Hardware-Efficient and Highly-Reconfigurable 4- and 2-Track Fault-Tolerant Designs for Mesh-Connected Arrays*

Nihar R. Mahapatra
mahapatr@cse.buffalo.edu
Dept. of Computer Science & Engineering
State University of New York at Buffalo
Buffalo, NY 14260-2000

Shantanu Dutt
dutt@eecs.uic.edu
Dept. of Electrical Eng. & Computer Sc.
University of Illinois at Chicago
Chicago, IL 60607-7053

Abstract

We consider m-track models for constructing fault-tolerant (FT) mesh systems which have one primary and m spare tracks per row and column, switches at the intersection of these tracks, and spare processors at the boundaries. A faulty system is reconfigured by finding for each fault u a *reconfiguration path* from the fault to a spare in which, starting from the fault u, a processor is replaced or "covered" by the nearest "available" succeeding processor on the path—a processor on the path is *not available* if it is faulty or is used as a "cover" on some other reconfiguration path. In previous work, a 1-track design that can support any set of node-disjoint straight reconfiguration paths, and a more reliable 3-track design that can support any set of node-disjoint rectilinear reconfiguration paths have been proposed. In this research note, we present: (1) A fundamental result regarding the universality of simple "one-to-one switches" in m-track 2-D mesh designs in terms of their reconfigurabilities. (2) A 4-track mesh design that can support any set of edge-disjoint (a much less restrictive criterion than node-disjointness) rectilinear reconfiguration paths, and that has 34% less switching overhead and significantly higher, actually close-to-optimal, reconfigurability compared to the previously proposed 3-track design. (3) A new 2-track design derived from the above 4-track design that we show can support the same set of reconfiguration paths as the previous 3-track design but with 33% less wiring overhead. (4) Results on the deterministic fault tolerance capabilities (the number of faults guaranteed reconfigurable) of our 4- and 2-track designs, and the previously proposed 1- and 3-track designs.

Keywords: Average fault tolerance, deterministic fault-tolerance, fault-tolerant multiprocessors, mesh systems, m-track designs, node covering, reconfiguration, VLSI/WSI arrays.

*S. Dutt was supported by NSF grant MIP-9210049.

1 Introduction

1.1 Motivation and Background

Parallel machines with a mesh topology (e.g., the Intel Paragon and MasPar systems) employing hundreds to thousands of processors are currently available. With increased processor count, the likelihood of one or more processor failures also increases. Processor failures result in a loss of the underlying mesh interconnection structure which is important for correct and non-degraded operation in many applications. Thus it is important to design mesh-connected processor systems with *structural fault tolerance* (SFT), which is the ability to reconfigure around faulty components (processors, switches, or links) in order to preserve the original processor interconnection structure. In this research note, we present SFT (or FT for brevity) design methods that can be applied to make cabinet-scale mesh multiprocessors fault tolerant; our design methods are also applicable to very large scale and wafer scale integrated processor arrays such as those used in signal and image processing applications [8].

We consider two types of models for incorporating fault tolerance into mesh systems, a "general m-track model," and a more specialized version of this model called a "special m-track model" used in most previous work [9, 13] and shown in Fig. 1; the FT designs based on these models are called *general* and *special m-track designs*, respectively. A *special m-track model* consists of an $n_1 \times n_2$ array of *primary* processors embedded in a rectangular grid with m spare tracks running along each grid line and with "processor-to-track" (PT) switches placed at the intersection of these tracks and the original interconnections between processors (henceforth to be called *primary tracks*), and "track-to-track" (TT) switches placed at the intersection of row and column spare tracks [13]. In a single-spare design, a spare processor $s^t_{i,*}$ is connected to the right or tail end of each row i and similarly a spare $s^t_{*,j}$ is linked to the bottom or tail end of each column j. A double-spare design has an additional spare $s^h_{i,*}$ connected to the left or head of each row i and an additional spare $s^h_{*,j}$ connected to the top or head of each column j. The array in Fig. 1 is a special 2-track double-spare design. Moreover, in some designs each processor may use *interchange* and/or *bypass switches* to allow flexibility in connecting its four internal input/output (i/o) terminals to the four primary tracks adjacent to it. The functionality of various switches will be described in later sections. The spare processors, the spare tracks, and the switches provide the necessary reconfiguration capability for the array and represent the hardware overhead for fault tolerance. Examples of special m-track designs are the 1-track design of [9], the 3-track design of [13], and our new 2-track design presented in Sec. 4.

In a non-FT cabinet-scale multiprocessor, each processor and its associated chips (e.g., memory and router chips) will be on a single printed circuit board (PCB) or a part thereof with all or most interprocessor connections spanning across PCBs. When such a cabinet-scale multiprocessor is made fault tolerant using the special m-track model, we can group along with each processor an m-track TT switch at its corner and two of the m-track PT switches on its sides, as shown by the dashed boxes in Fig. 1 for processors $(0,0)$, $(0,1)$, $(1,0)$, and $(1,1)$, and put them on the same PCB as the processor. Partitioning the array into a mesh of such modules that contain processors and associated switches, we see that one primary and m spare tracks

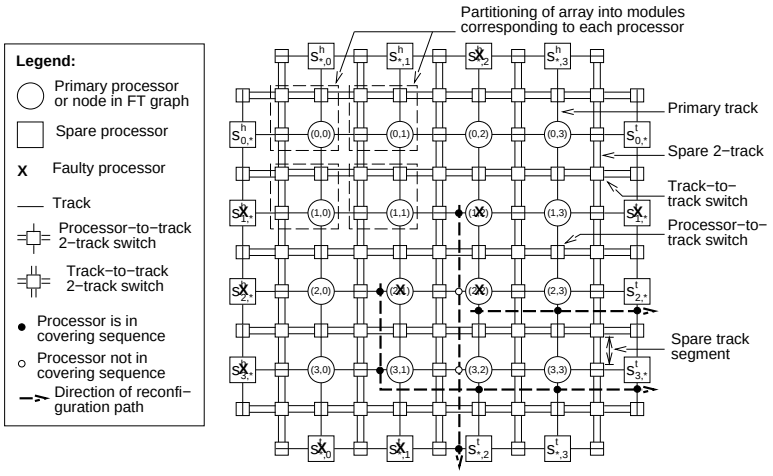


Figure 1: The m-track reconfiguration model for $m = 2$ showing reconfiguration paths for faults $(1, 2)$, $(2, 1)$ and $(2, 2)$.

run between any two adjacent modules. For a given number of spare processors, the number of spare tracks quite accurately characterizes the hardware complexity of a design. This is because wires that run between different modules, and hence between different PCBs, have a much larger width than a similar number of wires within a module, where the integration density is at least an order of magnitude higher. Therefore, in order to capture the hardware complexities of a larger class of FT mesh designs that may not necessarily fit the special m-track model (i.e., those in which not all switches needed for reconfiguration will necessarily be at the intersection of row and column spare tracks and at the intersection of primary and spare tracks), but which nevertheless have a hardware complexity similar to special m-track designs, we consider general m-track models for incorporating fault tolerance defined as follows. A *general m-track model* consists of an $n_1 \times n_2$ array of modules embedded in a rectangular grid, with each module containing a primary processor and some switching capability that can connect the processor to primary or spare tracks, and with one primary and m spare tracks running between any two adjacent modules. Note that the special m-track model fits the general m-track model, and hence is a special case of the latter. An example of a general m-track design, that is not a special m-track one, is the 4-track design presented in Sec. 3.

Given an N-processor (i.e., one with N primary processors) m-track mesh system with some faulty processors, the reconfiguration problem is to reconnect the processors so that an N-processor fault-free array can be realized. It is assumed that because of their relatively lower hardware complexities, switches and links are fault-free. A faulty system is reconfigured by finding for each fault a *covering sequence* which is an ordered sequence of processors beginning with the faulty processor and ending in a spare processor such that each processor replaces or "covers" the preceding processor in the sequence by connecting to the covered processor's neighbors and taking over its function. A processor v_2 covers another processor v_1 by using switches associated with intermediate processors, and track segments between these switches, on a path from v_1 to v_2 called the *covering path*. A *reconfiguration path* for a fault u is a path from u to a spare through a superset of processors in its covering sequence obtained by concatenating all covering paths between consecutive processors in the covering sequence. For example, in Fig. 1, the covering sequence for fault $(1, 2)$ is

$((1,2), s^t_{*,2})$, while the corresponding reconfiguration path is $\langle (1,2), (2,2), (3,2), s^t_{*,2} \rangle$. Since in the m-track model a processor can cover another processor only via the switches of intervening processors, reconfiguration paths always consist of adjacent processors and hence are *continuous*. Covering sequences, on the other hand, may be *discontinuous*—this is the case in the above example in which processors $(2,2)$ and $(3,2)$ are *skipped* in $(1,2)$'s reconfiguration path. Note that the distance between a covering processor and a covered processor on a reconfiguration path determines the wire length between new neighbors in the reconfigured array. In order to keep this wire length small, we assume that if permitted by the switching capabilities in the design, every processor on a reconfiguration path lies on at least some covering sequence. Henceforth, we will distinguish between covering sequences and reconfiguration paths only when the two are different, otherwise we will simply use the latter term. Next, we briefly summarize results from previous work and our new research in this research note.

1.2 Previous Work and New Results

In previous work, schemes with different numbers of spare tracks and switches, and with different reconfiguration capabilities have been proposed. A special 1-track design with only PT switches and with bypass switches in each processor was proposed in [9] and was shown to be able to support any set of node-disjoint (i.e., nonintersecting) straight reconfiguration paths without near-misses, i.e., a set F of faults is tolerable in this design if $|F|$ node-disjoint straight reconfiguration paths, one for each fault, with no near-miss situations can be found. A *near-miss* situation is said to occur when two reconfiguration paths in adjacent processor rows or columns overlap by at least two processors and run in opposite directions [9] (see, e.g., reconfiguration paths for $(5,7)$ and $(7,8)$ in Fig. 5(a) and Fig. 6(a)). The above result was generalized in [7] to the m-track case by assuming that there are m spare rows/columns on the four sides of the array. Recently, it was shown in [13] that this design does not provide adequate reconfigurability for small values of m (e.g., $m = 2$), and although the reconfigurability improves with increasing m, the hardware overhead is high while the spare-processor utilization is low. Also, in [13], a special 3-track design with both PT and TT switches was presented as a "good" compromise between hardware overhead and reconfigurability. This design was proved to be able to support any set of node-disjoint rectilinear reconfiguration paths, including bent reconfiguration paths, e.g., that for fault $(2,1)$ in Fig. 1 (provided there is no intersecting path), which the earlier designs of [9, 7] could not. A variety of other designs for FT meshes have also been proposed in the past [1, 3, 10]. However, the 3-track design of [13] is generally considered to be among the best current designs. Therefore we will compare our new designs proposed in this research note to it. A survey of different FT mesh design methods can be found in [14].

Recently, we proposed a very efficient and general method called *probabilistic node covering* for constructing FT multiprocessor systems of arbitrary topology [6]. This method uses linear error correcting codes (ECCs) to determine the interconnection between the spare switches used for reconfiguration. The number of faults the FT multiprocessor can tolerate on the average is close to s, the number of spares, which is much larger than the worst-case number k of faults that it can tolerate—hence the name "probabilistic" node covering.

In contrast, previously proposed deterministic k-FT systems with comparable switch and link complexities and with k spares can only tolerate any k or fewer faults [2, 4, 5, 12]. Since hardware complexities in most multiprocessor systems are wire dominated, probabilistic node covering designs provide high reliability at low cost.

In this research note, we first show that simple "one-to-one" PT and TT switches in special m-track designs provide all the reconfiguration capability possible in such designs. Then, we present two efficient FT mesh designs derived from the probabilistic node covering method that have different hardware overheads and reconfiguration capabilities. The first design is a general 4-track design that is obtained by using the 2D-parity code in a probabilistic node-covering based FT mesh system; the switches in this design are simpler than the "one-to-one" switches of a special 4-track design. In this FT system, any set of edge-disjoint rectilinear reconfiguration paths can be supported, thus vastly increasing the chances of reconfiguration over that of the 3-track design of [13], which requires that these paths be node-disjoint. Although our design uses an extra spare track compared to the 3-track design of [13], it has about 34% less switching overhead and markedly better spare utilization and reliability.

The second design is derived from the first and is a special 2-track design with bypass and interchange switches at every processor. Like the special 3-track design of [13], it can support any set of node-disjoint rectilinear reconfiguration paths and therefore has the same reliability. The switching overhead of this design is about the same as that of [13]. However, since the spare-track area is the dominant overhead of a design, our 2-track design offers almost 33% reduction in hardware overhead over the 3-track design of [13], but the same reconfiguration capabilities.

The remaining sections describe our two new FT designs and compare them to previous ones. Due to space constraints, some of our results are stated without proofs or only with proof outlines; detailed proofs can be found in [11].

2 Sufficient Functionality of m-Track Switches in Special m-Track Designs

Here we state a fundamental result regarding the universality of "one-to-one" switches. A *one-to-one switch*, either the PT or TT kind, is one that can connect a horizontal (vertical) track segment to either: (1) any one of the vertical (horizontal) track segments, or (2) to only the corresponding horizontal (vertical) track segment on the other side of the switch—thus the term one-to-one, or (3) tristate it. A *many-to-many switch* is one that can connect a track segment (horizontal or vertical) to any one of a subset of track segments incident on the switch or tristate it. An *all-to-all switch* is the most general form of a many-to-many switch in that it can connect any track segment to any one of the other track segments incident on the switch or tristate it.

Theorem 1 *Any new connections between processors (as decreed by some reconfiguration algorithm) that are possible in a special m-track design employing all-to-all switches, are also possible in one that employs only one-to-one switches.* □

Henceforth, any reference to a special m-track design will be to one that employs only one-to-one PT and TT switches, unless otherwise indicated. Both the 3-track design of [13] and our new 2-track design in

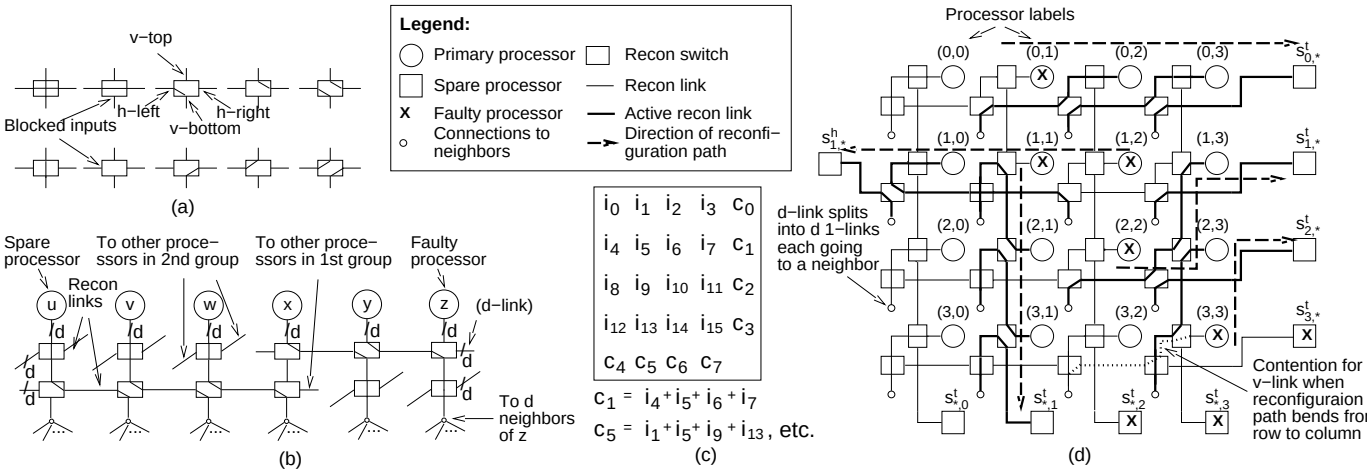


Figure 2: (a) Possible settings of a recon-switch. (b) Reconfiguration of faulty processor z in the probabilistic FT method via recon-switch settings. (c) Information bit groups in the 2D-parity code. (d) Illustration of various reconfiguration paths in a 16-processor probabilistic FT system derived from the 2D-parity code.

this paper are such designs. Since a many-to-many switch provides only a subset of the functionality of an all-to-all switch, Theorem 1 leads to the following corollary.

Corollary 1 *If reconfiguration is possible in a special m-track design employing many-to-many PT and TT switches, then it is also possible in a special m-track design employing only one-to-one PT and TT switches.*

In the next section, we derive the 4-track FT mesh design and establish its reconfiguration capabilities.

3 The 4-Track FT Mesh Design

3.1 Probabilistic Node Covering Preliminaries

In the probabilistic node covering method, we use linear error correcting codes (ECCs) to partition the given multiprocessor (of any topology) into a set of nondisjoint processor groups. Processors in each group are linearly ordered with adjacent processors connected via *recon switches* to facilitate the replacement of any processor v in the group by any other processor u in either the same group or in an intersecting group, as we will shortly see. A recon switch is a 2×2 switch with possible states as shown in Fig. 2(a). It has four bidirectional terminals labeled h-left, h-right, v-top, and v-bottom and can connect any pair of them together as shown. These terminals can have any width (number of wires), and a switch with d wires at each terminal is termed a d-link switch. Note that a recon switch is nothing but a one-to-one 1-track switch (1-track PT and TT switches are the same).

To see how reconfiguration is done with the aid of recon switches, consider the example in Fig. 2(b) which shows a subset of processors in the multiprocessor. Here each processor belongs to two processor groups and is connected to its neighbors, d in number, via two series-connected recon switches, one for each group, that have their v-tops and v-bottoms connected. Processors in the same group are interlinked by connecting the h-right of a recon switch in a processor i to the h-left of a recon switch in the next processor j in the group. These connections between recon switches are the spare tracks used in the design and are referred to as *recon links*. In the example shown, processors u, v, w, and x belong to one group and processors x, y, and z to

another. Now suppose that processor z is faulty and that u is a spare processor. Then the faulty processor can be reconfigured around as follows. First, the d links from y are connected to the d neighbors of z by setting their recon switches as shown so that processor y replaces or "covers" processor z. A similar setting of recon switches allows x to cover y. Next, processor w replaces processor x by connecting to the latter's neighbors via a different recon switch of x. Note that there is a "bend" at x, i.e., the reconfiguration path for z moves from one group to another at the common processor x. Next, v replaces w, and finally the spare processor u replaces v and the reconfiguration is complete. Therefore the covering sequence for z is (z, y, x, w, v, u) which in this case is also the sequence of processors on the reconfiguration path for z. Next, we show how the 2D parity code can be used to obtain a 4-track FT design for mesh multiprocessors.

3.2 Derivation of 4-Track Design from the 2D Parity Code

A linear ECC has N information bits i_j, $0 \leq j < N$, and g check bits c_k, $0 \leq k < g$; the check bits are related to the information bits by linear equations, and can be used to determine if there are any errors in the code word (the combination of information and check bits), and possibly correct them. In the 2D-parity code, the N information bits are arranged in two dimensions as an $n_1 \times n_2$ array, where $N = n_1 \cdot n_2$, and each row and column of this array is a group with which a parity check bit is associated; see Fig. 2(c) in which two example parity equations (for c_1 and c_5) using modulo 2 addition are also given. Each check bit serves to detect errors in any information bit in its group. A probabilistic node covering design is derived from the 2D-parity code by constructing processor groups in the following manner. Each primary processor is uniquely associated with an information bit and each spare processor with a check bit. Each parity group and its check bit in the ECC then give rise to a processor group and its associated spare. The 2D-parity code can detect any two errors and can correct any single error. ECCs are used as the basis for formulating FT processor groups, since the *deterministic error detectability* (the maximum number of errors guaranteed detectable in the worst-case) and the *average error correctability* of ECCs are lower bounds on the *deterministic fault tolerance* (the maximum number of faults guaranteed tolerable in the worst-case) and the *average fault tolerance*, respectively, of the corresponding FT multiprocessor [6].

The probabilistic FT design corresponding to the 2D-parity code of Fig. 2(c) is shown in Fig. 2(d)—the normal setting of recon switches of a processor when the multiprocessor is not faulty is the same as that for the switches of processor $(2, 0)$ in the figure. Note that the top recon switches of processors in a column are interlinked via their h-lefts and h-rights to form a column group, and similarly the bottom recon switches in a row are interlinked to form a row group. In a single-spare design, spares are attached to the tail end of each row and column group. A double-spare design has an additional spare attached to the head end of each row and column group (e.g., $s^h_{1,*}$ in row 1).

The above design method can be used for any multiprocessor. In a mesh FT system, the rows and columns of the mesh correspond to the row and column groups of Fig. 2(b) and the d-links from each processor go to the four adjacent neighbors of the mesh. Figure 5(a) depicts a more detailed view of a part of the mesh FT system in which the two 4-link ($d = 4$ for the mesh) recon switches associated with a processor have been

split into four pairs of 1-link recon switches, a pair for each neighbor of a processor. The connections between a processor and its four neighbors via recon switches are explicitly shown in this figure.

3.3 Hardware Complexities of the 4- and 3-Track Designs

By grouping each processor with its four pairs of recon switches into a module, as shown by the dashed boxes around processors $(4,4)$ and $(4,5)$ in Fig. 5(a), and thereby partitioning the array, we notice that there are 4 spare tracks (and one primary track) running between any two adjacent modules. From the definition introduced in Sec. 1.1, we conclude that the above design is indeed a general 4-track design as previously stated, and thus has a 33% higher spare-track overhead compared to the 3-track design of [13]. Below, we summarize the switching overheads of these two designs.

Proposition 1 *The 4-track design has a switching area overhead of 60 square units per processor using a two conducting-layer technology.*

Proposition 2 *The 3-track design of [13] has a switching area overhead of 91 square units per processor using a two conducting-layer technology.*

Therefore, although the 4-track design uses an extra spare track, it has 34% less switching area overhead. Moreover, our 4-track design has much better reconfiguration capabilities as we will show analytically in the next subsection and empirically in Sec. 6.

3.4 Reconfiguration Capabilities of the 4-Track Design

In this subsection we establish the reconfiguration capabilities of the above described 4-track FT mesh design. A key idea behind reconfiguration in the node-covering technique is that by connecting the left, right, etc., i/o ports of a covering processor to the corresponding recon switches of a covered processor and from there to the i/o terminals of its bottom recon switches that connect to the bottom recon switches of its neighbors, correct connections between the covering processor and the neighbors of the covered processor are established. In Fig. 2(d), for example, to replace processor $(2,1)$ by processor $(3,1)$, we route the links from $(3,1)$ to the bottom recon switch of $(2,1)$ where $(3,1)$ can be connected to $(2,1)$'s neighbors. Note that even if processor $(2,1)$'s neighbors change, as is the case for processor $(2,2)$, its right neighbor, which gets replaced by processor $(1,3)$, connections from $(3,1)$ to these new neighbors will be correctly established. This is because connections from these new neighbors are also brought to the location where the previous neighbor was connected to processor $(2,1)$. The above idea will also be used with suitable modifications to derive our 2-track design in the next section.

Note that reconfiguration paths in all four directions (such as those for processors $(0,1)$, $(1,2)$, $(1,1)$ and $(3,3)$ in Fig. 2(d)) can be supported. Note also that some bent reconfiguration paths such as those for faults $(2,2)$ and $(3,3)$ in Fig. 2(c) can be supported. The next theorem identifies precisely the types of reconfiguration paths that the 4-track design can support.

Theorem 2 *The 4-track design can support any arbitrary set of edge-disjoint rectilinear reconfiguration paths except those in which either (1) two paths in opposite directions along a row intersect at a non-faulty processor and bend away*

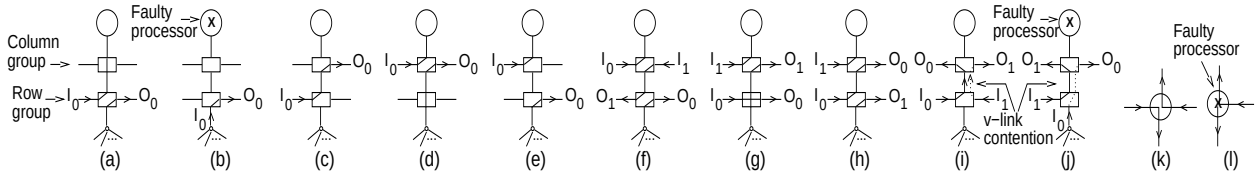


Figure 3: Establishing the reconfiguration capabilities of the 4-track design. Single reconfiguration path passes through processor (a) along row group, (b) along row group when the processor is faulty, (c) entering along row group, and bending and skipping to leave along column group, (d) along column group, and (e) entering along column group and bending to leave along row group. Two reconfiguration paths pass through processor (f) both entering along column and leaving along row groups, (g) one entering along row and another along column group, and both leaving along same group, i.e., without bending, (h) one entering along row and another along column group, and both bending, (i) both leaving along column group when processor is not faulty, and (j) both leaving along column group when processor is faulty. (k) Example of reconfiguration path pattern at processor corresponding to [i]. (l) Example of reconfiguration path pattern at processor corresponding to [j].

in opposite directions along the column at that processor, or (2) one path enters a faulty processor along a row while the other begins at the faulty processor, and both paths leave the processor along its column in opposite directions.

Proof: Recall from the way reconfiguration is achieved in the node-covering technique that the recon d-link between two consecutive processors on a reconfiguration path is used to set up new connections. Consequently, if a set of reconfiguration paths is edge-disjoint, the only possible point of contention is at the recon switches of processors that these paths pass through. In Fig. 3 we depict the various cases that might arise when one or two reconfiguration paths pass through a processor; obviously not more than two paths can pass through a processor if the paths must be edge-disjoint. We denote the terminal at which a path i enters by I_i and the terminal at which it leaves by O_i. We need to determine the cases of reconfiguration path(s) passing through the processor that cannot be supported; whether or not the processor is skipped in the reconfiguration path(s) is immaterial. Clearly, if a single straight or bent path passes through a processor, it can be supported (see Figs. 3(a) through (e)). From Fig. 3(a) and (b) it is evident that the case of a path entering along a row group (i.e., entering at the bottom recon switch) is similar, as far as routing paths through the switches is concerned, to the case when the processor is faulty and the path enters at the bottom terminal of the bottom recon switch. Therefore, we do not explicitly consider the other cases in Fig. 3 corresponding to a faulty processor. Next, in Figs. 3(f) through (j) we consider the different cases that arise when two reconfiguration paths enter or leave along the same or different groups. The only case in which two paths cannot be supported is when both paths enter along a row group (i.e., horizontally) or one enters along a row group and the other begins at the processor which is faulty, and both leave along a column group (i.e., both leave vertically in opposite directions), in which case there is contention for the v-link connecting the two recon switches (Figs. 3(i) through (l)). □

Theorem 3 *The 4-track design can support any arbitrary set of edge-disjoint rectilinear reconfiguration paths when there is an extra v-link between the top and bottom recon switches.*

Proof: From the proof of Theorem 2 and the routing of paths (one solid and one dashed) shown in Figs. 3(i) and (j), it is clear that the provision of an additional v-link between the top and bottom recon switches will allow all types of edge-disjoint rectilinear reconfiguration paths to be supported. □

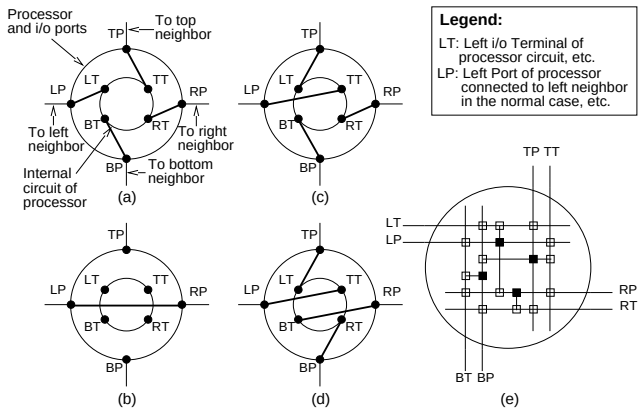


Figure 4: *(a)* Normal connections in bypass-cum-interchange switch used in the 2-track design of Sec. 4: left port (LP) connected to left i/o terminal (LT), and so on. *(b)* Bypass in bypass-cum-interchange switch: Left port connected to right port. *(c)* Single interchange in bypass-cum-interchange switch: left i/o terminal connected to top port and top i/o terminal to left port. *(d)* Double interchange in bypass-cum-interchange switch: both left-top and right-bottom connections are interchanged; other states of the bypass-cum-interchange switch are similar to the states in (b), (c) and (d). *(e)* Layout of a bypass-cum-interchange switch. White squares denote programmable switches that when switched on connect the two lines intersecting them and black squares denote shorts or vias.

We will see in Sec. 6 that even without the extra v-link, the reconfigurability of the 4-track design is very high.

4 The 2-Track FT Mesh Design

In this section, we determine the hardware complexity and reconfiguration capabilities of our new 2-track design, which, as stated previously in Sec. 2, is a special 2-track design employing one-to-one switches. It is assumed that each processor has a *bypass-cum-interchange switch* that allows it to implement both bypass and interchange functions; see Figs 4(a) through (d). In the figure, the left, right, etc., i/o terminals are the corresponding i/o terminals of the circuitry in the processor. The left, right, etc., ports are the external terminals on the corresponding sides of the processor that go to the left, right, etc., neighbors, respectively. Normally, the left i/o terminal is connected to the left port, and so on ((Fig. 4(a)). The *bypass* function allows to connect any one port of a processor to the port on the opposite side, e.g., the left port may be directly connected to the right port, thus bypassing the processor (Fig. 4(b)). The *interchange* function is used to interchange the positions of any one or two pairs of adjacent i/o lines of a processor, e.g., the left i/o line of the processor may be made to appear at its top port and the top i/o line to appear at its left port (Fig. 4(c)), or both left-top and right-bottom connections may be simultaneously interchanged (Fig. 4(d)). Next, we determine the hardware complexity of the 2-track design.

4.1 Hardware Complexity

First of all, the number of spare tracks per row or column used in this design is just two, which represents a 33% reduction in spare-track overhead compared to the 3-track design of [13]. Below we state the switching overhead of the 2-track design.

Proposition 3 *The 2-track design has a switching area overhead of* 91 *square units per processor using a two conducting-layer technology.*

Thus the switching overheads in our 2-track design and the 3-track design of [13] (see Proposition 2) are the same. However, note that since the bypass and interchange switches control lines within the processor, they can be implemented along with the regular circuitry of the processor in an efficient manner. Therefore, in practice, the switching overhead of the 2-track design will be less than that in the 3-track design.

4.2 Reconfiguration Capabilities

In the following, we will constructively prove that our 2-track design can support any set of node-disjoint rectilinear reconfiguration paths. Recall that in [13] a 3-track design was used to support the same set of reconfiguration paths, and hence as stated earlier, our design represents a major (about 33%) reduction in hardware requirements while offering the same reliability. As noted earlier, reconfiguration in the 2-track design is derived from that of our 4-track design. An overview of this derivation is given next.

4.2.1 Derivation of the 2-Track Design from the 4-Track Design—An Overview

The derivation consists of two steps. First, we make simple modifications to the hardware of the 4-track design to reduce it to a special 2-track design with many-to-many switches that is *intermediate* between the 4-track design, and the special 2-track design with one-to-one switches discussed earlier and shown in Fig. 1. Later we will utilize this relationship between the 4-track and intermediate 2-track designs to reconnect processors in a faulty special 2-track design (with one-to-one switches). Thus the intermediate 2-track design serves as a conceptual link between our 4-track and special 2-track designs. Henceforth, whenever we refer to a 2-track design, we will mean a special 2-track design with one-to-one switches unless we specify otherwise. Recall that in the 4-track design (Fig. 6(b)), the four vertical spare tracks in a column are used in vertical edges of reconfiguration paths to connect a processor on the path to its new left, right, top, and bottom new neighbors via the left, right, top, and bottom, respectively, column-group recon switches of the processor. Similarly, the four horizontal spare tracks in a row are used to support horizontal edges of reconfiguration paths via the row-group recon switches of the processor. To obtain the intermediate 2-track design, we remove from each column of the 4-track design the two vertical spare tracks and the column-group recon switches meant for connecting to the top and bottom neighbors, and from each row the two horizontal spare tracks and the row-group recon switches meant for connecting to the left and right neighbors, as can be seen by comparing Figs. 6(b) and (c). Each processor now has only one recon switch corresponding to each of its neighbors (as opposed to two in the 4-track design) with one spare track passing through the switch, so that between any two adjacent processors in a row or column there are only two recon switches connected in series (instead of the original four) and two vertical or horizontal, respectively, spare tracks passing through them. Next, we introduce an additional recon-link between the above two recon switches as we did in Theorem 3 for the original two recon switches associated with every neighbor of a processor. This is shown for the switches between processors $(6, 6)$ and $(6, 7)$ in Fig. 6(c).

Finally, we place a one-to-one 1-track switch wherever a vertical and a horizontal spare track intersect as shown in Fig. 6(c). The four one-to-one 1-track switches thus located at the intersection of two vertical and

two horizontal spare tracks between any two processor columns and rows, respectively, correspond to a 2-track TT switch in Fig. 1 and are equivalent in functionality to a one-to-one 2-track TT switch. A set of four such switches between processors $(5,6)$, $(5,7)$, $(6,6)$, and $(6,7)$ is indicated in Fig. 6(c). Also, by grouping the two recon switches between adjacent processors, as shown for those between processors $(6,6)$ and $(6,7)$, we see that they correspond to the 2-track PT switches in the 2-track model of Fig. 1. These switch pairs provide some more functionality than one-to-one 2-track switches. For instance, they allow cross connections as shown for the switch pair between processors $(6,7)$ and $(6,8)$ in Fig. 5(b) which one-to-one 2-track switches do not. Thus the intermediate design derived above from the 4-track design and shown in Fig. 6(c) is a special 2-track design (since it fits the special 2-track model) with many-to-many switches.

In the second step of the derivation, we modify the way we route new connections in the 4-track design so that for any set of node-disjoint rectilinear reconfiguration paths, the new connections can be supported by the intermediate 2-track design. Finally, by using the routing method given in the proof of Theorem 1 and from Corollary 1, we can support these new connections in a special 2-track design with one-to-one switches.

Recall that in the 4-track design we use different spare tracks to connect to different new neighbors of a processor on a reconfiguration path. We are able to eliminate two spare tracks in intermediate 2-track reconfiguration basically by making use of the fact that any two adjacent processors on a reconfiguration path (whether straight or bent) that does not intersect any other reconfiguration path remain adjacent after reconfiguration. Therefore for any processor on a reconfiguration path, two of its new connections can be made by utilizing the existing primary track connection between it and its two adjacent processors on the path with the help of bypass and interchange switches. For the other two new neighbors which do not lie on the reconfiguration path, the node-covering technique of connecting ports of a covering processor to the corresponding switches of a covered processor used in the 4-track design, is used to make connections to them. However, the routing of these new connections is a modification of the routings in the original 4-track design, the modification being necessitated by interchanges which cause the i/o terminals of a processor to appear at a port other than the normal one. In the proofs to the lemmas below, we spell out how 4-track reconfiguration is modified to intermediate 2-track reconfiguration, and then obtain from Theorem 1 the corresponding special 2-track reconfiguration.

The proof of our claim regarding the reconfiguration capability of the special 2-track design is simplified by demonstrating 2-track sufficiency for the two types of subpaths that can be present in a *general reconfiguration path*, viz., *straight* and *completely bent* or *stair-case* subpaths. Therefore we will first consider these two special cases of reconfiguration paths, viz., *node-disjoint straight paths* and *node-disjoint completely bent paths*, and establish 2-track sufficiency for them. Since node-disjoint reconfiguration paths are being considered, we assume that all processors on a reconfiguration path are in the covering sequence for the corresponding fault, i.e., no processor on the path is skipped.

4.2.2 Two-Track Sufficiency for Node-Disjoint Straight Paths

Lemma 1 *The special 2-track design employing one-to-one PT and TT switches can support any set of node-disjoint*

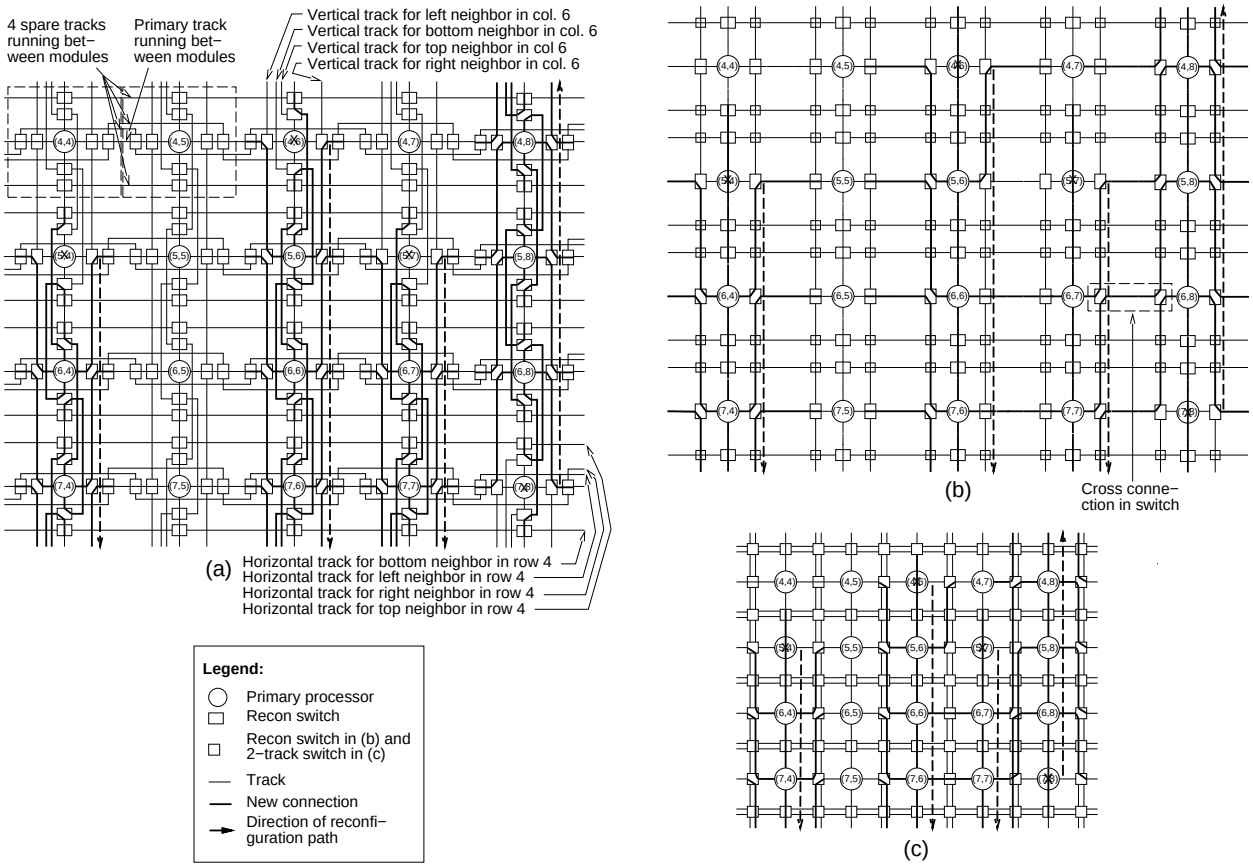


Figure 5: Supporting straight reconfiguration paths with or without near-misses in the (a) 4-track design, (b) intermediate 2-track design with many-to-many switches, and (c) special 2-track design with one-to-one switches.

straight reconfiguration paths.

Proof: An isolated vertical straight path (e.g., the one for fault $(5,4)$ in Fig. 5(a)) requires only a single vertical track on either side for 4-track reconfiguration. This is because: (1) processors on the vertical path remain top-bottom neighbors even after reconfiguration, so that the original primary-track top-bottom connections between these processors need not be disturbed; (2) faulty processor $(5,4)$'s cover, processor $(6,4)$, can access its new top neighbor $(4,4)$ in the intermediate 2-track design via a direct connection that bypasses fault $(5,4)$ as shown in Fig. 5(b); and (3) connections to the new left and right neighbors of a processor on the path are made by routing the left and right port of a covering processor to the left and right recon switch, respectively, of a covered processor using the vertical track segment connecting them. By symmetry, a horizontal straight path will require only a single horizontal track on either side. Note that the reconfiguration procedure described is valid whether or not there are adjacent parallel (e.g., those for faults $(4,6)$ and $(5,7)$) or near-miss straight paths (e.g., those for faults $(5,7)$ and $(7,8)$), since only one out of the two tracks is used for each reconfiguration path. Finally, from Corollary 1, we conclude that node-disjoint straight reconfiguration paths can also be supported by a special 2-track design (see Fig. 5(c)). □

Note that the TT switches in the 2-track model are not needed to support straight reconfiguration paths. These are needed only for bent reconfiguration paths which we consider next.

4.2.3 Two-Track Sufficiency for Node-Disjoint Completely Bent Paths

Next, we consider the case of completely bent or stair-case paths. We first state a useful result derived in [13] and illustrated in Fig. 6(a).

Lemma 2 [13] *Near-misses among bent rectilinear reconfiguration paths can always be removed.*

Lemma 3 *The special 2-track design employing one-to-one PT and TT switches can support any set of node-disjoint completely bent rectilinear reconfiguration paths.*

Proof Outline: Since from Lemma 2 near-misses among bent paths can always be removed, we need not consider near-miss situations. First, consider an isolated completely bent path going in the north-east direction as shown in the 4-track design of Fig. 6(b). Note that processors along the completely bent path that were originally left-right (bottom-top) neighbors become bottom-top (left-right) neighbors after reconfiguration, since they replace processors that are bottom-top (left-right) neighbors. Therefore we let the direct primary-track connections between processors on the reconfiguration path remain as they are, but interchange the left and bottom terminals and the right and top terminals as shown in Figs. 6(c) and (d) so that the desired connections are made. This way, two of the new-neighbor connections for each processor on the path are made using primary tracks.

Now, we only need to make connections to the remaining two neighbors that do not lie on the reconfiguration path. In Fig. 6(c), there are two sets of processors that can be identified in terms of the connections remaining to be made. The first set of processors (e.g., $(6,5)$) have their right and bottom, and the second

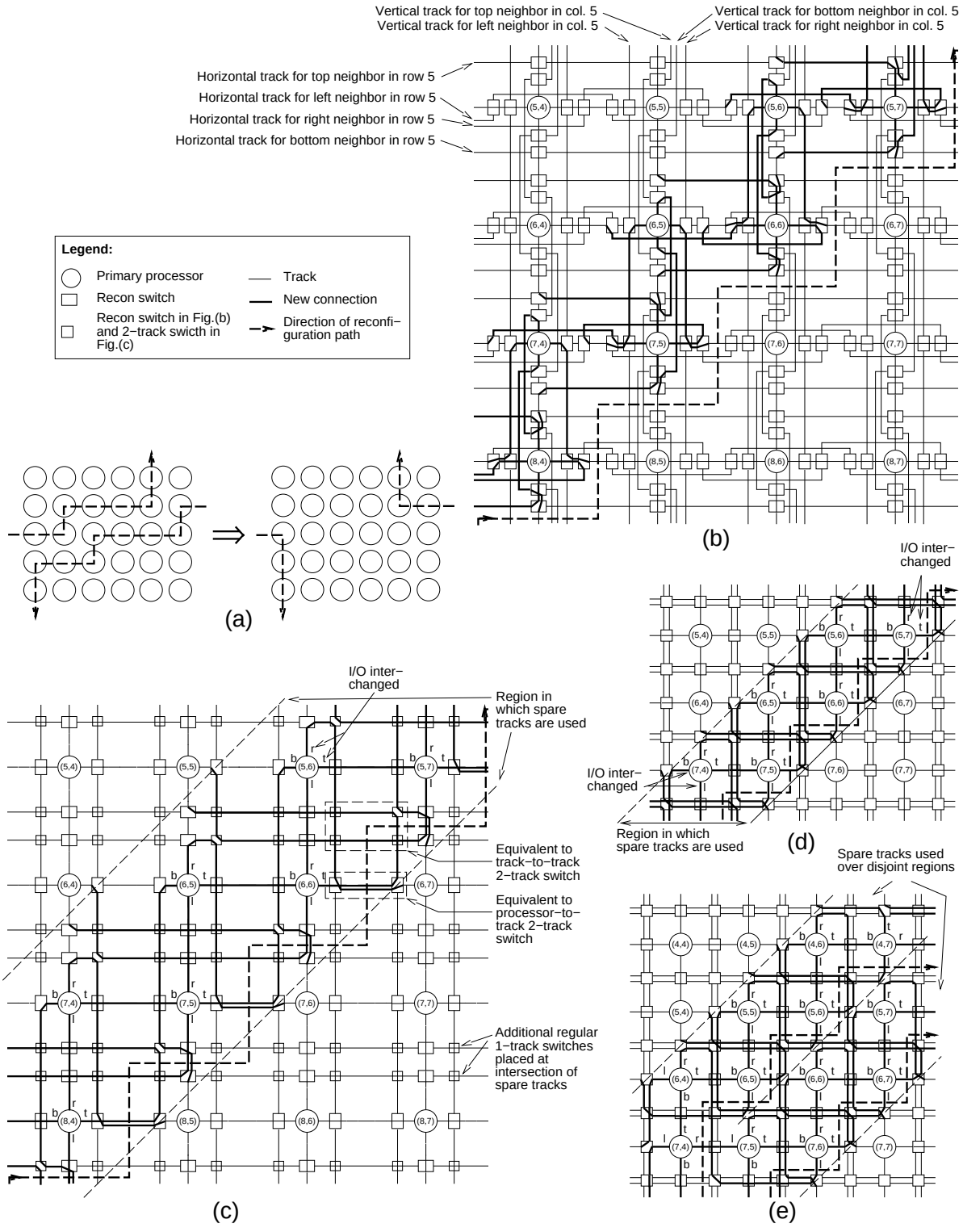


Figure 6: (a) Elimination of near-misses among bent paths by redirecting the original paths so as to get rid of the overlapping portion. Supporting an isolated completely bent path $\langle \ldots, (7,4), (7,5), (6,5), (6,6), (5,6), (5,7), \ldots \rangle$ in the (b) 4-track design, (c) intermediate 2-track design with many-to-many switches, and (d) special 2-track design with one-to-one switches. (e) Supporting adjacent parallel completely bent paths $\langle \ldots, (7,4), (6,4), (6,5), (5,5), (5,6), (4,6), (4,7), \ldots \rangle$ and $\langle \ldots, (7,5), (7,6), (6,6), (6,7), (5,7), \ldots \rangle$ in the special 2-track design with one-to-one switches.

set (e.g., $(6,6)$) have their top and left connections yet to be made. As stated in Sec. 4.2.1, these connections are established by routing connections from the port where the i/o line for the relevant neighbor is available to the corresponding recon switch of the covered processor. Note that processors in each set use the same routing pattern for establishing connections to their remaining two neighbors, and this is achieved without track contention.

Thus connections to all four new neighbors are established for all processors. By symmetry, we can handle completely bent paths in other directions, e.g., those going in the north-west direction. From Corollary 1, we can support a single completely bent path in a special 2-track design too as illustrated in Fig. 6(d).

A single completely bent path occupies two tracks in the track segments to its top and right that surround it as shown in Figs. 6(c) and (d). When two or more completely bent paths run side by side as in Fig. 6(e), we use the same routing scheme as in the single completely bent path case above. Since the track segments over which the adjacent paths utilize tracks are disjoint (see Fig. 6(e)), and since the cases of adjacent bent paths going in other directions is symmetrical, we conclude that two tracks are sufficient for making the necessary connections required by any set of node-disjoint completely bent paths. □

4.2.4 Two-Track Sufficiency for Node-Disjoint General Paths

Theorem 4 *The special 2-track design employing one-to-one PT and TT switches can support any set of node-disjoint rectilinear reconfiguration paths.*

Proof Outline: The proof follows from the fact that an isolated general path or adjacent parallel general paths consist of isolated or adjacent parallel subpaths, respectively, that are either straight or completely bent. We have already shown that such subpaths can be supported in a special 2-track design in Lemmas 1 and 3. □

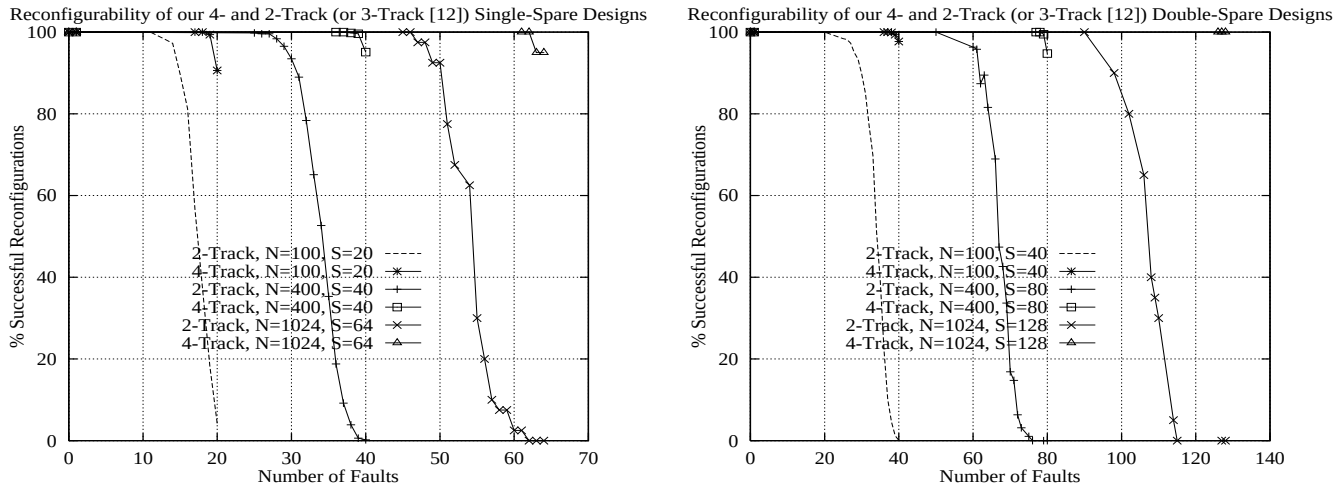
5 Deterministic Fault Tolerance (DFT) Capabilities

In this section, we give the DFT capabilities, defined as the maximum number of faults guaranteed reconfigurable, of our 4- and 2-track (or [13]'s 3-track) designs, and the 1-track design of [9]. The DFT capabilities are given both at the *corner* and *interior* of the FT layouts as follows. For single-spare designs, the corner DFT capability is computed at the corner of the layout that is not adjacent to spares, which is the worst-case; in the double-spare case, the particular corner is immaterial.

Theorem 5 [11] *The corner DFT capability of the 4-track design is* $\min(S,5)$ *when there is one spare each per row and column and* $\min(S,10)$ *when there are two spares each per row and column, and the interior DFT capability is* $\min(S,18)$*, where S is the total number of spares.*

Theorem 6 [11] *The corner DFT capability of the 2-track design is* $\min(S,2)$ *when there is one spare each per row and column and* $\min(S,4)$ *when there are two spares each per row and column, and the interior DFT capability is also* $\min(S,4)$*, where S is the total number of spares. The DFT capabilities of the 3-track design of* [13] *are identical.*

Theorem 7 [11] *Both the corner and interior DFT capabilities of the 1-track design of [9] are* $\min(S,2)$ *when there is one spare each per row and column and are* $\min(S,4)$ *when there are two spares each per row and column, where S is the total number of spares.*



(a) (b)

Figure 7: Reconfigurability of 4- and 2-track (or 3-track [13]) designs for arrays of 100, 400 and 1024 processors for (a) single- and (b) double-spare cases.

Note that the DFT capabilities of the 4-track design are very good and quite a bit higher than those of the 2- and 1-track designs. Although the DFT capabilities of the 2- and 1-track designs are similar, there are many more DFT fault patterns in the latter design than in the former, since DFT faults patterns in the 1-track design need not necessarily be clustered as in the 2-track case [11]. Hence, as the results in [13] show, our 2-track design or 3-track design of [13] is found to have much better average reconfigurability than the 1-track design.

6 Reconfigurability of the 4- and 2-Track Designs

Here we present reconfigurability results for our 4- and 2-track FT mesh designs obtained by using the maxflow-based reconfiguration algorithms of [6] (for the special case of the 2-D mesh architecture and the 2D-parity code) and [13], respectively, and Monte Carlo simulations averaged over 300-1000 samples. As noted earlier, the reconfigurability of the 2-track design is the same as that of the 3-track design of [13]. The 4-track design simulated has a single v-link between the two recon switches of a processor like the design implied in Theorem 2 and unlike that in Theorem 3. In Figs. 7(a) and (b) we plot the percentage of successful reconfigurations for various fault sizes for the single and double-spare cases, respectively, of the above designs. We denote the number of processors by N and the number of spares by S. Note that the reconfigurability of the 4-track design is 100% for all fault sizes except those of size $S-2$ or greater—of course it is possible to tolerate only faults of sizes S or less. Even for these large-sized faults, the reconfigurability is 90% or better and in most cases close to 100%. Furthermore, this is much better than the reconfigurability of the 3-track design of [13] (or our 2-track design) for large fault sizes. These results also demonstrate that the spare-processor utilization in the 4-track design is close-to-optimal.

7 Conclusions

We presented hardware efficient and highly reconfigurable 4- and 2-track designs for constructing FT mesh systems. The 4-track mesh design was derived from a general probabilistic node covering method for constructing FT systems of arbitrary topology proposed in [6]. It was shown that this design can support any arbitrary set of edge-disjoint rectilinear reconfiguration paths. By contrast, a previous 3-track design proposed in [13], and generally considered to be the best currently known FT mesh design, can support only an arbitrary set of node-disjoint rectilinear reconfiguration paths. Although our 4-track design has 33% more spare-track overhead than the 3-track design, it has 34% less switching-area overhead, and much better (close-to-optimal) reconfigurability for larger fault sizes. Furthermore, the deterministic fault tolerance capabilities of the 4-track design were shown to be much better than those of the 3-track design. A new 2-track design derived from the 4-track design was also presented and shown to have the same reconfiguration capabilities as the previously proposed 3-track design. This 2-track design has 33% less spare wiring area overhead and similar switching overhead compared to the 3-track design. Our designs, with their much higher reconfigurability and/or smaller hardware overheads for fault tolerance, represent significant improvements over previous FT mesh designs.

References

[1] K.P. Belkhale and P. Banerjee, "Reconfiguration strategies for VLSI processor arrays and trees using a modified Diogenes approach," *IEEE Trans. Comp.*, Vol. 41, No. 1, pp. 83-96, Jan. 1992.

[2] J. Bruck, R. Cypher, and C.-T. Ho, "Fault-tolerant meshes with small degree," *Proc. ACM Symp. Parallel Algorithms and Architectures*, pp.1-10, 1993.

[3] M. Chean and J.A.B. Fortes, "The full-use-of-suitable-spares (FUSS) approach to hardware reconfiguration for fault-tolerant processor arrays," *IEEE Trans. Comp.*, Vol. 39, No. 4, pp. 564-571, Apr. 1990.

[4] F.R.K. Chung, F.T. Leighton, and A.L. Rosenberg, "Diogenes: A methodology for designing fault-tolerant VLSI processor arrays," *Proc. Thirteenth Fault Tolerant Comput. Symp.*, June 1983, pp. 26-31.

[5] S. Dutt and J.P. Hayes, "Some practical issues in the design of fault-tolerant multiprocessors", *IEEE Trans. Comp.*, Vol. 41, No. 5, pp. 588-598, May 1992.

[6] S. Dutt and N.R. Mahapatra, "Node covering, error correcting codes and multiprocessors with very high average fault tolerance," *IEEE Trans. on Computers*, Vol. 46, No. 9, pp. 997-1015, Sep. 1997.

[7] J.S.N. Jean, H.C. Fu, and S.Y. Kung, "Yield enhancement for WSI array processors using two-and-half-track switches," in *Proc. Int. Conf. Wafer Scale Integration*, pp. 243-250, San Francisco, CA, Jan. 23-25, 1990.

[8] S.Y. Kung, *VLSI Array Processors*, Prentice Hall, 1988.

[9] S.Y. Kung, S.N. Jean, and C.W. Chang, "Fault-tolerant array processors using single track switches," *IEEE Trans. Comp.*, Vol. 38, No. 4, pp. 501-514, Apr. 1989.

[10] F. Lombardi, M.G. Sami, and R. Stefanelli, "Reconfiguration of VLSI arrays by covering," *IEEE Trans. Comp.*, Vol. 8, No. 9, pp. 952-965, Sep. 1989.

[11] N.R. Mahapatra and S. Dutt, "Hardware-efficient and highly-reconfigurable 4- and 2-track fault-tolerant designs for mesh-connected arrays," Technical Report, Dept. of Electrical Eng. & Computer Sc., Univ. of Illinois at Chicago, Chicago, IL, 1998 (available at http://www.eecs.uic.edu/~dutt/publ.html).

[12] H. Tamaki, "Construction of the mesh and the torus tolerating a large number of faults," *Proc. ACM Symp. Parallel Algorithms and Architectures*, pp. 268-277, 1994.

[13] T.A. Varvarigou, V.P. Roychowdhury, and T. Kailath, "Reconfiguring processor arrays using multiple-track models: The 3-track-1-spare-approach," *IEEE Trans. Comp.*, Vol. 42, No. 11, pp. 1281-1293, Nov. 1993.

[14] Y.X. Wong, "Approximate estimation of reliability, mean-time-to-failure, and optimal redundancy of fault-tolerant processor arrays," Ph.D. dissertation, Purdue University, West Lafayette, IN, May 1990.