Pergamon

Computers Math. Applic. Vol. 34, No. 11, pp. 25–50, 1997

0898-1221/97 $17.00 + 0.00

PII: S0898-1221(97)00217-4

A Local-Sparing Design Methodology for Fault-Tolerant Multiprocessors

S. DUTT
Department of Electrical Engineering and Computer Science
University of Illinois-Chicago
Chicago, IL 60607-7053, U.S.A.

J. P. HAYES
Department of Electrical Engineering and Computer Science
University of Michigan
Ann Arbor, MI 48109-2122, U.S.A.

Abstract—We present a comprehensive design methodology for constructing low-cost multiprocessors that use local spares to tolerate the failure of either processor clusters or individual processors. We first formalize the concepts of global- and local-sparing in terms of graph automorphisms. We then present a method for partitioning a multiprocessor graph by its automorphisms and for incorporating local-sparing to tolerate faults. We emphasize local-sparing designs, since they offer higher reliability-to-cost ratios and can reconfigure faster and in a localized manner. When the spare clusters in each local subsystem are certain sizes, our designs are optimal in the number of spare intersubsystem links. They are all efficient (optimal in some cases) in terms of the number of spare intrasubsystem links. We present switch-based implementations that significantly reduces the spare link complexities of the designs. These implementations are equally efficient for any spare cluster size, so they yield efficient local-sparing designs that can tolerate individual processor faults (cluster size of one). Algorithms for fast, localized, and incremental reconfiguration of our FT designs are also developed. Finally, we demonstrate that our local-sparing designs have higher reliability-to-cost ratios than previous designs.

Keywords—Automorphisms, Fault-tolerant multiprocessors, Global-sparing, Local-sparing, Reconfiguration, Structural fault tolerance.

1. INTRODUCTION

Fault-tolerant (FT) multiprocessors are widely used in on-line transaction processing; they are essential in critical computations like aircraft control. Fault tolerance is also desirable in massively parallel systems that have a relatively high failure probability. Examples of such multiprocessors are the nCUBE2 hypercube computer, Thinking Machine's CM-5 (a fat-tree structured computer), and Intel's Paragon (a mesh-connected computer). Many parallel algorithms designed for such multiprocessors are tailored to exploit their interconnection structures [1–4]. Such structure-sensitive algorithms are thus vulnerable to faults that damage a multiprocessor's structure. Hence, it is important to design multiprocessor systems with *structural fault tolerance* (SFT), which is defined as the ability to reconfigure around faults to preserve interconnection structure. SFT (or FT for short) in general raises the availability and performability of medium- to large-scale multiprocessors.

This research was supported in part by the Office of Naval Research under Contract N00014-85K0531 and N00014-90J1860, and in part by NSF Grants MIP-9210049 and MIP-9200526.

A number of FT designs for specific multiprocessor architectures have been proposed based on graph theoretic models in which the processor-to-processor interconnection structure is represented by a graph [5–14]. Most previous work does not satisfactorily meet important design objectives like low hardware overhead, efficient reconfigurability, and applicability to a wide range of multiprocessor structures and faults. Among the few previous design methods that are applicable to any multiprocessor structure are the Diogenes method [9], the "decoupling networks" technique [8], and the "edge-skipping" technique for the class of diagonal graphs (which include meshes and hypercubes) [7]. Furthermore, most prior FT systems are designed to tolerate independent faults only. Recent work [15,16] suggests that in highly-integrated systems, processors often fail in clusters due to correlated factors like shared defective wafers and power supply fluctuations. Hence, when the probability of clustered processor failures is significant, it is more cost-effective to design multiprocessors that can tolerate a cluster of m physically adjacent processor failures rather than one that can tolerate m independent faults. As we will demonstrate later, our design method incorporates much lower complexity for the former type of FT design. In situations where clustered failures are less likely, it is, however, more cost-effective in terms of the number of spare processors to tolerate independent individual faults, for example, to tolerate any two faults in a subsystem rather than any two m-cluster faults, where $m > 1$. We thus also provide hardware-efficient designs to tolerate individual processor faults.

The goal of this work is to develop a methodology of FT design for clustered as well as individual processor faults, that is, both general and hardware-efficient, that can exploit local-sparing, and that allows reconfiguration to take place in an incremental, distributed, and localized manner. *Incremental reconfiguration* is defined as the ability to reconfigure around any new set of faults without undoing any earlier reconfiguration around previous faults—this reduces the down-time of a system when faults occur. *Distributed reconfiguration* allows different parts of the system to reconfigure concurrently—this not only makes reconfiguration faster, but also avoids reliance on a central controller. In designs with *localized reconfiguration* capability, each subsystem with its own local spares can reconfigure around its faults independent of the other subsystems—this makes reconfiguration simpler and faster.

The focus of this work is a graph automorphism-based method for designing FT multiprocessors with local-sparing; we refer to this as the ALS (for Automorphic Local-Sparing) method. Both cluster- and individual-fault-tolerant designs are presented, as well as switch-based implementations of these designs that significantly reduce the spare link overhead.

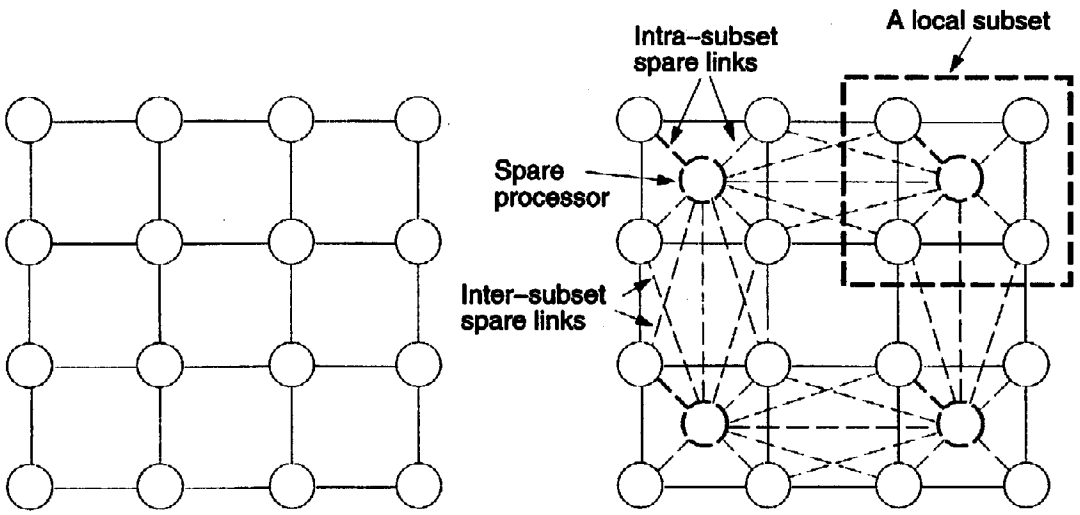
2. PRELIMINARIES

A graph $G(V, E)$ is used here to represent the structure of a distributed-memory multiprocessor system with a static interconnection network, where each node in the node set $V(G)$ of G represents a processor-router pair in the system, and the edge set $E(G)$ of G represents the dedicated interprocessor communication links. For a subset S of $V(G)$, $G(S)$ will denote the subgraph induced in G by the nodes in S. Unless otherwise specified, N and e will denote the number of nodes and edges, respectively, of G, while d will denote its maximum node degree.

Two types of failures in a multiprocessor system are of interest, processor failures and link failures. In our model, a *link failure* corresponds to the deletion of an edge from G, while a *processor failure* corresponds to the removal of a node and all edges incident on it from G. If F is the set of faults (faulty nodes or edges) in G, then $G - F$ denotes the graph obtained by deleting the faults from G as explained above. We consider only processor failures, since processors are much more complex than links and are thus are more likely to fail. Moreover, since a faulty processor effectively causes all incident links to be removed from the system, a link failure can be modeled as the failure of an incident processor.

G' is a *supergraph* of G if $E(G') \supset E(G)$ and $V(G') \supseteq V(G)$. G^e is an *edge-supergraph* of G if $E(G^e) \supset E(G)$ and $V(G^e) = V(G)$. A supergraph $G'[k, G]$ of G is a *k-fault-tolerant* (k-FT)

realization of G, if for any set F of k nodes of G', $G' - F$ contains a subgraph isomorphic to G. We use G' to denote either a generic FT supergraph of G, or a specific type of FT supergraph that is clear from the context. G is called the *basic graph*, and its nodes and edges are *primary* nodes and edges. The nodes and edges of the fault-free subgraph of G' serving as the currently active graph isomorphic to G are *active* nodes and edges, respectively, while the remaining nodes and edges of G' are *spare*. The following convention will be used in the figures. Spare nodes are shown as white circles with a light or dashed outline, spare edges as light or dashed lines, active nodes as white circles with solid outlines, active edges as solid lines, and faulty nodes as black circles.



(a) A 4×4 mesh $M_{4,4}$.

(b) Its local-sparing FT supergraph that can tolerate one fault in each of the four disjoint 2×2 submeshes.

Figure 1.

Instead of a global-sparing design of the foregoing kind, we can associate spares locally with each subset V_i of a partition $\{V_1, \ldots, V_t\}$, of $V(G)$. Spare edges are then added in G so that, for every set of faults F in the resulting supergraph G' that contains at most k_i faulty nodes from each V_i, $G' - F$ contains a subgraph isomorphic to G. Such a supergraph G' is denoted by $G'[\{k_1, \ldots, k_t\}, G]$. Figure 1b shows the supergraph $G'[\{1,1,1,1\}, M_{4,4}]$ of the 4×4 mesh $M_{4,4}$ (Figure 1a) that can tolerate one fault in each of the four corner 2×2 submeshes. It is also useful to design local-sparing systems that can tolerate fault clusters as opposed to independent faults in each subset V_i. A "cluster" fault pattern is one in which certain groups or clusters of processors fail due to correlated fault-causing events like radiation or voltage surges. Cluster fault patterns are more formally defined later.

Next, we review some basic terminology pertaining to graph automorphisms [17,18]. Two graphs G and H are said to be *isomorphic* if there is a bijection $\phi : V(G) \rightarrow V(H)$ such that $\{x, y\} \in E(G)$ if and only if $\{\phi(x), \phi(y)\} \in E(H)$; ϕ is an *isomorphism* from G to H. An isomorphism from G to itself is called an *automorphism* of G. Basically, an automorphism of G is a permutation of $V(G)$ that preserves adjacency. The set of all automorphisms of G forms a group under composition, and is denoted by $\mathrm{aut}(G)$. Two nodes x and y of G are said to be *similar* if there is an automorphism mapping x to y.

Any automorphism α can be expressed uniquely (up to ordering) as the product of disjoint cycles $(x_{1,0}, x_{1,1}, \ldots, x_{1,l_1-1})(x_{2,0}, x_{2,1}, \ldots, x_{2,l_2-1}) \cdots (x_{t,0}, x_{t,1}, \ldots, x_{l_t-1})$. This cyclic representation means that $\alpha(x_{1,0}) = x_{1,1}$, $\alpha(x_{1,1}) = x_{1,2}, \ldots, \alpha(x_{1,l_1-1}) = x_{1,0}, \ldots$, $\alpha(x_{t,0}) = x_{t,1}$, $\alpha(x_{t,1}) = x_{t,2}, \ldots, \alpha(x_{t,l_t-1}) = x_{t,0}$. If α is an automorphism of G, then so is each element of the

set subgrp(α) = $\{e, \alpha, \alpha^2, \ldots, \alpha^{o(\alpha)-1}\}$, where $o(\alpha)$ is the least integer i such that $\alpha^i = e$, the identity permutation, and is called the *order* of α. Here, α^p is recursively defined as $\alpha^p = \alpha \circ \alpha^{p-1}$, where $\circ$ denotes function composition. subgrp(α) is a subgroup of aut(G), and is said to be *generated* by α. A *circulant* graph is one that has at least one automorphism consisting of a single cycle that contains all N nodes.

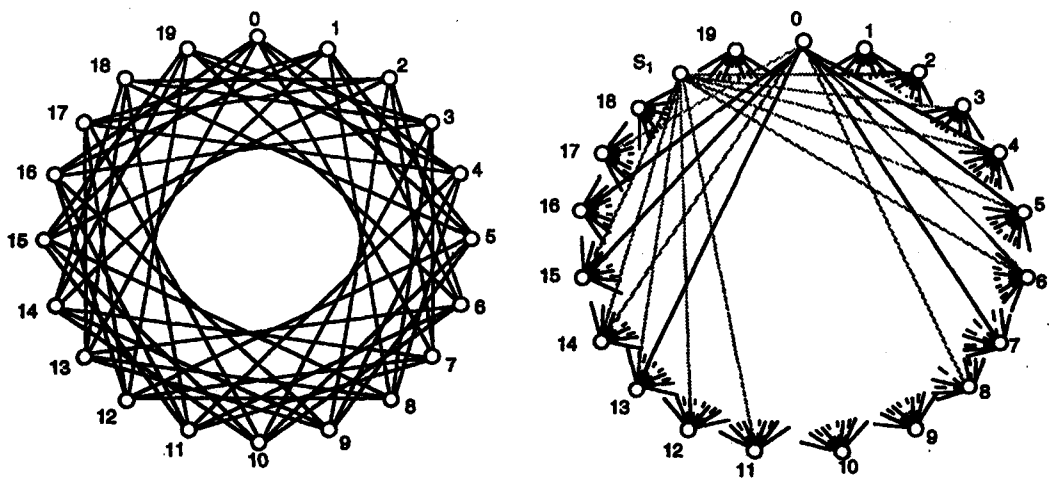
3. k-FT DESIGNS

Here, we recapitulate relevant aspects of the automorphism-based global-sparing k-FT design method presented in [19,20]. For a circulant graph G, we choose an automorphism α that has a single cycle containing all the N nodes. A spare node s_1 is arbitrarily inserted between any two nodes in this cycle to construct a single cycle permutation α_1 with $N+1$ nodes. For example, suppose G is a circulant graph with five nodes and $\alpha = (1,2,3,4,5)$ is an automorphism of G. We insert a spare node s_1 between nodes 2 and 3 to obtain the permutation $\alpha_1 = (1,2,s_1,3,4,5)$, and construct a supergraph G_1 of G with one spare node s_1 that has α_1 as an automorphism. We can then use the automorphisms (of G_1) in subgrp(α_1) to reconfigure around any single faulty node; subgrp(α_1) contains the following six automorphisms of G_1:

$$\begin{aligned}
\alpha_1 &= (1,2,s_1,3,4,5),\\
(\alpha_1)^2 &= (1,s_1,4)(2,3,5),\\
(\alpha_1)^3 &= (1,3)(2,4)(s_1,5),\\
(\alpha_1)^4 &= (1,4,s_1)(2,5,3),\\
(\alpha_1)^5 &= (1,5,4,3,s_1,2),\\
(\alpha_1)^6 &= e = (1)(2)(s_1)(3)(4)(5).
\end{aligned}$$

It can thus be seen that, for any faulty nodes u, there is some automorphism $(\alpha_1)^t$ in subgrp(α_1) that maps s_1 to u. Under this new mapping, only the nonfaulty nodes in G_1 are labeled as nodes of G; the faulty node is now labeled as the spare node. Furthermore, any automorphism of G_1 maps some subgraph J to an isomorphic subgraph K. Thus, the new labeling of the nonfaulty nodes of G_1 determines a nonfaulty copy of G, and G_1 is a 1-FT supergraph of G.

We now explain how G_1 can be constructed from G so that α_1 is an automorphism of G. Consider an arrangement of the nodes of G on a circle C according to the circular order of



(a) A circulant graph H with 20 nodes.

(b) A 1-FT circulant supergraph H_1 of H with spare node s_1.

Figure 2.

automorphism α of G. For example, Figure 2a shows such a circular arrangement of a circulant graph H according to the automorphism $(0, 1, \ldots, 19)$; this is called a *(symmetric) embedding* of G on a circle, and is denoted by circ(G). We define the *clockwise circular distance* (cd^+) between two nodes u and v, denoted $cd^+(u, v)$, on $\mathcal{C}$ as one plus the number of nodes encountered when going clockwise from u to v and visiting only the first $\lceil N/2 \rceil$ nodes inclusive of u on $\mathcal{C}$. Thus, $cd^+(u, v)$ is defined only if v lies in the first half of nodes of $\mathcal{C}$ when going clockwise from u. The *counterclockwise circular distance* (cd^-) between u and v is similarly defined. Note that for any two nodes u and v, exactly one of $cd^+(u, v)$ and $cd^-(u, v)$ is defined, unless N is odd and $cd^+(u, v) = cd^-(u, v) = \lceil N/2 \rceil$. The *circular distance* (cd) between two nodes u and v, denoted, $cd(u, v)$, is defined as either $cd^+(u, v)$ or $cd^-(u, v)$, whichever is defined. The cd of an edge (u, v) is the same as $cd(u, v)$. The cd^+s between each node u and its neighbors in the graph H of Figure 2a are four, five, and seven. Similarly, the cd^-s between each node u and its neighbors in H are also four, five, and seven. This is a characteristic of circulant graphs, viz., they can be laid out on a circle such that every node has neighbors at the same set of clockwise and counterclockwise cds from it.

The embedding circ(G), and hence G, can be characterized by its *distance sequence* $ds(G)$, which is the ordered set, increasing from left to right, of the cds between any node u in G and all its neighbors in one of the semicircles formed in circ(G) by the diameter through u. The distance sequence of H in Figure 2a is $(4, 5, 7)$.

Suppose that spare node s_1 is inserted between nodes u and v to form α_1 from α. To construct G_1, we insert s_1 between the same pair of nodes on circ(G). As a result, the cd of some edges of G increases by one. It is easy to show that for each cd d_i in $ds(G)$, there is an edge of G with cd d_i in circ(G), whose cd becomes $d_i + 1$ on inserting s_1. Similarly, it can be shown that the cd of at least one edge of G remains d_i after inserting s_1. Let D be the set of all edge cds obtained by inserting s_1 in circ(G). Then, $D = ds(G) \hat{+} 1$. To construct G_1, for every node u of G_1 and each $d_i \in D$, we insert spare edges of cd d_i incident on u, if such edges are not already present. G_1 has α_1 as an automorphism, and its distance sequence is D. In Figure 2b, edges have been added in this manner to construct a 1-FT supergraph H_1 of the graph H in Figure 2a; for clarity, only the edges incident on nodes s_1 and 0 are shown. It can be seen that $ds(H_1) = ds(H) \hat{+} 1 = (4, 5, 6, 7, 8)$.

Since G_1 is circulant, we can again construct a 1-FT supergraph G_2 of G_1 as described above. Note that if H is an i-FT supergraph of G, and H' is a j-FT supergraph of H, then H' is also an $(i + j)$-FT supergraph of G; G_2 is thus a 2-FT supergraph of G. A k-FT supergraph G_k of G can be obtained by iteratively constructing 1-FT supergraphs of $G, G_1, \ldots, G_{k-1}$ in the manner described above. This iterative construction procedure is called SUPER_CIRCULANT [20], and the k-FT supergraph of G it generates is denoted by $G'_{\text{auto}}[k, G]$. It is shown in [20] that SUPER_CIRCULANT constructs optimal k-FT supergraphs with node degree $k + d$ for certain classes of circulant graphs. In the worst case, the node degree of a k-FT supergraph is $(k+1)d$. A switch implementation method for the k-FT supergraphs using Nk 1-to-$(k+1)$ demultiplexers is also presented in [20] in which the node degree reduced to d; the switch and wiring complexity of this design is $\Theta(Nkd^2)$; this compares very favorably with the hardware complexity of previous design methods.

When G is noncirculant, a method called *cycle merging* is used to construct a circulant edge-supergraph of G. SUPER_CIRCULANT is then applied to this edge-supergraph. This approach works efficiently when G is "circulant-like", as is the case with n-dimensional hypercubes and 2-D meshes. The efficiency of the automorphic method in constructing k-FT hypercubes is demonstrated in [20]. For many graphs that are not circulant-like, for example, trees and pyramids, extension to a circulant graph can be expensive. In such cases, using the methodology developed in this paper, it is possible to design efficient local-sparing systems that associate spares with each cycle of a (multicycle) automorphism of G. Even for circulant-like graphs for which efficient k-FT designs are possible, automorphic local-sparing (ALS) designs yield lower link complexities, and

faster and localized reconfiguration compared to k-FT (global-sparing) designs. Thus, the ALS method should prove useful for incorporating fault tolerance in a wide range of multiprocessor structures.

In the rest of the paper, we use graph automorphisms to develop the method for designing FT multiprocessors with local spares. We partition the node set of G by the cycles of some automorphism $\alpha = C_1\, C_2, \ldots, C_t$ of G, i.e., each subset V_i of this partition of $V(G)$ is the cycle C_i of α. Automorphism α is called the *seed* automorphism for the resulting FT design. Clusters of spare nodes (of size ≥ 1) are associated locally with each cycle C_i and intracycle spare edges added to make them fault-tolerant. Finally, spare intercycle edges are added so that the resulting supergraph G' can tolerate fault clusters in each cycle. Note that for a cluster size of one, we obtain designs that tolerate independent single faults in each cycle.

In all our figures, the nodes of a cycle of an automorphism are embedded on a circle C that is evident from the circular arrangement of nodes on it. For brevity, we will state most of the results without formal proofs. We attempt to provide an intuitive explanation of all results in the text.

4. FAULT TOLERANCE IN CIRCULANT SUBGRAPHS

First, we describe the cluster-FT method in terms of circulant subgraphs of G. Suppose G is a noncirculant graph, and $\alpha = C_1\, C_2, \ldots, C_t$ is an automorphism of G. Recall that cycle C_i is represented as $C_i = (x_{i,0}, x_{i,1}, \ldots, x_{i,l_i-1})$, where $l_i = |C_i|$. It follows from the definition of a circulant graph that the subgraph $G(C_i)$ induced in G by C_i is circulant, and has C_i as a single-cycle automorphism. We will omit the G in all notation containing $G(C_i)$, whenever it is clear that we are referring to the subgraph induced by C_i rather than the cycle C_i itself. Let us add a cluster $\{s_{1,1}, \ldots, s_{1,m}\}$ of m consecutive spare nodes to $\mathrm{circ}(C_i)$ between two nodes $x_{i,s}$ and $x_{i,s+1}$. Then, generalizing the argument given in Section 3, we construct the circulant supergraph of C_i containing these consecutive m spares, by inserting edges with *cds* d_j and $d_j + m$ at every node, for every d_j in $ds(C_i)$. The resulting supergraph is denoted by $G'(C_i(m))$ or simply $C_i(m)$). Now $ds(C_i(m)) = ds(C_i) \hat{+} m$, and the permutation $(x_{i,0}, \ldots, x_{i,s}, s_{1,1}, \ldots, s_{1,m}, x_{i,s+1}, \ldots, x_{i,l_i-1})$, denoted by C_i^m, is an automorphism of $C_i(m)$. For a set of integers A and an integer r, $A \hat{+} r$ is defined as the set $A \cup \{a_i + r : a_i \in A\}$. For example, if $G(C_i)$ is a simple cycle, then $ds(C_i) = (1)$, and

$$ds(C_1(5)) = ds(C_1) \hat{+} 5 = (1) \hat{+} 5 = (1, 6).$$

The construction procedure CLUSTER_CIRCULANT (C_i, m) for $C_i(m_i)$ is formally described in Figure 3.

The supergraph $C_i(m)$ can tolerate any fault pattern that occurs among m consecutive nodes on C_i^m, since some automorphism in $\mathrm{subgrp}(C_i^m)$ can map the m consecutive spare nodes to

---

Procedure CLUSTER_CIRCULANT(m, C_i);
begin
  Insert m consecutive spare nodes between any two nodes of C_i;
  $ds(C_i(m)) := ds(C_i) \hat{+} m$;
  for each node $x_{i,s}$ of C_i **do**
    Insert all edges e incident on $x_{i,s}$ such that $cd^+(e) \in ds(C_i(m)) - ds(C_i)$;
  for each spare node $s_{i,j}$ inserted **do**
    Insert all edges e incident on $s_{i,j}$ such that $cd^+(e) \in ds(C_i(m)$;
  $G'(C_i(m)) :=$ the resulting supergraph;
end. /* CLUSTER_CIRCULANT */

Figure 3. Algorithm for constructing the supergraph $C_i(m)$ of C_i.

any cluster of m consecutive nodes within which the faults occur. We call any fault pattern affecting f consecutive nodes (only the first and last nodes must be faulty) of a circulant graph an *f-fault cluster*. Similarly, a group of s consecutive spare nodes inserted between any two primary nodes of C_i is called an *s-spare* cluster. We denote by $G'(C_i(m_1, m_2, \ldots, m_p))$, or simply $C_i(m_1, m_2, \ldots, m_p)$, the circulant supergraph of $G(C_i)$ constructed by p applications of procedure CLUSTER_CIRCULANT, where we insert m_j-spare cluster in the j^{th} iteration, $1 \leq j \leq p$. We also denote its resulting single-cycle automorphism by $C_i^{m_1,\ldots,m_p}$. $C_i(m_1, m_2, \ldots, m_p)$ can tolerate p clusters of faults, where the j^{th} cluster contains at most m_j faults, i.e., it is an m_j-fault cluster. When $m_1 = m_2 = \cdots = m_p = m$, we will denote $C_i(m_1, m_2, \ldots, m_p)$ by $C_i([m]^p)$ and its automorphism $C_i^{m_1,\ldots,m_p}$ by $C_i^{[m]^p}$.

THEOREM 1. *The supergraph $C_i(m_1, \ldots, m_p)$ of C_i constructed by p calls to procedure CLUSTER_CIRCULANT can tolerate all m_i-fault clusters, $1 \leq i \leq p$.* ∎

The supergraph $C_i(5, 7)$ is constructed by inserting five spare nodes in C_i, adding the required spare edges to obtain $C_i(5)$, and then inserting seven spare nodes and additional spare edges. We can reverse this insertion process to reconfigure around a 7-fault cluster and obtain a fault-free copy of $C_i(5)$. This supergraph can be further reconfigured around any 5-fault cluster to obtain a fault-free copy of C_i. The question, however, remains: can $C_i(5, 7)$ be reconfigured around a 5-fault cluster that might occur before a 7-fault cluster to obtain a fault-free copy of $C_i(7)$? If this is indeed possible, then we can incrementally reconfigure $C_i(5, 7)$ around 5- and 7-fault clusters irrespective of their order of occurrence.

The question of the incremental reconfigurability of $C_i(5, 7)$ is equivalent to asking whether $C_i(5, 7)$ is isomorphic to $C_i(7, 5)$. The answer to this question is yes, since

$$ds(C_i(5,7)) = \big((1) \hat{+} 5\big) \hat{+} 7 = (1,6) \hat{+} 7 = (1,6,8,13) = (1,6,8,9)$$

(see footnote[1]), while

$$ds\,(C_i'(7,5)) = \big((1) \hat{+} 7\big) \hat{+} 5 = (1,8) \hat{+} 5 = (1,6,8,13) = (1,6,8,9).$$

Hence, their distance sequences are the same, and two circulant graphs with the same distance sequences and number of nodes are isomorphic. Theorem 2 establishes that incremental reconfiguration is also possible in the general case of $C_i(m_1, \ldots, m_p)$.

THEOREM 2. *It is possible to incrementally reconfigure $C_i(m_1, \ldots, m_p)$ around any order of occurrence of f-fault clusters, where $f \in \{m_1, \ldots, m_p\}$.* ∎

It can be shown that procedure CLUSTER_CIRCULANT introduces mk spare nodes and between k to kd spare links per node (i.e., a node degree of $d + k$ and $d + kd$, respectively) to construct $C_i([m]^k)$. The former case is optimal in the number of spare links. We will later present a switch implementation of the supergraphs constructed by procedure CLUSTER_CIRCULANT that reduces their node degrees to that of the basic graph, and also reduces their spare link complexity.

5. LOCAL-SPARING DESIGNS

Two cycles C_i and C_j in automorphism α of G are said to be *adjacent* if there is at least one edge between them. If no two cycles of α are adjacent, then we can make each cycle C_i tolerant of any desired fault pattern F_i using CLUSTER_CIRCULANT. The resulting supergraph G' automatically becomes tolerant of fault pattern F_i in each cycle C_i. However, if there exist

[1]Any *cd* d_i measured in the clockwise direction from node u that is greater than $\lfloor l/2 \rfloor$, where l is the number of nodes (including spares) of a cycle, has to be converted to a cd^- of $l - d_i$. In the above case, $l = 22$, and thus, a *cd* of 13 is converted to $22 - 13 = 9$.

adjacent cycles in α, then we have to introduce redundancy into the intercycle edges, so that G' can tolerate such fault patterns in each cycle. Here we discuss how spare intercycle edges are inserted, and how their number can be minimized by using spare clusters of specific sizes.

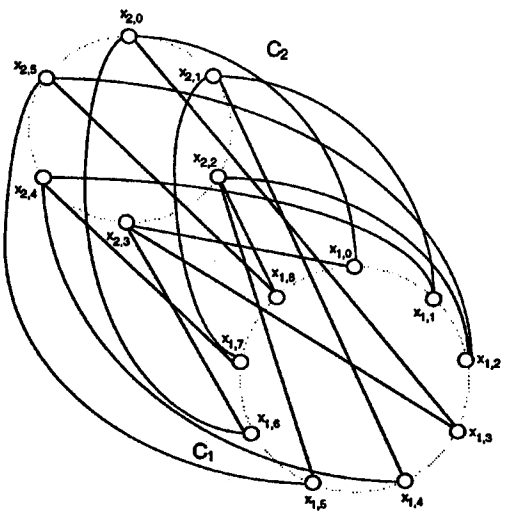
To simplify the discussion, suppose that we want to make each of the cycles C_i tolerant of any m_i-fault cluster. We can apply CLUSTER_CIRCULANT once for each cycle C_i to construct supergraphs $C_i(m_i)$ with automorphism $C_i^{m_i}$. Without loss of generality, we assume that the spare clusters are inserted in the cycles of α in the order of cycle numbers. As we will see later, the final supergraph constructed will be the same irrespective of the insertion order. Now, whenever we apply this construction process for any C_i, we should also add spare edges corresponding to the intercycle edges incident on C_i, if any, so that $\alpha_i' = C_1^{m_1}, \ldots, C_i^{m_i} C_{i+1}, \ldots, C_t$ becomes an automorphism of the partial supergraph G_i' of G constructed so far. In this way, we obtain a sequence of supergraphs G_i' of G with automorphisms α_i', for $1 \leq i \leq t$. The automorphisms in subgrp(α_i') can be used to reconfigure around any m_i-fault cluster in C_i to obtain a fault-free copy of G_{i-1}'.

5.1. Sample Construction

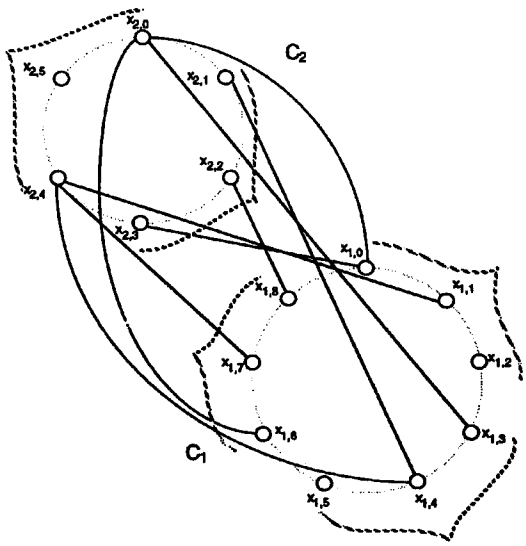
Consider the graph H in Figure 4a arranged according to the automorphism $\alpha = C_1 C_2$, where $C_1 = (x_{1,0}, \ldots, x_{1,8})$ and $C_2 = (x_{2,0}, \ldots, x_{2,5})$. The same graph is shown in Figure 4b, where, for clarity, only nodes $x_{2,0}$ and $x_{2,4}$ are shown with all their incident edges. Our aim is to insert a spare cluster of size $m_1 = 3$ in C_1 and some spare edges so that we obtain a supergraph with automorphism $\alpha_1' = C_1^3 C_2$. Subsequently, we want to insert a spare cluster of size $m_2 = 3$ in C_2 and more spare edges so that $\alpha_2' = C_1^3 C_2^3$ is an automorphism of the new supergraph, and is a super-automorphism of α_1'.

Let $g(C_i, C_j) = g_{i,j}$ be the greatest common divisor (gcd) of $|C_i|$ and $|C_j|$; thus, $g(C_1, C_2) = \gcd(6, 9) = 3$ in H. A partition $P_t(C_i)$ of a cycle C_i into subsets of consecutive t nodes, where t divides $|C_i|$, is called a *t-partition* of C_i. A 3-partition of both cycles of H is marked by braces around the subsets of nodes forming the partition in Figure 4b. It turns out that for any node in C_2, its neighbors in C_1 are in the same relative positions in each subset of a $g_{1,2}$-partition of C_1. For example, going clockwise around C_1, it can be seen that corresponding to the 3-partition $\{x_{1,0}, x_{1,1}, x_{1,2}\}$, $\{x_{1,3}, x_{1,4}, x_{1,5}\}$, $\{x_{1,6}, x_{1,7}, x_{1,8}\}$ of C_1 (shown by braces in Figure 4b), $x_{2,0}$ has the first nodes $x_{1,0}$, $x_{1,3}$, and $x_{1,6}$ in each subset as its neighbors, while $x_{2,4}$ has the second nodes $x_{1,1}$, $x_{1,4}$, and $x_{1,7}$ in each subset as its neighbors. Similarly, corresponding to the 3-partition of C_2 shown in Figure 4c (where edges incident on only nodes $x_{1,0}$, $x_{1,4}$, and $s_{1,1}$), $x_{1,0}$ has the third nodes in each subset as its neighbors, while $x_{1,4}$ has the first nodes in each subset as its neighbors. In the general case also, for any automorphism α of a graph G and any node $x_{i,s}$ in cycle C_i, there is a regular pattern of $x_{i,s}$'s neighbors in $\mathcal{P}$, where $\mathcal{P}$ is any $g(C_i, C_j)$-partition of an adjacent cycle C_j.

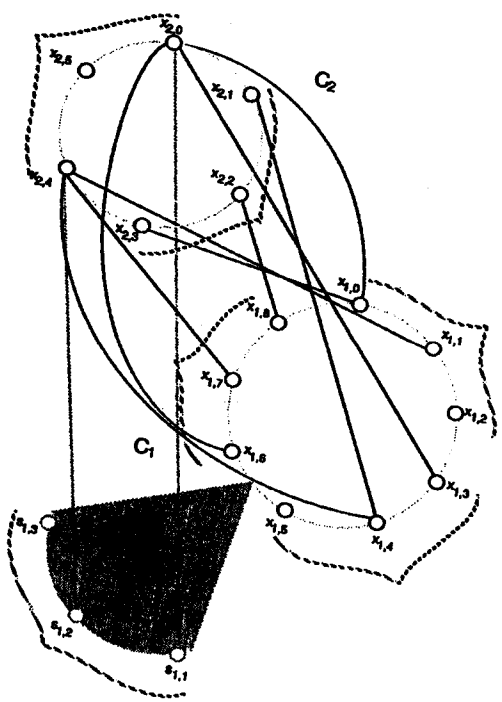
The above symmetry provides us with a means of incorporating fault tolerance in G so that relatively few spare intercycle edges are necessary. Before describing the design process further, we need to characterize the pattern of neighbors of a node $x_{i,s}$ of C_i in any subset $T = (x_{j,p}, x_{j,p+1}, \ldots, x_{j,p+t-1})$ of t consecutive nodes in an adjacent cycle C_j. This neighbor pattern is characterized by the ordered set $R = (b_2, b_1, \ldots, b_h)$, where $0 \leq b_1 < b_2 < \cdots < b_h < t$ are positive integers, and each $b_r = cw^+(x_{j,p}, x_{j,r})$, where $x_{j,p}$ is the first node of T, and $x_{j,r} \in T$ is a neighbor of $x_{i,s}$. The ordered set R is denoted by $np(x_{i,s}, T)$, for *neighbor pattern* of $x_{i,s}$ in T. Since the neighbor pattern of $x_{i,s}$ is the same in each subset of a $g(C_i, C_j)$-partition $P_{g(C_i,C_j)}(C_j)$, we denote this neighbor pattern by $np(x_{i,s}, P_{g(C_i,C_j)}(C_j))$. Thus in Figure 4b, $np(x_{2,0}, P_3(C_1)) = (0)$ and $np(x_{2,4}, P_3(C_1)) = (1)$, where $P_3(C_1)$ is the 3-partition of C_1 shown in Figure 4b. Theorem 3 formally establishes the above-described symmetry of intercycle neighbor patterns. We first give a useful lemma from number theory.



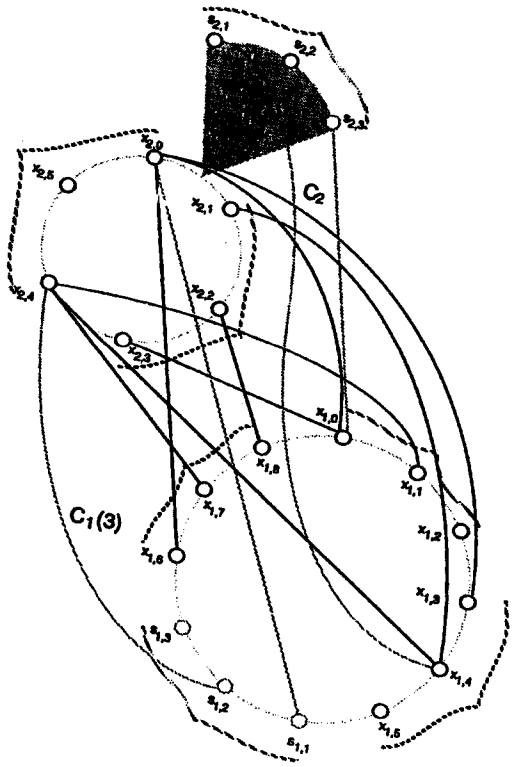
(a) H arranged by the cycles of its automorphism $\alpha = C_1\, C_2$.



(b) Part of H showing edges incident on nodes $x_{2,0}$ and $x_{2,4}$.



(c) Construction of the supergraph $H'(C_1(3), C_2)$.



(d) Construction of the supergraph $H'(C_1(3), C_2(3))$—only edges incident on nodes $x_{1,0}$, $x_{1,4}$, and $s_{1,1}$ are shown.

Figure 4.

LEMMA 1. *[21] If* $\gcd(k,m) = d$, *then the congruence* $kx \equiv l \pmod m$ *is solvable if and only if* $d \mid l$ *(*d *divides* l*). The congruence has then exactly* d *solutions.* ∎

THEOREM 3. *Let* G *be any graph, and let* $\alpha = C_1\, C_2, \ldots, C_t$ *be an automorphism of* G. *Further, suppose that* C_i *and* C_j *are any two adjacent cycles of* α, *and that* $\mathcal{P}$ *is any* $g(C_i, C_j)$*-partition of* C_j. *Then,* $np(x_{i,s}, T_1) = np(x_{i,s}, T_2)$ *for any node* $x_{i,s}$ *of* C_i *and any two subsets* T_2, T_2 *in* $\mathcal{P}$.

PROOF. Let $l_i = |C_i|$ and $l_j = |C_j|$. Further, let $np(x_{i,s}, T_1) = (b_1, b_2, \ldots, b_h)$, and let the first nodes of T_1 and T_2 be x_{j,p_1} and x_{j,p_2}, respectively. Since $\mathcal{P}$ is a $g(C_i, C_j)$-partition of C_j, $cd(x_{j,p_1}, x_{j,p_2}) = r\ g(C_i, C_j)$, for some integer $r \geq 1$. Now, x_{j,p_1+b_l} is a neighbor of $x_{i,s}$, for $1 \leq l \leq h$. For any integer d, α^d is also an automorphism of G. In particular, α^{xl_i} is an automorphism of G that maps node $x_{i,s}$ to itself ($(s+xl_i) \bmod l_i = s$), while it maps node x_{j,p_1+b_l} to node $x_{j,(p_1+b_l+xl_i) \bmod l_j}$. For simplicity, assume that $x_{j,0}$ is the first node of a subset in $\mathcal{P}$. It is easy to see that $(p_1 + b_l) \bmod l_j = p_1 + b_l$. Hence, $(p_1 + b_l + xl_i) \bmod l_j = p_1 + b_l + (xl_i \bmod l_j)$. From Lemma 1, there is a solution to the congruence

$$l_i x \equiv r\, g(C_i, C_j) (\bmod\ l_j),$$

i.e., there is an integer x such that α^{xl_i} maps x_{j,p_1+b_l} to $x_{j,p_1+r\,g(C_i,C_j)+b_l} = x_{j,p_2+b_l}$. Hence, x_{j,p_2+b_l} in T_2 is also a neighbor of $x_{i,s}$. This implies that $b_l \in np(x_{i,s}, T_1)$, and thus, that $np(x_{i,s}, T_1) \subseteq np(x_{i,s}, T_2)$. By symmetry, $np(x_{i,s}, T_2) \subseteq np(x_{i,s}, T_1)$, which proves the theorem. ∎

We now illustrate the FT design method for constructing the supergraphs of H (Figure 4a) with automorphisms $\alpha'_1 = C_1^3\, C_2$ and $\alpha'_2 = C_1^3\, C_2^3$. In Figure 4c, we insert the cluster S_1 of three spare nodes $\{s_{1,1}, s_{1,2}, s_{1,3}\}$, consecutively, on C_1 between the two braced subsets. The notation used for the spare nodes is that the first subscript is the cycle number i of C_i in which the spare nodes are inserted, while the second subscript l is the spare node number among those spare nodes that are inserted in C_i. After inserting the cluster S_1 of spare nodes in C_1, each node in C_2 is then connected to the spares so that $np(x_{2,s}, S_1) = np(x_{2,s}, P_3(C_1))$. Thus, $x_{2,0}$ is connected to the first spare node $s_{1,1}$, while $x_{2,4}$ is connected to the second spare $s_{1,2}$. As previously described, we also have to add spare edges within C_1, so that we construct a circulant supergraph $C_1(3)$ of C_1. However, in this case, neither cycles C_1 and C_2 have any intracycle edges, and we thus do not need to add any intracycle spare edges. The supergraph of H in Figure 4c has $\alpha'_1 = C_1^3\, C_2 = (x_{2,0}, \ldots, x_{2,5})(x_{1,0}, \ldots, x_{1,5}, s_{1,1}, \ldots, s_{1,3}, x_{1,6}, \ldots, x_{1,8})$ as an automorphism. We denote this supergraph by $H'(C_1(3), C_2)$. In general, the supergraph G' of G obtained by inserting an m_i-spare cluster in cycle C_i is denoted by $G'(C_1(m_1), \ldots, C_i(m_i), C_{i+1}, \ldots, C_t)$. The automorphism α'_1 of $H'(C_1(3), C_2)$, is called the *base* automorphism of $H'(C_1(3), C_2)$, and is denoted by $\alpha(C_1(3), C_2)$. Note the difference between the seed and base automorphisms of a FT supergraph. The seed automorphism is an automorphism α of G using which we construct a FT supergraph G'. The automorphism α' of G' that is obtained directly from α by inserting spare nodes in its cycles is the base automorphism of G'. Reconfiguration of G' around faults is done by using an automorphism in subgrp(α') of the base automorphism α' to map the spare node(s) of G' to the faulty one(s).

Going back to our example, we now insert spare nodes in cycle C_2 of H. At this point, $g(C_1(3), C_1) = \gcd(12, 6) = 6$. However, $np(u, T_1) = np(u, T_2)$ for any $u \in C_1(3)$, where T_1 and T_2 are any two subsets of a 3-partition of C_2. This follows from the manner in which we add intercycle spare edges between $C_1(3)$ and C_2 (for each node of C_2, we get the same neighbor pattern in all the subsets of any 3-partition of $C_1(3)$), and is established for the general case by Theorem 4. Note that $np(u, R_1) = np(u, R_2)$ is guaranteed by Theorem 3 for any two subsets R_1 and R_2 in any 6-partition of C_2. We now insert the cluster $S_1 = \{s_{2,1}, \ldots, s_{2,3}\}$ of three spare nodes between the two braced subsets of C_2, and again connect each node of $C_1(3)$ to S_1 so that its neighbor patterns in S_1 are the same as in the other two braced (primary)

subsets of $C_2(3)$. For example, $x_{1,0}$ and $s_{1,1}$ are connected to the first spare $s_{2,1}$, and $x_{1,4}$ is connected to the second spare $s_{2,2}$ in Figure 4d. We thus obtain the supergraph $H'(C_1(3), C_2(3))$ with base automorphism $\alpha(C_1(3), C_2(3)) = C_1^3 C_2^3 = (x_{1,0}, \ldots, x_{1,5}, s_{1,1}, \ldots, s_{1,3}, x_{1,6}, \ldots, x_{1,8})$ $(x_{2,0}, s_{2,1}, \ldots, s_{2,3}, x_{2,1}, \ldots, x_{2,5})$; only the edges incident on nodes $x_{2,0}$, $x_{2,4}$, $x_{1,0}$, and $x_{1,4}$ are shown in the figure.

The fact that $np(u, T_1) = np(u, T_2)$ for any u in $C_1(3)$ and any subsets T_1 and T_2 of a 3-partition of C_2, in spite of the fact that $g(C_1(3), C_2) = 6$, is stated for the general case in the next theorem.

THEOREM 4. *Let G be any graph and let C_i and C_j be adjacent cycles in any automorphism α of G. Suppose $m_j = p\ g(C_i, C_j)$ spare nodes are inserted in C_j to obtain the supergraph $G'(C_j(m_j), C_i)$ of $G(C_j, C_i)$, where $p \geq 1$.*

(a) *For any two subsets T_1 and T_2 of any $g(C_i, C_j)$-partition of $C_j(m_j)$ and any node $x_{i,s}$ of C_i, $np(x_{i,s}, T_1) = np(x_{i,s}, T_2)$.*
(b) *For any two subsets R_1 and R_2 of any $g(C_i, C_j)$-partition of C_i and any node $x_{j,t}$ of $C_j(m_j)$, $np(x_{j,t}, R_1) = np(x_{j,t}, R_2)$.*

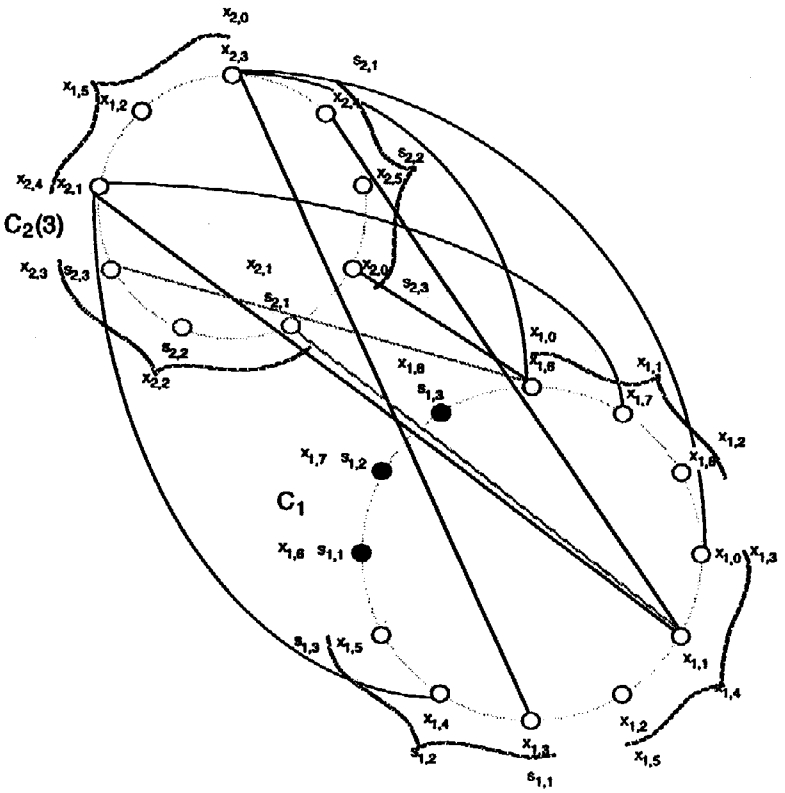
The base automorphism $\alpha(C_1(3), C_2(3))$ of $H'(C_1(3), C_2(3))$ (Figure 4d), or any power of this automorphism, can be used to map the 3-spare cluster of $C_2(3)$ to any 3-fault cluster in it, thus producing a fault-free copy of $H'(C_1(3), C_2)$. Subsequently, on the occurrence of a 3-fault cluster in $C_1(3)$, the base automorphism $\alpha(C_1(3), C_2)$ of $H'(C_1(3), C_2)$ can map the three spare nodes of $C_1(3)$ to the fault cluster to get a nonfaulty copy of H. We are once more faced with the question of whether such an incremental reconfiguration of $H'(C_1(3), C_2(3))$ can also be achieved when a 3-fault cluster first occurs in C_1 and then in C_2, i.e., when the order of the occurrence of the fault clusters is different from the order in which their associated spare clusters were inserted. The answer to this is also in the affirmative, if the number of spares inserted in any iteration in C_i is a multiple of $g_i = \text{lcm}\{g(C_i, C_j) : C_j \text{ is adjacent to } C_i\}$. This is obviously satisfied for $H'(C_1(3), C_2(3))$, and such an incremental reconfiguration is shown in Figures 5a and 5b. Note that in these figures, the braces are shifted so that each braced subset has the same labels as in the fault-free case. In Figure 5b, the node newly labeled as $x_{2,3}$, for example, has the same neighbor pattern in the new partition (reflected by the shifted braces), as the node previously labeled as $x_{2,3}$; this preservation of neighbor patterns after relabeling is the basis for correct reconfiguration. This is stated in Theorem 5, which establishes that as long as the reconfigured and original graphs have the same neighbor patterns between every pair of adjacent cycles, they are isomorphic. This is the central property that is used for designing and reconfiguring ALS-based FT supergraphs and their switch implementations.

Theorem 5 uses the following concept of "distance-from-zero". Given a "0^{th}" node $x_{i,0}$ in each cycle C_i of α, the other nodes in C_i are then labeled $x_{i,1}, \ldots, x_{i,l_i-1}$ going cw from $x_{i,0}$ on C_i. We define the *distance from zero* of a node $x_{i,s}$ of C_i in an adjacent cycle C_j, denoted by $dfz(x_{i,s}, C_j)$ as $dfz(x_{i,s}, C_j) = \min\{cd^+(x_{j,0}, x_{j,h}) \mid x_{j,h} \text{ is a neighbor of } x_{i,s}\}$, if the cd^+ between $x_{j,0}$ and at least one neighbor of $x_{i,s}$ is defined, otherwise, $dfz(x_{i,s}, C_j) = \min\{l_i - cd^-(x_{j,0}, x_{j,h}) \mid x_{j,h} \text{ is a neighbor of } x_{i,s}\}$. Hence, referring to Figure 4a, $dfz(x_{2,0}, C_1) = 0$, while $dfz(x_{2,5}, C_1) = 2$.

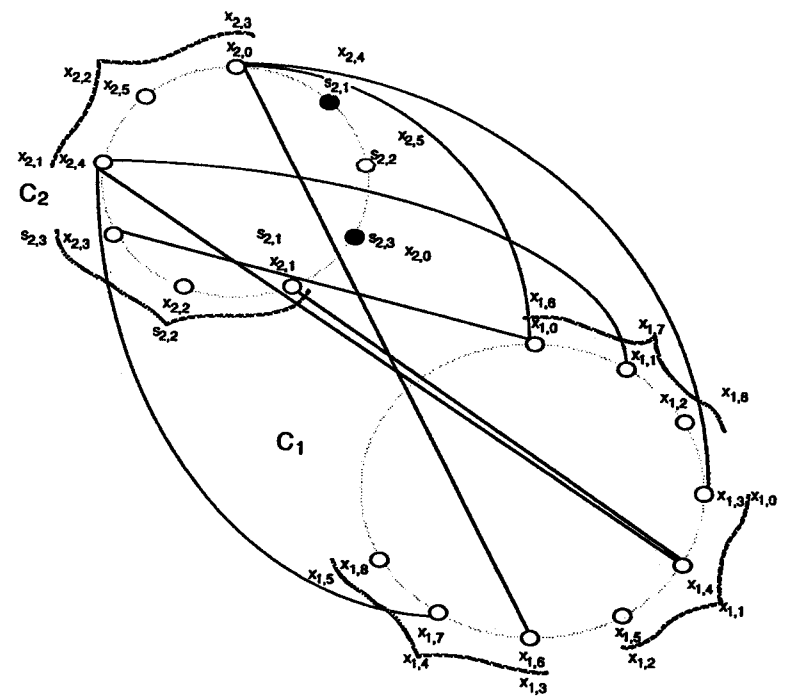
THEOREM 5. *Two graphs G_1 and G_2 are isomorphic if there exists an automorphism $\alpha_1 = C_1 C_2 \ldots C_t$ of G_1, and an automorphism $\alpha_2 = B_1 B_2 \ldots B_t$ of G_2, such that there is a bijection f between the cycles of α_1 and α_2 and a labeling of the 0^{th} nodes of each C_i and B_i satisfying:*

(1) $|C_i| = |f(C_i)|$,
(2) $ds(C_i) = ds(f(C_i))$,
(3) $np(C_i, C_j) = np(f(C_i), f(C_j))$, *and*
(4) $dfz(C_i, C_j) = dfz(f(C_i), f(C_j))$,

for all $i, j, 1 \leq i, j \leq t$. ∎



(a) Reconfiguration of $H'(C_1(3), C_2(3))$ around a 3-fault cluster in $C_1(3)$ into a fault-free copy of $H'(C_1, C_2(3))$.



(b) Reconfiguration of $H'(C_1, C_2(3))$ around a 3-fault cluster in $C_2(3)$ into a fault-free copy of H.

Figure 5.

In general, the question of incremental reconfiguration of the supergraph $G'(C_1(m_1),\ldots, C_t(m_t))$ of G, for any order of occurrence of the fault clusters, is also equivalent to asking whether $G'(C_1(m_1),\ldots,C_t(m_t))$ is isomorphic to $G'(\tau(C_1(m_1),\ldots,C_t(m_t)))$, where τ is any permutation of the set $\{C_1(m_1),\ldots,C_t(m_t)\}$. The latter question is answered in Theorem 7 in the next section.

5.2. General Construction Method

The method outlined above for constructing a FT supergraph of H by inserting $g_{1,2} = 3$ spare nodes in each cycle C_1 and C_2, can be extended to spare clusters of size of $3r$ for any $r > 1$. This is done by simply partitioning the $3r$ spares into groups of three in C_1 (C_2), and again connecting each node u (v) of C_2 (C_1) to the spare nodes in the other cycle so as to maintain u's (v's) neighbor pattern in that cycle. The supergraph of H constructed in this manner can tolerate a $3r$-fault cluster or, equivalently, r adjacent 3-fault clusters.

One reason for inserting spare cluster sizes of $g(C_1, C_2) = 3$ in each cycle of H is to facilitate incremental reconfiguration, which is not be possible if spare clusters of arbitrary sizes are used. Second, three is the smallest spare cluster size needed to minimize the number of spare intercycle edges between C_1 and C_2. For example, we introduce only one spare edge per node in the supergraph $H'(C_1(3), C_2(3))$. On the other hand, if we insert just one spare node in each cycle, then each node in $C_1(1)$ has to have edges to all nodes of $C_2(2)$, and vice versa, so that $C_1^1 C_2^1$ becomes an automorphism of $H'(C_1(1), C_2(1))$, resulting in a very expensive supergraph. This is formalized in the next theorem.

THEOREM 6. *If C_i and C_j are adjacent cycles of an automorphism α of any graph G, then the total number of edges to C_j required to be incident from each node of C_i when a cluster of m_j spares are introduced in C_j is* $\min(X_1, X_2)$, *where* $X_1 = l_j + m_j$ *and* $X_2 = b(l_j + m_j)/\gcd(g(C_i, C_j), m_j)$. *Furthermore, $bl_j/g(C_i, C_j)$ is the number of neighbors in C_j that each node of C_i has in the basic graph G, where $b \geq 1$.*

PROOF OUTLINE. It follows from the proof of Theorem 3 that if $x_{j,t}$ is a neighbor of $x_{i,s}$, then $x_{j,t+g(C_i,C_j)}$ is also a neighbor of $x_{i,s}$. Thus, the number of neighbors in C_j of a node in C_i is always a multiple of $l_j/g(C_i, C_j)$. If m_j nodes are introduced in C_j, then in order to have C_i and $C_j(m_j)$ as cycles in the super-automorphism of α in the resulting supergraph G', it is necessary that in G' every node $x_{j,r}$ be a neighbor of $x_{i,s}$, where $cd^+(x_{j,t}, x_{j,r}) = g(C_i, C_j(m_j))$, and $x_{j,t}$ is a neighbor of $x_{i,s}$ in G. Counting the total number of neighbors of $x_{i,s}$ that are introduced in this manner in $C_j(m_j)$ proves the theorem. ∎

Thus, it can be seen that for $m_j = 1$, each node of C_i has to have $l_j + 1$ edges to $C_j(1)$, while for $m_j = g(C_1, C_2)$ each node of C_i needs to have $bl_j/g(C_1, C_2) + b$ edges to C_j, i.e., only b spare edges, which is the minimum possible. Hence, there is a tradeoff between the number of spare nodes and the number of spare edges. In our construction process, we choose to have fewer spare edges at the expense of more spare nodes. In doing so, we also obtain the advantages of incremental reconfigurability, and higher reliability in situations where faults occur in clusters.

A cycle C_i of G can have more than one adjacent cycle C_j. In order to satisfy the multiple requirement of inserting $g(C_i, C_j)$ spare nodes in C_i, for each adjacent cycle C_j, we have to insert $g_i = \text{lcm}\{g(C_i, C_j) : C_j \text{ is adjacent to } C_i\}$ spare nodes in C_i. The best case occurs when g_i is small compared to l_i, the number of nodes in C_i; it can be seen that g_i is bounded above by l_i.

The procedure ALS of Figure 6 when given the graph G, a seed automorphism α of G, and the integer $p = 1$, inserts one cluster of g_i spare nodes in each cycle C_i of α and the required spare edges to construct a supergraph that can tolerate one g_i-fault cluster in each C_i. When $p > 1$, ALS repeats the above construction process p times to yield a supergraph that can tolerate p g_i-fault clusters in each cycle of α. We denote by $G'_{\text{als}}(p, G)$, the FT supergraph of G constructed by procedure ALS(G, α, p). ALS first determines the values of $g(C_i, C_j)$ and g_i for the cycles of α, and retains this information as the arrays $g[1..t, 1..t]$ and $g[1..t]$, respectively, and uses them in

```
Procedure ALS(G, α, p);
/* ALS constructs a supergraph G' of G that can tolerate p g_i-fault clusters in each subset C_i of the node-set
of G, where C_i is a cycle of the automorphism α = C_1 C_2 ... C_t of G. */
begin
    for each pair of adjacent cycles (C_i, C_j) of α do
        g[i,j] := g(C_i, C_j) = gcd(|C_i|, |C_j|);
    for each cycle C_i do
        g[i] := lcm{g[i,j] : C_j is adjacent to C_i};
    for m := 1 to p do
        /* Note that by Theorem 4, the arrays g[i] and g[i,j] remain the same in each iteration */
        (G', α') := CONSTRUCT_SUPERGRAPH(G, α, g[1..t, 1..t], g[1..t]);
        /* Construct a supergraph of G using seed automorphism α that can tolerate a g[i]-fault cluster in
        each cycle C_i */
        G := G'; α := α'; /* Rename the supergraph and its automorphism */
    endfor
    return(G', α')
end. /* ALS */

Procedure CONSTRUCT_SUPERGRAPH(G, α, g[1..t, 1..t], g[1..t]);
/* Construct a supergraph of G using seed automorphism α that can tolerate a g[i]-fault cluster in each cycle C_i */

/* For each pair C_i, C_j of adjacent cycles, there is a symmetry of neighbor patterns for nodes of C_j in
a g[i,j]-partition of C_i */
begin
    Let α = C_1, ..., C_t;
    for i := 1 to t do
        /* Insert the spare clusters in the cycles in the order C_1, ..., C_t */
        CLUSTER_CIRCULANT(g[i], C_i);
        /* Construct the circulant supergraph G'(C_i(g[i])) of G(C_i) with g[i] consecutive spare nodes */
        C_i^{g[i]} := (x_{i,0}, ..., x_{i,s}, s_{i,1}, ..., s_{i,g[i]}, x_{i,s+1}, ..., x_{i,l_i-1}); /* New automorphism of C_i(g_i) assuming
        that the spare nodes are inserted between primary nodes x_{i,s} and x_{i,s+1}. */
        for each C_j adjacent to C_i do
            Starting with a primary node that is adjacent in the cw direction to a spare node
            on C_i(g[i]), determine a g[i,j]-partition P_i of the nodes of C_i; /* g[i,j] divides g[i] */
            Partition the g[i] spare nodes into subsets S_{i,y} of consecutive g[i,j] spares on C_i(g[i]);
            for each primary and spare node (if already inserted) u of C_j do
                Connect u to each subset S_{i,y} of spare nodes so that
                np(u, S_{i,y}) = np(u, P_i);
            endfor;
        endfor;
    endfor;
    Let G' be the resulting supergraph, and α' := C_1^{g[1]} ... C_t^{g[t]} its automorphism;
    return (the final supergraph G', α' := C_1^{g[1]} ... C_t^{g[t]});
end. /* CONSTRUCT_SUPERGRAPH */
```

Figure 6. Algorithm for constructing a FT supergraph that tolerates p g_i-fault clusters in each cycle C_i of seed automorphism α.

each iteration. For each new iteration, the G and α variables are updated to the supergraph G' and its base automorphism α' constructed in the previous iteration. From Theorem 4, it follows that the values of the two arrays $g[i,j]$ and $g[i]$ do not have to be changed for each new iteration of ALS.

We thus conclude that procedure ALS constructs a supergraph $G'(C_1(g_1), \ldots, C_t(g_t))$ of G that can tolerate a g_i-fault cluster in each C_i of seed automorphism α. The next result addresses the question of incremental reconfiguration.

THEOREM 7. *Let G be any graph, and $\alpha = C_1\, C_2, \ldots, C_t$ be any automorphism of G. If each m_i is some multiple of $g_i = lcm\{g(C_i, C_j) : C_j$ is adjacent to $C_i\}$, for $i = 1, \ldots, t$, then the supergraphs $G'(C_1(m_1), \ldots, C_t(m_t))$ and $G'(\tau(C_1(m_1), \ldots, C_t(m_t)))$ are isomorphic, where τ is any permutation of the set $\{C_1(m_1), \ldots, C_t(m_t)\}$. It is thus possible to incrementally reconfigure $G'(C_1(m_1), \ldots, C_t(m_t))$ around any order of occurrence of m_i-fault clusters in cycle C_i.* ∎

In Section 6, we will present an incremental and localized reconfiguration algorithm LOCAL_AUTO _RECON for ALS-designed FT multiprocessors in Section 6. In addition to being able to reconfigure incrementally, it also localizes reconfiguration to the cycle in which the faults have occurred.

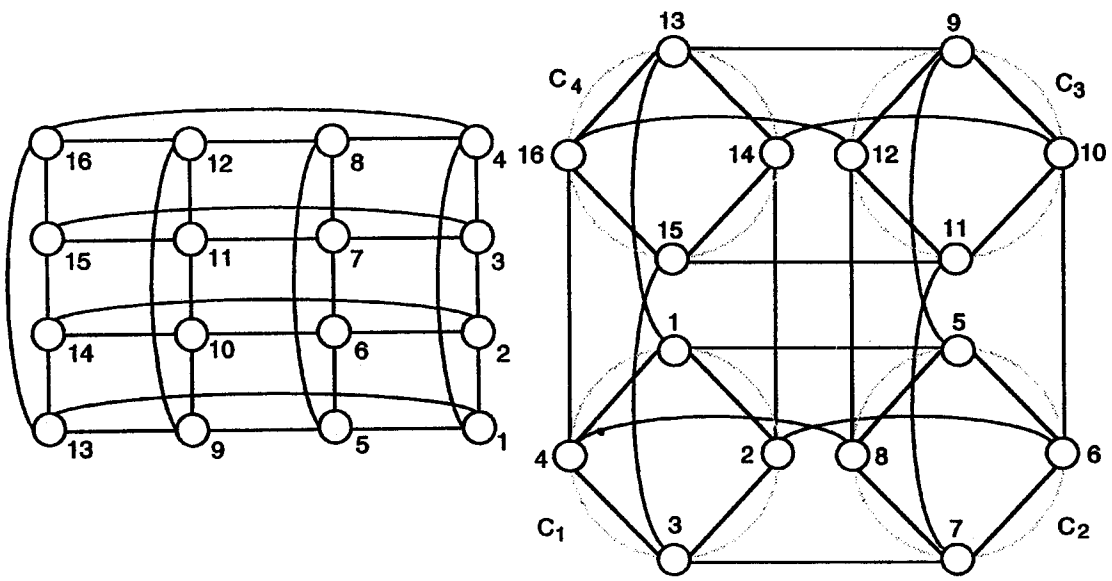
The next theorem establishes the spare-link complexity introduced by the ALS method.

THEOREM 8. *Let $\alpha = C_1 C_2, \ldots, C_t$ be the seed automorphism of G that procedure ALS uses to construct the supergraph $G'(C_1(g_1), \ldots, C_t(g_t))$, where g_i is as defined previously. Then, ALS introduces the following spare link complexities in G' (for $p = 1$).*

(a) *Each node of $C_i(g_i)$ has between k to kd_i intracycle spare links incident on it, where d_i is the intracycle node degree of C_i.*

(b) *Each node of $C_i(g_i)$ has $d_{i,j}g_j/l_j$ spare links incident on the nodes of an adjacent cycle $C_j(g_j)$, where $d_{i,j}$ is the number of links of G incident from a node in C_i to nodes in C_j. This is optimal in the number of intercycle spare links.* ∎

Hence, in the worst case that each C_i has the same number N/t of nodes, $g_i = N/t$, and the node degree in G' is double the node degree in G, and thus, the number of spare links is $3Nd/2$. Thus, we obtain 100% node redundancy, and a 300% link redundancy. This case occurs quite frequently for most multiprocessor graphs of interest, as will be demonstrated shortly. These worst-case redundancies for $G'_{\text{als}}(1, G)$ are the same as the redundancies obtained by incorporating dedicated sparing for each node (where each node has a spare exclusively for itself), and are smaller than those of the interstitial design of [14]. Moreover, ALS designs can actually be realized with much smaller spare link overhead than that stated above using a switch-based implementation that is presented in Section 7. Also, this switch implementation is equally efficient and allows incremental reconfiguration for arbitrary-size spare clusters, and thus, it is no longer necessary to have 100% node redundancy when all cycles have the same number N/t of nodes. We will later compare the reliabilities of ALS designs to those of dedicated-sparing and interstitial-redundancy designs.



(a) A 4 × 4 torus $T_{4,4}$.

(b) A layout of $T_{4,4}$ according to its decomposition into the cycles of automorphism $\gamma = C_1 \ldots C_4$.

Figure 7.

5.3. An Example

Here, we apply ALS to incorporate fault tolerance in two-dimensional toruses $T_{n,n}$, and n-dimensional hypercubes Q_n. We also illustrate reconfiguration of the resulting supergraphs around fault clusters.

Figure 7a shows a 4×4 torus $T_{4,4}$. As $T_{4,4}$ is isomorphic to Q_4, this example also serves to illustrate the application of ALS to hypercubes. The permutation $\gamma = C_1\,C_2\,C_3\,C_4$, is an automorphism of $T_{4,4}$, where each C_i is a column of the torus in top-down order.[2] Figure 7b shows an arrangement of the nodes of $T_{4,4}$ according to the cycles of γ. Figure 8a shows the supergraph $G'_{\text{als}}(1, T_{4,4})$ of $T_{4,4}$ constructed using γ as the seed automorphism. As in the previous case of $T(3,3)$, here too $g_i = l_i = 4$ for each of the four cycles C_i of γ, and we again require 100% node redundancy.

Figure 8b shows a 4-fault cluster $F_1 = (6, 7, 8, s_{2,1})$ in $C_2(4)$, and reconfiguration around it using the automorphism $(\gamma_4)^5$ in subgrp(γ_4) to map the 4-spare cluster $(s_{2,1}, \ldots, s_{2,4})$ to F_1. Here, γ_4 is the base automorphism $C_1(4)C_2(4)C_3(4)C_4(4)$. A fault-free copy of $G'(C_1(4), C_2, C_3(4), C_4(4))$ is obtained in this manner—the copy of $T_{4,4}$ in this fault-free supergraph is shown by dark edges. Since only one level of mapping appears in Figure 8b, the new label of each node is placed near the node, while the old label is placed away from it and in italics. Note that by Theorem 8, both $G'_{\text{als}}(T(3,3))$ and $G'_{\text{als}}(T_{4,4})$ illustrated in this section are optimal in the number of intercycle spare links.

We will later present a switch implementation of $G'_{\text{als}}(1, G)$ that reduces the node degree to that of G, and also significantly reduces the number and lengths of the spare links. As mentioned earlier, the switch implementation is also equally efficient for spare clusters of any arbitrary size. Thus, for cluster sizes of one, we obtain designs that tolerate independent faults in each cycle.

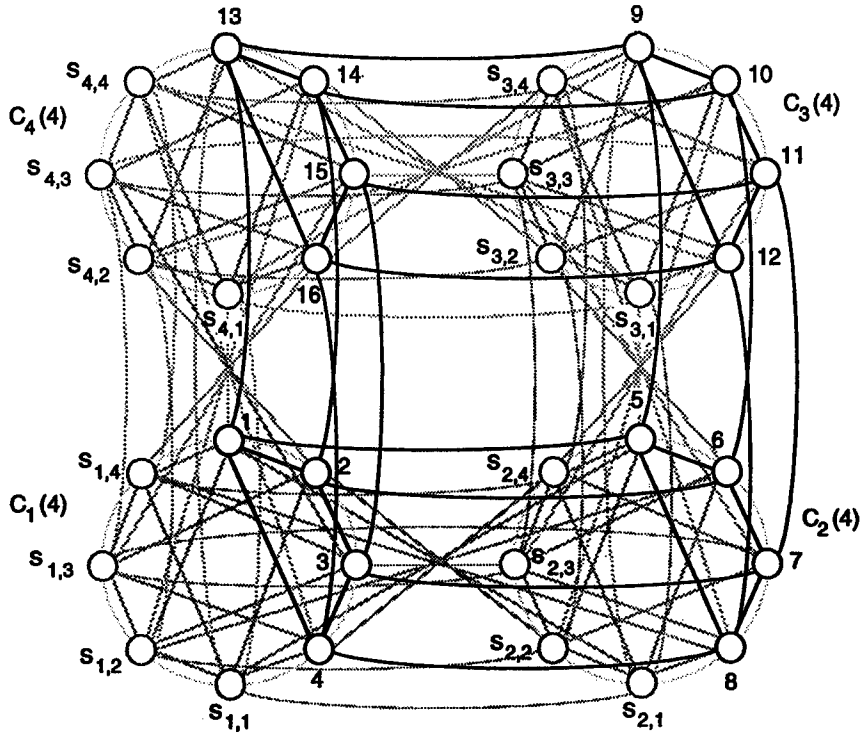
6. RECONFIGURATION

With local-sparing, it is desirable for reconfiguration to occur only in the faulty cycle without disturbing the processing in the other cycles. We next present a reconfiguration algorithm LOCAL_AUTO_RECON with this property.

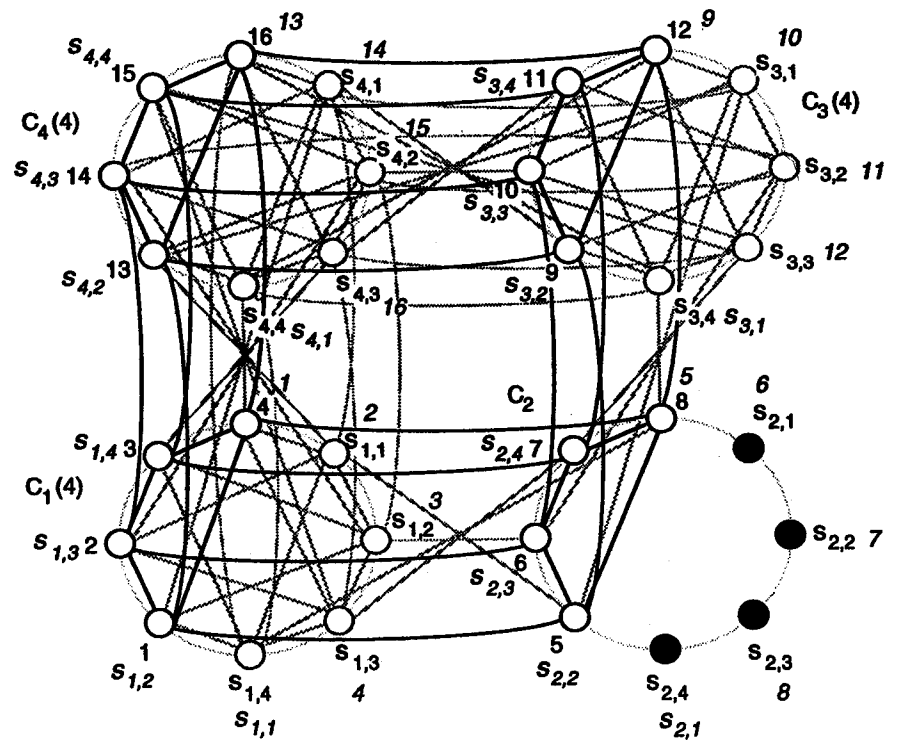
6.1. Localized Reconfiguration

At first sight, it might seem that localized reconfiguration is not possible in ALS designs, since the automorphism $(\alpha')^z$ used for reconfiguration relabels nodes in every cycle. Localized reconfiguration is, however, possible in such designs. Consider again the reconfiguration we described for the torus of Figure 8b with one 4-fault cluster in cycle C_2. The automorphism $(\gamma_4)^5$ is used to obtain a fault-free copy of $G'(C_1(4), C_2, C_3(4), C_4(4))$, whose base automorphism is $\gamma_3 = C_1(4)\,C_2\,C_3(4)\,C_4(4)$. Now, without any more faults occurring, if automorphism $(\gamma_3)^{-5}$ is used to relabel the nodes in the nonfaulty graph, then the nodes of the fault-free cycles $C_1(4)$, $C_3(4)$, and $C_4(4)$ are relabeled by their original labels before the 4-fault cluster occurred, while only the nodes of the faulty cycle C_2 get new labels. In effect, the node labels and thus the nodes of the nonfaulty cycles are undisturbed. This example of localized reconfiguration can be extrapolated to the following algorithm, which is easily shown to hold for the general case. Let α' be the base automorphism of the entire FT system. When a fault cluster occurs in a cycle C_i, determine z

[2] In the case of Q_4, each C_i of the above automorphism γ is a two-dimensional subcube Q_2. In the general case of Q_n, the permutation whose cycles are disjoint two-dimensional subcubes is an automorphism. If larger cycles are desired, they can be obtained by generating automorphisms of Q_n by bit permutation and complementation of the node labels as explained in [19,20]. However, the cycles obtained in this way will not necessarily have similar structures and cardinalities, as is the case with the cycles of γ. If regularity is essential to realizing the FT supergraph in an efficient and modular fashion, then Q_n can be partitioned into smaller Q_ms ($m < n$), and each Q_m be made circulant as described in [19,20]. An automorphism of this extended Q_n can be shown to be one in which each cycle is a disjoint circulant Q_m, and can thus serve as a good seed automorphism for the final FT design.



(a) The FT supergraph $G'_{\text{als}}(1, T_{4,4})$ of $T_{4,4}$.



(b) Reconfiguration around the 4-fault cluster $(6, 7, 8, s_{2,1})$ in $C_2(4)$.

Figure 8.

so that $(\alpha')^z$ is the reconfiguring automorphism, and let α'_1 be the base automorphism of the reconfigured system (z is essentially the cd^+ between the first node in the spare cluster and the

first node in the fault cluster). Relabel only the nodes of the faulty cycle C_i by first applying $(\alpha')^z$ in C_i, and then $(\alpha'_1)^{-z}$ in the reconfigured C_i that does not contain the fault cluster. Since the nodes of the nonfaulty cycles will get back their original labels by application of these two automorphisms, it is not necessary to relabel them. This modified reconfiguration algorithm LOCAL_AUTO_RECON is given formally in Figure 9. It is easy to determine z in a distributed manner in the faulty cycle. In a hypercube, for example, if each C_i represents an augmented subcube (see Section 5.3), then z can be determined in $\log(l_i+g_i) = \Theta(\log l_i)$ time, where $l_i = |C_i|$. In the worst case, z can be determined in $\Theta(l_i)$ time for any architecture. It is then a simple matter for each processor in $C_i(g_i)$ to relabel itself. Finally, in the switch implementation presented next, the switches can be reconfigured in time $\Theta(d)$ in a distributed manner.

Procedure LOCAL_AUTO_RECON($G', \alpha', \{x_{i,s}, x_{i,s+1}, \ldots, x_{i,s+g_i-1}\}$);
/* LOCAL_AUTO_RECON will reconfigure the supergraph G' with base automorphism α' for one fault cluster $\{x_{i,s}, x_{i,s+1}, \ldots, x_{i,s+g_i-1}\}$ in cycle C'_i by relabeling only the nodes of C'_i. For multiple fault-clusters, LOCAL_AUTO_RECON can be repeatedly called until G' is completely reconfigured. */
begin
Identify a cluster $\{s_{i,l}, \ldots s_{i,l+g_i-1}\}$ of g_i spare nodes in C'_i.
if such a cluster of spares does not exist
then return("error", "no more spares in C'_i")
else begin
Let $cd^-(s_{i,l}, x_{i,s}) = t$;
for all nodes u in C'_i **do** Relabel u by $(\alpha')^z(u)$;
/* Only the nodes of C'_i are relabeled */
Delete the nodes now labeled as the spares $\{s_{i,l}, \ldots s_{i,l+g_i-1}\}$ and all edges incident on them from G';
Delete the (newly labeled) spare nodes $\{s_{i,l}, \ldots s_{i,l+g_i-1}\}$ from cycle C'_i;
In α'_1 be the automorphism (of the reconfigured subgraph) obtained by replacing the old C'_i in α' by the new one obtained after the above deletions;
/* Now apply $(\alpha'_1)^{-z}$ to the nodes of the new C'_i */
for all nodes u in the new C'_i **do** Relabel u by $(\alpha'_1)^{-z}(u)$;
return("success", G', α'_1);
end; /* **else** */
end. /* LOCAL_AUTO_RECON */

Figure 9. A reconfiguration algorithm for ALS designs that relabels only the nodes of the faulty cycle.

7. SWITCH IMPLEMENTATION

Here we present a switch implementation method ALS_SW for ALS designs that takes advantage of the localized reconfiguration algorithm LOCAL_AUTO_RECON.

7.1. Switch Implementation for Intracycle Edges

We saw in Section 4 that if p spare clusters are added to a cycle C_i, then its resulting supercycle has edges with *cds* in the set $(\cdots(\{d_j\} \hat{+} m_1)\cdots \hat{+} m_1) = d_j, d_j + m_1, d_j + 2m_1, \ldots, d_j + pm_1$, etc., for every edge of *cd* d_j in C_i. Note that only one of the above spare edges will be used after reconfiguration around any set of the tolerated fault clusters to obtain an edge with *cd* d_j. Thus, a hierarchical switching structure using 1-to-2 demultiplexers (demuxes) can be designed that can

connect any link of $cd\ d_j$ from a node u to a subset of other processors in order to realize a link in $C_i(m_1, \ldots, m_1)$ with a cd in the set $\{d_j, d_j + m_1, \ldots, d_j + pm_1\}$ $(\cdots(\{d_j\} \hat{+} m_1) \cdots \hat{+} m_p)$. Such a switching structure is shown in Figure 10 for $p = 2$ in which two 1-to-2 demuxes are used to switch a link from processor u to connect it to a processor at a distance of either $cd\ d_j$, $d_j + m_1$ or $d_j + 2m_1$. The general switch implementation procedure (which is a part of ALS_SW) for intracycle links when p m-spare clusters are inserted in C_i is given as follows.

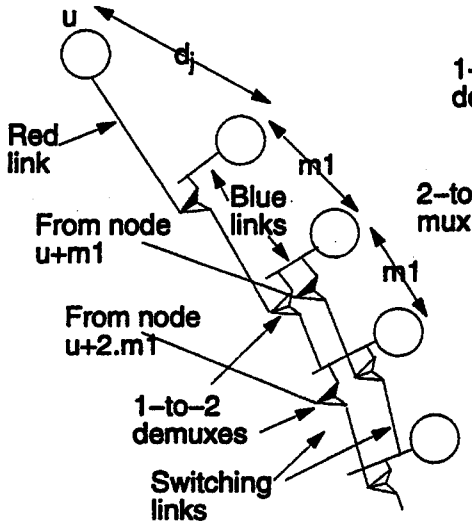
1. Attach (bidirectional) links to the d_i I/O ports of each primary and spare processor, and divide these links into $\lfloor d_i/2 \rfloor$ red and $\lfloor d_i/2 \rfloor$ blue links (d_i is the intracycle node degree of C_i). If d_i is odd, then one link has been left out; include it as a red link for the first $l_i/2$ nodes (in order) in C_i, and as a blue link in the other $l_i/2$—note that if d_i is odd, then l_i has to be even, since $d_i l_i$ is always even. For each node, label its red links by a unique cd in $ds(C_i)$, and similarly label its blue links. If d_i is odd, then $|ds(C_i)| = \lceil d_i/2 \rceil$ is one greater than either the number of blue or the number of red links. In that case, do not label the red or blue links, whichever is fewer, by the largest cd. Also note that when d_i is odd, the largest cd in $ds(C_i)$ is always $l_i/2$.
2. Connect each red link labeled d_j from a processor u to the input of a 1-to-$(p+1)$ demux, and connect the outputs of this demux to the blue links labeled d_j of the $p+1$ processors that are at cds of $d_j, d_j + m, d_j + 2m, \ldots, d_j + pm$ from processor u in $C_i([m]^p)$.

Note that $p+1$ outputs from $p+1$ different demuxes will connect to any blue link of a processor, and that exactly one of these outputs will be active and the others disabled, if the processor is nonfaulty and not a spare in the reconfigured labeling; see Figure 10. If the processor is either faulty or a spare, then all connections to its blue links are disabled. Similarly, the demuxes whose inputs are connected to red links from spare or faulty processors will have all their outputs disabled. A 1-to-$(p+1)$ demux can be implemented by p 1-to-2 demuxes, with one output of a demux connected to the input of the next as shown in Figure 10 for $p = 2$. Hence, a switch implementation of $C_i([m]^p)$ requires pe_i 1-to-2 demuxes and $(p-1)e_i$ links (called *switching links*) to interconnect subsets of p demuxes, where e_i is the number of intracycle links in C_i.

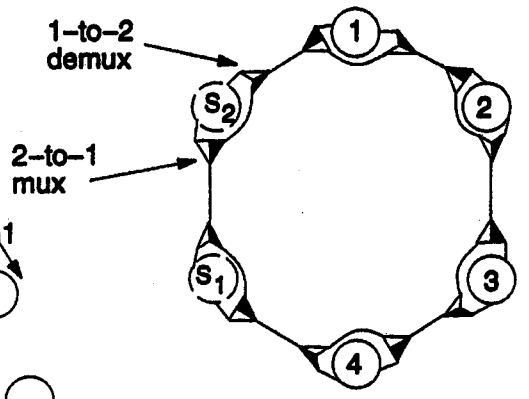
In the special case that the subgraph C_i is a cycle graph, bypassing switches using mux-demux pairs can be used to form a cyclic connection of the nonfaulty nodes by bypassing the faulty ones; see Figure 10b. Figure 10c shows the reprogramming of some mux-demux pairs to bypass a 2-fault cluster. In this case, n_i 2-to-1 muxes and n_i 1-to-2 demuxes are needed, where $n_i = l_i + mp$ is the number of nodes in $C_i([m]^p)$.

7.2. Switch Implementation for Intercycle Links

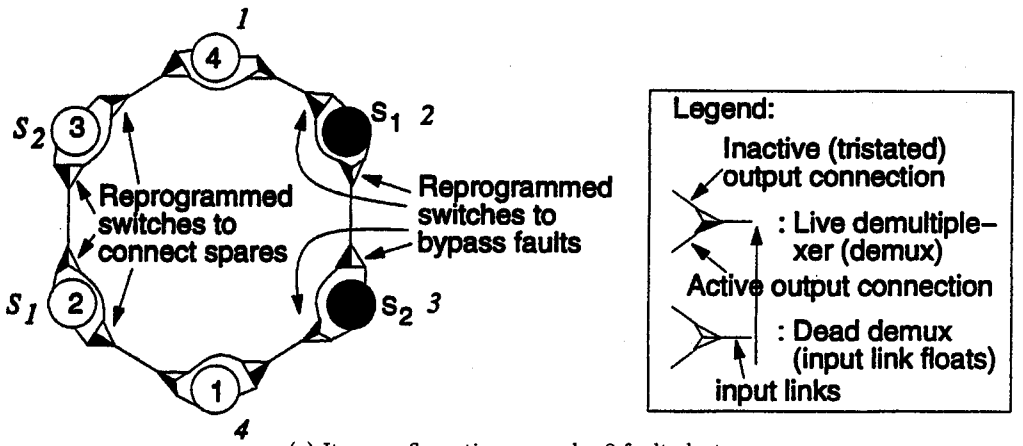
If spare clusters of size g_i are inserted in cycle C_i, then in the switching method to realize the resulting FT supergraph, each intercycle link incident on a primary node $x_{i,s}$ of C_i is connected to $x_{i,s}$ via a 1-to-2 demux, with the second output of the demux going to a node at a cd^- of g_i from $x_{i,s}$ in $C_i(g_i)$. Similarly, if this link is incident at node $x_{j,t}$ in an adjacent cycle C_j, then it is also connected at $x_{j,t}$'s I/O port via a 1-to-2 demux, with the second output of the demux going to a node at a cd^- of g_j from $x_{j,t}$ in $C_j(g_j)$. When $g_i = g(C_i, C_j)$, then it is easy to see that this switch implementation for intercycle links between C_i and C_j realizes the supergraph between cycles C_i and C_j constructed by the ALS method. When a g_i-fault cluster occurs in $C_i(g_i)$, define a $P_{g_i,j}$-partition in which the fault cluster is a subset. For reconfiguration, all intercycle links incident on nodes that either are in the fault cluster or are between the fault cluster and the spare cluster going in the ccw direction are switched to nodes at a cd^- of $g_i = g_{i,j}$ away. Thus, links from one subset of the $P_{g_i,j}$-partition are switched to an adjacent subset until the spare subset is switched in; the neighbor pattern for any node in C_j for the above $P_{g_i,j}$-partition is maintained after the switching, and thus from Theorem 5, the system is correctly reconfigured. Note that LOCAL_AUTO_RECON only relabels the nodes of the faulty cycle $C_i(g_i)$; the labels of all other nodes remain unchanged, and hence, switching need only take place for intercycle



(a) Switching structures for intracycle links for two m-spare clusters.



(b) Switch implementation of a 2-cluster FT cycle graph.



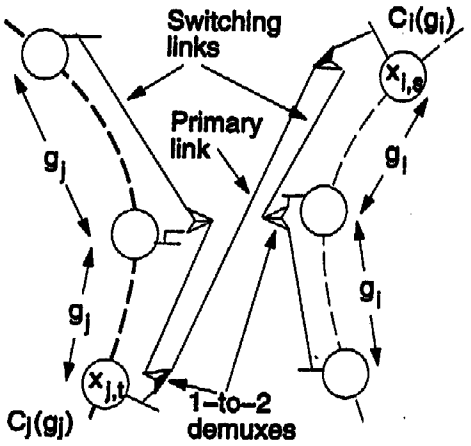
(c) Its reconfiguration around a 2-fault cluster.

Figure 10.

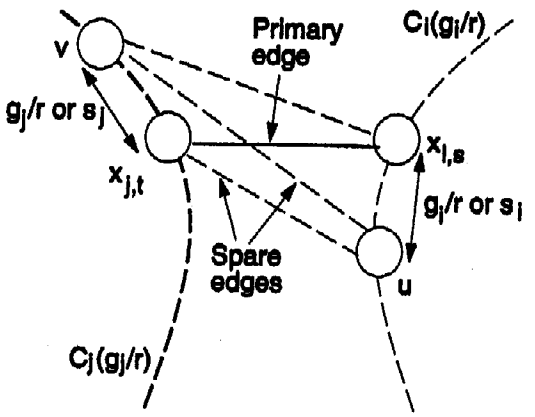
links incident on C_i, and only for the demuxes at C_i's end. When a fault cluster occurs later in $C_j(g_j)$, the intercycle links incident on this cycle are switched in a similar manner. This switch implementation also applies when $g_i = mg_{i,j}$ for $m > 1$.

Furthermore, when $p > 1$ clusters of g_i spare nodes are inserted in each cycle C_i, the above switch implementation given for $p = 1$ is easily extended to the $p > 1$ case by connecting each intercycle link incident at node $x_{i,s}$ to nodes at cd^-s of $g_i, 2g_i, \ldots, pg_i$ from $x_{i,s}$ via a 1-to-$(p+1)$ demux; this is shown in Figure 11a for $p = 2$. Figure 12b shows the switch implementation of the supergraph $G'_{\text{als}}(1, T(3,3))$ of the tree $T(3,3)$ (Figure 12a).

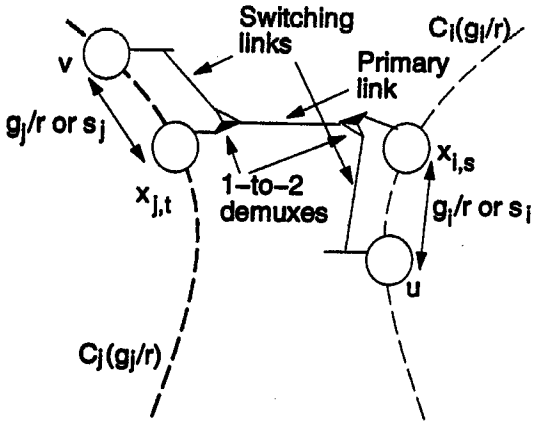
It can be shown using Theorem 6, that inserting spare cluster sizes of g_i/r (for $r > 1$) in C_i increases the number of spare intercycle links. This increase is primarily because we insist that the permutation $\alpha' = C_1^{g_1/r} \ldots C_t^{g_t/r}$ be an automorphism of the corresponding supergraph G'. However, strictly speaking, it is not necessary for α' to be an automorphism for correct reconfiguration, and thus, all the spare edges postulated in Theorem 6 are not needed. In fact, if we consider α' as a base permutation (instead of a base automorphism) of G' that is used by LOCAL_AUTO_RECON to relabel the nodes when a fault cluster occurs, then many fewer intercycle spare edges are required for correct reconfiguration. Specifically, for each intercycle edge $(x_{i,s}, x_{j,t})$, the only spare edges needed in this case are the following.



(a) Switch implementation for intercycle links when two g_i-spare clusters are inserted in each C_i.



(b) Spare links required for primary link $(x_{i,s}, x_{j,t})$ when one (g_i/r)-spare cluster or an s_i-spare cluster for arbitrary s_i is inserted in each C_i.



(c) The switch implementation that realizes these spare links.

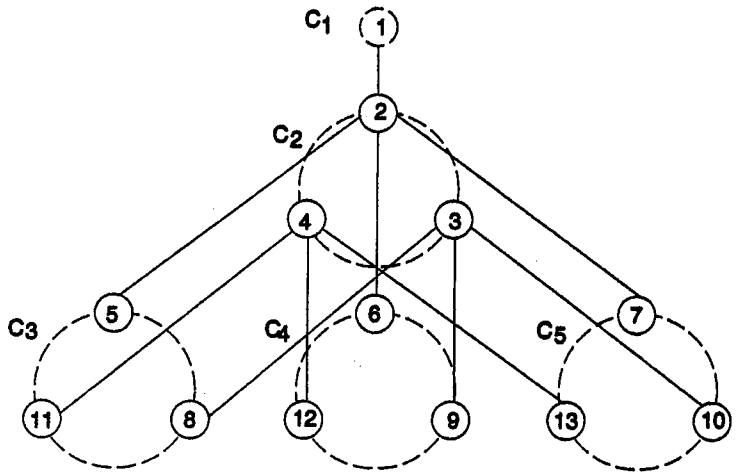
Figure 11.

(1) One from $x_{i,s}$ to the node u at a cd^- of g_j/r from $x_{j,t}$ in $C_j(g_j/r)$.
(2) One spare edge from $x_{j,t}$ to the node v at a cd^- of g_i/r from $x_{i,s}$.
(3) A third spare edge between u and v.

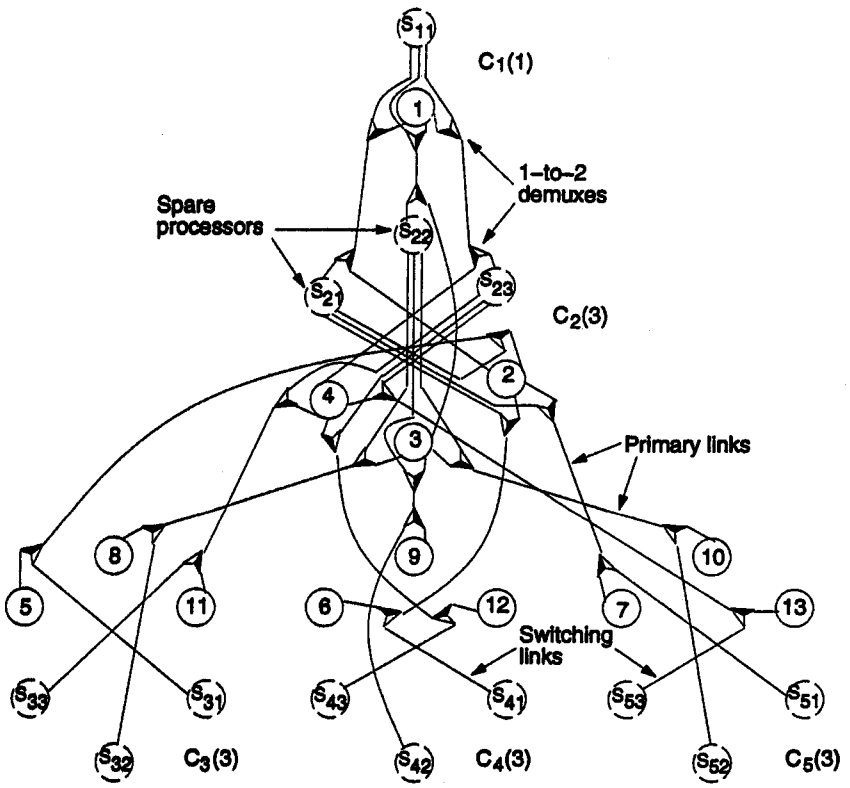
These spare edges are shown in Figure 11b. Figure 11c depicts the switch implementation for realizing these spare links using 1-to-2 demuxes. The above switch implementation also applies when the spare cluster size is some arbitrary number s_i. Thus for $s_i = 1$, the design tolerates independent processor faults.

The switch implementation part of ALS_SW for intercycle links when s_i-spare clusters are inserted in cycles C_i (for arbitrary s_i) is as follows.

1. For each node $x_{i,s}$ of each cycle C_i, label $x_{i,s}$'s I/O ports that connect to intercycle links by (j, y), if the port connects to a node in an adjacent cycle C_j, where y ranges from 1 to $d_{i,j}$ ($d_{i,j}$ is the number of neighbors in C_j for each node in C_i). Connect a bidirectional link to each intercycle port.
2. For each intercycle link $(x_{i,s}, x_{j,t})$, connect the link at both ends to the input of a 1-to-2 demux. At $x_{i,s}$'s end, connect one output of the demux to the link attached to a unique intercycle I/O port of $x_{i,s}$ labeled by some (j, y), and the other output of the demux to a similarly labeled I/O port link of the node at a cd^- of s_i from $x_{i,s}$ in $C_i(s_i)$. Connect



(a) Layout of tree $T(3,3)$ according to the cycles of automorphism $\beta = (1)(2,3,4)(5,8,11)(6,9,12)(7,10,13)$.



(b) Switch implementation of the FT tree $T(3,3)$ $G'_{\text{als}}(1, T_{3,3})$ constructed by procedure ALS.

Figure 12.

the outputs of the demux at $x_{j,t}$'s end similarly to $x_{j,t}$ and a node at a cd^- of s_j from it in $C_j(s_j)$.

THEOREM 9. *When one s_i-spare cluster is inserted in each cycle C_i (for arbitrary s_i), the switch implementation method ALS_SW for intra- and intercycle links realizes a design that can tolerate an s_i-fault cluster in each C_i. Similarly, for p insertions of spare clusters in each cycle, the ALS_SW design tolerates p s_i-fault clusters in each C_i.* ∎

PROOF. The reason that the above-mentioned set of spare links, and hence, also the switch implementation of Figure 11c, are sufficient for correct reconfiguration is as follows. When $(\alpha')^z$ is used by LOCAL_AUTO_RECON for node relabeling to reconfigure around an s_i-fault cluster in C_i, followed by $(\alpha_1')^{-z}$, all nodes in the nonfaulty cycles retain their original labels, while nodes of C_i either retain their original label or get a label of the node at a cd of $s_i r$ from it. Hence, for link $(x_{i,s}, x_{j,t})$, the label of one end node $x_{j,t}$ remains unchanged, while either the label of $x_{i,s}$ is unchanged or the node v at a cd^- of s_i from $x_{i,s}$ is newly labeled as $x_{i,s}$. This requires the link to be switched to the node newly labeled as $x_{i,s}$ which is accomplished by the 1-to-2 demux at $x_{i,s}$'s end. Similarly, when a s_j-fault cluster occurs in C_j, the demux at $x_{j,t}$'s end might need to connect the $(x_{i,s}, x_{j,t})$ link to either $x_{j,t}$ or the node u at a cd^- of s_j from it; the latter is newly labeled as $x_{j,t}$. ∎

Note that when $s_i = g_i$ for all cycles C_i, the supergraph depicted by Figure 11b is exactly the one constructed by procedure ALS—either the node v at a cd^- of g_i from $x_{i,s}$ is a primary node, in which case the required link from $x_{j,t}$ to v is already present, or v is a spare, in which case this link is added by ALS. Figure 13 shows the above switch implementation when 2-spare clusters are inserted in the cycles of the automorphism of the 4 × 4 torus of Figure 7.

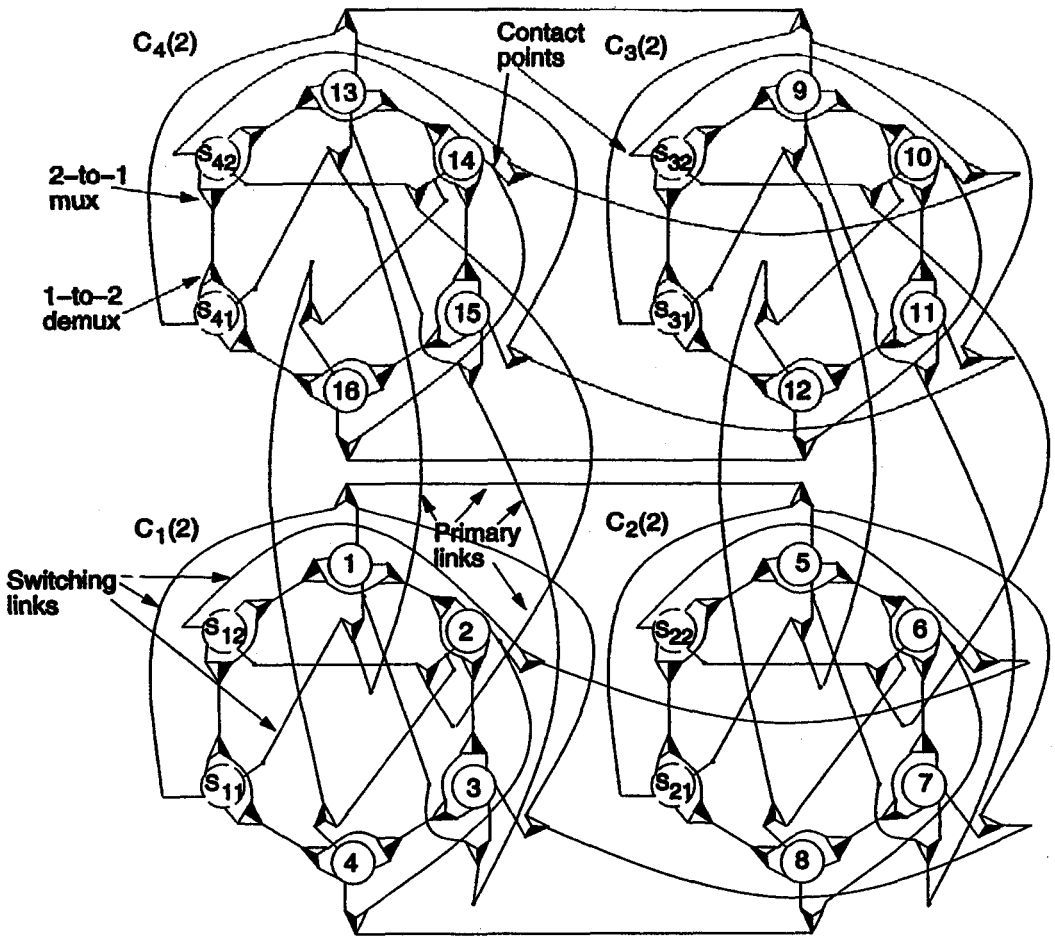


Figure 13. Switch implementation to tolerate a 2-fault cluster in each of the four cycles of automorphism γ (specified earlier) of a 4 × 4 torus (or a four-dimensional hypercube).

When $p > 1$ s_i-spare clusters are inserted in each C_i, the above switch implementation extends to connecting each intercycle link via 1-to-$(p+1)$ demuxes at both ends to its original end nodes, and to nodes at cd^-s of s_i $(s_j), 2s_i$ $(2s_j), \ldots, ps_i$ (ps_j) from them at C_i's (respectively, C_j's) end. Thus, the redundant hardware required comprises $2pe_2$ 1-to-2 demuxes and $2(p-1)e_2$ short

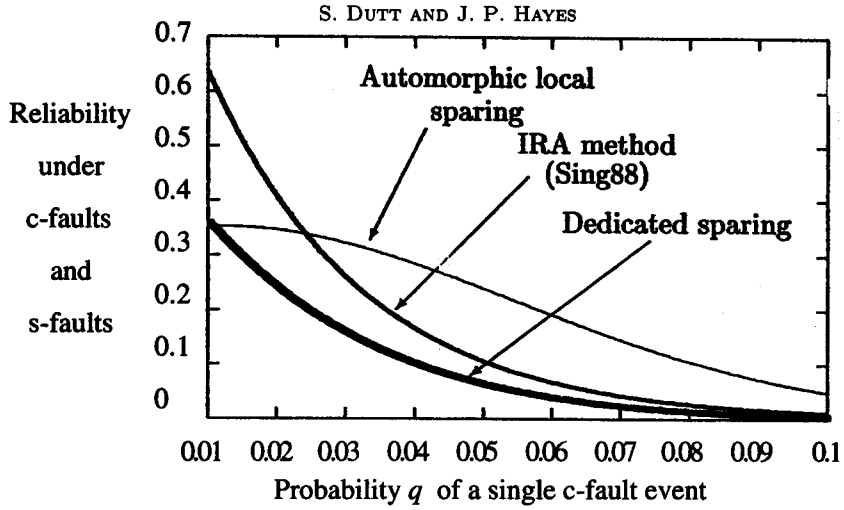


Figure 14. Reliability comparison of three FT 8 × 8 torus designs under c-fault and s-fault events, satisfying $p + q = 0.1$.

switching links that interconnect some of these demuxes, where e_2 is the number of intercycle links of G.

8. RELIABILITY ESTIMATION

Figure 14 compares the reliability of three designs, dedicated sparing, interstitial redundancy (IRA) [14], and ALS applied to an 8 × 8 torus $T_{8,8}$ under the dual fault model, i.e., when both c-fault events (which cause clustered faults) and s-fault events (which cause only single processor faults) can occur. This fault model and the derivation of the reliabilities of these designs under this model has been extensively developed in [22]. It is assumed in Figure 14 that the sum of the probabilities of single c-fault and s-fault events, q and p, respectively, is a constant 0.1, i.e., $p + q = 0.1$. The reliabilities are plotted against q. For values of q less than 0.025, the IRA design has the best reliability, while for higher values of q the ALS design's reliability is better than the other two designs by factors of two to three. Thus Figure 14 demonstrates that $G'_{\text{als}}(1, T_{8,8})$ is a much more reliable design when the values of q and p are comparable, and also when q is greater than p. It is worth noting that the IRA design for $T_{8,8}$ actually has more than 100% node redundancy (81 spare and 64 primary nodes), and also that it has an edge redundancy of 500%. In comparison, the other two designs have exactly 100% node and 300% edge redundancies. Furthermore, by using the switch implementation described in Section 7 to realize $G'_{\text{als}}(1, T_{8,8})$, its redundancy reduces even further. Finally, it is necessary in actual ALS designs to implement the cycles and their associated spares as separate units, either on VLSI chips, multichip modules, or printed circuit boards.

There should also be some form of electromagnetic shielding between these units so as to localize the effect of a c-fault causing event like radiation to one or a few cycles only. For a similar reason, it will be helpful to have separate power sources for each unit, since power fluctuations are also c-fault causing events. Adjacent processors in a cycle should also be physically adjacent in the the unit in which the cycle is realized. Thus, when processors fail in clusters in a unit, the fault cluster will correspond to a cluster of adjacent processors on the cycle, around which ALS designs can reconfigure.

9. CONCLUSIONS

We have developed an automorphism-based methodology called ALS to systematically incorporate redundancy in multiprocessor systems using local groups of spare processors for tolerating either clusters of faulty processors or independent single faults. A cluster-FT design is more suitable in a hazardous environment in which processors can fail in clusters (due to frequent exposure

to radiation, for example). This method uses a suitable seed automorphism of the graph representing the basic multiprocessor in order to partition it into smaller subsystems with which local spares are associated. While it can be difficult to find an automorphism of an arbitrary graph (to serve as a seed for ALS designs), it is, however, relatively easy to find automorphisms of common multiprocessor interconnections like hypercubes, toruses, trees, and general k-ary d-dimensional cubes, as demonstrated in Section 5.3.

ALS designs use an optimal number of intercycle spare links when the spare clusters in each cycle C_i are of specific sizes rg_i for $r \geq 1$, and they are optimal or near-optimal in the number of intracycle spare links. We also showed that these designs can be incrementally reconfigured, and presented efficient switch implementations for them. The switch implementation is equally efficient in terms of the number of switches and switching links for any spare cluster size, including a cluster size of one. We also presented a fast, distributed, and localized reconfiguration algorithm of time complexity $\Theta(d)$ for our switch implementations, where d is the maximum processor degree of the multiprocessor. Finally, we showed that in the presence of cluster-faults, ALS designs have better reliability-to-cost ratios than previous FT designs that use comparable or greater amounts of spare hardware.

The ALS method appears to be the first truly general approach to the design of FT local-sparing systems, and places it on a a firm theoretical basis. It leads to designs that are not only efficient, but also possess a number of attractive properties mostly neglected in prior work, like applicability to any multiprocessor structure and any number of faults, low redundancy, and incremental, distributed and localized reconfigurability.

REFERENCES

1. E. Dekel, D. Nassimi and S. Sahni, Parallel matrix and graph algorithms, *SIAM Journal on Computing* **10** (4), 657–675, (November 1981).
2. H. Meijer and S.J. Akl, Optimal computation of prefix sums on a binary tree of processors, *Int. J. Parallel Programming* **16** (2), 127–136, (April 1987).
3. D. Nassimi and S. Sahni, Parallel permutation and sorting algorithms and a new generalized connection network, *Journal of the ACM* **29** (3), 642–667, (July 1982).
4. M.C. Pease, The indirect binary n-cube microprocessor array, *IEEE Trans. Comput.* **C-26** (5), 458–473, (May 1977).
5. P. Banerjee, S.Y. Kuo and W.K. Fuchs, Reconfigurable cube-connected cycles architecture, In *Proc. Sixteenth Fault Tolerant Comput. Symp.*, June 1986, pp. 286–291.
6. J. Bruck, R. Cypher and C.-T. Ho, Wildcard dimensions, coding theory and fault-tolerant meshes and hypercubes, In *Proceedings of 23rd Fault Tolerant Comput. Symp.*, June 1993, pp. 260–267.
7. J. Bruck, R. Cypher and C.-T. Ho, Fault-tolerant meshes and hypercubes with minimal number of spares, *IEEE Trans. Comput.* **42** (9), 1089–1104, (September 1993).
8. S-C. Chau and A.L. Liestman, A proposal for a fault-tolerant binary hypercube, In *Proceedings of Nineteenth Fault Tolerant Comput. Symp.*, Chicago, June 1989, pp. 323–330.
9. F.R.K. Chung, F.T. Leighton and A.L. Rosenberg, Diogenes: A methodology for designing fault-tolerant VLSI processor arrays, In *Proceedings of Thirteenth Fault Tolerant Comput. Symp.*, June 1983, pp. 26–31.
10. J.P. Hayes, A graph model for fault tolerant computing systems, *IEEE Trans. Comput.* **C-25**, 875–883, (September 1976).
11. M.B. Lowrie and W.K. Fuchs, Reconfigurable tree architectures using subtree oriented fault tolerance, *IEEE Trans. Comput.* **C-36**, 1172–1182, (October 1987).
12. C.S. Raghavendra, A. Avizienis and M.D. Ercegovac, Fault tolerance in binary tree architectures, *IEEE Trans. Comput.* **C-33**, 568–572, (June 1984).
13. D.A. Rennels, On implementing fault-tolerance in binary hypercubes, In *Proceedings of Sixteenth Fault Tolerant Comput. Symp.*, June 1986, pp. 344–349.
14. A.D. Singh, Interstitial redundancy: An area efficient fault tolerance scheme for large area VLSI processor arrays, *IEEE Trans. Comput.*, 1398–1410, (November 1988).
15. J.A. Trotter and W.R. Moore, Imperfectly connected 2-D arrays for image processing, In *Proceedings of Nineteenth Fault Tolerant Computing Symp.*, June 1989, pp. 88–92.
16. A.D. Singh and C.M. Krishna, Chip test optimization using defect clustering information, In *Proceedings of Twenty Second Fault-Tolerant Comput. Symp.*, July 1992, pp. 366–373.
17. N. Biggs, *Finite Groups of Automorphisms*, Cambridge University Press, (1971).
18. H.P. Yap, *Some Topics in Graph Theory*, pp. 96–97, Cambridge University Press, (1986).
19. S. Dutt and J.P. Hayes, An automorphic approach to the design of fault-tolerant multiprocessors, In *Proceedings of Nineteenth Fault Tolerant Comput. Symp.*, Chicago, June 1989, pp. 496–503.

20. S. Dutt and J.P. Hayes, Designing fault-tolerant systems using automorphisms, *Journal of Parallel and Distributed Computing*, 249–268, (July 1991).
21. G.H. Hardy and E.M. Wright, *An Introduction to the Theory of Numbers*, Fifth edition, Oxford University Press, New York, (1979).
22. S. Dutt, Designing and reconfiguring fault-tolerant multiprocessor systems, Report CSE-TR-73-90, Ph.D. Thesis, Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, (August 1990).
23. S. Dutt and J.P. Hayes, On designing and reconfiguring k-fault-tolerant tree architectures, *IEEE Trans. Comput.* **C-39**, 490–503, (April 1990).
24. S. Dutt and J.P. Hayes, Some practical issues in the design of fault-tolerant multiprocessors, *IEEE Trans. Comput.*, 588–598, (May 1992).
25. I.N. Herstein, *Topics in Algebra*, John Wiley and Sons, New York, (1975).
26. W.D. Hillis, *The Connection Machine*, MIT Press, Cambridge, MA, (1985).
27. J.G. Kuhl and S.M. Reddy, Distributed fault tolerance for large multiprocessor systems, In *Proceedings of Seventh Annual Int. Symp. on Comput. Architecture*, May 1980, pp. 23–30.
28. F.P. Preparata, G. Metze and R.T. Chen, On the connection assignment problem of diagnosable systems, *IEEE Trans. Electron. Comput.* **EC-16**, 848–854, (December 1967).