, of different fault-tolerant approaches. The simulation results show that in most cases the systems with our fault-tolerant embedding schemes (for 2SRE's and 1SRE's) have better performance than those with other fault-tolerant approaches. The results show a good validation of our embedding methods.

A. Problem of Protein Sequence Comparison

One of the central questions of molecular biology is the discovery of the semantics of DNA. Just knowing the syntax, that is, the sequence, tells the biologist little. The biologist must understand the biochemical functions of the DNA. To understand the semantics, one needs to know the relationship between DNA and proteins. Proteins are sequences made from 20 different amino acids. A piece of DNA can be encode for a protein. Since proteins are responsible for important biochemical functions within a living cell, a fundamental question of the biologist is to determine what proteins do.

There are many approaches one could take to uncover the semantics of proteins. An approach that has proved to be both tractable and valuable is searching for sequence homologies or similarities. This approach exploits the mechanism of evolution as well as the underlying physical laws for atomic interactions. Thus a central computational problem in biology is the attempt to discover common structure among sets of sequences: given a large collection of DNA or proteins determine those sequences that are close in the sense of evolution, and further determine their common structure. From this approach, research can begin to make important strides into understanding the semantics of DNA and the function and structure of proteins [LiMW89]. One of the first successes for this approach was the discovery of similarity between some cancer-causing genes and proteins such as human growth factors.

An aspect of the search for evolutionary structure of DNA and proteins is the immense size of the problems. As we will soon see, the basic computations involved in comparing even two sequences are quite non-trivial. However, the amount of biological material that must eventually be searched is very large. The essence of the problem is that given a set of proteins sequences, efficient alignment-matching algorithms are needed that can deal elegantly with insertion, deletion, substitution, and even gaps in the series of sequence elements. One way of measuring the optimality of an alignment is by computing a score based on a matrix of weights reflecting the similarity between pairs of sequences. In some situations a penalty is subtracted for each gap introduced. Such a score can be computed by a dynamic programming algorithm in time proportional to the product of the lengths of the sequences.

B. The Dynamic Programming

The subsequence matching problem can be formulated as follows:

Given two sequences A, B, of symbols chosen from a same domain

$$A = (a_1, a_2, \ldots, a_n),\ B = (b_1, b_2, \ldots, b_m),$$

find the subsequences

$$A' = (a_{i_1}, a_{i_2}, \ldots, a_{i_x}),\ B' = (b_{j_1}, b_{j_2}, \ldots, b_{j_x})$$

$$\textit{where } 1 \le i_1 < i_2 < \cdots < i_x \le n,\ 1 \le j_1 < j_2 < \cdots < j_x \le m$$

which maximize the comparison function $C(A', B')$. C can depend on the symbols a_{i_l}, b_{j_k} in A' and B' and on the numbers of symbols in A and B which are omitted between successive symbols in A' and B' (gaps).

For such comparison functions, one can use a dynamic programming algorithm to determine the best subsequence match for a given pair of sequences A, B in serial time O(mn) where n and m are the length of the sequences A and B. This dynamic programming algorithm can best be understood by considering the matrix

$$C_{r,s} = \max \begin{cases} 0 \\ C_{r-1,s-1} + D(a_r, b_s) \\ C_{r-1,s} + g \\ C_{r,s-1} + g \end{cases}$$

where the gap constant $g < 0$, and D is a correlation function between single elements [LaMe88].

A parallel version of the dynamic programming algorithm is quite straightforward to derive [EdWa87]. Since computing the value of $C_{r,s}$ only depends on knowing the values of $C_{r-1,s}$, $C_{r,s-1}$, and $C_{r-1,s-1}$, we see that all of the elements on one anti-diagonal of the matrix can be computed simultaneously if the values along the two previous anti-diagonals are known. That is, for a fixed value of t, the matrix elements $C_{t-s,s}$ can be computed simultaneously for all s provided that they are known for $t-1$ and $t-2$. Thus, one can parallelize the above algorithm by computing successive anti-diagonals of the matrix $C_{r,s}$ on successive time steps. This is represented schematically in Figure 27. The algorithm requires $n+m-1$ time steps and m processors to compare proteins of length m and n.

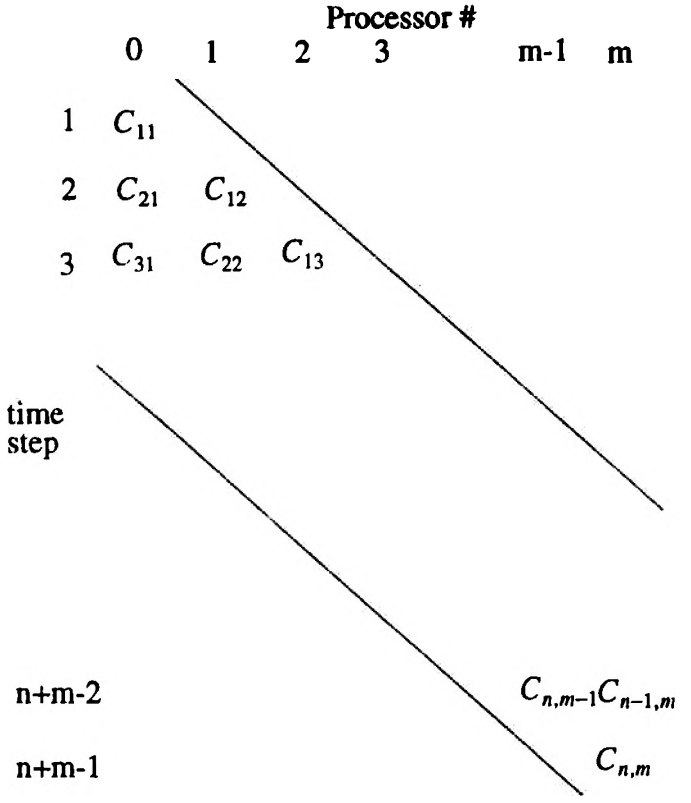


Figure 27. Diagram indicating activity of processor i at time step t. If $1 \leq t - p \leq n$, then processor i computes $C_{t-p,p+1}$ at step t. Otherwise, the processor is inactive.

C. Implementation of Parallel Protein Sequence Comparison

A parallel version of protein sequence comparison based on the idea of dynamic programming was implemented on the data parallel CM-2 Connection Machine [LaMe88]. We will briefly describe the implementation in this section. Later (in the next section), we will use this implementation as a base to extend it to fault-tolerant versions of parallel protein sequence comparison system.

The implementation of the parallel protein sequence comparison in [LaMe88] is summarized as follows. Since the CM-2 has far more processors than most proteins have amino acids, it is possible to perform the dynamic programming algorithm on many proteins at the same time. To this end, the proteins in the chosen data base are first sorted by length and then partitioned sequentially into set S_d such that the total number of amino acids in the proteins of each S_d is as large as possible but is less than or equal to the total number of processors available in the CM-2. The proteins in each set S_d are then compared in parallel to the entire database, one protein at a time. If M is the total number of such sets S_d, and K is the total number of proteins in the database, then the basic structure of the serial part of the program to compare all pairs of proteins is:

for d = 1 to M **do**
  set up S_d on *CM-2*
  for k = 1 to K do
    compare in parallel all proteins in S_d *to* P_k
    retrieve results and store on front end

where setting up S_d involves laying out the amino acids in the proteins in S_d in linear fashion into the memory of the processors with the CM-2 configured as a linear array (each processor holds one amino acid, we designate this acid as "source acid"). The inner loop which compares S_d to P_k is implemented using the parallelized dynamic programming algorithm described in previous section. Thus, on each pass through the inner loop, all the proteins in S_d will be compared simultaneously with some protein P_k which we will designate henceforth as a "target protein".

The code for the inner loop now proceeds as follows:

```
{initialization}
  Left[i] ← 0
  Diag[i] ← 0
  Now[i] ← 0
  Cmax[i] ← 0
  Tgt[i] ← 0
  H[i] ← 0

{compute score matrix C}
  for t = 1 to T(d, P_k) do begin
    Tgt[i] ← Tgt[i − 1]
    if H[i] = 1 then Tgt[i] ← P_k(t)
    Diag[i] ← Left[i]
    Left[i] ← Now[i − 1]
    Now[i] ← max{ 0, Left[i]+g, Now[i]+g, Diag[i]+D(SD[i], Tgt[i]) }
    Cmax[i] ← max{ Cmax[i], Now[i] }
    Backup[t] ← Now[i]
  end { for }
  Cmax[] ← max-scan( Cmax[]), with segmentation H[]
```

Algorithm notes:

1. The variables, *Now*, *Left*, and *Diag*, are used to compute score matrix *C*. The relative positions for these three variables in the matrix *C* is shown in Figure 28.

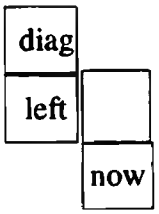


Figure 28. The relative positions for variable *Now*, *Left*, and *Diag* in the matrix *C*.

2. The amino acids from the target protein will be sent from the front end to the Connection Machine memories one per time step. They will shift across a variable *Tgt* (as pipeline form), and thus be aligned with the amino acids sequences held in the *SD* variable, so that the comparison can be performed.

3. $T(d, P_k)$ is the time needed to compare S_d to a target protein P_k, which is equal to the length of the longest protein in S_d plus the length of P_k.

4. The variable H is used to mark the beginning of each protein in a S_d. $H(i)$ is equal to 1 if processor i contain the first amino acid of a protein in S_d, and 0 otherwise. The variable *SD* holds a source amino acid.

5. All assignment statements are assumed to be carries out in all processors, unless otherwise indicated.

6. Assignment statements involving data in different processors are accomplished using linear array communication.

7. Assume that $P_k(t) = 0$, for $t > l(P_k)$.

8. The evaluation of the function $D(SD[i], Tgt[i])$ was implemented using an indirect table lookup, where the 5-bit byte $SD[i]$ and $Tgt[i]$ are concatenated into a 10-bit

word used as an offset into a locally stored table. Setting $D(0, a) = D(a, 0) = 0$ for all a ensured that the values of C computed in processor during "inactive" time steps of were less than the maximum valid of C computed in the same processor during the same comparison. Thus, it was not necessary to explicitly determine which processors were active at each time step.

D. Several Fault-Tolerant Approaches

Before we discuss several fault-tolerant approaches to the system of parallel protein sequence comparison, we discuss how to design a model of the total running time of the entire system in order to compare the performance of different fault-tolerant approaches. We will make use of the following variables in the model:

- K is the total number of proteins in the data base.
- M is the total number of such sets S_d.
- L is the length of a protein. Here, we assume every protein in the data base has the same length.
- N is the total number of processors ($N = 2^p$, p is a positive integer), and let $f(N)$ be the number of *active* processors involved in the computation.
- W, the work-load of each processor, is equal to the number of source amino acids held by each active processor. We assume that every active processor hold the same number of source amino acids.
- Let the time for a processor executing one iteration of the innermost loop (i.e., for t = 1 to $T(d, P_k)$ do begin ...) at the condition of $W = 1$ be $T_{comm} + T_{comp}$, where

T_{comm} and T_{comp} are the communication time and computation time needed, respectively.

- Let the time of completing one reconfiguration step be T_{1-rcfg}.

With the implementation of the system of parallel protein sequence comparison and the variables defined above, our model of the running time of the entire fault-tolerant system is described as follows:

- The size of the set S_d is $\left\lfloor \frac{W * f(N)}{L} \right\rfloor$. That is, such many proteins will be compared to one target protein simultaneously.

- M, the total number of sets S_d, is $\left\lceil \frac{K}{\left\lfloor \frac{W * f(N)}{L} \right\rfloor} \right\rceil$.

- The time for comparing proteins in a set S_d to a target protein is

$$(T_{comm} + W * T_{comp}) * (L + L - 1)$$

- The total running time for the parallel protein sequence comparison system in a fault-free situation is about

$$T_{FF} = (T_{comm} + W * T_{comp}) * (2L - 1) * K * \left\lceil \frac{K}{\left\lfloor \frac{W * f(N)}{L} \right\rfloor} \right\rceil$$

- The total running time for a fault-tolerant version of the protein sequence comparison system (tolerate one fault) is about

$$T_{1-FT} = g(T_{FF}) + c * T_{1-rcfg}$$

where g is a function on T_{FF} which depends on what kind of fault-tolerant approach applied, and c is a constant.

We assume that the communication model in a hypercube is circuit switching. For this model, a hardware communication circuit between two communicating nodes must be established before communication begins, and a link of the circuit is released at a time after the last bit of the message is transmitted. We, therefore, define the communication time needed for two communicating nodes in a hypercube as follows,

$$t_{comm} = t_{cong} + t_{hops}$$

$$= t_{cong} + [\tau_s + (h+1)\tau'_p + \tau_t S]$$

where t_{comm} is the time needed to send an S-byte message from one node to another. For the circuit switching model, if a circuit cannot be established because a desired link is being used by other packets, the circuit is said to be blocked. Here we assume that when a circuit is blocked, the partial circuit may be torn down, with establishment to be attempted later. t_{cong} here denotes the waiting time for reestablishment. Note that, if the mapping of the linear array in a hypercube is dilation-1, then it will be congestion-1 also and no edges of a hypercube will be contained in more than one mapping linear array edge. That is, if the mapping is dilation-1, t_{cong}, the communication delay due to congestion, will be zero. t_{hops} is the ideal communication time between two communicating nodes such that the edge congestion of the desired circuit between these two nodes are all one. The value of t_{hops} is determined by the five terms: τ_s, h, τ_p', τ_t, and S, where τ_s is the communication latency, h is the hops between the two communicating nodes, τ_p' is the processing time required at a router, τ_t is the time needed to transmit one byte of data, and S is the size (in bytes) of message sent. According to the algorithm of parallel

protein sequence comparison, each processor in the linear array will send messages to its right neighbor twice; therefore, $T_{comm} = 2 * t_{comm} = 2 * (t_{cong} + t_{hops})$.

Here, we compare the total running time for five different fault-tolerant versions of parallel protein sequence comparison system. These five different fault-tolerant approaches are:

1. **Remapping & Rerunning**: When a fault occurs, simply reconstruct the linear array (remain dilation-1) and rerun the entire program again. On average this approach will take about one and a half times of the fault-free running time. The total running time for this approach is about,

$$t_{cong} = 0$$

$$t_{hops} = \tau_s + (1 + 1)\tau'_p + \tau_t S$$

$$T_{comm} = 2 * (t_{cong} + t_{hops})$$

$$T_{1-FT} = 1.5 * [\ (T_{comm} + W * T_{comp}) * (2L-1) * K * \left\lceil \frac{K}{\left\lfloor \frac{W*(N-1)}{L} \right\rfloor} \right\rceil\ ] + T_{1-rcfg}$$

2. **Graceful Degradation**: Let all the nodes in a hypercube do the computation. If a fault occurs, simply assign the faulty node's work to its neighbor on its right and continue the processing. In this situation, the computation load of this right neighbor will be increased double, while its communication load will be remain the same. It is possible that the desired path from the faulty node's left neighbor to the faulty node's right neighbor be congestion-2 (then $t_{cong} \neq 0$); however, here we assume that this path has congestion-1. The total running time with this approach is about,

$$t_{cong} = 0$$

$$t_{hops} = \tau_s + (1+1)\tau'_p + \tau_t S$$

$$T_{comm} = 2 * (t_{cong} + t_{hops})$$

$$T_{1-FT} = (T_{comm} + 2 * W * T_{comp}) * (2L-1) * K * \left\lceil \frac{K}{\left\lfloor \frac{W*(N-1)}{L} \right\rfloor} \right\rceil + T_{1-rcfg}$$

3. **One Designated Spare Node**: In the beginning, select one designated spare node and let the rest of nodes all do the computation. If a node becomes faulty during processing, just replace this faulty node with this designated spare node. For this approach, it is *very* possible that the length (or hops) of the desired path from the left or right neighbor of the faulty node to the designated spare node is equal to the dimension of the embedding hypercube, and, moreover, the desired path is

congestion-2. These factors (number of hops and congestion) have to take into account for calculating the communication time. From the algorithm of parallel protein sequence comparison, we can derive that t_{cong} is equal to $\tau_s + (1+1)\ \tau'_p + \tau_t S$, which is the time for sending S-byte message from one node to its neighbor, and the ideal elapsed time of a link from being used to being released. For simplicity, we also assume that the path from the faulty node's left neighbor to the designated spare node and the path from the designated spare node to the faulty node's right neighbor are edge-disjoint. The total running time for this approach is about,

$$t_{cong} = \tau_s + (1+1)\tau'_p + \tau_t S$$

$$t_{hops} = \tau_s + (\log_2^N + 1)\tau'_p + \tau_t S$$

$$T_{comm} = 2 * (t_{cong} + t_{hops})$$

$$T_{1-FT} = (T_{comm} + W * T_{comp}) * (2L-1) * K * \left\lceil \frac{K}{\left\lfloor \frac{W * (N-1)}{L} \right\rfloor} \right\rceil + T_{1-rcfg}$$

4. **1-SRE**: Using the 1-step recoverable embedding scheme to construct the linear array, then running the program on it. In this case, three quarters of the nodes in a hypercube will be involved in the computation, and any one single fault can be recovered within one reconfiguration step. The total running time for this approach is about,

$$t_{cong} = 0$$

$$t_{hops} = \tau_s + (1+1)\tau'_p + \tau_t S$$

$$T_{comm} = 2 * (t_{cong} + t_{hops})$$

$$T_{1-FT} = (T_{comm} + W * T_{comp}) * (2L-1) * K * \left\lceil \frac{K}{\left\lfloor \frac{W * \frac{3}{4} N}{L} \right\rfloor} \right\rceil + T_{1-rcfg}$$

5. **2-SRE**: Using our 2-step recoverable embedding scheme to construct the linear array, then running the program on it. In this case, seven eights of the nodes in a hypercube will be involved in the computation, and any one single fault can be recovered within 2 reconfiguration steps. The total running time for this approach is about,

$$t_{cong} = 0$$

$$t_{hops} = \tau_s + (1+1)\tau'_p + \tau_t S$$

$$T_{comm} = 2 * (t_{cong} + t_{hops})$$

$$T_{1-FT} = (T_{comm} + W * T_{comp}) * (2L-1) * K * \left\lceil \frac{K}{\left\lfloor \frac{W * \frac{7}{8} N}{L} \right\rfloor} \right\rceil + 2 * T_{1-rcfg}$$

Along with the communication model assumption (i.e., circuit switch), the parameters that we choose for comparing the five different fault-tolerant approaches are described as follows.

The dimension of the hypercube is 16

. K = 5000 (proteins)

. L = 200 (amino acids)

. τ_s = 233 (μ sec)

. τ_p' = 3.4 (μ sec)

. τ_t = 0.38 (μ sec)

. S = 4 (bytes)

. T_{comp} is about 50 (μ sec)

. T_{1-rcfg} is about 600 (μ sec)

The comparison results for five different fault-tolerant approaches with W, the work load of each processor, equal to 1, 2, 3, and 4, are given in Table XII, XIII, XIV, and XV, respectively.

Table XII. The total running time for five different fault-tolerant systems of parallel protein sequence comparison with W = 1.

Fault-Tolerant Approach	Total Running Time (sec)
Remapping & Rerunning	25502.804
Graceful Degradation	18597.870
One Designated Spare Node	35663.578
1-SRE	22314.954
2-SRE	19127.104

Table XIII. The total running time for five different fault-tolerant systems of parallel protein sequence comparison with $W = 2$.

Fault-Tolerant Approach	Total Running Time (sec)
Remapping & Rerunning	13948.402
Graceful Degradation	10894.935
One Designated Spare Node	18629.790
1-SRE	12786.036
2-SRE	10461.303

Table XIV. The total running time for five different fault-tolerant systems of parallel protein sequence comparison with $W = 3$.

Fault-Tolerant Approach	Total Running Time (sec)
Remapping & Rerunning	11359.051
Graceful Degradation	9368.202
One Designated Spare Node	14570.842
1-SRE	8834.819
2-SRE	7572.703

Table XV. The total running time of five different fault-tolerant systems of parallel protein sequence comparison with $W = 4$.

Fault-Tolerant Approach	Total Running Time (sec)
Remapping & Rerunning	8171.202
Graceful Degradation	7043.468
One Designated Spare Node	10112.895
1-SRE	6809.335
2-SRE	6809.364

From these four tables, we can see that our schemes (1-SRE and 2-SRE) will always outperform the fault-tolerant approaches, Remapping & Rerunning and One Designated Spare Node. Note that, for $W = 1$ and $W = 2$, the Graceful Degradation has better performance than our schemes. This is because in the original algorithm of the parallel protein sequence comparison, the computation time needed (i.e., 50 μsec) in the inner most loop is much smaller than the communication time needed (i.e., 241.32 μsec) in the same loop. When $W = 1$ and 2, the communication time still much dominate the total time needed for the inner most loop. However, when the value of W increases, the domination situation changes, and the computation time becomes the more important factor for deciding the total time of the inner most loop. We claim that for W is large, our schemes will have better performance than the Graceful Degradation. The following theorem supports our claim.

Theorem 8.1: When W, the work load of each processor, increases, our schemes (1-SRE and 2-SRE) tend to have better performance than the Graceful Degradation. That is, the total running time of the fault-tolerant parallel protein sequence comparison system with our approaches will take less time than that with Graceful Degradation.

Proof: We consider the ratio of the total running time of the fault-tolerant system with Graceful Degradation to that with our 1-SRE approach, when $W \rightarrow \infty$.

Note that,

$$\lim_{W\to\infty} \left\lceil \frac{K}{\left\lfloor \frac{W * (N-1)}{L} \right\rfloor} \right\rceil = 1 \quad \text{and} \quad \lim_{W\to\infty} \left\lceil \frac{K}{\left\lfloor \frac{W * \frac{3}{4} N}{L} \right\rfloor} \right\rceil = 1$$

So, we have,

$$\lim_{W\to\infty} \frac{T_{1-FT} \text{ for } \textit{Graceful Degradation}}{T_{1-FT} \text{ for } 1-SRE}$$

$$= \lim_{W\to\infty} \frac{(T_{comm} + 2 * W * T_{comp}) * (2L-1) * K * \left\lceil \frac{K}{\left\lfloor \frac{W * (N-1)}{L} \right\rfloor} \right\rceil + T_{1-rcfg}}{(T_{comm} + W * T_{comp}) * (2L-1) * K * \left\lceil \frac{K}{\left\lfloor \frac{W * \frac{3}{4} N}{L} \right\rfloor} \right\rceil + T_{1-rcfg}}$$

$$= \lim_{W\to\infty} \frac{(2L-l) * K * (2 * W * T_{comp})}{(2L-l) * K * (W * T_{comp})}$$

$$= 2$$

Since the ratio is > 1, we show that the T_{1-fT} for Graceful Degradation is larger than T_{1-FT} for our approach for 1-SRE, when $W \rightarrow \infty$. □

IX. OPEN PROBLEMS AND PROPOSED RESEARCH

In this chapter, we discuss some related open problems which have direct relevance to the topics of the preceding chapters, and state several proposed work for further research.

A. Gap between Existence and Non-Existence of 1SRE's

In Chapter VI, we described a composition approach to systematically construct 1SRE's for rings $\leq (3/4)2^d$. While in Chapter VII, we introduced a sufficient condition for the non-existence of 1SRE's for rings $> (3/4)2^d$. However, the composition approach and the sufficient condition do not cover all the rings. There is a gap between the rings with 1SRE's and without 1SRE's, since for some rings $> (3/4)2^d$ the existence of 1SRE's in a d-cube is undecidable. The size of the gap increase as the dimension of the embedding hypercube grows. However, while the dimension of the hypercube grows, the number of spare nodes grow also, and it becomes more possible to find some 1SRE's for rings $> (3/4)2^d$ in a d-cube since there are more spare nodes to choose for recovery. So, for higher dimensional cubes, the breaking of the length limitation, $(3/4)2^d$, for 1SRE's is possible. Currently, we have found a 1SRE of a length-386 = $(3/4)2^9 + 2$ ring in a 9-cube, and this implies finding more 1SRE's in higher dimensional hypercubes is possible. However, there seems to be no algorithmatic technique to describe the pattern of such 1SRE's in a higher dimensional hypercube. Meanwhile a stronger sufficient condition for the non-existence of 1SRE's is also needed.

B. Multiple Node Faults

Liang and Tsai considered the problem of multi-failure fault-tolerance of hypercubes in [LiTs90]. In their paper, new reconfiguration algorithms were proposed to reallocate the function of failed nodes to spare nodes, and two types of multiple faults were considered, clustered faults (a connected sequence of active nodes fail simultaneously) and concurrent faults (more than one segments of nodes fail at the same time). With the new reconfiguration algorithm, they investigated the the multi-failure reconfigurability of two proposed fault-tolerant embedded rings, Mapping II and Mapping III in [ChLT88b].

Our schemes described in Chapter IV, V, and VI which are designed for tolerating one single fault, will, in many cases, tolerate more than a single fault. For example, it can be shown that our scheme for 1SRE's (in Chapter VI) can tolerate 2 (sequential) faults if the Hamming distance of the 2 faulty nodes is 1, 3, or ≥ 5. We conjecture that our scheme for 1SRE's can also tolerate multiple faults (> 2) if the Hamming distance of any pair of faulty nodes is 1, 3, or ≥ 5. We could characterize and enumerate all multiple-fault scenarios that can be handled by our single-fault embedding schemes; however, it would be more satisfying to *guarantee* the tolerance of multiple faults. To accomplish this, a more complicated reconfiguration algorithm which also accommodates the existing xor-reconfiguration algorithm is needed.

C. Single and Multiple Link Faults

Not only nodes may be faulty, also links between processors in hypercubes are subject to become faulty. Since there are more number of edges (2^d) than that of nodes

($d2^{d-1}$) in a hypercube, it is predictable that the reconfigurable embedding schemes assuming faults in links only will tolerate more number of faults than those assuming faults in nodes only.

By treating one of the active nodes incident to the faulty link faulty, our proposed embedding schemes in Chapter IV, V, and VI are also able to tolerate one single faulty link. A result better than $(3/4)2^d$ should be achievable for successfully constructing 1SRE's to tolerate one single faulty link. Similar to the case of multiple node faults, we can characterize and enumerate all multiple-fault scenarios that can be handle by the single-fault scheme, as well as generate new schemes for tolerating multiple faults.

D. Conjecture of NP-completeness

Although in Chapter VI we derive a composition method to systematically construct 1SRE's for rings of lengths $\leq (3/4)2^d$ in a d-cube, the existence of 1SRE's for some ring of lengths $> (3/4)2^d$ in a d-cube ($d \geq 6$) is unknown. It is easy to see that the general problem of finding 1SRE's: Given an even integer k and an integer d, is there a 1SRE of a length-k ring in a d-cube? is in NP since we can guess an embedding and check it easily in polynomial time [GaJo79]. However, the problem is remained being a conjecture of NP-completeness although much effort was put into by us trying to show it.

BIBLIOGRAPHY

[AfPP85] Afrati, F., Papadimitriou, C. H., and Papageorgiou, G "The Complexity of Cubical graphs," *Information Processing Letters*, Vol. 66 (1985), pp. 53-60.

[AlBS87] Alspach, B., Bermond, J.-C., and Sotteau, D. "Decomposition into Cycles I. Hamiltonian Decomposition," Technical Report No. 87-12, Simon Fraser University, 1987.

[Andr76] Andrews, G. *The Theory of Partitions.* Reading: Addision-Wesley Publishing Company, 1976.

[ArGr81] Armstrong, J. R. and Gray, F. G. "Fault Diagnosis in a Boolean n-Cube Array of Microprocessors," *IEEE Transactions of Computers*, Vol. 30 (1981), pp. 587-590.

[Bhat83] Bhat, K. V. S. "An Efficient Approach for Fault Diagnosis in a Boolean n-Cube Array of Microprocessors," *IEEE Transactions of Computers*, Vol. 32 (1983), pp. 1070-1071.

[Bhlp85] Bhatt, S. N. and Ipsen, I. C. F. "How to Embed Trees in Hypercubes," Research Report No. 443, Department of Computer Science, Yale University, Dec. 1985.

[BHGS90] Boehncke, K., Heller, H., Grubmüller, H. and Schulten, K. "Molecular Dynamics Simulations on Systolic Ring of Transputers," *Proceedings of the 3rd Conference of the North American Transputer Users Group*, (1990), pp.83-94.

[Bokh81] Bokhari, S. H. "On the Mapping Problem," *IEEE Transactions on Computers*, Vol. 30 (1981), pp. 202-214.

[BrSc85] Brandenburg, J. E. and Scott, D. S. "Embeddings of Communication Trees and Grids into Hypercubes," Technical Report No. 280182-001, Intel Scientific Computers, 1985.

[BuPr90] Bui, T. N. and Prabhu, U. "Reconfigurable Embeddings of a Multi-Dimensional Mesh in a Hypercube," *Proceedings of the 28th Allerton Conference on Communication, Control, and Computing*, (1990), pp. 226-235.

[Chan88] Chan, M. Y. "Dilation-2 Embeddings of Grids into Hypercubes," *Proceedings of International Conference on Parallel Processing*, (1988), pp. 295-298.

[Chan89a] Chan, M. Y. "Embeddings of 3-Dimensional Grid into Optimal Hypercube," *Proceedings of the 4th Conference on Hypercubes, Concurrent Computers, and Applications*, Vol. I (1989), pp. 297-299.

[Chan89b] Chan, M. Y. "Embedding of d-dimensional grids into optimal hypercubes," *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, (1989).

[ChCh88] Chan, M. Y. and Chin, F. Y. L. "On Embedding Rectangular Grids in Hypercubes," *IEEE Transactions on Computers*, Vol. 37 (1988), pp. 1285-1288.

[ChCP91] Chan, M. Y., Chin, F. Y. L., and Poon, C. K. "Optimal Specified Root Embedding of Full Binary Trees in Faulty Hypercubes," Springer Lecture Notes in Computer Science, No. 557 (1991), pp.241-250.

[ChLe91a] Chan, M. Y. and Lee, F. Y. L. "Distributed Fault-Tolerant Embeddings of Rings in Hypercubes," *Journal of Parallel Distributed Computing*, Vol 11 (1991), pp. 63-71.

[ChLe91b] Chan, M. Y. and Lee, S.-J. "On the Existence of Hamiltonian Circuits in Faulty Hypercubes," *SIAM Journal on Discrete Mathematics*, Vol. 4 (1991), pp. 511-527.

[ChLT88a] Chen, S. K., Liang, C. T., and Tsai, W. T. "Loops and Multi-Dimensional Grids on Hypercubes : Mapping and Reconfiguration Algorithms," *Proceedings of International Conference on Parallel Processing*, (1988), pp. 315-322.

[ChLT88b] Chen, S. K., Liang, C. T., and Tsai, W. T. "An Efficient Multi-Dimensional Grids Reconfiguration Algorithm on Hypercubes," *Proceedings of Fault-Tolerant Computing Symposium*, (1988), pp. 368-373.

[CyKV87] Cybenko, G., Krumme, D. W., and Venkataraman, K. N. "Fixed Hypercube Embedding," *Information Processing Letters*, Vol. 25 (1987), pp. 35-39.

[DeJe86] Deshpande, S. R. and Jenevein, R. M. "Scalability of a Binary Tree on a Hypercube," *Proceedings of International Conference on Parallel Processing*, (1986), pp. 661-668.

[Doug77] Douglas, R. J. "Bounds on the Number of Hamiltonian Circuits in the n-cube," *Discrete Mathematics*, Vol. 17 (1977), pp. 143-146.

[EdWa87] Edmiston, E. and Wagner, R. A. "Parallelization of the Dynamic Programming Algorithm for Comparison of Sequences," Proceedings of International Conference on Parallel Processing, (1987), pp. 78-80.

[GaJo79] Garey, E. and Johnson, D. *Computer and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: W.H. Freeman, 1979.

[GaPS90] Gallivan, K. A., Plemmons R. J., and Sameh, A. H. "Parallel Algorithms for Dense Linear Algebra Computations," *SIAM Review*, Vol. 32 (1990), pp. 54-135.

[Gilb58] Gilbert, E. N. "Gray Codes and Paths on the *n*-Cube," *Bell System Technical Journal*, Vol. 37 (1958), pp. 815-826.

[GoVa89] Golub, G. H. and Van Loan C. F. *Matrix Computations*. Baltimore: The Johns Hopkins University Press, 2nd, 1989.

[HoJo89] Ho, C.-T. and Johnsson, S. L. "Dilation d Embedding of a Hyper-Pyramid into a Hypercube," *Proceedings of Supercomputing Conference*, (1989), pp. 294-303.

[HoJo90] Ho, C.-T. and Johnsson, S. L. "Embedding Meshes in Boolean Cubes by Graph Decomposition," *Journal of Parallel and Distributed Computing*, Vol. 8 (1990), pp. 325-339.

[IpSS86] Ipsen, I. C. F., Saad, Y., and Schultz, M. "Dense Linear Systems on a Ring of Processors," *Linear Algebra and Its Applications*, Vol. 77 (1986), pp. 205-239.

[JaTh90] Jackson, B. W. and Thoro, D. *Applied Combinatorics with Problem Solving*. Addison-Wesley Publishing Company, 1990.

[KrHI91] Krishnakumer, N., Hedge, V., and Iyenger, S. S. "Fault Tolerant Based Embeddings of Quadtrees into Hypercubes," *Proceedings of International Conference on Parallel Processing*, Vol. III (1991), pp. 244-249.

[KrVC86] Krumme, D. W., VenKataraman, K. N., and Cybenko, G. "Hypercube Embedding is NP-Complete," *Proceedings of the 1st Conference on Hypercube Multiprocessors*, (1986), pp. 148-157.

[KuRR88] Kumar, V., Rao, V. N., and Ramesh, K. "Parallel Depth First Search on the Ring Architecture," *Proceedings of the International Conference on Parallel Processing*, (1988), pp. 128-132.

[LaMe88] Lander, E. and Mesirov, J. P. "Protein Sequence Comparison on a Data Parallel Computer," *Proceeding of International Conference on Parallel Processing*, (1988), pp. 257-263.

[LaWh90] Lai, T. H. and White, W. "Mapping Pyramid Algorithms into Hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 9 (1990), pp. 42-54.

[LaZB92] Latifi, S., Zheng, S.-Q., and Bagherzadeh, N. "Optimal Ring Embedding in Hypercubes with Faulty Links," *Proceedings of Fault-Tolerant Computing Symposium*, (1992), pp. 178-184.

[Leet90] Lee, T. C. "Quick Recovery of Embedded Structures in Hypercube Computers," *Proceedings of the 5th Distributed Memory Computing Conference*, (1990), pp. 1426-1435.

[Leig92] Leighton, F. T. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes*, San Mateo, CA: M. Kaufmann Publishers, 1992.

[LiMc92a] Liu, J. and McMillin, B. M. "A Divide and Conquer Ring Embedding Scheme in Hypercubes with Efficient Recovery Ability," *Proceedings of International Conference on Parallel Processing*, Vol. III (1992), pp. 38-45.

[LiMc92b] Liu, J. and McMillin, B. M. "An Enhanced Reconfigurable Embedding Scheme for Rings in Hypercubes," *Proceedings of International Conference on Parallel and Distributed Systems*, (1992), pp. 298-305.

[Lium78] Liu, M. T. "Distributed Loop Computer Networks," *Advances in Computers*, Vol. 17 (1978), pp. 163-221.

[LiMW89] Lipton, R. J., Marr, T. G., and Welsh, J. D. "Computational Approaches to Discovering Semantics in Molecular Biology," *Proceedings of the IEEE*, Vol. 77 (1989), pp. 1056-1060.

[LiSM92] Liu, J., Sager, T. J., and McMillin, B. M. "An Improved Characterization of 1-Step Recoverable Embeddings: Rings in Hypercubes," Technical Report No. CSC-92-07, Department of Computer Science, University of Missouri, Rolla, April, 1992.

[LiTs90] Liang, C. T. and Tsai, W. T. "Multi-Failure Fault-Tolerance of Embedded Loops on Hypercubes: Issues and Performance Study," *Proceedings of Symposium on Parallel and Distributed Processing*, (1990), pp. 511-518.

[PaSt92] Palmer, J. and Steele Jr., G. L. "Connection Machine Model CM-5 System Overview," *Proceedings of 4th Symposium on the Frontiers of Massively Parallel Computation*, (1992), pp. 474-483.

[PrMe88] Provost, F. J. and Melhem, R. "Distributed Fault-Tolerant Embeddings of Binary Trees and Rings in Hypercubes," *Proceedings of International Workshop on Defect and Fault Tolerance in VLSI Systems*, (1988), pp. 339-346.

[PrMe92] Provost, F. J. and Melhem, R. "A Distributed Algorithm for Embedding Trees in Hypercubes with Modifications for Run-Time Fault Tolerance," *Journal of Parallel and Distributed Computing*, Vol. 14 (1992), pp. 85-89.

[Same85] Sameh, A. "On Some Parallel Algorithms on A Ring of Processors," *Comput. Phys. Comm.*, Vol. 37 (1985), pp. 159-166.

[SaSc88] Saad, Y. and Schultz M. H. "Topological Properties of Hypercubes," *IEEE Transactions on Computer*, Vol. 37 (1988), pp. 867-872.

[SeYv91] Serge, M. and Yves, R. "Elastic Load-Balancing for Image Processing Algorithms," *Proceedings of the 1st International Conference of the Austrian Center for Parallel Computation, (1991), pp. 438-451.*

[Stou86] Stout, Q. F. *Hypercubes and Pyramids*, in Pyramidal Systems for Computer Vision, Cantoni, V. and Levialdi, S. editors. New York: Springer-Verlag, 1986.

[TsLa93] Tseng, Y.-C. and Lai, T.-H. "Resilient and Flexible Ring Embeddings in an Injured Hypercube," Technical Report No. OSU-CISRC-7/93-TR-26, Department of Computer and Information Science, The Ohio State University, 1993.

[Vara91] Varadarajan, R. "Embedding Shuffle Networks in Hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 11 (1991), pp. 252-256.

[WaCo90] Wagner, A. and Corneil, D. G. "Embedding Trees in a Hypercube is NP-Complete," *SIAM Journal of Computing*, Vol. 19 (1990), pp. 570-590.

[WaCy92] Wang, A. and Cypher, R. "Fault-tolerant Embeddings of Rings, Meshes, Tori in Hypercubes," *Proceedings of the 4th IEEE symposium on Parallel and Distributed Processing*, (1992), pp. 20-29.

[WaCM91] Wang, A., Cypher, R., and Mayr, E. "Embedding Complete Binary Trees in Faulty Hypercubes," *Proceedings of the 3th IEEE Symposium on Parallel and Distributed Processing*, (1991), pp. 112-119.

[WaOz90] Wang, J. and Özgüner, F. "Embeddings, Communication and Performance of Algorithms in Faulty Hypercubes," *Proceedings of the 5th Distributed Memory Computing Conference*, (1990), pp. 1455-1460.

[Wuay85] Wu, A. Y. "Embeddings of Tree Networks into Hypercubes," *Journal of Parallel and Distributed Computing*, Vol. 2 (1985), pp. 238-249.

[YaHa86] Yanney, R. M. and Hayes, J. P. "Distributed Recovery in Fault-Tolerant Multiprocessors Networks," *IEEE Transactions on Computer*, Vol. 35 (1986), pp. 871-879.

[YaTR91] Yang, P.-J., Tien, S.-B., and Raghavendra, C. S. "Embedding of Multidimensional Meshes on to Faulty Hypercubes," *Proceedings of International Conference on Parallel Processing*, Vol. I (1991), pp. 571-574.

VITA

Jun-Lin Liu was born in Tainan, Taiwan on Feburary 17, 1962. He received his Bachelor of Engineering in Computer Science in 1984 from Tamkang University at Tamsui, Taiwan. After two-year military service and one-year computer job, he came to the United States of America in December 1987. He has been a graduate student of Computer Science Department of University of Missouri at Rolla since January 1988. He earned his Master of Science in Computer Science in May, 1989, and has been in the Ph.D program since then. His research areas of interest include: Fault Tolerance, Parallel and Distributed Computing, Parallel Algorithms, and Graph and Combinatorial theory. He is a student member of the ACM and IEEE Computer Society.